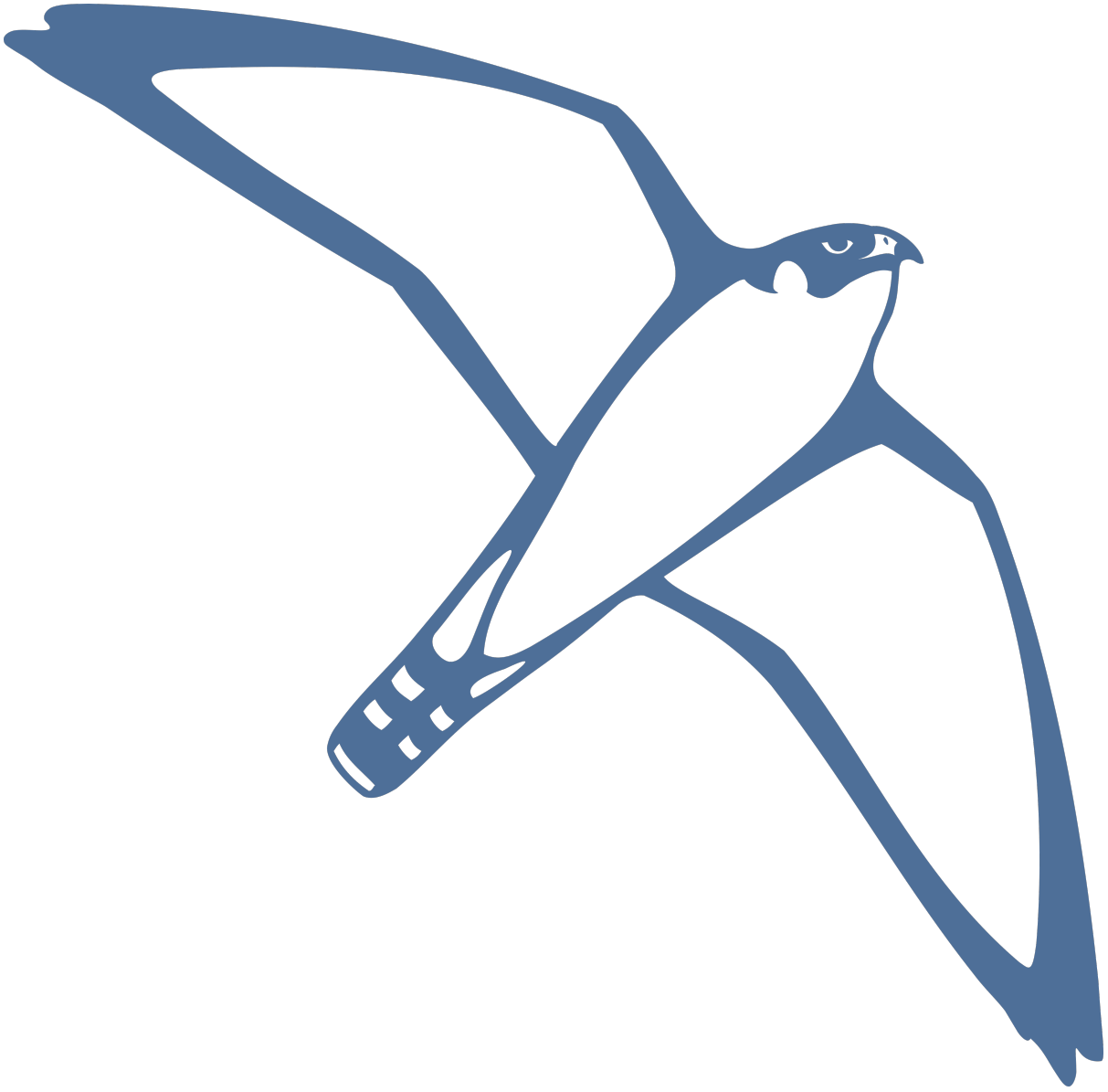


# GROMACS Documentation

***Release 2027.0-dev-20260918-b081421e***

**GROMACS development team**

**Sep 19, 2026**

# CONTENTS


**1 Downloads** **2**
- 1.1 Source code . . . 2
- 1.2 Regression tests . . . 2

**2 Installation guide** **3**
- 2.1 Installation guide for exotic configurations . . . 3
  - 2.1.1 Special instructions for building GROMACS on less-common systems . . . 3
- 2.2 Introduction to building GROMACS . . . 5
  - 2.2.1 Quick and dirty installation . . . 5
  - 2.2.2 Quick and dirty cluster installation . . . 6
  - 2.2.3 Typical installation . . . 6
  - 2.2.4 Building older versions . . . 7
- 2.3 Prerequisites . . . 7
  - 2.3.1 Platform . . . 7
  - 2.3.2 Compiler . . . 7
  - 2.3.3 Compiling with parallelization options . . . 8
  - 2.3.4 CMake . . . 12
  - 2.3.5 Fast Fourier Transform library . . . 12
  - 2.3.6 Other optional build components . . . 15
- 2.4 Doing a build of GROMACS . . . 16
  - 2.4.1 Configuring with CMake . . . 16
  - 2.4.2 Compiling and linking . . . 29
  - 2.4.3 Installing GROMACS . . . 29
  - 2.4.4 Getting access to GROMACS after installation . . . 30
  - 2.4.5 Testing GROMACS for correctness . . . 30
  - 2.4.6 Testing GROMACS for performance . . . 31
  - 2.4.7 Having difficulty? . . . 31
- 2.5 Special instructions for some platforms . . . 32
  - 2.5.1 Building on Windows . . . 32
  - 2.5.2 Building on Cray . . . 32
  - 2.5.3 NVIDIA Grace . . . 32
- 2.6 Tested platforms . . . 33
- 2.7 Support . . . 33

**3 User guide** **34**
- 3.1 Known issues affecting users of GROMACS . . . 34
  - 3.1.1 “Cannot find a working standard library” error with ROCm Clang . . . 34
  - 3.1.2 Expanded ensemble does not checkpoint correctly . . . 35
  - 3.1.3 Compiling with GCC 12-14 on POWER9 architectures . . . 35

3.1.4 NbnxmTest crash with oneAPI 2024.1 . . . 35
3.1.5 Severe performance regression with SVE and LLVM 20 . . . 35
3.1.6 CMake `OpenMP_CUDA` detection issue on Windows . . . 35
3.2 Getting started . . . 36
3.2.1 Flow Chart . . . 36
3.2.2 Setting up your environment . . . 38
3.2.3 Flowchart of typical simulation . . . 38
3.2.4 Important files . . . 38
3.2.5 Tutorial material . . . 40
3.2.6 Background reading . . . 40
3.3 System preparation . . . 40
3.3.1 Steps to consider . . . 41
3.3.2 Tips and tricks . . . 42
3.4 Managing long simulations . . . 42
3.4.1 Appending to output files . . . 43
3.4.2 Backing up your files . . . 43
3.4.3 Extending a .tpr file . . . 44
3.4.4 Changing mdp options for a restart . . . 44
3.4.5 Restarts without checkpoint files . . . 44
3.4.6 Are continuations exact? . . . 44
3.4.7 Reproducibility . . . 45
3.5 Answers to frequently asked questions (FAQs) . . . 46
3.5.1 Questions regarding GROMACS installation . . . 46
3.5.2 Questions concerning system preparation and preprocessing . . . 46
3.5.3 Questions regarding simulation methodology . . . 47
3.5.4 Parameterization and Force Fields . . . 48
3.5.5 Analysis and Visualization . . . 48
3.6 Force fields in GROMACS . . . 49
3.6.1 AMBER . . . 49
3.6.2 CHARMM . . . 49
3.6.3 GROMOS . . . 50
3.6.4 OPLS . . . 51
3.7 Molecular dynamics parameters (.mdp options) . . . 51
3.7.1 General information . . . 51
3.8 Useful mdrun features . . . 104
3.8.1 Re-running a simulation . . . 104
3.8.2 Running a simulation in reproducible mode . . . 104
3.8.3 Halting running simulations . . . 105
3.8.4 Running multi-simulations . . . 105
3.8.5 Controlling the length of the simulation . . . 106
3.9 Getting good performance from mdrun . . . 107
3.9.1 Performance checklist . . . 107
3.9.2 Hardware background information . . . 110
3.9.3 Work distribution by parallelization in GROMACS . . . 111
3.9.4 Parallelization schemes . . . 112
3.9.5 Running mdrun within a single node . . . 116
3.9.6 Running mdrun on more than one node . . . 121
3.9.7 Avoiding communication for constraints . . . 123
3.9.8 Finding out how to run mdrun better . . . 124
3.9.9 Running mdrun with GPUs . . . 126
3.9.10 Running the OpenCL version of mdrun . . . 131

3.9.11 Running SYCL version of mdrun . . . 132
3.9.12 Running HIP version of mdrun . . . 132
3.10 Common errors when using GROMACS . . . 132
3.10.1 Common errors during usage . . . 132
3.10.2 Errors in pdb2gmx . . . 133
3.10.3 Errors in grompp . . . 135
3.10.4 Errors in mdrun . . . 139
3.11 Command-line reference . . . 142
3.11.1 molecular dynamics simulation suite . . . 143
3.11.2 gmx anaeig . . . 151
3.11.3 gmx analyze . . . 154
3.11.4 gmx angle . . . 158
3.11.5 gmx awh . . . 160
3.11.6 gmx bar . . . 161
3.11.7 gmx bundle . . . 164
3.11.8 gmx check . . . 165
3.11.9 gmx chi . . . 167
3.11.10 gmx cluster . . . 171
3.11.11 gmx clustsize . . . 175
3.11.12 gmx confrms . . . 177
3.11.13 gmx convert-tpr . . . 178
3.11.14 gmx convert-trj . . . 179
3.11.15 gmx covar . . . 181
3.11.16 gmx current . . . 183
3.11.17 gmx density . . . 185
3.11.18 gmx densmap . . . 187
3.11.19 gmx densorder . . . 189
3.11.20 gmx dielectric . . . 191
3.11.21 gmx dipoles . . . 193
3.11.22 gmx disre . . . 196
3.11.23 gmx distance . . . 198
3.11.24 gmx dos . . . 200
3.11.25 gmx dssp . . . 202
3.11.26 gmx dump . . . 205
3.11.27 gmx dyecoupl . . . 206
3.11.28 gmx editconf . . . 208
3.11.29 gmx eneconv . . . 211
3.11.30 gmx enemat . . . 212
3.11.31 gmx energy . . . 214
3.11.32 gmx extract-cluster . . . 218
3.11.33 gmx filter . . . 220
3.11.34 gmx freevolume . . . 222
3.11.35 gmx gangle . . . 223
3.11.36 gmx genconf . . . 226
3.11.37 gmx genion . . . 227
3.11.38 gmx genrestr . . . 228
3.11.39 gmx grompp . . . 230
3.11.40 gmx gyrate . . . 233
3.11.41 gmx gyrate-legacy . . . 234
3.11.42 gmx h2order . . . 236
3.11.43 gmx hbond . . . 237

3.11.44 gmx hbond-legacy . . . 240
3.11.45 gmx helix . . . 244
3.11.46 gmx helixorient . . . 245
3.11.47 gmx help . . . 247
3.11.48 gmx hydorder . . . 247
3.11.49 gmx insert-molecules . . . 248
3.11.50 gmx lie . . . 250
3.11.51 gmx make_edi . . . 251
3.11.52 gmx make_ndx . . . 255
3.11.53 gmx mdmat . . . 256
3.11.54 gmx mdrun . . . 258
3.11.55 gmx mindist . . . 264
3.11.56 gmx mk_angndx . . . 266
3.11.57 gmx msd . . . 266
3.11.58 gmx nmeig . . . 269
3.11.59 gmx nmens . . . 271
3.11.60 gmx nmr . . . 272
3.11.61 gmx nmtraj . . . 274
3.11.62 gmx nonbonded-benchmark . . . 275
3.11.63 gmx order . . . 277
3.11.64 gmx pairdist . . . 279
3.11.65 gmx pdb2gmx . . . 281
3.11.66 gmx pme_error . . . 285
3.11.67 gmx polystat . . . 286
3.11.68 gmx potential . . . 287
3.11.69 gmx principal . . . 289
3.11.70 gmx rama . . . 290
3.11.71 gmx rdf . . . 291
3.11.72 gmx report-methods . . . 294
3.11.73 gmx rms . . . 294
3.11.74 gmx rmsdist . . . 297
3.11.75 gmx rmsf . . . 299
3.11.76 gmx rotacf . . . 301
3.11.77 gmx rotmat . . . 302
3.11.78 gmx saltbr . . . 304
3.11.79 gmx sans-legacy . . . 305
3.11.80 gmx sasa . . . 307
3.11.81 gmx saxs-legacy . . . 309
3.11.82 gmx scattering . . . 310
3.11.83 gmx select . . . 312
3.11.84 gmx sham . . . 315
3.11.85 gmx sigeps . . . 317
3.11.86 gmx solvate . . . 319
3.11.87 gmx sorient . . . 320
3.11.88 gmx spatial . . . 322
3.11.89 gmx spol . . . 325
3.11.90 gmx tcaf . . . 326
3.11.91 gmx traj . . . 328
3.11.92 gmx trajectory . . . 331
3.11.93 gmx trjcat . . . 333
3.11.94 gmx trjconv . . . 334

3.11.95 gmx trjorder . . . 339
3.11.96 gmx tune_pme . . . 340
3.11.97 gmx vanhove . . . 346
3.11.98 gmx velacc . . . 348
3.11.99 gmx wham . . . 349
3.11.100gmx wheel . . . 354
3.11.101gmx x2top . . . 355
3.11.102gmx xpm2ps . . . 357
3.11.103Command-line interface and conventions . . . 359
3.11.104Commands by name . . . 360
3.11.105Commands by topic . . . 363
3.11.106Special topics . . . 371
3.11.107Command changes between versions . . . 380
3.12 Terminology . . . 386
3.12.1 Pressure . . . 386
3.12.2 Periodic boundary conditions . . . 386
3.12.3 Thermostats . . . 387
3.12.4 Energy conservation . . . 389
3.12.5 Average structure . . . 389
3.12.6 Blowing up . . . 390
3.12.7 Diagnosing an unstable system . . . 391
3.12.8 Molecular dynamics . . . 392
3.12.9 Force field . . . 393
3.13 Validation . . . 393
3.13.1 Experimental features . . . 393
3.13.2 Features with validation pending . . . 394
3.13.3 Feature lifecycle stages . . . 395
3.14 Environment Variables . . . 396
3.14.1 Output Control . . . 396
3.14.2 Debugging . . . 397
3.14.3 Performance and Run Control . . . 398
3.14.4 OpenCL management . . . 403
3.14.5 SYCL management . . . 404
3.14.6 Analysis and Core Functions . . . 404
3.14.7 MDModules . . . 405
3.15 Floating point arithmetic . . . 405
3.16 Security when using GROMACS . . . 406
3.17 Policy for deprecating GROMACS functionality . . . 406

**4 Short How-To guides** **407**
4.1 Beginners . . . 407
4.1.1 Resources . . . 407
4.2 Adding a Residue to a Force Field . . . 407
4.2.1 Adding a new residue . . . 407
4.2.2 Modifying a force field . . . 408
4.3 Water solvation . . . 408
4.4 Non water solvent . . . 408
4.4.1 Making a non-aqueous solvent box . . . 409
4.5 Mixed solvent . . . 409
4.6 Making Disulfide Bonds . . . 409
4.7 Running membrane simulations in GROMACS . . . 410

4.7.1 Running Membrane Simulations . . . 410
4.7.2 Adding waters with genbox . . . 410
4.7.3 External material . . . 411
4.8 Parameterization of novel molecules . . . 411
4.8.1 Exotic Species . . . 412
4.9 Potential of Mean Force . . . 412
4.10 Single-Point Energy . . . 413
4.11 Carbon Nanotube . . . 413
4.11.1 Robert Johnson's Tips . . . 413
4.11.2 Andrea Minoia's tutorial . . . 414
4.12 Visualization Software . . . 415
4.12.1 Topology bonds vs Rendered bonds . . . 415
4.13 Extracting Trajectory Information . . . 415
4.14 External tools to perform trajectory analysis . . . 416
4.15 Plotting Data . . . 416
4.15.1 Software . . . 416
4.16 Micelle Clustering . . . 417

**5 Reference Manual 418**
5.1 Preface and Disclaimer . . . 418
5.1.1 Citation information . . . 419
5.1.2 GROMACS is *Free Software* . . . 419
5.2 Introduction . . . 420
5.2.1 Computational Chemistry and Molecular Modeling . . . 420
5.2.2 Molecular Dynamics Simulations . . . 421
5.2.3 Energy Minimization and Search Methods . . . 423
5.3 Definitions and Units . . . 425
5.3.1 Notation . . . 425
5.3.2 MD units . . . 425
5.3.3 Reduced units . . . 427
5.3.4 Mixed or Double precision . . . 427
5.4 Algorithms . . . 429
5.4.1 Periodic boundary conditions . . . 429
5.4.2 The group concept . . . 432
5.4.3 Molecular Dynamics . . . 433
5.4.4 Shell molecular dynamics . . . 460
5.4.5 Constraint algorithms . . . 461
5.4.6 Simulated Annealing . . . 465
5.4.7 Stochastic Dynamics . . . 465
5.4.8 Brownian Dynamics . . . 466
5.4.9 Energy Minimization . . . 467
5.4.10 Normal-Mode Analysis . . . 468
5.4.11 Free energy calculations . . . 469
5.4.12 Replica exchange . . . 471
5.4.13 Essential Dynamics sampling . . . 473
5.4.14 Expanded Ensemble . . . 473
5.4.15 Parallelization . . . 474
5.4.16 Domain decomposition . . . 474
5.5 Interaction function and force fields . . . 481
5.5.1 Non-bonded interactions . . . 481
5.5.2 Bonded interactions . . . 487

5.5.3 Restraints . . . 499
5.5.4 Polarization . . . 510
5.5.5 Free energy interactions . . . 511
5.5.6 Methods . . . 519
5.5.7 Virtual interaction sites . . . 520
5.5.8 Long Range Electrostatics . . . 525
5.5.9 Long Range Van der Waals interactions . . . 528
5.5.10 Force field . . . 532
5.6 Topologies . . . 537
5.6.1 Particle type . . . 537
5.6.2 Parameter files . . . 540
5.6.3 Molecule definition . . . 543
5.6.4 Constraint algorithms . . . 545
5.6.5 pdb2gmx input files . . . 545
5.6.6 File formats . . . 554
5.6.7 Force field organization . . . 570
5.7 File formats . . . 573
5.7.1 Summary of file formats . . . 573
5.7.2 File format details . . . 575
5.8 Special Topics . . . 590
5.8.1 Free energy implementation . . . 590
5.8.2 Potential of mean force . . . 591
5.8.3 Non-equilibrium pulling . . . 592
5.8.4 Collective variables: the pull code . . . 592
5.8.5 Adaptive biasing with AWH . . . 598
5.8.6 Enforced Rotation . . . 609
5.8.7 Electric fields . . . 619
5.8.8 Computational Electrophysiology . . . 620
5.8.9 Calculating a PMF using the free-energy code . . . 624
5.8.10 Removing fastest degrees of freedom . . . 624
5.8.11 Viscosity calculation . . . 627
5.8.12 Shear simulations . . . 629
5.8.13 Tabulated interaction functions . . . 630
5.8.14 Hybrid Quantum-Classical simulations (QM/MM) with CP2K interface . . . 631
5.8.15 MiMiC Hybrid Quantum Mechanical/Molecular Mechanical simulations . . . 635
5.8.16 Using VMD plug-ins for trajectory file I/O . . . 639
5.8.17 Interactive Molecular Dynamics . . . 640
5.8.18 Embedding proteins into the membranes . . . 641
5.8.19 Applying forces from three-dimensional densities . . . 642
5.8.20 Collective Variable simulations with the Colvars module . . . 646
5.8.21 Using PLUMED . . . 647
5.8.22 Neural Network Potentials . . . 647
5.8.23 Fast Multipole Method (FMM) . . . 650
5.9 Run parameters and Programs . . . 652
5.9.1 Online documentation . . . 652
5.9.2 File types . . . 652
5.9.3 Run Parameters . . . 652
5.10 Analysis . . . 653
5.10.1 Using Groups . . . 653
5.10.2 Looking at your trajectory . . . 656
5.10.3 General properties . . . 656

5.10.4 Radial distribution functions . . . 657
5.10.5 Correlation functions . . . 658
5.10.6 Curve fitting in GROMACS . . . 660
5.10.7 Mean Square Displacement . . . 662
5.10.8 Bonds/distances, angles and dihedrals . . . 662
5.10.9 Radius of gyration and distances . . . 664
5.10.10 Root mean square deviations in structure . . . 665
5.10.11 Covariance analysis . . . 666
5.10.12 Dihedral principal component analysis . . . 667
5.10.13 Hydrogen bonds . . . 668
5.10.14 Protein-related items . . . 669
5.10.15 Interface-related items . . . 671
5.11 Some implementation details . . . 673
5.11.1 Single Sum Virial in GROMACS . . . 673
5.11.2 Optimizations . . . 676
5.12 Averages and fluctuations . . . 678
5.12.1 Formulae for averaging . . . 678
5.12.2 Implementation . . . 679
5.13 Bibliography . . . 683

**6 gmxapi Python package** **694**
6.1 Full installation instructions . . . 694
6.1.1 Overview . . . 695
6.1.2 Background . . . 696
6.1.3 Installing the Python package . . . 699
6.1.4 Accessing gmxapi documentation . . . 705
6.1.5 Testing . . . 705
6.1.6 Troubleshooting . . . 706
6.2 Using the Python package . . . 708
6.2.1 Notes on parallelism and MPI . . . 709
6.2.2 Running simple simulations . . . 710
6.2.3 Running ensemble simulations . . . 711
6.2.4 Input arguments and “ensemble” syntax . . . 711
6.2.5 Accessing command line tools . . . 712
6.2.6 Preparing simulations . . . 712
6.2.7 Using arbitrary Python functions . . . 713
6.2.8 Subgraphs . . . 714
6.2.9 Looping . . . 714
6.2.10 Logging . . . 715
6.2.11 More . . . 716
6.3 gmxapi Python module reference . . . 716
6.3.1 Interface concepts . . . 716
6.3.2 gmxapi basic package . . . 718
6.3.3 Simulation module . . . 722
6.3.4 Utilities . . . 724
6.3.5 Status messages and Logging . . . 726
6.3.6 Exceptions module . . . 727
6.3.7 gmx.version module . . . 728
6.3.8 Core API . . . 730

**7 (Non-)Bonded LIBrary (NB-LIB) API** **734**
7.1 Guide to Writing MD Programs . . . 734

7.1.1 Global Definitions . . . 734
7.1.2 Define Particle Data . . . 735
7.1.3 Defining Coordinates, Velocities and Force Buffers . . . 736
7.1.4 Writing the MD Program . . . 737

**8 Developer Guide** **742**
8.1 Contribute to GROMACS . . . 742
8.1.1 Checklist . . . 743
8.1.2 Preparing code for submission . . . 744
8.1.3 Alternatives . . . 744
8.1.4 Do you have more questions? . . . 745
8.1.5 Removing functionality . . . 745
8.2 Codebase overview . . . 745
8.2.1 Source code organization . . . 746
8.2.2 Documentation organization . . . 748
8.3 Build system overview . . . 750
8.3.1 Build types . . . 750
8.3.2 CMake cache variables . . . 751
8.3.3 External libraries . . . 756
8.3.4 Special targets . . . 756
8.3.5 Passing information to source code . . . 758
8.4 Change Management . . . 759
8.4.1 Getting started . . . 759
8.4.2 Labels . . . 761
8.4.3 Code Review . . . 761
8.4.4 More git tips . . . 763
8.5 Relocatable binaries . . . 766
8.5.1 Finding shared libraries . . . 766
8.5.2 Finding data files . . . 767
8.5.3 Known issues . . . 768
8.6 Documentation generation . . . 769
8.6.1 Building the GROMACS documentation . . . 769
8.6.2 Needed build tools . . . 770
8.7 Style guidelines . . . 771
8.7.1 Guidelines for code formatting . . . 771
8.7.2 Guidelines for #include directives . . . 772
8.7.3 Naming conventions . . . 773
8.7.4 Allowed language features . . . 776
8.7.5 Guidelines for creating meaningful issue reports . . . 781
8.7.6 Guidelines for formatting of git commits . . . 783
8.7.7 Error handling . . . 784
8.8 Development-time tools . . . 786
8.8.1 Using Doxygen . . . 786
8.8.2 Automation and Infrastructure . . . 802
8.8.3 Source tree checker scripts . . . 815
8.8.4 Automatic source code formatting . . . 818
8.8.5 Unit testing . . . 824
8.8.6 Physical validation . . . 827
8.9 Known issues relevant for developers . . . 830
8.9.1 Issues with GPU timer with OpenCL . . . 831
8.9.2 GPU emulation does not work . . . 831

8.9.3 OpenCL on NVIDIA Volta and later broken . . . 831
8.9.4 PME decomposition automated task assignment broken . . . 831

**9 Doxygen documentation** **832**

**10 C++ API** **833**
10.1 Public C++ API . . . 833
10.1.1 Overview . . . 833
10.1.2 Client build system support . . . 833
10.1.3 `gmxapi` CMake package . . . 834
10.1.4 `gromacs` (and `gromacs$GROMACS_SUFFIX` packages) . . . 835

**11 Release notes** **838**
11.1 GROMACS 2027 series . . . 838
11.1.1 Patch releases . . . 838
11.1.2 Major release . . . 838
11.2 GROMACS 2026 series . . . 841
11.2.1 Patch releases . . . 841
11.2.2 Major release . . . 851
11.3 Older (unmaintained) GROMACS series . . . 861
11.4 GROMACS 2025 series . . . 861
11.4.1 Patch releases . . . 861
11.4.2 Major release . . . 872
11.5 GROMACS 2024 series . . . 880
11.5.1 Patch releases . . . 880
11.5.2 Major release . . . 896
11.6 GROMACS 2023 series . . . 903
11.6.1 Patch releases . . . 903
11.6.2 Major release . . . 918
11.7 GROMACS 2022 series . . . 926
11.7.1 Patch releases . . . 926
11.7.2 Major release . . . 940
11.8 GROMACS 2021 series . . . 954
11.8.1 Patch releases . . . 954
11.8.2 Major release . . . 966
11.9 GROMACS 2020 series . . . 977
11.9.1 Patch releases . . . 977
11.9.2 Major release . . . 991
11.10 GROMACS 2019 series . . . 999
11.10.1 Patch releases . . . 999
11.10.2 Major release . . . 1013
11.11 GROMACS 2018 series . . . 1021
11.11.1 Patch releases . . . 1021
11.11.2 Major release . . . 1039
11.12 GROMACS 2016 series . . . 1056
11.12.1 Patch releases . . . 1056
11.12.2 Major release . . . 1076

**Python Module Index** **1095**

The release notes can be found online at http://manual.gromacs.org/current/release-notes/index.html

CHAPTER

ONE

# DOWNLOADS

This is not a release build of GROMACS, so please reference one of the GROMACS papers and the base release of the manual.

This is not a release build of GROMACS. Please reference one of the GROMACS papers, as well as the base release that this version is built from. Also, please state what modifcations have been performed or where the version was sourced from.

## 1.1 Source code

- As ftp ftp://ftp.gromacs.org/gromacs/gromacs-2027.0-dev.tar.gz
- As https https://ftp.gromacs.org/gromacs/gromacs-2027.0-dev.tar.gz
- (md5sum unknown)

Other source code versions may be found at the web site.

## 1.2 Regression tests

- https://ftp.gromacs.org/regressiontests/regressiontests-2027.0.tar.gz
- (md5sum unknown)

CHAPTER

TWO

# INSTALLATION GUIDE

## 2.1 Installation guide for exotic configurations

### 2.1.1 Special instructions for building GROMACS on less-common systems

These instructions pertain to building GROMACS 2027.0-dev. This document is complementary to the up-to-date installation instructions instructions.

The configurations listed here are expected to work, but **are not recommended for typical users**.

#### SYCL GPU acceleration for AMD and NVIDIA GPUs using Intel oneAPI DPC++

AMD and NVIDIA GPUs can also be used with Intel oneAPI BaseKit and Codeplay oneAPI plugins.

For most users, we recommend using *CUDA* (page 20) for NVIDIA GPUs and *AdaptiveCpp* (page 22) for AMD GPUs instead.

#### AMD GPUs

After installing Intel oneAPI toolkit 2024.0 or newer, a compatible ROCm version, and the Codeplay plugin, set up the environment by running `source /opt/intel/oneapi/setvars.sh` or loading an appropriate **`module load`** on an HPC system.

Then, configure GROMACS using the following command (replace `gfxXYZ` with the target architecture):

```
cmake .. -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx \
         -DGMX_GPU=SYCL -DGMX_SYCL=DPCPP \
         -DGMX_GPU_NB_CLUSTER_SIZE=8 -DGMX_GPU_FFT_LIBRARY=vkfft \
         -DSYCL_CXX_FLAGS_EXTRA='-fsycl-targets=amd_gpu_gfxXYZ'
```

### NVIDIA GPUs

After installing Intel oneAPI toolkit 2024.0 or newer, a compatible CUDA version, and the Codeplay plugin, set up the environment by running `source /opt/intel/oneapi/setvars.sh` or loading an appropriate **`module load`** on an HPC system.

Then, configure GROMACS using the following command:

```
cmake .. -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx \
         -DGMX_GPU=SYCL -DGMX_SYCL=DPCPP \
         -DGMX_GPU_NB_CLUSTER_SIZE=8 -DGMX_GPU_FFT_LIBRARY=vkfft \
         -DSYCL_CXX_FLAGS_EXTRA=-fsycl-targets=nvptx64-nvidia-cuda
```

For more recent NVIDIA GPUs, compiling for a specific compute capability can be beneficial for performance. This is possible by setting the `-fsycl-targets` parameter of `SYCL_CXX_FLAGS_EXTRA`. For example for an Ampere architecture GPU such as the NVIDIA A100, set `-DSYCL_CXX_FLAGS_EXTRA=-fsycl-targets=nvidia_gpu_sm_80`. Possible values are given in the DPC++ user manual.

### SYCL GPU acceleration for NVIDIA GPUs using AdaptiveCpp

For most users, we recommend using *CUDA* (page 20) for NVIDIA GPUs.

Build and install AdaptiveCpp with CUDA backend (we recommend using the mainline Clang, not the ROCm-bundled one).

Then, use the following command to build GROMACS (make sure to use the same compiler and set target GPU architecture instead of `sm_XY`):

```
cmake .. -DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++ \
         -DGMX_GPU=SYCL -DGMX_SYCL=ACPP -DACPP_TARGETS='cuda:sm_XY'
```

### Static linking

Dynamic linking of the GROMACS executables will lead to a smaller disk footprint when installed, and so is the default on platforms where we believe it has been tested repeatedly and found to work. In general, this includes Linux, Windows, Mac OS X and BSD systems. Static binaries take more space, but on some hardware and/or under some conditions they are recommended or even necessary, most commonly when you are running large parallel simulation using MPI libraries (e.g. Cray).

- To link GROMACS binaries statically against the internal GROMACS libraries, set `-DBUILD_SHARED_LIBS=OFF`.
- To link statically against external (non-system) libraries as well, set `-DGMX_PREFER_STATIC_LIBS=ON`. Note, that in general `cmake` picks up whatever is available, so this option only instructs `cmake` to prefer static libraries when both static and shared are available. If no static version of an external library is available, even when the aforementioned option is `ON`, the shared library will be used. Also note that the resulting binaries will still be dynamically linked against system libraries on platforms where that is the default. To use static system libraries, additional compiler/linker flags are necessary, e.g. `-static-libgcc -static-libstdc++`.

- To attempt to link a fully static binary set `-DGMX_BUILD_SHARED_EXE=OFF`. This will prevent CMake from explicitly setting any dynamic linking flags. This option also sets `-DBUILD_SHARED_LIBS=OFF` and `-DGMX_PREFER_STATIC_LIBS=ON` by default, but the above caveats apply. For compilers which don't default to static linking, the required flags have to be specified. On Linux, this is usually `CFLAGS=-static CXXFLAGS=-static`.

#### Building on Solaris

The built-in GROMACS processor detection does not work on Solaris, so it is strongly recommended that you build GROMACS with `-DGMX_HWLOC=on` and ensure that the `CMAKE_PREFIX_PATH` includes the path where the hwloc headers and libraries can be found. Version 2.0 or later of hwloc is required.

#### RISC-V with VEC unit

GROMACS runs on RISC-V. The non-bonded kernel can be ran on the VEC vector unit, when available. To enable this, add `-DGMX_ENABLE_NBNXM_CPU_VECTORIZATION=on` to the CMake flags. A clang compiler is required with version >=19.

If you want to check which loops have been vectorized, add `-Rpass=loop-vectorize -Rpass-missed=loop-vectorize -Rpass-analysis=loop-vectorize` to the `CMAKE_CXX_FLAGS`.

Please note the early state and instability of the compilers at the time of writing. If you experience errors, try adding `-fno-vectorize`, after `-O3` when present, to `CMAKE_CXX_FLAGS` to disable vectorization of the code that's not explicitly vectorized.

## 2.2 Introduction to building GROMACS

These instructions pertain to building GROMACS 2027.0-dev. You might also want to check the up-to-date installation instructions.

### 2.2.1 Quick and dirty installation

1. Get the latest version of your C and C++ compilers.
2. Check that you have CMake version 3.28 or later.
3. Get and unpack the latest version of the GROMACS tarball.
4. Make a separate build directory and change to it.
5. Run `cmake` with the path to the source as an argument
6. Run `make`, `make check`, and `make install`
7. Source `GMXRC` to get access to GROMACS

Or, as a sequence of commands to execute:

```
tar xfz gromacs-2027.0-dev.tar.gz
cd gromacs-2027.0-dev
mkdir build
```

```
cd build
cmake .. -DGMX_BUILD_OWN_FFTW=ON -DREGRESSIONTEST_DOWNLOAD=ON
make
make check
sudo make install
source /usr/local/gromacs/bin/GMXRC
```

This will download and build first the prerequisite FFT library followed by GROMACS. If you already have FFTW installed, you can remove that argument to `cmake`. Overall, this build of GROMACS will be correct and reasonably fast on the machine upon which `cmake` ran. On another machine, it may not run, or may not run fast. If you want to get the maximum value for your hardware with GROMACS, you will have to read further. Sadly, the interactions of hardware, libraries, and compilers are only going to continue to get more complex.

### 2.2.2 Quick and dirty cluster installation

On a cluster where users are expected to be running across multiple nodes using MPI, make one installation similar to the above, and another using `-DGMX_MPI=on`. The latter will install binaries and libraries named using a default suffix of `_mpi` ie `gmx_mpi`. Hence it is safe and common practice to install this into the same location where the non-MPI build is installed.

### 2.2.3 Typical installation

As above, and with further details below, but you should consider using the following *CMake options* (page 17) with the appropriate value instead of `xxx` :

- `-DCMAKE_C_COMPILER=xxx` equal to the name of the C99 *compiler* (page 7) you wish to use (or the environment variable `CC`)
- `-DCMAKE_CXX_COMPILER=xxx` equal to the name of the C++17 *compiler* (page 7) you wish to use (or the environment variable `CXX`)
- `-DGMX_MPI=on` to build using *MPI support* (page 10)
- `-DGMX_GPU=CUDA` to build with NVIDIA CUDA support enabled.
- `-DGMX_GPU=OpenCL` to build with OpenCL support enabled.
- `-DGMX_GPU=SYCL` to build with SYCL support enabled (using Intel oneAPI DPC++ by default).
- `-DGMX_SYCL=ACPP` to build with SYCL support using AdaptiveCpp (hipSYCL), requires `-DGMX_GPU=SYCL`.
- `-DGMX_SIMD=xxx` to specify the level of *SIMD support* (page 18) of the node on which GROMACS will run
- `-DGMX_DOUBLE=on` to build GROMACS in double precision (slower, and not normally useful)
- `-DCMAKE_PREFIX_PATH=xxx` to add a non-standard location for CMake to *search for libraries, headers or programs* (page 19)
- `-DCMAKE_INSTALL_PREFIX=xxx` to install GROMACS to a *non-standard location* (page 17) (default `/usr/local/gromacs`)
- `-DBUILD_SHARED_LIBS=off` to turn off the building of shared libraries to help with *static linking* (page 4)

- `-DGMX_FFT_LIBRARY=xxx` to select whether to use `fftw3`, `mkl` or `fftpack` libraries for *FFT support* (page 12)
- `-DCMAKE_BUILD_TYPE=Debug` to build GROMACS in debug mode

### 2.2.4 Building older versions

Installation instructions for old GROMACS versions can be found at the GROMACS documentation page.

## 2.3 Prerequisites

### 2.3.1 Platform

GROMACS can be compiled for many operating systems and architectures. These include any distribution of Linux, macOS or Windows, and architectures including 64-bit x86 (AMD64/x86-64), several PowerPC including POWER9, ARM v8, and RISC-V.

### 2.3.2 Compiler

GROMACS can be compiled on any platform with ANSI C99 and C++17 compilers, and their respective standard C/C++ libraries. Good performance on an OS and architecture requires choosing a good compiler. We recommend gcc, because it is free, widely available and frequently provides the best performance.

You should strive to use the most recent version of your compiler. Since we require full C++17 support the minimum compiler versions supported by the GROMACS team are

- GNU (gcc/libstdc++) 11
- LLVM (clang/libc++) 14
- Microsoft (MSVC) 2019

Other compilers may work (Cray, Pathscale, older clang) but do not offer competitive performance. We recommend against PGI because the performance with C++ is very bad.

The Intel classic compiler (icc/icpc) is no longer supported in GROMACS. Use Intel's newer clang-based compiler from oneAPI, or gcc.

The xlc compiler is not supported and version 16.1 does not compile on POWER architectures for GROMACS-2027.0-dev. We recommend to use the GCC compiler. Note: there are *known issues* (page 34) with some versions of GCC 12-14.

You may also need the most recent version of other compiler toolchain components beside the compiler itself (e.g. assembler or linker); these are often shipped by your OS distribution's binutils package.

C++17 support requires adequate support in both the compiler and the C++ library. The gcc and MSVC compilers include their own standard libraries and require no further configuration. If your vendor's compiler also manages the standard library library via compiler flags, these will be honored. For configuration of other compilers, read on.

On Linux, the clang compilers typically use for their C++ library the libstdc++ which comes with g++. For GROMACS, we require the compiler to support libstc++ version 11 or higher. To select a particular

libstdc++ library for a compiler whose default standard library does not work, provide the path to g++ with `-DGMX_GPLUSPLUS_PATH=/path/to/g++`. Note that if you then build a further project that depends on GROMACS you will need to arrange to use the same compiler and libstdc++.

To build with clang and llvm's libcxx standard library, use `-DCMAKE_CXX_FLAGS=-stdlib=libc++`.

If you are running on Mac OS X, Apple has unfortunately explicitly disabled OpenMP support in their Clang-based compiler, and running without OpenMP support means you would need to use thread-MPI for any parallelism - which is the reason the GROMACS configuration script now stops rather than just issues a warning you might miss. Instead of turning off OpenMP, you can try to download the unsupported libomp distributed by the R project or compile your own version - but this will likely have to be updated any time you upgrade the major Mac OS version. Alternatively, you can download a version of gcc; just make sure you actually use your downloaded gcc version, since Apple by default links /usr/bin/gcc to their own compiler.

For all non-x86 platforms, your best option is typically to use gcc or the vendor's default or recommended compiler, and check for specialized information below.

For updated versions of gcc to add to your Linux OS, see

- Ubuntu: Ubuntu toolchain ppa page
- RHEL/CentOS: EPEL page or the RedHat Developer Toolset

### 2.3.3 Compiling with parallelization options

For maximum performance you will need to examine how you will use GROMACS and what hardware you plan to run on. Often OpenMP parallelism is an advantage for GROMACS, but support for this is generally built into your compiler and detected automatically.

#### GPU support

GROMACS supports a variety of GPU acceleration options. For end-users, here are the recommended options based on your hardware:

- AMD GPUs: SYCL (with AdaptiveCpp)
- Apple M-series: OpenCL
- Intel GPUs: SYCL (with Intel oneAPI DPC++)
- NVIDIA GPUs: CUDA

#### CUDA

CUDA is the recommended backend for NVIDIA GPUs.

Supported hardware:

- NVIDIA GPUs (all supported by the CUDA toolkit)

Requirements:

- CUDA toolkit version 12.1 or newer
- GPU with compute capability 5.0 or higher

Best practices:

- Use the latest CUDA version and NVIDIA driver compatible with your hardware
- Match your gcc version with nvcc's host compiler (prefer the latest gcc/clang version supported by nvcc)

More information can be found in the *CUDA GPU acceleration* (page 20) section.

### OpenCL

OpenCL is deprecated, but is currently the only backend supporting Apple M-series GPUs.

Supported hardware:

- AMD GCN-based GPUs (RDNA-series GPUs, such as RX 5500 or RX 6900, are not supported)
- Apple M-series GPUs
- Intel GPUs (special compilation options required; Intel DataCenter GPU Max are not supported)
- NVIDIA GPUs (only prior to Volta architecture; newer GPUs, such as V100 or GTX 10xx-series, are not supported)

Requirements:

- The minimum OpenCL version unknown.

More information can be found in the *OpenCL GPU acceleration* (page 20) section.

### SYCL

SYCL is the recommended backend for Intel and AMD GPUs. For Intel GPUs, we recommend using the Intel oneAPI DPC++ compiler, while for AMD GPUs we recommend using AdaptiveCpp compiler with ROCm runtime.

Supported hardware:

- AMD GPUs: GFX9 (Vega, Raven), CDNA-series, RDNA-series GPUs (using either oneAPI or AdaptiveCpp)
- Intel GPUs: All current integrated/discrete GPUs (using oneAPI)
- NVIDIA GPUs: All GPUs (using either oneAPI or AdaptiveCpp)

Requirements:

- oneAPI DPC++ compiler: 2024.0 or newer (Codeplay plugin required for NVIDIA/AMD Support), or
- AdaptiveCpp: 24.02 or newer.

Limitations:

- Intel GPUs and SSCP/generic compilation flow not supported with AdaptiveCpp.
- ROCm or CUDA toolkits are required for AMD and NVIDIA GPUs respectively.

More information can be found in the *SYCL GPU acceleration Intel* (page 22) and *SYCL GPU acceleration AMD* (page 22) sections.

### HIP

Supported hardware:

- AMD GPUs: GFX9, CDNA 1/2,3 RDNA 1/2/3 GPUs

Requirements:

- ROCm runtime 5.2 or newer

Limitations:

- Available from GROMACS 2025
- GROMACS 2025 supports only main non-bonded kernels
- GROMACS 2026 has full offload support

Experimental branch: * Experimental feature branch with remaining AMD specific optimizations: HIP feature branch * Supported by AMD, with aim to merge all remaining code during the coming release cycle * Updated together with 2026 releases to be based on latest fixes * For support, contact acmnpv here

More information can be found in the *AMD-HIP* (page 23) section.

### Important Notes

- Only one GPU backend can be configured per build
- CPU code always runs alongside GPU acceleration
- Choose latest drivers while watching for performance regressions on older hardware

### MPI support

GROMACS can run in parallel on multiple cores of a single workstation using its built-in thread-MPI. No user action is required in order to enable this.

If you wish to run in parallel on multiple machines across a network, you will need to have an MPI library installed that supports the MPI 3.0 standard. That's true for any MPI library version released since about 2009, but the GROMACS team recommends the latest version (for best performance) of either your vendor's library, OpenMPI or MPICH.

To compile with MPI set your compiler to the normal (non-MPI) compiler and add `-DGMX_MPI=on` to the cmake options. It is possible to set the compiler to the MPI compiler wrapper but it is neither necessary nor recommended.

### GPU-aware MPI support

In simulations using multiple GPUs, an MPI implementation with GPU support allows communication to be performed directly between the distinct GPU memory spaces without staging through CPU memory, often resulting in higher bandwidth and lower latency communication. The only current support for this in GROMACS is with a CUDA build targeting Nvidia GPUs using "CUDA-aware" MPI libraries. For more details, see Introduction to CUDA-aware MPI.

To use CUDA-aware MPI for direct GPU communication we recommend using the latest OpenMPI version (>=4.1.0) with the latest UCX version (>=1.10), since most GROMACS internal testing on CUDA-aware support has been performed using these versions. OpenMPI with CUDA-aware support can be built following the procedure in these OpenMPI build instructions.

For GPU-aware MPI support of Intel GPUs, use Intel MPI no earlier than version 2018.8. Such a version is found in the oneAPI SDKs starting from version 2023.0. At runtime, the LevelZero SYCL backend must be used (setting environment variable `ONEAPI_DEVICE_SELECTOR=level_zero:gpu` will typically suffice) and GPU-aware support in the MPI runtime selected.

For GPU-aware MPI support on AMD GPUs, several MPI implementations with UCX support can work, we recommend the latest OpenMPI version (>=4.1.4) with the latest UCX (>=1.13) since most of our testing was done using these version. Other MPI flavors such as Cray MPICH are also GPU-aware and compatible with ROCm.

With `GMX_MPI=ON`, GROMACS attempts to automatically detect GPU support in the underlying MPI library at compile time, and enables direct GPU communication when this is detected. However, there are some cases when GROMACS may fail to detect existing GPU-aware MPI support, in which case it can be manually enabled by setting environment variable `GMX_FORCE_GPU_AWARE_MPI=1` at runtime (although such cases still lack substantial testing, so we urge the user to carefully check correctness of results against those using default build options, and report any issues).

### NVSHMEM Support for GPU kernel-initiated communication

In simulations using multiple GPUs, NVSHMEM provides a programming interface that allows GPU-initiated direct communication between distinct GPU memory spaces. This approach leverages NVSHMEM's global address space, often resulting in higher bandwidth and lower latency communication.

Support for this feature in GROMACS is currently enabled by building with NVSHMEM support (`GMX_NVSHMEM=ON`) and specifying the NVSHMEM root directory (`NVSHMEM_ROOT=<Path-to-NVSHMEM-Lib-Root-dir>`). This setup targets NVIDIA GPUs and utilizes the NVSHMEM library for efficient inter-GPU data transfers. It should be noted that the NVSHMEM build is not compatible with cuFFTMp, an issue that may be resolved in a future release.

This is an experimental feature. The current implementation performs well for small system sizes (up to 300,000 particles). Performance improvements are planned for future releases, where we expect the NVSHMEM-based implementation to be faster across all input sizes compared to MPI.

### 2.3.4 CMake

GROMACS builds with the CMake build system, requiring at least version 3.28. You can check whether CMake is installed, and what version it is, with `cmake --version`. If you need to install CMake, then first check whether your platform's package management system provides a suitable version, or visit the CMake installation page for pre-compiled binaries, source code and installation instructions. The GROMACS team recommends you install the most recent version of CMake you can.

### 2.3.5 Fast Fourier Transform library

Many simulations in GROMACS make extensive use of fast Fourier transforms, and a software library to perform these is always required. We recommend FFTW (version 3 or higher only) or Intel MKL. The choice of library can be set with `cmake -DGMX_FFT_LIBRARY=<name>`, where `<name>` is one of `fftw3`, `mkl`, or `fftpack`. FFTPACK is bundled with GROMACS as a fallback, and is acceptable if simulation performance is not a priority. When choosing MKL, GROMACS will also use MKL for BLAS and LAPACK (see *linear algebra libraries* (page 26)). Generally, there is no advantage in using MKL with GROMACS, and FFTW is often faster. With PME GPU offload support using CUDA, a GPU-based FFT library is required. The CUDA-based GPU FFT library cuFFT is part of the CUDA toolkit (required for all CUDA builds) and therefore no additional software component is needed when building with CUDA GPU acceleration.

#### Using FFTW

FFTW is likely to be available for your platform via its package management system, but there can be compatibility and significant performance issues associated with these packages. In particular, GROMACS simulations are normally run in "mixed" floating-point precision, which is suited for the use of single precision in FFTW. The default FFTW package is normally in double precision, and good compiler options to use for FFTW when linked to GROMACS may not have been used. Accordingly, the GROMACS team recommends either

- that you permit the GROMACS installation to download and build FFTW from source automatically for you (use `cmake -DGMX_BUILD_OWN_FFTW=ON`), or
- that you build FFTW from the source code.

If you build FFTW from source yourself, get the most recent version and follow the FFTW installation guide. Choose the precision for FFTW (i.e. single/float vs. double) to match whether you will later use mixed or double precision for GROMACS. There is no need to compile FFTW with threading or MPI support, but it does no harm. On x86 hardware, compile with all of `--enable-sse2`, `--enable-avx`, and `--enable-avx2` flags. On Intel processors supporting 512-wide AVX, add `--enable-avx512` too. FFTW will create a fat library with codelets for all different instruction sets, and pick the fastest supported one at runtime. On ARM architectures with SIMD support use `--enable-neon` flag; on IBM Power8 and later, use `--enable-vsx` flag. If you are using a Cray, there is a special modified (commercial) version of FFTs using the FFTW interface which can be slightly faster.

Relying on `-DGMX_BUILD_OWN_FFTW=ON` works well in typical situations, but does not work on Windows, when using `ninja` build system, when cross-compiling, with custom toolchain configurations, etc. In such cases, please build FFTW manually.

### Using MKL

To target either Intel CPUs or GPUs, use OneAPI MKL (>=2021.3) by setting up the environment, e.g., through `source /opt/intel/oneapi/setvars.sh` or `source /opt/intel/oneapi/mkl/latest/env/vars.sh` or manually setting environment variable `MKLROOT=/full/path/to/mkl`. Then run CMake with setting `-DGMX_FFT_LIBRARY=mkl` and/or `-DGMX_GPU_FFT_LIBRARY=mkl`.

### Using oneMath Interface Library

The oneMath interface library (earlier called oneMKL interface library, not to be confused with Intel oneMKL) enables the SYCL backend for GROMACS with cuFFT, rocFFT, or closed-source oneMath using Intel DPC++ and Codeplay's plugins for NVIDIA and AMD GPUs. To use, Intel DPC++ must be installed along with Codeplay's plugins for NVIDIA and AMD GPUs as required, and CUDA and/or ROCm as required. The environment should be initialized as with the MKL instructions above.

To use the oneMath interface library, download, build and install oneMath as directed in the oneMath documentation, making sure that suitable DFT backends are enabled. Then, when building GROMACS, set `-DGMX_GPU_FFT_LIBRARY=ONEMATH`.

### Using double-batched FFT library

Generally MKL will provide better performance on Intel GPUs, however this alternative open-source library from Intel is useful for very large FFT sizes in GROMACS.

```
cmake -DGMX_GPU_FFT_LIBRARY=BBFFT \
      -DCMAKE_PREFIX_PATH=$PATH_TO_BBFFT_INSTALL
```

Note: in GROMACS 2023, the option was called `DBFFT`.

### Using ARM Performance Libraries

The ARM Performance Libraries provides FFT transforms implementation for ARM architectures. Preliminary support is provided for ARMPL in GROMACS through its FFTW-compatible API. Assuming that the ARM HPC toolchain environment including the ARMPL paths are set up (e.g. through loading the appropriate modules like `module load Module-Prefix/arm-hpc-compiler-X.Y/armpl/X.Y`) use the following cmake options:

```
cmake -DGMX_FFT_LIBRARY=fftw3 \
      -DFFTWF_LIBRARY="${ARMPL_DIR}/lib/libarmpl_lp64.so" \
      -DFFTWF_INCLUDE_DIR=${ARMPL_DIR}/include
```

### Using cuFFTMp

Decomposition of PME work to multiple GPUs is supported with NVIDIA GPUs when using a CUDA build. This requires building GROMACS with the NVIDIA cuFFTMp (cuFFT Multi-process) library, shipped with the NVIDIA HPC SDK, which provides distributed FFTs including across multiple compute nodes. To enable cuFFTMp support use the following cmake options:

```
cmake -DGMX_USE_CUFFTMP=ON \
      -DcuFFTMp_ROOT=<path to NVIDIA HPC SDK math_libs folder>
```

Please make sure cuFFTMp's hardware and software requirements are met before trying to use GPU PME decomposition feature. In particular, cuFFTMp internally uses NVSHMEM, and it is vital that the NVSHMEM and cuFFTMp versions in use are compatible. Some versions of the NVIDIA HPC SDK include two versions of NVSHMEM, where the cuFFTMp compatible variant can be found at `Linux_x86_64/<SDK_version>/comm_libs/<CUDA_version>/nvshmem_cufftmp_compat`. If that directory does not exist in the SDK, then there only exists a single (compatible) version at `Linux_x86_64/<SDK_version>/comm_libs/<CUDA_version>/nvshmem`. The version can be selected by, prior to both compilation and running, updating the LD_LIBRARY_PATH environment variable as follows:

```
export LD_LIBRARY_PATH=<path to compatible NVSHMEM folder>/lib:$LD_
↪LIBRARY_PATH
```

It is advisable to refer to the NVSHMEM FAQ page for any issues faced at runtime.

### Using heFFTe

Decomposition of PME work to multiple GPUs is supported with PME offloaded to any vendor's GPU when building GROMACS linked to the heFFTe library. HeFFTe uses GPU-aware MPI to provide distributed FFTs including across multiple compute nodes. It requires a CUDA build to target NVIDIA GPUs and a SYCL build to target Intel or AMD GPUs. To enable heFFTe support, use the following cmake options:

```
cmake -DGMX_USE_HEFFTE=ON \
      -DHeffte_ROOT=<path to heFFTe folder>
```

You will need an installation of heFFTe configured to use the same GPU-aware MPI library that will be used by GROMACS, and with support that matches the intended GROMACS build. It is best to use the same C++ compiler and standard library also. When targeting Intel GPUs, add `-DHeffte_ENABLE_ONEAPI=ON -DHeffte_ONEMKL_ROOT=<path to oneMKL folder>`. When targeting AMD GPUs, add `-DHeffte_ENABLE_ROCM=ON -DHeffte_ROCM_ROOT=<path to ROCm folder>`.

#### Using VkFFT

VkFFT is a multi-backend GPU-accelerated multidimensional Fast Fourier Transform library which aims to provide an open-source alternative to vendor libraries.

GROMACS includes VkFFT support with two goals: portability across GPU platforms and performance improvements. VkFFT can be used with OpenCL and SYCL backends:

- For SYCL builds, VkFFT provides a portable backend which currently can be used on AMD and NVIDIA GPUs with AdaptiveCpp and Intel oneAPI DPC++; it generally outperforms rocFFT hence it is recommended as default on AMD. Note that VkFFT is not supported with PME decomposition (which requires HeFFTe) since HeFFTe does not have a VkFFT backend.
- For OpenCL builds, VkFFT provides an alternative to ClFFT. It is the default on macOS and when building with Visual Studio. On other platforms it is not extensively tested, but it likely outperforms ClFFT and can be enabled during cmake configuration.
- For *AMD-HIP* (page 23), VkFFT is the default FFT backend, as it supports both consumer and data center hardware.

To enable VkFFT support, use the following CMake option:

```
cmake -DGMX_GPU_FFT_LIBRARY=VKFFT
```

GROMACS bundles VkFFT with its source code, but an external VkFFT can also be used (e.g. to benefit from improvements in VkFFT releases more recent than the bundled version) in the following manner:

```
cmake -DGMX_GPU_FFT_LIBRARY=VKFFT \
      -DGMX_EXTERNAL_VKFFT=ON -DVKFFT_INCLUDE_DIR=<path to VkFFT␣
↪directory>
```

### 2.3.6 Other optional build components

- Run-time detection of hardware capabilities can be improved by linking with hwloc. By default this is turned off since it might not be supported everywhere, but if you have hwloc installed it should work by just setting `-DGMX_HWLOC=ON`
- Hardware-optimized BLAS and LAPACK libraries are useful for a few of the GROMACS utilities focused on normal modes and matrix manipulation, but they do not provide any benefits for normal simulations. Configuring these is discussed at *linear algebra libraries* (page 26).
- An external TNG library for trajectory-file handling can be used by setting `-DGMX_EXTERNAL_TNG=yes`, but TNG 1.7.10 is bundled in the GROMACS source already.
- The lmfit library for Levenberg-Marquardt curve fitting is used in GROMACS. Only lmfit 7.0 is supported. A reduced version of that library is bundled in the GROMACS distribution, and the default build uses it. That default may be explicitly enabled with `-DGMX_USE_LMFIT=internal`. To use an external lmfit library, set `-DGMX_USE_LMFIT=external`, and adjust `CMAKE_PREFIX_PATH` as needed. lmfit support can be disabled with `-DGMX_USE_LMFIT=none`.
- zlib is used by TNG for compressing some kinds of trajectory data.
- Building the GROMACS documentation is optional, and requires and other software. Refer to https://manual.gromacs.org/current/dev-manual/documentation-generation.html or the `docs/dev-manual/documentation-generation.rst` file in the sources.

- The GROMACS utility programs often write data files in formats suitable for the Grace plotting tool, but it is straightforward to use these files in other plotting programs, too.
- Set `-DGMX_PYTHON_PACKAGE=ON` when configuring GROMACS with CMake to enable additional CMake targets for the gmxapi Python package and sample_restraint package from the main GROMACS CMake build. This supports additional testing and documentation generation.

## 2.4 Doing a build of GROMACS

This section will cover a general build of GROMACS with *CMake* (page 12), but it is not an exhaustive discussion of how to use CMake. There are many resources available on the web, which we suggest you search for when you encounter problems not covered here. The material below applies specifically to builds on Unix-like systems, including Linux, and Mac OS X. For other platforms, see the specialist instructions below.

### 2.4.1 Configuring with CMake

CMake will run many tests on your system and do its best to work out how to build GROMACS for you. If your build machine is the same as your target machine, then you can be sure that the defaults and detection will be pretty good. However, if you want to control aspects of the build, or you are compiling on a cluster head node for back-end nodes with a different architecture, there are a few things you should consider specifying.

The best way to use CMake to configure GROMACS is to do an "out-of-source" build, by making another directory from which you will run CMake. This can be outside the source directory, or a subdirectory of it. It also means you can never corrupt your source code by trying to build it! So, the only required argument on the CMake command line is the name of the directory containing the `CMakeLists.txt` file of the code you want to build. For example, download the source tarball and use

```
tar xzf gromacs-2027.0-dev.tgz
cd gromacs-2027.0-dev
mkdir build-gromacs
cd build-gromacs
cmake ..
```

You will see `cmake` report a sequence of results of tests and detections done by the GROMACS build system. These are written to the `cmake` cache, kept in `CMakeCache.txt`. You can edit this file by hand, but this is not recommended because you could make a mistake. You should not attempt to move or copy this file to do another build, because file paths are hard-coded within it. If you mess things up, just delete this file and start again with `cmake`.

If there is a serious problem detected at this stage, then you will see a fatal error and some suggestions for how to overcome it. If you are not sure how to deal with that, please start by searching on the web (most computer problems already have known solutions!) and then consult the user discussion forum. There are also informational warnings that you might like to take on board or not. Piping the output of `cmake` through `less` or `tee` can be useful, too.

Once `cmake` returns, you can see all the settings that were chosen and information about them by using e.g. the curses interface

```
ccmake ..
```

You can actually use `ccmake` (available on most Unix platforms) directly in the first step, but then most of the status messages will merely blink in the lower part of the terminal rather than be written to standard output. Most platforms including Linux, Windows, and Mac OS X even have native graphical user interfaces for `cmake`, and it can create project files for almost any build environment you want (including Visual Studio or Xcode). Check out running CMake for general advice on what you are seeing and how to navigate and change things. The settings you might normally want to change are already presented. You may make changes, then re-configure (using `c`), so that it gets a chance to make changes that depend on yours and perform more checking. It may take several configuration passes to reach the desired configuration, in particular if you need to resolve errors.

When you have reached the desired configuration with `ccmake`, the build system can be generated by pressing `g`. This requires that the previous configuration pass did not reveal any additional settings (if it did, you need to configure once more with `c`). With `cmake`, the build system is generated after each pass that does not produce errors.

You cannot attempt to change compilers after the initial run of `cmake`. If you need to change, clean up, and start again.

### Where to install GROMACS

GROMACS is installed in the directory to which `CMAKE_INSTALL_PREFIX` points. It may not be the source directory or the build directory. You require write permissions to this directory. Thus, without super-user privileges, `CMAKE_INSTALL_PREFIX` will have to be within your home directory. Even if you do have super-user privileges, you should use them only for the installation phase, and never for configuring, building, or running GROMACS!

### Using CMake command-line options

Once you become comfortable with setting and changing options, you may know in advance how you will configure GROMACS. If so, you can speed things up by invoking `cmake` and passing the various options at once on the command line. This can be done by setting cache variable at the cmake invocation using `-DOPTION=VALUE`. Note that some environment variables are also taken into account, in particular variables like `CC` and `CXX`.

For example, the following command line

```
cmake .. -DGMX_GPU=CUDA -DGMX_MPI=ON \
         -DCMAKE_INSTALL_PREFIX=/home/marydoe/programs
```

can be used to build with CUDA GPUs, MPI and install in a custom location. You can even save that in a shell script to make it even easier next time. You can also do this kind of thing with `ccmake`, but you should avoid this, because the options set with `-D` will not be able to be changed interactively in that run of `ccmake`.

### SIMD support

GROMACS has extensive support for detecting and using the SIMD capabilities of many modern HPC CPU architectures. If you are building GROMACS on the same hardware you will run it on, then you don't need to read more about this, unless you are getting configuration warnings you do not understand. By default, the GROMACS build system will detect the SIMD instruction set supported by the CPU architecture (on which the configuring is done), and thus pick the best available SIMD parallelization supported by GROMACS. The build system will also check that the compiler and linker used also support the selected SIMD instruction set and issue a fatal error if they do not.

Valid values are listed below, and the applicable value with the largest number in the list is generally the one you should choose. In most cases, choosing an inappropriate higher number will lead to compiling a binary that will not run. However, on a number of processor architectures choosing the highest supported value can lead to performance loss, e.g. on Intel Skylake-X/SP and AMD Zen (first generation).

1. `None` For use only on an architecture either lacking SIMD, or to which GROMACS has not yet been ported and none of the options below are applicable.
2. `SSE2` This SIMD instruction set was introduced in Intel processors in 2001, and AMD in 2003. Essentially all x86 machines in existence have this, so it might be a good choice if you need to support dinosaur x86 computers too.
3. `SSE4.1` Present in all Intel core processors since 2007, but notably not in AMD Magny-Cours. Still, almost all recent processors support this, so this can also be considered a good baseline if you are content with slow simulations and prefer portability between reasonably modern processors.
4. `AVX_128_FMA` AMD Bulldozer, Piledriver (and later Family 15h) processors have this but it is NOT supported on any AMD processors since Zen1.
5. `AVX_256` Intel processors since Sandy Bridge (2011). While this code will work on the AMD Bulldozer and Piledriver processors, it is significantly less efficient than the `AVX_128_FMA` choice above - do not be fooled to assume that 256 is better than 128 in this case.
6. `AVX2_128` AMD Zen/Zen2 and Hygon Dhyana microarchitecture processors; it will enable AVX2 with 3-way fused multiply-add instructions. While these microarchitectures do support 256-bit AVX2 instructions, hence `AVX2_256` is also supported, 128-bit will generally be faster, in particular when the non-bonded tasks run on the CPU – hence the default `AVX2_128`. With GPU offload, however, `AVX2_256` can be faster on Zen processors.
7. `AVX2_256` Present on Intel Haswell (and later) processors (2013) and AMD Zen3 and later (2020); it will also enable 3-way fused multiply-add instructions.
8. `AVX_512` Skylake-X desktop and Skylake-SP Xeon processors (2017) and AMD Zen4 (2022); on Intel it will generally be fastest on the higher-end desktop and server processors with two 512-bit fused multiply-add units (e.g. Core i9 and Xeon Gold). However, certain desktop and server models (e.g. Xeon Bronze and Silver) come with only one AVX512 FMA unit and therefore on these processors `AVX2_256` is faster (compile- and runtime checks try to inform about such cases). On AMD it is beneficial to use starting with Zen4. Additionally, with GPU accelerated runs `AVX2_256` can also be faster on high-end Skylake CPUs with both 512-bit FMA units enabled.
9. `IBM_VSX` Power7, Power8, Power9 and later have this.
10. `ARM_NEON_ASIMD` 64-bit ARMv8 and later. For maximum performance on NVIDIA Grace (ARMv9), we strongly suggest at least GNU >= 13, LLVM >= 16.

11. ARM_SVE 64-bit ARMv8 and later with the Scalable Vector Extensions (SVE). The SVE vector length is fixed at CMake configure time. The default vector length is automatically detected, and this can be changed via the GMX_SIMD_ARM_SVE_LENGTH CMake variable. If compiling for a different target architecture than the compilation machine, GMX_SIMD_ARM_SVE_LENGTH should be set to the hardware vector length implemented by the target machine. There is no expected performance benefit from setting a smaller value than the implemented vector length, and setting a larger length can lead to unexpected crashes. Minimum required compiler versions are GNU >= 10, LLVM >=13, or ARM >= 21.1. For maximum performance we strongly suggest the latest gcc compilers, or at least LLVM 14 or ARM 22.0. Lower performance has been observed with LLVM 13 and Arm compiler 21.1.

The CMake configure system will check that the compiler you have chosen can target the architecture you have chosen. mdrun will check further at runtime, so if in doubt, choose the lowest number you think might work, and see what mdrun says. The configure system also works around many known issues in many versions of common HPC compilers.

A further GMX_SIMD=Reference option exists, which is a special SIMD-like implementation written in plain C that developers can use when developing support in GROMACS for new SIMD architectures. It is not designed for use in production simulations, but if you are using an architecture with SIMD support to which GROMACS has not yet been ported, you may wish to try this option instead of the default GMX_SIMD=None, as it can often out-perform this when the auto-vectorization in your compiler does a good job. And post on the GROMACS user discussion forum, because GROMACS can probably be ported to new SIMD architectures in a few days.

### CMake advanced options

The options that are displayed in the default view of ccmake are ones that we think a reasonable number of users might want to consider changing. There are a lot more options available, which you can see by toggling the advanced mode in ccmake on and off with t. Even there, most of the variables that you might want to change have a CMAKE_ or GMX_ prefix. There are also some options that will be visible or not according to whether their preconditions are satisfied.

### Helping CMake find the right libraries, headers, or programs

If libraries are installed in non-default locations, their location can be specified using the following variables:

- CMAKE_INCLUDE_PATH for header files
- CMAKE_LIBRARY_PATH for libraries
- CMAKE_PREFIX_PATH for header, libraries and binaries (e.g. /usr/local).

The respective include, lib, or bin is appended to the path. For each of these variables, a list of paths can be specified (on Unix, separated with ":"). These can be set as environment variables like:

```
CMAKE_PREFIX_PATH=/opt/fftw:/opt/cuda cmake ..
```

(assuming bash shell). Alternatively, these variables are also cmake options, so they can be set like -DCMAKE_PREFIX_PATH=/opt/fftw:/opt/cuda.

The CC and CXX environment variables are also useful for indicating to cmake which compilers to use. Similarly, CFLAGS/CXXFLAGS can be used to pass compiler options, but note that these will be

appended to those set by GROMACS for your build platform and build type. You can customize some of this with advanced CMake options, such as CMAKE_C_FLAGS and its relatives.

See also the page on CMake environment variables.

### CUDA GPU acceleration

If you have the CUDA Toolkit installed, you can use cmake with:

```
cmake .. -DGMX_GPU=CUDA -DCUDAToolkit_ROOT=/usr/local/cuda
```

(or whichever path has your installation). In some cases, you might need to specify manually which of your C++ compilers should be used, e.g. with the advanced option CMAKE_CUDA_HOST_COMPILER.

By default, code will be generated for the most common CUDA architectures. However, to reduce build time and binary size we do not generate code for every single possible architecture, which in rare cases (say, Tegra systems) can result in the default build not being able to use some GPUs. If this happens, or if you want to remove some architectures to reduce binary size and build time, you can alter the target CUDA architectures. This can be done with the CMAKE_CUDA_ARCHITECTURES CMake variable, which takes a semicolon delimited string with the two or three digit suffixes of CUDA architectures names, for instance "60;75;86". For details, see the "CUDA_ARCHITECTURES" section of the CMake documentation.

The GPU acceleration has been tested on AMD64/x86-64 platforms with Linux and Windows operating systems, but Linux is the best-tested and supported of these. Linux running on POWER 8/9 and ARM CPUs also works well.

Experimental support is available for compiling CUDA code, both for host and device, using clang. A CUDA toolkit is still required but it is used only for GPU device code generation and to link against the CUDA runtime library. The clang CUDA support simplifies compilation and provides benefits for development (e.g. allows the use code sanitizers in CUDA host-code). Additionally, using clang for both CPU and GPU compilation can be beneficial to avoid compatibility issues between the GNU toolchain and the CUDA toolkit. clang for CUDA can be triggered using the GMX_CLANG_CUDA=ON CMake option. Note that this is mainly a developer-oriented feature but its performance is generally close to that of code compiled with nvcc.

### OpenCL GPU acceleration

The primary targets of the GROMACS OpenCL support is accelerating simulations on AMD and Intel hardware. For AMD, we target both discrete GPUs and APUs (integrated CPU+GPU chips), and for Intel we target the integrated GPUs found on modern workstation and mobile hardware. The GROMACS OpenCL on NVIDIA GPUs works, but performance and other limitations make it less practical (for details see the user guide).

To build GROMACS with OpenCL support enabled, two components are required: the OpenCL headers and the wrapper library that acts as a client driver loader (so-called ICD loader). The additional, runtime-only dependency is the vendor-specific GPU driver for the device targeted. This also contains the OpenCL compiler. As the GPU compute kernels are compiled on-demand at run time, this vendor-specific compiler and driver is not needed for building GROMACS. The former, compile-time dependencies are standard components, hence stock versions can be obtained from most Linux distribution repositories (e.g. opencl-headers and ocl-icd-libopencl1 on Debian/Ubuntu). Only the compatibility with the required OpenCL version unknown needs to be ensured. Alternatively, the

headers and library can also be obtained from vendor SDKs, which must be installed in a path found in `CMAKE_PREFIX_PATH`.

To trigger an OpenCL build the following CMake flags must be set

```
cmake .. -DGMX_GPU=OpenCL
```

To build with support for Intel integrated GPUs, it is required to add `-DGMX_GPU_NB_CLUSTER_SIZE=4` to the cmake command line, so that the GPU kernels match the characteristics of the hardware. The Neo driver is recommended.

On Mac OS, an AMD GPU can be used only with OS version 10.10.4 and higher; earlier OS versions are known to run incorrectly.

By default, on Linux, any clFFT library on the system will be used with GROMACS, but if none is found then the code will fall back on a version bundled with GROMACS. To require GROMACS to link with an external library, use

```
cmake .. -DGMX_GPU=OpenCL -DclFFT_ROOT_DIR=/path/to/your/clFFT \
         -DGMX_EXTERNAL_CLFFT=TRUE
```

On Windows with MSVC and on macOS, VkFFT is used instead of clFFT, but this can provide performance benefits on other platforms as well.

### SYCL GPU acceleration

SYCL is a modern portable heterogeneous acceleration API, with multiple implementations targeting different hardware platforms (similar to OpenCL).

GROMACS can be used with different SYCL compilers/runtimes to target the following hardware:

- Intel GPUs using Intel oneAPI DPC++ (both OpenCL and LevelZero backends),
- AMD GPUs with AdaptiveCpp (previously known as hipSYCL),

There is also experimental support for:

- AMD GPUs with oneAPI with Codeplay AMD plugin,
- NVIDIA GPUs with either AdaptiveCpp or oneAPI with Codeplay NVIDIA plugin.

In table form:

| GPU vendor | AdaptiveCpp | Intel oneAPI DPC++ |
|---|---|---|
| Intel | not supported | supported |
| AMD | supported | experimental (requires Codeplay plugin) |
| NVIDIA | experimental | experimental (requires Codeplay plugin) |

Here, "experimental support" means that the combination has received limited testing and is expected to work (with possible limitations), but is not recommended for production use. Please refer to *a separate section in the installation guide* (page 3) to use them.

The SYCL support in GROMACS is intended to replace OpenCL as an acceleration mechanism for AMD and Intel hardware.

For NVIDIA GPUs, we strongly advise using CUDA. Apple M1/M2 GPUs are not supported with SYCL but can be used with OpenCL.

Codeplay ComputeCpp is not supported. Open-source Intel LLVM can be used in the same way as Intel oneAPI DPC++.

Note: SYCL support in GROMACS and the underlying compilers and runtimes are less mature than either OpenCL or CUDA. Please, pay extra attention to simulation correctness when you are using it.

### SYCL GPU acceleration for Intel GPUs

You should install the recent Intel oneAPI DPC++ compiler toolkit. For GROMACS 2025, version 2025.0 is recommended, and 2024.0 is the earliest supported. Using open-source Intel LLVM is possible, but not extensively tested. We also recommend installing the most recent Neo driver.

With the toolkit installed and added to the environment (usually by running `source /opt/intel/oneapi/setvars.sh` or using an appropriate **`module load`** on an HPC system), the following CMake flags must be set:

```
cmake .. -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx \
         -DGMX_GPU=SYCL -DGMX_SYCL=DPCPP
```

When compiling for Intel Data Center GPU Max (also knows as Ponte Vecchio / PVC), Intel Xe2 GPUs (Lunar Lake, Arc Battlemage) and newer, we recommend passing additional flags for compatibility and improved performance:

```
cmake .. -DCMAKE_C_COMPILER=icx -DCMAKE_CXX_COMPILER=icpx \
         -DGMX_GPU=SYCL -DGMX_SYCL=DPCPP \
         -DGMX_GPU_NB_NUM_CLUSTER_PER_BIN_X=1 -DGMX_GPU_NB_CLUSTER_
↪SIZE=8
```

Note that `GMX_GPU_NB_NUM_CLUSTER_PER_CELL` changed its name in GROMACS 2027.0 to `GMX_GPU_NB_NUM_CLUSTER_PER_BIN` without changing its function.

You might also consider using *double-batched FFT library* (page 13).

### SYCL GPU acceleration for AMD GPUs

Using AdaptiveCpp 24.02.0 and ROCm 5.7-6.2 is recommended. The earliest supported version is AdaptiveCpp 24.02.

We strongly recommend using the clang compiler bundled with ROCm for building both AdaptiveCpp and GROMACS. Mainline Clang releases can also work.

The following CMake command can be used **when configuring AdaptiveCpp** to ensure that the proper Clang is used (assuming `ROCM_PATH` is set correctly, e.g. to `/opt/rocm` in the case of default installation):

```
cmake .. -DCMAKE_C_COMPILER=${ROCM_PATH}/llvm/bin/clang \
         -DCMAKE_CXX_COMPILER=${ROCM_PATH}/llvm/bin/clang++ \
         -DLLVM_DIR=${ROCM_PATH}/llvm/lib/cmake/llvm/ \
         -DACPP_COMPILER_FEATURE_PROFILE=minimal
```

After compiling and installing AdaptiveCpp, the following settings can be used for building GROMACS itself (set ACPP_TARGETS to the target hardware):

```
cmake .. -DCMAKE_C_COMPILER=${ROCM_PATH}/llvm/bin/clang \
         -DCMAKE_CXX_COMPILER=${ROCM_PATH}/llvm/bin/clang++ \
         -DGMX_GPU=SYCL -DGMX_SYCL=ACPP -DACPP_TARGETS='hip:gfxXYZ'
```

Multiple target architectures can be specified, e.g., -DACPP_TARGETS='hip:gfx908,gfx90a'. Having both RDNA (gfx1xyz) and GCN/CDNA (gfx9xx) devices in the same build is possible but will incur a minor performance penalty compared to building for GCN/CDNA devices only. If you have multiple AMD GPUs of different generations in the same system (e.g., integrated APU and a discrete GPU) the ROCm runtime requires code to be available for each device at runtime, so you need to specify every device in ACPP_TARGETS when compiling to avoid ROCm crashes at initialization.

By default, VkFFT is used to perform FFT on GPU. You can switch to rocFFT by passing -DGMX_GPU_FFT_LIBRARY=rocFFT CMake flag. Please note that rocFFT is not officially supported and tends not to work on most consumer GPUs.

AMD GPUs can also be targeted via Intel oneAPI DPC++; please refer to *a separate section* (page 3) for the build instructions.

### SYCL GPU compilation options

The following flags can be passed to CMake in order to tune GROMACS:

**-DGMX_GPU_NB_CLUSTER_SIZE**
: changes the data layout of non-bonded kernels. When compiling with Intel oneAPI DPC++, the default value is 4, which is optimal for most Intel GPUs except Data Center MAX (Ponte Vecchio), for which 8 is better. When compiling with AdaptiveCpp, the default value is 8, which is the only supported value for AMD and NVIDIA devices.

**-DGMX_GPU_NB_NUM_CLUSTER_PER_BIN_X,**
**-DGMX_GPU_NB_NUM_CLUSTER_PER_BIN_Y, -DGMX_GPU_NB_NUM_CLUSTER_PER_BIN_Z**
: Sets the number of clusters along X, Y, or Z in a pair-search grid bin, default 2. When targeting Intel Ponte Vecchio GPUs, set -DGMX_GPU_NB_NUM_CLUSTER_PER_BIN_X=1 and leave the other values as the default.

**-DGMX_GPU_NB_DISABLE_CLUSTER_PAIR_SPLIT**
: Disables cluster pair splitting in the GPU non-bonded kernels. This is only supported in SYCL, and it is compatible with and improves performance on GPUs with 64-wide execution like AMD GCN and CDNA family. This option is automatically enabled in all builds that target GCN or CDNA GPUs (but not RDNA).

### AMD HIP GPU acceleration

HIP is the AMD interoperability layer for the ROCm toolkit used to target AMD devices.

In GROMACS 2026 there is full support for using HIP as the GPU backend on AMD devices.

### Build instructions

In order to use HIP as the device backend, you need to have the ROCm toolkit installed, including the rocPrim libraries. The minimum version required by GROMACS is ROCm 5.2, but we recommend a recent version to take advantage of library improvements.

You can then configure the build like this

```
cmake .. -DCMAKE_HIP_COMPILER=${ROCM_PATH}/bin/amdclang++ \
         -DCMAKE_PREFIX_PATH=${ROCM_PATH} \
         -DGMX_GPU=HIP
```

By default GROMACS will generate code for a range of different CDNA devices. In case you want to narrow the scope of the code generation, or want to target RDNA or GCN devices, you can specify the architectures using this flag

`-DGMX_HIP_TARGET_ARCH=gfxXYZ,gfxABCD`

When detecting a 64-wide execution architecture and no 32-wide versions, GROMACS will automatically configure with

`-DGMX_GPU_NB_DISABLE_CLUSTER_PAIR_SPLIT=ON`

to improve performance on those devices. In case any 32-wide architectures are present, the maximum execution width will be restricted to be 32-wide, even on devices that support 64-wide execution.

When GROMACS is built with explicit 64-wide execution (and conflicting support for 32-wide devices), any 32-wide devices detected will be not be used.

It is possible to select the GPU FFT backend using the

`-DGMX_GPU_FFT_LIBRARY=rocFFT/VkFFT`

variable. Please be aware that `rocFFT` only works on devices that are officially supported by ROCm. If an incompatible device is detected, GROMACS will not use the it when linked against `rocFFT`, even though the rest of the code is compiled to support it. In this case, select `VkFFT` as the GPU FFT library.

### H5MD trajectory output file format

H5MD is a specification for output data from molecular dynamics simulations built on the HDF5 format. In GROMACS 2026 this is added as an experimental feature.

### Build instructions

To enable support for H5MD output GROMACS you need to have an installed copy of the HDF5 library. The minimum supported version is 1.10.7. If the library is installed you can build with H5MD support by

```
cmake .. -DGMX_USE_HDF5=ON
```

Some distributed versions of the HDF5 library may be incorrectly configured, leading to CMake not detecting its path or version. In such cases, adding `-DHDF5_C_COMPILER_EXECUTABLE=''` (ie. passing an empty string) may solve the issue.

### Static linking

Please refer to *a dedicated section* (page 4).

### gmxapi C++ API

For dynamic linking builds and on non-Windows platforms, an extra library and headers are installed by setting `-DGMXAPI=ON` (default). Build targets `gmxapi-cppdocs` and `gmxapi-cppdocs-dev` produce documentation in `docs/api-user` and `docs/api-dev`, respectively. For more project information and use cases, refer to the tracked Issue 2585, associated GitHub gmxapi projects, or DOI 10.1093/bioinformatics/bty484.

gmxapi is not yet tested on Windows or with static linking, but these use cases are targeted for future versions.

### Portability of a GROMACS build

A GROMACS build will normally not be portable, not even across hardware with the same base instruction set, like x86. Non-portable hardware-specific optimizations are selected at configure-time, such as the SIMD instruction set used in the compute kernels. This selection will be done by the build system based on the capabilities of the build host machine or otherwise specified to `cmake` during configuration.

Often it is possible to ensure portability by choosing the least common denominator of SIMD support, e.g. SSE2 for x86. In rare cases of very old x86 machines, ensure that you use `cmake -DGMX_USE_RDTSCP=off` if any of the target CPU architectures does not support the `RDTSCP` instruction. However, we discourage attempts to use a single GROMACS installation when the execution environment is heterogeneous, such as a mix of AVX and earlier hardware, because this will lead to programs (especially mdrun) that run slowly on the new hardware. Building two full installations and locally managing how to call the correct one (e.g. using a module system) is the recommended approach. Alternatively, one can use different suffixes to install several versions of GROMACS in the same location. To achieve this, one can first build a full installation with the least-common-denominator SIMD instruction set, e.g. `-DGMX_SIMD=SSE2`, in order for simple commands like `gmx grompp` to work on all machines, then build specialized `gmx` binaries for each architecture present in the heterogeneous environment. By using custom binary and library suffixes (with CMake variables `-DGMX_BINARY_SUFFIX=xxx` and `-DGMX_LIBS_SUFFIX=xxx`), these can be installed to the same location.

Portability of binaries across GPUs is generally better, targeting multiple generations of GPUs from the same vendor is in most cases possible with a single GROMACS build. CUDA builds will by default be able to run on any NVIDIA GPU supported by the CUDA toolkit used since the GROMACS build system generates code for these at build-time. With SYCL multiple target architectures of the same GPU vendor can be selected when using AdaptiveCpp (i.e. only AMD or only NVIDIA). The SSCP/generic compilation mode of AdaptiveCpp is currently not supported. With OpenCL, due to just-in-time compilation of GPU code for the device in use, this is not a concern.

### Linear algebra libraries

As mentioned above, sometimes vendor BLAS and LAPACK libraries can provide performance enhancements for GROMACS when doing normal-mode analysis or covariance analysis. For simplicity, the text below will refer only to BLAS, but the same options are available for LAPACK. By default, CMake will search for BLAS, use it if it is found, and otherwise fall back on a version of BLAS internal to GROMACS. The `cmake` option `-DGMX_EXTERNAL_BLAS=on` will be set accordingly. The internal versions are fine for normal use. If you need to specify a non-standard path to search, use `-DCMAKE_PREFIX_PATH=/path/to/search`. If you need to specify a library with a non-standard name (e.g. ESSL on Power machines or ARMPL on ARM machines), then set `-DGMX_BLAS_USER=/path/to/reach/lib/libwhatever.a`.

If you are using Intel MKL for FFT, then the BLAS and LAPACK it provides are used automatically. This could be over-ridden with `GMX_BLAS_USER`, etc.

On Apple platforms where the Accelerate Framework is available, these will be automatically used for BLAS and LAPACK. This could be over-ridden with `GMX_BLAS_USER`, etc.

### Building with MiMiC QM/MM support

MiMiC QM/MM interface integration will require linking against MiMiC communication library, that establishes the communication channel between GROMACS and CPMD. The MiMiC Communication library can be downloaded here. Compile and install it. Check that the installation folder of the MiMiC library is added to CMAKE_PREFIX_PATH if it is installed in non-standard location. Building QM/MM-capable version requires double-precision version of GROMACS compiled with MPI support:

- `-DGMX_DOUBLE=ON -DGMX_MPI=ON -DGMX_MIMIC=ON`

### Building with CP2K QM/MM support

CP2K QM/MM interface integration will require linking against libcp2k library, that incorporates CP2K functionality into GROMACS.

1. Download, compile and install CP2K (version 8.1 or higher is required). CP2K latest distribution can be downloaded here. For CP2K specific instructions, please follow. You can also check instructions on the official CP2K web-page.
2. Make `libcp2k.a` library by executing the following command:

   ```
   make ARCH=<your arch file> VERSION=<your version like psmp>␣
   ↪libcp2k
   ```

   The library archive (*e.g.* `libcp2k.a`) should appear in the *<cp2k dir>*`/lib/`*<arch>*`/`*<version>*`/` directory.
3. Configure GROMACS with **`cmake`**, adding the following flags:

   Build should be static: `-DBUILD_SHARED_LIBS=OFF -DGMXAPI=OFF -DGMX_INSTALL_NBLIB_API=OFF`

   Double precision in general is better than single for QM/MM (however both options are viable): `-DGMX_DOUBLE=ON`

   FFT, BLAS and LAPACK libraries should be the same between CP2K and GROMACS. Use the following flags to do so:

- `-DGMX_FFT_LIBRARY=<your library like fftw3> -DFFTWF_LIBRARY=<path to library> -DFFTWF_INCLUDE_DIR=<path to directory with headers>`
- `-DGMX_BLAS_USER=<path to your BLAS>`
- `-DGMX_LAPACK_USER=<path to your LAPACK>`

4. Compilation of QM/MM interface is controled by the following flags:

**`-DGMX_CP2K=ON`**
Activates QM/MM interface compilation

**`-DCP2K_DIR="<path to cp2k>/lib/local/psmp`**
Directory with libcp2k.a library

**`-DCP2K_LINKER_FLAGS="<combination of LDFLAGS and LIBS>"` (optional for CP2K 9.1 or newer)**
Other libraries used by CP2K. Typically that should be combination of LDFLAGS and LIBS from the ARCH file used for CP2K compilation. Sometimes ARCH file could have several lines defining LDFLAGS and LIBS or even split one line into several using "\". In that case all of them should be concatenated into one long string without any extra slashes or quotes. For CP2K versions 9.1 or newer, CP2K_LINKER_FLAGS is not required but still might be used in very specific situations.

### Building with Colvars support

GROMACS bundles the Colvars library in its source distribution. The library and its interface with GROMACS are enabled by default when building GROMACS. This behavior may also be enabled explicitly with `-DGMX_USE_COLVARS=internal`. Alternatively, Colvars support may be disabled with `-DGMX_USE_COLVARS=none`. How to use Colvars in a GROMACS simulation is described in the User Guide, as well as in the Colvars documentation.

### Building with PLUMED support

GROMACS bundles the interface from version 2.10 of the PLUMED library in its source distribution. The interface is compatible with any PLUMED version. The interface is enabled by default with GROMACS unless GROMACS is built on Windows. You can explicitly enable the interface with `-DGMX_USE_PLUMED=ON` or deactivate it with `-DGMX_USE_PLUMED=OFF`. By default the option is set to `AUTO`, during the configuration CMake will try to activate PLUMED and in case it does not succeed it will output a "soft" warning. If the user forces the option `ON`, when PLUMED cannot be activated the configuration will fail with an error message. The User Guide contains the instructions on how to use PLUMED in a GROMACS simulation.

### Building with Neural Network potential support

To build GROMACS with support for Neural Network potentials, it has to be compiled with a suitable machine learning library. At the moment, only models trained in Pytorch are supported. To be able to load them in GROMACS, it has to be built with the Pytorch C++ API or Libtorch, which can be downloaded from the Pytorch website. The website offers versions including pre-CXX11 and CXX11 ABI versions on Linux. You must use the same ABI version as you use when building the rest of GROMACS, which does not support or test the pre-CXX11 ABI. So get (or build) the CXX11 ABI version of Libtorch. For the same reason, it is also not possible to use the Libtorch version that ships with a conda installation of Pytorch, because it is built with the pre-CXX11 ABI by default. The NNP interface is enabled by default when a Libtorch installation is found in the `CMAKE_PREFIX_PATH`, or `Torch_DIR` is set to a `TorchConfig.cmake` or `torch-config.cmake` usually found under `share/cmake/Torch/` in the libtorch installation directory. It may also be explicitly enabled with `-DGMX_NNPOT=TORCH` or disabled with `-DGMX_NNPOT=OFF`.

In addition, GROMACS provides support to specify custom Pytorch extensions at build time that may be used by the NNP model. The path to the extension library may be specified via the `TORCH_EXTENSION_PATH` variable. Note that CMake will search for a file called `libtorch_extension.so` in the specified directory.

### Changing the names of GROMACS binaries and libraries

It is sometimes convenient to have different versions of the same GROMACS programs installed. The most common use cases have been single and double precision, and with and without MPI. This mechanism can also be used to install side-by-side multiple versions of mdrun optimized for different CPU architectures, as mentioned previously.

By default, GROMACS will suffix programs and libraries for such builds with `_d` for double precision and/or `_mpi` for MPI (and nothing otherwise). This can be controlled manually with `GMX_DEFAULT_SUFFIX (ON/OFF)`, `GMX_BINARY_SUFFIX` (takes a string) and `GMX_LIBS_SUFFIX` (also takes a string). For instance, to set a custom suffix for programs and libraries, one might specify:

```
cmake .. -DGMX_DEFAULT_SUFFIX=OFF -DGMX_BINARY_SUFFIX=_mod \
         -DGMX_LIBS_SUFFIX=_mod
```

Thus the names of all programs and libraries will be appended with `_mod`.

### Changing installation tree structure

By default, a few different directories under `CMAKE_INSTALL_PREFIX` are used when when GROMACS is installed. Some of these can be changed, which is mainly useful for packaging GROMACS for various distributions. The directories are listed below, with additional notes about some of them. Unless otherwise noted, the directories can be renamed by editing the installation paths in the main CMakeLists.txt.

**bin/**
: The standard location for executables and some scripts. Some of the scripts hardcode the absolute installation prefix, which needs to be changed if the scripts are relocated. The name of the directory can be changed using `CMAKE_INSTALL_BINDIR` CMake variable.

**include/gromacs/**
: The standard location for installed headers.

**lib/**
: The standard location for libraries. The default depends on the system, and is determined by CMake. The name of the directory can be changed using `CMAKE_INSTALL_LIBDIR` CMake variable.

**lib/pkgconfig/**
: Information about the installed `libgromacs` library for `pkg-config` is installed here. The `lib/` part adapts to the installation location of the libraries. The installed files contain the installation prefix as absolute paths.

**share/cmake/**
: CMake package configuration files are installed here.

**share/gromacs/**
: Various data files and some documentation go here. The first part can be changed using `CMAKE_INSTALL_DATADIR`, and the second by using `GMX_INSTALL_DATASUBDIR` Using these CMake variables is the preferred way of changing the installation path for `share/gromacs/top/`, since the path to this directory is built into `libgromacs` as well as some scripts, both as a relative and as an absolute path (the latter as a fallback if everything else fails).

**share/man/**
: Installed man pages go here.

### 2.4.2 Compiling and linking

Once you have configured with `cmake`, you can build GROMACS with `make`. It is expected that this will always complete successfully, and give few or no warnings. The CMake-time tests GROMACS makes on the settings you choose are pretty extensive, but there are probably a few cases we have not thought of yet. Search the web first for solutions to problems, but if you need help, ask on the user discussion forum, being sure to provide as much information as possible about what you did, the system you are building on, and what went wrong. This may mean scrolling back a long way through the output of `make` to find the first error message!

If you have a multi-core or multi-CPU machine with `N` processors, then using

```
make -j N
```

will generally speed things up by quite a bit. Other build generator systems supported by `cmake` (e.g. `ninja`) also work well.

### 2.4.3 Installing GROMACS

Finally, `make install` will install GROMACS in the directory given in `CMAKE_INSTALL_PREFIX`. If this is a system directory, then you will need permission to write there, and you should use super-user privileges only for `make install` and not the whole procedure.

### 2.4.4 Getting access to GROMACS after installation

GROMACS installs the script `GMXRC` in the `bin` subdirectory of the installation directory (e.g. `/usr/local/gromacs/bin/GMXRC`), which you should source from your shell:

```
source /your/installation/prefix/here/bin/GMXRC
```

It will detect what kind of shell you are running and set up your environment for using GROMACS. You may wish to arrange for your login scripts to do this automatically; please search the web for instructions on how to do this for your shell.

Many of the GROMACS programs rely on data installed in the `share/gromacs` subdirectory of the installation directory. By default, the programs will use the environment variables set in the `GMXRC` script, and if this is not available they will try to guess the path based on their own location. This usually works well unless you change the names of directories inside the install tree. If you still need to do that, you might want to recompile with the new install location properly set, or edit the `GMXRC` script.

GROMACS also installs a CMake cache file to help with building client software (using the -C option when configuring the client software with CMake.) For an installation at `/your/installation/prefix/here`, hints files will be installed at `/your/installation/prefix/share/cmake/gromacs${GMX_LIBS_SUFFIX}/gromacs-hints${GMX_LIBS_SUFFIX}.cmake` where `${GMX_LIBS_SUFFIX}` is *as documented above* (page 28).

### 2.4.5 Testing GROMACS for correctness

Since 2011, the GROMACS development uses an automated system where every new code change is subject to regression testing on a number of platforms and software combinations. While this improves reliability quite a lot, not everything is tested, and since we increasingly rely on cutting edge compiler features there is non-negligible risk that the default compiler on your system could have bugs. We have tried our best to test and refuse to use known bad versions in `cmake`, but we strongly recommend that you run through the tests yourself. It only takes a few minutes, after which you can trust your build.

The simplest way to run the checks is to build GROMACS with `-DREGRESSIONTEST_DOWNLOAD`, and run `make check`. GROMACS will automatically download and run the tests for you. Alternatively, you can download and unpack the GROMACS regression test suite https://ftp.gromacs.org/regressiontests/regressiontests-2027.0.tar.gz tarball yourself and use the advanced `cmake` option `REGRESSIONTEST_PATH` to specify the path to the unpacked tarball, which will then be used for testing. If the above does not work, then please read on.

The regression tests are also available from the download section. Once you have downloaded them, unpack the tarball, source `GMXRC` as described above, and run `./gmxtest.pl all` inside the regression tests folder. You can find more options (e.g. adding `double` when using double precision, or `-only expanded` to run just the tests whose names match "expanded") if you just execute the script without options.

Hopefully, you will get a report that all tests have passed. If there are individual failed tests it could be a sign of a compiler bug, or that a tolerance is just a tiny bit too tight. Check the output files the script directs you to, and try a different or newer compiler if the errors appear to be real. If you cannot get it to pass the regression tests, you might try dropping a line to the GROMACS user discussion forum, but then you should include a detailed description of your hardware, and the output of `gmx mdrun -version` (which contains valuable diagnostic information in the header).

#### Non-standard suffix

If your `gmx` program has been suffixed in a non-standard way, then the `./gmxtest.pl -suffix` option will let you specify that suffix to the test machinery. You can use `./gmxtest.pl -double` to test the double-precision version. You can use `./gmxtest.pl -crosscompiling` to stop the test harness attempting to check that the programs can be run. You can use `./gmxtest.pl -mpirun srun` if your command to run an MPI program is called `srun`.

#### Running MPI-enabled tests

The `make check` target also runs integration-style tests that may run with MPI if `GMX_MPI=ON` was set. To make these work with various possible MPI libraries, you may need to set the CMake variables `MPIEXEC`, `MPIEXEC_NUMPROC_FLAG`, `MPIEXEC_PREFLAGS` and `MPIEXEC_POSTFLAGS` so that `mdrun-mpi-test_mpi` would run on multiple ranks via the shell command

```
${MPIEXEC} ${MPIEXEC_NUMPROC_FLAG} ${NUMPROC} ${MPIEXEC_PREFLAGS} \
      mdrun-mpi-test_mpi ${MPIEXEC_POSTFLAGS} -otherflags
```

A typical example for SLURM is

```
cmake .. -DGMX_MPI=on -DMPIEXEC=srun -DMPIEXEC_NUMPROC_FLAG=-n \
         -DMPIEXEC_PREFLAGS= -DMPIEXEC_POSTFLAGS=
```

### 2.4.6 Testing GROMACS for performance

We are still working on a set of benchmark systems for testing the performance of GROMACS. Until that is ready, we recommend that you try a few different parallelization options, and experiment with tools such as `gmx tune_pme`.

### 2.4.7 Having difficulty?

You are not alone - this can be a complex task! If you encounter a problem with installing GROMACS, then there are a number of locations where you can find assistance. It is recommended that you follow these steps to find the solution:

1. Read the installation instructions again, taking note that you have followed each and every step correctly.
2. Search the GROMACS webpage and user discussion forum for information on the error. Adding `site:https://gromacs.bioexcel.eu/c/gromacs-user-forum/5` to a Google search may help filter better results. It is also a good idea to check the gmx-users mailing list archive at `https://mailman-1.sys.kth.se/pipermail/gromacs.org_-gmx-users`
3. Search the internet using a search engine such as Google.
4. Ask for assistance on the GROMACS user discussion forum. Be sure to give a full description of what you have done and why you think it did not work. Give details about the system on which you are installing. Copy and paste your command line and as much of the output as you think might be relevant - certainly from the first indication of a problem. In particular, please try to include at least the header from the mdrun logfile, and preferably the entire file. People who

might volunteer to help you do not have time to ask you interactive detailed follow-up questions, so you will get an answer faster if you provide as much information as you think could possibly help. High quality bug reports tend to receive rapid high quality answers.

## 2.5 Special instructions for some platforms

Some less common configurations are described in a *separate manual section* (page 3).

### 2.5.1 Building on Windows

Building on Windows using native compilers is rather similar to building on Unix, so please start by reading the above. Then, download and unpack the GROMACS source archive. Make a folder in which to do the out-of-source build of GROMACS. For example, make it within the folder unpacked from the source archive, and call it `build-gromacs`.

For CMake, you can either use the graphical user interface provided on Windows, or you can use a command line shell with instructions similar to the UNIX ones above. If you open a shell from within your IDE (e.g. Microsoft Visual Studio), it will configure the environment for you, but you might need to tweak this in order to get either a 32-bit or 64-bit build environment. The latter provides the fastest executable. If you use a normal Windows command shell, then you will need to either set up the environment to find your compilers and libraries yourself, or run the `vcvarsall.bat` batch script provided by MSVC (just like sourcing a bash script under Unix).

With the graphical user interface, you will be asked about what compilers to use at the initial configuration stage, and if you use the command line they can be set in a similar way as under UNIX.

Unfortunately, `-DGMX_BUILD_OWN_FFTW=ON` (see *Using FFTW* (page 12)) does not work on Windows, because there is no supported way to build FFTW on Windows. You can either build FFTW some other way (e.g. MinGW), or use the built-in fftpack (which may be slow), or *using MKL* (page 13).

For the build, you can either load the generated solutions file into e.g. Visual Studio, or use the command line with `cmake --build` so the right tools get used.

### 2.5.2 Building on Cray

GROMACS builds mostly out of the box on modern Cray machines, but you may need to specify the use of static binaries with `-DGMX_BUILD_SHARED_EXE=off`, and you may need to set the F77 environmental variable to `ftn` when compiling FFTW. The ARM ThunderX2 Cray XC50 machines differ only in that the recommended compiler is the ARM HPC Compiler (`armclang`).

### 2.5.3 NVIDIA Grace

For best performance on Grace, use GCC >= 13.1 or LLVM >= 17, and set the `-DCMAKE_CXX_FLAGS=-mcpu=neoverse-v2 -DCMAKE_C_FLAGS=-mcpu=neoverse-v2` flags when configuring GROMACS.

With short-range non-bonded interactions calculations on the CPU (`-nb cpu` or when building without GPU support), performance can be improved by also setting the `-DGMX_SIMD=ARM_NEON_ASIMD` CMake option.

At minimum any compiler being used for Grace should implement neoverse-v2, such as GNU >= 12.3 and LLVM >= 16. There is a significant improvement in Arm performance between gcc-13 and gcc-12 so GNU >= 13.1 is strongly recommended. The `-mcpu=neoverse-v2` flag ensures that the compiler is not defaulting to the older Armv8-A target.

On both GNU and LLVM, the GROMACS CPU version of the short-range non bonded interactions implemented with `NEON SIMD` instructions significantly outperforms the SVE version. This can be selected by setting `GMX_SIMD=ARM_NEON_ASIMD` at compilation. There can be a small performance benefit to using SVE for CPU work outside this kernel, therefore when the short-range non bonded interactions run on the GPU it is recommended to stay with `GMX_SIMD=ARM_SVE` which is the default option when available.

## 2.6 Tested platforms

While it is our best belief that GROMACS will build and run pretty much everywhere, it is important that we tell you where we really know it works because we have tested it. Every commit in our git source code repository is currently tested with a range of configuration options on x86 with gcc versions 11-14, clang versions including 14, 18, and 19, CUDA versions 12.1, 12.5.1, and 12.6, nvcxx version 24.7 HIP version 5.7.1 and 6.2.2 AdaptiveCPP 24.02 and 24.10 with ROCm 5.7.1 and 6.2 (respectively), and oneAPI version 2024.0 and 2024.2 (including CUDA 12.0.1 and ROCm 6.1.3 backends).

For this testing, we use Ubuntu 22.04 and 24.04 operating systems. Other compiler, library, and OS versions are tested less frequently. For details, you can have a look at the continuous integration server used by the GitLab project, which uses GitLab runner on a local k8s x86 cluster with NVIDIA, AMD, and Intel GPU support.

We test irregularly on ARM v8, Fujitsu A64FX, Cray, Power9, and other environments, and with other compilers and compiler versions, too.

## 2.7 Support

Please refer to the manual for documentation, downloads, and release notes for any GROMACS release.

Visit the user forums for discussions and advice.

Report bugs at https://gitlab.com/gromacs/gromacs/-/issues

CHAPTER

THREE

# USER GUIDE

This guide provides

- material introducing GROMACS
- practical advice for making effective use of GROMACS.

For getting, building and installing GROMACS, see the *Installation guide* (page 3). For background on algorithms and implementations, see the *reference manual part* (page 418) of the documentation. If you have questions not answered by these resources, please visit the GROMACS user discussion forum and search for a potential answer or ask a question from the community.

This is not a release build of GROMACS, so please reference one of the GROMACS papers and the base release of the manual.

This is not a release build of GROMACS. Please reference one of the GROMACS papers, as well as the base release that this version is built from. Also, please state what modifcations have been performed or where the version was sourced from.

## 3.1 Known issues affecting users of GROMACS

Here is a non-exhaustive list of issues that are we are aware of that are affecting regular users of GROMACS.

### 3.1.1 "Cannot find a working standard library" error with ROCm Clang

Some Clang installations don't contain a compatible C++ standard library. In such cases, you might have to install `g++` and help CMake find it by setting `-DGMX_GPLUSGPLUS_PATH=/path/to/bin/g++`.

On Ubuntu 22.04, installing GCC 12 standard library (with `sudo apt install libstdc++-12-dev`) usually works well even without setting `-DGMX_GPLUSGPLUS_PATH`.

Issue 4679

### 3.1.2 Expanded ensemble does not checkpoint correctly

In the legacy simulator, because of shortcomings in the implementation, successful expanded-ensemble MC steps that occurred on checkpoint steps were not recorded in the checkpoint. If that checkpoint was used for a restart, then it would not necessarily behave correctly and reproducibly afterwards. So checkpointing of expanded-ensemble simulations is disabled for the legacy simulator.

Checkpointing of expanded ensemble in the modular simulator works correctly.

To work around the issue, either avoid `-update gpu` (so that it uses the modular simulator path which does not have the bug), or use an older version of GROMACS (which does do the buggy checkpointing), or refrain from restarting from checkpoints in the affected case.

Issue 4629

### 3.1.3 Compiling with GCC 12-14 on POWER9 architectures

There are multiple failing unit tests after compilation with some versions of GCC 12-14 on POWER9 architectures. The problem is fixed in GCC 12.5, GCC 13.4, and GCC 14.2, respectively. GCC 11 and earlier, as well as GCC 15, are *not* affected.

Issue 4823

### 3.1.4 NbnxmTest crash with oneAPI 2024.1

When building with oneAPI 2024.1, the `NbnxmTest` test can segfault in some cases. Using oneAPI 2024.2 or newer should resolve the issue.

Issue 5247

### 3.1.5 Severe performance regression with SVE and LLVM 20

There is a major performance regression on AArch64 CPUs when using SVE SIMD and building with LLVM 20.

The problem can be worked around by using LLVM 19 or setting `-DGMX_SIMD=ARM_NEON_ASIMD`.

The issue is expected to be fixed in LLVM 21.

Issue 5390

### 3.1.6 CMake `OpenMP_CUDA` detection issue on Windows

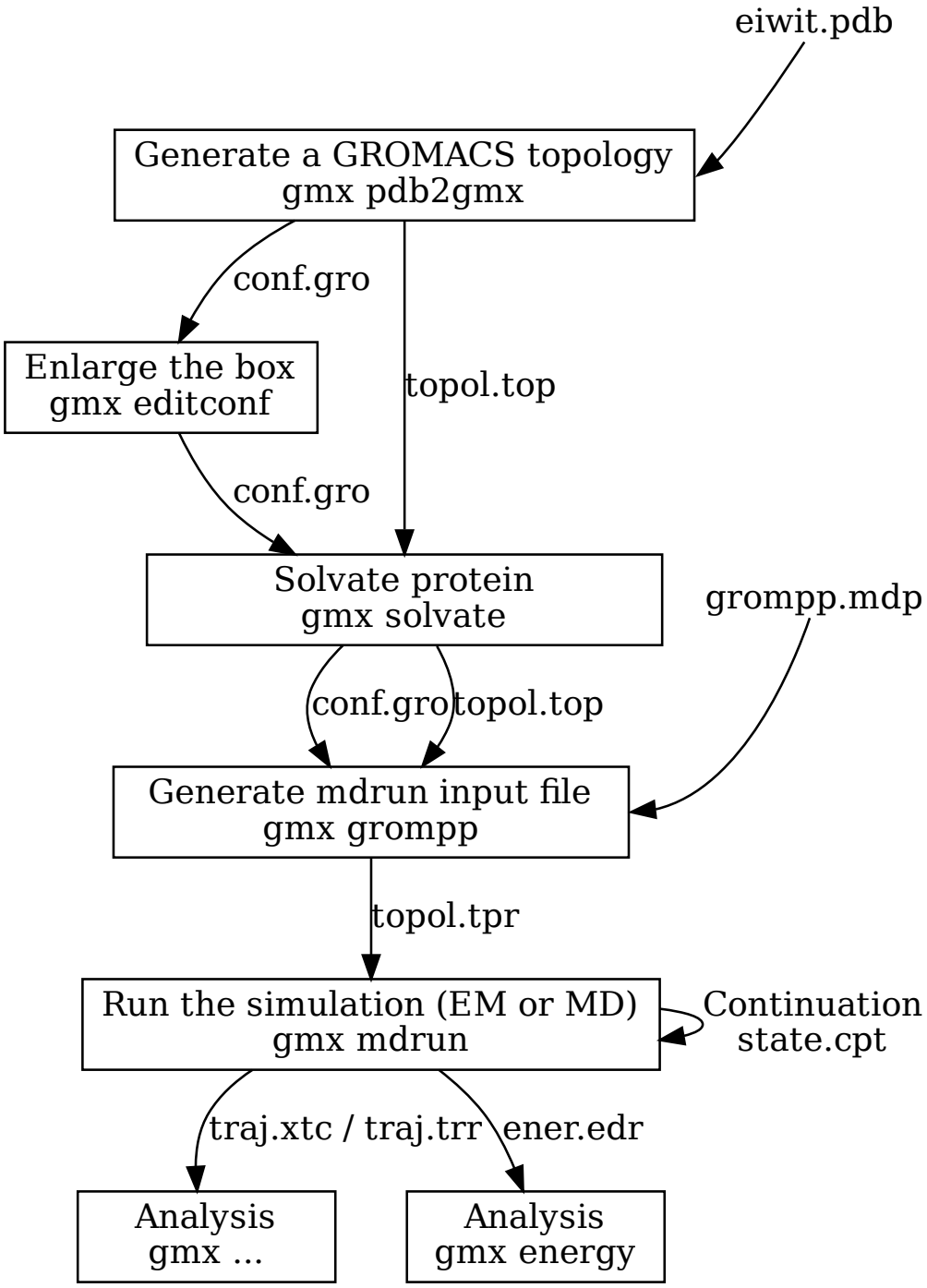
CMake prior to 4.3 fails to detect `OpenMP_CUDA` flags when building CUDA version on Windows with Visual Studio, causing build failures.

Workarounds: - use `-G Ninja` generator, or - set `-DOpenMP_CUDA_FLAGS=-Xcompiler=-openmp -DOpenMP_CUDA_LIB_NAMES=""` flags manually.

## 3.2 Getting started

### 3.2.1 Flow Chart

This is a flow chart of a typical GROMACS MD run of a protein in a box of water. Several steps of energy minimization may be necessary, these consist of cycles: *gmx grompp* (page 230) -> *gmx mdrun* (page 258).




In this chapter we assume the reader is familiar with Molecular Dynamics and familiar with Unix, including the use of a text editor such as `jot`, `emacs` or `vi`. We furthermore assume the GROMACS software is installed properly on your system. When you see a line like

```
ls -l
```

you are supposed to type the contents of that line on your computer terminal.

### 3.2.2 Setting up your environment

In order to check whether you have access to GROMACS, please start by entering the command:

```
gmx -version
```

This command should print out information about the version of GROMACS installed. If this, in contrast, returns the phrase

```
gmx: command not found.
```

then you have to find where your version of GROMACS is installed. In the default case, the binaries are located in `/usr/local/gromacs/bin`, however, you can ask your local system administrator for more information, and then follow the advice for *Getting access to GROMACS after installation* (page 30).

### 3.2.3 Flowchart of typical simulation

A typical simulation workflow with GROMACS is *illustrated here* (page 36).

### 3.2.4 Important files

Here is an overview of the most important GROMACS file types that you will encounter.

#### Molecular Topology file (`.top`)

The molecular topology file is generated by the program *gmx pdb2gmx* (page 281). *gmx pdb2gmx* (page 281) translates a *pdb* (page 581) structure file of any peptide or protein to a molecular topology file. This topology file contains a complete description of all the interactions in your peptide or protein.

#### Topology #include file mechanism

When constructing a system topology in a *top* (page 584) file for presentation to grompp, GROMACS uses a built-in version of the so-called C preprocessor, cpp (in GROMACS 3, it really was cpp). cpp interprets lines like:

```
#include "ions.itp"
```

by looking for the indicated file in the current directory, the GROMACS share/top directory as indicated by the GMXLIB environment variable, and any directory indicated by a `-I` flag in the value of the include *run parameter* (page 51) in the *mdp* (page 579) file. It either finds this file or reports a warning. (Note that when you supply a directory name, you should use Unix-style forward slashes '/', not Windows-style backslashes '' for separators.) When found, it then uses the contents exactly as if you had cut and pasted the included file into the main file yourself. Note that you should not go and do this copy-and-paste yourself, since the main purposes of the include file mechanism are to re-use previous work, make future changes easier, and prevent typos.

Further, `cpp` interprets code such as:

```
#ifdef POSRES_WATER
; Position restraint for each water oxygen
[ position_restraints ]
;  i funct       fcx        fcy        fcz
    1    1       1000       1000       1000
#endif
```

by testing whether the preprocessor variable `POSRES_WATER` was defined somewhere (i.e. "if defined"). This could be done with `#define POSRES_WATER` earlier in the *top* (page 584) file (or its `#include` files), with a `-D` flag in the `include` run parameter as above, or on the command line to `cpp`. The function of the `-D` flag is borrowed from the similar usage in `cpp`. The string that follows `-D` must match exactly; using `-DPOSRES` will not trigger `#ifdef POSRE` or `#ifdef DPOSRES`. This mechanism allows you to change your *mdp* (page 579) file to choose whether or not you want position restraints on your solvent, rather than your *top* (page 584) file. Note that preprocessor variables are not the same as shell environment variables.

### Molecular Structure file (`.gro`, `.pdb`)

When *gmx pdb2gmx* (page 281) is executed to generate a molecular topology, it also translates the structure file (*pdb* (page 581) file) to a GROMOS structure file (*gro* (page 577) file). The main difference between a *pdb* (page 581) file and a gromos file is their format and that a *gro* (page 577) file can also hold velocities. However, if you do not need the velocities, you can also use a *pdb* (page 581) file in all programs. To generate a box of solvent molecules around the peptide, the program *gmx solvate* (page 319) is used. First the program *gmx editconf* (page 208) should be used to define a box of appropriate size around the molecule. *gmx solvate* (page 319) solvates a solute molecule (the peptide) into any solvent (in this case, water). The output of *gmx solvate* (page 319) is a gromos structure file of the peptide solvated in water. *gmx solvate* (page 319) also changes the molecular topology file (generated by *gmx pdb2gmx* (page 281)) to add solvent to the topology.

### Molecular Dynamics parameter file (`.mdp`)

The Molecular Dynamics Parameter (*mdp* (page 579)) file contains all information about the Molecular Dynamics simulation itself e.g. time-step, number of steps, temperature, pressure etc. The easiest way of handling such a file is by adapting a *sample mdp file* (page 579) file.

### Index file (`.ndx`)

Sometimes you may need an index file to specify actions on groups of atoms (e.g. temperature coupling, accelerations, freezing). Usually the default index groups will be sufficient, so for this demo we will not consider the use of index files.

#### Run input file (`.tpr`)

The next step is to combine the molecular structure (*gro* (page 577) file), topology (*top* (page 584) file) MD-parameters (*mdp* (page 579) file) and (optionally) the index file (*ndx* (page 580)) to generate a run input file (*tpr* (page 586) extension). This file contains all information needed to start a simulation with GROMACS. The *gmx grompp* (page 230) program processes all input files and generates the run input *tpr* (page 586) file.

#### Trajectory file (`.trr`, `.tng`, or `.xtc`)

Once the run input file is available, we can start the simulation. The program which starts the simulation is called *gmx mdrun* (page 258). The only input file of *gmx mdrun* (page 258) that you usually need in order to start a run is the run input file (*tpr* (page 586) file). The typical output files of *gmx mdrun* (page 258) are the trajectory file (*trr* (page 586) file), a logfile (*log* (page 578) file), and perhaps a checkpoint file (*cpt* (page 575) file).

### 3.2.5 Tutorial material

There are several tutorials available that cover aspects of using GROMACS. There are also third-party-tutorials. Further information can also be found in the *How to* (page 407) section.

### 3.2.6 Background reading


- Berendsen, H.J.C., Postma, J.P.M., van Gunsteren, W.F., Hermans, J. (1981) Intermolecular Forces, chapter Interaction models for water in relation to protein hydration, pp 331-342. Dordrecht: D. Reidel Publishing Company Dordrecht
- Kabsch, W., Sander, C. (1983). Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. Biopolymers **22**, 2577–2637.
- Mierke, D.F., Kessler, H. (1991). Molecular dynamics with dimethyl sulfoxide as a solvent. Conformation of a cyclic hexapeptide. J. Am. Chem. Soc. **113**, 9446.
- Stryer, L. (1988). Biochemistry vol. 1, p. 211. New York: Freeman, 3 edition.


## 3.3 System preparation

There are many ways to prepare a simulation system to run with GROMACS. These often vary with the kind of scientific question being considered, or the model physics involved. A protein-ligand atomistic free-energy simulation might need a multi-state topology, while a coarse-grained simulation might need to manage defaults that suit systems with higher density.

### 3.3.1 Steps to consider

The following general guidance should help with planning successful simulations. Some stages are optional for some kinds of simulations.

1. Clearly identify the property or phenomena of interest to be studied by performing the simulation. Do not continue further until you are clear on this! Do not run your simulation and then seek to work out how to use it to test your hypothesis, because it may be unsuitable, or the required information was not saved.
2. Select the appropriate tools to be able to perform the simulation and observe the property or phenomena of interest. It is important to read and familiarize yourself with publications by other researchers on similar systems. Choices of tools include:
   - software with which to perform the simulation (consideration of force field may influence this decision)
   - the force field, which describes how the particles within the system interact with each other. Select one that is appropriate for the system being studied and the property or phenomena of interest. This is a very important and non-trivial step!
   - how you will analyze your simulation data to make your observations.
3. Obtain or generate the initial coordinate file for each molecule to be placed within the system. Many different software packages are able to build molecular structures and assemble them into suitable configurations.
4. Generate the raw starting structure for the system by placing the molecules within the coordinate file as appropriate. Molecules may be specifically placed or arranged randomly. Several non-GROMACS tools are useful here; within GROMACS *gmx solvate* (page 319), *gmx insert-molecules* (page 248) and *gmx genconf* (page 226) solve frequent problems.
5. Obtain or generate the topology file for the system, using (for example) *gmx pdb2gmx* (page 281), *gmx x2top* (page 355), SwissParam (for CHARMM forcefield), CHARMM-GUI , Automated Topology Builder (for GROMOS96 53A6) or your favourite text editor in concert with *Chapter 5* (page 481) of the GROMACS Reference Manual. For the AMBER force fields, antechamber or acpype might be appropriate.
6. Describe a simulation box (e.g. using *gmx editconf* (page 208)) whose size is appropriate for the eventual density you would like, fill it with solvent (e.g. using *gmx solvate* (page 319)), and add any counter-ions needed to neutralize the system (e.g. using *gmx grompp* (page 230) and *gmx insert-molecules* (page 248)). In these steps you may need to edit your topology file to stay current with your coordinate file.
7. Run an energy minimization on the system (using *gmx grompp* (page 230) and *gmx mdrun* (page 258)). This is required to sort out any bad starting structures caused during generation of the system, which may cause the production simulation to crash. It may be necessary also to minimize your solute structure in vacuo before introducing solvent molecules (or your lipid bilayer or whatever else). You should consider using flexible water models and not using bond constraints or frozen groups. The use of position restraints and/or distance restraints should be evaluated carefully.
8. Select the appropriate simulation parameters for the equilibration simulation (defined in *mdp* (page 579) file). You need to choose simulation parameters that are consistent with how force field was derived. You may need to simulate at NVT with position restraints on your solvent and/or solute to get the temperature almost right, then relax to NPT to fix the density (with the

recommendation to use the c-rescale barostat), then move further (if needed) to reach your production simulation ensemble (e.g. NVT, NVE). If you have problems here with the system *blowing up* (page 390), consider using the suggestions on that page, e.g. position restraints on solutes, or not using bond constraints, or using smaller integration timesteps, or several gentler heating stage(s).

9. Run the equilibration simulation for sufficient time so that the system relaxes sufficiently in the target ensemble to allow the production run to be commenced (using *gmx grompp* (page 230) and *gmx mdrun* (page 258), then *gmx energy* (page 214) and *Visualization Software* (page 415)).

10. Select the appropriate simulation parameters for the production simulation (defined in *mdp* (page 579) file). In particular, be careful not to re-generate the velocities. You still need to be consistent with how the force field was derived and how to measure the property or phenomena of interest.

### 3.3.2 Tips and tricks

#### Database files

The `share/top` directory of a GROMACS installation contains numerous plain-text helper files with the `.dat` file extension. Some of the command-line tools (see *Command-line reference* (page 142)) refer to these, and each tool documents which files it uses, and how they are used.

If you need to modify these files (e.g. to introduce new atom types with VDW radii into `vdwradii.dat`), you can copy the file from your installation directory into your working directory, and the GROMACS tools will automatically load the copy from your working directory rather than the standard one. To suppress all the standard definitions, use an empty file in the working directory.

## 3.4 Managing long simulations

Molecular simulations often extend beyond the lifetime of a single UNIX command-line process. It is useful to be able to stop and restart the simulation in a way that is equivalent to a single run. When *gmx mdrun* (page 258) is halted, it writes a checkpoint file that can restart the simulation exactly as if there was no interruption. To do this, the checkpoint retains a full-precision version of the positions and velocities, along with state information necessary to restart algorithms e.g. that implement coupling to external thermal reservoirs. A restart can be attempted using e.g. a *gro* (page 577) file with velocities, but since the *gro* (page 577) file has significantly less precision, and none of the coupling algorithms will have their state carried over, such a restart is less continuous than a normal MD step.

Such a checkpoint file is also written periodically by *gmx mdrun* (page 258) during the run. The interval is given by the `-cpt` flag to *gmx mdrun* (page 258). When *gmx mdrun* (page 258) attempts to write each successive checkpoint file, it first renames the old file with the suffix `_prev`, so that even if something goes wrong while writing the new checkpoint file, only recent progress can be lost.

*gmx mdrun* (page 258) can be halted in several ways:

- the number of simulation *nsteps* (page 53) can expire
- the user issues a termination signal (e.g. with Ctrl-C on the terminal)
- the job scheduler issues a termination signal when time expires
- when *gmx mdrun* (page 258) detects that the length specified with `-maxh` has elapsed (this option is useful to help cooperate with a job scheduler, but can be problematic if jobs can be suspended)

- some kind of catastrophic failure, such as loss of power, or a disk filling up, or a network failing

To use the checkpoint file for a restart, use a command line such as

```
gmx mdrun -cpi state
```

which directs mdrun to use the checkpoint file (which is named `state.cpt` by default). You can choose to give the output checkpoint file a different name with the `-cpo` flag, but if so then you must provide that name as input to `-cpi` when you later use that file. You can query the contents of checkpoint files with *gmx check* (page 165) and *gmx dump* (page 205).

### 3.4.1 Appending to output files

By default, *gmx mdrun* (page 258) will append to the old output files. If the previous part ended in a regular way, then the performance data at the end of the log file will will be removed, some new information about the run context written, and the simulation will proceed. Otherwise, mdrun will truncate all the output files back to the time of the last written checkpoint file, and continue from there, as if the simulation stopped at that checkpoint in a regular way.

You can choose not to append the output files by using the `-noappend` flag, which forces mdrun to write each output to a separate file, whose name includes a ".partXXXX" string to describe which simulation part is contained in this file. This numbering starts from zero and increases monotonically as simulations are restarted, but does not reflect the number of simulation steps in each part. The *simulation-part* (page 53) option can be used to set this number manually in *gmx grompp* (page 230), which can be useful if data has been lost, e.g. through filesystem failure or user error.

Appending will not work if any output files have been modified or removed after mdrun wrote them, because the checkpoint file maintains a checksum of each file that it will verify before it writes to them again. In such cases, you must either restore the file, name them as the checkpoint file expects, or continue with `-noappend`. If your original run used `-deffnm`, and you want appending, then your continuations must also use `-deffnm`.

### 3.4.2 Backing up your files

You should arrange to back up your simulation files frequently. Network file systems on clusters can be configured in more or less conservative ways, and this can lead *gmx mdrun* (page 258) to be told that a checkpoint file has been written to disk when actually it is still in memory somewhere and vulnerable to a power failure or disk that fills or fails in the meantime. The UNIX tool rsync can be a useful way to periodically copy your simulation output to a remote storage location, which works safely even while the simulation is underway. Keeping a copy of the final checkpoint file from each part of a job submitted to a cluster can be useful if a file system is unreliable.

### 3.4.3 Extending a .tpr file

If the simulation described by *tpr* (page 586) file has completed and should be extended, use the *gmx convert-tpr* (page 178) tool to extend the run, e.g.

```
gmx convert-tpr -s previous.tpr -extend timetoextendby -o next.tpr
gmx mdrun -s next.tpr -cpi state.cpt
```

The time can also be extended using the `-until` and `-nsteps` options. Note that the original *mdp* (page 579) file may have generated velocities, but that is a one-time operation within *gmx grompp* (page 230) that is never performed again by any other tool.

### 3.4.4 Changing mdp options for a restart

If you wish to make changes to your simulations settings other than length, then you should do so in the *mdp* (page 579) file or topology, and then call

```
gmx grompp -f possibly-changed.mdp -p possibly-changed.top -c␣
↪original.gro -t state.cpt -o new.tpr
gmx mdrun -s new.tpr -cpi state.cpt
```

to instruct *gmx grompp* (page 230) to copy the full-precision coordinates and velocities in the checkpoint file into the new *tpr* (page 586) file. You should consider your choices for *tinit* (page 53), *init-step* (page 53), *nsteps* (page 53) and *simulation-part* (page 53). You should generally not regenerate velocities with *gen-vel* (page 69), and generally select *continuation* (page 70) so that constraints are not re-applied before the first integration step.

### 3.4.5 Restarts without checkpoint files

It used to be possible to continue simulations without the checkpoint files. As this approach could be unreliable or lead to unphysical results, only restarts from checkpoints are permitted now.

### 3.4.6 Are continuations exact?

If you had a computer with unlimited precision, or if you integrated the time-discretized equations of motion by hand, exact continuation would lead to identical results. But since practical computers have limited precision and MD is chaotic, trajectories will diverge very rapidly even if one bit is different. Such trajectories will all be equally valid, but eventually very different. Continuation using a checkpoint file, using the same code compiled with the same compiler and running on the same computer architecture using the same number of processors without GPUs (see next section) would lead to binary identical results. However, by default the actual work load will be balanced across the hardware according to the observed execution times. Such trajectories are in principle not reproducible, and in particular a run that took place in more than one part will not be identical with an equivalent run in one part - but neither of them is better in any sense.

### 3.4.7 Reproducibility

The following factors affect the reproducibility of a simulation, and thus its output:

- Precision (mixed / double) with double giving "better" reproducibility.
- Number of cores, due to different order in which forces are accumulated. For instance (a+b)+c is not necessarily binary identical to a+(b+c) in floating-point arithmetic.
- Type of processors. Even within the same processor family there can be slight differences.
- Optimization level when compiling.
- Optimizations at run time: e.g. the FFTW library that is typically used for fast Fourier transforms determines at startup which version of their algorithms is fastest, and uses that for the remainder of the calculations. Since the speed estimate is not deterministic, the results may vary from run to run.
- Random numbers used for instance as a seed for generating velocities (in GROMACS at the preprocessing stage).
- Uninitialized variables in the code (but there should not be any)
- Dynamic linking to different versions of shared libraries (e.g. for FFTs)
- Dynamic load balancing, since particles are redistributed to processors based on elapsed wallclock time, which will lead to (a+b)+c != a+(b+c) issues as above
- Number of PME-only ranks (for parallel PME simulations)
- MPI reductions typically do not guarantee the order of the operations, and so the absence of associativity for floating-point arithmetic means the result of a reduction depends on the order actually chosen
- On GPUs, the reduction of e.g. non-bonded forces has a non-deterministic summation order, so any fast implementation is non-reproducible by design.

The important question is whether it is a problem if simulations are not completely reproducible. The answer is yes and no. Reproducibility is a cornerstone of science in general, and hence it is important. The Central Limit Theorem tells us that in the case of infinitely long simulations, all observables converge to their equilibrium values. Molecular simulations in GROMACS adhere to this theorem, and hence, for instance, the energy of your system will converge to a finite value, the diffusion constant of your water molecules will converge to a finite value, and so on. That means all the important observables, which are the values you would like to get out of your simulation, are reproducible. Each individual trajectory is not reproducible, however.

However, there are a few cases where it would be useful if trajectories were reproducible, too. These include developers doing debugging, and searching for a rare event in a trajectory when, if it occurs, you want to have manually saved your checkpoint file so you can restart the simulation under different conditions, e.g. writing output much more frequently.

In order to obtain this reproducible trajectory, it is important to look over the list above and eliminate the factors that could affect it. Further, using

```
gmx mdrun -reprod
```

will eliminate all sources of non-reproducibility that it can, i.e. same executable + same hardware + same shared libraries + same run input file + same command line parameters will lead to reproducible results.

## 3.5 Answers to frequently asked questions (FAQs)

### 3.5.1 Questions regarding GROMACS installation

1. Do I need to compile all utilities with MPI?

   With one rarely-used exception (*pme_error* (page 285)), only *mdrun* (page 258) is able to use the *MPI* (page 10) parallelism. So you only need to use the `-DGMX_MPI=on` flag when *configuring* (page 16) for a build intended to run the main simulation engine *mdrun* (page 258). Generally that is desirable when running on a multi-node cluster, and necessary when using multi-simulation algorithms. Usually also installing a build of GROMACS configured without MPI is convenient for users.

2. Should my version be compiled using double precision?

   In general, GROMACS only needs to be build in its default mixed-precision mode. For more details, see the discussion in Chapter 2 of the reference manual. Sometimes, usage may also depend on your target system, and should be decided upon according to the *individual instructions* (page 32).

### 3.5.2 Questions concerning system preparation and preprocessing

1. Where can I find a solvent *coordinate file* (page 573) for use with *solvate* (page 319)?

   Suitable equilibrated boxes of solvent *structure files* (page 573) can be found in the `$GMXDIR/share/gromacs/top` directory. That location will be searched by default by *solvate* (page 319), for example by using `-cs spc216.gro` as an argument. Other solvent boxes can be prepared by the user as described on the manual page for *solvate* (page 319) and elsewhere. Note that suitable topology files will be needed for the solvent boxes to be useful in *grompp* (page 230). These are available for some force fields, and may be found in the respective subfolder of `$GMXDIR/share/gromacs/top`.

2. How to prevent *solvate* (page 319) from placing waters in undesired places?

   Water placement is generally well behaved when solvating proteins, but can be difficult when setting up membrane or micelle simulations. In those cases, waters may be placed in between the alkyl chains of the lipids, leading to problems later *during the simulation* (page 390). You can either remove those waters by hand (and do the accounting for molecule types in the *topology* (page 584) file), or set up a local copy of the `vdwradii.dat` file from the `$GMXLIB` directory, specific for your project and located in your working directory. In it, you can increase the vdW radius of the atoms, to suppress such interstitial insertions. Recommended e.g. at a common tutorial is the use of 0.375 instead of 0.15.

1. How do I provide multiple definitions of bonds / dihedrals in a topology?

   You can add additional bonded terms beyond those that are normally defined for a residue (e.g. when defining a special ligand) by including additional copies of the respective lines under the `[ bonds ]`, `[ pairs ]`, `[ angles ]` and `[ dihedrals ]` sections in the `[ moleculetype ]` section for your molecule, found either in the *itp* (page 578) file or the *topology* (page 584) file. This will **add** those extra terms to the potential energy evaluation, but **will not** remove the previous ones. So be careful with duplicate entries. Also keep in mind that this **does not** apply to duplicated entries for `[ bondtypes ]`, `[ angletypes ]`, or `[ dihedraltypes ]`, in force-field definition files, where duplicates overwrite the previous values.

2. Do I really need a *gro* (page 577) file?

   The *gro* (page 577) file is used in GROMACS as a unified *structure file* (page 573) format that can be read by all utilities. The large majority of GROMACS routines can also use other file types such as *pdb* (page 581), with the limitations that no velocities are available in *this case* (page 39). If you need a text-based format with more digits of precision, the *g96* (page 576) format is suitable and supported.

3. Do I always need to run *pdb2gmx* (page 281) when I already produced an *itp* (page 578) file elsewhere?

   You don't need to prepare additional files if you already have all *itp* (page 578) and *top* (page 584) files prepared through other tools.

   Examples for those can be found in the *System Preparation section of this user guide* (page 40).

4. How can I build in missing atoms?

   GROMACS has no support for building coordinates of missing non-hydrogen atoms. If your system is missing some part, you will have to add the missing pieces using external programs to avoid the *missing atom* (page 134) error. This can be done using programs such as Chimera in combination with Modeller, Swiss PDB Viewer, Maestro. **Do not run** a simulation that had missing atoms unless you know exactly why it will be stable.

5. Why is the total charge of my system not an integer like it should be?

   In *floating point* (page 405) math, real numbers can not be displayed to arbitrary precision (for more on this, see e.g. Wikipedia). This means that very small differences to the final integer value will persist, and GROMACS will not lie to you and round those values up or down. If your charge differs from the integer value by a larger amount, e.g. at least 0.01, this usually means that something went wrong during your system preparation

### 3.5.3 Questions regarding simulation methodology

1. Should I couple a handful of ions to their own temperature-coupling bath?

   **No**. You need to consider the minimal size of your temperature coupling groups, as explained in *Thermostats* (page 387) and more specifically in *What not to do* (page 388), as well as the implementation of your chosen thermostat as described in the reference manual.

2. Why do my grompp restarts always start from time zero?

   You can choose different values for `tinit` (page 53) and `init-step` (page 53).

3. Why can't I do conjugate gradient minimization with constraints?

   Minimization with the conjugate gradient scheme can not be performed with constraints as described in the reference manual, and some additional information on Wikipedia.

4. How do I hold atoms in place in my energy minimization or simulation?

   Groups may be frozen in place using `freeze groups` (see the reference manual). It is more common to use a set of position restraints, to place penalties on movement of the atoms. Files that control this kind of behaviour can be created using *genrestr* (page 228).

5. How do I extend a completed a simulation to longer times?

Please see the section on *Managing long simulations* (page 42). You can either prepare a new *mdp* (page 579) file, or extend the simulation time in the original *tpr* (page 586) file using *convert-tpr* (page 178).

6. How should I compute a single-point energy?

   This is best achieved with the `-rerun` option to *mdrun* (page 258). See the *Re-running a simulation* (page 104) section.

### 3.5.4 Parameterization and Force Fields

1. I want to simulate a molecule (protein, DNA, etc.) which complexes with various transition metal ions, iron-sulfur clusters, or other exotic species. Parameters for these exotic species aren't available in force field X. What should I do?

   First, you should consider how well *MD* (page 392) will actually describe your system (e.g. see some of the recent literature). Many species are infeasible to model without either atomic polarizability, or QM treatments. Then you need to prepare your own set of parameters and add a new residue to your *force field* (page 393) of choice. Then you will have to validate that your system behaves in a physical way, before continuing your simulation studies. You could also try to build a more simplified model that does not rely on the complicated additions, as long as it still represents the correct *real* object in the laboratory.

2. Should I take parameters from one force field and apply them inside another that is missing them?

   **NO**. Molecules parametrized for a given *force field* (page 393) will not behave in a physical manner when interacting with other molecules that have been parametrized according to different standards. If your required molecule is not included in the force field you need to use, you will have to parametrize it yourself according to the methodology of this force field.

### 3.5.5 Analysis and Visualization

1. Why am I seeing bonds being created when I watch the trajectory?

   Most visualization softwares determine the bond status of atoms depending on a set of predefined distances. So the bonding pattern created by them might not be the one defined in your *topology* (page 584) file. What matters is the information encoded in there. If the software has read a *tpr* (page 586) file, then the information is in reliable agreement with the topology you supplied to *grompp* (page 230).

2. When visualizing a trajectory from a simulation using PBC, why are there holes or my peptide leaving the simulation box?

   Those holes and molecules moving around are just a result of molecules ranging over the *box boundaries and wrapping around* (page 386), and are not a reason for concern. You can fix the visualization using *trjconv* (page 334) to prepare the structure for analysis.

3. Why is my total simulation time not an integer like it should be?

   As the simulation time is calculated using *floating point arithmetic* (page 405), rounding errors can occur but are not of concern.

## 3.6 Force fields in GROMACS

### 3.6.1 AMBER

AMBER (Assisted Model Building and Energy Refinement) refers both to a set of molecular mechanical *force fields* (page 393) for the simulation of biomolecules and a package of molecular simulation programs.

GROMACS supports the following AMBER force fields natively:

- AMBER94
- AMBER96
- AMBER99
- AMBER99SB
- AMBER99SB-ILDN
- AMBER03
- AMBER14SB
- AMBER19SB
- AMBERGS

If you intend to use AMBER19SB in GROMACS, please use the OPC or OPC3 model for water molecules.

Information concerning the force field can be found using the following information:

- AMBER Force Fields - background about the AMBER force fields
- AMBER Programs - information about the AMBER suite of programs for molecular simulation
- ANTECHAMBER/GAFF - Generalized Amber Force Field (GAFF) which is supposed to provide parameters suitable for small molecules that are compatible with the AMBER protein/nucleic acid force fields. It is available either together with AMBER, or through the antechamber package, which is also distributed separately. There are scripts available for converting AMBER systems (set up, for example, with GAFF) to GROMACS (amb2gmx.pl, or ACPYPE), but they do require AmberTools installation to work.

### 3.6.2 CHARMM

CHARMM (Chemistry at HARvard Macromolecular Mechanics) is a both a set of force fields and a software package for *molecular dynamics* (page 392) simulations and analysis. Includes united atom (CHARMM19) and all atom (CHARMM22, CHARMM27, CHARMM36) *force fields* (page 393). The CHARMM27 force field has been ported to GROMACS and is officially supported. CHARMM36 force field files can be obtained from the MacKerell lab website, which regularly produces up-to-date CHARMM force field files in GROMACS format.

For using CHARMM36 in GROMACS, please use the following settings in the *mdp* (page 579) file:

```
constraints = h-bonds
cutoff-scheme = Verlet
```

(continues on next page)

(continued from previous page)

```
vdwtype = cutoff
vdw-modifier = force-switch
rlist = 1.2
rvdw = 1.2
rvdw-switch = 1.0
coulombtype = PME
rcoulomb = 1.2
DispCorr = no
```

Note that dispersion correction should be applied in the case of lipid monolayers, but not bilayers.

Please also note that the switching distance is a matter of some debate in lipid bilayer simulations, and it is dependent to some extent on the nature of the lipid. Some studies have found that an 0.8-1.0 nm switch is appropriate, others argue 0.8-1.2 nm is best, and yet others stand by 1.0-1.2 nm. The user is cautioned to thoroughly investigate the force field literature for their chosen lipid(s) before beginning a simulation!

### 3.6.3 GROMOS

**Warning**

The GROMOS force fields have been parametrized with a physically incorrect multiple-time-stepping scheme for a twin-range cut-off. When used with a single-range cut-off (or a correct Trotter multiple-time-stepping scheme), physical properties, such as the density, might differ from the intended values. Since there are researchers actively working on validating GROMOS with modern integrators we have not yet removed the GROMOS force fields, but you should be aware of these issues and check if molecules in your system are affected before proceeding. Further information is available in GitLab Issue 2884 , and a longer explanation of our decision to remove physically incorrect algorithms can be found at DOI:10.26434/chemrxiv.11474583.v1 .

GROMOS is is a general-purpose molecular dynamics computer simulation package for the study of biomolecular systems. It also incorporates its own force field covering proteins, nucleotides, sugars etc. and can be applied to chemical and physical systems ranging from glasses and liquid crystals, to polymers and crystals and solutions of biomolecules.

GROMACS supports the GROMOS force fields, with all parameters provided in the distribution for 43a1, 43a2, 45a3, 53a5, 53a6 and 54a7. The GROMOS force fields are *united atom force fields* (page 393), i.e. without explicit aliphatic (non-polar) hydrogens.

- GROMOS 53a6 - in GROMACS format (J. Comput. Chem. 2004 vol. 25 (13): 1656-1676).
- GROMOS 53a5 - in GROMACS format (J. Comput. Chem. 2004 vol. 25 (13): 1656-1676).
- GROMOS 43a1p - 43a1 modified to contain SEP (phosphoserine), TPO (phosphothreonine), and PTR (phosphotyrosine) (all PO42- forms), and SEPH, TPOH, PTRH (PO4H- forms).

### 3.6.4 OPLS

OPLS (Optimized Potential for Liquid Simulations) is a set of force fields developed by Prof. William L. Jorgensen for condensed phase simulations, with the latest version being OPLS-AA/M.

The standard implementations for those force fields are the *BOSS* and *MCPRO* programs developed by the Jorgensen group

As there is no central web-page to point to, the user is advised to consult the original literature for the united atom (OPLS-UA) and all atom (OPLS-AA) force fields, as well as the Jorgensen group page

## 3.7 Molecular dynamics parameters (.mdp options)

### 3.7.1 General information

Default values are given in parentheses, or listed first among choices. The first option in the list is always the default option. Units are given in square brackets. The difference between a dash and an underscore is ignored.

A *sample mdp file* (page 579) is available. This should be appropriate to start a normal simulation. Edit it to suit your specific needs and desires.

#### Preprocessing

**include**

directories to include in your topology. Format: `-I/home/john/mylib -I../otherlib`

**define**

defines to pass to the preprocessor, default is no defines. You can use any defines to control options in your customized topology files. Options that act on existing *top* (page 584) file mechanisms include

`-DFLEXIBLE` will use flexible water instead of rigid water into your topology, this can be useful for normal mode analysis.

`-DPOSRES` will trigger the inclusion of `posre.itp` into your topology, used for implementing position restraints.

#### Run control

**integrator**

(Despite the name, this list includes algorithms that are not actually integrators over time. *integrator=steep* (page 52) and all entries following it are in this category)

**md**

A leap-frog algorithm for integrating Newton's equations of motion.

**md-vv**

A velocity Verlet algorithm for integrating Newton's equations of motion. For constant NVE simulations started from corresponding points in the same trajectory, the trajectories are analytically, but not binary, identical to the *integrator=md* (page 51) leap-frog integrator.

The kinetic energy is determined from the whole step velocities and is therefore slightly too high. The advantage of this integrator is more accurate, reversible Nose-Hoover and Parrinello-Rahman coupling integration based on Trotter expansion, as well as (slightly too small) full step velocity output. This all comes at the cost of extra computation, especially with constraints and extra communication in parallel. Note that for nearly all production simulations the *integrator=md* (page 51) integrator is accurate enough.

**md-vv-avek**

A velocity Verlet algorithm identical to *integrator=md-vv* (page 51), except that the kinetic energy is determined as the average of the two half step kinetic energies as in the *integrator=md* (page 51) integrator, and this thus more accurate. With Nose-Hoover and/or Parrinello-Rahman coupling this comes with a slight increase in computational cost.

**sd**

An accurate and efficient leap-frog stochastic dynamics integrator. With constraints, coordinates needs to be constrained twice per integration step. Depending on the computational cost of the force calculation, this can take a significant part of the simulation time. The temperature for one or more groups of atoms (*tc-grps* (page 65)) is set with *ref-t* (page 65), the inverse friction constant for each group is set with *tau-t* (page 65). The parameters *tcoupl* (page 64) and *nsttcouple* (page 65) are ignored. The random generator is initialized with *ld-seed* (page 55). When used as a thermostat, an appropriate value for *tau-t* (page 65) is 2 ps, since this results in a friction that is lower than the internal friction of water, while it is high enough to remove excess heat NOTE: temperature deviations decay twice as fast as with a Berendsen thermostat with the same *tau-t* (page 65).

**bd**

An Euler integrator for Brownian or position Langevin dynamics. The velocity is the force divided by a friction coefficient (*bd-fric* (page 55)) plus random thermal noise (*ref-t* (page 65)). When *bd-fric* (page 55) is 0, the friction coefficient for each particle is calculated as mass/ *tau-t* (page 65), as for the integrator *integrator=sd* (page 52). The random generator is initialized with *ld-seed* (page 55).

**steep**

A steepest descent algorithm for energy minimization. The maximum step size is *emstep* (page 55), the tolerance is *emtol* (page 55).

**cg**

A conjugate gradient algorithm for energy minimization, the tolerance is *emtol* (page 55). CG is more efficient when a steepest descent step is done every once in a while, this is determined by *nstcgsteep* (page 55). For a minimization prior to a normal mode analysis, which requires a very high accuracy, GROMACS should be compiled in double precision.

**l-bfgs**

A quasi-Newtonian algorithm for energy minimization according to the low-memory Broyden-Fletcher-Goldfarb-Shanno approach. In practice this seems to converge faster than Conjugate Gradients, but due to the correction steps necessary it is not (yet) parallelized.

**nm**

Normal mode analysis is performed on the structure in the *tpr* (page 586) file. GROMACS should be compiled in double precision.

**tpi**

Test particle insertion. The last molecule in the topology is the test particle. A trajectory must be provided to `mdrun -rerun`. This trajectory should not contain the molecule to

be inserted. Insertions are performed *nsteps* (page 53) times in each frame at random locations and with random orientations of the molecule. When *nstlist* (page 57) is larger than one, *nstlist* (page 57) insertions are performed in a sphere with radius *rtpi* (page 56) around a the same random location using the same pair list. Since pair list construction is expensive, one can perform several extra insertions with the same list almost for free. The random seed is set with *ld-seed* (page 55). The temperature for the Boltzmann weighting is set with *ref-t* (page 65), this should match the temperature of the simulation of the original trajectory. Dispersion correction is implemented correctly for TPI. All relevant quantities are written to the file specified with `mdrun -tpi`. The distribution of insertion energies is written to the file specified with `mdrun -tpid`. No trajectory or energy file is written. Parallel TPI gives identical results to single-node TPI. For charged molecules, using PME with a fine grid is most accurate and also efficient, since the potential in the system only needs to be calculated once per frame.

**tpic**

Test particle insertion into a predefined cavity location. The procedure is the same as for *integrator=tpi* (page 52), except that one coordinate extra is read from the trajectory, which is used as the insertion location. The molecule to be inserted should be centered at 0,0,0. GROMACS does not do this for you, since for different situations a different way of centering might be optimal. Also *rtpi* (page 56) sets the radius for the sphere around this location. Neighbor searching is done only once per frame, *nstlist* (page 57) is not used. Parallel *integrator=tpic* (page 53) gives identical results to single-rank *integrator=tpic* (page 53).

**mimic**

Enable MiMiC QM/MM coupling to run hybrid molecular dynamics. Keey in mind that its required to launch CPMD compiled with MiMiC as well. In this mode all options regarding integration (T-coupling, P-coupling, timestep and number of steps) are ignored as CPMD will do the integration instead. Options related to forces computation (cutoffs, PME parameters, etc.) are working as usual. Atom selection to define QM atoms is read from *QMMM-grps* (page 94)

**tinit**

(0) [ps] starting time for your run (only makes sense for time-based integrators)

**dt**

(0.001) [ps] time step for integration (only makes sense for time-based integrators)

**nsteps**

(0) maximum number of steps to integrate or minimize, -1 is no maximum

**init-step**

(0) The starting step. The time at step i in a run is calculated as: t = *tinit* (page 53) + *dt* (page 53) * (*init-step* (page 53) + i). The free-energy lambda is calculated as: lambda = *init-lambda* (page 85) + *delta-lambda* (page 85) * (*init-step* (page 53) + i). Also non-equilibrium MD parameters can depend on the step number. Thus for exact restarts or redoing part of a run it might be necessary to set *init-step* (page 53) to the step number of the restart frame. *gmx convert-tpr* (page 178) does this automatically.

**simulation-part**

(0) A simulation can consist of multiple parts, each of which has a part number. This option specifies what that number will be, which helps keep track of parts that are logically the same simulation. This option is generally useful to set only when coping with a crashed simulation where files were lost.

**mts**

**no**

Evaluate all forces at every integration step.

**yes**

Use a multiple timing-stepping integrator to evaluate some forces, as specified by *mts-level2-forces* (page 54) every *mts-level2-factor* (page 54) integration steps. All other forces are evaluated at every step. MTS is currently only supported with *integrator=md* (page 51).

**mts-levels**

(2) The number of levels for the multiple time-stepping scheme. Currently only 2 is supported.

**mts-level2-forces**

(longrange-nonbonded) A list of one or more force groups that will be evaluated only every *mts-level2-factor* (page 54) steps. Supported entries are: `longrange-nonbonded`, `nonbonded`, `pair`, `dihedral`, `angle`, `pull` and `awh`. With `pair` the listed pair forces (such as 1-4) are selected. With `dihedral` all dihedrals are selected, including cmap. All other forces, including all restraints, are evaluated and integrated every step. When PME or Ewald is used for electrostatics and/or LJ interactions, `longrange-nonbonded` can not be omitted here.

**mts-level2-factor**

(2) [steps] Interval for computing the forces in level 2 of the multiple time-stepping scheme

**mass-repartition-factor**

(1) [] Scales the masses of the lightest atoms in the system by this factor to the mass mMin. All atoms with a mass lower than mMin also have their mass set to that mMin. The mass change is subtracted from the mass of the atom the light atom is bound to. If there is no bound atom a warning is generated. If there is more than one atom bound an error is generated. If the mass of the bound atom would become lower than mMin an error is generated. For typical atomistic systems only the masses of hydrogens are scaled. With *constraints=h-bonds* (page 69), a factor of 3 will usually enable a time step of 4 fs.

**comm-mode**

**Linear**

Remove center of mass translational velocity

**Angular**

Remove center of mass translational and rotational velocity

**Linear-acceleration-correction**

Remove center of mass translational velocity. Correct the center of mass position assuming linear acceleration over *nstcomm* (page 54) steps. This is useful for cases where an acceleration is expected on the center of mass which is nearly constant over *nstcomm* (page 54) steps. This can occur for example when pulling on a group using an absolute reference.

**None**

No restriction on the center of mass motion

**nstcomm**

(100) [steps] interval for center of mass motion removal

**comm-grps**

group(s) for center of mass motion removal, default is the whole system

### Langevin dynamics

**bd-fric**

(0) [amu ps$^{-1}$] Brownian dynamics friction coefficient. When *bd-fric* (page 55) is 0, the friction coefficient for each particle is calculated as mass/ *tau-t* (page 65).

**ld-seed**

(-1) [integer] used to initialize random generator for thermal noise for stochastic and Brownian dynamics. When *ld-seed* (page 55) is set to -1, a pseudo random seed is used. When running BD or SD on multiple processors, each processor uses a seed equal to *ld-seed* (page 55) plus the processor number.

### Energy minimization

**emtol**

(10.0) [kJ mol$^{-1}$ nm$^{-1}$] the minimization is converged when the maximum force is smaller than this value

**emstep**

(0.01) [nm] initial step-size

**nstcgsteep**

(1000) [steps] interval of performing 1 steepest descent step while doing conjugate gradient energy minimization.

**nbfgscorr**

(10) Number of correction steps to use for L-BFGS minimization. A higher number is (at least theoretically) more accurate, but slower.

### Shell Molecular Dynamics

When shells or flexible constraints are present in the system the positions of the shells and the lengths of the flexible constraints are optimized at every time step until either the RMS force on the shells and constraints is less than *emtol* (page 55), or a maximum number of iterations *niter* (page 55) has been reached. Minimization is converged when the maximum force is smaller than *emtol* (page 55). For shell MD this value should be 1.0 at most.

**niter**

(20) maximum number of iterations for optimizing the shell positions and the flexible constraints.

**fcstep**

(0) [ps$^2$] the step size for optimizing the flexible constraints. Should be chosen as mu/(d2V/dq2) where mu is the reduced mass of two particles in a flexible constraint and d2V/dq2 is the second derivative of the potential in the constraint direction. Hopefully this number does not differ too much between the flexible constraints, as the number of iterations and thus the runtime is very sensitive to fcstep. Try several values!

### Test particle insertion

**rtpi**

(0.05) [nm] the test particle insertion radius, see integrators *integrator=tpi* (page 52) and *integrator=tpic* (page 53)

### Output control

**nstxout**

(0) [steps] number of steps that elapse between writing coordinates to the output trajectory file (*trr* (page 586)), the first and last coordinates are always written unless 0, which means coordinates are not written into the trajectory file.

**nstvout**

(0) [steps] number of steps that elapse between writing velocities to the output trajectory file (*trr* (page 586)), the first and last velocities are always written unless 0, which means velocities are not written into the trajectory file.

**nstfout**

(0) [steps] number of steps that elapse between writing forces to the output trajectory file (*trr* (page 586)), the first and last forces are always written, unless 0, which means forces are not written into the trajectory file.

**nstlog**

(1000) [steps] number of steps that elapse between writing energies to the log file, the first and last energies are always written.

**nstcalcenergy**

(100) number of steps that elapse between calculating the energies, 0 is never. This option is only relevant with dynamics. This option affects the performance in parallel simulations, because calculating energies requires global communication between all processes which can become a bottleneck at high parallelization.

**nstenergy**

(1000) [steps] number of steps that elapse between writing energies to the energy file (*edr* (page 576)), the first and last energies are always written, should be a multiple of *nstcalcenergy* (page 56). Note that the exact sums and fluctuations over all MD steps modulo *nstcalcenergy* (page 56) are stored in the energy file, so *gmx energy* (page 214) can report exact energy averages and fluctuations also when *nstenergy* (page 56) > 1

**nstxout-compressed**

(0) [steps] number of steps that elapse between writing position coordinates using lossy compression (*xtc* (page 588) file), the first and last coordinates are always written, unless 0, which means that there is no compressed coordinates output.

**compressed-x-precision**

(1000) [real] precision with which to write to the compressed trajectory file

**compressed-x-grps**

group(s) to write to the compressed trajectory file, by default the whole system is written (if *nstxout-compressed* (page 56) > 0)

**energygrps**

group(s) for which to write to write short-ranged non-bonded potential energies to the energy file (not supported on GPUs)

### Neighbor searching

**cutoff-scheme**

**Verlet**

Generate a pair list with buffering. The buffer size is automatically set based on *verlet-buffer-tolerance* (page 58), unless this is set to -1, in which case *rlist* (page 58) will be used.

**group**

Generate a pair list for groups of atoms, corresponding to the charge groups in the topology. This option is no longer supported.

**nstlist**

(10) [steps]

**>0**

Interval between steps that update the neighbor list. When dynamics and *verlet-buffer-tolerance* (page 58) set, *nstlist* (page 57) is actually a minimum value and *gmx mdrun* (page 258) might increase it, unless it is set to 1. With parallel simulations and/or non-bonded force calculation on the GPU, a value of 20 or 40 often gives the best performance. With energy minimization this parameter is not used as the pair list is updated when at least one atom has moved by more than half the pair list buffer size.

**0**

The neighbor list is only constructed once and never updated. This is mainly useful for vacuum simulations in which all particles see each other. But vacuum simulations are (temporarily) not supported.

**<0**

Unused.

**pbc**

**xyz**

Use periodic boundary conditions in all directions.

**no**

Use no periodic boundary conditions, ignore the box. To simulate without cut-offs, set all cut-offs and *nstlist* (page 57) to 0. For best performance without cut-offs on a single MPI rank, set *nstlist* (page 57) to zero.

**xy**

Use periodic boundary conditions in x and y directions only. This can be used in combination with *walls* (page 71). Without walls or with only one wall the system size is infinite in the z direction. Therefore pressure coupling or Ewald summation methods can not be used. These disadvantages do not apply when two walls are used.

**periodic-molecules**

**no**

molecules are finite, fast molecular PBC can be used

**yes**

for systems with molecules that couple to themselves through the periodic boundary conditions, this requires a slower PBC algorithm and molecules are not made whole in the output

**verlet-buffer-tolerance**

(0.005) [kJ mol$^{-1}$ ps$^{-1}$]

Used when performing a simulation with dynamics. This sets the maximum allowed error for pair interactions per particle caused by the Verlet buffer, which indirectly sets *rlist* (page 58). As both *nstlist* (page 57) and the Verlet buffer size are fixed (for performance reasons), particle pairs not in the pair list can occasionally get within the cut-off distance during *nstlist* (page 57) -1 steps. This causes very small jumps in the energy. In a constant-temperature ensemble, these very small energy jumps can be estimated for a given cut-off and *rlist* (page 58). The estimate assumes a homogeneous particle distribution, hence the errors might be slightly underestimated for multi-phase systems. (See the reference manual for details). For longer pair-list life-time (*nstlist* (page 57) -1) * *dt* (page 53) the buffer is overestimated, because the interactions between particles are ignored. Combined with cancellation of errors, the actual drift of the total energy is usually one to two orders of magnitude smaller. Note that the generated buffer size takes into account that the GROMACS pair-list setup leads to a reduction in the drift by a factor 10, compared to a simple particle-pair based list. Without dynamics (energy minimization etc.), the buffer is 5% of the cut-off. For NVE simulations the initial temperature is used, unless this is zero, in which case a buffer of 10% is used. For NVE simulations the tolerance usually needs to be lowered to achieve proper energy conservation on the nanosecond time scale. To override the automated buffer setting, use *verlet-buffer-tolerance* (page 58) =-1 and set *rlist* (page 58) manually.

**verlet-buffer-pressure-tolerance**

(0.5) [bar]

Used when performing a simulation with dynamics and only active when *verlet-buffer-tolerance* (page 58) is positive. This sets the maximum tolerated error in the average pressure due to missing Lennard-Jones interactions of particle pairs that are not in the pair list, but come within *rvdw* (page 62) range as the pair list ages. As for the drift tolerance, the (over)estimate of the pressure error is tight at short times. At longer time it turns into a significant overestimate, because interactions limit the displacement of particles. Note that the default tolerance of 0.5 bar corresponds to a maximum relative deviation of the density of liquid water of 2e-5.

**rlist**

(1) [nm] Cut-off distance for the short-range neighbor list. With dynamics, this is by default set by the *verlet-buffer-tolerance* (page 58) and *verlet-buffer-pressure-tolerance* (page 58) options and the value of *rlist* (page 58) is ignored. Without dynamics, this is by default set to the maximum cut-off plus 5% buffer, except for test particle insertion, where the buffer is managed exactly and automatically. For NVE simulations, where the automated setting is not possible, the advised procedure is to run *gmx grompp* (page 230) with an NVT setup with the expected temperature and copy the resulting value of *rlist* (page 58) to the NVE setup.

### Electrostatics

**coulombtype**

**Cut-off**

Plain cut-off with pair list radius *rlist* (page 58) and Coulomb cut-off *rcoulomb* (page 60), where *rlist* (page 58) >= *rcoulomb* (page 60). Note that with the (default) setting of *coulomb-modifier* (page 60) =Potential-shift not only the potentials between interacting pairs are shifted to be zero at the cut-off, but the same shift is also applied to excluded pairs. This does not lead to forces between excluded pairs, but does add a constant offset to the total Coulomb potential.

**Ewald**

Classical Ewald sum electrostatics. The real-space cut-off *rcoulomb* (page 60) should be equal to *rlist* (page 58). Use *e.g.* *rlist* (page 58) =0.9, *rcoulomb* (page 60) =0.9. The highest magnitude of wave vectors used in reciprocal space is controlled by *fourierspacing* (page 62). The relative accuracy of direct/reciprocal space is controlled by *ewald-rtol* (page 63).

NOTE: Ewald scales as $O(N^{3/2})$ and is thus extremely slow for large systems. It is included mainly for reference - in most cases PME will perform much better.

**PME**

Fast smooth Particle-Mesh Ewald (SPME) electrostatics. Direct space is similar to the Ewald sum, while the reciprocal part is performed with FFTs. Grid dimensions are controlled with *fourierspacing* (page 62) and the interpolation order with *pme-order* (page 63). With a grid spacing of 0.1 nm and cubic interpolation the electrostatic forces have an accuracy of $2\text{-}3*10^{-4}$. Since the error from the vdw-cutoff is larger than this you might try 0.15 nm. When running in parallel the interpolation parallelizes better than the FFT, so try decreasing grid dimensions while increasing interpolation.

**P3M-AD**

Particle-Particle Particle-Mesh algorithm with analytical derivative for for long-range electrostatic interactions. The method and code is identical to SPME, except that the influence function is optimized for the grid. This gives a slight increase in accuracy.

**Reaction-Field**

Reaction field electrostatics with Coulomb cut-off *rcoulomb* (page 60), where *rlist* (page 58) >= *rvdw* (page 62). The dielectric constant beyond the cut-off is *epsilon-rf* (page 60). The dielectric constant can be set to infinity by setting *epsilon-rf* (page 60) =0.

**User**

Currently unsupported. *gmx mdrun* (page 258) will now expect to find a file `table.xvg` with user-defined potential functions for repulsion, dispersion and Coulomb. When pair interactions are present, *gmx mdrun* (page 258) also expects to find a file `tablep.xvg` for the pair interactions. When the same interactions should be used for non-bonded and pair interactions the user can specify the same file name for both table files. These files should contain 7 columns: the `x` value, `f(x)`, `-f'(x)`, `g(x)`, `-g'(x)`, `h(x)`, `-h'(x)`, where `f(x)` is the Coulomb function, `g(x)` the dispersion function and `h(x)` the repulsion function. When *vdwtype* (page 61) is not set to User the values for `g`, `-g'`, `h` and `-h'` are ignored. For the non-bonded interactions `x` values should run from 0 to the largest cut-off distance + *table-extension* (page 62) and should be uniformly spaced. For the pair

interactions the table length in the file will be used. The optimal spacing, which is used for non-user tables, is `0.002 nm` when you run in mixed precision or `0.0005 nm` when you run in double precision. The function value at `x=0` is not important. More information is in the printed manual.

**PME-Switch**

Currently unsupported. A combination of PME and a switch function for the direct-space part (see above). *rcoulomb* (page 60) is allowed to be smaller than *rlist* (page 58).

**PME-User**

Currently unsupported. A combination of PME and user tables (see above). *rcoulomb* (page 60) is allowed to be smaller than *rlist* (page 58). The PME mesh contribution is subtracted from the user table by *gmx mdrun* (page 258). Because of this subtraction the user tables should contain about 10 decimal places.

**PME-User-Switch**

Currently unsupported. A combination of PME-User and a switching function (see above). The switching function is applied to final particle-particle interaction, *i.e.* both to the user supplied function and the PME Mesh correction part.

**coulomb-modifier**

**Potential-shift**

Shift the Coulomb potential by a constant such that it is zero at the cut-off. This makes the potential the integral of the force. Note that this does not affect the forces or the sampling.

**None**

Use an unmodified Coulomb potential. This can be useful when comparing energies with those computed with other software.

**rcoulomb-switch**

(0) [nm] where to start switching the Coulomb potential, only relevant when force or potential switching is used

**rcoulomb**

(1) [nm] The distance for the Coulomb cut-off. Note that with PME this value can be increased by the PME tuning in *gmx mdrun* (page 258) along with the PME grid spacing.

**epsilon-r**

(1) The relative dielectric constant. A value of 0 means infinity.

**epsilon-rf**

(0) The relative dielectric constant of the reaction field. This is only used with reaction-field electrostatics. A value of 0 means infinity.

### Van der Waals

**vdwtype**

**Cut-off**

Plain cut-off with pair list radius *rlist* (page 58) and VdW cut-off *rvdw* (page 62), where *rlist* (page 58) >= *rvdw* (page 62).

**PME**

Fast smooth Particle-mesh Ewald (SPME) for VdW interactions. The grid dimensions are controlled with *fourierspacing* (page 62) in the same way as for electrostatics, and the interpolation order is controlled with *pme-order* (page 63). The relative accuracy of direct/reciprocal space is controlled by *ewald-rtol-lj* (page 63), and the specific combination rules that are to be used by the reciprocal routine are set using *lj-pme-comb-rule* (page 63).

**Shift**

This functionality is deprecated and replaced by using *vdwtype=Cut-off* (page 61) with *vdw-modifier=Force-switch* (page 61). The LJ (not Buckingham) potential is decreased over the whole range and the forces decay smoothly to zero between *rvdw-switch* (page 62) and *rvdw* (page 62).

**Switch**

This functionality is deprecated and replaced by using *vdwtype=Cut-off* (page 61) with *vdw-modifier=Potential-switch* (page 61). The LJ (not Buckingham) potential is normal out to *rvdw-switch* (page 62), after which it is switched off to reach zero at *rvdw* (page 62). Both the potential and force functions are continuously smooth, but be aware that all switch functions will give rise to a bulge (increase) in the force (since we are switching the potential).

**User**

Currently unsupported. See *coulombtype=User* (page 59) for instructions. The function value at zero is not important. When you want to use LJ correction, make sure that *rvdw* (page 62) corresponds to the cut-off in the user-defined function. When *coulombtype* (page 59) is not set to User the values for the `f` and `-f'` columns are ignored.

**vdw-modifier**

**Potential-shift**

Shift the Van der Waals potential by a constant such that it is zero at the cut-off. This makes the potential the integral of the force. Note that this does not affect the forces or the sampling.

**None**

Use an unmodified Van der Waals potential. This can be useful when comparing energies with those computed with other software.

**Force-switch**

Smoothly switches the forces to zero between *rvdw-switch* (page 62) and *rvdw* (page 62). This shifts the potential shift over the whole range and switches it to zero at the cut-off. Note that this is more expensive to calculate than a plain cut-off and it is not required for energy conservation, since Potential-shift conserves energy just as well.

**Potential-switch**

Smoothly switches the potential to zero between *rvdw-switch* (page 62) and *rvdw* (page 62). Note that this introduces articifically large forces in the switching region and is much more expensive to calculate. This option should only be used if the force field you are using requires this.

**rvdw-switch**

(0) [nm] where to start switching the LJ force and possibly the potential, only relevant when force or potential switching is used

**rvdw**

(1) [nm] distance for the LJ or Buckingham cut-off

**DispCorr**

**no**

do not apply any correction

**EnerPres**

apply long-range dispersion corrections for Energy and Pressure

**Ener**

apply long-range dispersion corrections for Energy only

### Tables

**table-extension**

(1) [nm] Extension of the non-bonded potential lookup tables beyond the largest cut-off distance. With actual non-bonded interactions the tables are never accessed beyond the cut-off. But a longer table length might be needed for the 1-4 interactions, which are always tabulated irrespective of the use of tables for the non-bonded interactions.

**energygrp-table**

Currently unsupported. When user tables are used for electrostatics and/or VdW, here one can give pairs of energy groups for which separate user tables should be used. The two energy groups will be appended to the table file name, in order of their definition in *energygrps* (page 56), separated by underscores. For example, if `energygrps = Na Cl Sol` and `energygrp-table = Na Na Na Cl`, *gmx mdrun* (page 258) will read `table_Na_Na.xvg` and `table_Na_Cl.xvg` in addition to the normal `table.xvg` which will be used for all other energy group pairs.

### Ewald

**fourierspacing**

(0.12) [nm] For ordinary Ewald, the ratio of the box dimensions and the spacing determines a lower bound for the number of wave vectors to use in each (signed) direction. For PME and P3M, that ratio determines a lower bound for the number of Fourier-space grid points that will be used along that axis. In all cases, the number for each direction can be overridden by entering a non-zero value for that *fourier-nx* (page 63) direction. For optimizing the relative load of the particle-particle interactions and the mesh part of PME, it is useful to know that the accuracy of the electrostatics remains nearly constant when the Coulomb cut-off and the PME grid spacing

are scaled by the same factor. Note that this spacing can be scaled up along with *rcoulomb* (page 60) by the PME tuning in *gmx mdrun* (page 258).

**fourier-nx**

**fourier-ny**

**fourier-nz**

(0) Highest magnitude of wave vectors in reciprocal space when using Ewald. Grid size when using PME or P3M. These values override *fourierspacing* (page 62) per direction. The best choice is powers of 2, 3, 5 and 7. Avoid large primes. Note that these grid sizes can be reduced along with scaling up *rcoulomb* (page 60) by the PME tuning in *gmx mdrun* (page 258).

**pme-order**

(4) The number of grid points along a dimension to which a charge is mapped. The actual order of the PME interpolation is one less, e.g. the default of 4 gives cubic interpolation. Supported values are 3 to 12 (max 8 for P3M-AD). When running in parallel, it can be worth to switch to 5 and simultaneously increase the grid spacing. Note that on the CPU only values 4 and 5 have SIMD acceleration and GPUs only support the value 4.

**ewald-rtol**

($10^{-5}$) The relative strength of the Ewald-shifted direct potential at *rcoulomb* (page 60) is given by *ewald-rtol* (page 63). Decreasing this will give a more accurate direct sum, but then you need more wave vectors for the reciprocal sum.

**ewald-rtol-lj**

($10^{-3}$) When doing PME for VdW-interactions, *ewald-rtol-lj* (page 63) is used to control the relative strength of the dispersion potential at *rvdw* (page 62) in the same way as *ewald-rtol* (page 63) controls the electrostatic potential.

**lj-pme-comb-rule**

(Geometric) The combination rules used to combine VdW-parameters in the reciprocal part of LJ-PME. Geometric rules are much faster than Lorentz-Berthelot and usually the recommended choice, even when the rest of the force field uses the Lorentz-Berthelot rules.

**Geometric**

Apply geometric combination rules

**Lorentz-Berthelot**

Apply Lorentz-Berthelot combination rules

**ewald-geometry**

**3d**

The Ewald sum is performed in all three dimensions.

**3dc**

The reciprocal sum is still performed in 3D, but a force and potential correction applied in the `z` dimension to produce a pseudo-2D summation. If your system has a slab geometry in the `x-y` plane you can try to increase the `z`-dimension of the box (a box height of 3 times the slab height is usually ok) and use this option.

**epsilon-surface**

(0) This controls the dipole correction to the Ewald summation in 3D. The default value of zero means it is turned off. Turn it on by setting it to the value of the relative permittivity of the

imaginary surface around your infinite system. Be careful - you should not use this if you have free mobile charges in your system. This value does not affect the slab 3DC variant of the long-range corrections.

### Temperature coupling

**ensemble-temperature-setting**

**auto**

With this setting *gmx grompp* (page 230) will determine which of the next three settings is available and choose the appropriate one. When all atoms are coupled to a temperature bath with the same temperature, a constant ensemble temperature is chosen and the value is taken from the temperature bath.

**constant**

The system has a constant ensemble temperature given by *ensemble-temperature* (page 64). A constant ensemble temperature is required for certain sampling algorithms such as AWH.

**variable**

The system has a variable ensemble temperature due to simulated annealing or simulated tempering. The system ensemble temperature is set dynamically during the simulation.

**not-available**

The system has no ensemble temperature.

**ensemble-temperature**

(-1) [K]

The ensemble temperature for the system. The input value is only used with `ensemble-temperature-setting=constant`. By default the ensemble temperature is copied from the temperature of the thermal bath (when used).

**tcoupl**

**no**

No temperature coupling.

**berendsen**

Temperature coupling with a Berendsen thermostat to a bath with temperature *ref-t* (page 65), with time constant *tau-t* (page 65). Several groups can be coupled separately, these are specified in the *tc-grps* (page 65) field separated by spaces. This is a historical thermostat needed to be able to reproduce previous simulations, but we strongly recommend not to use it for new production runs. Consult the manual for details.

**nose-hoover**

Temperature coupling using a Nose-Hoover extended ensemble. The reference temperature and coupling groups are selected as above, but in this case *tau-t* (page 65) controls the period of the temperature fluctuations at equilibrium, which is slightly different from a relaxation time. For NVT simulations the conserved energy quantity is written to the energy and log files.

**andersen**

Temperature coupling by randomizing a fraction of the particle velocities at each timestep. Reference temperature and coupling groups are selected as above. *tau-t* (page 65) is the average time between randomization of each molecule. Inhibits particle dynamics somewhat, but has little or no ergodicity issues. Currently only implemented with velocity Verlet, and not implemented with constraints.

**andersen-massive**

Temperature coupling by randomizing velocities of all particles at infrequent timesteps. Reference temperature and coupling groups are selected as above. *tau-t* (page 65) is the time between randomization of all molecules. Inhibits particle dynamics somewhat, but has little or no ergodicity issues. Currently only implemented with velocity Verlet.

**v-rescale**

Temperature coupling using velocity rescaling with a stochastic term (JCP 126, 014101). This thermostat is similar to Berendsen coupling, with the same scaling using *tau-t* (page 65), but the stochastic term ensures that a proper canonical ensemble is generated. The random seed is set with *ld-seed* (page 55). This thermostat works correctly even for *tau-t* (page 65) =0. For NVT simulations the conserved energy quantity is written to the energy and log file.

**nsttcouple**

(-1) The interval between steps that couple the temperature. The default value of -1 sets *nsttcouple* (page 65) equal to 100, or fewer steps if required for accurate integration (5 steps per tau for first order coupling, 20 steps per tau for second order coupling). Note that the default value is large in order to reduce the overhead of the additional computation and communication required for obtaining the kinetic energy. For velocity Verlet integrators *nsttcouple* (page 65) is set to 1.

**nh-chain-length**

(10) The number of chained Nose-Hoover thermostats for velocity Verlet integrators, the leap-frog *integrator=md* (page 51) integrator only supports 1. Data for the NH chain variables is not printed to the *edr* (page 576) file by default, but can be turned on with the *print-nose-hoover-chain-variables* (page 65) option.

**print-nose-hoover-chain-variables**

**no**

Do not store Nose-Hoover chain variables in the energy file.

**yes**

Store all positions and velocities of the Nose-Hoover chain in the energy file.

**tc-grps**

groups to couple to separate temperature baths

**tau-t**

[ps] time constant for coupling (one for each group in *tc-grps* (page 65)), -1 means no temperature coupling

**ref-t**

[K] reference temperature for coupling (one for each group in *tc-grps* (page 65))

### Pressure coupling

**pcoupl**

**no**

No pressure coupling. This means a fixed box size.

**Berendsen**

Exponential relaxation pressure coupling with time constant *tau-p* (page 67). The box is scaled every *nstpcouple* (page 67) steps. This barostat does not yield a correct thermodynamic ensemble; it is only included to be able to reproduce previous runs, and we strongly recommend against using it for new simulations. See the manual for details.

**C-rescale**

Exponential relaxation pressure coupling with time constant *tau-p* (page 67), including a stochastic term to enforce correct volume fluctuations. The box is scaled every *nstpcouple* (page 67) steps. It can be used for both equilibration and production.

**Parrinello-Rahman**

Extended-ensemble pressure coupling where the box vectors are subject to an equation of motion. The equation of motion for the atoms is coupled to this. No instantaneous scaling takes place. As for Nose-Hoover temperature coupling the time constant *tau-p* (page 67) is the period of pressure fluctuations at equilibrium. This is a good method when you want to apply pressure scaling during data collection, but beware that you can get very large oscillations if you are starting from a different pressure. For simulations where the exact fluctations of the NPT ensemble are important, or if the pressure coupling time is very short, it may not be appropriate, as the previous time step pressure is used in some steps of the GROMACS implementation for the current time step pressure.

**MTTK**

Martyna-Tuckerman-Tobias-Klein implementation, only useable with *integrator=md-vv* (page 51) or *integrator=md-vv-avek* (page 52), very similar to Parrinello-Rahman. As for Nose-Hoover temperature coupling the time constant *tau-p* (page 67) is the period of pressure fluctuations at equilibrium. This is probably a better method when you want to apply pressure scaling during data collection, but beware that you can get very large oscillations if you are starting from a different pressure. This requires a constant ensemble temperature for the system. It only supports isotropic scaling, and only works without constraints. MTTK coupling is deprecated.

**pcoupltype**

Specifies the kind of isotropy of the pressure coupling used. Each kind takes one or more values for *compressibility* (page 67) and *ref-p* (page 67). Only a single value is permitted for *tau-p* (page 67).

**isotropic**

Isotropic pressure coupling with time constant *tau-p* (page 67). One value each for *compressibility* (page 67) and *ref-p* (page 67) is required.

**semiisotropic**

Pressure coupling which is isotropic in the `x` and `y` direction, but different in the `z` direction. This can be useful for membrane simulations. Two values each for *compressibility* (page 67) and *ref-p* (page 67) are required, for `x/y` and `z` directions respectively.

**anisotropic**

Same as before, but 6 values are needed for `xx`, `yy`, `zz`, `xy/yx`, `xz/zx` and `yz/zy` components, respectively. When the off-diagonal compressibilities are set to zero, a rectangular box will stay rectangular. Beware that anisotropic scaling can lead to extreme deformation of the simulation box.

**surface-tension**

Surface tension coupling for surfaces parallel to the xy-plane. Uses normal pressure coupling for the `z`-direction, while the surface tension is coupled to the `x/y` dimensions of the box. The first *ref-p* (page 67) value is the reference surface tension times the number of surfaces `bar nm`, the second value is the reference `z`-pressure `bar`. The two *compressibility* (page 67) values are the compressibility in the `x/y` and `z` direction respectively. The value for the `z`-compressibility should be reasonably accurate since it influences the convergence of the surface-tension, it can also be set to zero to have a box with constant height.

**nstpcouple**

(-1) The interval between steps that couple the pressure. The default value of -1 sets *nstpcouple* (page 67) equal to 100, or fewer steps if required for accurate integration (5 steps per tau for first order coupling, 20 steps per tau for second order coupling). Note that the default value is large in order to reduce the overhead of the additional computation and communication required for obtaining the virial and kinetic energy. For velocity Verlet integrators *nsttcouple* (page 65) is set to 1.

**tau-p**

(5) [ps] The time constant for pressure coupling (one value for all directions).

**compressibility**

[bar$^{-1}$] The compressibility (NOTE: this is now really in bar$^{-1}$) For water at 1 atm and 300 K the compressibility is 4.5e-5 bar$^{-1}$. The number of required values is implied by *pcoupltype* (page 66).

**ref-p**

[bar] The reference pressure for coupling. The number of required values is implied by *pcoupltype* (page 66).

**refcoord-scaling**

**no**

The reference coordinates for position restraints are not modified. Note that with this option the virial and pressure might be ill defined, see *here* (page 500) for more details.

**all**

The reference coordinates are scaled with the scaling matrix of the pressure coupling.

**com**

Scale the center of mass of the reference coordinates with the scaling matrix of the pressure coupling. The vectors of each reference coordinate to the center of mass are not scaled. Only one COM is used, even when there are multiple molecules with position restraints. For calculating the COM of the reference coordinates in the starting configuration, periodic boundary conditions are not taken into account. Note that with this option the virial and pressure might be ill defined, see *here* (page 500) for more details.

### Simulated annealing

Simulated annealing is controlled separately for each temperature group in GROMACS. The reference temperature is a piecewise linear function, but you can use an arbitrary number of points for each group, and choose either a single sequence or a periodic behaviour for each group. The actual annealing is performed by dynamically changing the reference temperature used in the thermostat algorithm selected, so remember that the system will usually not instantaneously reach the reference temperature!

**annealing**

Type of annealing for each temperature group

**no**

No simulated annealing - just couple to reference temperature value.

**single**

A single sequence of annealing points. If your simulation is longer than the time of the last point, the temperature will be coupled to this constant value after the annealing sequence has reached the last time point.

**periodic**

The annealing will start over at the first reference point once the last reference time is reached. This is repeated until the simulation ends.

**annealing-npoints**

A list with the number of annealing reference/control points used for each temperature group. Use 0 for groups that are not annealed. The number of entries should equal the number of temperature groups.

**annealing-time**

List of times at the annealing reference/control points for each group. If you are using periodic annealing, the times will be used modulo the last value, *i.e.* if the values are 0, 5, 10, and 15, the coupling will restart at the 0ps value after 15ps, 30ps, 45ps, etc. The number of entries should equal the sum of the numbers given in *annealing-npoints* (page 68).

**annealing-temp**

List of temperatures at the annealing reference/control points for each group. The number of entries should equal the sum of the numbers given in *annealing-npoints* (page 68).

Confused? OK, let's use an example. Assume you have two temperature groups, set the group selections to `annealing = single periodic`, the number of points of each group to `annealing-npoints = 3 4`, the times to `annealing-time = 0 3 6 0 2 4 6` and finally temperatures to `annealing-temp = 298 280 270 298 320 320 298`. The first group will be coupled to 298K at 0ps, but the reference temperature will drop linearly to reach 280K at 3ps, and then linearly between 280K and 270K from 3ps to 6ps. After this is stays constant, at 270K. The second group is coupled to 298K at 0ps, it increases linearly to 320K at 2ps, where it stays constant until 4ps. Between 4ps and 6ps it decreases to 298K, and then it starts over with the same pattern again, *i.e.* rising linearly from 298K to 320K between 6ps and 8ps. Check the summary printed by *gmx grompp* (page 230) if you are unsure!

### Velocity generation

**gen-vel**

**no**

Do not generate velocities. The velocities are set to zero when there are no velocities in the input structure file.

**yes**

Generate velocities in *gmx grompp* (page 230) according to a Maxwell distribution at temperature *gen-temp* (page 69), with random seed *gen-seed* (page 69). This is only meaningful with *integrator=md* (page 51).

**gen-temp**

(300) [K] temperature for Maxwell distribution

**gen-seed**

(-1) [integer] used to initialize random generator for random velocities, when *gen-seed* (page 69) is set to -1, a pseudo random seed is used.

### Bonds

**constraints**

Controls which bonds in the topology will be converted to rigid holonomic constraints. Note that typical rigid water models do not have bonds, but rather a specialized `[settles]` directive, so are not affected by this keyword.

**none**

No bonds converted to constraints.

**h-bonds**

Convert the bonds with H-atoms to constraints.

**all-bonds**

Convert all bonds to constraints. Note that many force fields have been parameterized with flexible bonds between heavy atoms and that constraining these bonds can increase energy barriers of dihedrals.

**h-angles**

Convert all bonds to constraints and convert the angles that involve H-atoms to bond-constraints. This option is deprecated.

**all-angles**

Convert all bonds to constraints and all angles to bond-constraints. This option is deprecated.

**constraint-algorithm**

Chooses which solver satisfies any non-SETTLE holonomic constraints.

**LINCS**

LINear Constraint Solver. With domain decomposition the parallel version P-LINCS is used. The accuracy in set with *lincs-order* (page 70), which sets the number of matrices in the expansion for the matrix inversion. After the matrix inversion correction the algorithm does an iterative correction to compensate for lengthening due to rotation. The number of such

iterations can be controlled with *lincs-iter* (page 70). The root mean square relative constraint deviation is printed to the log file every *nstlog* (page 56) steps. If a bond rotates more than *lincs-warnangle* (page 70) in one step, a warning will be printed both to the log file and to `stderr`. LINCS should not be used with coupled angle constraints.

**SHAKE**

SHAKE is slightly slower and less stable than LINCS, but does work with angle constraints. The relative tolerance is set with *shake-tol* (page 70), 0.0001 is a good value for "normal" MD. SHAKE does not support constraints between atoms on different decomposition domains, so it can only be used with domain decomposition when so-called update-groups are used, which is usually the case when only bonds involving hydrogens are constrained. SHAKE can not be used with energy minimization.

**continuation**

This option was formerly known as `unconstrained-start`.

**no**

apply constraints to the start configuration and reset shells

**yes**

do not apply constraints to the start configuration and do not reset shells, useful for exact continuation and reruns

**shake-tol**

(0.0001) relative tolerance for SHAKE

**lincs-order**

(4) Highest order in the expansion of the constraint coupling matrix. When constraints form triangles, an additional expansion of the same order is applied on top of the normal expansion only for the couplings within such triangles. For "normal" MD simulations an order of 4 usually suffices, 6 is needed for large time-steps with virtual sites or BD. For accurate energy minimization in double precision an order of 8 or more might be required. Note that in single precision an order higher than 6 will often lead to worse accuracy due to amplification of rounding errors. With domain decomposition, the cell size is limited by the distance spanned by *lincs-order* (page 70) +1 constraints. When one wants to scale further than this limit, one can decrease *lincs-order* (page 70) and increase *lincs-iter* (page 70), since the accuracy does not deteriorate when (1+ *lincs-iter* (page 70) )* *lincs-order* (page 70) remains constant.

**lincs-iter**

(1) Number of iterations to correct for rotational lengthening in LINCS. For normal runs a single step is sufficient, but for NVE runs where you want to conserve energy accurately or for accurate energy minimization in double precision you might want to increase it to 2. Note that in single precision using more than 1 iteration will often lead to worse accuracy due to amplification of rounding errors.

**lincs-warnangle**

(30) [deg] maximum angle that a bond can rotate before LINCS will complain

**morse**

**no**

bonds are represented by a harmonic potential

**yes**

bonds are represented by a Morse potential

### Energy group exclusions

**energygrp-excl**

Exclusion between pairs of energy groups are currently not supported.

### Walls

**nwall**

(0) When set to 1 there is a wall at `z=0`, when set to 2 there is also a wall at `z=z-box`. Walls can only be used with *pbc* (page 57) =`xy`. When set to 2, pressure coupling and Ewald summation can be used (it is usually best to use semiisotropic pressure coupling with the `x/y` compressibility set to 0, as otherwise the surface area will change). Walls interact wit the rest of the system through an optional *wall-atomtype* (page 71). Energy groups `wall0` and `wall1` (for *nwall* (page 71) =2) are added automatically to monitor the interaction of energy groups with each wall. The center of mass motion removal will be turned off in the `z`-direction.

**wall-atomtype**

the atom type name in the force field for each wall. By (for example) defining a special wall atom type in the topology with its own combination rules, this allows for independent tuning of the interaction of each atomtype with the walls.

**wall-type**

**9-3**

LJ integrated over the volume behind the wall: 9-3 potential

**10-4**

LJ integrated over the wall surface: 10-4 potential

**12-6**

direct LJ potential with the `z` distance from the wall

**table**

user-defined potentials indexed with the `z` distance from the wall, the tables are read analogously to the *energygrp-table* (page 62) option, where the first name is for a "normal" energy group and the second name is `wall0` or `wall1`, only the dispersion and repulsion columns are used

**wall-r-linpot**

(-1) [nm] Below this distance from the wall the potential is continued linearly and thus the force is constant. Setting this option to a postive value is especially useful for equilibration when some atoms are beyond a wall. When the value is <=0 (<0 for *wall-type* (page 71) =table), a fatal error is generated when atoms are beyond a wall.

**wall-density**

[$nm^{-3}$] / [$nm^{-2}$] the number density of the atoms for each wall for wall types 9-3 and 10-4

**wall-ewald-zfac**

(3) The scaling factor for the third box vector for Ewald summation only, the minimum is 2. Ewald summation can only be used with *nwall* (page 71) =2, where one should use *ewald-geometry* (page 63) =`3dc`. The empty layer in the box serves to decrease the unphysical Coulomb interaction between periodic images.

### COM pulling

Sets whether pulling on collective variables is active. Note that where pulling coordinates are applicable, there can be more than one (set with *pull-ncoords* (page 73)) and multiple related *mdp* (page 579) variables will exist accordingly. Documentation references to things like *pull-coord1-vec* (page 76) should be understood to apply to to the applicable pulling coordinate, eg. the second pull coordinate is described by pull-coord2-vec, pull-coord2-k, and so on.

**pull**

**no**

No center of mass pulling. All the following pull options will be ignored (and if present in the *mdp* (page 579) file, they unfortunately generate warnings)

**yes**

Center of mass pulling will be applied on 1 or more groups using 1 or more pull coordinates.

**pull-cylinder-r**

(1.5) [nm] the radius of the cylinder for *pull-coord1-geometry=cylinder* (page 75)

**pull-constr-tol**

($10^{-6}$) the relative constraint tolerance for constraint pulling

**pull-print-com**

**no**

do not print the COM for any group

**yes**

print the COM of all groups for all pull coordinates to the `pullx.xvg` file.

**pull-print-ref-value**

**no**

do not print the reference value for each pull coordinate

**yes**

print the reference value for each pull coordinate to the `pullx.xvg` file.

**pull-print-components**

**no**

only print the distance for each pull coordinate

**yes**

print the distance and Cartesian components selected in *pull-coord1-dim* (page 76) to the `pullx.xvg` file.

**pull-nstxout**

(50) interval for writing out the COMs of all the pull groups (0 is never) to the `pullx.xvg` file.

**pull-nstfout**

(50) interval for writing out the force of all the pulled groups (0 is never) to the `pullf.xvg` file.

**pull-pbc-ref-prev-step-com**

**no**

Use the reference atom (*pull-group1-pbcatom* (page 73)) for the treatment of periodic boundary conditions.

**yes**

Use the COM of the previous step as reference for the treatment of periodic boundary conditions. The reference is initialized using the reference atom (*pull-group1-pbcatom* (page 73)), which should be located centrally in the group. Using the COM from the previous step can be useful if one or more pull groups are large or very flexible.

**pull-xout-average**

**no**

Write the instantaneous coordinates for all the pulled groups.

**yes**

Write the average coordinates (since last output) for all the pulled groups. N.b., some analysis tools might expect instantaneous pull output.

**pull-fout-average**

**no**

Write the instantaneous force for all the pulled groups.

**yes**

Write the average force (since last output) for all the pulled groups. N.b., some analysis tools might expect instantaneous pull output.

**pull-ngroups**

(1) The number of pull groups, not including the absolute reference group, when used. Pull groups can be reused in multiple pull coordinates. Below only the pull options for group 1 are given, further groups simply increase the group index number.

**pull-ncoords**

(1) The number of pull coordinates. Below only the pull options for coordinate 1 are given, further coordinates simply increase the coordinate index number.

**pull-group1-name**

The name of the pull group, is looked up in the index file or in the default groups to obtain the atoms involved.

**pull-group1-weights**

Optional relative weights which are multiplied with the masses of the atoms to give the total weight for the COM. The number of weights should be 0, meaning all 1, or the number of atoms in the pull group.

**pull-group1-pbcatom**

(0) The reference atom for the treatment of periodic boundary conditions inside the group (this has no effect on the treatment of the pbc between groups). This option is only important when the diameter of the pull group is larger than half the shortest box vector. For determining the COM, all atoms in the group are put at their periodic image which is closest to *pull-group1-pbcatom* (page 73). A value of 0 means that the middle atom (number wise) is used, which is only safe for small groups. *gmx grompp* (page 230) checks that the maximum distance from the reference atom (specifically chosen, or not) to the other atoms in the group is not too large. This parameter is not used with *pull-coord1-geometry* (page 74) cylinder. A value of -1 turns on cosine

weighting, which is useful for a group of molecules in a periodic system, *e.g.* a water slab (see Engin et al. J. Chem. Phys. B 2010).

**pull-coord1-type**

**umbrella**

Center of mass pulling using an umbrella potential between the reference group and one or more groups.

**constraint**

Center of mass pulling using a constraint between the reference group and one or more groups. The setup is identical to the option umbrella, except for the fact that a rigid constraint is applied instead of a harmonic potential. Note that this type is not supported in combination with multiple time stepping.

**constant-force**

Center of mass pulling using a linear potential and therefore a constant force. For this option there is no reference position and therefore the parameters *pull-coord1-init* (page 76) and *pull-coord1-rate* (page 76) are not used.

**flat-bottom**

At distances above *pull-coord1-init* (page 76) a harmonic potential is applied, otherwise no potential is applied.

**flat-bottom-high**

At distances below *pull-coord1-init* (page 76) a harmonic potential is applied, otherwise no potential is applied.

**external-potential**

An external potential that needs to be provided by another module.

**pull-coord1-potential-provider**

The name of the external module that provides the potential for the case where *pull-coord1-type=external-potential* (page 74).

**pull-coord1-geometry**

**distance**

Pull along the vector connecting the two groups. Components can be selected with *pull-coord1-dim* (page 76).

**direction**

Pull in the direction of *pull-coord1-vec* (page 76).

**direction-periodic**

As *pull-coord1-geometry=direction* (page 74), but does not apply periodic box vector corrections to keep the distance within half the box length. This is (only) useful for pushing groups apart by more than half the box length by continuously changing the reference location using a pull rate. With this geometry the box should not be dynamic (*e.g.* no pressure scaling) in the pull dimensions and the pull force is not added to the virial.

**direction-relative**

As *pull-coord1-geometry=direction* (page 74), but the pull vector is the vector that points from the COM of a third to the COM of a fourth pull group. This means that 4 groups need to be supplied in *pull-coord1-groups* (page 76). Note that the pull force

will give rise to a torque on the pull vector, which is turn leads to forces perpendicular to the pull vector on the two groups defining the vector. If you want a pull group to move between the two groups defining the vector, simply use the union of these two groups as the reference group.

**cylinder**

Designed for pulling with respect to a layer where the reference COM is given by a local cylindrical part of the reference group. The pulling is in the direction of *pull-coord1-vec* (page 76). From the first of the two groups in *pull-coord1-groups* (page 76) a cylinder is selected around the axis going through the COM of the second group with direction *pull-coord1-vec* (page 76) with radius *pull-cylinder-r* (page 72). Weights of the atoms decrease continuously to zero as the radial distance goes from 0 to *pull-cylinder-r* (page 72) (mass weighting is also used). The radial dependence gives rise to radial forces on both pull groups. Note that the radius should be smaller than half the box size. For tilted cylinders they should be even smaller than half the box size since the distance of an atom in the reference group from the COM of the pull group has both a radial and an axial component. This geometry is not supported with constraint pulling.

**angle**

Pull along an angle defined by four groups. The angle is defined as the angle between two vectors: the vector connecting the COM of the first group to the COM of the second group and the vector connecting the COM of the third group to the COM of the fourth group.

**angle-axis**

As *pull-coord1-geometry=angle* (page 75) but the second vector is given by *pull-coord1-vec* (page 76). Thus, only the two groups that define the first vector need to be given.

**dihedral**

Pull along a dihedral angle defined by six groups. These pairwise define three vectors: the vector connecting the COM of group 1 to the COM of group 2, the COM of group 3 to the COM of group 4, and the COM of group 5 to the COM group 6. The dihedral angle is then defined as the angle between two planes: the plane spanned by the the two first vectors and the plane spanned the two last vectors.

**transformation**

Transforms other pull coordinates using a mathematical expression defined by *pull-coord1-expression* (page 75). Pull coordinates of lower indices, and time, can be used as variables to this pull coordinate. Thus, pull transformation coordinates should have a higher pull coordinate index than all pull coordinates they transform.

**pull-coord1-expression**

Mathematical expression to transform pull coordinates of lower indices to a new one. The pull coordinates are referred to as variables in the equation so that pull-coord1's value becomes 'x1', pull-coord2 value becomes 'x2' etc. Time can also be used a variable, becoming 't'. Note that angular coordinates use units of radians in the expression. The mathematical expression are evaluated using muParser. Only relevant if *pull-coord1-geometry=transformation* (page 75).

**pull-coord1-dx**

(1e-9) Size of finite difference to use in numerical derivation of the pull coordinate with respect to other pull coordinates. The current implementation uses a simple first order finite difference method to perform derivation so that f'(x) = (f(x+dx)-f(x))/dx Only relevant if *pull-coord1-geometry=transformation* (page 75).

**pull-coord1-groups**

The group indices on which this pull coordinate will operate. The number of group indices required is geometry dependent. The first index is the reference group and can be 0, in which case an absolute reference of *pull-coord1-origin* (page 76) is used. With an absolute reference the system is no longer translation invariant and one should think about what to do with the center of mass motion.

**pull-coord1-dim**

(Y Y Y) Selects the dimensions that this pull coordinate acts on and that are printed to the output files when *pull-print-components* (page 72) = *pull-coord1-start=yes* (page 76). With *pull-coord1-geometry* (page 74) = *pull-coord1-geometry=distance* (page 74), only Cartesian components set to Y contribute to the distance. Thus setting this to Y Y N results in a distance in the x/y plane. With other geometries all dimensions with non-zero entries in *pull-coord1-vec* (page 76) should be set to Y, the values for other dimensions only affect the output.

**pull-coord1-origin**

(0.0 0.0 0.0) The pull reference position for use with an absolute reference.

**pull-coord1-vec**

(0.0 0.0 0.0) The pull direction. *gmx grompp* (page 230) normalizes the vector.

**pull-coord1-start**

**no**

do not modify *pull-coord1-init* (page 76)

**yes**

add the COM distance of the starting conformation to *pull-coord1-init* (page 76)

**pull-coord1-init**

(0.0) [nm] or [deg] The reference distance or reference angle at t=0.

**pull-coord1-rate**

(0) [nm/ps] or [deg/ps] The rate of change of the reference position or reference angle.

**pull-coord1-k**

(0) [kJ mol$^{-1}$ nm$^{-2}$] or [kJ mol$^{-1}$ nm$^{-1}$] or [kJ mol$^{-1}$ rad$^{-2}$] or [kJ mol$^{-1}$ rad$^{-1}$] The force constant. For umbrella pulling this is the harmonic force constant in kJ mol$^{-1}$ nm$^{-2}$ (or kJ mol$^{-1}$ rad$^{-2}$ for angles). For constant force pulling this is the force constant of the linear potential, and thus the negative (!) of the constant force in kJ mol$^{-1}$ nm$^{-1}$ (or kJ mol$^{-1}$ rad$^{-1}$ for angles). Note that for angles the force constant is expressed in terms of radians (while *pull-coord1-init* (page 76) and *pull-coord1-rate* (page 76) are expressed in degrees).

**pull-coord1-kB**

(pull-k1) [kJ mol$^{-1}$ nm$^{-2}$] or [kJ mol$^{-1}$ nm$^{-1}$] or [kJ mol$^{-1}$ rad$^{-2}$] or [kJ mol$^{-1}$ rad$^{-1}$] As *pull-coord1-k* (page 76), but for state B. This is only used when *free-energy* (page 84) is turned on. The force constant is then (1 - lambda) * *pull-coord1-k* (page 76) + lambda * *pull-coord1-kB* (page 76).

### AWH adaptive biasing

**awh**

**no**

No biasing.

**yes**

Adaptively bias a reaction coordinate using the AWH method and estimate the corresponding PMF. This requires a constant ensemble temperature to be available. The PMF and other AWH data are written to the energy file (*edr* (page 576)) at an interval set by *awh-nstout* (page 78) and can be extracted with the `gmx awh` tool. The AWH coordinate can be multidimensional and is defined by mapping each dimension to a pull coordinate index (and/or up to one alchemical free lambda state dimension *free-energy* (page 84)). This is only allowed if *pull-coord1-type=external-potential* (page 74) and *pull-coord1-potential-provider* (page 74) = `awh` for the concerned pull coordinate indices. Pull geometry 'direction-periodic' and transformation coordinates that depend on time are not supported by AWH.

**awh-potential**

**convolved**

The applied biasing potential is the convolution of the bias function and a set of harmonic umbrella potentials (see *awh-potential=umbrella* (page 77) below). This results in a smooth potential function and force. The resolution of the potential is set by the force constant of each umbrella, see *awh1-dim1-force-constant* (page 81). This option is not compatible with using the free energy lambda state as an AWH reaction coordinate dimension.

**umbrella**

The potential bias is applied by controlling the position of a harmonic potential using Monte-Carlo sampling. The force constant is set with *awh1-dim1-force-constant* (page 81). The umbrella location is sampled using Monte-Carlo every *awh-nstsample* (page 78) steps. This is option is required when using the free energy lambda state as an AWH reaction coordinate dimension. Apart from that, this option is mainly for comparison and testing purposes as there are no advantages to using an umbrella.

**awh-share-multisim**

**no**

AWH will not share biases across simulations started with *gmx mdrun* (page 258) option `-multidir`. The biases will be independent.

**yes**

With *gmx mdrun* (page 258) and option `-multidir` the bias and PMF estimates for biases with *awh1-share-group* (page 80) >0 will be shared across simulations with the biases with the same *awh1-share-group* (page 80) value. The simulations should have the same AWH settings for sharing to make sense. *gmx mdrun* (page 258) will check whether the simulations are technically compatible for sharing, but the user should check that bias sharing physically makes sense.

**awh-seed**

(-1) Random seed for Monte-Carlo sampling the umbrella position, where -1 indicates to generate a seed. Only used with *awh-potential=umbrella* (page 77).

**awh-nstout**

(100000) Number of steps between printing AWH data to the energy file, should be a multiple of *nstenergy* (page 56).

**awh-nstsample**

(10) Number of steps between sampling of the coordinate value. This sampling is the basis for updating the bias and estimating the PMF and other AWH observables.

**awh-nsamples-update**

(100) The number of coordinate samples used for each AWH update. The update interval in steps is *awh-nstsample* (page 78) times this value.

**awh-nbias**

(1) The number of biases, each acting on its own coordinate. The following options should be specified for each bias although below only the options for bias number 1 is shown. Options for other bias indices are obtained by replacing '1' by the bias index.

**awh1-error-init**

(10.0) [kJ mol$^{-1}$] Estimated initial average error of the PMF for this bias. This value together with an estimate of the crossing time, based on the length of the sampling interval and the given diffusion constant(s) *awh1-dim1-diffusion* (page 81), determine the initial biasing rate. With multiple dimensions, the longest crossing time is used. The error is obviously not known *a priori*. Only a rough estimate of *awh1-error-init* (page 78) is needed however. As a general guideline, leave *awh1-error-init* (page 78) to its default value when starting a new simulation. On the other hand, when there is *a priori* knowledge of the PMF (e.g. when an initial PMF estimate is provided, see the *awh1-user-data* (page 79) option) then *awh1-error-init* (page 78) should reflect that knowledge.

**awh1-growth**

Each bias keeps a reference weight histogram for the coordinate samples. Its size sets the magnitude of the bias function and free energy estimate updates (few samples corresponds to large updates and vice versa). Thus, its growth rate sets the maximum convergence rate.

**exp-linear**

By default, there is an initial stage in which the histogram grows close to exponentially (but slower than the sampling rate). In the final stage that follows, the growth rate is linear and equal to the sampling rate (set by *awh-nstsample* (page 78)). The initial stage is typically necessary for efficient convergence when starting a new simulation where high free energy barriers have not yet been flattened by the bias.

**linear**

As *awh1-growth=exp-linear* (page 78) but skip the initial stage. This may be useful if there is *a priori* knowledge (see *awh1-error-init* (page 78)) which eliminates the need for an initial stage. This is also the setting compatible with *awh1-target=local-boltzmann* (page 79).

**awh1-growth-factor**

(2) [] The growth factor $\gamma$ during the exponential phase with *awh1-growth=exp-linear* (page 78). Should be larger than 1.

**awh1-equilibrate-histogram**

**yes**

Before entering the initial stage (see *awh1-growth=exp-linear* (page 78)), make sure the histogram of sampled weights is following the target distribution closely enough (specifically, at least 80% of the target region needs to have a local relative error of less than awh1-histogram-tolerance). This option is particularly important when *awh1-share-group* (page 80) > 0 and the initial configurations poorly represent the target distribution.

**no**

Do not equilibrate the histogram.

**awh1-histogram-tolerance**

(0.3) [] The relative tolerance for the histogram of sampled weigths, used with *awh1-equilibrate-histogram=yes* (page 78). The value of 0.3 is sufficiently large to not slow down the convergence, even when a single walker is used.

**awh1-target**

**constant**

The bias is tuned towards a constant (uniform) coordinate distribution in the defined sampling interval (defined by [*awh1-dim1-start* (page 81), *awh1-dim1-end* (page 81)]).

**cutoff**

Similar to *awh1-target=constant* (page 79), but the target distribution is proportional to 1/(1 + exp(F - *awh1-target=cutoff* (page 79))), where F is the free energy relative to the estimated global minimum. This provides a smooth switch of a flat target distribution in regions with free energy lower than the cut-off to a Boltzmann distribution in regions with free energy higher than the cut-off.

**boltzmann**

The target distribution is a Boltzmann distribtution with a scaled beta (inverse temperature) factor given by *awh1-target-beta-scaling* (page 79). *E.g.*, a value of 0.1 would give the same coordinate distribution as sampling with a simulation temperature scaled by 10.

**local-boltzmann**

Same target distribution and use of *awh1-target-beta-scaling* (page 79) but the convergence towards the target distribution is inherently local *i.e.*, the rate of change of the bias only depends on the local sampling. This local convergence property is only compatible with *awh1-growth=linear* (page 78), since for *awh1-growth=exp-linear* (page 78) histograms are globally rescaled in the initial stage.

**awh1-target-beta-scaling**

(0) For *awh1-target=boltzmann* (page 79) and *awh1-target=local-boltzmann* (page 79) it is the unitless beta scaling factor taking values in (0,1).

**awh1-target-cutoff**

(0) [kJ mol$^{-1}$] For *awh1-target=cutoff* (page 79) this is the cutoff, should be > 0.

**awh1-user-data**

**no**

Initialize the PMF and target distribution with default values.

**yes**

Initialize the PMF and target distribution with user provided data. For *awh-nbias* (page 78) = 1, *gmx mdrun* (page 258) will expect a file `awhinit.xvg` to be present in the run directory. For multiple biases, *gmx mdrun* (page 258) expects files `awhinit1.xvg`, `awhinit2.xvg`, etc. The file name can be changed with the `-awh` option. The first *awh1-ndim* (page 80) columns of each input file should contain the coordinate values, such that each row defines a point in coordinate space. Column *awh1-ndim* (page 80) + 1 should contain the PMF value (in kT) for each point. The target distribution column can either follow the PMF (column *awh1-ndim* (page 80) + 2) or be in the same column as written by *gmx awh* (page 160).

**awh1-share-group**

**0**

Do not share the bias.

**positive**

Share the bias and PMF estimates between simulations. This currently only works between biases with the same index. Note that currently sharing within a single simulation is not supported. The bias will be shared across simulations that specify the same value for *awh1-share-group* (page 80). To enable this, use *awh-share-multisim=yes* (page 77) and the *gmx mdrun* (page 258) option `-multidir`. Sharing may increase convergence initially, although the starting configurations can be critical, especially when sharing between many biases. N.b., multiple walkers sharing a degenerate reaction coordinate may have problems overlapping their sampling, possibly making it difficult to cover the sampling interval.

**awh1-target-metric-scaling**

**no**

Do not scale the target distribution based on the AWH friction metric.

**yes**

Scale the target distribution based on the AWH friction metric. Regions with high friction (long autocorrelation times) will be sampled more. The diffusion metric is the inverse of the friction metric. This scaling can be used with any *awh1-target* (page 79) type and is applied after user-provided target distribution modifications (*awh1-user-data* (page 79)), if any. If *awh1-growth=exp-linear* (page 78), the target distribution scaling starts after leaving the initial phase.

**awh1-target-metric-scaling-limit**

(10) The upper limit of scaling, relative to the average, when `awh1-target-metric-scaling` is enabled. The lower limit will be the inverse of this value. This upper limit should be > 1.

**awh1-ndim**

(1) [integer] Number of dimensions of the coordinate, each dimension maps to 1 pull coordinate. The following options should be specified for each such dimension. Below only the options for dimension number 1 is shown. Options for other dimension indices are obtained by replacing '1' by the dimension index.

**awh1-dim1-coord-provider**

**pull**

The pull module is providing the reaction coordinate for this dimension. With multiple time-stepping, AWH and pull should be in the same MTS level.

**fep-lambda**

The free energy *free-energy* (page 84) lambda state is the reaction coordinate for this dimension. The lambda states to use are specified by *fep-lambdas* (page 85), *vdw-lambdas* (page 85), *coul-lambdas* (page 85) etc. This is not compatible with delta-lambda. It also requires `calc-lambda-neighbors=-1`. With multiple time-stepping, AWH should be in the slow level. This option requires *awh-potential=umbrella* (page 77).

**awh1-dim1-coord-index**

(1) Index of the pull coordinate defining this coordinate dimension.

**awh1-dim1-force-constant**

(0) [kJ $mol^{-1}$ $nm^{-2}$] or [kJ $mol^{-1}$ $rad^{-2}$] Force constant for the (convolved) umbrella potential(s) along this coordinate dimension.

**awh1-dim1-start**

(0.0) [nm] or [deg] Start value of the sampling interval along this dimension. The range of allowed values depends on the relevant pull geometry (see *pull-coord1-geometry* (page 74)). For dihedral geometries *awh1-dim1-start* (page 81) greater than *awh1-dim1-end* (page 81) is allowed. The interval will then wrap around from +period/2 to -period/2. For the direction geometry, the dimension is made periodic when the direction is along a box vector and covers more than 95% of the box length. Note that one should not apply pressure coupling along a periodic dimension.

**awh1-dim1-end**

(0.0) [nm] or [deg] End value defining the sampling interval together with *awh1-dim1-start* (page 81).

**awh1-dim1-diffusion**

($10^{-5}$) [$nm^2/ps$], [$rad^2/ps$] or [$ps^{-1}$] Estimated diffusion constant for this coordinate dimension determining the initial biasing rate. This needs only be a rough estimate and should not critically affect the results unless it is set to something very low, leading to slow convergence, or very high, forcing the system far from equilibrium. Not setting this value explicitly generates a warning.

**awh1-dim1-cover-diameter**

(0.0) [nm] or [deg] Diameter that needs to be sampled by a single simulation around a coordinate value before the point is considered covered in the initial stage (see *awh1-growth=exp-linear* (page 78)). A value > 0 ensures that for each covering there is a continuous transition of this diameter across each coordinate value. This is trivially true for independent simulations but not for for multiple bias-sharing simulations (*awh1-share-group* (page 80)>0). For a diameter = 0, covering occurs as soon as the simulations have sampled the whole interval, which for many sharing simulations does not guarantee transitions across free energy barriers. On the other hand, when the diameter >= the sampling interval length, covering occurs when a single simulation has independently sampled the whole interval.

### Enforced rotation

These *mdp* (page 579) parameters can be used enforce the rotation of a group of atoms, e.g. a protein subunit. The reference manual describes in detail 13 different potentials that can be used to achieve such a rotation.

**rotation**

**no**

No enforced rotation will be applied. All enforced rotation options will be ignored (and if present in the *mdp* (page 579) file, they unfortunately generate warnings).

**yes**

Apply the rotation potential specified by *rot-type0* (page 82) to the group of atoms given under the *rot-group0* (page 82) option.

**rot-ngroups**

(1) Number of rotation groups.

**rot-group0**

Name of rotation group 0 in the index file.

**rot-type0**

(iso) Type of rotation potential that is applied to rotation group 0. Can be of of the following: `iso`, `iso-pf`, `pm`, `pm-pf`, `rm`, `rm-pf`, `rm2`, `rm2-pf`, `flex`, `flex-t`, `flex2`, or `flex2-t`.

**rot-massw0**

(no) Use mass weighted rotation group positions.

**rot-vec0**

(1.0 0.0 0.0) Rotation vector, will get normalized.

**rot-pivot0**

(0.0 0.0 0.0) [nm] Pivot point for the potentials `iso`, `pm`, `rm`, and `rm2`.

**rot-rate0**

(0) [degree $ps^{-1}$] Reference rotation rate of group 0.

**rot-k0**

(0) [kJ $mol^{-1}$ $nm^{-2}$] Force constant for group 0.

**rot-slab-dist0**

(1.5) [nm] Slab distance, if a flexible axis rotation type was chosen.

**rot-min-gauss0**

(0.001) Minimum value (cutoff) of Gaussian function for the force to be evaluated (for the flexible axis potentials).

**rot-eps0**

(0.0001) [$nm^2$] Value of additive constant epsilon for `rm2*` and `flex2*` potentials.

**rot-fit-method0**

(rmsd) Fitting method when determining the actual angle of a rotation group (can be one of `rmsd`, `norm`, or `potential`).

**rot-potfit-nsteps0**

(21) For fit type `potential`, the number of angular positions around the reference angle for which the rotation potential is evaluated.

**rot-potfit-step0**

(0.25) For fit type `potential`, the distance in degrees between two angular positions.

**rot-nstrout**

(100) Output interval (in steps) for the angle of the rotation group, as well as for the torque and the rotation potential energy.

**rot-nstsout**

(1000) Output interval (in steps) for per-slab data of the flexible axis potentials, i.e. angles, torques and slab centers.

### NMR refinement

**disre**

**no**

ignore distance restraint information in topology file

**simple**

simple (per-molecule) distance restraints.

**ensemble**

distance restraints over an ensemble of molecules in one simulation box. Normally, one would perform ensemble averaging over multiple simulations, using `mdrun -multidir`. The environment variable `GMX_DISRE_ENSEMBLE_SIZE` sets the number of systems within each ensemble (usually equal to the number of directories supplied to `mdrun -multidir`).

**disre-weighting**

**equal**

divide the restraint force equally over all atom pairs in the restraint

**conservative**

the forces are the derivative of the restraint potential, this results in an weighting of the atom pairs to the reciprocal seventh power of the displacement. The forces are conservative when *disre-tau* (page 83) is zero.

**disre-mixed**

**no**

the violation used in the calculation of the restraint force is the time-averaged violation

**yes**

the violation used in the calculation of the restraint force is the square root of the product of the time-averaged violation and the instantaneous violation

**disre-fc**

(1000) [kJ mol$^{-1}$ nm$^{-2}$] force constant for distance restraints, which is multiplied by a (possibly) different factor for each restraint given in the `fac` column of the interaction in the topology file.

**disre-tau**

(0) [ps] time constant for distance restraints running average. A value of zero turns off time averaging.

**nstdisreout**

(100) [steps] period between steps when the running time-averaged and instantaneous distances of all atom pairs involved in restraints are written to the energy file (can make the energy file very large)

**orire**

**no**

ignore orientation restraint information in topology file

**yes**

use orientation restraints, ensemble averaging can be performed with `mdrun -multidir`

**orire-fc**

(0) [kJ mol$^{-1}$] force constant for orientation restraints, which is multiplied by a (possibly) different weight factor for each restraint, can be set to zero to obtain the orientations from a free simulation

**orire-tau**

(0) [ps] time constant for orientation restraints running average. A value of zero turns off time averaging.

**orire-fitgrp**

fit group for orientation restraining. This group of atoms is used to determine the rotation **R** of the system with respect to the reference orientation. The reference orientation is the starting conformation of the first subsystem. For a protein, backbone is a reasonable choice

**nstorireout**

(100) [steps] period between steps when the running time-averaged and instantaneous orientations for all restraints, and the molecular order tensor are written to the energy file (can make the energy file very large)

### Free energy calculations

**free-energy**

**no**

Only use topology A.

**yes**

Interpolate between topology A (lambda=0) to topology B (lambda=1) and write the derivative of the Hamiltonian with respect to lambda (as specified with *dhdl-derivatives* (page 88)), or the Hamiltonian differences with respect to other "foreign" lambda values (as specified with *calc-lambda-neighbors* (page 86)) to the energy file and/or to `dhdl.xvg`, where they can be processed by, for example *gmx bar* (page 161). The potentials, bond-lengths and angles are interpolated linearly as described in the manual. When *sc-alpha* (page 86) is larger than zero, soft-core potentials are used for the LJ and Coulomb interactions.

**expanded**

Turns on expanded ensemble simulation, where the alchemical state becomes a dynamic variable, allowing jumping between different Hamiltonians. See the expanded ensemble options for controlling how expanded ensemble simulations are performed. The different Hamiltonians used in expanded ensemble simulations are defined by the other free energy options.

**init-lambda**

(-1) starting value for lambda (float). Generally, this should only be used with slow growth (*i.e.* nonzero *delta-lambda* (page 85)). In other cases, *init-lambda-state* (page 85) should be specified instead. If a lambda vector is given, *init-lambda* (page 85) is used to interpolate the vector instead of setting lambda directly. Must be greater than or equal to 0.

**delta-lambda**

(0) increment per time step for lambda

**init-lambda-state**

(-1) starting value for the lambda state (integer). Specifies which column of the lambda vector (*coul-lambdas* (page 85), *vdw-lambdas* (page 85), *bonded-lambdas* (page 85), *restraint-lambdas* (page 85), *mass-lambdas* (page 86), *temperature-lambdas* (page 86), *fep-lambdas* (page 85)) should be used. This is a zero-based index: `init-lambda-state=0` means the first column, and so on.

**fep-lambdas**

[array] Zero, one or more lambda values for which Delta H values will be determined and written to dhdl.xvg every *nstdhdl* (page 88) steps. Values must be greater than or equal to 0; values greater than 1 are allowed but should be used carefully. Free energy differences between different lambda values can then be determined with *gmx bar* (page 161). *fep-lambdas* (page 85) is different from the other -lambdas keywords because all components of the lambda vector that are not specified will use *fep-lambdas* (page 85).

**coul-lambdas**

[array] Zero, one or more lambda values for which Delta H values will be determined and written to dhdl.xvg every *nstdhdl* (page 88) steps. Values must be greater than or equal to 0; values greater than 1 are allowed but should be used carefully. If soft-core potentials are used, values must be between 0 and 1. Only the electrostatic interactions are controlled with this component of the lambda vector (and only if the lambda=0 and lambda=1 states have differing electrostatic interactions).

**vdw-lambdas**

[array] Zero, one or more lambda values for which Delta H values will be determined and written to dhdl.xvg every *nstdhdl* (page 88) steps. Values must be greater than or equal to 0; values greater than 1 are allowed but should be used carefully. If soft-core potentials are used, values must be between 0 and 1. Only the van der Waals interactions are controlled with this component of the lambda vector.

**bonded-lambdas**

[array] Zero, one or more lambda values for which Delta H values will be determined and written to dhdl.xvg every *nstdhdl* (page 88) steps. Values must be greater than or equal to 0; values greater than 1 are allowed but should be used carefully. Only the bonded interactions are controlled with this component of the lambda vector.

**restraint-lambdas**

[array] Zero, one or more lambda values for which Delta H values will be determined and written to dhdl.xvg every *nstdhdl* (page 88) steps. Values must be greater than or equal to 0; values greater than 1 are allowed but should be used carefully. Only the restraint interactions: dihedral restraints, and the pull code restraints are controlled with this component of the lambda vector.

**mass-lambdas**

[array] Zero, one or more lambda values for which Delta H values will be determined and written to dhdl.xvg every *nstdhdl* (page 88) steps. Values must be greater than or equal to 0; values greater than 1 are allowed but should be used carefully. Only the particle masses are controlled with this component of the lambda vector.

**temperature-lambdas**

[array] Zero, one or more lambda values for which Delta H values will be determined and written to dhdl.xvg every *nstdhdl* (page 88) steps. Values must be greater than or equal to 0; values greater than 1 are allowed but should be used carefully. Only the temperatures are controlled with this component of the lambda vector. Note that these lambdas should not be used for replica exchange, only for simulated tempering.

**calc-lambda-neighbors**

(1) Controls the number of lambda values for which Delta H values will be calculated and written out, if *init-lambda-state* (page 85) has been set. These lambda values are referred to as "foreign" lambdas. A positive value will limit the number of lambda points calculated to only the nth neighbors of *init-lambda-state* (page 85): for example, if *init-lambda-state* (page 85) is 5 and this parameter has a value of 2, energies for lambda points 3-7 will be calculated and writen out. A value of -1 means all lambda points will be written out. For normal BAR such as with *gmx bar* (page 161), a value of 1 is sufficient, while for MBAR -1 should be used.

**sc-function**

(beutler)

**beutler**

Beutler *et al.* soft-core function

**gapsys**

Gapsys *et al.* soft-core function

**sc-alpha**

(0) for *sc-function=beutler* (page 86) the soft-core alpha parameter, a value of 0 results in linear interpolation of the LJ and Coulomb interactions. Used only with *sc-function=beutler* (page 86)

**sc-r-power**

(6) power 6 for the radial term in the soft-core equation. Used only with *sc-function=beutler* (page 86)

**sc-coul**

(no) Whether to apply the soft-core free energy interaction transformation to the Coulombic interaction of a molecule. Default is no, as it is generally more efficient to turn off the Coulombic interactions linearly before turning off the van der Waals interactions. Note that it is only taken into account when there are multiple lambda components, and you can still turn off soft-core interactions by setting *sc-alpha* (page 86) to 0. Used only with *sc-function=beutler* (page 86)

**sc-power**

(1) the power for lambda in the soft-core function, only the values 1 and 2 are supported. Used only with *sc-function=beutler* (page 86)

**sc-sigma**

(0.3) [nm] for *sc-function=beutler* (page 86) the soft-core sigma for particles which have a C6 or C12 parameter equal to zero or a sigma smaller than *sc-sigma* (page 87). Used only with *sc-function=beutler* (page 86)

**sc-gapsys-scale-linpoint-lj**

(0.85) for *sc-function=gapsys* (page 86) it is the unitless alphaLJ parameter. It controls the softness of the van der Waals interactions by scaling the point for linearizing the vdw force. Setting it to 0 will result in the standard hard-core van der Waals interactions. Used only with *sc-function=gapsys* (page 86)

**sc-gapsys-scale-linpoint-q**

(0.3) [nm/e^2] For *sc-function=gapsys* (page 86) the alphaQ parameter with a default value of 0.3. It controls the softness of the Coulombic interactions. Setting it to 0 will result in the standard hard-core Coulombic interactions. Used only with *sc-function=gapsys* (page 86)

**sc-gapsys-sigma-lj**

(0.3) [nm] for *sc-function=gapsys* (page 86) the soft-core sigma for particles which have a C6 or C12 parameter equal to zero. Used only with *sc-function=gapsys* (page 86)

**couple-moltype**

Here one can supply a molecule type (as defined in the topology) for calculating solvation or coupling free energies. There is a special option `system` that couples all molecule types in the system. This can be useful for equilibrating a system starting from (nearly) random coordinates. *free-energy* (page 84) has to be turned on. The Van der Waals interactions and/or charges in this molecule type can be turned on or off between lambda=0 and lambda=1, depending on the settings of *couple-lambda0* (page 87) and *couple-lambda1* (page 87). If you want to decouple one of several copies of a molecule, you need to copy and rename the molecule definition in the topology.

**couple-lambda0**

**vdw-q**

all interactions are on at lambda=0

**vdw**

the charges are zero (no Coulomb interactions) at lambda=0

**q**

the Van der Waals interactions are turned off at lambda=0; soft-core interactions will be required to avoid singularities

**none**

the Van der Waals interactions are turned off and the charges are zero at lambda=0; soft-core interactions will be required to avoid singularities.

**couple-lambda1**

analogous to *couple-lambda0* (page 87), but for lambda=1

**couple-intramol**

**no**

All intra-molecular non-bonded interactions for moleculetype *couple-moltype* (page 87) are replaced by exclusions and explicit pair interactions. In this manner the decoupled state of the molecule corresponds to the proper vacuum state without periodicity effects.

**yes**

The intra-molecular Van der Waals and Coulomb interactions are also turned on/off. This can be useful for partitioning free-energies of relatively large molecules, where the intramolecular non-bonded interactions might lead to kinetically trapped vacuum conformations. The 1-4 pair interactions are not turned off.

**nstdhdl**

(100) the interval for writing dH/dlambda and possibly Delta H to dhdl.xvg, 0 means no ouput, should be a multiple of *nstcalcenergy* (page 56).

**dhdl-derivatives**

(yes)

If yes (the default), the derivatives of the Hamiltonian with respect to lambda at each *nstdhdl* (page 88) step are written out. These values are needed for interpolation of linear energy differences with *gmx bar* (page 161) (although the same can also be achieved with the right *calc-lambda-neighbors* (page 86) setting, that may not be as flexible), or with thermodynamic integration

**dhdl-print-energy**

(no)

Include either the total or the potential energy in the dhdl file. Options are 'no', 'potential', or 'total'. This information is needed for later free energy analysis if the states of interest are at different temperatures. If all states are at the same temperature, this information is not needed. 'potential' is useful in case one is using `mdrun -rerun` to generate the `dhdl.xvg` file. When rerunning from an existing trajectory, the kinetic energy will often not be correct, and thus one must compute the residual free energy from the potential alone, with the kinetic energy component computed analytically.

**separate-dhdl-file**

**yes**

The free energy values that are calculated (as specified with *calc-lambda-neighbors* (page 86) and *dhdl-derivatives* (page 88) settings) are written out to a separate file, with the default name `dhdl.xvg`. This file can be used directly with *gmx bar* (page 161).

**no**

The free energy values are written out to the energy output file (`ener.edr`, in accumulated blocks at every *nstenergy* (page 56) steps), where they can be extracted with *gmx energy* (page 214) or used directly with *gmx bar* (page 161).

**dh-hist-size**

(0) If nonzero, specifies the size of the histogram into which the Delta H values (specified with *calc-lambda-neighbors* (page 86)) and the derivative dH/dl values are binned, and written to ener.edr. This can be used to save disk space while calculating free energy differences. One histogram gets written for each foreign lambda and two for the dH/dl, at every *nstenergy* (page 56) step. Be aware that incorrect histogram settings (too small size or too wide bins) can introduce errors. Do not use histograms unless you are certain you need it.

**dh-hist-spacing**

(0.1) Specifies the bin width of the histograms, in energy units. Used in conjunction with *dh-hist-size* (page 88). This size limits the accuracy with which free energies can be calculated. Do not use histograms unless you are certain you need it.

### Expanded Ensemble calculations

**nstexpanded**

The number of integration steps beween attempted moves changing the system Hamiltonian in expanded ensemble simulations. Must be a multiple of *nstcalcenergy* (page 56), but can be greater or less than *nstdhdl* (page 88).

**lmc-stats**

**no**

No Monte Carlo in state space is performed.

**metropolis-transition**

Uses the Metropolis weights to update the expanded ensemble weight of each state. Min{1,exp(-(beta_new u_new - beta_old u_old)}

**barker-transition**

Uses the Barker transition critera to update the expanded ensemble weight of each state i, defined by exp(-beta_new u_new)/(exp(-beta_new u_new)+exp(-beta_old u_old))

**wang-landau**

Uses the Wang-Landau algorithm (in state space, not energy space) to update the expanded ensemble weights.

**min-variance**

Uses the minimum variance updating method of Escobedo et al. to update the expanded ensemble weights. Weights will not be the free energies, but will rather emphasize states that need more sampling to give even uncertainty.

**lmc-mc-move**

**no**

No Monte Carlo in state space is performed.

**metropolis-transition**

Randomly chooses a new state up or down, then uses the Metropolis criteria to decide whether to accept or reject: Min{1,exp(-(beta_new u_new - beta_old u_old)}

**barker-transition**

Randomly chooses a new state up or down, then uses the Barker transition criteria to decide whether to accept or reject: exp(-beta_new u_new)/(exp(-beta_new u_new)+exp(-beta_old u_old))

**gibbs**

Uses the conditional weights of the state given the coordinate (exp(-beta_i u_i) / sum_k exp(beta_i u_i) to decide which state to move to.

**metropolized-gibbs**

Uses the conditional weights of the state given the coordinate (exp(-beta_i u_i) / sum_k exp(beta_i u_i) to decide which state to move to, EXCLUDING the current state, then uses a rejection step to ensure detailed balance. Always more efficient that Gibbs, though only marginally so in many situations, such as when only the nearest neighbors have decent phase space overlap.

**lmc-seed**

(-1) random seed to use for Monte Carlo moves in state space. When *lmc-seed* (page 90) is set to -1, a pseudo random seed is us

**mc-temperature**

Temperature used for acceptance/rejection for Monte Carlo moves. If not specified, the temperature of the simulation specified in the first group of *ref-t* (page 65) is used.

**wl-ratio**

(0.8) The cutoff for the histogram of state occupancies to be reset, and the free energy incrementor to be changed from delta to delta * *wl-scale* (page 90). If we define the Nratio = (number of samples at each histogram) / (average number of samples at each histogram). *wl-ratio* (page 90) of 0.8 means that means that the histogram is only considered flat if all Nratio > 0.8 AND simultaneously all 1/Nratio > 0.8.

**wl-scale**

(0.8) Each time the histogram is considered flat, then the current value of the Wang-Landau incrementor for the free energies is multiplied by *wl-scale* (page 90). Value must be between 0 and 1.

**init-wl-delta**

(1.0) The initial value of the Wang-Landau incrementor in kT. Some value near 1 kT is usually most efficient, though sometimes a value of 2-3 in units of kT works better if the free energy differences are large.

**wl-oneovert**

(no) Set Wang-Landau incrementor to scale with 1/(simulation time) in the large sample limit. There is significant evidence that the standard Wang-Landau algorithms in state space presented here result in free energies getting 'burned in' to incorrect values that depend on the initial state. when *wl-oneovert* (page 90) is true, then when the incrementor becomes less than 1/N, where N is the number of samples collected (and thus proportional to the data collection time, hence '1 over t'), then the Wang-Lambda incrementor is set to 1/N, decreasing every step. Once this occurs, *wl-ratio* (page 90) is ignored, but the weights will still stop updating when the equilibration criteria set in *lmc-weights-equil* (page 92) is achieved.

**lmc-repeats**

(1) Controls the number of times that each Monte Carlo swap type is performed each iteration. In the limit of large numbers of Monte Carlo repeats, then all methods converge to Gibbs sampling. The value will generally not need to be different from 1.

**lmc-gibbsdelta**

(-1) Limit Gibbs sampling to selected numbers of neighboring states. For Gibbs sampling, it is sometimes inefficient to perform Gibbs sampling over all of the states that are defined. A positive value of *lmc-gibbsdelta* (page 90) means that only states plus or minus *lmc-gibbsdelta* (page 90) are considered in exchanges up and down. A value of -1 means that all states are considered. For less than 100 states, it is probably not that expensive to include all states.

**lmc-forced-nstart**

(0) Force initial state space sampling to generate weights. In order to come up with reasonable initial weights, this setting allows the simulation to drive from the initial to the final lambda state, with *lmc-forced-nstart* (page 90) steps at each state before moving on to the next lambda state. If *lmc-forced-nstart* (page 90) is sufficiently long (thousands of steps, perhaps), then the weights will be close to correct. However, in most cases, it is probably better to simply run the standard weight equilibration algorithms.

**nst-transition-matrix**

(-1) Interval of outputting the expanded ensemble transition matrix. A negative number means it will only be printed at the end of the simulation.

**symmetrized-transition-matrix**

(no) Whether to symmetrize the empirical transition matrix. In the infinite limit the matrix will be symmetric, but will diverge with statistical noise for short timescales. Forced symmetrization, by using the matrix T_sym = 1/2 (T + transpose(T)), removes problems like the existence of (small magnitude) negative eigenvalues.

**mininum-var-min**

(100) The min-variance strategy (option of *lmc-stats* (page 89) is only valid for larger number of samples, and can get stuck if too few samples are used at each state. *mininum-var-min* (page 91) is the minimum number of samples that each state that are allowed before the min-variance strategy is activated if selected.

**init-lambda-weights**

The initial weights (free energies) used for the expanded ensemble states. Default is a vector of zero weights. format is similar to the lambda vector settings in *fep-lambdas* (page 85), except the weights can be any floating point number. Units are kT. Its length must match the lambda vector lengths.

**init-wl-histogram-counts**

The initial counts used for the Wang-Landau histogram of visiting expanded ensemble states. The flatness of this histogram is used to decide whether to decrement the histogram-building incrementor. This option is only generally useful if continuing a shorter simulation from a previous one, as the smaller the incrementor gets, the longer it takes for the histogram to become flat, often longer than a short simulation takes, requiring the histogram population to be carried over from the previous simulation. The default is a vector of zeros. The format is similar to the lambda vector settings in *fep-lambdas* (page 85). The value can be a floating point number or an integer, as some methods increment multiple histogram bins at the same time with fractional weights. Its length must match the lambda vector lengths.

**init-lambda-counts**

The initial counts used for the number of times each expanded ensemble state is visited states. Several algorithms set by *lmc-weights-equil* (page 92) use various functions of the number of visits to each state states to decide whether to switch to different phases of weight determination. These include `number-all-lambda` which requires the mumber of times each lambda state is visited to be equal to or greater than this number, `number-samples`, which requires the total number of visits to all lambda states to be greater than or equal to this, and `count-ratio`, which requires the number of states visited at each state to be within a given ratio of equal visitation. This option is only generally useful if continuing a shorter simulation from a previous one, as most methods will reach the triggering conditions with relatively low number of samples collected. The default is a vector of zeros. The format is similar to the lambda vector settings in

*fep-lambdas* (page 85). Unlike `init-wl-histogram`, the value can only be an integer. Its length must match the lambda vector lengths.

**lmc-weights-equil**

**no**

Expanded ensemble weights continue to be updated throughout the simulation.

**yes**

The input expanded ensemble weights are treated as equilibrated, and are not updated throughout the simulation.

**wl-delta**

Expanded ensemble weight updating is stopped when the Wang-Landau incrementor falls below this value.

**number-all-lambda**

Expanded ensemble weight updating is stopped when the number of samples at all of the lambda states is greater than this value.

**number-steps**

Expanded ensemble weight updating is stopped when the number of steps is greater than the level specified by this value.

**number-samples**

Expanded ensemble weight updating is stopped when the number of total samples across all lambda states is greater than the level specified by this value.

**count-ratio**

Expanded ensemble weight updating is stopped when the ratio of samples at the least sampled lambda state and most sampled lambda state greater than this value.

**simulated-tempering**

(no) Turn simulated tempering on or off. Simulated tempering is implemented as expanded ensemble sampling with different temperatures instead of different Hamiltonians.

**sim-temp-low**

(300) [K] Low temperature for simulated tempering.

**sim-temp-high**

(300) [K] High temperature for simulated tempering.

**simulated-tempering-scaling**

Controls the way that the temperatures at intermediate lambdas are calculated from the *temperature-lambdas* (page 86) part of the lambda vector.

**linear**

Linearly interpolates the temperatures using the values of *temperature-lambdas* (page 86), *i.e.* if *sim-temp-low* (page 92) =300, *sim-temp-high* (page 92) =400, then lambda=0.5 correspond to a temperature of 350. A nonlinear set of temperatures can always be implemented with uneven spacing in lambda.

**geometric**

Interpolates temperatures geometrically between *sim-temp-low* (page 92) and *sim-temp-high* (page 92). The i:th state has temperature *sim-temp-low* (page 92)

* (*sim-temp-high* (page 92) / *sim-temp-low* (page 92)) raised to the power of (i/(ntemps-1)). This should give roughly equal exchange for constant heat capacity, though of course things simulations that involve protein folding have very high heat capacity peaks.

**exponential**

Interpolates temperatures exponentially between *sim-temp-low* (page 92) and *sim-temp-high* (page 92). The i:th state has temperature *sim-temp-low* (page 92) + (*sim-temp-high* (page 92) - *sim-temp-low* (page 92))*((exp(*temperature-lambdas* (page 86) (i))-1)/(exp(1.0)-i)).

### Non-equilibrium MD

**acc-grps**

groups for constant acceleration (*e.g.* `Protein Sol`) all atoms in groups Protein and Sol will experience constant acceleration as specified in the *accelerate* (page 93) line. Note that the kinetic energy of the center of mass of accelerated groups contributes to the kinetic energy and temperature of the system. If this is not desired, make each accelerate group also a separate temperature coupling group.

**accelerate**

(0) [nm ps$^{-2}$] acceleration for *acc-grps* (page 93); x, y and z for each group (*e.g.* `0.1 0.0 0.0 -0.1 0.0 0.0` means that first group has constant acceleration of 0.1 nm ps$^{-2}$ in X direction, second group the opposite).

**freezegrps**

Groups that are to be frozen (*i.e.* their X, Y, and/or Z position will not be updated; *e.g.* `Lipid SOL`). *freezedim* (page 93) specifies for which dimension(s) the freezing applies. Note that the virial and pressure are usually not meaningful when frozen atoms are present. Note that coordinates of frozen atoms are not scaled by pressure-coupling algorithms.

**freezedim**

dimensions for which groups in *freezegrps* (page 93) should be frozen, specify `Y` or `N` for X, Y and Z and for each group (*e.g.* `Y Y N N N N` means that particles in the first group can move only in Z direction. The particles in the second group can move in any direction).

**cos-acceleration**

(0) [nm ps$^{-2}$] the amplitude of the acceleration profile for calculating the viscosity. The acceleration is in the X-direction and the magnitude is *cos-acceleration* (page 93) cos(2 pi z/boxheight). Two terms are added to the energy file: the amplitude of the velocity profile and 1/viscosity.

**deform**

(0 0 0 0 0 0) [nm ps$^{-1}$] The velocities of deformation for the box elements: a(x) b(y) c(z) b(x) c(x) c(y). Each step the box elements for which *deform* (page 93) is non-zero are calculated as: box(ts)+(t-ts)*deform, off-diagonal elements are corrected for periodicity. The time ts is set to t at the first step and at steps at which x and v are written to trajectory to ensure exact restarts. Deformation can be used together with semiisotropic or anisotropic pressure coupling when the appropriate compressibilities are set to zero. The diagonal elements can be used to strain a solid. The off-diagonal elements can be used to shear a solid or a liquid. Note that the atom positions are not affected directly by this option. Instead, the deform option only modifies the velocities of particles that are shifted by a periodic box vector such that their new velocities match the virtual velocity flow field corresponding to the box deformation. As the deform option never accelerates

the remaining particles in the system, the matching velocity flow field should be set up at the beginning of the simulation to make the particles follow the deformation. This can be done with the *deform-init-flow* (page 94) option. The flow field is removed from the kinetic energy by *gmx mdrun* (page 258) so the actual temperature and pressure of the system are reported.

**deform-init-flow**

**no**

Do not modify the velocities. Only use this option when the velocities of the atoms in the initial configuration already obey the flow field.

**yes**

When the *deform* (page 93) option is active, add a velocity profile corresponding to the box deformation to the initial velocities. This is done after computing observables from the initial state such as the initial temperature.

### Electric fields

**electric-field-x**

**electric-field-y**

**electric-field-z**

Here you can specify an electric field that optionally can be alternating and pulsed. The general expression for the field has the form of a gaussian laser pulse:

$$E(t) = E_0 \exp\left[-\frac{(t-t_0)^2}{2\sigma^2}\right] \cos\left[\omega(t-t_0)\right]$$

For example, the four parameters for direction x are set in the fields of *electric-field-x* (page 94) (and similar for `electric-field-y` and `electric-field-z`) like

```
electric-field-x = E0 omega t0 sigma
```

with units (respectively) V nm$^{-1}$, ps$^{-1}$, ps, ps.

In the special case that `sigma = 0`, the exponential term is omitted and only the cosine term is used. In this case, `t0` must be set to 0. If also `omega = 0` a static electric field is applied.

Read more at *Electric fields* (page 619) and in ref. *146* (page 690).

### Mixed quantum/classical molecular dynamics

**QMMM-grps**

groups to be described at the QM level for MiMiC QM/MM

**QMMM**

**no**

QM/MM is no longer supported via these .mdp options. For MiMic, use no here.

### Computational Electrophysiology

Use these options to switch on and control ion/water position exchanges in "Computational Electrophysiology" simulation setups. (See the reference manual for details).

**swapcoords**

**no**

Do not enable ion/water position exchanges.

**X ; Y ; Z**

Allow for ion/water position exchanges along the chosen direction. In a typical setup with the membranes parallel to the x-y plane, ion/water pairs need to be exchanged in Z direction to sustain the requested ion concentrations in the compartments.

**swap-frequency**

(1) The swap attempt frequency, i.e. every how many time steps the ion counts per compartment are determined and exchanges made if necessary. Normally, it is not necessary to check at every time step. For typical Computational Electrophysiology setups, a value of about 100 is sufficient and yields a negligible performance impact.

**split-group0**

Name of the index group of the membrane-embedded part of channel #0. The center of mass of these atoms defines one of the compartment boundaries and should be chosen such that it is near the center of the membrane.

**split-group1**

Defines the position of the other compartment boundary.

**massw-split0**

(no) Defines whether or not mass-weighting is used to calculate the split group center.

**no**

Use the geometrical center.

**yes**

Use the center of mass.

**massw-split1**

(no) As above, but for *split-group1* (page 95).

**solvent-group**

Name of the index group of solvent molecules.

**coupl-steps**

(10) Average the number of ions per compartment over these many swap attempt steps. This can be used to prevent that ions near a compartment boundary (diffusing through a channel, e.g.) lead to unwanted back and forth swaps.

**iontypes**

(1) The number of different ion types to be controlled. These are during the simulation exchanged with solvent molecules to reach the desired reference numbers.

**iontype0-name**

Name of the first ion type.

**iontype0-in-A**

(-1) Requested (=reference) number of ions of type 0 in compartment A. The default value of -1 means: use the number of ions as found in time step 0 as reference value.

**iontype0-in-B**

(-1) Reference number of ions of type 0 for compartment B.

**bulk-offsetA**

(0.0) Offset of the first swap layer from the compartment A midplane. By default (i.e. bulk offset = 0.0), ion/water exchanges happen between layers at maximum distance (= bulk concentration) to the split group layers. However, an offset b (-1.0 < b < +1.0) can be specified to offset the bulk layer from the middle at 0.0 towards one of the compartment-partitioning layers (at +/- 1.0).

**bulk-offsetB**

(0.0) Offset of the other swap layer from the compartment B midplane.

**threshold**

(1) Only swap ions if threshold difference to requested count is reached.

**cyl0-r**

(2.0) [nm] Radius of the split cylinder #0. Two split cylinders (mimicking the channel pores) can optionally be defined relative to the center of the split group. With the help of these cylinders it can be counted which ions have passed which channel. The split cylinder definition has no impact on whether or not ion/water swaps are done.

**cyl0-up**

(1.0) [nm] Upper extension of the split cylinder #0.

**cyl0-down**

(1.0) [nm] Lower extension of the split cylinder #0.

**cyl1-r**

(2.0) [nm] Radius of the split cylinder #1.

**cyl1-up**

(1.0) [nm] Upper extension of the split cylinder #1.

**cyl1-down**

(1.0) [nm] Lower extension of the split cylinder #1.

### Density-guided simulations

These options enable and control the calculation and application of additional forces that are derived from three-dimensional densities, e.g., from cryo electron-microscopy experiments. (See the reference manual for details)

**density-guided-simulation-active**

(no) Activate density-guided simulations.

**density-guided-simulation-group**

(protein) The atoms that are subject to the forces from the density-guided simulation and contribute to the simulated density.

**density-guided-simulation-similarity-measure**

(inner-product) Similarity measure between the density that is calculated from the atom positions and the reference density.

**inner-product**

Takes the sum of the product of reference density and simulated density voxel values.

**relative-entropy**

Uses the negative relative entropy (or Kullback-Leibler divergence) between reference density and simulated density as similarity measure. Negative density values are ignored.

**cross-correlation**

Uses the Pearson correlation coefficient between reference density and simulated density as similarity measure.

**density-guided-simulation-atom-spreading-weight**

(unity) Determines the multiplication factor for the Gaussian kernel when spreading atoms on the grid.

**unity**

Every atom in the density fitting group is assigned the same unit factor.

**mass**

Atoms contribute to the simulated density proportional to their mass.

**charge**

Atoms contribute to the simulated density proportional to their charge.

**density-guided-simulation-force-constant**

(1e+09) [kJ $mol^{-1}$] The scaling factor for density-guided simulation forces. May also be negative.

**density-guided-simulation-gaussian-transform-spreading-width**

(0.2) [nm] The Gaussian RMS width for the spread kernel for the simulated density.

**density-guided-simulation-gaussian-transform-spreading-range-in-multiples-of-wid**

(4) The range after which the gaussian is cut off in multiples of the Gaussian RMS width described above.

**density-guided-simulation-reference-density-filename**

(reference.mrc) Reference density file name using an absolute path or a path relative to the to the folder from which *gmx mdrun* (page 258) is called.

**density-guided-simulation-nst**

(1) Interval in steps at which the density fitting forces are evaluated and applied. The forces are scaled by this number when applied (See the reference manual for details).

**density-guided-simulation-normalize-densities**

(true) Normalize the sum of density voxel values to one for the reference density as well as the simulated density.

**density-guided-simulation-adaptive-force-scaling**

(false) Adapt the force constant to ensure a steady increase in similarity between simulated and reference density.

**true**

Use adaptive force scaling.

**density-guided-simulation-adaptive-force-scaling-time-constant**

(4) [ps] Couple force constant to increase in similarity with reference density with this time constant. Larger times result in looser coupling.

**density-guided-simulation-shift-vector**

(0,0,0) [nm] Add this vector to all atoms in the density-guided-simulation-group before calculating forces and energies for density-guided simulations. Affects only the density-guided simulation forces and energies. Corresponds to a shift of the input density in the opposite direction by (-1) * density-guided-simulation-shift-vector.

**density-guided-simulation-transformation-matrix**

(1,0,0,0,1,0,0,0,1) Multiply all atoms with this matrix in the density-guided-simulation-group before calculating forces and energies for density-guided simulations. Affects only the density-guided simulation forces and energies. Corresponds to a transformation of the input density by the inverse of this matrix. The matrix is given in row-major order. This option allows, e.g., rotation of the density-guided atom group around the z-axis by $\theta$ degrees by using the following input: $(\cos\theta, -\sin\theta, 0, \sin\theta, \cos\theta, 0, 0, 0, 1)$ .

### QM/MM simulations with CP2K Interface

These options enable and control the calculation and application of additional QM/MM forces that are computed by the CP2K package if it is linked into GROMACS. For further details about QM/MM interface implementation follow *Hybrid Quantum-Classical simulations (QM/MM) with CP2K interface* (page 631).

**qmmm-cp2k-active**

(false) Activate QM/MM simulations. Requires CP2K to be linked with GROMACS

**qmmm-cp2k-qmgroup**

(System) Index group with atoms that are treated with QM.

**qmmm-cp2k-qmmethod**

(PBE) Method used to describe the QM part of the system.

**PBE**

DFT using PBE functional and DZVP-MOLOPT basis set.

**PBE-D3**

DFT using PBE functional, DZVP-MOLOPT basis set and D3 dispersion correction.

**BLYP**

DFT using BLYP functional and DZVP-MOLOPT basis set.

**BLYP-D3**

DFT using BLYP functional, DZVP-MOLOPT basis set and D3 dispersion correction.

**B3LYP**

DFT using B3LYP hybrid functional and 6-31G* basis set.

**B3LYP-D3**

DFT using B3LYP hybrid functional, 6-31G* basis set and D3 dispersion correction.

**PBE0**
DFT using PBE0 hybrid functional and 6-31G* basis set.

**PBE0-D3**
DFT using PBE0 hybrid functional, 6-31G* basis set and D3 dispersion correction.

**CAM-B3LYP**
DFT using CAM-B3LYP range-separated functional and 6-31G* basis set.

**CAM-B3LYP-D3**
DFT using CAM-B3LYP range-separated functional, 6-31G* basis set and D3 dispersion correction.

**WB97X**
DFT using wB97X range-separated functional and 6-31G* basis set.

**WB97X-D3**
DFT using wB97X-D3 range-separated functional with built-in D3 dispersion correction and 6-31G* basis set.

**SCC-DFTB**
Self-consistent-charge DFTB using CP2K's bundled `DFTB/scc` parameters. The bundled parameters support C, H, N, O, P, S, and Zn, except for the P–Zn pair.

**GFN1-xTB**
GFN1-xTB through CP2K's tblite interface.

**GFN2-xTB**
GFN2-xTB through CP2K's tblite interface.

**INPUT**
Provide an external input file for CP2K when running *gmx grompp* (page 230) with the `-qmi` command-line option. External input files are subject to the limitations that are described in *Hybrid Quantum-Classical simulations (QM/MM) with CP2K interface* (page 631).

**qmmm-cp2k-dftb-electrostatic-coupling**
(point-charge) Electrostatic QM/MM coupling scheme for tight-binding methods in CP2K.

**point-charge**
Periodic QM/MM electrostatics through short-range regularized point charges with smooth particle mesh Ewald (SPME).

**gauss**
Periodic QM/MM electrostatics through GEEP (Gauss) MM charges with smooth particle mesh Ewald (SPME). This requires a CP2K 2027.1 or higher.

**qmmm-cp2k-qmcharge**
(0) Total charge of the QM part.

**qmmm-cp2k-qmmultiplicity**
(1) Multiplicity or spin-state of QM part. Default value 1 means singlet state.

**qmmm-cp2k-qmfilenames**
() Names of the CP2K files that will be generated during the simulation. When using the default, empty, value the name of the simulation input file will be used with an additional `_cp2k` suffix.

### Collective variables (Colvars) module

These options enable and control the features provided by the collective variables (Colvars) module, a software library for enhanced sampling methods in molecular simulations. The Colvars module is described in ref. *195* (page 693) as well as other references that are reported in the log file when the corresponding features are used. For further details about Colvars interface implementation follow *Collective Variable simulations with the Colvars module* (page 646).

**colvars-active**

(false) Activate Colvars computation in the current run. Requires that the Colvars library was compiled with GROMACS, which is the default in a typical installation.

**colvars-configfile**

Name of the Colvars configuration file, using options specific to Colvars that are documented at: https://colvars.github.io/gromacs-2026/colvars-refman-gromacs.html. The file name can be either an absolute path, or a path relative to the working directory when *gmx grompp* (page 230) is called.

**colvars-seed**

(-1) [integer] Seed used to initialize the random generator associated with certain stochastic methods implemented within Colvars. The default value of -1 generates a random seed.

The current implementation of the Colvars-GROMACS interface gathers the relevant atomic coordinates on one MPI rank, where all collective variables and their forces are computed. Take this fact into account when choosing how many atoms to include in selections.

### NNP/MM simulations with neural network potentials in the NNPot interface

These options enable and control the calculation and application of additional forces derived from neural network potentials via the nnpot interface, if GROMACS is built with LibTorch support. For further details about NNP/MM interface implementation follow *Neural Network Potentials* (page 647).

**nnpot-active**

(false) Activate NNP/MM simulations via the NNPot interface. Requires GROMACS to be built with LibTorch support.

**nnpot-modelfile**

(model.pt) Path to a TorchScript-compiled model, either absolute or relative to the simulation directory.

**nnpot-input-group**

(System) Index group defining the input atoms for the NNP subsystem. Defaults to `System`, which performs a pure NNP simulation.

**nnpot-embedding**

(mechanical) Type of embedding scheme to use for NNP/MM simulations.

**mechanical**

Mechanical embedding scheme, NNP-MM interactions are treated classically.

**electrostatic-model**

Electrostatic embedding scheme, NNP-MM interactions are computed by the NNP model. In this case, the NNP model is expected to return the total energy, as well as forces on NNP and MM atoms.

**nnpot-model-input[1-9]**

Names of input fields to be filled in by GROMACS at each step and passed to the NNP model. Up to 9 inputs can be specified:

**atom-positions**

Positions of the NNP atoms specified by *nnpot-input-group* (page 100).

**atom-numbers**

Atomic numbers of the NNP atoms specified by *nnpot-input-group* (page 100).

**atom-pairs**

Pairs of NNP atoms specified by *nnpot-input-group* (page 100), filter by the *pair-cutoff* (page 101), as a vector of shape (N_pairs, 2).

**pair-shifts**

Periodic box shift vectors for the NNP atom pairs, as a vector of shape (N_pairs, 3).

**atom-positions-mm**

Positions of the MM atoms surrounding the NNP region, needed for `electrostatic-model` embedding.

**atom-charges-mm**

Charges of the MM atoms surrounding the NNP region, needed for `electrostatic-model` embedding.

**nnp-charge**

Total charge of the NNP atoms.

**box**

Simulation box vectors.

**pbc**

Periodic boundary conditions.

**pair-cutoff**

(0.0) [nm] Cutoff distance for pairs of NNP atoms. Positive cutoff value must be specified when requesting `atom-pairs` input.

**nnpot-link-type**

(H) Type of link atoms to be used in the NNP/MM simulation, specified by element symbol. Defaults to hydrogen (H).

**nnpot-link-distance**

(0.1) [nm] Distance between link atom and the bonded MM atom.

### Fast Multipole Method (FMM) Interface

These options enable an FMM backend and control its settings. For further details about the FMM interface implementation follow *Fast Multipole Method (FMM)* (page 650).

**fmm-backend**

(inactive) Activate an FMM library for electrostatics.

**inactive**
No FMM library is used.

**exafmm**
Use ExaFMM as FMM library.

**fmsolvr**
Use FMSolvr as FMM library.

### Options controlling ExaFmm

**fmm-exafmm-direct-provider**
(GROMACS) Selects direct interaction provider.

**GROMACS**
Use GROMACS to compute direct interactions.

**FMM**
Use ExaFMM to compute direct interactions.

**fmm-exafmm-direct-range**
(2) Specifies whether ExaFMM includes only immediate or also second-nearest neighboring cells in direct interactions. Accepts `1` or `2`. Must be `2` when using GROMACS as direct provider.

**fmm-exafmm-order**
(6) Multipole expansion order. Must be greater than `0`.

**fmm-exafmm-tree-type**
(uniform) Tree structure used by ExaFMM.

**uniform**
Use uniform tree structure.

**adaptive**
Use adaptive tree structure.

**fmm-exafmm-tree-depth**
(0) Tree depth for uniform tree. Required when using `FMM` as direct provider. Must not be set for adaptive tree.

**fmm-exafmm-max-particles-per-cell**
(0) Maximum particles per cell for adaptive tree. Required and positive when `fmm-exafmm-tree-type = adaptive`. Must not be set for uniform tree.

### Options controlling FMSolvr

**fmm-fmsolvr-order**

(8) Multipole expansion order. Must be greater than `0`.

**fmm-fmsolvr-tree-depth**

(3) Tree depth controlling spatial subdivision.

**fmm-fmsolvr-direct-provider**

(FMM) Selects direct interaction provider.

**GROMACS**

Use GROMACS to compute direct interactions.

**FMM**

Use FMSolvr to compute direct interactions.

**fmm-fmsolvr-direct-range**

(1) Direct interaction settings. Must be `1`, as FMSolvr currently supports direct interactions only with immediate neighboring cells.

**fmm-fmsolvr-dipole-compensation**

(yes) Enables dipole compensation.

**fmm-fmsolvr-sparse**

(no) Enable performance optimizations for sparse simulation systems with a lot of vacuum.

### User defined thingies

**user1-grps**

**user2-grps**

**userint1 (0)**

**userint2 (0)**

**userint3 (0)**

**userint4 (0)**

**userreal1 (0)**

**userreal2 (0)**

**userreal3 (0)**

**userreal4 (0)**

These you can use if you modify code. You can pass integers and reals and groups to your subroutine. Check the inputrec definition in `src/gromacs/mdtypes/inputrec.h`

#### Removed features

These features have been removed from GROMACS, but so that old *mdp* (page 579) and *tpr* (page 586) files cannot be mistakenly misused, we still parse these options. *gmx grompp* (page 230) and *gmx mdrun* (page 258) will issue a fatal error if they are set.

**adress**

(no)

**implicit-solvent**

(no)

## 3.8 Useful mdrun features

This section discusses features in *gmx mdrun* (page 258) that do not fit well elsewhere.

### 3.8.1 Re-running a simulation

The rerun feature allows you to take any trajectory file `traj.trr` and compute quantities based upon the coordinates in that file using the model physics supplied in the `topol.tpr` file. It can be used with command lines like `mdrun -s topol -rerun traj.trr`. That *tpr* (page 586) could be different from the one that generated the trajectory. This can be used to compute the energy or forces for exactly the coordinates supplied as input, or to extract quantities based on subsets of the molecular system (see *gmx convert-tpr* (page 178) and *gmx trjconv* (page 334)). It is easier to do a correct "single-point" energy evaluation with this feature than a 0-step simulation.

Neighbor searching is performed for every frame in the trajectory independently of the value in *nstlist* (page 57), since *gmx mdrun* (page 258) can no longer assume anything about how the structures were generated. Naturally, no update or constraint algorithms are ever used.

The rerun feature cannot, in general, compute many of the quantities reported during full simulations. It does only take positions as input (ignoring potentially present velocities), and does only report potential energies, volume and density, dH/dl terms, and restraint information. It does notably not report kinetic, total or conserved energy, temperature, virial or pressure.

### 3.8.2 Running a simulation in reproducible mode

It is generally difficult to run an efficient parallel MD simulation that is based primarily on floating-point arithmetic and is fully reproducible. By default, *gmx mdrun* (page 258) will observe how things are going and vary how the simulation is conducted in order to optimize throughput. However, there is a "reproducible mode" available with `mdrun -reprod` that will systematically eliminate all sources of variation within that run; repeated invocations on the same input and hardware will be binary identical. However, running in this mode on different hardware, or with a different compiler, etc. will not be reproducible. This should normally only be used when investigating possible problems.

### 3.8.3 Halting running simulations

When *gmx mdrun* (page 258) receives a TERM or INT signal (e.g. when ctrl+C is pressed), it will stop at the next neighbor search step or at the second global communication step, whichever happens later. When *gmx mdrun* (page 258) receives a second TERM or INT signal and reproducibility is not requested, it will stop at the first global communication step. In both cases all the usual output will be written to file and a checkpoint file is written at the last step. When *gmx mdrun* (page 258) receives an ABRT signal or the third TERM or INT signal, it will abort directly without writing a new checkpoint file. When running with MPI, a signal to one of the *gmx mdrun* (page 258) ranks is sufficient, this signal should not be sent to mpirun or the *gmx mdrun* (page 258) process that is the parent of the others.

### 3.8.4 Running multi-simulations

There are numerous situations where running a related set of simulations within the same invocation of mdrun are necessary or useful. Running a replica-exchange simulation requires it, as do simulations using ensemble-based distance or orientation restraints. Running a related series of lambda points for a free-energy computation is also convenient to do this way, but beware of the potential side-effects related to resource utilization and load balance discussed later.

This feature requires configuring GROMACS with an *external MPI library* (page 10) so that the set of simulations can communicate. The `n` simulations within the set can use internal MPI parallelism also, so that `mpirun -np x gmx_mpi mdrun` for `x` a multiple of `n` will use `x/n` ranks per simulation.

To launch a multi-simulation, the `-multidir` option is used. For the input and output files of a multi-simulation a set of `n` subdirectories is required, one for each simulation. Place all the relevant input files in those directories (e.g. named `topol.tpr`), and launch a multi-simualtion with `mpirun -np x gmx_mpi mdrun -s topol -multidir <names-of-directories>`. If the order of the simulations within the multi-simulation is significant, you are responsible for ordering their names when you provide them to `-multidir`. Be careful with shells that do filename globbing dictionary-style, e.g. `dir1 dir10 dir11 ... dir2 ....`

#### Examples running multi-simulations

```
mpirun -np 32 gmx_mpi mdrun -multidir a b c d
```

Starts a multi-simulation on 32 ranks with 4 simulations. The input and output files are found in directories `a`, `b`, `c`, and `d`.

```
mpirun -np 32 gmx_mpi mdrun -multidir a b c d -gputasks␣
↪0000000011111111
```

Starts the same multi-simulation as before. On a machine with two physical nodes and two GPUs per node, there will be 16 MPI ranks per node, and 8 MPI ranks per simulation. The 16 MPI ranks doing PP work on a node are mapped to the GPUs with IDs 0 and 1, even though they come from more than one simulation. They are mapped in the order indicated, so that the PP ranks from each simulation use a single GPU. However, the order `0101010101010101` could run faster.

#### Running replica-exchange simulations

When running a multi-simulation, using `gmx mdrun -replex n` means that a replica exchange is attempted every given number of steps. The number of replicas is set with `-multidir` option, described above. All run input files should use a different value for the coupling parameter (e.g. temperature), which ascends over the set of input files. The random seed for replica exchange is set with `-reseed`. After every exchange, the velocities are scaled and neighbor searching is performed. See the Reference Manual for more details on how replica exchange functions in GROMACS.

#### Multi-simulation performance considerations

The frequency of communication across a multi-simulation can have an impact on performance. This is highly algorithm dependent, but in general it is recommended to set up a multi-simulation to do inter-simulation communication as infrequently as possible but as frequently as necessary. However, even when members of multi-simulation do not communicate frequently (or at all), and therefore the associated performance overhead is small or even negligible, load imbalance can still have a significant impact on performance and resource utilization. Current multi-simulation algorithms use a fixed interval for data exchange (e.g. replica exchange every `N` steps) and therefore all members of a multi-simulation need to reach this step before the collective communication can happen and any of them can proceed to step `N+1`. Hence, the slowest member of the multi-simulation will determine the performance of the entire ensemble. This load imbalance will not only limit performance but will also leave resources idle; e.g. if one of the simulations in an `n`-way multi-simulation runs at half the performance of the rest, the resources assigned to the `n-1` faster running simulations will be left idle for approximately half of the wall-time of the entire multi-simulation job. The source of this imbalance can range from inherent workload imbalance across the simulations within a multi-simulation to differences in hardware speed or inter-node network performance variability affecting a subset of ranks and therefore only some of the simulations. Reducing the amount of resources left idle requires reducing the load imbalance, which may involve splitting up non-communicating multi-simulations, or making sure to request a "compact" allocation on a cluster (if the job scheduler allows). Note that imbalance also applies to non-communicating multi-simulations like FEP calculations since the resources assigned to earlier finishing simulations can not be relinquished until the entire MPI job can finish.

### 3.8.5 Controlling the length of the simulation

Normally, the length of an MD simulation is best managed through the *mdp* (page 579) option *nsteps* (page 53), however there are situations where more control is useful. `gmx mdrun -nsteps 100` overrides the *mdp* (page 579) file and executes 100 steps. `gmx mdrun -maxh 2.5` will terminate the simulation shortly before 2.5 hours elapse, which can be useful when running under cluster queues (as long as the queuing system does not ever suspend the simulation).

## 3.9 Getting good performance from mdrun

Here we give an overview on the parallelization and acceleration schemes employed by GROMACS. The aim is to provide an understanding of the underlying mechanisms that make GROMACS one of the fastest molecular dynamics simulation packages. The information presented should help choosing appropriate parallelization options, run configuration, as well as acceleration options to achieve optimal simulation performance. We start with quick checklist. That is followed by more detailed discussions of different aspects affecting performance.

### 3.9.1 Performance checklist

There are many different aspects that affect the performance of simulations in GROMACS. Most simulations require a lot of computational resources, therefore it can be worthwhile to optimize the use of those resources. Several issues mentioned in the list below could lead to a performance difference of a factor of 2. So it can be useful go through the checklist.

#### GROMACS configuration

- Do not use double precision unless you are absolutely sure you need it.
- Compile the FFTW library (yourself) with the correct flags on x86 (in most cases, the correct flags are automatically configured).
- On x86, use gcc as the compiler (not icc, pgi or the Cray compiler).
- On POWER, use gcc instead of IBM's xlc.
- Use a new compiler version.
- MPI library: OpenMPI usually has good performance and causes little trouble.
- Make sure your compiler supports OpenMP (some versions of Clang do not).
- If you have GPUs that support either CUDA, OpenCL, SYCL or HIP use them.
  - Configure with `-DGMX_GPU=CUDA`, `-DGMX_GPU=OpenCL`, `-DGMX_GPU=SYCL` or `-DGMX_GPU=HIP`.
  - For GPUs, use the newest available SDK for your GPU to take advantage of the latest performance enhancements.
  - Use a recent GPU driver.
  - Make sure you use an *gmx mdrun* (page 258) with `GMX_SIMD` appropriate for the CPU architecture; the log file will contain a warning note if suboptimal setting is used. However, prefer `AVX2` over `AVX512` in GPU or highly parallel MPI runs (for more information see the *intra-core parallelization information* (page 112)).
  - If compiling on a cluster head node, make sure that `GMX_SIMD` is appropriate for the compute nodes.

### Run setup

- For an approximately spherical solute, use a rhombic dodecahedron unit cell.
- When using a time-step of <=2.5 fs, use *constraints=h-bonds* (page 69) (and not *constraints=all-bonds* (page 69)), since:
  - this is faster, especially with GPUs;
  - it is necessary in order to be able to use GPU-resident mode;
  - and most force fields have been parametrized with only bonds involving hydrogens constrained.
- When running on CPUs only or with GPUs but integration on the CPU, you can use multiple time stepping to perform the PME mesh calculations every second step: set `mts` to `yes`.
- You can often increase the time-step to 4 fs by repartitioning hydrogen masses using the `mass-repartition-factor` mdp option. This does not affect equilibrium distributions, but makes dynamics slightly slower.
- For massively parallel runs with PME, you might need to try different numbers of PME ranks (`gmx mdrun -npme ???`) to achieve best performance; *gmx tune_pme* (page 340) can help automate this search.
- For massively parallel runs (also `gmx mdrun -multidir`), or with a slow network, global communication can become a bottleneck and you can reduce it by choosing larger periods for algorithms such as temperature and pressure coupling).

### Checking and improving performance

- Look at the end of the `md.log` file to see the performance and the cycle counters and wall-clock time for different parts of the MD calculation. The PP/PME load ratio is also printed, with a warning when a lot of performance is lost due to imbalance.
- Six performance metrics are reported in the `md.log`. Select the appropriate metric for benchmarking based on the purpose of the simulation.

**ns/day**
: This is a commonly used metric for simulation throughput that is specific to simulation systems, conditions, and hardware.

**hour/ns**
: As the inverse of ns/day, hour/ns can be used to estimate the time to obtain a desired simulation time.

**ms/step**
: This is a time step-independent metric that directly measures the wall-time per timestep and can be used to measure performance with a given system.

**Matom*steps/s**
: This is a metric for simulation throughput that is timestep-independent and normalized by system size. It provides a measure of simulation efficiency, as the amount of work scales nearly linearly with the number of atoms, given the same type of system and simulation settings.

**Mnbf/s (Mega non-bonded forces per second)**
: This throughput metric is time step-independent and partially normalized by system size;

When non-bonded interactions dominate in the simulations, Mnbf/s can be used to compare hardware or algorithmic efficiency.

**MFlops (Mega floating-point operations per second)**
This is not an MD-specific metric and gives an estimate of the achieved flop rate, which can be compared with the theoretical peak flop rate of the hardware. Note that this is an estimate of the useful floating point operations in the code, not the actual operations emitted by the compiler or measured during profiling. MFlops and Mnbf/s are only printed when the environment variable `GMX_DETAILED_PERF_STATS` is set.

- Adjust the number of PME ranks and/or the cut-off and PME grid-spacing when there is a large PP/PME imbalance. Note that even with a small reported imbalance, the automated PME-tuning might have reduced the initial imbalance. You could still gain performance by changing the mdp parameters or increasing the number of PME ranks.
- (Especially) In GPU-resident runs (`-update gpu`):
  - Frequent virial or energy computation can have a large overhead (and this will not show up in the cycle counters). To reduce this overhead, increase `nstcalcenergy`;
  - Frequent temperature or pressure coupling can have significant overhead; to reduce this, make sure to have as infrequent coupling as your algorithms allow (typically >=50-100 steps).
- If the neighbor searching and/or domain decomposition takes a lot of time, increase `nstlist`. If a Verlet buffer tolerance is used, this is done automatically by *gmx mdrun* (page 258) and the pair-list buffer is increased to keep the energy drift constant.
  - especially with multi-GPU runs, the automatic increasing of `nstlist` at `mdrun` startup can be conservative and larger value is often be optimal (e.g. `nstlist=200-300` with PME and default Verlet buffer tolerance).
  - odd values of nstlist should be avoided when using CUDA Graphs to minimize the overhead associated with graph instantiation.
- If `Comm. energies` takes a lot of time (a note will be printed in the log file), increase `nstcalcenergy`.
- If all communication takes a lot of time, you might be running on too many cores, or you could try running combined MPI/OpenMP parallelization with 2 or 4 OpenMP threads per MPI process.
- In multi-GPU runs, avoid using as many ranks as cores (or hardware threads) since this introduces a major inefficiency due to overheads associated to GPUs sharing by several MPI ranks. Use at most a few ranks per GPU, 1-3 ranks is generally optimal; with GPU-resident mode and direct GPU communication typically 1 rank/GPU is best.

The GROMACS build system and the *gmx mdrun* (page 258) tool have a lot of built-in and configurable intelligence to detect your hardware and make pretty effective use of it. For a lot of casual and serious use of *gmx mdrun* (page 258), the automatic machinery works well enough. But to get the most from your hardware to maximize your scientific quality, read on!

### 3.9.2 Hardware background information

Modern computer hardware is complex and heterogeneous, so we need to discuss a little bit of background information and set up some definitions. Experienced HPC users can skip this section.

**core**
: A hardware compute unit that actually executes instructions. There is normally more than one core in a processor, often many more.

**cache**
: A special kind of memory local to core(s) that is much faster to access than main memory, kind of like the top of a human's desk, compared to their filing cabinet. There are often several layers of caches associated with a core.

**socket**
: A group of cores that share some kind of locality, such as a shared cache. This makes it more efficient to spread computational work over cores within a socket than over cores in different sockets. Modern server and workstation class machines often have more than one CPU socket.

**node**
: A group of sockets that share coarser-level locality, such as shared access to the same memory without requiring any network hardware. A normal personal computer or racked server is a node. A node is often the smallest amount of a large compute cluster that a user can request to use.

**thread**
: A stream of instructions for a core to execute. There are many different programming abstractions that create and manage spreading computation over multiple threads, such as OpenMP, pthreads, winthreads, CUDA, SYCL, OpenCL, and OpenACC. Some kinds of hardware can map more than one software thread to a core; on Intel x86 processors this is called "hyper-threading", while the more general concept is often called SMT for "simultaneous multi-threading". IBM Power8 can for instance use up to 8 hardware threads per core. This feature can usually be enabled or disabled either in the hardware BIOS or through a setting in the Linux operating system. GROMACS can typically make use of this, for a moderate free performance boost. In most cases it will be enabled by default e.g. on new x86 processors, but in some cases the system administrators might have disabled it. If that is the case, ask if they can re-enable it for you. If you are not sure if it is enabled, check the output of the CPU information in the log file and compare with CPU specifications you find online.

**thread affinity (pinning)**
: By default, most operating systems allow software threads to migrate between cores (or hardware threads) to help automatically balance workload. However, the performance of *gmx mdrun* (page 258) can deteriorate if this is permitted and will degrade dramatically especially when relying on multi-threading within a rank. To avoid this, *gmx mdrun* (page 258) will by default set the affinity of its threads to individual cores/hardware threads, unless the user or software environment has already done so (or not the entire node is used for the run, i.e. there is potential for node sharing). Setting thread affinity is sometimes called thread "pinning".

**MPI (Message Passing Interface)**
: The dominant multi-node parallelization-scheme, which provides a standardized language in which programs can be written that work across more than one node.

**rank**
: In MPI, a rank is the smallest grouping of hardware used in the multi-node parallelization scheme. That grouping can be controlled by the user, and might correspond to a core, a socket, a node, or a group of nodes. The best choice varies with the hardware, software and compute task. Sometimes

an MPI rank is called an MPI process.

**GPU**
A graphics processing unit, which is often faster and more efficient than conventional processors for particular kinds of compute workloads. A GPU is always associated with a particular node, and often a particular socket within that node.

**OpenMP**
A standardized technique supported by many compilers to share a compute workload over multiple cores. Often combined with MPI to achieve hybrid MPI/OpenMP parallelism.

**CUDA**
A proprietary parallel computing framework and API developed by NVIDIA that allows targeting their accelerator hardware. GROMACS uses CUDA for GPU acceleration support with NVIDIA hardware.

**OpenCL**
An open standard-based parallel computing framework that consists of a C99-based compiler and a programming API for targeting heterogeneous and accelerator hardware. GROMACS uses OpenCL for GPU acceleration on AMD devices (both GPUs and APUs), Intel integrated GPUs, and Apple Silicon integrated GPUs; some NVIDIA hardware is also supported. In GROMACS, OpenCL has been deprecated in favor of SYCL.

**SYCL**
An open standard based on C++17 for targeting heterogeneous systems. SYCL has several implementations, of which GROMACS supports two: Intel oneAPI DPC++ and AdaptiveCpp. GROMACS can use SYCL for GPU acceleration on AMD and Intel GPUs. There is experimental support for NVIDIA GPUs too.

**HIP**
A parallel computing framework and API developed by AMD to target their (and theoretically NVIDIA) accelerator hardware. GROMACS can use HIP for GPU acceleration on GCN, CDNA and RDNA AMD hardware.

**SIMD**
A type of CPU instruction by which modern CPU cores can execute multiple floating-point instructions in a single cycle.

### 3.9.3 Work distribution by parallelization in GROMACS

The algorithms in *gmx mdrun* (page 258) and their implementations are most relevant when choosing how to make good use of the hardware. For details, see the *Reference Manual* (page 418). The most important of these are

**Domain Decomposition**
The domain decomposition (DD) algorithm decomposes the (short-ranged) component of the non-bonded interactions into domains that share spatial locality, which permits the use of efficient algorithms. Each domain handles all of the particle-particle (PP) interactions for its members, and is mapped to a single MPI rank. Within a PP rank, OpenMP threads can share the workload, and some work can be offloaded to a GPU. The PP rank also handles any bonded interactions for the members of its domain. A GPU may perform work for more than one PP rank, but it is normally most efficient to use a single PP rank per GPU and for that rank to have thousands of particles. When the work of a PP rank is done on the CPU, *mdrun* (page 258) will make extensive use of the SIMD capabilities of the core. There are various *command-line options* (page 114) to control the behaviour of the DD algorithm.

**Particle-mesh Ewald**

The particle-mesh Ewald (PME) algorithm treats the long-ranged component of the non-bonded interactions (Coulomb and possibly also Lennard-Jones). Either all, or just a subset of ranks may participate in the work for computing the long-ranged component (often inaccurately called simply the "PME" component). Because the algorithm uses a 3D FFT that requires global communication, its parallel efficiency gets worse as more ranks participate, which can mean it is fastest to use just a subset of ranks (e.g. one-quarter to one-half of the ranks). If there are separate PME ranks, then the remaining ranks handle the PP work. Otherwise, all ranks do both PP and PME work.

### 3.9.4 Parallelization schemes

GROMACS, being performance-oriented, has a strong focus on efficient parallelization. There are multiple parallelization schemes available, therefore a simulation can be run on a given hardware with different choices of run configuration.

#### Intra-core parallelization via SIMD: SSE, AVX, etc.

One level of performance improvement available in GROMACS is through the use of `Single Instruction Multiple Data (SIMD)` instructions. In detail information for those can be found under *SIMD support* (page 18) in the installation guide.

In GROMACS, SIMD instructions are used to parallelize the parts of the code with the highest impact on performance (nonbonded and bonded force calculation, PME and neighbour searching), through the use of hardware specific SIMD kernels. Those form one of the three levels of non-bonded kernels that are available: reference or generic kernels (slow but useful for producing reference values for testing), optimized plain-C kernels (can be used cross-platform but still slow) and SIMD intrinsics accelerated kernels.

The SIMD intrinsic code is compiled by the compiler. Technically, it is possible to compile different levels of acceleration into one binary, but this is difficult to manage with acceleration in many parts of the code. Thus, you need to configure and compile GROMACS for the SIMD capabilities of the target CPU. By default, the build system will detect the highest supported acceleration of the host where the compilation is carried out. For cross-compiling for a machine with a different highest SIMD instructions set, in order to set the target acceleration, the `-DGMX_SIMD` CMake option can be used. To use a single installation on multiple different machines, it is convenient to compile the analysis tools with the lowest common SIMD instruction set (as these rely little on SIMD acceleration), but for best performance *mdrun* (page 258) should be compiled be compiled separately with the highest (latest) `native` SIMD instruction set of the target architecture (supported by GROMACS).

Some Intel CPU architectures bring tradeoffs between the maximum clock frequency of the CPU (ie. its speed), and the width of the SIMD instructions it executes (ie its throughput at a given speed). In particular, the Intel `Skylake` and `Cascade Lake` processors (e.g. Xeon SP Gold/Platinum), can offer better throughput when using narrower SIMD because of the better clock frequency available. Consider building *mdrun* (page 258) configured with `GMX_SIMD=AVX2_256` instead of `GMX_SIMD=AVX512` for better performance in GPU accelerated or highly parallel MPI runs.

Some of the latest ARM based CPU, such as the Fujitsu A64fx, support the Scalable Vector Extensions (SVE). Though SVE can be used to generate fairly efficient Vector Length Agnostic (VLA) code, this is not a good fit for GROMACS, as the SIMD vector length is fixed at CMake time. The default is to automatically detect the default vector length at CMake time (via the `/proc/sys/abi/sve_default_vector_length` pseudo-file, and this can be changed by configuring with `GMX_SIMD_`-

ARM_SVE_LENGTH=<len>. The supported vector lengths are 128, 256, 512 and 1024. Since the SIMD short-range non-bonded kernels only support up to 16 floating point numbers per SIMD vector, 1024 bits vector length is only valid in double precision (e.g. -DGMX_DOUBLE=on). Note that even if *mdrun* (page 258) does check the SIMD vector length at runtime, running with a different vector length than the one used at CMake time is undefined behavior, and *mdrun* (page 258) might crash before reaching the check (that would abort with a user-friendly error message).

### Process(-or) level parallelization via OpenMP

GROMACS *mdrun* (page 258) supports OpenMP multithreading for all parts of the code. OpenMP is enabled by default and can be turned on/off at configure time with the GMX_OPENMP CMake variable and at run-time with the -ntomp option (or the OMP_NUM_THREADS environment variable). The OpenMP implementation is quite efficient and scales well for up to 12-24 threads on Intel and 6-8 threads on AMD CPUs.

### Node level parallelization via GPU offloading and thread-MPI

### Multithreading with thread-MPI

The thread-MPI library implements a subset of the MPI specification, based on the system threading support. Both POSIX pthreads and Windows threads are supported. Acting as a drop-in replacement for MPI, thread-MPI enables compiling and running *mdrun* (page 258) on a single machine (i.e. not across a network) without MPI. Additionally, thread-MPI does in some cases make *mdrun* (page 258) run slightly faster than with MPI.

Thread-MPI is included in the GROMACS source and it is the default parallelization mode, practically rendering the serial *mdrun* (page 258) deprecated. Compilation with thread-MPI is controlled by the GMX_THREAD_MPI CMake variable.

Thread-MPI is compatible with most *mdrun* (page 258) features and parallelization schemes, including OpenMP, GPUs; it is not compatible with MPI and multi-simulation runs.

By default, the thread-MPI *mdrun* (page 258) will use all available cores in the machine by starting an appropriate number of ranks or OpenMP threads to occupy all of them. The number of ranks can be controlled using the -nt and -ntmpi options. -nt represents the total number of threads to be used (which can be a mix of thread-MPI and OpenMP threads).

### Hybrid/heterogeneous acceleration

Hybrid acceleration means distributing compute work between available CPUs and GPUs to improve simulation performance. New non-bonded algorithms have been developed with the aim of efficient acceleration both on CPUs and GPUs.

The most compute-intensive parts of simulations, non-bonded force calculation, as well as possibly the PME, bonded force calculation and update and constraints can be offloaded to GPUs and carried out simultaneously with remaining CPU work. Native GPU acceleration is supported for the most commonly used algorithms in GROMACS. For more information about the GPU kernels, please see the *Installation guide* (page 8).

The native GPU acceleration can be turned on or off, either at run-time using the *mdrun* (page 258) -nb option, or at configuration time using the GMX_GPU CMake variable.

To efficiently use all compute resource available, CPU and GPU computation is done simultaneously. Overlapping with the OpenMP multithreaded bonded force and PME long-range electrostatic calculations on the CPU, non-bonded forces are calculated on the GPU. Multiple GPUs, both in a single node as well as across multiple nodes, are supported using domain-decomposition. A single GPU is assigned to the non-bonded workload of a domain, therefore, the number of GPUs used has to match the number of MPI processes (or thread-MPI threads) the simulation is started with. The available CPU cores are partitioned among the processes (or thread-MPI threads) and a set of cores with a GPU do the calculations on the respective domain.

With PME electrostatics, *mdrun* (page 258) supports automated CPU-GPU load-balancing by shifting workload from the PME mesh calculations, done on the CPU, to the particle-particle non-bonded calculations, done on the GPU. At startup, a few stages of tuning are executed during the first few thousand MD steps. These stages involve scaling the electrostatics cut-off and PME grid spacing to determine the value that gives optimal CPU-GPU load balance. The cut-off value provided using the *rcoulomb* (page 60) =`rvdw` *mdp* (page 579) option represents the minimum electrostatics cut-off the tuning starts with and therefore should be chosen as small as possible (but still reasonable for the physics simulated). The Lennard-Jones cut-off `rvdw` is kept fixed. We do not allow scaling to shorter cut-off as we do not want to change `rvdw` because that would affect the validity of the force-field parameters and there would be no performance gain.

While the automated CPU-GPU load balancing always attempts to find the optimal cut-off setting, it might not always be possible to balance CPU and GPU workload. This happens when the CPU threads finish calculating the bonded forces and PME before the GPU finishes the non-bonded force calculation, even with the shortest possible cut-off. In such cases the CPU will wait for the GPU and this time will show up as `Wait GPU NB local` in the cycle and timing summary table at the end of the log file.

### Parallelization over multiple nodes via MPI

At the heart of the MPI parallelization in GROMACS is the neutral-territory *domain decomposition* (page 111) with dynamic load balancing. To parallelize simulations across multiple machines (e.g. nodes of a cluster) *mdrun* (page 258) needs to be compiled with MPI which can be enabled using the `GMX_MPI` CMake variable.

### Controlling the domain decomposition algorithm

This section lists options that affect how the domain decomposition algorithm decomposes the workload to the available parallel hardware.

**`-rdd`**
: Can be used to set the required maximum distance for inter charge-group bonded interactions. Communication for two-body bonded interactions below the non-bonded cut-off distance always comes for free with the non-bonded communication. Particles beyond the non-bonded cut-off are only communicated when they have missing bonded interactions; this means that the extra cost is minor and nearly independent of the value of `-rdd`. With dynamic load balancing, option `-rdd` also sets the lower limit for the domain decomposition cell sizes. By default `-rdd` is determined by *gmx mdrun* (page 258) based on the initial coordinates. The chosen value will be a balance between interaction range and communication cost.

**`-ddcheck`**
: On by default. When inter charge-group bonded interactions are beyond the bonded cut-off distance, *gmx mdrun* (page 258) terminates with an error message. For pair interactions and tabulated bonds that do not generate exclusions, this check can be turned off with the option `-noddcheck`.

**-rcon**

When constraints are present, option `-rcon` influences the cell size limit as well. Particles connected by NC constraints, where NC is the LINCS order plus 1, should not be beyond the smallest cell size. A error message is generated when this happens, and the user should change the decomposition or decrease the LINCS order and increase the number of LINCS iterations. By default *gmx mdrun* (page 258) estimates the minimum cell size required for P-LINCS in a conservative fashion. For high parallelization, it can be useful to set the distance required for P-LINCS with `-rcon`.

**-dds**

Sets the minimum allowed x, y and/or z scaling of the cells with dynamic load balancing. *gmx mdrun* (page 258) will ensure that the cells can scale down by at least this factor. This option is used for the automated spatial decomposition (when not using `-dd`) as well as for determining the number of grid pulses, which in turn sets the minimum allowed cell size. Under certain circumstances the value of `-dds` might need to be adjusted to account for high or low spatial inhomogeneity of the system.

### Multi-level parallelization: MPI and OpenMP

The multi-core trend in CPU development substantiates the need for multi-level parallelization. Current multiprocessor machines can have 2-4 CPUs with a core count as high as 64. As the memory and cache subsystem is lagging more and more behind the multicore evolution, this emphasizes non-uniform memory access (NUMA) effects, which can become a performance bottleneck. At the same time, all cores share a network interface. In a purely MPI-parallel scheme, all MPI processes use the same network interface, and although MPI intra-node communication is generally efficient, communication between nodes can become a limiting factor to parallelization. This is especially pronounced in the case of highly parallel simulations with PME (which is very communication intensive) and with `''fat''` nodes connected by a slow network. Multi-level parallelism aims to address the NUMA and communication related issues by employing efficient intra-node parallelism, typically multithreading.

Combining OpenMP with MPI creates an additional overhead especially when running separate multithreaded PME ranks. Depending on the architecture, input system size, as well as other factors, MPI+OpenMP runs can be as fast and faster already at small number of processes (e.g. multi-processor Intel Westmere or Sandy Bridge), but can also be considerably slower (e.g. multi-processor AMD Interlagos machines). However, there is a more pronounced benefit of multi-level parallelization in highly parallel runs.

### Separate PME ranks

On CPU ranks, particle-particle (PP) and PME calculations are done in the same process one after another. As PME requires all-to-all global communication, this is most of the time the limiting factor to scaling on a large number of cores. By designating a subset of ranks for PME calculations only, performance of parallel runs can be greatly improved.

OpenMP multithreading in PME ranks is also possible. Using multi-threading in PME can can improve performance at high parallelization. The reason for this is that with N>1 threads the number of processes communicating, and therefore the number of messages, is reduced by a factor of N. But note that modern communication networks can process several messages simultaneously, such that it could be advantageous to have more processes communicating.

Separate PME ranks are not used at low parallelization, the switch at higher parallelization happens automatically (at > 16 processes). The number of PME ranks is estimated by mdrun. If the PME load

is higher than the PP load, mdrun will automatically balance the load, but this leads to additional (non-bonded) calculations. This avoids the idling of a large fraction of the ranks; usually 3/4 of the ranks are PP ranks. But to ensure the best absolute performance of highly parallel runs, it is advisable to tweak this number which is automated by the *tune_pme* (page 340) tool.

The number of PME ranks can be set manually on the *mdrun* (page 258) command line using the `-npme` option, the number of PME threads can be specified on the command line with `-ntomp_pme` or alternatively using the `GMX_PME_NUM_THREADS` environment variable. The latter is especially useful when running on compute nodes with different number of cores as it enables setting different number of PME threads on different nodes.

### 3.9.5 Running mdrun within a single node

*gmx mdrun* (page 258) can be configured and compiled in several different ways that are efficient to use within a single *node*. The default configuration using a suitable compiler will deploy a multi-level hybrid parallelism that uses CUDA/SYCL/OpenCL, OpenMP and the threading platform native to the hardware. For programming convenience, in GROMACS, those native threads are used to implement on a single node the same MPI scheme as would be used between nodes, but much more efficient; this is called thread-MPI. From a user's perspective, real MPI and thread-MPI look almost the same, and GROMACS refers to MPI ranks to mean either kind, except where noted. A real external MPI can be used for *gmx mdrun* (page 258) within a single node, but runs more slowly than the thread-MPI version.

By default, *gmx mdrun* (page 258) will inspect the hardware available at run time and do its best to make fairly efficient use of the whole node. The log file, stdout and stderr are used to print diagnostics that inform the user about the choices made and possible consequences.

A number of command-line parameters are available to modify the default behavior.

**`-nt`**
: The total number of threads to use. The default, 0, will start as many threads as available cores. Whether the threads are thread-MPI ranks, and/or OpenMP threads within such ranks depends on other settings.

**`-ntmpi`**
: The total number of thread-MPI ranks to use. The default, 0, will start one rank per GPU (if present), and otherwise one rank per core.

**`-ntomp`**
: The total number of OpenMP threads per rank to start. The default, 0, will start one thread on each available core. Alternatively, *mdrun* (page 258) will honor the appropriate system environment variable (e.g. `OMP_NUM_THREADS`) if set. Note that the maximum number of OpenMP threads (per rank) is, for efficiency reasons, limited to 128. While it is rarely beneficial to use a number of threads higher than this, the GMX_OPENMP_MAX_THREADS CMake variable can be used to increase the limit.

**`-npme`**
: The total number of ranks to dedicate to the long-ranged component of PME, if used. The default, -1, will dedicate ranks only if the total number of threads is at least 12, and will use around a quarter of the ranks for the long-ranged component.

**`-ntomp_pme`**
: When using PME with separate PME ranks, the total number of OpenMP threads per separate PME rank. The default, 0, copies the value from `-ntomp`.

**-pin**

Can be set to "auto," "on," "inherit," or "off" to control whether *mdrun* (page 258) will attempt to set the affinity of threads to cores. Defaults to "auto," which means that if *mdrun* (page 258) detects that all cores on the node are used for *mdrun* (page 258) and the affinity is left default (not set by an external tool such as an MPI launcher or `numactl`), then it behaves like "on," and attempts to set the affinities. With "on," mdrun will set thread affinities, overriding any external CPU affinity settings. With "inherit," mdrun will set thread affinities within the external affinity mask: that is, if an external tool or a job scheduler set CPU affinities limiting each process to a specific set of cores/hardware threads, ref:*mdrun* (page 258) rank(s) will pin thread(s) within these boundaries. The behavior of "on" and "inherit" is similar, both enable thread pinning, but they differ in scope: "on" pins threads across all available CPU cores in the system, while "inherit" only pins threads within the external affinity mask. This scope difference also affects how the `-pinoffset` and `-pinstride` options work.

**-pinoffset**

If `-pin on` or `-pin inherit`, specifies the logical core number to which *mdrun* (page 258) should pin the first thread. When running more than one instance of *mdrun* (page 258) on a node, use this option to to avoid pinning threads from different *mdrun* (page 258) instances to the same core. With `-pin on`, the offset is applied relative to all the available CPUs, while with `-pin inherit`, the offset is applied relative to the cores within the external affinity mask.

**-pinstride**

If `-pin on` or `-pin inherit`, specifies the stride in logical core numbers for the cores to which *mdrun* (page 258) should pin its threads. When running more than one instance of *mdrun* (page 258) on a node, use this option to avoid pinning threads from different *mdrun* (page 258) instances to the same core. Use the default, 0, to minimize the number of threads per physical core - this lets *mdrun* (page 258) manage the hardware-, OS- and configuration-specific details of how to map logical cores to physical cores. With `-pin on`, the stride is applied across all available CPUs, while with `-pin inherit`, the stride is applied only within the external affinity mask.

**-ddorder**

Can be set to "interleave," "pp_pme" or "cartesian." Defaults to "interleave," which means that any separate PME ranks will be mapped to MPI ranks in an order like PP, PP, PME, PP, PP, PME, etc. This generally makes the best use of the available hardware. "pp_pme" maps all PP ranks first, then all PME ranks. "cartesian" is a special-purpose mapping generally useful only on special torus networks with accelerated global communication for Cartesian communicators. Has no effect if there are no separate PME ranks.

**-nb**

Used to set where to execute the short-range non-bonded interactions. Can be set to "auto," "cpu" or "gpu." Defaults to "auto," which uses a compatible GPU if available. Setting "cpu" requires that no GPU is used. Setting "gpu" requires that a compatible GPU is available and will be used.

**-pme**

Used to set where to execute the long-range non-bonded interactions. Can be set to "auto," "cpu" or "gpu." Defaults to "auto," which uses a compatible GPU if available. Setting "gpu" requires that a compatible GPU is available. Multiple PME ranks are not supported with PME on GPU, so if a GPU is used for the PME calculation -npme must be set to 1.

**-bonded**

Used to set where to execute the bonded interactions that are part of the PP workload for a domain. Can be set to "auto," "cpu" or "gpu." Defaults to "auto," which uses a compatible CUDA or SYCL GPU only when one is available, a GPU is handling short-ranged interactions, and the CPU is handling long-ranged interaction work (electrostatic or LJ). The work for the bonded interac-

tions takes place on the same GPU as the short-ranged interactions, and cannot be independently assigned. Setting "gpu" requires that a compatible GPU is available and will be used.

**-update**
Used to set where to execute update and constraints, when present. Can be set to "auto," "cpu" or "gpu." Defaults to "auto," which uses GPU when possible and falls back to CPU. Setting "gpu" requires that a compatible GPU is available. Update and constraints on a GPU are currently not supported with mass and constraints free-energy perturbation, virtual sites, Ewald surface correction, replica exchange, constraint pulling, orientation restraints and computational electrophysiology. The combination of constraints and domain decomposition is only supported when all constraints are within so-called "update groups" (see *Avoiding communication for constraints* (page 123)).

**-gpu_id**
A string that specifies the ID numbers of the GPUs that are available to be used by ranks on each node. For example, "12" specifies that the GPUs with IDs 1 and 2 (as reported by the GPU runtime) can be used by *mdrun* (page 258). This is useful when sharing a node with other computations, or if a GPU that is dedicated to a display should not be used by GROMACS. Without specifying this parameter, *mdrun* (page 258) will utilize all GPUs. When many GPUs are present, a comma may be used to separate the IDs, so "12,13" would make GPUs 12 and 13 available to *mdrun* (page 258). It could be necessary to use different GPUs on different nodes of a simulation, in which case the environment variable `GMX_GPU_ID` can be set differently for the ranks on different nodes to achieve that result. In GROMACS versions preceding 2018 this parameter used to specify both GPU availability and GPU task assignment. The latter is now done with the `-gputasks` parameter.

**-gputasks**
A string that specifies the ID numbers of the GPUs to be used by corresponding GPU tasks on this node. For example, "0011" specifies that the first two GPU tasks will use GPU 0, and the other two use GPU 1. When using this option, the number of ranks must be known to *mdrun* (page 258), as well as where tasks of different types should be run, such as by using `-nb gpu` - only the tasks which are set to run on GPUs count for parsing the mapping. See *Assigning tasks to GPUs* (page 128) for more details. Note that `-gpu_id` and `-gputasks` can not be used at the same time! In GROMACS versions preceding 2018 only a single type of GPU task ("PP") could be run on any rank. Now that there is some support for running PME on GPUs, the number of GPU tasks (and the number of GPU IDs expected in the `-gputasks` string) can actually be 3 for a single-rank simulation. The IDs still have to be the same in this case, as using multiple GPUs per single rank is not yet implemented. The order of GPU tasks per rank in the string is PP first, PME second. The order of ranks with different kinds of GPU tasks is the same by default, but can be influenced with the `-ddorder` option and gets quite complex when using multiple nodes. Note that the bonded interactions for a PP task may run on the same GPU as the short-ranged work, or on the CPU, which can be controlled with the `-bonded` flag. The GPU task assignment (whether manually set, or automated), will be reported in the *mdrun* (page 258) output on the first physical node of the simulation. For example:

```
gmx mdrun -gputasks 0001 -nb gpu -pme gpu -npme 1 -ntmpi 4
```

will produce the following output in the log file/terminal:

```
On host tcbl14 2 GPUs selected for this run.
Mapping of GPU IDs to the 4 GPU tasks in the 4 ranks on this␣
 ↪node:
PP:0,PP:0,PP:0,PME:1
```

In this case, 3 ranks are set by user to compute PP work on GPU 0, and 1 rank to compute PME on GPU 1. The detailed indexing of the GPUs is also reported in the log file.

For more information about GPU tasks, please refer to *Types of GPU tasks* (page 126).

**-pmefft**
Allows choosing whether to execute the 3D FFT computation on a CPU or GPU. Can be set to "auto," "cpu" or "gpu." When PME is offloaded to a GPU `-pmefft gpu` is the default, and the entire PME calculation is executed on the GPU. However, in some cases, e.g. with a relatively slow or older generation GPU combined with fast CPU cores in a run, moving some work off of the GPU back to the CPU by computing FFTs on the CPU can improve performance.

### Examples for mdrun on one node

```
gmx mdrun
```

Starts *mdrun* (page 258) using all the available resources. *mdrun* (page 258) will automatically choose a fairly efficient division into thread-MPI ranks, OpenMP threads and assign work to compatible GPUs. Details will vary with hardware and the kind of simulation being run.

```
gmx mdrun -nt 8
```

Starts *mdrun* (page 258) using 8 threads, which might be thread-MPI or OpenMP threads depending on hardware and the kind of simulation being run.

```
gmx mdrun -ntmpi 2 -ntomp 4
```

Starts *mdrun* (page 258) using eight total threads, with two thread-MPI ranks and four OpenMP threads per rank. You should only use these options when seeking optimal performance, and must take care that the ranks you create can have all of their OpenMP threads run on the same socket. The number of ranks should be a multiple of the number of sockets, and the number of cores per node should be a multiple of the number of threads per rank.

```
gmx mdrun -ntmpi 4 -nb gpu -pme cpu
```

Starts *mdrun* (page 258) using four thread-MPI ranks. The CPU cores available will be split evenly between the ranks using OpenMP threads. The long-range component of the forces are calculated on CPUs. This may be optimal on hardware where the CPUs are relatively powerful compared to the GPUs. The bonded part of force calculation will automatically be assigned to the GPU, since the long-range component of the forces are calculated on CPU(s).

```
gmx mdrun -ntmpi 1 -nb gpu -pme gpu -bonded gpu -update gpu
```

Starts *mdrun* (page 258) using a single thread-MPI rank that will use all available CPU cores. All interaction types that can run on a GPU will do so. This may be optimal on hardware where the CPUs are extremely weak compared to the GPUs.

```
gmx mdrun -ntmpi 4 -nb gpu -pme cpu -gputasks 0011
```

Starts *mdrun* (page 258) using four thread-MPI ranks, and maps them to GPUs with IDs 0 and 1. The CPU cores available will be split evenly between the ranks using OpenMP threads, with the first two ranks offloading short-range nonbonded force calculations to GPU 0, and the last two ranks offloading

to GPU 1. The long-range component of the forces are calculated on CPUs. This may be optimal on hardware where the CPUs are relatively powerful compared to the GPUs.

```
gmx mdrun -ntmpi 4 -nb gpu -pme gpu -npme 1 -gputasks 0001
```

Starts *mdrun* (page 258) using four thread-MPI ranks, one of which is dedicated to the long-range PME calculation. The first 3 threads offload their short-range non-bonded calculations to the GPU with ID 0, the 4th (PME) thread offloads its calculations to the GPU with ID 1.

```
gmx mdrun -ntmpi 4 -nb gpu -pme gpu -npme 1 -gputasks 0011
```

Similar to the above example, with 3 ranks assigned to calculating short-range non-bonded forces, and one rank assigned to calculate the long-range forces. In this case, 2 of the 3 short-range ranks offload their nonbonded force calculations to GPU 0. The GPU with ID 1 calculates the short-ranged forces of the 3rd short-range rank, as well as the long-range forces of the PME-dedicated rank. Whether this or the above example is optimal will depend on the capabilities of the individual GPUs and the system composition.

```
gmx mdrun -gpu_id 12
```

Starts *mdrun* (page 258) using GPUs with IDs 1 and 2 (e.g. because GPU 0 is dedicated to running a display). This requires two thread-MPI ranks, and will split the available CPU cores between them using OpenMP threads.

```
gmx mdrun -nt 6 -pin on -pinoffset 0 -pinstride 1
gmx mdrun -nt 6 -pin on -pinoffset 6 -pinstride 1
```

Starts two *mdrun* (page 258) processes, each with six total threads arranged so that the processes affect each other as little as possible by being assigned to disjoint sets of physical cores. Threads will have their affinities set to particular logical cores, beginning from the first and 7th logical cores, respectively. The above would work well on an Intel CPU with six physical cores and hyper-threading enabled. Use this kind of setup only if restricting *mdrun* (page 258) to a subset of cores to share a node with other processes. A word of caution: The mapping of logical CPUs/cores to physical cores may differ between operating systems. On Linux, `cat /proc/cpuinfo` can be examined to determine this mapping.

```
mpirun -np 2 gmx_mpi mdrun
```

When using a *gmx mdrun* (page 258) compiled with external MPI, this will start two ranks and as many OpenMP threads as the hardware and MPI setup will permit. If the MPI setup is restricted to one node, then the resulting *gmx mdrun* (page 258) will be local to that node.

```
gmx mdrun -ntmpi 8 -nb gpu -pme gpu -npme 1 -bonded gpu -update gpu
```

Starts *mdrun* (page 258) using eight thread-MPI ranks that will use all available CPU cores and GPUs. All interaction types that can run on a GPU will do so. A single GPU (and MPI rank) are dedicated to the long-range forces. This may be optimal on hardware using data-center GPUs.

```
mpirun -np 8 gmx_mpi mdrun -nb gpu -pme gpu -npme 2 -bonded gpu -
↪update gpu
```

Starts *mdrun* (page 258) using eight MPI ranks that will allow `mpirun` to decide on the mapping of CPU cores and GPUs to MPI ranks. All interaction types that can run on a GPU will do so. Two GPUs (and MPI ranks) are dedicated to the long-range forces, which will work only when GROMACS is

configured with a suitable library for a distributed GPU 3D-FFT (e.g. cuFFTMp or HeFFTe), and when the environment variable `GMX_GPU_PME_DECOMPOSITION` is set (to enable this run path, which is currently pending validation). This may be optimal on an HPC node with specialized interconnects like NVLINK.

### 3.9.6 Running mdrun on more than one node

This requires configuring GROMACS to build with an external MPI library. By default, this *mdrun* (page 258) executable is run with `gmx_mpi mdrun`. All of the considerations for running single-node *mdrun* (page 258) still apply, except that `-ntmpi` and `-nt` cause a fatal error, and instead the number of ranks is controlled by the MPI environment. Settings such as `-npme` are much more important when using multiple nodes. Configuring the MPI environment to produce one rank per core is generally good until one approaches the strong-scaling limit. At that point, using OpenMP to spread the work of an MPI rank over more than one core is needed to continue to improve absolute performance. The location of the scaling limit depends on the processor, presence of GPUs, network, and simulation algorithm, but it is worth measuring at around ~200 particles/core if you need maximum throughput.

There are further command-line parameters that are relevant in these cases.

**`-tunepme`**
: Defaults to "on." If "on," a simulation will optimize *`rcoulomb`* (page 60) and *`fourierspacing`* (page 62) parameters for PME by scaling both by the same value. This results in an equivalent model physics that has shifted computational load between ranks and/or GPUs and can be optimized to maximize throughput. Some *mdrun* (page 258) features are not compatible with this, and these ignore this option. Note that acceptable equivalence can depend on the use case; multi-replica simulations such as replica exchange can tune PME differently for different replicas and thus compute potential energies that are slightly different depending on the PME parameters in use by that replica, e.g. on a different discrete Fourier-space grid.

**`-dlb`**
: Can be set to "auto," "no," or "yes." Defaults to "auto." Doing Dynamic Load Balancing between MPI ranks is needed to maximize performance. This is particularly important for molecular systems with heterogeneous particle or interaction density. When a certain threshold for performance loss is exceeded, DLB activates and shifts particles between ranks to improve performance. If available, using `-bonded gpu` is expected to improve the ability of DLB to maximize performance. DLB is not compatible with GPU-resident parallelization (with `-update gpu`) and therefore it remains switched off in such simulations.

During the simulation, *gmx mdrun* (page 258) must communicate between all PP ranks to compute quantities such as kinetic energy for log file reporting, or perhaps temperature coupling. By default, this happens whenever necessary to honor several *mdp options* (page 51), so that the period between communication phases is the greatest common divisor of *`nstcalcenergy`* (page 56), *`nsttcouple`* (page 65), and *`nstpcouple`* (page 67).

Note that `-tunepme` has more effect when there is more than one *node*, because the cost of communication for the PP and PME ranks differs. It still shifts load between PP and PME ranks, but does not change the number of separate PME ranks in use.

Note also that `-dlb` and `-tunepme` can interfere with each other, so if you experience performance variation that could result from this, you may wish to tune PME separately, and run the result with `mdrun -notunepme -dlb yes`.

The *gmx tune_pme* (page 340) utility is available to search a wider range of parameter space, including making safe modifications to the *tpr* (page 586) file, and varying `-npme`. It is only aware of the number

of ranks created by the MPI environment, and does not explicitly manage any aspect of OpenMP during the optimization.

### Examples for mdrun on more than one node

The examples and explanations for for single-node *mdrun* (page 258) are still relevant, but `-ntmpi` is no longer the way to choose the number of MPI ranks.

```
mpirun -np 16 gmx_mpi mdrun
```

Starts *gmx mdrun* (page 258) with 16 ranks, which are mapped to the hardware by the MPI library, e.g. as specified in an MPI hostfile. The available cores will be automatically split among ranks using OpenMP threads, depending on the hardware and any environment settings such as `OMP_NUM_THREADS`.

```
mpirun -np 16 gmx_mpi mdrun -npme 5
```

Starts *gmx mdrun* (page 258) with 16 ranks, as above, and require that 5 of them are dedicated to the PME component.

```
mpirun -np 16 gmx_mpi mdrun -ntomp 2 -npme 6 -ntomp_pme 1
```

Starts *gmx mdrun* (page 258) with 16 ranks, as above, and require that six of them are dedicated to the PME component with one OpenMP thread each. The remaining ten do the PP component, with two OpenMP threads each.

```
mpirun -np 4 gmx_mpi mdrun -ntomp 6 -nb gpu -gputasks 00
```

Starts *gmx mdrun* (page 258) on a machine with two nodes, using four total ranks, each rank with six OpenMP threads, and both ranks on a node sharing GPU with ID 0.

```
mpirun -np 8 gmx_mpi mdrun -ntomp 3 -gputasks 0000
```

Using a same/similar hardware as above, starts *gmx mdrun* (page 258) on a machine with two nodes, using eight total ranks, each rank with three OpenMP threads, and all four ranks on a node sharing GPU with ID 0. This may or may not be faster than the previous setup on the same hardware.

```
mpirun -np 20 gmx_mpi mdrun -ntomp 4 -gputasks 00
```

Starts *gmx mdrun* (page 258) with 20 ranks, and assigns the CPU cores evenly across ranks each to one OpenMP thread. This setup is likely to be suitable when there are ten nodes, each with one GPU, and each node has two sockets each of four cores.

```
mpirun -np 10 gmx_mpi mdrun -gpu_id 1
```

Starts *gmx mdrun* (page 258) with 10 ranks, and assigns the CPU cores evenly across ranks each to one OpenMP thread. This setup is likely to be suitable when there are ten nodes, each with two GPUs, but another job on each node is using GPU 0. The job scheduler should set the affinity of threads of both jobs to their allocated cores, or the performance of *mdrun* (page 258) will suffer greatly.

```
mpirun -np 20 gmx_mpi mdrun -gpu_id 01
```

Starts *gmx mdrun* (page 258) with 20 ranks. This setup is likely to be suitable when there are ten nodes, each with two GPUs, but there is no need to specify `-gpu_id` for the normal case where all the GPUs on the node are available for use.

```
mpirun -np 16 gmx_mpi mdrun -nb gpu -pme gpu -npme 4 -bonded gpu -
↪update gpu
```

Starts *mdrun* (page 258) using 16 MPI ranks that will allow `mpirun` to decide on the mapping of CPU cores and GPUs to MPI ranks. Normally you would do this on e.g. two nodes each containing eight GPUs. All interaction types that can run on a GPU will do so. Four GPUs (and MPI ranks) are dedicated to the long-range forces, which will work only when GROMACS is configured with a suitable library for a distributed GPU 3D-FFT (e.g. cuFFTMp or HeFFTe), and when the environment variable `GMX_GPU_PME_DECOMPOSITION` is set (to enable this run path, which is currently pending validation). This may be optimal on HPC nodes with specialized interconnects like NVLINK.

### 3.9.7 Avoiding communication for constraints

Because of the very short time it takes to perform an MD step, in particular close to the scaling limit, any communication will have a negative effect on performance due to latency overhead and synchronization. Most of the communication can not be avoided, but sometimes one can completely avoid communication of coordinates for constraints. The points listed below will improve performance in general and can have a particularly strong effect at the scaling limit which is around ~100 atoms/core or ~10000 atoms/GPU. Simulations that need to be done as fast as possible, or strong-scaling benchmarks should be constructed with these points in mind.

When possible, one should avoid the use of `constraints = all-bonds` with P-LINCS. This not only requires a lot of communication, it also sets an artificial minimum on the size of domains. If you are using an atomistic force field and integrating with a time step of 2 fs, you can usually change to constraints `constraints = h-bonds` without changing other settings. These are actually the settings most force fields were parameterized with, so this is also scientifically better.

To completely avoid communication for constraints and/or to have the update run on a GPU when using domain decomposition, the system needs to support so-called "update groups" (or no constraints at all). Update groups are grouped of atoms that are moved as one group between domains, thereby avoiding the need to communicate for constraints or virtual sites involving only atoms within the group. Update groups are supported when all atoms involved in coupled constraints are coupled directly to one central atom and consecutively ordered, not interdispersed with non-constrained atoms. An example is a compactly described methyl group. For atomistic force fields with `constraints = h-bonds` this means in practice that in the topology hydrogens come adjacent to their connected heavy atom. In addition, when virtual sites are present, the constructing atoms should all be constrained together and the virtual site and constructing atoms should be consecutive, but the order does not matter. The TIP4P water model is an example of this. Whether or not update groups are used is noted in the log file. When they cannot be used, the reason for disabling them is also noted.

### 3.9.8 Finding out how to run mdrun better

The Wallcycle module is used for runtime performance measurement of *gmx mdrun* (page 258). At the end of the log file of each run, the "Real cycle and time accounting" section provides a table with runtime statistics for different parts of the *gmx mdrun* (page 258) code in rows of the table. The table contains columns indicating the number of ranks and threads that executed the respective part of the run, wall-time and cycle count aggregates (across all threads and ranks) averaged over the entire run. The last column also shows what percentage of the total runtime each row represents. Note that the *gmx mdrun* (page 258) timer resetting functionalities (`-resethway` and `-resetstep`) reset the performance counters and therefore are useful to avoid startup overhead and performance instability (e.g. due to load balancing) at the beginning of the run.

The performance counters are:

- Particle-particle during Particle mesh Ewald
- Domain decomposition
- Domain decomposition communication load
- Domain decomposition communication bounds
- Virtual site constraints
- Send X to Particle mesh Ewald
- Neighbor search
- Launch GPU operations
- Communication of coordinates
- Force
- Waiting + Communication of force
- Particle mesh Ewald
- PME redist. X/F
- PME spread
- PME gather
- PME 3D-FFT
- PME 3D-FFT Communication
- PME solve Lennard-Jones
- PME solve LJ
- PME solve Elec
- PME wait for particle-particle
- Wait + Receive PME force
- Wait GPU nonlocal
- Wait GPU local
- Wait PME GPU spread
- Wait PME GPU gather

- Reduce PME GPU Force
- Non-bonded position/force buffer operations
- Virtual site spread
- COM pull force
- AWH (accelerated weight histogram method)
- Write trajectory
- Update
- Constraints
- Communication of energies
- Enforced rotation
- Add rotational forces
- Position swapping
- Interactive MD
- MD Graph

As performance data is collected for every run, they are essential to assessing and tuning the performance of *gmx mdrun* (page 258) performance. Therefore, they benefit both developers as well as users of the program. The counters are an average of the time/cycles different parts of the simulation take, hence can not directly reveal fluctuations during a single run (although comparisons across multiple runs are still very useful).

Counters will appear in an MD log file only if the related parts of the code were executed during the *gmx mdrun* (page 258) run. There is also a special counter called "Rest" which indicates the amount of time not accounted for by any of the counters above. Therefore, a significant amount "Rest" time (more than a few percent) will often be an indication of parallelization inefficiency (e.g. serial code) and it is recommended to be reported to the developers.

An additional set of subcounters can offer more fine-grained inspection of performance. They are:

- Domain decomposition redistribution
- DD neighbor search grid + sort
- DD setup communication
- DD make topology
- DD make constraints
- DD topology other
- Neighbor search grid local
- NS grid non-local
- NS search local
- NS search non-local
- Bonded force
- Bonded-FEP force

- Restraints force
- Listed buffer operations
- Nonbonded pruning
- Nonbonded force
- Launch non-bonded GPU tasks
- Launch PME GPU tasks
- Ewald force correction
- Non-bonded position buffer operations
- Non-bonded force buffer operations

Subcounters are geared toward developers and have to be enabled during compilation. See *Build system overview* (page 750) for more information.

### 3.9.9 Running mdrun with GPUs

#### Types of GPU tasks

To better understand the later sections on different GPU use cases for calculation of *short range* (page 127), *PME* (page 127), *bonded interactions* (page 127) and *update and constraints* (page 127) we first introduce the concept of different GPU tasks. When thinking about running a simulation, several different kinds of interactions between the atoms have to be calculated (for more information please refer to the reference manual). The calculation can thus be split into several distinct parts that are largely independent of each other (hence can be calculated in any order, e.g. sequentially or concurrently), with the information from each of them combined at the end of time step to obtain the final forces on each atom and to propagate the system to the next time point. For a better understanding also please see the section on *domain decomposition* (page 111).

Of all calculations required for an MD step, GROMACS aims to optimize performance bottom-up for each step from the lowest level (SIMD unit, cores, sockets, accelerators, etc.). Therefore many of the individual computation units are highly tuned for the lowest level of hardware parallelism: the SIMD units. Additionally, with GPU accelerators used as *co-processors*, some of the work can be *offloaded*, that is calculated simultaneously/concurrently with the CPU on the accelerator device, with the result being communicated to the CPU. Right now, GROMACS supports GPU accelerator offload of two tasks: the short-range *nonbonded interactions in real space* (page 127), and *PME* (page 127).

GROMACS supports two major offload modes: force-offload and GPU-resident. The former involves offloading some of or all interaction calculations with integration on the CPU (hence requiring per-step data movement). In the GPU-resident mode by offloading integration and constraints (when used) less data movement is necessary.

The force-offload mode is the more broadly supported GPU-acceleration mode with short-range nonbonded offload supported on a wide range of GPU accelerators (NVIDIA, AMD, and Intel). This is compatible with the grand majority of the features and parallelization modes and can be used to scale to large machines. Simultaneously offloading both short-range nonbonded and long-range PME work to GPU accelerators has some restrictions in terms of feature and parallelization compatibility (please see the *section below* (page 127)). Offloading (most types of) bonded interactions is supported in CUDA and SYCL. The GPU-resident mode is supported with CUDA and SYCL, but it has additional limitations as described in *the GPU update section* (page 127).

### GPU computation of short range nonbonded interactions

Using the GPU for the short-ranged nonbonded interactions provides the majority of the available speed-up compared to run using only the CPU. Here, the GPU acts as an accelerator that can effectively parallelize this problem and thus reduce the calculation time.

### GPU accelerated calculation of PME

GROMACS allows offloading of the PME calculation to the GPU, to further reduce the load on the CPU and improve usage overlap between CPU and GPU. Here, the solving of PME will be performed in addition to the calculation of the short range interactions on the same GPU as the short range interactions.

### Known limitations

**Please note again the limitations outlined below!**

- Only a PME order of 4 is supported on GPUs.
- Multiple ranks (hence multiple GPUs) computing PME have limited support: experimental PME decomposition in hybrid mode (`-pmefft cpu`) with CUDA from the 2022 release and full GPU PME decomposition since the 2023 release with CUDA or SYCL, and since the 2026 release with HIP (when GROMACS is built with *cuFFTMp* (page 14) or *HeFFTe* (page 14)).
- Only dynamical integrators are supported (ie. leap-frog, Velocity Verlet, stochastic dynamics)
- LJ PME is not supported on GPUs.
- When GROMACS is built without a GPU FFT library (`-DGMX_GPU_FFT_LIBRARY=none`), only hybrid mode (`-pmefft cpu`) is supported.

### GPU accelerated calculation of bonded interactions (CUDA, SYCL, HIP)

GROMACS allows the offloading of the bonded part of the PP workload to a compatible GPU. This is treated as part of the PP work, and requires that the short-ranged non-bonded task also runs on a GPU. Typically, there is a performance advantage to offloading bonded interactions in particular when the amount of CPU resources per GPU is relatively little (either because the CPU is weak or there are few CPU cores assigned to a GPU in a run) or when there are other computations on the CPU. A typical case for the latter is free-energy calculations.

### GPU accelerated calculation of constraints and coordinate update (CUDA, SYCL, HIP)

GROMACS makes it possible to also perform the coordinate update and (if requested) constraint calculation on a GPU. This parallelization mode is referred to as "GPU-resident" as all force and coordinate data can remain resident on the GPU for a number of steps (typically between temperature/pressure coupling or neighbor searching steps). The GPU-resident mode allows executing all (supported) computation of a simulation step on the GPU. This has the benefit that there is less coupling between CPU host and GPU and on typical MD steps data does not need to be transferred between CPU and GPU in contrast to the force-offload scheme requires coordinates and forces to be transferred every step between the CPU and GPU. The GPU-resident scheme however is still able to carry out part of the computation on the CPU concurrently with GPU calculation. This helps supporting the broad range of GROMACS

features not all of which are ported to GPUs. At the same time, it also allows improving performance by making use of the otherwise mostly idle CPU. It can often be advantageous to move the bonded or PME calculation back to the CPU, but the details of this will depending on the relative performance if the CPU cores paired in a simulation with a GPU.

GPU-resident mode is enabled by default (when supported) with an automatic fallback to CPU update when the build configuration or simulation settings are incompatible with it. It is possible to change the default behaviour by setting the `GMX_FORCE_UPDATE_DEFAULT_CPU` environment variable. In this case simulations following the default behavior (ie. `-update auto`) will run the update on the CPU.

Using this parallelization mode is typically advantageous in cases where a fast GPU is used with a slower CPU, in particular if there is only single simulation assigned to a GPU. However, in typical throughput cases where multiple runs are assigned to each GPU, offloading everything, especially without moving back some of the work to the CPU can perform worse than the parallelization mode where only force computation is offloaded.

### Assigning tasks to GPUs

Depending on which tasks should be performed on which hardware, different kinds of calculations can be combined on the same or different GPUs, according to the information provided for running *mdrun* (page 258).

It is possible to assign the calculation of the different computational tasks to the same GPU, meaning that they will share the computational resources on the same device, or to different processing units that will each perform one task each.

One overview over the possible task assignments is given below:

GROMACS version 2018:

> Two different types of assignable GPU accelerated tasks are available, (short-range) non-bonded and PME. Each PP rank has a nonbonded task that can be offloaded to a GPU. If there is only one rank with a PME task (including if that rank is a PME-only rank), then that task can be offloaded to a GPU. Such a PME task can run wholly on the GPU, or have its latter stages run only on the CPU.
>
> Limitations are that PME on GPU does not support PME domain decomposition, so that only one PME task can be offloaded to a single GPU assigned to a separate PME rank, while the nonbonded can be decomposed and offloaded to multiple GPUs.

GROMACS version 2019:

> No new assignable GPU tasks are available, but any bonded interactions may run on the same GPU as the short-ranged interactions for a PP task. This can be influenced with the `-bonded` flag.

GROMACS version 2020:

> Update and constraints can run on the same GPU as the short-ranged nonbonded and bonded interactions for a PP task. This can be influenced with the `-update` flag.

GROMACS version 2021/2022:

> Communication and auxiliary tasks can also be offloaded in CUDA builds. In domain-decomposition halo exchange and PP-PME communication, instead of staging transfers between GPUs though the CPU, direct GPU–GPU communication is possible. As an auxiliary tasks for halo exchange data packing and unpacking is performed which is also offloaded to

the GPU. In the 2021 release this is supported with thread-MPI and from the 2022 release it is also supported using GPU-aware MPI. Direct GPU communication is not enabled by default and can be triggered using the `GMX_ENABLE_DIRECT_GPU_COMM` environment variable (will only have an effect on supported systems).

GROMACS version 2023:

Update now runs by default on the GPU with supported simulation settings; note that this is only available with CUDA and SYCL not with OpenCL.

PME decomposition support adds additional parallelization-related auxiliary GPU tasks including grid packing and reduction operations as well as distributed GPU FFT computation.

Experimental support for CUDA-graphs scheduling has been added, which supports most GPU-resident runs that do not require CPU force computation.

GROMACS version 2025:

Direct GPU communication is now enabled by default on supported setups. The feature can be disabled using the `GMX_DISABLE_DIRECT_GPU_COMM` environment variable.

### Performance considerations for GPU tasks

1) The performance balance depends on the speed and number of CPU cores you have vs the speed and number of GPUs you have.
2) The GPU-resident parallelization mode (with update/constraints offloaded) is less sensitive to the appropriate CPU-GPU balance than the force-offload mode.
3) With slow/old GPUs and/or fast/modern CPUs with many cores, it might make more sense to let the CPU do PME calculation, with the GPUs focused on the nonbonded calculation.
4) With fast/modern GPUs and/or slow/old CPUs with few cores, it generally helps to have the GPU do PME.
5) Offloading bonded work to a GPU will often not improve simulation performance as efficient CPU-based kernels can complete the bonded computation before the GPU is done with other offloaded work. Therefore, *gmx mdrun* (page 258) will default to no bonded offload when PME is offloaded. Typical cases where performance can improve with bonded offload are: with significant bonded work (e.g. pure lipid or mostly polymer systems with little solvent), with very few and/or slow CPU cores per GPU, or when the CPU does other computation (e.g. PME, free energy).
6) On most modern hardware GPU-resident mode (default) is faster than force-offload mode, although it may leave the CPU idle. Moving back the bonded work to the CPU (`-bonded cpu`) is a better way to make use of a fast CPU than leaving integration and constraints on the CPU. The only exception may be multi-simulations with a significant number of simulations assigned to each GPU.
7) Direct GPU communication will in most cases outperform staged communication (both with thread-MPI and MPI). Ideally it should be combined with GPU-resident mode to maximize the benefit.
8) The only way to know for sure which alternative is best for your machine is to test and check performance.

### Reducing overheads in GPU accelerated runs

In order for CPU cores and GPU(s) to execute concurrently, tasks are launched and executed asynchronously on the GPU(s) while the CPU cores execute non-offloaded force computation (like computing bonded forces or free energy computation). Asynchronous task launches are handled by the GPU device driver and require CPU involvement. Therefore, scheduling GPU tasks requires CPU resources that can compete with other CPU tasks and cause interference that could lead to slowdown.

Delays in CPU execution are caused by the latency of launching GPU tasks, an overhead that can become significant as simulation ns/day increases (i.e. with shorter wall-time per step). The cost of launching GPU work is measured by *gmx mdrun* (page 258) and reported in the performance summary section of the log file ("Launch PP GPU ops."/"Launch PME GPU ops." rows). A few percent of runtime spent in launching work is normal, but in fast-iterating and multi-GPU parallel runs, costs of 10% or larger can be observed. Whether this has a significant performance impact depends on how much work within the main MD step is assigned to the CPU. With most or all force computation offloaded, and when the CPU is not involved in communication (e.g. with thread-MPI and direct GPU communication enabled) it may be that large launch costs do not lead to large performance losses. However, when the CPU is assigned computation (e.g. in free energy or pull/AWH simulations) or MPI communication is launched from the CPU (even with GPU-aware MPI), the GPU launch cost will compete with other CPU work and therefore represent overheads. In general, a user can do little to avoid such overheads, but there are a few cases where tweaks can give performance benefits. In OpenCL runs, timing of GPU tasks is by default enabled and, while in most cases its impact is small, in fast runs performance can be affected. In these cases, when more than a few percent of "Launch GPU ops" time is observed, it is recommended to turn off timing by setting the `GMX_DISABLE_GPU_TIMING` environment variable. In parallel runs with many ranks sharing a GPU, launch overheads can also be reduced by starting fewer thread-MPI or MPI ranks per GPU; e.g. most often one rank per thread or core is not optimal. The CUDA graphs functionality (added in GROMACS 2023) targets reducing such overheads and improving GPU work scheduling efficiency and therefore it can provide significant improvements especially for small simulation systems running on fast GPUs. Since it is a new feature, in the 2023 release CUDA-graph support needs to be triggered using the `GMX_CUDA_GRAPH` environment variable.

The second type of overhead, interference of the GPU runtime or driver with CPU computation, is caused by the scheduling and coordination of GPU tasks. A separate GPU runtime/driver thread requires CPU resources which may compete with the concurrently running non-offloaded tasks (if present), potentially degrading the performance of this CPU work. To minimize the overhead it can be useful to leave at least one CPU hardware thread unused when launching *gmx mdrun* (page 258), especially on CPUs with high core counts and/or simultaneous multithreading enabled. E.g. on a machine with a 16-core CPU and 32 threads, try `gmx mdrun -ntomp 31 -pin on`. This will leave some CPU resources for the GPU task scheduling potentially reducing interference with CPU computation. Note that assigning fewer resources to *gmx mdrun* (page 258) CPU computation involves a tradeoff which, with many CPU cores per GPU, may not be significant, but in some cases (e.g. with multi-rank MPI runs) it may lead to complex resource assignment and may outweigh the benefits of reduced GPU scheduling overheads, so we recommend to test the alternatives before adopting such techniques.

### 3.9.10 Running the OpenCL version of mdrun

Currently supported hardware architectures are:

- GCN-based and CDNA-based AMD GPUs;
- NVIDIA GPUs prior to Volta;
- Intel iGPUs.

Make sure that you have the latest drivers installed. For AMD GPUs, the compute-oriented ROCm stack is recommended; alternatively, the AMDGPU-PRO stack is also compatible; using the outdated and unsupported `fglrx` proprietary driver and runtime is not recommended (but for certain older hardware that may be the only way to obtain support). In addition Mesa version 17.0 or newer with LLVM 4.0 or newer is also supported. For NVIDIA GPUs, using the proprietary driver is required as the open source nouveau driver (available in Mesa) does not provide the OpenCL support. For Intel integrated GPUs, the Neo driver is recommended.

The minimum OpenCL version required is unknown. See also the *known limitations* (page 131).

Devices from the AMD GCN architectures (all series) are compatible and regularly tested; NVIDIA Kepler and later (compute capability 3.0) are known to work, but before doing production runs always make sure that the GROMACS tests pass successfully on the hardware.

The OpenCL GPU kernels are compiled at run time. Hence, building the OpenCL program can take a few seconds, introducing a slight delay in the *gmx mdrun* (page 258) startup. This is not normally a problem for long production MD, but you might prefer to do some kinds of work, e.g. that runs very few steps, on just the CPU (e.g. see `-nb` above).

The same `-gpu_id` option (or `GMX_GPU_ID` environment variable) used to select CUDA or SYCL devices, or to define a mapping of GPUs to PP ranks, is used for OpenCL devices.

Some other *OpenCL management* (page 403) environment variables may be of interest to developers.

#### Known limitations of the OpenCL support

Limitations in the current OpenCL support of interest to GROMACS users:

- Intel integrated GPUs are supported. Intel CPUs are not supported. Set `-DGMX_GPU_NB_CLUSTER_SIZE=4` when compiling GROMACS to run on consumer Intel GPUs (as opposed to Ponte Vecchio / Data Center Max GPUs).
- Due to blocking behavior of some asynchronous task enqueuing functions in the NVIDIA OpenCL runtime, with the affected driver versions there is almost no performance gain when using NVIDIA GPUs. The issue affects NVIDIA driver versions up to 349 series, but it known to be fixed 352 and later driver releases.
- On NVIDIA GPUs the OpenCL kernels achieve much lower performance than the equivalent CUDA kernels due to limitations of the NVIDIA OpenCL compiler.
- On the NVIDIA Volta and Turing architectures the OpenCL code is known to produce incorrect results with driver version up to 440.x (most likely due to compiler issues). Runs typically fail on these architectures.

### 3.9.11 Running SYCL version of mdrun

Make sure that you have the latest drivers installed and check the *installation guide* (page 21) for the list of compatible hardware and software and the recommended compile-time options.

Please keep in mind the following environment variables that might be useful:

- When using oneAPI runtime:
  - `SYCL_CACHE_PERSISTENT=1`: enables caching of GPU kernels, reducing *gmx mdrun* (page 258) startup time.

In addition to `-gpu_id` option, backend-specific environment variables, like `ONEAPI_DEVICE_SELECTOR` or `ROCR_VISIBLE_DEVICES`, could be used to select GPUs.

### 3.9.12 Running HIP version of mdrun

Please ensure you have a recent version of the ROCm toolkit and check the *AMD HIP installation guide* (page 23).

If you are using CDNA hardware, please ensure that your GROMACS build has been configured to use 64-wide execution on the device. For using RDNA hardware, try to use VkFFT (the default) as the GMX_GPU_FFT_LIBRARY, as official support for RDNA is not universal in rocFFT, and could lead to the runtime refusing to use the device.

## 3.10 Common errors when using GROMACS

The vast majority of error messages generated by GROMACS are descriptive, informing the user where the exact error lies. Some errors that arise are noted below, along with more details on what the issue is and how to solve it.

### 3.10.1 Common errors during usage

#### Out of memory when allocating

The program has attempted to assign memory to be used in the calculation, but is unable to due to insufficient memory.

Possible solutions are:

- reduce the scope of the number of atoms selected for analysis.
- reduce the length of trajectory file being processed.
- in some cases confusion between Ångström and nm may lead to users generating a *pdb2gmx* (page 281) water box that is $10^3$ times larger than what they think it is (e.g. *gmx solvate* (page 319)).
- use a computer with more memory.
- install more memory in the computer.

The user should bear in mind that the cost in time and/or memory for various activities will scale with the number of atoms/groups/residues $N$ or the simulation length $T$ as order N, NlogN, or $N^2$ (or maybe

worse!) and the same for *T*, depending on the type of activity. If it takes a long time, have a think about what you are doing, and the underlying algorithm (see the Reference manual, man page, or use the -h flag for the utility), and see if there is something sensible you can do that has better scaling properties.

### 3.10.2 Errors in pdb2gmx

#### Residue 'XXX' not found in residue topology database

This means that the force field you have selected while running *pdb2gmx* (page 281) does not have an entry in the *residue database* (page 582) for XXX. The *residue database* (page 582) entry is necessary both for stand-alone molecules (e.g. formaldehyde) or a peptide (standard or non-standard). This entry defines the atom types, connectivity, bonded and non-bonded interaction types for the residue and is necessary to use *pdb2gmx* (page 281) to build a *top* (page 584) file. A *residue database* (page 582) entry may be missing simply because the database does not contain the residue at all, or because the name is different.

For new users, this error appears because they are running *pdb2gmx* (page 281) on a *PDB* (page 581) file they have, without consideration of the contents of the file. A *force field* (page 393) is not magical, it can only deal with molecules or residues (building blocks) that are provided in the *residue database* (page 582) or included otherwise.

If you want to use *pdb2gmx* (page 281) to automatically generate your topology, you have to ensure that the appropriate *rtp* (page 582) entry is present within the desired *force field* (page 393) and has the same name as the building block you are trying to use. If you call your molecule "HIS," then *pdb2gmx* (page 281) will try to build histidine, based on the `[ HIS ]` entry in the *rtp* (page 582) file, so it will look for the exact atomic entries for histidine, no more no less.

If you want a *topology* (page 584) for an arbitrary molecule, you cannot use *pdb2gmx* (page 281) (unless you build the *rtp* (page 582) entry yourself). You will have to build that entry by hand, or use another program (such as *x2top* (page 355) or one of the scripts contributed by users) to build the *top* (page 584) file.

If there is not an entry for this residue in the database, then the options for obtaining the force field parameters are:

- see if there is a different name being used for the residue in the *residue database* (page 582) and rename as appropriate,
- parameterize the residue / molecule yourself (lots of work, even for an expert),
- find a *topology file* (page 584) for the molecule, convert it to an *itp* (page 578) file and include it in your *top* (page 584) file,
- use another *force field* (page 393) which has parameters available for this,
- search the primary literature for publications for parameters for the residue that are consistent with the force field that is being used.

Once you have determined the parameters and topology for your residue, see *adding a residue to a force field* (page 407) for instructions on how to proceed.

### Long bonds and/or missing atoms

There are probably atoms missing earlier in the *pdb* (page 581) file which makes *pdb2gmx* (page 281) go crazy. Check the screen output of *pdb2gmx* (page 281), as it will tell you which one is missing. Then add the atoms in your *pdb* (page 581) file, energy minimization will put them in the right place, or fix the side chain with e.g. the WHAT IF program.

### Chain identifier 'X' was used in two non-sequential blocks

This means that within the *coordinate file* (page 573) fed to *pdb2gmx* (page 281), the X chain has been split, possibly by the incorrect insertion of one molecule within another. The solution is simple: move the inserted molecule to a location within the file so that it is not splitting another molecule. This message may also mean that the same chain identifier has been used for two separate chains. In that case, rename the second chain to a unique identifier.

### WARNING: atom X is missing in residue XXX Y in the pdb file

Related to the long bonds/missing atoms error above, this error is usually quite obvious in its meaning. That is, *pdb2gmx* (page 281) expects certain atoms within the given residue, based on the entries in the force field *rtp* (page 582) file. There are several cases to which this error applies:

- Missing hydrogen atoms; the error message may be suggesting that an entry in the *hdb* (page 578) file is missing. More likely, the nomenclature of your hydrogen atoms simply does not match what is expected by the *rtp* (page 582) entry. In this case, use `-ignh` to allow *pdb2gmx* (page 281) to add the correct hydrogens for you, or re-name the problematic atoms.
- A terminal residue (usually the N-terminus) is missing H atoms; this usually suggests that the proper `-ter` option has not been supplied or chosen properly. In the case of the *AMBER force fields* (page 49), nomenclature is typically the problem. N-terminal and C-terminal residues must be prefixed by N and C, respectively. For example, an N-terminal alanine should not be listed in the *pdb* (page 581) file as `ALA`, but rather `NALA`, as specified in the ffamber instructions.
- Atoms are simply missing in the structure file provided to *pdb2gmx* (page 281); look for `REMARK 465` and `REMARK 470` entries in the *pdb* (page 581) file. These atoms will have to be modeled in using external software. There is no GROMACS tool to re-construct incomplete models.

Contrary to what the error message says, the use of the option `-missing` is almost always inappropriate. The `-missing` option should only be used to generate specialized topologies for amino acid-like molecules to take advantage of *rtp* (page 582) entries. If you find yourself using `-missing` in order to generate a topology for a protein or nucleic acid, do not; the topology produced is likely physically unrealistic.

### Atom X in residue YYY not found in rtp entry

If you are attempting to assemble a topology using *pdb2gmx* (page 281), the atom names are expected to match those found in the *rtp* (page 582) file that define the building block(s) in your structure. In most cases, the problem arises from a naming mismatch, so simply re-name the atoms in your *coordinate file* (page 573) appropriately. In other cases, you may be supplying a structure that has residues that do not conform to the expectations of the *force field* (page 393), in which case you should investigate why such a difference is occurring and make a decision based on what you find - use a different *force field* (page 393), manually edit the structure, etc.

#### No force fields found (files with name 'forcefield.itp' in subdirectories ending on '.ff')

This means your environment is not configured to use GROMACS properly, because *pdb2gmx* (page 281) cannot find its databases of forcefield information. This could happen because a GROMACS installation was moved from one location to another. Either follow the instructions about *Getting access to GROMACS after installation* (page 30) or re-install GROMACS before doing so.

### 3.10.3 Errors in grompp

#### Found a second defaults directive file

This is caused by the `[defaults]` directive appearing more than once in the *topology* (page 584) or *force field* (page 393) files for the system - it can only appear once. A typical cause of this is a second defaults being set in an included *topology* (page 584) file, *itp* (page 578), that has been sourced from somewhere else. For specifications on how the topology files work, see the reference manual, Section 5.6.:

```
[ defaults ]
; nbfunc comb-rule gen-pairs fudgeLJ fudgeQQ
1       1       no       1.0       1.0
```

One solution is to simply comment out (or delete) the lines of code out in the file where it is included for the second time i.e.,:

```
;[ defaults ]
; nbfunc comb-rule gen-pairs fudgeLJ fudgeQQ
;1       1       no       1.0       1.0
```

A better approach to finding a solution is to re-think what you are doing. The `[defaults]` directive should only be appearing at the top of your *top* (page 584) file where you choose the *force field* (page 393). If you are trying to mix two *force fields* (page 393), then you are asking for trouble. If a molecule *itp* (page 578) file tries to choose a force field, then whoever produced it is asking for trouble.

#### Invalid order for directive xxx

The directives in the .top and .itp files have rules about the order in which they can appear, and this error is seen when the order is violated. Consider the examples and discussion in chapter 5 of the reference manual, and/or from tutorial material. The *include file mechanism* (page 38) cannot be used to `#include` a file in just any old location, because they contain directives and these have to be properly placed.

In particular, `Invalid order for directive defaults` is a result of defaults being set in the *topology* (page 584) or *force field* (page 393) files in the inappropriate location; the `[defaults]` section can only appear once and must be the first directive in the *topology* (page 584). The `[defaults]` directive is typically present in the *force field* (page 393) file (forcefield.itp), and is added to the *topology* (page 584) when you `#include` this file in the system topology.

If the directive in question is `[atomtypes]` (which is the most common source of this error) or any other bonded or nonbonded `[*types]` directive, typically the user is adding some non-standard species (ligand, solvent, etc) that introduces new atom types or parameters into the system. As indicated above,

these new types and parameters must appear before any `[moleculetype]` directive. The *force field* (page 393) has to be fully constructed before any molecules can be defined.

### Atom index n in position_restraints out of bounds

A common problem is placing position restraint files for multiple molecules out of order. Recall that a position restraint *itp* (page 578) file containing a `[ position_restraints ]` block can only belong to the `[ moleculetype ]` block that contains it. For example:

WRONG:

```
#include "topol_A.itp"
#include "topol_B.itp"
#include "ligand.itp"

#ifdef POSRES
#include "posre_A.itp"
#include "posre_B.itp"
#include "ligand_posre.itp"
#endif
```

RIGHT:

```
#include "topol_A.itp"
#ifdef POSRES
#include "posre_A.itp"
#endif

#include "topol_B.itp"
#ifdef POSRES
#include "posre_B.itp"
#endif

#include "ligand.itp"
#ifdef POSRES
#include "ligand_posre.itp"
#endif
```

Further, the atom index of each `[position_restraint]` must be relative to the `[moleculetype]`, not relative to the system (because the parsing has not reached `[molecules]` yet, there is no such concept as "system"). So you cannot use the output of a tool like *genrestr* (page 228) blindly (as `genrestr -h` warns).

### System has non-zero total charge

Notifies you that counter-ions may be required for the system to neutralize the charge or there may be problems with the topology.

If the charge is not very close to an integer, then this indicates that there is a problem with the *topology* (page 584). If *pdb2gmx* (page 281) has been used, then look at the right-hand comment column of the atom listing, which lists the cumulative charge. This should be an integer after every residue (and/or charge group where applicable). This will assist in finding the residue where things start departing from integer values. Also check the terminal capping groups that have been used.

If the charge is already close to an integer, then the difference is caused by *rounding errors* (page 405) and not a major problem.

Note for PME users: It is possible to use a uniform neutralizing background charge in PME to compensate for a system with a net background charge. This may however, especially for non-homogeneous systems, lead to unwanted artifacts, as shown in *181* (page 692) (http://pubs.acs.org/doi/abs/10.1021/ct400626b). Nevertheless, it is a standard practice to actually add counter-ions to make the system net neutral.

### Incorrect number of parameters

Look at the *topology* (page 584) file for the system. You have not given enough parameters for one of the bonded definitions. Sometimes this also occurs if you have mangled the *Include File Mechanism* (page 38) or the topology file format (see: reference manual Chapter 5) when you edited the file.

### Number of coordinates in coordinate file does not match topology

This is pointing out that, based on the information provided in the *topology* (page 584) file, *top* (page 584), the total number of atoms or particles within the system does not match exactly with what is provided within the *coordinate file* (page 573), often a *gro* (page 577) or a *pdb* (page 581).

The most common reason for this is simply that the user has failed to update the topology file after solvating or adding additional molecules to the system, or made a typographical error in the number of one of the molecules within the system. Ensure that the end of the topology file being used contains something like the following, that matches exactly with what is within the coordinate file being used, in terms of both numbers and order of the molecules:

```
[ molecules ]
; Compound    #mol
Protein       1
SOL           10189
NA+           10
```

### Fatal error: No such moleculetype XXX

Each type of molecule in your `[ molecules ]` section of your *top* (page 584) file must have a corresponding `[ moleculetype ]` section defined previously, either in the *top* (page 584) file or an *included* (page 38) *itp* (page 578) file. See the reference manual section 5.6.1 for the syntax description. Your *top* (page 584) file does not have such a definition for the indicated molecule. Check the contents of the relevant files, how you have named your molecules, and how you have tried to refer to them later. Pay attention to the status of `#ifdef` and / or `#include` statements.

### T-Coupling group XXX has fewer than 10% of the atoms

It is possible to specify separate *thermostats* (page 387) (temperature coupling groups) for every molecule type within a simulation. This is a particularly bad practice employed by many new users to molecular dynamics simulations. Doing so is a bad idea, as you can introduce errors and artifacts that are hard to predict. In many cases it is best to have all molecules within a single group, using the default `System` group. If separate coupling groups are required to avoid the `hot-solvent, cold-solute` problem, then ensure that they are of `sufficient size` and combine molecule types that appear together within the simulation. For example, for a protein in water with counter-ions, one would likely want to use `Protein` and `Non-Protein`.

### The cut-off length is longer than half the shortest box vector or longer than the smallest box diagonal element. Increase the box size or decrease rlist

This error is generated in the cases as noted within the message. The dimensions of the box are such that an atom will interact with itself (when using periodic boundary conditions), thus violating the minimum image convention. Such an event is totally unrealistic and will introduce some serious artefacts. The solution is again what is noted within the message, either increase the size of the simulation box so that it is at an absolute minimum twice the cut-off length in all three dimensions (take care here if are using pressure coupling, as the box dimensions will change over time and if they decrease even slightly, you will still be violating the minimum image convention) or decrease the cut-off length (depending on the *force field* (page 393) utilised, this may not be an option).

### Atom index (1) in bonds out of bounds

This kind of error looks like:

```
Fatal error:
[ file spc.itp, line 32 ]
Atom index (1) in bonds out of bounds (1-0).
This probably means that you have inserted topology
section "settles" in a part belonging to a different
molecule than you intended to. in that case move the
"settles" section to the right molecule.
```

This error is fairly self-explanatory. You should look at your *top* (page 584) file and check that all of the `[molecules]` sections contain all of the data pertaining to that molecule, and no other data. That is, you cannot `#include` another molecule type (*itp* (page 578) file) before the previous `[moleculetype]` has ended. Consult the examples in chapter 5 of the reference manual for information on the required ordering of the different `[sections]`. Pay attention to the contents of any files you have *included* (page 38) with `#include` directives.

#### XXX non-matching atom names

This error usually indicates that the order of the *topology* (page 584) file does not match that of the *coordinate file* (page 573). When running *grompp* (page 230), the program reads through the *topology* (page 584), mapping the supplied parameters to the atoms in the *coordinate* (page 573) file. If there is a mismatch, this error is generated. To remedy the problem, make sure that the contents of your `[ molecules ]` directive matches the exact order of the atoms in the coordinate file.

In a few cases, the error is harmless. Perhaps you are using a *coordinate* (page 573) file that has the old (pre-4.5) ion nomenclature. In this case, allowing *grompp* (page 230) to re-assign names is harmless. For just about any other situation, when this error comes up, **it should not be ignored**. Just because the `-maxwarn` option is available does not mean you should use it in the blind hope of your simulation working. It will almost certainly *blow up* (page 390).

#### Invalid line in coordinate file for atom X

This error arises if the format of the *gro* (page 577) file is broken in some way. The most common explanation is that the second line in the *gro* (page 577) file specifies an incorrect number of atoms, causing *grompp* (page 230) to continue searching for atoms but finding box vectors.

#### An input file contains a line longer than 4095 characters

This error is usually due to a problem with line endings, which are different in DOS/Windows and Unix/Linux/Mac. This can be addressed using separate tools, such as dos2unix or in most text editors. Reading directly from network file systems, such as Samba, may cause the same problems. In that case it is recommended to copy the files to a local file system and try again.

### 3.10.4 Errors in mdrun

#### Stepsize too small, or no change in energy. Converged to machine precision, but not to the requested $F_{max}$

This may not be an error as such. It is simply informing you that during the energy minimization process mdrun reached the limit possible to minimize the structure with your current parameters. It does not mean that the system has not been minimized fully, but in some situations that may be the case. If the system has a significant amount of water present, then an $E_{pot}$ of the order of $-10^5$ to $-10^6$ (in conjunction with an $F_{max}$ between 10 and 1000 kJ mol$^{-1}$ nm$^{-1}$) is typically a reasonable value for starting most MD simulations from the resulting structure. The most important result is likely the value of $F_{max}$, as it describes the slope of the potential energy surface, i.e. how far from an energy minimum your structure lies. Only for special purposes, such as normal mode analysis type of calculations, it may be necessary to minimize further. Further minimization may be achieved by using a different energy minimization method or by making use of double precision-enabled GROMACS.

### Energy minimization has stopped because the force on at least one atom is not finite

This likely indicates that (at least) two atoms are too close in the input coordinates, and the forces exerted on each other are greater in magnitude than can be expressed to the extent of the precision of GROMACS, and therefore minimization cannot proceed. It is sometimes possible to minimize systems that have infinite forces with the use of soft-core potentials, which scale down the magnitude of Lennard-Jones interactions with the use of the GROMACS free energy code. This approach is an accepted workflow for equilibration of some coarse-grained systems such as Martini.

### LINCS/SETTLE/SHAKE warnings

Sometimes, when running dynamics, *mdrun* (page 258) may suddenly stop (perhaps after writing several *pdb* (page 581) files) after a series of warnings about the constraint algorithms (e.g. LINCS, SETTLE or SHAKE) are written to the *log* (page 578) file. These algorithms often used to constrain bond lengths and/or angles. When a system is *blowing up* (page 390) (i.e. exploding due to diverging forces), the constraints are usually the first thing to fail. This does not necessarily mean you need to troubleshoot the constraint algorithm. Usually it is a sign of something more fundamentally wrong (physically unrealistic) with your system. See also the advice here about *diagnosing unstable systems* (page 391).

### 1-4 interaction not within cut-off

Some of your atoms have moved so two atoms separated by three bonds are separated by more than the cut-off distance. **This is BAD**. Most importantly, **do not increase your cut-off**! This error actually indicates that the atoms have very large velocities, which usually means that (part of) your molecule(s) is (are) *blowing up* (page 390). If you are using LINCS for constraints, you probably also already got a number of LINCS warnings. When using SHAKE this will give rise to a SHAKE error, which halts your simulation before the `1-4 not within cutoff` error can appear.

There can be a number of reasons for the large velocities in your system. If it happens at the beginning of the simulation, your system might be not equilibrated well enough (e.g. it contains some bad contacts). Try a(nother) round of energy minimization to fix this. Otherwise you might have a very high temperature, and/or a timestep that is too large. Experiment with these parameters until the error stops occurring. If this does not help, check the validity of the parameters in your *topology* (page 584)!

### Simulation running but no output

Not an error as such, but mdrun appears to be chewing up CPU time but nothing is being written to the output files. There are a number of reasons why this may occur:

- Your simulation might simply be (very) *slow* (page 107), and since output is buffered, it can take quite some time for output to appear in the respective files. If you are trying to fix some problems and you want to get output as fast as possible, you can set the environment variable `GMX_LOG_BUFFER` to 0.
- Something might be going wrong in your simulation, causing e.g. not-a-numbers (NAN) to be generated (these are the result of e.g. division by zero). Subsequent calculations with NAN's will generate floating point exceptions which slow everything down by orders of magnitude.
- You might have all `nst*` parameters (see your *mdp* (page 579) file) set to 0, this will suppress most output.

- Your disk might be full. Eventually this will lead to *mdrun* (page 258) crashing, but since output is buffered, it might take a while for mdrun to realize it cannot write.

### Can not do Conjugate Gradients with constraints

This means you cannot do energy minimization with the conjugate gradient algorithm if your topology has constraints defined. Please check the reference manual.

### Pressure scaling more than 1%

This error tends to be generated when the simulation box begins to oscillate (due to large pressures and / or small coupling constants), the system starts to resonate and *then crashes* (page 390). This can mean that the system is not equilibrated sufficiently before using pressure coupling. Therefore, better / more equilibration may fix the issue.

It is recommended to observe the system trajectory prior and during the crash. This may indicate if a particular part of the system / structure is the problem.

In some cases, if the system has been equilibrated sufficiently, this error can mean that the pressure coupling constant, `tau-p` (page 67), is too small (particularly when using the Berendsen weak coupling method). Increasing that value will slow down the response to pressure changes and may stop the resonance from occurring. You are also more likely to see this error if you use Parrinello-Rahman pressure coupling on a system that is not yet equilibrated - use the C-rescale method instead.

This error can also appear when using a timestep that is too large, e.g. 5 fs, in the absence of constraints and / or virtual sites.

### Range Checking error

This usually means your simulation is *blowing up* (page 390). Probably you need to do better energy minimization and/or equilibration and/or topology design.

### X particles communicated to PME node Y are more than a cell length out of the domain decomposition cell of their charge group

This is another way that *mdrun* (page 258) tells you your system is *blowing up* (page 390). If you have particles that are flying across the system, you will get this fatal error. The message indicates that some piece of your system is tearing apart (hence out of the "cell of their charge group"). Refer to the *Blowing Up* (page 390) page for advice on how to fix this issue.

### A charge group moved too far between two domain decomposition steps.

See information above.

#### Software inconsistency error: Some interactions seem to be assigned multiple times

See information above

#### There is no domain decomposition for n ranks that is compatible with the given box and a minimum cell size of x nm

This means you tried to run a parallel calculation, and when *mdrun* (page 258) tried to partition your simulation cell into chunks, it could not. The minimum cell size is controlled by the size of the largest charge group or bonded interaction and the largest of `rvdw`, `rlist` and `rcoulomb`, some other effects of bond constraints, and a safety margin. Thus it is not possible to run a small simulation with large numbers of processors. So, if *grompp* (page 230) warned you about a large charge group, pay attention and reconsider its size. *mdrun* (page 258) prints a breakdown of how it computed this minimum size in the *log* (page 578) file, so you can perhaps find a cause there.

If you did not think you were running a parallel calculation, be aware that from 4.5, GROMACS uses thread-based parallelism by default. To prevent this, give *mdrun* (page 258) the `-ntmpi 1` command line option. Otherwise, you might be using an MPI-enabled GROMACS and not be aware of the fact.

#### The X-size of the box times the triclinic skew factor is smaller than the number of DD cells times the smallest allowed cell size

This error occurs when running a parallel simulation where the simulation box is too small to be divided among the requested number of domain decomposition (DD) cells. GROMACS splits the simulation box into one cell per parallel rank, and each cell must be at least as large as the cutoff radius (plus a safety margin). When the box size along a given axis, multiplied by the triclinic skew factor, is smaller than the number of DD cells along that axis multiplied by the minimum allowed cell size, the simulation cannot proceed. The possible solutions are:

- Letting *mdrun* (page 258) choose the DD decomposition automatically by removing explicit `-dd` flags.
- Increasing the simulation box size.
- Reducing the number of MPI ranks or thread-MPI threads (`-ntmpi`).

## 3.11 Command-line reference

GROMACS includes many tools for preparing, running and analyzing molecular dynamics simulations. These are all structured as part of a single **gmx** wrapper binary, and invoked with commands like **gmx grompp**. or **gmx mdrun**. Documentation for these can be found at the respective sections below, as well as on man pages (e.g., *gmx-grompp(1)*) and with `gmx help` *command* or `gmx` *command* `-h`.

If you have installed an MPI version of GROMACS, by default the **gmx** binary is called **gmx_mpi** and you should adapt accordingly.

### 3.11.1 molecular dynamics simulation suite

#### Synopsis

```
gmx [-[no]h] [-[no]quiet] [-[no]version] [-[no]copyright] [-nice
↪<int>]
    [-[no]backup]
```

#### Description

GROMACS is a full-featured suite of programs to perform molecular dynamics simulations, i.e., to simulate the behavior of systems with hundreds to millions of particles using Newtonian equations of motion. It is primarily used for research on proteins, lipids, and polymers, but can be applied to a wide variety of chemical and biological research questions.

#### Options

Other options:

**-[no]h (no)**
Print help and quit

**-[no]quiet (no)**
Do not print common startup info or quotes

**-[no]version (no)**
Print extended version information and quit

**-[no]copyright (no)**
Print copyright information on startup

**-nice <int> (19)**
Set the nicelevel (default depends on command)

**-[no]backup (yes)**
Write backups if output files exist

#### gmx commands

The following commands are available. Please refer to their individual man pages or `gmx help <command>` for further details.

#### Trajectory analysis

***gmx-gangle(1)***
Calculate angles

***gmx-convert-trj(1)***
Converts between different trajectory types

***gmx-distance(1)***
Calculate distances between pairs of positions

***gmx-dssp(1)***
Calculate protein secondary structure via DSSP algorithm

***gmx-extract-cluster(1)***
Allows extracting frames corresponding to clusters from trajectory

***gmx-freevolume(1)***
Calculate free volume

***gmx-hbond(1)***
Compute and analyze hydrogen bonds.

***gmx-msd(1)***
Compute mean squared displacements

***gmx-pairdist(1)***
Calculate pairwise distances between groups of positions

***gmx-rdf(1)***
Calculate radial distribution functions

***gmx-sasa(1)***
Compute solvent accessible surface area

***gmx-scattering(1)***
Calculate small angle scattering profiles for SANS or SAXS

***gmx-select(1)***
Print general information about selections

***gmx-trajectory(1)***
Print coordinates, velocities, and/or forces for selections

***gmx-gyrate(1)***
Calculate radius of gyration of a molecule

### Generating topologies and coordinates

***gmx-editconf(1)***
Edit the box and write subgroups

***gmx-x2top(1)***
Generate a primitive topology from coordinates

***gmx-solvate(1)***
Solvate a system

***gmx-insert-molecules(1)***
Insert molecules into existing vacancies

***gmx-genconf(1)***
Multiply a conformation in 'random' orientations

***gmx-genion(1)***
Generate monoatomic ions on energetically favorable positions

***gmx-genrestr(1)***
Generate position restraints or distance restraints for index groups

***gmx-pdb2gmx(1)***
Convert coordinate files to topology and FF-compliant coordinate files

### Running a simulation

***gmx-grompp(1)***
Make a run input file

***gmx-mdrun(1)***
Perform a simulation, do a normal mode analysis or an energy minimization

***gmx-convert-tpr(1)***
Make a modified run-input file

### Viewing trajectories

***gmx-nmtraj(1)***
Generate a virtual oscillating trajectory from an eigenvector

### Processing energies

***gmx-enemat(1)***
Extract an energy matrix from an energy file

***gmx-energy(1)***
Writes energies to xvg files and display averages

***gmx-mdrun(1)***
(Re)calculate energies for trajectory frames with -rerun

### Converting files

***gmx-editconf(1)***
Convert and manipulates structure files

***gmx-eneconv(1)***
Convert energy files

***gmx-sigeps(1)***
Convert c6/12 or c6/cn combinations to and from sigma/epsilon

***gmx-trjcat(1)***
Concatenate trajectory files

***gmx-trjconv(1)***
Convert and manipulates trajectory files

***gmx-xpm2ps(1)***
Convert XPM (XPixelMap) matrices to postscript or XPM

### Tools

***gmx-analyze(1)***
: Analyze data sets

***gmx-awh(1)***
: Extract data from an accelerated weight histogram (AWH) run

***gmx-filter(1)***
: Frequency filter trajectories, useful for making smooth movies

***gmx-lie(1)***
: Estimate free energy from linear combinations

***gmx-pme_error(1)***
: Estimate the error of using PME with a given input file

***gmx-sham(1)***
: Compute free energies or other histograms from histograms

***gmx-spatial(1)***
: Calculate the spatial distribution function

***gmx-traj(1)***
: Plot x, v, f, box, temperature and rotational energy from trajectories

***gmx-tune_pme(1)***
: Time mdrun as a function of PME ranks to optimize settings

***gmx-wham(1)***
: Perform weighted histogram analysis after umbrella sampling

***gmx-check(1)***
: Check and compare files

***gmx-dump(1)***
: Make binary files human readable

***gmx-make_ndx(1)***
: Make index files

***gmx-mk_angndx(1)***
: Generate index files for 'gmx angle'

***gmx-trjorder(1)***
: Order molecules according to their distance to a group

***gmx-xpm2ps(1)***
: Convert XPM (XPixelMap) matrices to postscript or XPM

***gmx-report-methods(1)***
: Write short summary about the simulation setup to a text file and/or to the standard output.

### Distances between structures

***gmx-cluster(1)***
Cluster structures

***gmx-confrms(1)***
Fit two structures and calculates the RMSD

***gmx-rms(1)***
Calculate RMSDs with a reference structure and RMSD matrices

***gmx-rmsf(1)***
Calculate atomic fluctuations

### Distances in structures over time

***gmx-mindist(1)***
Calculate the minimum distance between two groups

***gmx-mdmat(1)***
Calculate residue contact maps

***gmx-polystat(1)***
Calculate static properties of polymers

***gmx-rmsdist(1)***
Calculate atom pair distances averaged with power -2, -3 or -6

### Mass distribution properties over time

***gmx-gyrate-legacy(1)***
Calculate the radius of gyration

***gmx-polystat(1)***
Calculate static properties of polymers

***gmx-rdf(1)***
Calculate radial distribution functions

***gmx-rotacf(1)***
Calculate the rotational correlation function for molecules

***gmx-rotmat(1)***
Plot the rotation matrix for fitting to a reference structure

***gmx-sans-legacy(1)***
Compute small angle neutron scattering spectra

***gmx-saxs-legacy(1)***
Compute small angle X-ray scattering spectra

***gmx-traj(1)***
Plot x, v, f, box, temperature and rotational energy from trajectories

***gmx-vanhove(1)***
Compute Van Hove displacement and correlation functions

### Analyzing bonded interactions

***gmx-angle(1)***
Calculate distributions and correlations for angles and dihedrals

***gmx-mk_angndx(1)***
Generate index files for 'gmx angle'

### Structural properties

***gmx-bundle(1)***
Analyze bundles of axes, e.g., helices

***gmx-clustsize(1)***
Calculate size distributions of atomic clusters

***gmx-disre(1)***
Analyze distance restraints

***gmx-hbond-legacy(1)***
Compute and analyze hydrogen bonds

***gmx-order(1)***
Compute the order parameter per atom for carbon tails

***gmx-principal(1)***
Calculate principal axes of inertia for a group of atoms

***gmx-rdf(1)***
Calculate radial distribution functions

***gmx-saltbr(1)***
Compute salt bridges

***gmx-sorient(1)***
Analyze solvent orientation around solutes

***gmx-spol(1)***
Analyze solvent dipole orientation and polarization around solutes

### Kinetic properties

***gmx-bar(1)***
Calculate free energy difference estimates through Bennett's acceptance ratio

***gmx-current(1)***
Calculate dielectric constants and current autocorrelation function

***gmx-dos(1)***
Analyze density of states and properties based on that

***gmx-dyecoupl(1)***
Extract dye dynamics from trajectories

***gmx-principal(1)***
Calculate principal axes of inertia for a group of atoms

***gmx-tcaf(1)***
Calculate viscosities of liquids

***gmx-traj(1)***
Plot x, v, f, box, temperature and rotational energy from trajectories

***gmx-vanhove(1)***
Compute Van Hove displacement and correlation functions

***gmx-velacc(1)***
Calculate velocity autocorrelation functions

### Electrostatic properties

***gmx-current(1)***
Calculate dielectric constants and current autocorrelation function

***gmx-dielectric(1)***
Calculate frequency dependent dielectric constants

***gmx-dipoles(1)***
Compute the total dipole plus fluctuations

***gmx-potential(1)***
Calculate the electrostatic potential across the box

***gmx-spol(1)***
Analyze solvent dipole orientation and polarization around solutes

***gmx-genion(1)***
Generate monoatomic ions on energetically favorable positions

### Protein-specific analysis

***gmx-chi(1)***
Calculate everything you want to know about chi and other dihedrals

***gmx-helix(1)***
Calculate basic properties of alpha helices

***gmx-helixorient(1)***
Calculate local pitch/bending/rotation/orientation inside helices

***gmx-rama(1)***
Compute Ramachandran plots

***gmx-wheel(1)***
Plot helical wheels

### Interfaces

***gmx-bundle(1)***
Analyze bundles of axes, e.g., helices

***gmx-density(1)***
Calculate the density of the system

***gmx-densmap(1)***
Calculate 2D planar or axial-radial density maps

***gmx-densorder(1)***
Calculate surface fluctuations

***gmx-h2order(1)***
Compute the orientation of water molecules

***gmx-hydorder(1)***
Compute tetrahedrality parameters around a given atom

***gmx-order(1)***
Compute the order parameter per atom for carbon tails

***gmx-potential(1)***
Calculate the electrostatic potential across the box

### Covariance analysis

***gmx-anaeig(1)***
Analyze the eigenvectors

***gmx-covar(1)***
Calculate and diagonalize the covariance matrix

***gmx-make_edi(1)***
Generate input files for essential dynamics sampling

### Normal modes

***gmx-anaeig(1)***
Analyze the normal modes

***gmx-nmeig(1)***
Diagonalize the Hessian for normal mode analysis

***gmx-nmtraj(1)***
Generate a virtual oscillating trajectory from an eigenvector

***gmx-nmens(1)***
Generate an ensemble of structures from the normal modes

***gmx-grompp(1)***
Make a run input file

***gmx-mdrun(1)***
Find a potential energy minimum and calculate the Hessian

### 3.11.2 gmx anaeig

#### Synopsis

```
gmx anaeig [-v [<.trr/.cpt/...>]] [-v2 [<.trr/.cpt/...>]]
           [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]]
           [-n [<.ndx>]] [-eig [<.xvg>]] [-eig2 [<.xvg>]]
           [-comp [<.xvg>]] [-rmsf [<.xvg>]] [-proj [<.xvg>]]
           [-2d [<.xvg>]] [-3d [<.gro/.g96/...>]]
           [-filt [<.xtc/.trr/...>]] [-extr [<.xtc/.trr/...>]]
           [-over [<.xvg>]] [-inpr [<.xpm>]] [-b <time>] [-e <time>]
           [-dt <time>] [-tu <enum>] [-[no]w] [-xvg <enum>]
           [-first <int>] [-last <int>] [-skip <int>] [-max <real>]
           [-nframes <int>] [-[no]split] [-[no]entropy]
           [-temp <real>] [-nevskip <int>]
```

#### Description

`gmx anaeig` analyzes eigenvectors. The eigenvectors can be of a covariance matrix (*gmx covar* (page 181)) or of a Normal Modes analysis (*gmx nmeig* (page 269)).

When a trajectory is projected on eigenvectors, all structures are fitted to the structure in the eigenvector file, if present, otherwise to the structure in the structure file. When no run input file is supplied, periodicity will not be taken into account. Most analyses are performed on eigenvectors `-first` to `-last`, but when `-first` is set to -1 you will be prompted for a selection.

`-comp`: plot the vector components per atom of eigenvectors `-first` to `-last`.

`-rmsf`: plot the RMS fluctuation per atom of eigenvectors `-first` to `-last` (requires `-eig`).

`-proj`: calculate projections of a trajectory on eigenvectors `-first` to `-last`. The projections of a trajectory on the eigenvectors of its covariance matrix are called principal components (pc's). It is often useful to check the cosine content of the pc's, since the pc's of random diffusion are cosines with the number of periods equal to half the pc index. The cosine content of the pc's can be calculated with the program *gmx analyze* (page 154).

`-2d`: calculate a 2d projection of a trajectory on eigenvectors `-first` and `-last`.

`-3d`: calculate a 3d projection of a trajectory on the first three selected eigenvectors.

`-filt`: filter the trajectory to show only the motion along eigenvectors `-first` to `-last`.

`-extr`: calculate the two extreme projections along a trajectory on the average structure and interpolate `-nframes` frames between them, or set your own extremes with `-max`. The eigenvector `-first` will be written unless `-first` and `-last` have been set explicitly, in which case all eigenvectors will be written to separate files. Chain identifiers will be added when writing a *.pdb* (page 581) file with two or three structures (you can use `rasmol -nmrpdb` to view such a *.pdb* (page 581) file).

### Overlap calculations between covariance analysis

**Note:** the analysis should use the same fitting structure

`-over`: calculate the subspace overlap of the eigenvectors in file `-v2` with eigenvectors `-first` to `-last` in file `-v`.

`-inpr`: calculate a matrix of inner-products between eigenvectors in files `-v` and `-v2`. All eigenvectors of both files will be used unless `-first` and `-last` have been set explicitly.

When `-v` and `-v2` are given, a single number for the overlap between the covariance matrices is generated. Note that the eigenvalues are by default read from the timestamp field in the eigenvector input files, but when `-eig`, or `-eig2` are given, the corresponding eigenvalues are used instead. The formulas are:

```
        difference = sqrt(tr((sqrt(M1) - sqrt(M2))^2))
normalized overlap = 1 - difference/sqrt(tr(M1) + tr(M2))
     shape overlap = 1 - sqrt(tr((sqrt(M1/tr(M1)) - sqrt(M2/
↪tr(M2)))^2))
```

where M1 and M2 are the two covariance matrices and tr is the trace of a matrix. The numbers are proportional to the overlap of the square root of the fluctuations. The normalized overlap is the most useful number, it is 1 for identical matrices and 0 when the sampled subspaces are orthogonal.

When the `-entropy` flag is given an entropy estimate will be computed based on the Quasiharmonic approach and based on Schlitter's formula.

### Options

Options to specify input files:

**`-v` [<.trr/.cpt/...>] (eigenvec.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**`-v2` [<.trr/.cpt/...>] (eigenvec2.trr) (Optional)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

**`-eig` [<.xvg>] (eigenval.xvg) (Optional)**
xvgr/xmgr file

**`-eig2` [<.xvg>] (eigenval2.xvg) (Optional)**
xvgr/xmgr file

Options to specify output files:

**`-comp` [<.xvg>] (eigcomp.xvg) (Optional)**
xvgr/xmgr file

**-rmsf [<.xvg>] (eigrmsf.xvg) (Optional)**
xvgr/xmgr file

**-proj [<.xvg>] (proj.xvg) (Optional)**
xvgr/xmgr file

**-2d [<.xvg>] (2dproj.xvg) (Optional)**
xvgr/xmgr file

**-3d [<.gro/.g96/...>] (3dproj.pdb) (Optional)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

**-filt [<.xtc/.trr/...>] (filtered.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-extr [<.xtc/.trr/...>] (extreme.pdb) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-over [<.xvg>] (overlap.xvg) (Optional)**
xvgr/xmgr file

**-inpr [<.xpm>] (inprod.xpm) (Optional)**
X PixMap compatible matrix file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-first <int> (1)**
First eigenvector for analysis (-1 is select)

**-last <int> (-1)**
Last eigenvector for analysis (-1 is till the last)

**-skip <int> (1)**
Only analyse every nr-th frame

**-max <real> (0)**
Maximum for projection of the eigenvector on the average structure, max=0 gives the extremes

**-nframes <int> (2)**
Number of frames for the extremes output

**-[no]split (no)**
Split eigenvector projections where time is zero

**-[no]entropy (no)**
Compute entropy according to the Quasiharmonic formula or Schlitter's method.

**-temp <real> (298.15)**
Temperature for entropy calculations

**-nevskip <int> (6)**
Number of eigenvalues to skip when computing the entropy due to the quasi harmonic approximation. When you do a rotational and/or translational fit prior to the covariance analysis, you get 3 or 6 eigenvalues that are very close to zero, and which should not be taken into account when computing the entropy.

### 3.11.3 gmx analyze

#### Synopsis

```
gmx analyze [-f [<.xvg>]] [-ac [<.xvg>]] [-msd [<.xvg>]] [-cc [<.
↪xvg>]]
            [-dist [<.xvg>]] [-av [<.xvg>]] [-ee [<.xvg>]]
            [-fitted [<.xvg>]] [-g [<.log>]] [-[no]w] [-xvg <enum>]
            [-[no]time] [-b <real>] [-e <real>] [-n <int>] [-[no]d]
            [-bw <real>] [-errbar <enum>] [-[no]integrate]
            [-aver_start <real>] [-[no]xydy] [-[no]regression]
            [-[no]luzar] [-temp <real>] [-fitstart <real>]
            [-fitend <real>] [-filter <real>] [-[no]power]
            [-[no]subav] [-[no]oneacf] [-acflen <int>]
            [-[no]normalize] [-P <enum>] [-fitfn <enum>]
            [-beginfit <real>] [-endfit <real>]
```

#### Description

`gmx analyze` reads an ASCII file and analyzes data sets. A line in the input file may start with a time (see option `-time`) and any number of *y*-values may follow. Multiple sets can also be read when they are separated by & (option `-n`); in this case only one *y*-value is read from each line. All lines starting with # and @ are skipped. All analyses can also be done for the derivative of a set (option `-d`).

All options, except for `-av` and `-power`, assume that the points are equidistant in time.

`gmx analyze` always shows the average and standard deviation of each set, as well as the relative deviation of the third and fourth cumulant from those of a Gaussian distribution with the same standard deviation.

Option `-ac` produces the autocorrelation function(s). Be sure that the time interval between data points is much shorter than the time scale of the autocorrelation.

Option `-cc` plots the resemblance of set i with a cosine of i/2 periods. The formula is:

```
2 (integral from 0 to T of y(t) cos(i pi t) dt)^2
/ integral from 0 to T of y^2(t) dt
```

This is useful for principal components obtained from covariance analysis, since the principal components of random diffusion are pure cosines.

Option `-msd` produces the mean square displacement(s).

Option `-dist` produces distribution plot(s).

Option `-av` produces the average over the sets. Error bars can be added with the option `-errbar`. The errorbars can represent the standard deviation, the error (assuming the points are independent) or the interval containing 90% of the points, by discarding 5% of the points at the top and the bottom.

Option `-ee` produces error estimates using block averaging. A set is divided in a number of blocks and averages are calculated for each block. The error for the total average is calculated from the variance between averages of the m blocks B_i as follows: error^2 = sum (B_i - <B>)^2 / (m*(m-1)). These errors are plotted as a function of the block size. Also an analytical block average curve is plotted, assuming that the autocorrelation is a sum of two exponentials. The analytical curve for the block average is:

```
f(t) = sigma``*``sqrt(2/T (  alpha
(tau_1 ((exp(-t/tau_1) - 1)
tau_1/t + 1)) +
(1-alpha) (tau_2
((exp(-t/tau_2) - 1) tau_2/t +
1)))),
```

where T is the total time. alpha, tau_1 and tau_2 are obtained by fitting f^2(t) to error^2. When the actual block average is very close to the analytical curve, the error is sigma``*``sqrt(2/T (a tau_1 + (1-a) tau_2)). The complete derivation is given in B. Hess, J. Chem. Phys. 116:209-217, 2002.

Option `-filter` prints the RMS high-frequency fluctuation of each set and over all sets with respect to a filtered average. The filter is proportional to cos(pi t/len) where t goes from -len/2 to len/2. len is supplied with the option `-filter`. This filter reduces oscillations with period len/2 and len by a factor of 0.79 and 0.33 respectively.

Option `-g` fits the data to the function given with option `-fitfn`.

Option `-power` fits the data to b t^a, which is accomplished by fitting to a t + b on log-log scale. All points after the first zero or with a negative value are ignored.

Option `-luzar` performs a Luzar & Chandler kinetics analysis on output from *gmx hbond* (page 237). The input file can be taken directly from `gmx hbond -ac`, and then the same result should be produced.

Option `-fitfn` performs curve fitting to a number of different curves that make sense in the context of molecular dynamics, mainly exponential curves. More information is in the manual. To check the output of the fitting procedure the option `-fitted` will print both the original data and the fitted function to a new data file. The fitting parameters are stored as comment in the output file.

### Options

Options to specify input files:

**`-f` [<.xvg>] (graph.xvg)**
xvgr/xmgr file

Options to specify output files:

**`-ac` [<.xvg>] (autocorr.xvg) (Optional)**
xvgr/xmgr file

**`-msd` [<.xvg>] (msd.xvg) (Optional)**
xvgr/xmgr file

**`-cc` [<.xvg>] (coscont.xvg) (Optional)**
xvgr/xmgr file

**`-dist` [<.xvg>] (distr.xvg) (Optional)**
xvgr/xmgr file

**`-av` [<.xvg>] (average.xvg) (Optional)**
xvgr/xmgr file

**`-ee` [<.xvg>] (errest.xvg) (Optional)**
xvgr/xmgr file

**`-fitted` [<.xvg>] (fitted.xvg) (Optional)**
xvgr/xmgr file

**`-g` [<.log>] (fitlog.log) (Optional)**
Log file

Other options:

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-[no]time` (yes)**
Expect a time in the input

**`-b` <real> (-1)**
First time to read from set

**`-e` <real> (-1)**
Last time to read from set

**`-n` <int> (1)**
Read this number of sets separated by &

**`-[no]d` (no)**
Use the derivative

**`-bw` <real> (0.1)**
Binwidth for the distribution

**`-errbar` <enum> (none)**
Error bars for `-av`: none, stddev, error, 90

**-[no]integrate (no)**
Integrate data function(s) numerically using trapezium rule

**-aver_start <real> (0)**
Start averaging the integral from here

**-[no]xydy (no)**
Interpret second data set as error in the y values for integrating

**-[no]regression (no)**
Perform a linear regression analysis on the data. If `-xydy` is set a second set will be interpreted as the error bar in the Y value. Otherwise, if multiple data sets are present a multilinear regression will be performed yielding the constant A that minimize chi^2 = (y - A_0 x_0 - A_1 x_1 - ... - A_N x_N)^2 where now Y is the first data set in the input file and x_i the others. Do read the information at the option `-time`.

**-[no]luzar (no)**
Do a Luzar and Chandler analysis on a correlation function and related as produced by *gmx hbond* (page 237). When in addition the `-xydy` flag is given the second and fourth column will be interpreted as errors in c(t) and n(t).

**-temp <real> (298.15)**
Temperature for the Luzar hydrogen bonding kinetics analysis (K)

**-fitstart <real> (1)**
Time (ps) from which to start fitting the correlation functions in order to obtain the forward and backward rate constants for HB breaking and formation

**-fitend <real> (60)**
Time (ps) where to stop fitting the correlation functions in order to obtain the forward and backward rate constants for HB breaking and formation. Only with `-gem`

**-filter <real> (0)**
Print the high-frequency fluctuation after filtering with a cosine filter of this length

**-[no]power (no)**
Fit data to: b t^a

**-[no]subav (yes)**
Subtract the average before autocorrelating

**-[no]oneacf (no)**
Calculate one ACF over all sets

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### 3.11.4 gmx angle

#### Synopsis

```
gmx angle [-f [<.xtc/.trr/...>]] [-n [<.ndx>]] [-od [<.xvg>]]
          [-ov [<.xvg>]] [-of [<.xvg>]] [-ot [<.xvg>]] [-oh [<.xvg>
↪]]
          [-oc [<.xvg>]] [-or [<.trr>]] [-b <time>] [-e <time>]
          [-dt <time>] [-[no]w] [-xvg <enum>] [-type <enum>]
          [-[no]all] [-binwidth <real>] [-[no]periodic]
          [-[no]chandler] [-[no]avercorr] [-acflen <int>]
          [-[no]normalize] [-P <enum>] [-fitfn <enum>]
          [-beginfit <real>] [-endfit <real>]
```

#### Description

`gmx angle` computes the angle distribution for a number of angles or dihedrals.

With option `-ov`, you can plot the average angle of a group of angles as a function of time. With the `-all` option, the first graph is the average and the rest are the individual angles.

With the `-of` option, `gmx angle` also calculates the fraction of trans dihedrals (only for ryckaert-bellemans dihedrals) as function of time, but this is probably only fun for a select few. An output file is only generated if `-type ryckaert-bellemans` is specified.

With option `-oc`, a dihedral correlation function is calculated.

It should be noted that the index file must contain atom triplets for angles or atom quadruplets for dihedrals. If this is not the case, the program will crash.

With option `-or`, a trajectory file is dumped containing cos and sin of selected dihedral angles, which subsequently can be used as input for a principal components analysis using *gmx covar* (page 181).

Option `-ot` plots when transitions occur between dihedral rotamers of multiplicity 3 and `-oh` records a histogram of the times between such transitions, assuming the input trajectory frames are equally spaced in time. Output files are only generated if `-type dihedral` is specified.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-n [<.ndx>] (angle.ndx)**
Index file

Options to specify output files:

**-od [<.xvg>] (angdist.xvg)**
xvgr/xmgr file

**-ov [<.xvg>] (angaver.xvg) (Optional)**
xvgr/xmgr file

**-of [<.xvg>] (dihfrac.xvg) (Optional)**
xvgr/xmgr file

**-ot [<.xvg>] (dihtrans.xvg) (Optional)**
xvgr/xmgr file

**-oh [<.xvg>] (trhisto.xvg) (Optional)**
xvgr/xmgr file

**-oc [<.xvg>] (dihcorr.xvg) (Optional)**
xvgr/xmgr file

**-or [<.trr>] (traj.trr) (Optional)**
Trajectory in portable xdr format

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-type <enum> (angle)**
Type of angle to analyse: angle, dihedral, improper, ryckaert-bellemans

**-[no]all (no)**
Plot all angles separately in the averages file, in the order of appearance in the index file.

**-binwidth <real> (1)**
binwidth (degrees) for calculating the distribution

**-[no]periodic (yes)**
Print dihedral angles modulo 360 degrees

**-[no]chandler (no)**
Use Chandler correlation function (N[trans] = 1, N[gauche] = 0) rather than cosine correlation function. Trans is defined as phi < -60 or phi > 60.

**-[no]avercorr (no)**
Average the correlation functions for the individual angles/dihedrals

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### Known Issues

- Counting transitions only works for dihedrals with multiplicity 3

## 3.11.5 gmx awh

### Synopsis

```
gmx awh [-f [<.edr>]] [-s [<.tpr>]] [-o [<.xvg>]] [-fric [<.xvg>]]
        [-b <time>] [-e <time>] [-[no]w] [-xvg <enum>] [-skip <int>]
        [-[no]more] [-[no]kt]
```

### Description

`gmx awh` extracts AWH data from an energy file. One or two files are written per AWH bias per time frame. The bias index, if more than one, is appended to the file, as well as the time of the frame. By default only the PMF is printed. With `-more` the bias, target and coordinate distributions are also printed, as well as the metric sqrt(det(friction_tensor)) normalized such that the average is 1. Option `-fric` prints all components of the friction tensor to an additional set of files.

### Options

Options to specify input files:

**-f [<.edr>] (ener.edr)**
Energy file

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

Options to specify output files:

**-o [<.xvg>] (awh.xvg)**
xvgr/xmgr file

**-fric [<.xvg>] (friction.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-skip <int> (0)**
Skip number of frames between data points

**-[no]more (no)**
Print more output

**-[no]kt (no)**
Print free energy output in units of kT instead of kJ/mol

### 3.11.6 gmx bar

#### Synopsis

```
gmx bar [-f [<.xvg> [...]]] [-g [<.edr> [...]]] [-o [<.xvg>]]
        [-oi [<.xvg>]] [-oh [<.xvg>]] [-[no]w] [-xvg <enum>]
        [-b <real>] [-e <real>] [-temp <real>] [-prec <int>]
        [-nbmin <int>] [-nbmax <int>] [-nbin <int>] [-[no]extp]
```

#### Description

`gmx bar` calculates free energy difference estimates through Bennett's acceptance ratio method (BAR). It also automatically adds series of individual free energies obtained with BAR into a combined free energy estimate.

Every individual BAR free energy difference relies on two simulations at different states: say state A and state B, as controlled by a parameter, lambda (see the *.mdp* (page 579) parameter `init_lambda`). The BAR method calculates a ratio of weighted average of the Hamiltonian difference of state B given state A and vice versa. The energy differences to the other state must be calculated explicitly during the simulation. This can be done with the *.mdp* (page 579) option `foreign_lambda`.

Input option `-f` expects multiple `dhdl.xvg` files. Two types of input files are supported:

- Files with more than one *y*-value. The files should have columns with dH/dlambda and Delta-lambda. The lambda values are inferred from the legends: lambda of the simulation from the legend of dH/dlambda and the foreign lambda values from the legends of Delta H
- Files with only one *y*-value. Using the `-extp` option for these files, it is assumed that the *y*-value is dH/dlambda and that the Hamiltonian depends linearly on lambda. The lambda value of the simulation is inferred from the subtitle (if present), otherwise from a number in the subdirectory in the file name.

The lambda of the simulation is parsed from `dhdl.xvg` file's legend containing the string 'dH', the foreign lambda values from the legend containing the capitalized letters 'D' and 'H'. The temperature is parsed from the legend line containing 'T ='.

The input option `-g` expects multiple *.edr* (page 576) files. These can contain either lists of energy differences (see the *.mdp* (page 579) option `separate_dhdl_file`), or a series of histograms (see the *.mdp* (page 579) options `dh_hist_size` and `dh_hist_spacing`). The temperature and lambda values are automatically deduced from the `ener.edr` file.

In addition to the *.mdp* (page 579) option `foreign_lambda`, the energy difference can also be extrapolated from the dH/dlambda values. This is done with the``-extp`` option, which assumes that the system's Hamiltonian depends linearly on lambda, which is not normally the case.

The free energy estimates are determined using BAR with bisection, with the precision of the output set with `-prec`. An error estimate taking into account time correlations is made by splitting the data into blocks and determining the free energy differences over those blocks and assuming the blocks are independent. The final error estimate is determined from the average variance over 5 blocks. A range of block numbers for error estimation can be provided with the options `-nbmin` and `-nbmax`.

`gmx bar` tries to aggregate samples with the same 'native' and 'foreign' lambda values, but always assumes independent samples. **Note** that when aggregating energy differences/derivatives with different sampling intervals, this is almost certainly not correct. Usually subsequent energies are correlated and different time intervals mean different degrees of correlation between samples.

The results are split in two parts: the last part contains the final results in kJ/mol, together with the error estimate for each part and the total. The first part contains detailed free energy difference estimates and phase space overlap measures in units of kT (together with their computed error estimate). The printed values are:

- lam_A: the lambda values for point A.
- lam_B: the lambda values for point B.
- DG: the free energy estimate.
- s_A: an estimate of the relative entropy of B in A.
- s_B: an estimate of the relative entropy of A in B.
- stdev: an estimate expected per-sample standard deviation.

The relative entropy of both states in each other's ensemble can be interpreted as a measure of phase space overlap: the relative entropy s_A of the work samples of lambda_B in the ensemble of lambda_A (and vice versa for s_B), is a measure of the 'distance' between Boltzmann distributions of the two states, that goes to zero for identical distributions. See Wu & Kofke, J. Chem. Phys. 123 084109 (2005) for more information.

The estimate of the expected per-sample standard deviation, as given in Bennett's original BAR paper: Bennett, J. Comp. Phys. 22, p 245 (1976). Eq. 10 therein gives an estimate of the quality of sampling (not directly of the actual statistical error, because it assumes independent samples).

To get a visual estimate of the phase space overlap, use the `-oh` option to write series of histograms, together with the `-nbin` option.

### Options

Options to specify input files:

**`-f` [<.xvg> [...]] (dhdl.xvg) (Optional)**
xvgr/xmgr file

**`-g` [<.edr> [...]] (ener.edr) (Optional)**
Energy file

Options to specify output files:

**`-o` [<.xvg>] (bar.xvg) (Optional)**
xvgr/xmgr file

**`-oi` [<.xvg>] (barint.xvg) (Optional)**
xvgr/xmgr file

**`-oh` [<.xvg>] (histogram.xvg) (Optional)**
xvgr/xmgr file

Other options:

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-b` <real> (0)**
Begin time for BAR

**`-e` <real> (-1)**
End time for BAR

**`-temp` <real> (-1)**
Temperature (K)

**`-prec` <int> (2)**
The number of digits after the decimal point

**`-nbmin` <int> (5)**
Minimum number of blocks for error estimation

**`-nbmax` <int> (5)**
Maximum number of blocks for error estimation

**`-nbin` <int> (100)**
Number of bins for histogram output

**`-[no]extp` (no)**
Whether to linearly extrapolate dH/dl values to use as energies

### 3.11.7 gmx bundle

#### Synopsis

```
gmx bundle [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>
↪]]
           [-ol [<.xvg>]] [-od [<.xvg>]] [-oz [<.xvg>]]
           [-ot [<.xvg>]] [-otr [<.xvg>]] [-otl [<.xvg>]]
           [-ok [<.xvg>]] [-okr [<.xvg>]] [-okl [<.xvg>]]
           [-oa [<.pdb>]] [-b <time>] [-e <time>] [-dt <time>]
           [-tu <enum>] [-xvg <enum>] [-na <int>] [-[no]z]
```

#### Description

`gmx bundle` analyzes bundles of axes. The axes can be for instance helix axes. The program reads two index groups and divides both of them in `-na` parts. The centers of mass of these parts define the tops and bottoms of the axes. Several quantities are written to file: the axis length, the distance and the z-shift of the axis mid-points with respect to the average center of all axes, the total tilt, the radial tilt and the lateral tilt with respect to the average axis.

With options `-ok`, `-okr` and `-okl` the total, radial and lateral kinks of the axes are plotted. An extra index group of kink atoms is required, which is also divided into `-na` parts. The kink angle is defined as the angle between the kink-top and the bottom-kink vectors.

With option `-oa` the top, mid (or kink when `-ok` is set) and bottom points of each axis are written to a *.pdb* (page 581) file each frame. The residue numbers correspond to the axis numbers. When viewing this file with Rasmol, use the command line option `-nmrpdb`, and type `set axis true` to display the reference axis.

#### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-ol` [<.xvg>] (bun_len.xvg)**
xvgr/xmgr file

**`-od` [<.xvg>] (bun_dist.xvg)**
xvgr/xmgr file

**`-oz` [<.xvg>] (bun_z.xvg)**
xvgr/xmgr file

**-ot [<.xvg>] (bun_tilt.xvg)**
xvgr/xmgr file

**-otr [<.xvg>] (bun_tiltr.xvg)**
xvgr/xmgr file

**-otl [<.xvg>] (bun_tiltl.xvg)**
xvgr/xmgr file

**-ok [<.xvg>] (bun_kink.xvg) (Optional)**
xvgr/xmgr file

**-okr [<.xvg>] (bun_kinkr.xvg) (Optional)**
xvgr/xmgr file

**-okl [<.xvg>] (bun_kinkl.xvg) (Optional)**
xvgr/xmgr file

**-oa [<.pdb>] (axes.pdb) (Optional)**
Protein data bank file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-na <int> (0)**
Number of axes

**-[no]z (no)**
Use the $z$-axis as reference instead of the average axis

### 3.11.8 gmx check

#### Synopsis

```
gmx check [-f [<.xtc/.trr/...>]] [-f2 [<.xtc/.trr/...>]] [-s1 [<.
↪tpr>]]
          [-s2 [<.tpr>]] [-c [<.tpr/.gro/...>]] [-e [<.edr>]]
          [-e2 [<.edr>]] [-n [<.ndx>]] [-m [<.tex>]] [-vdwfac <real>
↪]
          [-bonlo <real>] [-bonhi <real>] [-[no]rmsd] [-tol <real>]
          [-abstol <real>] [-[no]ab] [-lastener <string>]
```

## Description

`gmx check` reads a trajectory (*.tng* (page 584), *.trr* (page 586) or *.xtc* (page 588)), an energy file (*.edr* (page 576)) or an index file (*.ndx* (page 580)) and prints out useful information about them.

Option `-c` checks for presence of coordinates, velocities and box in the file, for close contacts (smaller than `-vdwfac` and not bonded, i.e. not between `-bonlo` and `-bonhi`, all relative to the sum of both Van der Waals radii) and atoms outside the box (these may occur often and are no problem). If velocities are present, an estimated temperature will be calculated from them.

If an index file, is given its contents will be summarized.

If both a trajectory and a *.tpr* (page 586) file are given (with `-s1`) the program will check whether the bond lengths defined in the tpr file are indeed correct in the trajectory. If not you may have non-matching files due to e.g. deshuffling or due to problems with virtual sites. With these flags, `gmx check` provides a quick check for such problems.

The program can compare two run input (*.tpr* (page 586)) files when both `-s1` and `-s2` are supplied. When comparing run input files this way, the default relative tolerance is reduced to 0.000001 and the absolute tolerance set to zero to find any differences not due to minor compiler optimization differences, although you can of course still set any other tolerances through the options. Similarly a pair of trajectory files can be compared (using the `-f2` option), or a pair of energy files (using the `-e2` option).

For free energy simulations the A and B state topology from one run input file can be compared with options `-s1` and `-ab`.

## Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-f2` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s1` [<.tpr>] (top1.tpr) (Optional)**
Portable xdr run input file

**`-s2` [<.tpr>] (top2.tpr) (Optional)**
Portable xdr run input file

**`-c` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-e` [<.edr>] (ener.edr) (Optional)**
Energy file

**`-e2` [<.edr>] (ener2.edr) (Optional)**
Energy file

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**-m [<.tex>] (doc.tex) (Optional)**
LaTeX file

Other options:

**-vdwfac <real> (0.8)**
Fraction of sum of VdW radii used as warning cutoff

**-bonlo <real> (0.4)**
Min. fract. of sum of VdW radii for bonded atoms

**-bonhi <real> (0.7)**
Max. fract. of sum of VdW radii for bonded atoms

**-[no]rmsd (no)**
Print RMSD for x, v and f

**-tol <real> (0.001)**
Relative tolerance for comparing real values defined as 2*(a-b)/(|a|+|b|)

**-abstol <real> (0.001)**
Absolute tolerance, useful when sums are close to zero.

**-[no]ab (no)**
Compare the A and B topology from one file

**-lastener <string>**
Last energy term to compare (if not given all are tested). It makes sense to go up until the Pressure.

### 3.11.9 gmx chi

#### Synopsis

```
gmx chi [-s [<.gro/.g96/...>]] [-f [<.xtc/.trr/...>]] [-ss [<.dat>]]
        [-o [<.xvg>]] [-p [<.pdb>]] [-jc [<.xvg>]] [-corr [<.xvg>]]
        [-g [<.log>]] [-ot [<.xvg>]] [-oh [<.xvg>]] [-rt [<.xvg>]]
        [-cp [<.xvg>]] [-b <time>] [-e <time>] [-dt <time>] [-[no]w]
        [-xvg <enum>] [-r0 <int>] [-rN <int>] [-[no]phi] [-[no]psi]
        [-[no]omega] [-[no]rama] [-[no]viol] [-[no]periodic]
        [-[no]all] [-[no]rad] [-[no]shift] [-binwidth <int>]
        [-core_rotamer <real>] [-maxchi <enum>] [-[no]normhisto]
        [-[no]ramomega] [-bfact <real>] [-[no]chi_prod] [-[no]HChi]
        [-bmax <real>] [-acflen <int>] [-[no]normalize] [-P <enum>]
        [-fitfn <enum>] [-beginfit <real>] [-endfit <real>]
```

#### Description

`gmx chi` computes phi, psi, omega, and chi dihedrals for all your amino acid backbone and sidechains. It can compute dihedral angle as a function of time, and as histogram distributions. The distributions `(histo-(dihedral)(RESIDUE).xvg)` are cumulative over all residues of each type.

If option `-corr` is given, the program will calculate dihedral autocorrelation functions. The function used is C(t) = <cos(chi(tau)) cos(chi(tau+t))>. The use of cosines rather than angles themselves, resolves the problem of periodicity. (Van der Spoel & Berendsen (1997), Biophys. J. 72, 2032-2041). Separate

files for each dihedral of each residue (`corr(dihedral)(RESIDUE)(nresnr).xvg`) are output, as well as a file containing the information for all residues (argument of `-corr`).

With option `-all`, the angles themselves as a function of time for each residue are printed to separate files `(dihedral)(RESIDUE)(nresnr).xvg`. These can be in radians or degrees.

A log file (argument `-g`) is also written. This contains

- information about the number of residues of each type.
- The NMR ^3J coupling constants from the Karplus equation.
- a table for each residue of the number of transitions between rotamers per nanosecond, and the order parameter S^2 of each dihedral.
- a table for each residue of the rotamer occupancy.

All rotamers are taken as 3-fold, except for omega and chi dihedrals to planar groups (i.e. chi_2 of aromatics, Asp and Asn; chi_3 of Glu and Gln; and chi_4 of Arg), which are 2-fold. "rotamer 0" means that the dihedral was not in the core region of each rotamer. The width of the core region can be set with `-core_rotamer`

The S^2 order parameters are also output to an *.xvg* (page 589) file (argument `-o` ) and optionally as a *.pdb* (page 581) file with the S^2 values as B-factor (argument `-p`). The total number of rotamer transitions per timestep (argument `-ot`), the number of transitions per rotamer (argument `-rt`), and the ^3J couplings (argument `-jc`), can also be written to *.xvg* (page 589) files. Note that the analysis of rotamer transitions assumes that the supplied trajectory frames are equally spaced in time.

If `-chi_prod` is set (and `-maxchi` > 0), cumulative rotamers, e.g. 1+9(chi_1-1)+3(chi_2-1)+ (chi_3-1) (if the residue has three 3-fold dihedrals and `-maxchi` >= 3) are calculated. As before, if any dihedral is not in the core region, the rotamer is taken to be 0. The occupancies of these cumulative rotamers (starting with rotamer 0) are written to the file that is the argument of `-cp`, and if the `-all` flag is given, the rotamers as functions of time are written to `chiproduct(RESIDUE)(nresnr).xvg` and their occupancies to `histo-chiproduct(RESIDUE)(nresnr).xvg`.

The option `-r` generates a contour plot of the average omega angle as a function of the phi and psi angles, that is, in a Ramachandran plot the average omega angle is plotted using color coding.

### Options

Options to specify input files:

**`-s` [<.gro/.g96/...>] (conf.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-ss` [<.dat>] (ssdump.dat) (Optional)**
Generic data file

Options to specify output files:

**`-o` [<.xvg>] (order.xvg)**
xvgr/xmgr file

**`-p` [<.pdb>] (order.pdb) (Optional)**
Protein data bank file

**-jc [<.xvg>] (Jcoupling.xvg)**
xvgr/xmgr file

**-corr [<.xvg>] (dihcorr.xvg) (Optional)**
xvgr/xmgr file

**-g [<.log>] (chi.log)**
Log file

**-ot [<.xvg>] (dihtrans.xvg) (Optional)**
xvgr/xmgr file

**-oh [<.xvg>] (trhisto.xvg) (Optional)**
xvgr/xmgr file

**-rt [<.xvg>] (restrans.xvg) (Optional)**
xvgr/xmgr file

**-cp [<.xvg>] (chiprodhisto.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-r0 <int> (1)**
starting residue

**-rN <int> (-1)**
last residue

**-[no]phi (no)**
Output for phi dihedral angles

**-[no]psi (no)**
Output for psi dihedral angles

**-[no]omega (no)**
Output for omega dihedrals (peptide bonds)

**-[no]rama (no)**
Generate phi/psi and chi_1/chi_2 Ramachandran plots

**-[no]viol (no)**
Write a file that gives 0 or 1 for violated Ramachandran angles

**-[no]periodic (yes)**
Print dihedral angles modulo 360 degrees

**-[no]all (no)**
Output separate files for every dihedral.

**-[no]rad (no)**
in angle vs time files, use radians rather than degrees.

**-[no]shift (no)**
Compute chemical shifts from phi/psi angles

**-binwidth <int> (1)**
bin width for histograms (degrees)

**-core_rotamer <real> (0.5)**
only the central -core_rotamer*(360/multiplicity) belongs to each rotamer (the rest is assigned to rotamer 0)

**-maxchi <enum> (0)**
calculate first ndih chi dihedrals: 0, 1, 2, 3, 4, 5, 6

**-[no]normhisto (yes)**
Normalize histograms

**-[no]ramomega (no)**
compute average omega as a function of phi/psi and plot it in an *.xpm* (page 587) plot

**-bfact <real> (-1)**
B-factor value for *.pdb* (page 581) file for atoms with no calculated dihedral order parameter

**-[no]chi_prod (no)**
compute a single cumulative rotamer for each residue

**-[no]HChi (no)**
Include dihedrals to sidechain hydrogens

**-bmax <real> (0)**
Maximum B-factor on any of the atoms that make up a dihedral, for the dihedral angle to be considered in the statistics. Applies to database work where a number of X-Ray structures is analyzed. -bmax <= 0 means no limit.

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### Known Issues

- N-terminal phi and C-terminal psi dihedrals are calculated in a non-standard way, using H-N-CA-C for phi instead of C(-)-N-CA-C, and N-CA-C-O for psi instead of N-CA-C-N(+). This causes (usually small) discrepancies with the output of other tools like *gmx rama* (page 290).
- Rotamers with multiplicity 2 are printed in `chi.log` as if they had multiplicity 3, with the 3rd (g(+)) always having probability 0

## 3.11.10 gmx cluster

### Synopsis

```
gmx cluster [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.
↪ndx>]]
            [-dm [<.xpm>]] [-om [<.xpm>]] [-o [<.xpm>]] [-g [<.log>
↪]]
            [-dist [<.xvg>]] [-ev [<.xvg>]] [-conv [<.xvg>]]
            [-sz [<.xvg>]] [-tr [<.xpm>]] [-ntr [<.xvg>]]
            [-clid [<.xvg>]] [-cl [<.xtc/.trr/...>]]
            [-clndx [<.ndx>]] [-b <time>] [-e <time>] [-dt <time>]
            [-tu <enum>] [-[no]w] [-xvg <enum>] [-[no]dista]
            [-nlevels <int>] [-cutoff <real>] [-[no]fit]
            [-max <real>] [-skip <int>] [-[no]av] [-wcl <int>]
            [-nst <int>] [-rmsmin <real>] [-method <enum>]
            [-minstruct <int>] [-[no]binary] [-M <int>] [-P <int>]
            [-seed <int>] [-niter <int>] [-nrandom <int>]
            [-kT <real>] [-[no]pbc]
```

### Description

`gmx cluster` can cluster structures using several different methods. Distances between structures can be determined from a trajectory or read from an *.xpm* (page 587) matrix file with the `-dm` option. RMS deviation after fitting or RMS deviation of atom-pair distances can be used to define the distance between structures.

single linkage: add a structure to a cluster when its distance to any element of the cluster is less than `cutoff`.

Jarvis Patrick: add a structure to a cluster when this structure and a structure in the cluster have each other as neighbors and they have a least `P` neighbors in common. The neighbors of a structure are the M closest structures or all structures within `cutoff`.

Monte Carlo: reorder the RMSD matrix using Monte Carlo such that the order of the frames is using the smallest possible increments. With this it is possible to make a smooth animation going from one structure to another with the largest possible (e.g.) RMSD between them, however the intermediate steps should be as small as possible. Applications could be to visualize a potential of mean force ensemble of simulations or a pulling simulation. Obviously the user has to prepare the trajectory well (e.g. by not superimposing frames). The final result can be inspect visually by looking at the matrix *.xpm* (page 587) file, which should vary smoothly from bottom to top.

diagonalization: diagonalize the RMSD matrix.

gromos: use algorithm as described in Daura *et al.* (*Angew. Chem. Int. Ed.* **1999**, *38*, pp 236-240). Count number of neighbors using cut-off, take structure with largest number of neighbors with all its neighbors as cluster and eliminate it from the pool of clusters. Repeat for remaining structures in pool.

When the clustering algorithm assigns each structure to exactly one cluster (single linkage, Jarvis Patrick and gromos) and a trajectory file is supplied, the structure with the smallest average distance to the others or the average structure or all structures for each cluster will be written to a trajectory file. When writing all structures, separate numbered files are made for each cluster.

Two output files are always written:

- `-o` writes the RMSD values in the upper left half of the matrix and a graphical depiction of the clusters in the lower right half When `-minstruct` = 1 the graphical depiction is black when two structures are in the same cluster. When `-minstruct` > 1 different colors will be used for each cluster.
- `-g` writes information on the options used and a detailed list of all clusters and their members.

Additionally, a number of optional output files can be written:

- `-dist` writes the RMSD distribution.
- `-ev` writes the eigenvectors of the RMSD matrix diagonalization.
- `-sz` writes the cluster sizes.
- `-tr` writes a matrix of the number transitions between cluster pairs.
- `-ntr` writes the total number of transitions to or from each cluster.
- `-clid` writes the cluster number as a function of time.
- `-clndx` writes the frame numbers corresponding to the clusters to the specified index file to be read into trjconv.
- `-cl` writes average (with option `-av`) or central structure of each cluster or writes numbered files with cluster members for a selected set of clusters (with option `-wcl`, depends on `-nst` and `-rmsmin`). The center of a cluster is the structure with the smallest average RMSD from all other structures of the cluster.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

**`-dm` [<.xpm>] (rmsd.xpm) (Optional)**
X PixMap compatible matrix file

Options to specify output files:

**-om [<.xpm>] (rmsd-raw.xpm)**
X PixMap compatible matrix file

**-o [<.xpm>] (rmsd-clust.xpm)**
X PixMap compatible matrix file

**-g [<.log>] (cluster.log)**
Log file

**-dist [<.xvg>] (rmsd-dist.xvg) (Optional)**
xvgr/xmgr file

**-ev [<.xvg>] (rmsd-eig.xvg) (Optional)**
xvgr/xmgr file

**-conv [<.xvg>] (mc-conv.xvg) (Optional)**
xvgr/xmgr file

**-sz [<.xvg>] (clust-size.xvg) (Optional)**
xvgr/xmgr file

**-tr [<.xpm>] (clust-trans.xpm) (Optional)**
X PixMap compatible matrix file

**-ntr [<.xvg>] (clust-trans.xvg) (Optional)**
xvgr/xmgr file

**-clid [<.xvg>] (clust-id.xvg) (Optional)**
xvgr/xmgr file

**-cl [<.xtc/.trr/...>] (clusters.pdb) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-clndx [<.ndx>] (clusters.ndx) (Optional)**
Index file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]dista (no)**
Use RMSD of distances instead of RMS deviation

**-nlevels <int> (40)**
Discretize RMSD matrix in this number of levels

**-cutoff <real> (0.1)**
RMSD cut-off (nm) for two structures to be neighbor

**-[no]fit (yes)**
Use least squares fitting before RMSD calculation

**-max <real> (-1)**
Maximum level in RMSD matrix

**-skip <int> (1)**
Only analyze every nr-th frame

**-[no]av (no)**
Write average instead of middle structure for each cluster

**-wcl <int> (0)**
Write the structures for this number of clusters to numbered files

**-nst <int> (1)**
Only write all structures if more than this number of structures per cluster

**-rmsmin <real> (0)**
minimum rms difference with rest of cluster for writing structures

**-method <enum> (linkage)**
Method for cluster determination: linkage, jarvis-patrick, monte-carlo, diagonalization, gromos

**-minstruct <int> (1)**
Minimum number of structures in cluster for coloring in the *.xpm* (page 587) file

**-[no]binary (no)**
Treat the RMSD matrix as consisting of 0 and 1, where the cut-off is given by `-cutoff`

**-M <int> (10)**
Number of nearest neighbors considered for Jarvis-Patrick algorithm, 0 is use cutoff

**-P <int> (3)**
Number of identical nearest neighbors required to form a cluster

**-seed <int> (0)**
Random number seed for Monte Carlo clustering algorithm (0 means generate)

**-niter <int> (10000)**
Number of iterations for MC

**-nrandom <int> (0)**
The first iterations for MC may be done complete random, to shuffle the frames

**-kT <real> (0.001)**
Boltzmann weighting factor for Monte Carlo optimization (zero turns off uphill steps)

**-[no]pbc (yes)**
PBC check

### 3.11.11 gmx clustsize

#### Synopsis

```
gmx clustsize [-f [<.xtc/.trr/...>]] [-s [<.tpr>]] [-n [<.ndx>]]
              [-o [<.xpm>]] [-ow [<.xpm>]] [-nc [<.xvg>]]
              [-mc [<.xvg>]] [-ac [<.xvg>]] [-hc [<.xvg>]]
              [-temp [<.xvg>]] [-mcn [<.ndx>]] [-b <time>] [-e <time>
↪]
              [-dt <time>] [-tu <enum>] [-[no]w] [-xvg <enum>]
              [-cut <real>] [-[no]mol] [-[no]pbc] [-nskip <int>]
              [-nlevels <int>] [-ndf <int>] [-rgblo <vector>]
              [-rgbhi <vector>]
```

#### Description

`gmx clustsize` computes the size distributions of molecular/atomic clusters in the gas phase. The output is given in the form of an *.xpm* (page 587) file. The total number of clusters is written to an *.xvg* (page 589) file.

When the `-mol` option is given clusters will be made out of molecules rather than atoms, which allows clustering of large molecules. In this case an index file would still contain atom numbers or your calculation will die with a SEGV.

When velocities are present in your trajectory, the temperature of the largest cluster will be printed in a separate *.xvg* (page 589) file assuming that the particles are free to move. If you are using constraints, please correct the temperature. For instance water simulated with SHAKE or SETTLE will yield a temperature that is 1.5 times too low. You can compensate for this with the `-ndf` option. Remember to take the removal of center of mass motion into account.

The `-mc` option will produce an index file containing the atom numbers of the largest cluster.

#### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr>] (topol.tpr) (Optional)**
Portable xdr run input file

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-o` [<.xpm>] (csize.xpm)**
X PixMap compatible matrix file

**`-ow` [<.xpm>] (csizew.xpm)**
X PixMap compatible matrix file

**-nc [<.xvg>] (nclust.xvg)**
xvgr/xmgr file

**-mc [<.xvg>] (maxclust.xvg)**
xvgr/xmgr file

**-ac [<.xvg>] (avclust.xvg)**
xvgr/xmgr file

**-hc [<.xvg>] (histo-clust.xvg)**
xvgr/xmgr file

**-temp [<.xvg>] (temp.xvg) (Optional)**
xvgr/xmgr file

**-mcn [<.ndx>] (maxclust.ndx) (Optional)**
Index file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-cut <real> (0.35)**
Largest distance (nm) to be considered in a cluster

**-[no]mol (no)**
Cluster molecules rather than atoms (needs *.tpr* (page 586) file)

**-[no]pbc (yes)**
Use periodic boundary conditions

**-nskip <int> (0)**
Number of frames to skip between writing

**-nlevels <int> (20)**
Number of levels of grey in *.xpm* (page 587) output

**-ndf <int> (-1)**
Number of degrees of freedom of the entire system for temperature calculation. If not set, the number of atoms times three is used.

**-rgblo <vector> (1 1 0)**
RGB values for the color of the lowest occupied cluster size

**-rgbhi <vector> (0 0 1)**
RGB values for the color of the highest occupied cluster size

### 3.11.12 gmx confrms

#### Synopsis

```
gmx confrms [-f1 [<.tpr/.gro/...>]] [-f2 [<.gro/.g96/...>]]
            [-n1 [<.ndx>]] [-n2 [<.ndx>]] [-o [<.gro/.g96/...>]]
            [-no [<.ndx>]] [-[no]w] [-[no]one] [-[no]mw] [-[no]pbc]
            [-[no]fit] [-[no]name] [-[no]label] [-[no]bfac]
```

#### Description

`gmx confrms` computes the root mean square deviation (RMSD) of two structures after least-squares fitting the second structure on the first one. The two structures do NOT need to have the same number of atoms, only the two index groups used for the fit need to be identical. With `-name` only matching atom names from the selected groups will be used for the fit and RMSD calculation. This can be useful when comparing mutants of a protein.

The superimposed structures are written to file. In a *.pdb* (page 581) file the two structures will be written as separate models (use `rasmol -nmrpdb`). Also in a *.pdb* (page 581) file, B-factors calculated from the atomic MSD values can be written with `-bfac`.

#### Options

Options to specify input files:

**-f1 [<.tpr/.gro/...>] (conf1.gro)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-f2 [<.gro/.g96/...>] (conf2.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**-n1 [<.ndx>] (fit1.ndx) (Optional)**
Index file

**-n2 [<.ndx>] (fit2.ndx) (Optional)**
Index file

Options to specify output files:

**-o [<.gro/.g96/...>] (fit.pdb)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

**-no [<.ndx>] (match.ndx) (Optional)**
Index file

Other options:

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-[no]one (no)**
Only write the fitted structure to file

**-[no]mw (yes)**
Mass-weighted fitting and RMSD

**-[no]pbc (no)**
Try to make molecules whole again

**-[no]fit (yes)**
Do least squares superposition of the target structure to the reference

**-[no]name (no)**
Only compare matching atom names

**-[no]label (no)**
Added chain labels A for first and B for second structure

**-[no]bfac (no)**
Output B-factors from atomic MSD values

### 3.11.13 gmx convert-tpr

#### Synopsis

```
gmx convert-tpr [-s [<.tpr/.gro/...>]] [-n [<.ndx>]]
                [-o [<.tpr/.gro/...>]] [-extend <time>] [-until <time>]
                [-nsteps <int>] [-[no]generate_velocities]
                [-velocity_temp <real>] [-velocity_seed <int>]
```

#### Description

`gmx convert-tpr` can edit run input files in three ways.

**1.** by modifying the number of steps in a run input file with options `-extend`, `-until` or `-nsteps` (nsteps=-1 means unlimited number of steps)

**2.** by creating a .tpx file for a subset of your original tpx file, which is useful when you want to remove the solvent from your .tpx file, or when you want to make e.g. a pure Calpha .tpx file. Note that you may need to use `-nsteps -1` (or similar) to get this to work. **WARNING: this .tpx file is not fully functional**.

**3.** by setting the charges of a specified group to zero. This is useful when doing free energy estimates using the LIE (Linear Interaction Energy) method.

#### Options

Options to specify input files:

**-s [<.tpr/.gro/...>] (topol.tpr)**
Run input file to modify: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
File containing additional index groups

Options to specify output files:

**-o [<.tpr/.gro/...>] (tprout.tpr) (Optional)**
Generated modified run input file: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

Other options:

**-extend <time> (0)**
Extend runtime by this amount (ps)

**-until <time> (0)**
Extend runtime until this ending time (ps)

**-nsteps <int> (0)**
Change the number of steps remaining to be made

**-[no]generate_velocities (no)**
Reassign velocities, using a generated seed unless one is explicitly set

**-velocity_temp <real> (300)**
Temperature to use when generating velocities

**-velocity_seed <int> (-1)**
Random seed for velocities. If value is -1, a new one is generated

### 3.11.14 gmx convert-trj

#### Synopsis

```
gmx convert-trj [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]]
                [-n [<.ndx>]] [-o [<.xtc/.trr/...>]] [-b <time>]
                [-e <time>] [-dt <time>] [-tu <enum>]
                [-fgroup <selection>] [-xvg <enum>] [-[no]rmpbc]
                [-[no]pbc] [-sf <file>] [-selrpos <enum>]
                [-select <selection>] [-vel <enum>] [-force <enum>]
                [-atoms <enum>] [-precision <int>] [-starttime <time>]
                [-timestep <time>] [-box <vector>]
```

#### Description

`gmx convert-trj` converts trajectory files between different formats. The module supports writing all GROMACS supported file formats from the supported input formats.

Included is also a selection of possible options to modify individual trajectory frames, including options to produce slimmer output files. It is also possible to replace the particle information stored in the input trajectory with those from a structure file

The module can also generate subsets of trajectories based on user supplied selections.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**`-o` [<.xtc/.trr/...>] (trajout.xtc)**
Output trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

Other options:

**`-b` <time> (0)**
First frame (ps) to read from trajectory

**`-e` <time> (0)**
Last frame (ps) to read from trajectory

**`-dt` <time> (0)**
Only use frame if t MOD dt == first time (ps)

**`-tu` <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**`-fgroup` <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**`-xvg` <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**`-[no]rmpbc` (yes)**
Make molecules whole for each frame

**`-[no]pbc` (yes)**
Use periodic boundary conditions for distance calculation

**`-sf` <file>**
Provide selections from files

**`-selrpos` <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**`-select` <selection>**
Selection of particles to write to the file

**`-vel` <enum> (preserved-if-present)**
Save velocities from frame if possible: preserved-if-present, always, never

**-force <enum> (preserved-if-present)**
Save forces from frame if possible: preserved-if-present, always, never

**-atoms <enum> (preserved-if-present)**
Decide on providing new atom information from topology or using current frame atom information: preserved-if-present, always-from-structure, never, always

**-precision <int> (3)**
Set output precision to custom value

**-starttime <time> (0)**
Change start time for first frame

**-timestep <time> (0)**
Change time between different frames

**-box <vector>**
New diagonal box vector for output frame

### 3.11.15 gmx covar

#### Synopsis

```
gmx covar [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>
↪]]
          [-o [<.xvg>]] [-v [<.trr/.cpt/...>]]
          [-av [<.gro/.g96/...>]] [-l [<.log>]] [-ascii [<.dat>]]
          [-xpm [<.xpm>]] [-xpma [<.xpm>]] [-b <time>] [-e <time>]
          [-dt <time>] [-tu <enum>] [-xvg <enum>] [-[no]fit]
          [-[no]ref] [-[no]mwa] [-last <int>] [-[no]pbc]
```

#### Description

`gmx covar` calculates and diagonalizes the (mass-weighted) covariance matrix. All structures are fitted to the structure in the structure file. When this is not a run input file periodicity will not be taken into account. When the fit and analysis groups are identical and the analysis is non mass-weighted, the fit will also be non mass-weighted.

The eigenvectors are written to a trajectory file (`-v`). When the same atoms are used for the fit and the covariance analysis, the reference structure for the fit is written first with t=-1. The average (or reference when `-ref` is used) structure is written with t=0, the eigenvectors are written as frames with the eigenvector number and eigenvalue as step number and timestamp, respectively.

The eigenvectors can be analyzed with *gmx anaeig* (page 151).

Option `-ascii` writes the whole covariance matrix to an ASCII file. The order of the elements is: x1x1, x1y1, x1z1, x1x2, ...

Option `-xpm` writes the whole covariance matrix to an *.xpm* (page 587) file.

Option `-xpma` writes the atomic covariance matrix to an *.xpm* (page 587) file, i.e. for each atom pair the sum of the xx, yy and zz covariances is written.

Note that the diagonalization of a matrix requires memory and time that will increase at least as fast as than the square of the number of atoms involved. It is easy to run out of memory, in which case this tool

will probably exit with a 'Segmentation fault'. You should consider carefully whether a reduced set of atoms will meet your needs for lower costs.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-o` [<.xvg>] (eigenval.xvg)**
xvgr/xmgr file

**`-v` [<.trr/.cpt/...>] (eigenvec.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**`-av` [<.gro/.g96/...>] (average.pdb)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

**`-l` [<.log>] (covar.log)**
Log file

**`-ascii` [<.dat>] (covar.dat) (Optional)**
Generic data file

**`-xpm` [<.xpm>] (covar.xpm) (Optional)**
X PixMap compatible matrix file

**`-xpma` [<.xpm>] (covara.xpm) (Optional)**
X PixMap compatible matrix file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-tu` <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-[no]fit` (yes)**
Fit to a reference structure

**-[no]ref (no)**
Use the deviation from the conformation in the structure file instead of from the average

**-[no]mwa (no)**
Mass-weighted covariance analysis

**-last <int> (-1)**
Last eigenvector to write away (-1 is till the last)

**-[no]pbc (yes)**
Apply corrections for periodic boundary conditions

### 3.11.16 gmx current

#### Synopsis

```
gmx current [-s [<.tpr/.gro/...>]] [-n [<.ndx>]] [-f [<.xtc/.trr/...
↪>]]
            [-o [<.xvg>]] [-caf [<.xvg>]] [-dsp [<.xvg>]]
            [-md [<.xvg>]] [-mj [<.xvg>]] [-mc [<.xvg>]] [-b <time>]
            [-e <time>] [-dt <time>] [-[no]w] [-xvg <enum>]
            [-sh <int>] [-[no]nojump] [-eps <real>] [-bfit <real>]
            [-efit <real>] [-bvit <real>] [-evit <real>]
            [-temp <real>]
```

#### Description

`gmx current` is a tool for calculating the current autocorrelation function, the correlation of the rotational and translational dipole moment of the system, and the resulting static dielectric constant. To obtain a reasonable result, the index group has to be neutral. Furthermore, the routine is capable of extracting the static conductivity from the current autocorrelation function, if velocities are given. Additionally, an Einstein-Helfand fit can be used to obtain the static conductivity.

The flag `-caf` is for the output of the current autocorrelation function and `-mc` writes the correlation of the rotational and translational part of the dipole moment in the corresponding file. However, this option is only available for trajectories containing velocities. Options `-sh` and `-tr` are responsible for the averaging and integration of the autocorrelation functions. Since averaging proceeds by shifting the starting point through the trajectory, the shift can be modified with `-sh` to enable the choice of uncorrelated starting points. Towards the end, statistical inaccuracy grows and integrating the correlation function only yields reliable values until a certain point, depending on the number of frames. The option `-tr` controls the region of the integral taken into account for calculating the static dielectric constant.

Option `-temp` sets the temperature required for the computation of the static dielectric constant.

Option `-eps` controls the dielectric constant of the surrounding medium for simulations using a Reaction Field or dipole corrections of the Ewald summation (`-eps`=0 corresponds to tin-foil boundary conditions).

`-[no]nojump` unfolds the coordinates to allow free diffusion. This is required to get a continuous translational dipole moment, required for the Einstein-Helfand fit. The results from the fit allow the determination of the dielectric constant for system of charged molecules. However, it is also possible to extract the dielectric constant from the fluctuations of the total dipole moment in folded coordinates. But this option has to be used with care, since only very short time spans fulfill the approximation that

the density of the molecules is approximately constant and the averages are already converged. To be on the safe side, the dielectric constant should be calculated with the help of the Einstein-Helfand method for the translational part of the dielectric constant.

### Options

Options to specify input files:

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

Options to specify output files:

**`-o` [<.xvg>] (current.xvg)**
xvgr/xmgr file

**`-caf` [<.xvg>] (caf.xvg) (Optional)**
xvgr/xmgr file

**`-dsp` [<.xvg>] (dsp.xvg)**
xvgr/xmgr file

**`-md` [<.xvg>] (md.xvg)**
xvgr/xmgr file

**`-mj` [<.xvg>] (mj.xvg)**
xvgr/xmgr file

**`-mc` [<.xvg>] (mc.xvg) (Optional)**
xvgr/xmgr file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-sh` <int> (1000)**
Shift of the frames for averaging the correlation functions and the mean-square displacement.

**`-[no]nojump` (yes)**
Removes jumps of atoms across the box.

**-eps <real> (0)**
Dielectric constant of the surrounding medium. The value zero corresponds to infinity (tin-foil boundary conditions).

**-bfit <real> (100)**
Begin of the fit of the straight line to the MSD of the translational fraction of the dipole moment.

**-efit <real> (400)**
End of the fit of the straight line to the MSD of the translational fraction of the dipole moment.

**-bvit <real> (0.5)**
Begin of the fit of the current autocorrelation function to a*t^b.

**-evit <real> (5)**
End of the fit of the current autocorrelation function to a*t^b.

**-temp <real> (300)**
Temperature for calculating epsilon.

### 3.11.17 gmx density

#### Synopsis

```
gmx density [-f [<.xtc/.trr/...>]] [-n [<.ndx>]] [-s [<.tpr>]]
            [-ei [<.dat>]] [-o [<.xvg>]] [-b <time>] [-e <time>]
            [-dt <time>] [-[no]w] [-xvg <enum>] [-d <string>]
            [-sl <int>] [-dens <enum>] [-ng <int>] [-[no]center]
            [-[no]symm]
```

#### Description

`gmx density` computes partial densities across the box, using an index file.

For the total density of NPT simulations, use *gmx energy* (page 214) instead.

Option `-center` performs the histogram binning relative to the center of an arbitrary group, in absolute box coordinates. If you are calculating profiles along the Z axis box dimension bZ, output would be from -bZ/2 to bZ/2 if you center based on the entire system. Note that this behaviour has changed in GROMACS 5.0; earlier versions merely performed a static binning in (0,bZ) and shifted the output. Now we compute the center for each frame and bin in (-bZ/2,bZ/2).

Option `-symm` symmetrizes the output around the center. This will automatically turn on `-center` too. The binning is now always performed in relative coordinates to account for changing box dimensions with pressure coupling, with the output scaled to the average box dimension along the output axis.

Densities are in kg/m^3, and number densities or electron densities can also be calculated. For electron densities, a file describing the number of electrons for each type of atom should be provided using `-ei`. It should look like:

```
2
atomname = nrelectrons
atomname = nrelectrons
```

The first line contains the number of lines to read from the file. There should be one line for each unique atom name in your system. The number of electrons for each atom is modified by its atomic partial charge.

IMPORTANT CONSIDERATIONS FOR BILAYERS

One of the most common usage scenarios is to calculate the density of various groups across a lipid bilayer, typically with the z axis being the normal direction. For short simulations, small systems, and fixed box sizes this will work fine, but for the more general case lipid bilayers can be complicated. The first problem that while both proteins and lipids have low volume compressibility, lipids have quite high area compressiblity. This means the shape of the box (thickness and area/lipid) will fluctuate substantially even for a fully relaxed system. Since GROMACS places the box between the origin and positive coordinates, this in turn means that a bilayer centered in the box will move a bit up/down due to these fluctuations, and smear out your profile. The easiest way to fix this (if you want pressure coupling) is to use the `-center` option that calculates the density profile with respect to the center of the box. Note that you can still center on the bilayer part even if you have a complex non-symmetric system with a bilayer and, say, membrane proteins - then our output will simply have more values on one side of the (center) origin reference.

Finally, large bilayers that are not subject to a surface tension will exhibit undulatory fluctuations, where there are 'waves' forming in the system. This is a fundamental property of the biological system, and if you are comparing against experiments you likely want to include the undulation smearing effect.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

**`-ei` [<.dat>] (electrons.dat) (Optional)**
Generic data file

Options to specify output files:

**`-o` [<.xvg>] (density.xvg)**
xvgr/xmgr file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-d <string> (Z)**
Take the normal on the membrane in direction X, Y or Z.

**-sl <int> (50)**
Divide the box in this number of slices.

**-dens <enum> (mass)**
Density: mass, number, charge, electron

**-ng <int> (1)**
Number of groups of which to compute densities.

**-[no]center (no)**
Perform the binning relative to the center of the (changing) box. Useful for bilayers.

**-[no]symm (no)**
Symmetrize the density along the axis, with respect to the center. Useful for bilayers.

### Known Issues

- When calculating electron densities, atomnames are used instead of types. This is bad.

## 3.11.18 gmx densmap

### Synopsis

```
gmx densmap [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.
↪ndx>]]
            [-od [<.dat>]] [-o [<.xpm>]] [-b <time>] [-e <time>]
            [-dt <time>] [-[no]w] [-bin <real>] [-aver <enum>]
            [-xmin <real>] [-xmax <real>] [-n1 <int>] [-n2 <int>]
            [-amax <real>] [-rmax <real>] [-[no]mirror] [-[no]sums]
            [-unit <enum>] [-dmin <real>] [-dmax <real>]
```

### Description

`gmx densmap` computes 2D number-density maps. It can make planar and axial-radial density maps. The output *.xpm* (page 587) file can be visualized with for instance xv and can be converted to postscript with `xpm2ps`. Optionally, output can be in text form to a *.dat* (page 576) file with `-od`, instead of the usual *.xpm* (page 587) file with `-o`.

The default analysis is a 2-D number-density map for a selected group of atoms in the x-y plane. The averaging direction can be changed with the option `-aver`. When `-xmin` and/or `-xmax` are set only atoms that are within the limit(s) in the averaging direction are taken into account. The grid spacing is set with the option `-bin`. When `-n1` or `-n2` is non-zero, the grid size is set by this option. Box size fluctuations are properly taken into account.

When options `-amax` and `-rmax` are set, an axial-radial number-density map is made. Three groups should be supplied, the centers of mass of the first two groups define the axis, the third defines the analysis group. The axial direction goes from -amax to +amax, where the center is defined as the

midpoint between the centers of mass and the positive direction goes from the first to the second center of mass. The radial direction goes from 0 to rmax or from -rmax to +rmax when the `-mirror` option has been set.

The normalization of the output is set with the `-unit` option. The default produces a true number density. Unit `nm-2` leaves out the normalization for the averaging or the angular direction. Option `count` produces the count for each grid cell. When you do not want the scale in the output to go from zero to the maximum density, you can set the maximum with the option `-dmax`.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-od` [<.dat>] (densmap.dat) (Optional)**
Generic data file

**`-o` [<.xpm>] (densmap.xpm)**
X PixMap compatible matrix file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-bin` <real> (0.02)**
Grid size (nm)

**`-aver` <enum> (z)**
The direction to average over: z, y, x

**`-xmin` <real> (-1)**
Minimum coordinate for averaging

**`-xmax` <real> (-1)**
Maximum coordinate for averaging

**`-n1` <int> (0)**
Number of grid cells in the first direction

**-n2 <int> (0)**
Number of grid cells in the second direction

**-amax <real> (0)**
Maximum axial distance from the center

**-rmax <real> (0)**
Maximum radial distance

**-[no]mirror (no)**
Add the mirror image below the axial axis

**-[no]sums (no)**
Print density sums (1D map) to stdout

**-unit <enum> (nm-3)**
Unit for the output: nm-3, nm-2, count

**-dmin <real> (0)**
Minimum density in output

**-dmax <real> (0)**
Maximum density in output (0 means calculate it)

### 3.11.19 gmx densorder

#### Synopsis

```
gmx densorder [-s [<.tpr>]] [-f [<.xtc/.trr/...>]] [-n [<.ndx>]]
              [-o [<.dat>]] [-or [<.out> [...]]] [-og [<.xpm> [...]]]
              [-Spect [<.out> [...]]] [-b <time>] [-e <time>]
              [-dt <time>] [-[no]w] [-[no]1d] [-bw <real>]
              [-bwn <real>] [-order <int>] [-axis <string>]
              [-method <enum>] [-d1 <real>] [-d2 <real>]
              [-tblock <int>] [-nlevel <int>]
```

#### Description

`gmx densorder` reduces a two-phase density distribution along an axis, computed over a MD trajectory, to 2D surfaces fluctuating in time, by a fit to a functional profile for interfacial densities. A time-averaged spatial representation of the interfaces can be output with the option `-tavg`.

#### Options

Options to specify input files:

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-n [<.ndx>] (index.ndx)**
Index file

Options to specify output files:

**-o [<.dat>] (Density4D.dat) (Optional)**
Generic data file

**-or [<.out> [...]] (hello.out) (Optional)**
Generic output file

**-og [<.xpm> [...]] (interface.xpm) (Optional)**
X PixMap compatible matrix file

**-Spect [<.out> [...]] (intfspect.out) (Optional)**
Generic output file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-[no]1d (no)**
Pseudo-1d interface geometry

**-bw <real> (0.2)**
Binwidth of density distribution tangential to interface

**-bwn <real> (0.05)**
Binwidth of density distribution normal to interface

**-order <int> (0)**
Order of Gaussian filter, order 0 equates to NO filtering

**-axis <string> (Z)**
Axis Direction - X, Y or Z

**-method <enum> (bisect)**
Interface location method: bisect, functional

**-d1 <real> (0)**
Bulk density phase 1 (at small z)

**-d2 <real> (1000)**
Bulk density phase 2 (at large z)

**-tblock <int> (100)**
Number of frames in one time-block average

**-nlevel <int> (100)**
Number of Height levels in 2D - XPixMaps

### 3.11.20 gmx dielectric

#### Synopsis

```
gmx dielectric [-f [<.xvg>]] [-d [<.xvg>]] [-o [<.xvg>]] [-c [<.xvg>
↪]]
               [-b <time>] [-e <time>] [-dt <time>] [-[no]w]
               [-xvg <enum>] [-[no]x1] [-eint <real>] [-bfit <real>]
               [-efit <real>] [-tail <real>] [-A <real>] [-tau1 <real>
↪]
               [-tau2 <real>] [-eps0 <real>] [-epsRF <real>]
               [-fix <int>] [-ffn <enum>] [-nsmooth <int>]
```

#### Description

`gmx dielectric` calculates frequency dependent dielectric constants from the autocorrelation function of the total dipole moment in your simulation. This ACF can be generated by *gmx dipoles* (page 193). The functional forms of the available functions are:

- One parameter: y = exp(-a_1 x),
- Two parameters: y = a_2 exp(-a_1 x),
- Three parameters: y = a_2 exp(-a_1 x) + (1 - a_2) exp(-a_3 x).

Start values for the fit procedure can be given on the command line. It is also possible to fix parameters at their start value, use `-fix` with the number of the parameter you want to fix.

Three output files are generated, the first contains the ACF, an exponential fit to it with 1, 2 or 3 parameters, and the numerical derivative of the combination data/fit. The second file contains the real and imaginary parts of the frequency-dependent dielectric constant, the last gives a plot known as the Cole-Cole plot, in which the imaginary component is plotted as a function of the real component. For a pure exponential relaxation (Debye relaxation) the latter plot should be one half of a circle.

#### Options

Options to specify input files:

**-f [<.xvg>] (dipcorr.xvg)**
xvgr/xmgr file

Options to specify output files:

**-d [<.xvg>] (deriv.xvg)**
xvgr/xmgr file

**-o [<.xvg>] (epsw.xvg)**
xvgr/xmgr file

**-c [<.xvg>] (cole.xvg)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]x1 (yes)**
use first column as *x*-axis rather than first data set

**-eint <real> (5)**
Time to end the integration of the data and start to use the fit

**-bfit <real> (5)**
Begin time of fit

**-efit <real> (500)**
End time of fit

**-tail <real> (500)**
Length of function including data and tail from fit

**-A <real> (0.5)**
Start value for fit parameter A

**-tau1 <real> (10)**
Start value for fit parameter tau1

**-tau2 <real> (1)**
Start value for fit parameter tau2

**-eps0 <real> (80)**
epsilon0 of your liquid

**-epsRF <real> (78.5)**
epsilon of the reaction field used in your simulation. A value of 0 means infinity.

**-fix <int> (0)**
Fix parameters at their start values, A (2), tau1 (1), or tau2 (4)

**-ffn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-nsmooth <int> (3)**
Number of points for smoothing

### 3.11.21 gmx dipoles

#### Synopsis

```
gmx dipoles [-en [<.edr>]] [-f [<.xtc/.trr/...>]] [-s [<.tpr>]]
            [-n [<.ndx>]] [-o [<.xvg>]] [-eps [<.xvg>]] [-a [<.xvg>
↪]]
            [-d [<.xvg>]] [-c [<.xvg>]] [-g [<.xvg>]]
            [-adip [<.xvg>]] [-dip3d [<.xvg>]] [-cos [<.xvg>]]
            [-cmap [<.xpm>]] [-slab [<.xvg>]] [-b <time>] [-e <time>
↪]
            [-dt <time>] [-[no]w] [-xvg <enum>] [-mu <real>]
            [-mumax <real>] [-epsilonRF <real>] [-skip <int>]
            [-temp <real>] [-corr <enum>] [-[no]pairs] [-[no]quad]
            [-ncos <int>] [-axis <string>] [-sl <int>]
            [-gkratom <int>] [-gkratom2 <int>] [-rcmax <real>]
            [-[no]phi] [-nlevels <int>] [-ndegrees <int>]
            [-acflen <int>] [-[no]normalize] [-P <enum>]
            [-fitfn <enum>] [-beginfit <real>] [-endfit <real>]
```

#### Description

`gmx dipoles` computes the total dipole plus fluctuations of a simulation system. From this you can compute e.g. the dielectric constant for low-dielectric media. For molecules with a net charge, the net charge is subtracted at center of mass of the molecule.

The file `Mtot.xvg` contains the total dipole moment of a frame, the components as well as the norm of the vector. The file `aver.xvg` contains <|mu|^2> and |<mu>|^2 during the simulation. The file `dipdist.xvg` contains the distribution of dipole moments during the simulation The value of `-mumax` is used as the highest value in the distribution graph.

Furthermore, the dipole autocorrelation function will be computed when option `-corr` is used. The output file name is given with the `-c` option. The correlation functions can be averaged over all molecules (`mol`), plotted per molecule separately (`molsep`) or it can be computed over the total dipole moment of the simulation box (`total`).

Option `-g` produces a plot of the distance dependent Kirkwood G-factor, as well as the average cosine of the angle between the dipoles as a function of the distance. The plot also includes gOO and hOO according to Nymand & Linse, J. Chem. Phys. 112 (2000) pp 6386-6395. In the same plot, we also include the energy per scale computed by taking the inner product of the dipoles divided by the distance to the third power.

EXAMPLES

```
gmx dipoles -corr mol -P 1 -o dip_sqr -mu 2.273 -mumax 5.0
```

This will calculate the autocorrelation function of the molecular dipoles using a first order Legendre polynomial of the angle of the dipole vector and itself a time t later. For this calculation 1001 frames will be used. Further, the dielectric constant will be calculated using an `-epsilonRF` of infinity (default), temperature of 300 K (default) and an average dipole moment of the molecule of 2.273 (SPC). For the distribution function a maximum of 5.0 will be used.

### Options

Options to specify input files:

**`-en` [<.edr>] (ener.edr) (Optional)**
Energy file

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-o` [<.xvg>] (Mtot.xvg)**
xvgr/xmgr file

**`-eps` [<.xvg>] (epsilon.xvg)**
xvgr/xmgr file

**`-a` [<.xvg>] (aver.xvg)**
xvgr/xmgr file

**`-d` [<.xvg>] (dipdist.xvg)**
xvgr/xmgr file

**`-c` [<.xvg>] (dipcorr.xvg) (Optional)**
xvgr/xmgr file

**`-g` [<.xvg>] (gkr.xvg) (Optional)**
xvgr/xmgr file

**`-adip` [<.xvg>] (adip.xvg) (Optional)**
xvgr/xmgr file

**`-dip3d` [<.xvg>] (dip3d.xvg) (Optional)**
xvgr/xmgr file

**`-cos` [<.xvg>] (cosaver.xvg) (Optional)**
xvgr/xmgr file

**`-cmap` [<.xpm>] (cmap.xpm) (Optional)**
X PixMap compatible matrix file

**`-slab` [<.xvg>] (slab.xvg) (Optional)**
xvgr/xmgr file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-mu <real> (-1)**
dipole of a single molecule (in Debye)

**-mumax <real> (5)**
max dipole in Debye (for histogram)

**-epsilonRF <real> (0)**
epsilon of the reaction field used during the simulation, needed for dielectric constant calculation. WARNING: 0.0 means infinity (default)

**-skip <int> (0)**
Skip steps in the output (but not in the computations)

**-temp <real> (300)**
Average temperature of the simulation (needed for dielectric constant calculation)

**-corr <enum> (none)**
Correlation function to calculate: none, mol, molsep, total

**-[no]pairs (yes)**
Calculate |cos(theta)| between all pairs of molecules. May be slow

**-[no]quad (no)**
Take quadrupole into account

**-ncos <int> (1)**
Must be 1 or 2. Determines whether the <cos(theta)> is computed between all molecules in one group, or between molecules in two different groups. This turns on the `-g` flag.

**-axis <string> (Z)**
Take the normal on the computational box in direction X, Y or Z.

**-sl <int> (10)**
Divide the box into this number of slices.

**-gkratom <int> (0)**
Use the n-th atom of a molecule (starting from 1) to calculate the distance between molecules rather than the center of charge (when 0) in the calculation of distance dependent Kirkwood factors

**-gkratom2 <int> (0)**
Same as previous option in case ncos = 2, i.e. dipole interaction between two groups of molecules

**-rcmax <real> (0)**
Maximum distance to use in the dipole orientation distribution (with ncos == 2). If zero, a criterion based on the box length will be used.

**-[no]phi (no)**
Plot the 'torsion angle' defined as the rotation of the two dipole vectors around the distance vector between the two molecules in the *.xpm* (page 587) file from the `-cmap` option. By default the cosine of the angle between the dipoles is plotted.

**-nlevels <int> (20)**
Number of colors in the cmap output

**-ndegrees <int> (90)**
Number of divisions on the *y*-axis in the cmap output (for 180 degrees)

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### 3.11.22 gmx disre

#### Synopsis

```
gmx disre [-s [<.tpr>]] [-f [<.xtc/.trr/...>]] [-n [<.ndx>]]
          [-c [<.ndx>]] [-ds [<.xvg>]] [-da [<.xvg>]] [-dn [<.xvg>]]
          [-dm [<.xvg>]] [-dr [<.xvg>]] [-l [<.log>]] [-q [<.pdb>]]
          [-x [<.xpm>]] [-b <time>] [-e <time>] [-dt <time>]␣
↪[-[no]w]
          [-xvg <enum>] [-ntop <int>] [-maxdr <real>]
          [-nlevels <int>] [-[no]third]
```

#### Description

`gmx disre` computes violations of distance restraints. The program always computes the instantaneous violations rather than time-averaged, because this analysis is done from a trajectory file afterwards it does not make sense to use time averaging. However, the time averaged values per restraint are given in the log file.

An index file may be used to select specific restraints by index group label for printing.

When the optional `-q` flag is given a *.pdb* (page 581) file coloured by the amount of average violations.

When the `-c` option is given, an index file will be read containing the frames in your trajectory corresponding to the clusters (defined in another manner) that you want to analyze. For these clusters the program will compute average violations using the third power averaging algorithm and print them in the log file.

### Options

Options to specify input files:

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-n` [<.ndx>] (viol.ndx) (Optional)**
Index file

**`-c` [<.ndx>] (clust.ndx) (Optional)**
Index file

Options to specify output files:

**`-ds` [<.xvg>] (drsum.xvg)**
xvgr/xmgr file

**`-da` [<.xvg>] (draver.xvg)**
xvgr/xmgr file

**`-dn` [<.xvg>] (drnum.xvg)**
xvgr/xmgr file

**`-dm` [<.xvg>] (drmax.xvg)**
xvgr/xmgr file

**`-dr` [<.xvg>] (restr.xvg)**
xvgr/xmgr file

**`-l` [<.log>] (disres.log)**
Log file

**`-q` [<.pdb>] (viol.pdb) (Optional)**
Protein data bank file

**`-x` [<.xpm>] (matrix.xpm) (Optional)**
X PixMap compatible matrix file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-ntop <int> (0)**
Number of large violations that are stored in the log file every step

**-maxdr <real> (0)**
Maximum distance violation in matrix output. If less than or equal to 0 the maximum will be determined by the data.

**-nlevels <int> (20)**
Number of levels in the matrix output

**-[no]third (yes)**
Use inverse third power averaging or linear for matrix output

### 3.11.23 gmx distance

#### Synopsis

```
gmx distance [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.
↪ndx>]]
             [-oav [<.xvg>]] [-oall [<.xvg>]] [-oxyz [<.xvg>]]
             [-oh [<.xvg>]] [-oallstat [<.xvg>]] [-b <time>]
             [-e <time>] [-dt <time>] [-tu <enum>]
             [-fgroup <selection>] [-xvg <enum>] [-[no]rmpbc]
             [-[no]pbc] [-sf <file>] [-selrpos <enum>]
             [-seltype <enum>] [-select <selection>] [-len <real>]
             [-tol <real>] [-binw <real>]
```

#### Description

`gmx distance` calculates distances between pairs of positions as a function of time. Each selection specifies an independent set of distances to calculate. Each selection should consist of pairs of positions, and the distances are computed between positions 1-2, 3-4, etc.

`-oav` writes the average distance as a function of time for each selection. `-oall` writes all the individual distances. `-oxyz` does the same, but the x, y, and z components of the distance are written instead of the norm. `-oh` writes a histogram of the distances for each selection. The location of the histogram is set with `-len` and `-tol`. Bin width is set with `-binw`. `-oallstat` writes out the average and standard deviation for each individual distance, calculated over the frames.

Note that `gmx distance` calculates distances between fixed pairs (1-2, 3-4, etc.) within a single selection. To calculate distances between two selections, including minimum, maximum, and pairwise distances, use *gmx pairdist* (page 279).

### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**-oav [<.xvg>] (distave.xvg) (Optional)**
Average distances as function of time

**-oall [<.xvg>] (dist.xvg) (Optional)**
All distances as function of time

**-oxyz [<.xvg>] (distxyz.xvg) (Optional)**
Distance components as function of time

**-oh [<.xvg>] (disthist.xvg) (Optional)**
Histogram of the distances

**-oallstat [<.xvg>] (diststat.xvg) (Optional)**
Statistics for individual distances

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-seltype <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-select <selection>**
Position pairs to calculate distances for

**-len <real> (0.1)**
Mean distance for histogramming

**-tol <real> (1)**
Width of full distribution as fraction of `-len`

**-binw <real> (0.001)**
Bin width for histogramming

### 3.11.24 gmx dos

#### Synopsis

```
gmx dos [-f [<.trr/.cpt/...>]] [-s [<.tpr>]] [-n [<.ndx>]]
        [-vacf [<.xvg>]] [-mvacf [<.xvg>]] [-dos [<.xvg>]]
        [-g [<.log>]] [-b <time>] [-e <time>] [-dt <time>] [-[no]w]
        [-xvg <enum>] [-[no]v] [-[no]recip] [-[no]abs]␣
↪[-[no]normdos]
        [-T <real>] [-acflen <int>] [-[no]normalize] [-P <enum>]
        [-fitfn <enum>] [-beginfit <real>] [-endfit <real>]
```

#### Description

`gmx dos` computes the Density of States from a simulations. In order for this to be meaningful the velocities must be saved in the trajecotry with sufficiently high frequency such as to cover all vibrations. For flexible systems that would be around a few fs between saving. Properties based on the DoS are printed on the standard output. Note that the density of states is calculated from the mass-weighted autocorrelation, and by default only from the square of the real component rather than absolute value. This means the shape can differ substantially from the plain vibrational power spectrum you can calculate with gmx velacc.

### Options

Options to specify input files:

**-f [<.trr/.cpt/...>] (traj.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**-vacf [<.xvg>] (vacf.xvg)**
xvgr/xmgr file

**-mvacf [<.xvg>] (mvacf.xvg)**
xvgr/xmgr file

**-dos [<.xvg>] (dos.xvg)**
xvgr/xmgr file

**-g [<.log>] (dos.log)**
Log file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]v (yes)**
Be loud and noisy.

**-[no]recip (no)**
Use cm^-1 on X-axis instead of 1/ps for DoS plots.

**-[no]abs (no)**
Use the absolute value of the Fourier transform of the VACF as the Density of States. Default is to use the real component only

**-[no]normdos (no)**
Normalize the DoS such that it adds up to 3N. This should usually not be necessary.

**-T <real> (298.15)**
Temperature in the simulation

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### Known Issues

- This program needs a lot of memory: total usage equals the number of atoms times 3 times number of frames times 4 (or 8 when run in double precision).

## 3.11.25 gmx dssp

### Synopsis

```
gmx dssp [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>]]
         [-o [<.dat>]] [-num [<.xvg>]] [-b <time>] [-e <time>]
         [-dt <time>] [-tu <enum>] [-fgroup <selection>]
         [-xvg <enum>] [-[no]rmpbc] [-[no]pbc] [-sf <file>]
         [-selrpos <enum>] [-seltype <enum>] [-sel <selection>]
         [-hmode <enum>] [-hbond <enum>] [-[no]nb] [-cutoff <real>]
         [-[no]clear] [-[no]pihelix] [-ppstretch <enum>]
         [-[no]polypro]
```

### Description

`gmx dssp` allows using the DSSP algorithm (namely, by detecting specific patterns of hydrogen bonds between amino acid residues) to determine the secondary structure of a protein.

One-symbol secondary structure designations that are used in the output file:

`H` — alpha-helix;

`B` — residue in isolated beta-bridge;

`E` — extended strand that participates in beta-ladder;

`G` — 3_10-helix;

`I` — pi-helix;

`P` — kappa-helix (poly-proline II helix);

`S` — bend;

`T` — hydrogen-bonded turn;

= — break;

~ — loop (no special secondary structure designation).

`-num` allows you to get a plot of the number of secondary structures of each type as a function of time at the output.

`-hmode` selects between using hydrogen atoms directly from the structure ("gromacs" option) and using hydrogen pseudo-atoms based on C and O atom coordinates of previous residue ("dssp" option). You should always use the "dssp" option for structures with absent hydrogen atoms!

`-hbond` selects between different definitions of hydrogen bond. "energy" means the calculation of a hydrogen bond using the electrostatic interaction energy and "geometry" means the calculation of the hydrogen bond using geometric criterion for the existence of a hydrogen bond.

`-nb` allows using GROMACS neighbor-search method to find residue pairs that may have a hydrogen bond instead of simply iterating over the residues among themselves.

`-cutoff` is a real value that defines maximum distance from residue to its neighbor residue used in `-nb`. Minimum (and also recommended) value is 0.9.

`-clear` allows you to ignore the analysis of the secondary structure residues that are missing one or more critical atoms (CA, C, N, O or H). Always use this option together with `-hmode dssp` for structures that lack hydrogen atoms!

`-pihelix` changes pattern-search algorithm towards preference of pi-helices.

`-ppstretch` defines stretch value of polyproline-helices. "shortened" means stretch with size 2 and "default" means stretch with size 3.

`-polypro` enables the search for polyproline helices (default behavior, equivalent to DSSP v4). Disabling this option will result in disabling the search for polyproline helices, reproducing the behavior of DSSP v2.

Note that `gmx dssp` currently is not capable of reproducing the secondary structure of proteins whose structure is determined by methods other than X-ray crystallography (structures in .pdb format with incorrect values in the CRYST1 line) due to the incorrect cell size in such structures.

Please note that the computation is always done in single precision, regardless of the precision for which GROMACS was configured.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**-o [<.dat>] (dssp.dat)**
Filename for DSSP output

**-num [<.xvg>] (num.xvg) (Optional)**
Output file name for secondary structures statistics for the trajectory

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-seltype <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-sel <selection>**
Group for DSSP

**-hmode <enum> (gromacs)**
Hydrogens pseudoatoms creating mode: gromacs, dssp

**-hbond <enum> (energy)**
Selects between different definitions of hydrogen bond: energy, geometry

**-[no]nb (yes)**
Use GROMACS neighbor-search method

**-cutoff <real> (0.9)**
Distance from residue to its neighbor residue in neighbor search. Must be >= 0.9

**-[no]clear (no)**
Clear defective residues from the structure

**-[no]pihelix (no)**
Prefer Pi Helices

**-ppstretch <enum> (default)**
Stretch value for PP-helices: shortened, default

**-[no]polypro (yes)**
Perform a search for polyproline helices

### 3.11.26 gmx dump

#### Synopsis

```
gmx dump [-s <.tpr>] [-f <.xtc/.trr/...>] [-e <.edr>] [-cp <.cpt>]
         [-p <.top>] [-mtx <.mtx>] [-om <.mdp>] [-[no]nr]
         [-[no]param] [-[no]sys] [-[no]orgir]
```

#### Description

`gmx dump` reads a run input file (*.tpr* (page 586)), a trajectory (*.trr* (page 586)/*.xtc* (page 588)/`tng`), an energy file (*.edr* (page 576)), a checkpoint file (*.cpt* (page 575)) or topology file (*.top* (page 584)) and prints that to standard output in a readable format. This program is essential for checking your run input file in case of problems.

#### Options

Options to specify input files:

**-s <.tpr> (Optional)**
Run input file to dump

**-f <.xtc/.trr/...> (Optional)**
Trajectory file to dump: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-e <.edr> (Optional)**
Energy file to dump

**-cp <.cpt> (Optional)**
Checkpoint file to dump

**-p <.top> (Optional)**
Topology file to dump

**-mtx <.mtx> (Optional)**
Hessian matrix to dump

Options to specify output files:

**-om <.mdp> (Optional)**
grompp input file from run input file

Other options:

**-[no]nr (yes)**
Show index numbers in output (leaving them out makes comparison easier, but creates a useless topology)

**-[no]param (no)**
Show parameters for each bonded interaction (for comparing dumps, it is useful to combine this with -nonr)

**-[no]sys (no)**
List the atoms and bonded interactions for the whole system instead of for each molecule type

**-[no]orgir (no)**
Show input parameters from tpr as they were written by the version that produced the file, instead of how the current version reads them

### Known Issues

- The *.mdp* (page 579) file produced by `-om` can not be read by grompp.

## 3.11.27 gmx dyecoupl

### Synopsis

```
gmx dyecoupl [-f [<.xtc/.trr/...>]] [-n [<.ndx>]] [-ot [<.xvg>]]
             [-oe [<.xvg>]] [-o [<.dat>]] [-rhist [<.xvg>]]
             [-khist [<.xvg>]] [-b <time>] [-e <time>] [-tu <enum>]
             [-[no]w] [-xvg <enum>] [-[no]pbcdist] [-[no]norm]
             [-bins <int>] [-R0 <real>]
```

### Description

`gmx dyecoupl` extracts dye dynamics from trajectory files. Currently, R and kappa^2 between dyes is extracted for (F)RET simulations with assumed dipolar coupling as in the Foerster equation. It further allows the calculation of R(t) and kappa^2(t), R and kappa^2 histograms and averages, as well as the instantaneous FRET efficiency E(t) for a specified Foerster radius R_0 (switch `-R0`). The input dyes have to be whole (see res and mol pbc options in `trjconv`). The dye transition dipole moment has to be defined by at least a single atom pair, however multiple atom pairs can be provided in the index file. The distance R is calculated on the basis of the COMs of the given atom pairs. The `-pbcdist` option calculates distances to the nearest periodic image instead to the distance in the box. This works however only, for periodic boundaries in all 3 dimensions. The `-norm` option (area-) normalizes the histograms.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-n` [<.ndx>] (index.ndx)**
Index file

Options to specify output files:

**`-ot` [<.xvg>] (rkappa.xvg) (Optional)**
xvgr/xmgr file

**`-oe` [<.xvg>] (insteff.xvg) (Optional)**
xvgr/xmgr file

**`-o` [<.dat>] (rkappa.dat) (Optional)**
Generic data file

**`-rhist` [<.xvg>] (rhist.xvg) (Optional)**
xvgr/xmgr file

**`-khist` [<.xvg>] (khist.xvg) (Optional)**
xvgr/xmgr file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-tu` <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-[no]pbcdist` (no)**
Distance R based on PBC

**`-[no]norm` (no)**
Normalize histograms

**`-bins` <int> (50)**
# of histogram bins

**`-R0` <real> (-1)**
Foerster radius including kappa^2=2/3 in nm

### 3.11.28 gmx editconf

#### Synopsis

```
gmx editconf [-f [<.gro/.g96/...>]] [-n [<.ndx>]] [-bf [<.dat>]]
             [-o [<.gro/.g96/...>]] [-mead [<.pqr>]] [-[no]w]
             [-[no]ndef] [-bt <enum>] [-box <vector>]
             [-angles <vector>] [-d <real>] [-[no]c]
             [-center <vector>] [-aligncenter <vector>]
             [-align <vector>] [-translate <vector>]
             [-rotate <vector>] [-[no]princ] [-scale <vector>]
             [-density <real>] [-[no]pbc] [-resnr <int>]␣
↪[-[no]grasp]
             [-rvdw <real>] [-[no]sig56] [-[no]vdwread] [-[no]atom]
             [-[no]legend] [-label <string>] [-[no]conect]
```

#### Description

`gmx editconf` converts generic structure format to *.gro* (page 577), `.g96` or *.pdb* (page 581).

The box can be modified with options `-box`, `-d` and `-angles`. Both `-box` and `-d` will center the system in the box, unless `-noc` is used. The `-center` option can be used to shift the geometric center of the system from the default of (x/2, y/2, z/2) implied by `-c` to some other value.

Option `-bt` determines the box type: `triclinic` is a triclinic box, `cubic` is a rectangular box with all sides equal `dodecahedron` represents a rhombic dodecahedron and `octahedron` is a truncated octahedron. The last two are special cases of a triclinic box. The length of the three box vectors of the truncated octahedron is the shortest distance between two opposite hexagons. Relative to a cubic box with some periodic image distance, the volume of a dodecahedron with this same periodic distance is 0.71 times that of the cube, and that of a truncated octahedron is 0.77 times.

Option `-box` requires only one value for a cubic, rhombic dodecahedral, or truncated octahedral box.

With `-d` and a `triclinic` box the size of the system in the *x*-, *y*-, and *z*-directions is used. With `-d` and `cubic`, `dodecahedron` or `octahedron` boxes, the dimensions are set to the diameter of the system (largest distance between atoms) plus twice the specified distance.

Option `-angles` is only meaningful with option `-box` and a triclinic box and cannot be used with option `-d`.

When `-n` or `-ndef` is set, a group can be selected for calculating the size and the geometric center, otherwise the whole system is used.

`-rotate` rotates the coordinates and velocities.

`-princ` aligns the principal axes of the system along the coordinate axes, with the longest axis aligned with the *x*-axis. This may allow you to decrease the box volume, but beware that molecules can rotate significantly in a nanosecond.

Scaling is applied before any of the other operations are performed. Boxes and coordinates can be scaled to give a certain density (option `-density`). Note that this may be inaccurate in case a *.gro* (page 577) file is given as input. A special feature of the scaling option is that when the factor -1 is given in one dimension, one obtains a mirror image, mirrored in one of the planes. When one uses -1 in three dimensions, a point-mirror image is obtained.

Groups are selected after all operations have been applied.

Periodicity can be removed in a crude manner. It is important that the box vectors at the bottom of your input file are correct when the periodicity is to be removed.

When writing *.pdb* (page 581) files, B-factors can be added with the `-bf` option. B-factors are read from a file with with following format: first line states number of entries in the file, next lines state an index followed by a B-factor. The B-factors will be attached per residue unless the number of B-factors is larger than the number of the residues or unless the `-atom` option is set. Obviously, any type of numeric data can be added instead of B-factors. `-legend` will produce a row of CA atoms with B-factors ranging from the minimum to the maximum value found, effectively making a legend for viewing.

With the option `-mead` a special *.pdb* (page 581) (.pqr) file for the MEAD electrostatics program (Poisson-Boltzmann solver) can be made. A further prerequisite is that the input file is a run input file. The B-factor field is then filled with the Van der Waals radius of the atoms while the occupancy field will hold the charge.

The option `-grasp` is similar, but it puts the charges in the B-factor and the radius in the occupancy.

Option `-align` allows alignment of the principal axis of a specified group against the given vector, with an optional center of rotation specified by `-aligncenter`.

Finally, with option `-label`, `editconf` can add a chain identifier to a *.pdb* (page 581) file, which can be useful for analysis with e.g. Rasmol.

To convert a truncated octrahedron file produced by a package which uses a cubic box with the corners cut off (such as GROMOS), use:

```
gmx editconf -f in -rotate 0 45 35.264 -bt o -box veclen -o out
```

where `veclen` is the size of the cubic box times sqrt(3)/2.

## Options

Options to specify input files:

**`-f` [<.gro/.g96/...>] (conf.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

**`-bf` [<.dat>] (bfact.dat) (Optional)**
Generic data file

Options to specify output files:

**`-o` [<.gro/.g96/...>] (out.gro) (Optional)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

**`-mead` [<.pqr>] (mead.pqr) (Optional)**
Coordinate file for MEAD

Other options:

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-[no]ndef (no)**
Choose output from default index groups

**-bt <enum> (triclinic)**
Box type for -box and -d: triclinic, cubic, dodecahedron, octahedron

**-box <vector> (0 0 0)**
Box vector lengths (a,b,c)

**-angles <vector> (90 90 90)**
Angles between the box vectors (bc,ac,ab)

**-d <real> (0)**
Distance between the solute and the box

**-[no]c (no)**
Center molecule in box (implied by -box and -d)

**-center <vector> (0 0 0)**
Shift the geometrical center to (x,y,z)

**-aligncenter <vector> (0 0 0)**
Center of rotation for alignment

**-align <vector> (0 0 0)**
Align to target vector

**-translate <vector> (0 0 0)**
Translation

**-rotate <vector> (0 0 0)**
Rotation around the X, Y and Z axes in degrees

**-[no]princ (no)**
Orient molecule(s) along their principal axes

**-scale <vector> (1 1 1)**
Scaling factor

**-density <real> (1000)**
Density (g/L) of the output box achieved by scaling

**-[no]pbc (no)**
Remove the periodicity (make molecule whole again)

**-resnr <int> (-1)**
Renumber residues starting from resnr

**-[no]grasp (no)**
Store the charge of the atom in the B-factor field and the radius of the atom in the occupancy field

**-rvdw <real> (0.12)**
Default Van der Waals radius (in nm) if one can not be found in the database or if no parameters are present in the topology file

**-[no]sig56 (no)**
Use rmin/2 (minimum in the Van der Waals potential) rather than sigma/2

**-[no]vdwread (no)**
Read the Van der Waals radii from the file vdwradii.dat rather than computing the radii based on the force field

**-[no]atom (no)**
Force B-factor attachment per atom

**-[no]legend (no)**
Make B-factor legend

**-label <string> (A)**
Add chain label for all residues

**-[no]conect (no)**
Add CONECT records to a *.pdb* (page 581) file when written. Can only be done when a topology (tpr file) is present

### Known Issues

- For complex molecules, the periodicity removal routine may break down,
- in that case you can use *gmx trjconv* (page 334).

## 3.11.29 gmx eneconv

### Synopsis

```
gmx eneconv [-f [<.edr> [...]]] [-o [<.edr>]] [-b <time>] [-e <time>
↪]
            [-dt <time>] [-offset <time>] [-[no]settime] [-[no]sort]
            [-[no]rmdh] [-scalefac <real>] [-[no]error]
```

### Description

With *multiple files* specified for the `-f` option:

Concatenates several energy files in sorted order. In the case of double time frames, the one in the later file is used. By specifying `-settime` you will be asked for the start time of each file. The input files are taken from the command line, such that the command `gmx eneconv -f *.edr -o fixed.edr` should do the trick.

With *one file* specified for `-f`:

Reads one energy file and writes another, applying the `-dt`, `-offset`, `-t0` and `-settime` options and converting to a different format if necessary (indicated by file extensions).

`-settime` is applied first, then `-dt`/`-offset` followed by `-b` and `-e` to select which frames to write.

### Options

Options to specify input files:

**-f [<.edr> [...]] (ener.edr)**
Energy file

Options to specify output files:

**-o [<.edr>] (fixed.edr)**
Energy file

Other options:

**-b <time> (-1)**
First time to use

**-e <time> (-1)**
Last time to use

**-dt <time> (0)**
Only write out frame when t MOD dt = offset

**-offset <time> (0)**
Time offset for `-dt` option

**-[no]settime (no)**
Change starting time interactively

**-[no]sort (yes)**
Sort energy files (not frames)

**-[no]rmdh (no)**
Remove free energy block data

**-scalefac <real> (1)**
Multiply energy component by this factor

**-[no]error (yes)**
Stop on errors in the file

### Known Issues

- When combining trajectories the sigma and E^2 (necessary for statistics) are not updated correctly. Only the actual energy is correct. One thus has to compute statistics in another way.

## 3.11.30 gmx enemat

### Synopsis

```
gmx enemat [-f [<.edr>]] [-groups [<.dat>]] [-eref [<.dat>]]
           [-emat [<.xpm>]] [-etot [<.xvg>]] [-b <time>] [-e <time>]
           [-dt <time>] [-[no]w] [-xvg <enum>] [-[no]sum]
           [-skip <int>] [-[no]mean] [-nlevels <int>] [-max <real>]
           [-min <real>] [-[no]coulsr] [-[no]coul14] [-[no]ljsr]
           [-[no]lj14] [-[no]bhamsr] [-[no]free] [-temp <real>]
```

## Description

`gmx enemat` extracts an energy matrix from the energy file (`-f`). With `-groups` a file must be supplied with on each line a group of atoms to be used. For these groups matrix of interaction energies will be extracted from the energy file by looking for energy groups with names corresponding to pairs of groups of atoms, e.g. if your `-groups` file contains:

```
2
Protein
SOL
```

then energy groups with names like 'Coul-SR:Protein-SOL' and 'LJ:Protein-SOL' are expected in the energy file (although `gmx enemat` is most useful if many groups are analyzed simultaneously). Matrices for different energy types are written out separately, as controlled by the `-[no]coul`, `-[no]coulr`, `-[no]coul14`, `-[no]lj`, `-[no]lj14`, `-[no]bham` and `-[no]free` options. Finally, the total interaction energy energy per group can be calculated (`-etot`).

An approximation of the free energy can be calculated using: E_free = E_0 + kT log(<exp((E-E_0)/kT)>), where '<>' stands for time-average. A file with reference free energies can be supplied to calculate the free energy difference with some reference state. Group names (e.g. residue names) in the reference file should correspond to the group names as used in the `-groups` file, but a appended number (e.g. residue number) in the `-groups` will be ignored in the comparison.

## Options

Options to specify input files:

**`-f` [<.edr>] (ener.edr) (Optional)**
Energy file

**`-groups` [<.dat>] (groups.dat)**
Generic data file

**`-eref` [<.dat>] (eref.dat) (Optional)**
Generic data file

Options to specify output files:

**`-emat` [<.xpm>] (emat.xpm)**
X PixMap compatible matrix file

**`-etot` [<.xvg>] (energy.xvg)**
xvgr/xmgr file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]sum (no)**
Sum the energy terms selected rather than display them all

**-skip <int> (0)**
Skip number of frames between data points

**-[no]mean (yes)**
with -groups extracts matrix of mean energies instead of matrix for each timestep

**-nlevels <int> (20)**
number of levels for matrix colors

**-max <real> (1e+20)**
max value for energies

**-min <real> (-1e+20)**
min value for energies

**-[no]coulsr (yes)**
extract Coulomb SR energies

**-[no]coul14 (no)**
extract Coulomb 1-4 energies

**-[no]ljsr (yes)**
extract Lennard-Jones SR energies

**-[no]lj14 (no)**
extract Lennard-Jones 1-4 energies

**-[no]bhamsr (no)**
extract Buckingham SR energies

**-[no]free (yes)**
calculate free energy

**-temp <real> (300)**
reference temperature for free energy calculation

### 3.11.31 gmx energy

#### Synopsis

```
gmx energy [-f [<.edr>]] [-f2 [<.edr>]] [-s [<.tpr>]] [-o [<.xvg>]]
           [-viol [<.xvg>]] [-pairs [<.xvg>]] [-corr [<.xvg>]]
           [-vis [<.xvg>]] [-evisco [<.xvg>]] [-eviscoi [<.xvg>]]
           [-ravg [<.xvg>]] [-odh [<.xvg>]] [-b <time>] [-e <time>]
           [-[no]w] [-xvg <enum>] [-[no]fee] [-fetemp <real>]
           [-zero <real>] [-[no]sum] [-[no]dp] [-nbmin <int>]
           [-nbmax <int>] [-[no]mutot] [-[no]aver] [-nmol <int>]
           [-[no]fluct_props] [-[no]driftcorr] [-[no]fluc]
           [-[no]orinst] [-[no]ovec] [-einstein_restarts <int>]
           [-einstein_blocks <int>] [-acflen <int>] [-[no]normalize]
           [-P <enum>] [-fitfn <enum>] [-beginfit <real>]
```

```
[-endfit <real>]
```

## Description

`gmx energy` extracts energy components from an energy file. The user is prompted to interactively select the desired energy terms.

Average, RMSD, and drift are calculated with full precision from the simulation (see printed manual). Drift is calculated by performing a least-squares fit of the data to a straight line. The reported total drift is the difference of the fit at the first and last point. An error estimate of the average is given based on a block averages over 5 blocks using the full-precision averages. The error estimate can be performed over multiple block lengths with the options `-nbmin` and `-nbmax`. **Note** that in most cases the energy files contains averages over all MD steps, or over many more points than the number of frames in energy file. This makes the `gmx energy` statistics output more accurate than the *.xvg* (page 589) output. When exact averages are not present in the energy file, the statistics mentioned above are simply over the single, per-frame energy values.

The term fluctuation gives the RMSD around the least-squares fit.

Some fluctuation-dependent properties can be calculated provided the correct energy terms are selected, and that the command line option `-fluct_props` is given. The following properties will be computed:

| Property | Energy terms needed |
|---|---|
| Heat capacity C_p (NPT sims): | Enthalpy, Temp |
| Heat capacity C_v (NVT sims): | Etot, Temp |
| Thermal expansion coeff. (NPT): | Enthalpy, Vol, Temp |
| Isothermal compressibility: | Vol, Temp |
| Adiabatic bulk modulus: | Vol, Temp |

You always need to set the number of molecules `-nmol`. The C_p/C_v computations do **not** include any corrections for quantum effects. Use the *gmx dos* (page 200) program if you need that (and you do).

Option `-odh` extracts and plots the free energy data (Hamiltoian differences and/or the Hamiltonian derivative dhdl) from the `ener.edr` file.

With `-fee` an estimate is calculated for the free-energy difference with an ideal gas state:

```
Delta A = A(N,V,T) - A_idealgas(N,V,T) = kT
ln(<exp(U_pot/kT)>)
Delta G = G(N,p,T) - G_idealgas(N,p,T) = kT
ln(<exp(U_pot/kT)>)
```

where k is Boltzmann's constant, T is set by `-fetemp` and the average is over the ensemble (or time in a trajectory). Note that this is in principle only correct when averaging over the whole (Boltzmann) ensemble and using the potential energy. This also allows for an entropy estimate using:

```
Delta S(N,V,T) = S(N,V,T) - S_idealgas(N,V,T) =
(<U_pot> - Delta A)/T
Delta S(N,p,T) = S(N,p,T) - S_idealgas(N,p,T) =
(<U_pot> + pV - Delta G)/T
```

When a second energy file is specified (`-f2`), a free energy difference is calculated:

```
dF = -kT
ln(<exp(-(E_B-E_A) /
kT)>_A),
```

where E_A and E_B are the energies from the first and second energy files, and the average is over the ensemble A. The running average of the free energy difference is printed to a file specified by `-ravg`. **Note** that the energies must both be calculated from the same trajectory.

For liquids, viscosities can be calculated by integrating the auto-correlation function of, or by using the Einstein formula for, the off-diagonal pressure elements. The option `-vis` turns calculation of the shear and bulk viscosity through integration of the auto-correlation function. For accurate results, this requires extremely frequent computation and output of the pressure tensor. The Einstein formula does not require frequent output and is therefore more convenient. Note that frequent pressure calculation (nstcalcenergy mdp parameter) is still needed. Option `-evicso` gives this shear viscosity estimate and option `-eviscoi` the integral. Using one of these two options also triggers the other. The viscosity is computed from integrals averaged over uniformly distributed `-einstein_restarts` starting points, which are sampled over one block out of `-einstein_blocks` of the trajectory.

### Options

Options to specify input files:

**`-f` [<.edr>] (ener.edr)**
Energy file

**`-f2` [<.edr>] (ener.edr) (Optional)**
Energy file

**`-s` [<.tpr>] (topol.tpr) (Optional)**
Portable xdr run input file

Options to specify output files:

**`-o` [<.xvg>] (energy.xvg)**
xvgr/xmgr file

**`-viol` [<.xvg>] (violaver.xvg) (Optional)**
xvgr/xmgr file

**`-pairs` [<.xvg>] (pairs.xvg) (Optional)**
xvgr/xmgr file

**`-corr` [<.xvg>] (enecorr.xvg) (Optional)**
xvgr/xmgr file

**`-vis` [<.xvg>] (visco.xvg) (Optional)**
xvgr/xmgr file

**`-evisco` [<.xvg>] (evisco.xvg) (Optional)**
xvgr/xmgr file

**`-eviscoi` [<.xvg>] (eviscoi.xvg) (Optional)**
xvgr/xmgr file

**`-ravg` [<.xvg>] (runavgdf.xvg) (Optional)**
xvgr/xmgr file

**-odh [<.xvg>] (dhdl.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]fee (no)**
Do a free energy estimate

**-fetemp <real> (300)**
Reference temperature for free energy calculation

**-zero <real> (0)**
Subtract a zero-point energy

**-[no]sum (no)**
Sum the energy terms selected rather than display them all

**-[no]dp (no)**
Print energies in high precision

**-nbmin <int> (5)**
Minimum number of blocks for error estimate

**-nbmax <int> (5)**
Maximum number of blocks for error estimate

**-[no]mutot (no)**
Compute the total dipole moment from the components

**-[no]aver (no)**
Also print the exact average and rmsd stored in the energy frames (only when 1 term is requested)

**-nmol <int> (1)**
Number of molecules in your sample: the energies are divided by this number

**-[no]fluct_props (no)**
Compute properties based on energy fluctuations, like heat capacity

**-[no]driftcorr (no)**
Useful only for calculations of fluctuation properties. The drift in the observables will be subtracted before computing the fluctuation properties.

**-[no]fluc (no)**
Calculate autocorrelation of energy fluctuations rather than energy itself

**-[no]orinst (no)**
Analyse instantaneous orientation data

**-[no]ovec (no)**
Also plot the eigenvectors with `-oten`

**-einstein_restarts <int> (100)**
Number of restarts for computing the viscosity using the Einstein relation

**-einstein_blocks <int> (4)**
Number of averaging windows for computing the viscosity using the Einstein relation

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### 3.11.32 gmx extract-cluster

#### Synopsis

```
gmx extract-cluster [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]]
              [-n [<.ndx>]] [-clusters [<.ndx>]]
              [-o [<.xtc/.trr/...>]] [-b <time>] [-e <time>]
              [-dt <time>] [-tu <enum>] [-fgroup <selection>]
              [-xvg <enum>] [-[no]rmpbc] [-[no]pbc] [-sf <file>]
              [-selrpos <enum>] [-select <selection>] [-vel <enum>]
              [-force <enum>] [-atoms <enum>] [-precision <int>]
              [-starttime <time>] [-timestep <time>] [-box <vector>]
```

#### Description

`gmx extract-cluster` can be used to extract trajectory frames that correspond to clusters obtained from running gmx cluster with the -clndx option. The module supports writing all GROMACS supported trajectory file formats.

Included is also a selection of possible options to change additional information.

It is possible to write only a selection of atoms to the output trajectory files for each cluster.

### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Extra index groups

**-clusters [<.ndx>] (cluster.ndx)**
Name of index file containing frame indices for each cluster, obtained from gmx cluster -clndx.

Options to specify output files:

**-o [<.xtc/.trr/...>] (trajout.xtc)**
Prefix for the name of the trajectory file written for each cluster.: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-select <selection>**
Selection of atoms to write to the file

**-vel <enum> (preserved-if-present)**
Save velocities from frame if possible: preserved-if-present, always, never

**-force <enum> (preserved-if-present)**
Save forces from frame if possible: preserved-if-present, always, never

**-atoms <enum> (preserved-if-present)**
Decide on providing new atom information from topology or using current frame atom information: preserved-if-present, always-from-structure, never, always

**-precision <int> (3)**
Set output precision to custom value

**-starttime <time> (0)**
Change start time for first frame

**-timestep <time> (0)**
Change time between different frames

**-box <vector>**
New diagonal box vector for output frame

### 3.11.33 gmx filter

#### Synopsis

```
gmx filter [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>
↪]]
           [-ol [<.xtc/.trr/...>]] [-oh [<.xtc/.trr/...>]]
           [-b <time>] [-e <time>] [-dt <time>] [-[no]w] [-nf <int>]
           [-[no]all] [-[no]nojump] [-[no]fit]
```

#### Description

`gmx filter` performs frequency filtering on a trajectory. The filter shape is cos(pi t/A) + 1 from -A to +A, where A is given by the option `-nf` times the time step in the input trajectory. This filter reduces fluctuations with period A by 85%, with period 2*A by 50% and with period 3*A by 17% for low-pass filtering. Both a low-pass and high-pass filtered trajectory can be written.

Option `-ol` writes a low-pass filtered trajectory. A frame is written every `-nf` input frames. This ratio of filter length and output interval ensures a good suppression of aliasing of high-frequency motion, which is useful for making smooth movies. Also averages of properties which are linear in the coordinates are preserved, since all input frames are weighted equally in the output. When all frames are needed, use the `-all` option.

Option `-oh` writes a high-pass filtered trajectory. The high-pass filtered coordinates are added to the coordinates from the structure file. When using high-pass filtering use `-fit` or make sure you use a trajectory that has been fitted on the coordinates in the structure file.

### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**-ol [<.xtc/.trr/...>] (lowpass.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-oh [<.xtc/.trr/...>] (highpass.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-nf <int> (10)**
Sets the filter length as well as the output interval for low-pass filtering

**-[no]all (no)**
Write all low-pass filtered frames

**-[no]nojump (yes)**
Remove jumps of atoms across the box

**-[no]fit (no)**
Fit all frames to a reference structure

### 3.11.34 gmx freevolume

#### Synopsis

```
gmx freevolume [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]]
               [-n [<.ndx>]] [-o [<.xvg>]] [-b <time>] [-e <time>]
               [-dt <time>] [-tu <enum>] [-fgroup <selection>]
               [-xvg <enum>] [-[no]rmpbc] [-sf <file>]
               [-selrpos <enum>] [-select <selection>] [-radius <real>
↪]
               [-seed <int>] [-ninsert <int>]
```

#### Description

`gmx freevolume` calculates the free volume in a box as a function of time. The free volume is plotted as a fraction of the total volume. The program tries to insert a probe with a given radius, into the simulations box and if the distance between the probe and any atom is less than the sums of the van der Waals radii of both atoms, the position is considered to be occupied, i.e. non-free. By using a probe radius of 0, the true free volume is computed. By using a larger radius, e.g. 0.14 nm, roughly corresponding to a water molecule, the free volume for a hypothetical particle with that size will be produced. Note however, that since atoms are treated as hard-spheres these number are very approximate, and typically only relative changes are meaningful, for instance by doing a series of simulations at different temperature.

The group specified by the selection is considered to delineate non-free volume. The number of insertions per unit of volume is important to get a converged result. About 1000/nm^3 yields an overall standard deviation that is determined by the fluctuations in the trajectory rather than by the fluctuations due to the random numbers.

The results are critically dependent on the van der Waals radii; we recommend to use the values due to Bondi (1964).

The Fractional Free Volume (FFV) that some authors like to use is given by 1 - 1.3*(1-Free Volume). This value is printed on the terminal.

#### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**`-o` [<.xvg>] (freevolume.xvg) (Optional)**
Computed free volume

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-select <selection>**
Atoms that are considered as part of the excluded volume

**-radius <real> (0)**
Radius of the probe to be inserted (nm, 0 yields the true free volume)

**-seed <int> (0)**
Seed for random number generator (0 means generate).

**-ninsert <int> (1000)**
Number of probe insertions per cubic nm to try for each frame in the trajectory.

### 3.11.35 gmx gangle

#### Synopsis

```
gmx gangle [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>
↪]]
           [-oav [<.xvg>]] [-oall [<.xvg>]] [-oh [<.xvg>]]
           [-b <time>] [-e <time>] [-dt <time>] [-tu <enum>]
           [-fgroup <selection>] [-xvg <enum>] [-[no]rmpbc]
           [-[no]pbc] [-sf <file>] [-selrpos <enum>]
           [-seltype <enum>] [-g1 <enum>] [-g2 <enum>] [-binw <real>
↪]
           [-group1 <selection>] [-group2 <selection>]
```

## Description

`gmx gangle` computes different types of angles between vectors. It supports both vectors defined by two positions and normals of planes defined by three positions. The z axis or the local normal of a sphere can also be used as one of the vectors. There are also convenience options 'angle' and 'dihedral' for calculating bond angles and dihedrals defined by three/four positions.

The type of the angle is specified with `-g1` and `-g2`. If `-g1` is `angle` or `dihedral`, `-g2` should not be specified. In this case, `-group1` should specify one or more selections, and each should contain triplets or quartets of positions that define the angles to be calculated.

If `-g1` is `vector` or `plane`, `-group1` should specify selections that contain either pairs (`vector`) or triplets (`plane`) of positions. For vectors, the positions set the endpoints of the vector, and for planes, the three positions are used to calculate the normal of the plane. In both cases, `-g2` specifies the other vector to use (see below).

With `-g2 vector` or `-g2 plane`, `-group2` should specify another set of vectors. `-group1` and `-group2` should specify the same number of selections. It is also allowed to only have a single selection for one of the options, in which case the same selection is used with each selection in the other group. Similarly, for each selection in `-group1`, the corresponding selection in `-group2` should specify the same number of vectors or a single vector. In the latter case, the angle is calculated between that single vector and each vector from the other selection.

With `-g2 sphnorm`, each selection in `-group2` should specify a single position that is the center of the sphere. The second vector is calculated as the vector from the center to the midpoint of the positions specified by `-group1`.

With `-g2 z`, `-group2` is not necessary, and angles between the first vectors and the positive Z axis are calculated.

With `-g2 t0`, `-group2` is not necessary, and angles are calculated from the vectors as they are in the first frame.

There are three options for output: `-oav` writes an xvg file with the time and the average angle for each frame. `-oall` writes all the individual angles. `-oh` writes a histogram of the angles. The bin width can be set with `-binw`. For `-oav` and `-oh`, separate average/histogram is computed for each selection in `-group1`.

## Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**`-oav` [<.xvg>] (angaver.xvg) (Optional)**
Average angles as a function of time

**-oall [<.xvg>] (angles.xvg) (Optional)**
All angles as a function of time

**-oh [<.xvg>] (anghist.xvg) (Optional)**
Histogram of the angles

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-seltype <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-g1 <enum> (angle)**
Type of analysis/first vector group: angle, dihedral, vector, plane

**-g2 <enum> (none)**
Type of second vector group: none, vector, plane, t0, z, sphnorm

**-binw <real> (1)**
Binwidth for -oh in degrees

**-group1 <selection>**
First analysis/vector selection

**-group2 <selection>**
Second analysis/vector selection

### 3.11.36 gmx genconf

#### Synopsis

```
gmx genconf [-f [<.gro/.g96/...>]] [-trj [<.xtc/.trr/...>]]
            [-o [<.gro/.g96/...>]] [-nbox <vector>] [-dist <vector>]
            [-seed <int>] [-[no]rot] [-maxrot <vector>]
            [-[no]renumber]
```

#### Description

`gmx genconf` multiplies a given coordinate file by simply stacking them on top of each other, like a small child playing with wooden blocks. The program makes a grid of *user-defined* proportions (`-nbox`), and interspaces the grid point with an extra space `-dist`.

When option `-rot` is used the program does not check for overlap between molecules on grid points. It is recommended to make the box in the input file at least as big as the coordinates + van der Waals radius.

If the optional trajectory file is given, conformations are not generated, but read from this file and translated appropriately to build the grid.

#### Options

Options to specify input files:

**`-f` [<.gro/.g96/...>] (conf.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**`-trj` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

Options to specify output files:

**`-o` [<.gro/.g96/...>] (out.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

Other options:

**`-nbox` <vector> (1 1 1)**
Number of boxes

**`-dist` <vector> (0 0 0)**
Distance between boxes

**`-seed` <int> (0)**
Random generator seed (0 means generate)

**`-[no]rot` (no)**
Randomly rotate conformations

**`-maxrot` <vector> (180 180 180)**
Maximum random rotation

**-[no]renumber** (yes)
Renumber residues

### Known Issues

- The program should allow for random displacement of lattice points.

## 3.11.37 gmx genion

### Synopsis

```
gmx genion [-s [<.tpr>]] [-n [<.ndx>]] [-p [<.top>]]
           [-o [<.gro/.g96/...>]] [-np <int>] [-pname <string>]
           [-pq <int>] [-nn <int>] [-nname <string>] [-nq <int>]
           [-rmin <real>] [-seed <int>] [-conc <real>]␣
↪[-[no]neutral]
```

### Description

`gmx genion` randomly replaces solvent molecules with monoatomic ions. The group of solvent molecules should be continuous and all molecules should have the same number of atoms. The user should add the ion molecules to the topology file or use the `-p` option to automatically modify the topology.

The ion molecule type, residue and atom names in all force fields are the capitalized element names without sign. This molecule name should be given with `-pname` or `-nname`, and the `[molecules]` section of your topology updated accordingly, either by hand or with `-p`. Do not use an atom name instead!

Ions which can have multiple charge states get the multiplicity added, without sign, for the uncommon states only.

For larger ions, e.g. sulfate we recommended using *gmx insert-molecules* (page 248).

### Options

Options to specify input files:

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify input/output files:

**-p [<.top>] (topol.top) (Optional)**
Topology file

Options to specify output files:

**-o [<.gro/.g96/...>] (out.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

Other options:

**-np <int> (0)**
Number of positive ions

**-pname <string> (NA)**
Name of the positive ion

**-pq <int> (1)**
Charge of the positive ion

**-nn <int> (0)**
Number of negative ions

**-nname <string> (CL)**
Name of the negative ion

**-nq <int> (-1)**
Charge of the negative ion

**-rmin <real> (0.6)**
Minimum distance between ions and non-solvent

**-seed <int> (0)**
Seed for random number generator (0 means generate)

**-conc <real> (0)**
Specify salt concentration (mol/liter). This will add sufficient ions to reach up to the specified concentration as computed from the volume of the cell in the input *.tpr* (page 586) file. Overrides the `-np` and `-nn` options.

**-[no]neutral (no)**
This option will add enough ions to neutralize the system. These ions are added on top of those specified with `-np`/`-nn` or `-conc`.

### Known Issues

- If you specify a salt concentration existing ions are not taken into account. In effect you therefore specify the amount of salt to be added.

## 3.11.38 gmx genrestr

### Synopsis

```
gmx genrestr [-f [<.gro/.g96/...>]] [-n [<.ndx>]] [-o [<.itp>]]
             [-of [<.ndx>]] [-fc <vector>] [-freeze <real>]
             [-[no]disre] [-disre_dist <real>] [-disre_frac <real>]
             [-disre_up2 <real>] [-cutoff <real>] [-[no]constr]
```

## Description

`gmx genrestr` produces an #include file for a topology containing a list of atom numbers and three force constants for the *x*-, *y*-, and *z*-direction based on the contents of the `-f` file. A single isotropic force constant may be given on the command line instead of three components.

WARNING: Position restraints are interactions within molecules, therefore they must be included within the correct `[ moleculetype ]` block in the topology. The atom indices within the `[ position_restraints ]` block must be within the range of the atom indices for that molecule type. Since the atom numbers in every moleculetype in the topology start at 1 and the numbers in the input file for `gmx genrestr` number consecutively from 1, `gmx genrestr` will only produce a useful file for the first molecule. You may wish to edit the resulting index file to remove the lines for later atoms, or construct a suitable index group to provide as input to `gmx genrestr`.

The `-of` option produces an index file that can be used for freezing atoms. In this case, the input file must be a *.pdb* (page 581) file.

With the `-disre` option, half a matrix of distance restraints is generated instead of position restraints. With this matrix, that one typically would apply to Calpha atoms in a protein, one can maintain the overall conformation of a protein without tieing it to a specific position (as with position restraints).

## Options

Options to specify input files:

**`-f` [<.gro/.g96/...>] (conf.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-o` [<.itp>] (posre.itp)**
Include file for topology

**`-of` [<.ndx>] (freeze.ndx) (Optional)**
Index file

Other options:

**`-fc` <vector> (1000 1000 1000)**
Force constants (kJ/mol nm^2)

**`-freeze` <real> (0)**
If the `-of` option or this one is given an index file will be written containing atom numbers of all atoms that have a B-factor less than the level given here

**`-[no]disre` (no)**
Generate a distance restraint matrix for all the atoms in index

**`-disre_dist` <real> (0.1)**
Distance range around the actual distance for generating distance restraints

**`-disre_frac` <real> (0)**
Fraction of distance to be used as interval rather than a fixed distance. If the fraction of the distance that you specify here is less than the distance given in the previous option, that one is used instead.

**-disre_up2 <real> (1)**
Distance between upper bound for distance restraints, and the distance at which the force becomes constant (see manual)

**-cutoff <real> (-1)**
Only generate distance restraints for atoms pairs within cutoff (nm)

**-[no]constr (no)**
Generate a constraint matrix rather than distance restraints. Constraints of type 2 will be generated that do generate exclusions.

### 3.11.39 gmx grompp

#### Synopsis

```
gmx grompp [-f [<.mdp>]] [-c [<.gro/.g96/...>]] [-r [<.gro/.g96/...>
↪]]
           [-rb [<.gro/.g96/...>]] [-n [<.ndx>]] [-p [<.top>]]
           [-t [<.trr/.cpt/...>]] [-e [<.edr>]] [-qmi [<.inp>]]
           [-ref [<.trr/.cpt/...>]] [-po [<.mdp>]] [-pp [<.top>]]
           [-o [<.tpr>]] [-imd [<.gro>]] [-[no]v] [-time <real>]
           [-[no]rmvsbds] [-maxwarn <int>] [-[no]zero] [-[no]renum]
```

#### Description

`gmx grompp` (the gromacs preprocessor) reads a molecular topology file, checks the validity of the file, expands the topology from a molecular description to an atomic description. The topology file contains information about molecule types and the number of molecules, the preprocessor copies each molecule as needed. There is no limitation on the number of molecule types. Bonds and bond-angles can be converted into constraints, separately for hydrogens and heavy atoms. Then a coordinate file is read and velocities can be generated from a Maxwellian distribution if requested. `gmx grompp` also reads parameters for *gmx mdrun* (page 258) (eg. number of MD steps, time step, cut-off). Eventually a binary file is produced that can serve as the sole input file for the MD program.

`gmx grompp` uses the atom names from the topology file. The atom names in the coordinate file (option `-c`) are only read to generate warnings when they do not match the atom names in the topology. Note that the atom names are irrelevant for the simulation as only the atom types are used for generating interaction parameters.

`gmx grompp` uses a built-in preprocessor to resolve includes, macros, etc. The preprocessor supports the following keywords:

```
#ifdef VARIABLE
#ifndef VARIABLE
#else
#endif
#define VARIABLE
#undef VARIABLE
#include "filename"
#include <filename>
```

The functioning of these statements in your topology may be modulated by using the following two flags in your *.mdp* (page 579) file:

```
define = -DVARIABLE1 -DVARIABLE2
include = -I/home/john/doe
```

For further information a C-programming textbook may help you out. Specifying the `-pp` flag will get the pre-processed topology file written out so that you can verify its contents.

When using position restraints, a file with restraint coordinates must be supplied with `-r` (can be the same file as supplied for `-c`). For free energy calculations, separate reference coordinates for the B topology can be supplied with `-rb`, otherwise they will be equal to those of the A topology.

Starting coordinates can be read from trajectory with `-t`. The last frame with coordinates and velocities will be read, unless the `-time` option is used. Only if this information is absent will the coordinates in the `-c` file be used. Note that these velocities will not be used when `gen_vel = yes` in your *.mdp* (page 579) file. An energy file can be supplied with `-e` to read Nose-Hoover and/or Parrinello-Rahman coupling variables.

`gmx grompp` can be used to restart simulations (preserving continuity) by supplying just a checkpoint file with `-t`. However, for simply changing the number of run steps to extend a run, using *gmx convert-tpr* (page 178) is more convenient than `gmx grompp`. You then supply the old checkpoint file directly to *gmx mdrun* (page 258) with `-cpi`. If you wish to change the ensemble or things like output frequency, then supplying the checkpoint file to `gmx grompp` with `-t` along with a new *.mdp* (page 579) file with `-f` is the recommended procedure. Actually preserving the ensemble (if possible) still requires passing the checkpoint file to *gmx mdrun* (page 258) `-cpi`.

By default, all bonded interactions which have constant energy due to virtual site constructions will be removed. If this constant energy is not zero, this will result in a shift in the total energy. All bonded interactions can be kept by turning off `-rmvsbds`. Additionally, all constraints for distances which will be constant anyway because of virtual site constructions will be removed. If any constraints remain which involve virtual sites, a fatal error will result.

To verify your run input file, please take note of all warnings on the screen, and correct where necessary. Do also look at the contents of the `mdout.mdp` file; this contains comment lines, as well as the input that `gmx grompp` has read. If in doubt, you can start `gmx grompp` with the `-debug` option which will give you more information in a file called `grompp.log` (along with real debug info). You can see the contents of the run input file with the *gmx dump* (page 205) program. *gmx check* (page 165) can be used to compare the contents of two run input files.

The `-maxwarn` option can be used to override warnings printed by `gmx grompp` that otherwise halt output. In some cases, warnings are harmless, but usually they are not. The user is advised to carefully interpret the output messages before attempting to bypass them with this option.

### Options

Options to specify input files:

**`-f` [<.mdp>] (grompp.mdp)**
grompp input file with MD parameters

**`-c` [<.gro/.g96/...>] (conf.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**-r [<.gro/.g96/...>] (restraint.gro) (Optional)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**-rb [<.gro/.g96/...>] (restraint.gro) (Optional)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

**-p [<.top>] (topol.top)**
Topology file

**-t [<.trr/.cpt/...>] (traj.trr) (Optional)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**-e [<.edr>] (ener.edr) (Optional)**
Energy file

**-qmi [<.inp>] (topol-qmmm.inp) (Optional)**
Input file for QM program

Options to specify input/output files:

**-ref [<.trr/.cpt/...>] (rotref.trr) (Optional)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

Options to specify output files:

**-po [<.mdp>] (mdout.mdp)**
grompp input file with MD parameters

**-pp [<.top>] (processed.top) (Optional)**
Topology file

**-o [<.tpr>] (topol.tpr)**
Portable xdr run input file

**-imd [<.gro>] (imdgroup.gro) (Optional)**
Coordinate file in Gromos-87 format

Other options:

**-[no]v (no)**
Be loud and noisy

**-time <real> (-1)**
Take frame at or first after this time.

**-[no]rmvsbds (yes)**
Remove constant bonded interactions with virtual sites

**-maxwarn <int> (0)**
Number of allowed warnings during input processing. Not for normal use and may generate unstable systems

**-[no]zero (no)**
Set parameters for bonded interactions without defaults to zero instead of generating an error

**-[no]renum (yes)**
Renumber atomtypes and minimize number of atomtypes

### 3.11.40 gmx gyrate

#### Synopsis

```
gmx gyrate [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>
↪]]
           [-o [<.xvg>]] [-b <time>] [-e <time>] [-dt <time>]
           [-tu <enum>] [-fgroup <selection>] [-xvg <enum>]
           [-[no]rmpbc] [-sf <file>] [-selrpos <enum>]
           [-seltype <enum>] [-sel <selection>] [-mode <enum>]
```

#### Description

`gmx gyrate` computes the radius of gyration of a molecule and the radii of gyration about the *x*-, *y*- and *z*-axes, as a function of time. The atoms are explicitly mass weighted.

The axis components corresponds to the mass-weighted root-mean-square of the radii components orthogonal to each axis, for example:

Rg(x) = sqrt((sum_i w_i (R_i(y)^2 + R_i(z)^2))/(sum_i w_i)).

where w_i is the weight value in the given situation (mass, charge, unit)

Note that this is a new implementation of the gyrate utility added in GROMACS 2024. If you need the old one, use `gmx gyrate-legacy`.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**-o [<.xvg>] (gyrate-taf.xvg)**
Filename for gyrate plot output

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-seltype <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-sel <selection>**
Select group to compute gyrate radius

**-mode <enum> (mass)**
Atom weighting mode: mass, charge, geometry

### 3.11.41 gmx gyrate-legacy

#### Synopsis

```
gmx gyrate-legacy [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]]
                  [-n [<.ndx>]] [-o [<.xvg>]] [-acf [<.xvg>]] [-b <time>]
                  [-e <time>] [-dt <time>] [-[no]w] [-xvg <enum>]
                  [-nmol <int>] [-[no]q] [-[no]p] [-[no]moi] [-nz <int>]
                  [-acflen <int>] [-[no]normalize] [-P <enum>]
                  [-fitfn <enum>] [-beginfit <real>] [-endfit <real>]
```

#### Description

`gmx gyrate-legacy` computes the radius of gyration of a molecule and the radii of gyration about the *x*-, *y*- and *z*-axes, as a function of time. The atoms are explicitly mass weighted.

The axis components corresponds to the mass-weighted root-mean-square of the radii components orthogonal to each axis, for example:

Rg(x) = sqrt((sum_i m_i (R_i(y)^2 + R_i(z)^2))/(sum_i m_i)).

With the `-nmol` option the radius of gyration will be calculated for multiple molecules by splitting the analysis group in equally sized parts.

With the option `-nz` 2D radii of gyration in the *x*-*y* plane of slices along the *z*-axis are calculated.

### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**-o [<.xvg>] (gyrate.xvg)**
xvgr/xmgr file

**-acf [<.xvg>] (moi-acf.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-nmol <int> (1)**
The number of molecules to analyze

**-[no]q (no)**
Use absolute value of the charge of an atom as weighting factor instead of mass

**-[no]p (no)**
Calculate the radii of gyration about the principal axes.

**-[no]moi (no)**
Calculate the moments of inertia (defined by the principal axes).

**-nz <int> (0)**
Calculate the 2D radii of gyration of this number of slices along the z-axis

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### 3.11.42 gmx h2order

#### Synopsis

```
gmx h2order [-f [<.xtc/.trr/...>]] [-n [<.ndx>]] [-nm [<.ndx>]]
            [-s [<.tpr>]] [-o [<.xvg>]] [-b <time>] [-e <time>]
            [-dt <time>] [-[no]w] [-xvg <enum>] [-d <enum>]
            [-sl <int>]
```

#### Description

`gmx h2order` computes the orientation of water molecules with respect to the normal of the box. The program determines the average cosine of the angle between the dipole moment of water and an axis of the box. The box is divided in slices and the average orientation per slice is printed. Each water molecule is assigned to a slice, per time frame, based on the position of the oxygen. When `-nm` is used, the angle between the water dipole and the axis from the center of mass to the oxygen is calculated instead of the angle between the dipole and a box axis.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-n [<.ndx>] (index.ndx)**
Index file

**-nm [<.ndx>] (index.ndx) (Optional)**
Index file

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

Options to specify output files:

**-o [<.xvg>] (order.xvg)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-d <enum> (Z)**
Take the normal on the membrane in direction X, Y or Z.: Z, Y, X

**-sl <int> (0)**
Calculate order parameter as function of boxlength, dividing the box in this number of slices.

### Known Issues

- The program assigns whole water molecules to a slice, based on the first atom of three in the index file group. It assumes an order O,H,H. Name is not important, but the order is. If this demand is not met, assigning molecules to slices is different.

## 3.11.43 gmx hbond

### Synopsis

```
gmx hbond [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>
↪]]
          [-o [<.ndx>]] [-num [<.xvg>]] [-dist [<.xvg>]]
          [-ang [<.xvg>]] [-dan [<.xvg>]] [-b <time>] [-e <time>]
          [-dt <time>] [-tu <enum>] [-fgroup <selection>]
          [-xvg <enum>] [-[no]rmpbc] [-[no]pbc] [-sf <file>]
          [-selrpos <enum>] [-seltype <enum>] [-r <selection>]
          [-t <selection>] [-[no]m] [-[no]pf] [-cutoff <real>]
          [-hbr <real>] [-hba <real>] [-de <string>] [-ae <string>]
```

### Description

`gmx hbond` allows using geometric definition of hydrogen bonds to define them throughout the structure.

`-r` specifies reference selection, relative to which the search for hydrogen bonds in target selection will develop. Note that all atoms in reference and target selections should be either absolutely identical or non-overlapping at all. Accepts dynamic selection.

`-t` specifies target selection, relative to which the search for hydrogen bonds in reference selection will develop. Note that all atoms in reference and target selections should be either absolutely identical or non-overlapping at all. Accepts dynamic selection.

`-m` forces to merge together information in output index file about hydrogen bonds if they differ only in hydrogen indices. This also means that information about hydrogen atoms in the hydrogen bonds would not be written in output index file at all.

`-pf` forces to write hydrogen bonds for each frame separately instead of writing hydrogen bonds for the whole system. Each information about hydrogen bonds in new frame will be stored in its own section of the output index file.

`-cutoff` is a real value that defines distance from donor to acceptor (and vice versa) that used in neighbor search. Minimum (and also recommended) value is 0.35.

`-hbr` Sets the cutoff that is used when calculating hydrogen bond distances. Recommended value: 0.35.

`-hba` Sets the cutoff that is used when calculating hydrogen bond angles. Recommended value: 30.

`-de` Specifies the atomic elements that will be selected from the topology to check if a given element is a potential hydrogen bond donor.

`-ae` Specifies the atomic elements that will be selected from the topology to check if a given element is a potential hydrogen bond acceptor.

`-num` allows you to get a plot of the number of hydrogen bonds as a function of time at the output.

`-dist` allows you to get a plot of the distance distribution of all hydrogen bonds at the output.

`-ang` allows you to get a plot of the angular distribution of all hydrogen bonds at the output.

`-dan` allows you to get a plot of the number of analyzed donors and acceptors for each frame at the output.

Note that this is a new implementation of the hbond utility added in GROMACS 2024. If you need the old one, use `gmx hbond-legacy`.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**`-o` [<.ndx>] (hbond.ndx)**
Index file that contains selected groups', acceptors', donors' and hydrogens' indices and hydrogen bond pairs between or within selected groups.

**`-num` [<.xvg>] (hbnum.xvg) (Optional)**
Number of hydrogen bonds as a function of time.

**`-dist` [<.xvg>] (hbdist.xvg) (Optional)**
Distance distribution of all hydrogen bonds.

**-ang [<.xvg>] (hbang.xvg) (Optional)**
Angle distribution of all hydrogen bonds.

**-dan [<.xvg>] (hbdan.xvg) (Optional)**
Number of donors and acceptors analyzed for each frame.

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-seltype <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-r <selection>**
Reference selection, relative to which the search for hydrogen bonds in target selection will develop.

**-t <selection>**
Target selection, relative to which the search for hydrogen bonds in reference selection will develop.

**-[no]m (no)**
Merge together information about hydrogen bonds if they differ only in hydrogen indices.

**-[no]pf (no)**
Write hydrogen bonds for each frame separately instead of writing hydrogen bonds for the whole system.

**-cutoff <real> (0.35)**
Distance from donor to acceptor (and vice versa) that used in neighbor search (nm). Must be > 0.

**-hbr <real> (0.35)**
Hydrogen bond cutoff distance, between donor and acceptor (nm). The value must not exceed the neighbor search cutoff and must be > 0.

**-hba <real> (30)**
A-D-H hydrogen bond cutoff angle (degrees). Must be > 0.

**-de <string> (N O)**
Donor elements. Default elements: N, O.

**-ae <string> (N O)**
Acceptor elements. Default elements: N, O.

### 3.11.44 gmx hbond-legacy

#### Synopsis

```
gmx hbond-legacy [-f [<.xtc/.trr/...>]] [-s [<.tpr>]] [-n [<.ndx>]]
              [-num [<.xvg>]] [-g [<.log>]] [-ac [<.xvg>]]
              [-dist [<.xvg>]] [-ang [<.xvg>]] [-hx [<.xvg>]]
              [-hbn [<.ndx>]] [-hbm [<.xpm>]] [-don [<.xvg>]]
              [-dan [<.xvg>]] [-life [<.xvg>]] [-nhbdist [<.xvg>]]
              [-b <time>] [-e <time>] [-dt <time>] [-tu <enum>]
              [-xvg <enum>] [-a <real>] [-r <real>] [-[no]da]
              [-r2 <real>] [-abin <real>] [-rbin <real>]␣
↪[-[no]nitacc]
              [-[no]contact] [-shell <real>] [-fitstart <real>]
              [-fitend <real>] [-temp <real>] [-dump <int>]
              [-max_hb <real>] [-[no]merge] [-nthreads <int>]
              [-acflen <int>] [-[no]normalize] [-P <enum>]
              [-fitfn <enum>] [-beginfit <real>] [-endfit <real>]
```

#### Description

`gmx hbond-legacy` computes and analyzes hydrogen bonds. Hydrogen bonds are determined based on cutoffs for the angle Hydrogen - Donor - Acceptor (zero is extended) and the distance Donor - Acceptor (or Hydrogen - Acceptor using `-noda`). OH and NH groups are regarded as donors, O is an acceptor always, N is an acceptor by default, but this can be switched using `-nitacc`. Dummy hydrogen atoms are assumed to be connected to the first preceding non-hydrogen atom.

You need to specify two groups for analysis, which must be either identical or non-overlapping. All hydrogen bonds between the two groups are analyzed.

If you set `-shell`, you will be asked for an additional index group which should contain exactly one atom. In this case, only hydrogen bonds between atoms within the shell distance from the one atom are considered.

With option -ac, rate constants for hydrogen bonding can be derived with the model of Luzar and Chandler (Nature 379:55, 1996; J. Chem. Phys. 113:23, 2000). If contact kinetics are analyzed by using the -contact option, then n(t) can be defined as either all pairs that are not within contact distance r at

time t (corresponding to leaving the -r2 option at the default value 0) or all pairs that are within distance r2 (corresponding to setting a second cut-off value with option -r2). See mentioned literature for more details and definitions.

**Output:**

- `-num`: number of hydrogen bonds as a function of time.
- `-ac`: average over all autocorrelations of the existence functions (either 0 or 1) of all hydrogen bonds.
- `-dist`: distance distribution of all hydrogen bonds.
- `-ang`: angle distribution of all hydrogen bonds.
- `-hx`: the number of n-n+i hydrogen bonds as a function of time where n and n+i stand for residue numbers and i ranges from 0 to 6. This includes the n-n+3, n-n+4 and n-n+5 hydrogen bonds associated with helices in proteins.
- `-hbn`: all selected groups, donors, hydrogens and acceptors for selected groups, all hydrogen bonded atoms from all groups and all solvent atoms involved in insertion. Output is limited unless `-nomerge` is set.
- `-hbm`: existence matrix for all hydrogen bonds over all frames, this also contains information on solvent insertion into hydrogen bonds. Ordering is identical to that in `-hbn` index file.
- `-dan`: write out the number of donors and acceptors analyzed for each timeframe. This is especially useful when using `-shell`.
- `-nhbdist`: compute the number of HBonds per hydrogen in order to compare results to Raman Spectroscopy.

Note: options `-ac`, `-life`, `-hbn` and `-hbm` require an amount of memory proportional to the total numbers of donors times the total number of acceptors in the selected group(s).

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-num` [<.xvg>] (hbnum.xvg)**
xvgr/xmgr file

**`-g` [<.log>] (hbond.log) (Optional)**
Log file

**`-ac` [<.xvg>] (hbac.xvg) (Optional)**
xvgr/xmgr file

**-dist [<.xvg>] (hbdist.xvg) (Optional)**
xvgr/xmgr file

**-ang [<.xvg>] (hbang.xvg) (Optional)**
xvgr/xmgr file

**-hx [<.xvg>] (hbhelix.xvg) (Optional)**
xvgr/xmgr file

**-hbn [<.ndx>] (hbond.ndx) (Optional)**
Index file

**-hbm [<.xpm>] (hbmap.xpm) (Optional)**
X PixMap compatible matrix file

**-don [<.xvg>] (donor.xvg) (Optional)**
xvgr/xmgr file

**-dan [<.xvg>] (danum.xvg) (Optional)**
xvgr/xmgr file

**-life [<.xvg>] (hblife.xvg) (Optional)**
xvgr/xmgr file

**-nhbdist [<.xvg>] (nhbdist.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-a <real> (30)**
Cutoff angle (degrees, Hydrogen - Donor - Acceptor)

**-r <real> (0.35)**
Cutoff radius (nm, X - Acceptor, see next option)

**-[no]da (yes)**
Use distance Donor-Acceptor (if TRUE) or Hydrogen-Acceptor (FALSE)

**-r2 <real> (0)**
Second cutoff radius. Mainly useful with `-contact` and `-ac`

**-abin <real> (1)**
Binwidth angle distribution (degrees)

**-rbin <real> (0.005)**
Binwidth distance distribution (nm)

**-[no]nitacc (yes)**
Regard nitrogen atoms as acceptors

**-[no]contact (no)**
Do not look for hydrogen bonds, but merely for contacts within the cut-off distance

**-shell <real> (-1)**
when > 0, only calculate hydrogen bonds within # nm shell around one particle

**-fitstart <real> (1)**
Time (ps) from which to start fitting the correlation functions in order to obtain the forward and backward rate constants for HB breaking and formation. With `-gemfit` we suggest `-fitstart 0`

**-fitend <real> (60)**
Time (ps) to which to stop fitting the correlation functions in order to obtain the forward and backward rate constants for HB breaking and formation (only with `-gemfit`)

**-temp <real> (298.15)**
Temperature (K) for computing the Gibbs energy corresponding to HB breaking and reforming

**-dump <int> (0)**
Dump the first N hydrogen bond ACFs in a single *.xvg* (page 589) file for debugging

**-max_hb <real> (0)**
Theoretical maximum number of hydrogen bonds used for normalizing HB autocorrelation function. Can be useful in case the program estimates it wrongly

**-[no]merge (yes)**
H-bonds between the same donor and acceptor, but with different hydrogen are treated as a single H-bond. Mainly important for the ACF. Not compatible with options that depend on knowing a specific hydrogen: `-noad`, `-ang`.

**-nthreads <int> (0)**
Number of threads used for the parallel loop over autocorrelations. nThreads <= 0 means maximum number of threads. Requires linking with OpenMP. The number of threads is limited by the number of cores (before OpenMP v.3 ) or environment variable OMP_THREAD_LIMIT (OpenMP v.3)

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### 3.11.45 gmx helix

#### Synopsis

```
gmx helix [-s [<.tpr>]] [-n [<.ndx>]] [-f [<.xtc/.trr/...>]]
          [-cz [<.gro/.g96/...>]] [-b <time>] [-e <time>]
          [-dt <time>] [-[no]w] [-r0 <int>] [-[no]q] [-[no]F]
          [-[no]db] [-[no]ev] [-ahxstart <int>] [-ahxend <int>]
```

#### Description

`gmx helix` computes all kinds of helix properties. First, the peptide is checked to find the longest helical part, as determined by hydrogen bonds and phi/psi angles. That bit is fitted to an ideal helix around the $z$-axis and centered around the origin. Then the following properties are computed:

- Helix radius (file `radius.xvg`). This is merely the RMS deviation in two dimensions for all Calpha atoms. it is calculated as sqrt((sum_i (x^2(i)+y^2(i)))/N) where N is the number of backbone atoms. For an ideal helix the radius is 0.23 nm.
- Twist (file `twist.xvg`). The average helical angle per residue is calculated. For an alpha-helix it is 100 degrees, for 3-10 helices it will be smaller, and for 5-helices it will be larger.
- Rise per residue (file `rise.xvg`). The helical rise per residue is plotted as the difference in $z$-coordinate between Calpha atoms. For an ideal helix, this is 0.15 nm.
- Total helix length (file `len-ahx.xvg`). The total length of the helix in nm. This is simply the average rise (see above) times the number of helical residues (see below).
- Helix dipole, backbone only (file `dip-ahx.xvg`).
- RMS deviation from ideal helix, calculated for the Calpha atoms only (file `rms-ahx.xvg`).
- Average Calpha - Calpha dihedral angle (file `phi-ahx.xvg`).
- Average phi and psi angles (file `phipsi.xvg`).
- Ellipticity at 222 nm according to Hirst and Brooks.

#### Options

Options to specify input files:

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

**`-n` [<.ndx>] (index.ndx)**
Index file

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

Options to specify output files:

**`-cz` [<.gro/.g96/...>] (zconf.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-r0 <int> (1)**
The first residue number in the sequence

**-[no]q (no)**
Check at every step which part of the sequence is helical

**-[no]F (yes)**
Toggle fit to a perfect helix

**-[no]db (no)**
Print debug info

**-[no]ev (no)**
Write a new 'trajectory' file for ED

**-ahxstart <int> (0)**
First residue in helix

**-ahxend <int> (0)**
Last residue in helix

### 3.11.46 gmx helixorient

#### Synopsis

```
gmx helixorient [-s [<.tpr>]] [-f [<.xtc/.trr/...>]] [-n [<.ndx>]]
                [-oaxis [<.dat>]] [-ocenter [<.dat>]] [-orise [<.xvg>]]
                [-oradius [<.xvg>]] [-otwist [<.xvg>]]
                [-obending [<.xvg>]] [-otilt [<.xvg>]] [-orot [<.xvg>]]
                [-b <time>] [-e <time>] [-dt <time>] [-xvg <enum>]
                [-[no]sidechain] [-[no]incremental]
```

#### Description

`gmx helixorient` calculates the coordinates and direction of the average axis inside an alpha helix, and the direction/vectors of both the Calpha and (optionally) a sidechain atom relative to the axis.

As input, you need to specify an index group with Calpha atoms corresponding to an alpha-helix of continuous residues. Sidechain directions require a second index group of the same size, containing the heavy atom in each residue that should represent the sidechain.

**Note** that this program does not do any fitting of structures.

We need four Calpha coordinates to define the local direction of the helix axis.

The tilt/rotation is calculated from Euler rotations, where we define the helix axis as the local *x*-axis, the residues/Calpha vector as *y*, and the *z*-axis from their cross product. We use the Euler Y-Z-X rotation, meaning we first tilt the helix axis (1) around and (2) orthogonal to the residues vector, and finally apply the (3) rotation around it. For debugging or other purposes, we also write out the actual Euler rotation angles as `theta[1-3].xvg`

### Options

Options to specify input files:

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-oaxis` [<.dat>] (helixaxis.dat)**
Generic data file

**`-ocenter` [<.dat>] (center.dat)**
Generic data file

**`-orise` [<.xvg>] (rise.xvg)**
xvgr/xmgr file

**`-oradius` [<.xvg>] (radius.xvg)**
xvgr/xmgr file

**`-otwist` [<.xvg>] (twist.xvg)**
xvgr/xmgr file

**`-obending` [<.xvg>] (bending.xvg)**
xvgr/xmgr file

**`-otilt` [<.xvg>] (tilt.xvg)**
xvgr/xmgr file

**`-orot` [<.xvg>] (rotation.xvg)**
xvgr/xmgr file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]sidechain (no)**
Calculate sidechain directions relative to helix axis too.

**-[no]incremental (no)**
Calculate incremental rather than total rotation/tilt.

### 3.11.47 gmx help

### 3.11.48 gmx hydorder

#### Synopsis

```
gmx hydorder [-f [<.xtc/.trr/...>]] [-n [<.ndx>]] [-s [<.tpr>]]
             [-o [<.xpm> [...]]] [-or [<.out> [...]]]
             [-Spect [<.out> [...]]] [-b <time>] [-e <time>]
             [-dt <time>] [-[no]w] [-d <enum>] [-bw <real>]
             [-sgang1 <real>] [-sgang2 <real>] [-tblock <int>]
             [-nlevel <int>]
```

#### Description

`gmx hydorder` computes the tetrahedrality order parameters around a given atom. Both angle an distance order parameters are calculated. See P.-L. Chau and A.J. Hardwick, Mol. Phys., 93, (1998), 511-518. for more details.

`gmx hydorder` calculates the order parameter in a 3d-mesh in the box, and with 2 phases in the box gives the user the option to define a 2D interface in time separating the faces by specifying parameters `-sgang1` and `-sgang2` (it is important to select these judiciously).

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-n [<.ndx>] (index.ndx)**
Index file

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

Options to specify output files:

**-o [<.xpm> [...]] (intf.xpm)**
X PixMap compatible matrix file

**-or [<.out> [...]] (raw.out) (Optional)**
Generic output file

**-Spect [<.out> [...]] (intfspect.out) (Optional)**
Generic output file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-d <enum> (z)**
Direction of the normal on the membrane: z, x, y

**-bw <real> (1)**
Binwidth of box mesh

**-sgang1 <real> (1)**
tetrahedral angle parameter in Phase 1 (bulk)

**-sgang2 <real> (1)**
tetrahedral angle parameter in Phase 2 (bulk)

**-tblock <int> (1)**
Number of frames in one time-block average

**-nlevel <int> (100)**
Number of Height levels in 2D - XPixMaps

### 3.11.49 gmx insert-molecules

#### Synopsis

```
gmx insert-molecules [-f [<.gro/.g96/...>]] [-ci [<.gro/.g96/...>]]
             [-ip [<.dat>]] [-n [<.ndx>]] [-o [<.gro/.g96/...>]]
             [-replace <selection>] [-sf <file>] [-selrpos <enum>]
             [-box <vector>] [-nmol <int>] [-conc <real>]
             [-try <int>] [-seed <int>] [-radius <real>]
             [-scale <real>] [-dr <vector>] [-rot <enum>]
```

#### Description

`gmx insert-molecules` inserts `-nmol` copies of the system specified in the `-ci` input file. The number of copies can also be determined by the concentration `-conc` in mol/liter and box volume. The insertions take place either into vacant space in the solute conformation given with `-f`, or into an empty box given by `-box`. Specifying both `-f` and `-box` behaves like `-f`, but places a new box around the solute before insertions. Any velocities present are discarded.

It is possible to also insert into a solvated configuration and replace solvent atoms with the inserted atoms. To do this, use `-replace` to specify a selection that identifies the atoms that can be replaced.

The tool assumes that all molecules in this selection consist of single residues: each residue from this selection that overlaps with the inserted molecules will be removed instead of preventing insertion.

By default, the insertion positions are random (with initial seed specified by `-seed`). The program iterates until `-nmol` molecules have been inserted in the box. Molecules are not inserted where the distance between any existing atom and any atom of the inserted molecule is less than the sum based on the van der Waals radii of both atoms. A database (`vdwradii.dat`) of van der Waals radii is read by the program, and the resulting radii scaled by `-scale`. If radii are not found in the database, those atoms are assigned the (pre-scaled) distance `-radius`. Note that the usefulness of those radii depends on the atom names, and thus varies widely with force field.

A total of `-nmol` * `-try` insertion attempts are made before giving up. Increase `-try` if you have several small holes to fill. Option `-rot` specifies whether the insertion molecules are randomly oriented before insertion attempts.

Alternatively, the molecules can be inserted only at positions defined in positions.dat (`-ip`). That file should have 3 columns (x,y,z), that give the displacements compared to the input molecule position (`-ci`). Hence, if that file should contain the absolute positions, the molecule must be centered on (0,0,0) before using `gmx insert-molecules` (e.g. from *gmx editconf* (page 208) `-center`). Comments in that file starting with # are ignored. Option `-dr` defines the maximally allowed displacements during insertial trials. `-try` and `-rot` work as in the default mode (see above).

### Options

Options to specify input files:

**`-f` [<.gro/.g96/...>] (protein.gro) (Optional)**
Existing configuration to insert into: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**`-ci` [<.gro/.g96/...>] (insert.gro)**
Configuration to insert: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**`-ip` [<.dat>] (positions.dat) (Optional)**
Predefined insertion trial positions

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**`-o` [<.gro/.g96/...>] (out.gro)**
Output configuration after insertion: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

Other options:

**`-replace` <selection>**
Atoms that can be removed if overlapping

**`-sf` <file>**
Provide selections from files

**`-selrpos` <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-box <vector> (0 0 0)**
Box size (in nm)

**-nmol <int> (0)**
Number of extra molecules to insert

**-conc <real> (0)**
Concentration (in mol/liter) of extra molecules to insert. This overrides -nmol

**-try <int> (10)**
Try inserting -nmol times -try times

**-seed <int> (0)**
Random generator seed (0 means generate)

**-radius <real> (0.105)**
Default van der Waals distance

**-scale <real> (0.57)**
Scale factor to multiply Van der Waals radii from the database in share/gromacs/top/vdwradii.dat. The default value of 0.57 yields density close to 1000 g/l for proteins in water.

**-dr <vector> (0 0 0)**
Allowed displacement in x/y/z from positions in -ip file

**-rot <enum> (xyz)**
Rotate inserted molecules randomly: xyz, z, none

### 3.11.50 gmx lie

#### Synopsis

```
gmx lie [-f [<.edr>]] [-o [<.xvg>]] [-b <time>] [-e <time>] [-dt
↪<time>]
        [-[no]w] [-xvg <enum>] [-Elj <real>] [-Eqq <real>]
        [-Clj <real>] [-Cqq <real>] [-ligand <string>]
```

#### Description

gmx lie computes a free energy estimate based on an energy analysis from nonbonded energies. One needs an energy file with the following components: Coul-(A-B) LJ-SR (A-B) etc.

To utilize g_lie correctly, two simulations are required: one with the molecule of interest bound to its receptor and one with the molecule in water. Both need to utilize energygrps such that Coul-SR(A-B), LJ-SR(A-B), etc. terms are written to the *.edr* (page 576) file. Values from the molecule-in-water simulation are necessary for supplying suitable values for -Elj and -Eqq.

### Options

Options to specify input files:

**-f [<.edr>] (ener.edr)**
Energy file

Options to specify output files:

**-o [<.xvg>] (lie.xvg)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-Elj <real> (0)**
Lennard-Jones interaction between ligand and solvent

**-Eqq <real> (0)**
Coulomb interaction between ligand and solvent

**-Clj <real> (0.181)**
Factor in the LIE equation for Lennard-Jones component of energy

**-Cqq <real> (0.5)**
Factor in the LIE equation for Coulomb component of energy

**-ligand <string> (none)**
Name of the ligand in the energy file

## 3.11.51 gmx make_edi

### Synopsis

```
gmx make_edi [-f [<.trr/.cpt/...>]] [-eig [<.xvg>]]
             [-s [<.tpr/.gro/...>]] [-n [<.ndx>]]
             [-tar [<.gro/.g96/...>]] [-ori [<.gro/.g96/...>]]
             [-o [<.edi>]] [-xvg <enum>] [-mon <string>]
             [-linfix <string>] [-linacc <string>] [-radfix <string>
↪]
             [-radacc <string>] [-radcon <string>] [-flood <string>]
             [-outfrq <int>] [-slope <real>] [-linstep <string>]
```

```
[-accdir <string>] [-radstep <real>] [-maxedsteps <int>
↪]
[-eqsteps <int>] [-deltaF0 <real>] [-deltaF <real>]
[-tau <real>] [-Eflnull <real>] [-T <real>]
[-alpha <real>] [-[no]restrain] [-[no]hessian]
[-[no]harmonic] [-constF <string>]
```

## Description

gmx make_edi generates an essential dynamics (ED) sampling input file to be used with mdrun based on eigenvectors of a covariance matrix (*gmx covar* (page 181)) or from a normal modes analysis (*gmx nmeig* (page 269)). ED sampling can be used to manipulate the position along collective coordinates (eigenvectors) of (biological) macromolecules during a simulation. Particularly, it may be used to enhance the sampling efficiency of MD simulations by stimulating the system to explore new regions along these collective coordinates. A number of different algorithms are implemented to drive the system along the eigenvectors (-linfix, -linacc, -radfix, -radacc, -radcon), to keep the position along a certain (set of) coordinate(s) fixed (-linfix), or to only monitor the projections of the positions onto these coordinates (-mon).

References:

A. Amadei, A.B.M. Linssen, B.L. de Groot, D.M.F. van Aalten and H.J.C. Berendsen; An efficient method for sampling the essential subspace of proteins., J. Biomol. Struct. Dyn. 13:615-626 (1996)

B.L. de Groot, A. Amadei, D.M.F. van Aalten and H.J.C. Berendsen; Towards an exhaustive sampling of the configurational spaces of the two forms of the peptide hormone guanylin, J. Biomol. Struct. Dyn. 13 : 741-751 (1996)

B.L. de Groot, A.Amadei, R.M. Scheek, N.A.J. van Nuland and H.J.C. Berendsen; An extended sampling of the configurational space of HPr from E. coli Proteins: Struct. Funct. Gen. 26: 314-322 (1996)

You will be prompted for one or more index groups that correspond to the eigenvectors, reference structure, target positions, etc.

-mon: monitor projections of the coordinates onto selected eigenvectors.

-linfix: perform fixed-step linear expansion along selected eigenvectors.

-linacc: perform acceptance linear expansion along selected eigenvectors. (steps in the desired directions will be accepted, others will be rejected).

-radfix: perform fixed-step radius expansion along selected eigenvectors.

-radacc: perform acceptance radius expansion along selected eigenvectors. (steps in the desired direction will be accepted, others will be rejected). **Note:** by default the starting MD structure will be taken as origin of the first expansion cycle for radius expansion. If -ori is specified, you will be able to read in a structure file that defines an external origin.

-radcon: perform acceptance radius contraction along selected eigenvectors towards a target structure specified with -tar.

NOTE: each eigenvector can be selected only once.

-outfrq: frequency (in steps) of writing out projections etc. to *.xvg* (page 589) file

-slope: minimal slope in acceptance radius expansion. A new expansion cycle will be started if the spontaneous increase of the radius (in nm/step) is less than the value specified.

`-maxedsteps`: maximum number of steps per cycle in radius expansion before a new cycle is started.

Note on the parallel implementation: since ED sampling is a 'global' thing (collective coordinates etc.), at least on the 'protein' side, ED sampling is not very parallel-friendly from an implementation point of view. Because parallel ED requires some extra communication, expect the performance to be lower as in a free MD simulation, especially on a large number of ranks and/or when the ED group contains a lot of atoms.

Please also note that if your ED group contains more than a single protein, then the *.tpr* (page 586) file must contain the correct PBC representation of the ED group. Take a look on the initial RMSD from the reference structure, which is printed out at the start of the simulation; if this is much higher than expected, one of the ED molecules might be shifted by a box vector.

All ED-related output of `mdrun` (specify with `-eo`) is written to a *.xvg* (page 589) file as a function of time in intervals of OUTFRQ steps.

**Note** that you can impose multiple ED constraints and flooding potentials in a single simulation (on different molecules) if several *.edi* (page 576) files were concatenated first. The constraints are applied in the order they appear in the *.edi* (page 576) file. Depending on what was specified in the *.edi* (page 576) input file, the output file contains for each ED dataset

- the RMSD of the fitted molecule to the reference structure (for atoms involved in fitting prior to calculating the ED constraints)
- projections of the positions onto selected eigenvectors

FLOODING:

with `-flood`, you can specify which eigenvectors are used to compute a flooding potential, which will lead to extra forces expelling the structure out of the region described by the covariance matrix. If you switch -restrain the potential is inverted and the structure is kept in that region.

The origin is normally the average structure stored in the `eigvec.trr` file. It can be changed with `-ori` to an arbitrary position in configuration space. With `-tau`, `-deltaF0`, and `-Eflnull` you control the flooding behaviour. Efl is the flooding strength, it is updated according to the rule of adaptive flooding. Tau is the time constant of adaptive flooding, high tau means slow adaption (i.e. growth). DeltaF0 is the flooding strength you want to reach after tau ps of simulation. To use constant Efl set `-tau` to zero.

`-alpha` is a fudge parameter to control the width of the flooding potential. A value of 2 has been found to give good results for most standard cases in flooding of proteins. alpha basically accounts for incomplete sampling, if you sampled further the width of the ensemble would increase, this is mimicked by alpha > 1. For restraining, alpha < 1 can give you smaller width in the restraining potential.

RESTART and FLOODING: If you want to restart a crashed flooding simulation please find the values deltaF and Efl in the output file and manually put them into the *.edi* (page 576) file under DELTA_F0 and EFL_NULL.

### Options

Options to specify input files:

**-f [<.trr/.cpt/...>] (eigenvec.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**-eig [<.xvg>] (eigenval.xvg) (Optional)**
xvgr/xmgr file

**-s [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

**-tar [<.gro/.g96/...>] (target.gro) (Optional)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**-ori [<.gro/.g96/...>] (origin.gro) (Optional)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

Options to specify output files:

**-o [<.edi>] (sam.edi)**
ED sampling input

Other options:

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-mon <string>**
Indices of eigenvectors for projections of x (e.g. 1,2-5,9) or 1-100:10 means 1 11 21 31 ... 91

**-linfix <string>**
Indices of eigenvectors for fixed increment linear sampling

**-linacc <string>**
Indices of eigenvectors for acceptance linear sampling

**-radfix <string>**
Indices of eigenvectors for fixed increment radius expansion

**-radacc <string>**
Indices of eigenvectors for acceptance radius expansion

**-radcon <string>**
Indices of eigenvectors for acceptance radius contraction

**-flood <string>**
Indices of eigenvectors for flooding

**-outfrq <int> (100)**
Frequency (in steps) of writing output in *.xvg* (page 589) file

**-slope <real> (0)**
Minimal slope in acceptance radius expansion

**-linstep <string>**
Stepsizes (nm/step) for fixed increment linear sampling (put in quotes! "1.0 2.3 5.1 -3.1")

**-accdir <string>**
Directions for acceptance linear sampling - only sign counts! (put in quotes! "-1 +1 -1.1")

**-radstep <real> (0)**
Stepsize (nm/step) for fixed increment radius expansion

**-maxedsteps <int> (0)**
Maximum number of steps per cycle

**-eqsteps <int> (0)**
Number of steps to run without any perturbations

**-deltaF0 <real> (150)**
Target destabilization energy for flooding

**-deltaF <real> (0)**
Start deltaF with this parameter - default 0, nonzero values only needed for restart

**-tau <real> (0.1)**
Coupling constant for adaption of flooding strength according to deltaF0, 0 = infinity i.e. constant flooding strength

**-Eflnull <real> (0)**
The starting value of the flooding strength. The flooding strength is updated according to the adaptive flooding scheme. For a constant flooding strength use `-tau` 0.

**-T <real> (300)**
T is temperature, the value is needed if you want to do flooding

**-alpha <real> (1)**
Scale width of gaussian flooding potential with alpha^2

**-[no]restrain (no)**
Use the flooding potential with inverted sign -> effects as quasiharmonic restraining potential

**-[no]hessian (no)**
The eigenvectors and eigenvalues are from a Hessian matrix

**-[no]harmonic (no)**
The eigenvalues are interpreted as spring constant

**-constF <string>**
Constant force flooding: manually set the forces for the eigenvectors selected with -flood (put in quotes! "1.0 2.3 5.1 -3.1"). No other flooding parameters are needed when specifying the forces directly.

### 3.11.52 gmx make_ndx

#### Synopsis

```
gmx make_ndx [-f [<.gro/.g96/...>]] [-n [<.ndx> [...]]] [-o [<.ndx>
↪]]
             [-natoms <int>] [-[no]twin]
```

### Description

Index groups are necessary for almost every GROMACS program. All these programs can generate default index groups. You ONLY have to use `gmx make_ndx` when you need SPECIAL index groups. There is a default index group for the whole system, 9 default index groups for proteins, and a default index group is generated for every other residue name.

When no index file is supplied, also `gmx make_ndx` will generate the default groups. With the index editor you can select on atom, residue and chain names and numbers. When a run input file is supplied you can also select on atom type. You can use boolean operations, you can split groups into chains, residues or atoms. You can delete and rename groups. Type 'h' in the editor for more details.

The atom numbering in the editor and the index file starts at 1.

The `-twin` switch duplicates all index groups with an offset of `-natoms`, which is useful for Computational Electrophysiology double-layer membrane setups.

See also *gmx select* (page 312) `-on`, which provides an alternative way for constructing index groups. It covers nearly all of `gmx make_ndx` functionality, and in many cases much more.

### Options

Options to specify input files:

**`-f` [<.gro/.g96/...>] (conf.gro) (Optional)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**`-n` [<.ndx> [...]] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-o` [<.ndx>] (index.ndx)**
Index file

Other options:

**`-natoms` <int> (0)**
set number of atoms (default: read from coordinate or index file)

**`-[no]twin` (no)**
Duplicate all index groups with an offset of -natoms

## 3.11.53 gmx mdmat

### Synopsis

```
gmx mdmat [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>
↪]]
          [-mean [<.xpm>]] [-frames [<.xpm>]] [-no [<.xvg>]]
          [-b <time>] [-e <time>] [-dt <time>] [-xvg <enum>]
          [-t <real>] [-nlevels <int>]
```

### Description

`gmx mdmat` makes distance matrices consisting of the smallest distance between residue pairs. With `-frames`, these distance matrices can be stored in order to see differences in tertiary structure as a function of time. If you choose your options unwisely, this may generate a large output file. By default, only an averaged matrix over the whole trajectory is output. Also a count of the number of different atomic contacts between residues over the whole trajectory can be made. The output can be processed with *gmx xpm2ps* (page 357) to make a PostScript (tm) plot.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-mean` [<.xpm>] (dm.xpm)**
X PixMap compatible matrix file

**`-frames` [<.xpm>] (dmf.xpm) (Optional)**
X PixMap compatible matrix file

**`-no` [<.xvg>] (num.xvg) (Optional)**
xvgr/xmgr file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-t` <real> (1.5)**
trunc distance

**`-nlevels` <int> (40)**
Discretize distance in this number of levels

### 3.11.54 gmx mdrun

#### Synopsis

```
gmx mdrun [-s [<.tpr>]] [-cpi [<.cpt>]] [-table [<.xvg>]]
          [-tablep [<.xvg>]] [-tableb [<.xvg> [...]]]
          [-rerun [<.xtc/.trr/...>]] [-ei [<.edi>]]
          [-multidir [<dir> [...]]] [-awh [<.xvg>]]
          [-plumed [<.dat>]] [-membed [<.dat>]] [-mp [<.top>]]
          [-mn [<.ndx>]] [-o [<.trr/.cpt/...>]] [-x [<.xtc/.tng>]]
          [-cpo [<.cpt>]] [-c [<.gro/.g96/...>]] [-e [<.edr>]]
          [-g [<.log>]] [-dhdl [<.xvg>]] [-field [<.xvg>]]
          [-tpi [<.xvg>]] [-tpid [<.xvg>]] [-eo [<.xvg>]]
          [-px [<.xvg>]] [-pf [<.xvg>]] [-ro [<.xvg>]] [-ra [<.log>
↪]]
          [-rs [<.log>]] [-rt [<.log>]] [-mtx [<.mtx>]]
          [-if [<.xvg>]] [-swap [<.xvg>]] [-deffnm <string>]
          [-xvg <enum>] [-dd <vector>] [-ddorder <enum>]
          [-npme <int>] [-nt <int>] [-ntmpi <int>] [-ntomp <int>]
          [-ntomp_pme <int>] [-pin <enum>] [-pinoffset <int>]
          [-pinstride <int>] [-gpu_id <string>] [-gputasks <string>]
          [-[no]ddcheck] [-rdd <real>] [-rcon <real>] [-dlb <enum>]
          [-dds <real>] [-nb <enum>] [-nbfe <enum>] [-nstlist <int>]
          [-[no]tunepme] [-pme <enum>] [-pmefft <enum>]
          [-bonded <enum>] [-update <enum>] [-[no]v] [-pforce <real>
↪]
          [-[no]reprod] [-cpt <real>] [-[no]cpnum] [-[no]append]
          [-nsteps <int>] [-maxh <real>] [-replex <int>] [-nex <int>
↪]
          [-reseed <int>]
```

#### Description

`gmx mdrun` is the main computational chemistry engine within GROMACS. Obviously, it performs Molecular Dynamics simulations, but it can also perform Stochastic Dynamics, Energy Minimization, test particle insertion or (re)calculation of energies. Normal mode analysis is another option. In this case `mdrun` builds a Hessian matrix from single conformation. For usual Normal Modes-like calculations, make sure that the structure provided is properly energy-minimized. The generated matrix can be diagonalized by *gmx nmeig* (page 269).

The `mdrun` program reads the run input file (`-s`) and distributes the topology over ranks if needed. `mdrun` produces at least four output files. A single log file (`-g`) is written. The trajectory file (`-o`), contains coordinates, velocities and optionally forces. The structure file (`-c`) contains the coordinates and velocities of the last step. The energy file (`-e`) contains energies, the temperature, pressure, etc, a lot of these things are also printed in the log file. Optionally coordinates can be written to a compressed trajectory file (`-x`).

The option `-dhdl` is only used when free energy calculation is turned on.

Running mdrun efficiently in parallel is a complex topic, many aspects of which are covered in the online User Guide. You should look there for practical advice on using many of the options available in mdrun.

ED (essential dynamics) sampling and/or additional flooding potentials are switched on by using the `-ei` flag followed by an *.edi* (page 576) file. The *.edi* (page 576) file can be produced with the `make_edi` tool or by using options in the essdyn menu of the WHAT IF program. `mdrun` produces a *.xvg* (page 589) output file that contains projections of positions, velocities and forces onto selected eigenvectors.

When user-defined potential functions have been selected in the *.mdp* (page 579) file the `-table` option is used to pass `mdrun` a formatted table with potential functions. The file is read from either the current directory or from the `GMXLIB` directory. A number of pre-formatted tables are presented in the `GMXLIB` dir, for 6-8, 6-9, 6-10, 6-11, 6-12 Lennard-Jones potentials with normal Coulomb. When pair interactions are present, a separate table for pair interaction functions is read using the `-tablep` option.

When tabulated bonded functions are present in the topology, interaction functions are read using the `-tableb` option. For each different tabulated interaction type used, a table file name must be given. For the topology to work, a file name given here must match a character sequence before the file extension. That sequence is: an underscore, then a 'b' for bonds, an 'a' for angles or a 'd' for dihedrals, and finally the matching table number index used in the topology. Note that, these options are deprecated, and in future will be available via grompp.

The options `-px` and `-pf` are used for writing pull COM coordinates and forces when pulling is selected in the *.mdp* (page 579) file.

The option `-membed` does what used to be g_membed, i.e. embed a protein into a membrane. This module requires a number of settings that are provided in a data file that is the argument of this option. For more details in membrane embedding, see the documentation in the user guide. The options `-mn` and `-mp` are used to provide the index and topology files used for the embedding.

The option `-pforce` is useful when you suspect a simulation crashes due to too large forces. With this option coordinates and forces of atoms with a force larger than a certain value will be printed to stderr. It will also terminate the run when non-finite forces are present.

Checkpoints containing the complete state of the system are written at regular intervals (option `-cpt`) to the file `-cpo`, unless option `-cpt` is set to -1. The previous checkpoint is backed up to `state_prev.cpt` to make sure that a recent state of the system is always available, even when the simulation is terminated while writing a checkpoint. With `-cpnum` all checkpoint files are kept and appended with the step number. A simulation can be continued by reading the full state from file with option `-cpi`. This option is intelligent in the way that if no checkpoint file is found, GROMACS just assumes a normal run and starts from the first step of the *.tpr* (page 586) file. By default the output will be appending to the existing output files. The checkpoint file contains checksums of all output files, such that you will never loose data when some output files are modified, corrupt or removed. There are three scenarios with `-cpi`:

* no files with matching names are present: new output files are written

* all files are present with names and checksums matching those stored in the checkpoint file: files are appended

* otherwise no files are modified and a fatal error is generated

With `-noappend` new output files are opened and the simulation part number is added to all output file names. Note that in all cases the checkpoint file itself is not renamed and will be overwritten, unless its name does not match the `-cpo` option.

With checkpointing the output is appended to previously written output files, unless `-noappend` is used or none of the previous output files are present (except for the checkpoint file). The integrity of the files to be appended is verified using checksums which are stored in the checkpoint file. This ensures that output can not be mixed up or corrupted due to file appending. When only some of the previous

output files are present, a fatal error is generated and no old output files are modified and no new output files are opened. The result with appending will be the same as from a single run. The contents will be binary identical, unless you use a different number of ranks or dynamic load balancing or the FFT library uses optimizations through timing.

With option `-maxh` a simulation is terminated and a checkpoint file is written at the first neighbor search step where the run time exceeds `-maxh`*0.99 hours. This option is particularly useful in combination with setting `nsteps` to -1 either in the mdp or using the similarly named command line option (although the latter is deprecated). This results in an infinite run, terminated only when the time limit set by `-maxh` is reached (if any) or upon receiving a signal.

Interactive molecular dynamics (IMD) can be activated by using at least one of the three IMD switches: The `-imdterm` switch allows one to terminate the simulation from the molecular viewer (e.g. VMD). With `-imdwait`, `mdrun` pauses whenever no IMD client is connected. Pulling from the IMD remote can be turned on by `-imdpull`. The port `mdrun` listens to can be altered by `-imdport`.The file pointed to by `-if` contains atom indices and forces if IMD pulling is used.

### Options

Options to specify input files:

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

**`-cpi` [<.cpt>] (state.cpt) (Optional)**
Checkpoint file

**`-table` [<.xvg>] (table.xvg) (Optional)**
xvgr/xmgr file

**`-tablep` [<.xvg>] (tablep.xvg) (Optional)**
xvgr/xmgr file

**`-tableb` [<.xvg> [...]] (table.xvg) (Optional)**
xvgr/xmgr file

**`-rerun` [<.xtc/.trr/...>] (rerun.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-ei` [<.edi>] (sam.edi) (Optional)**
ED sampling input

**`-multidir` [<dir> [...]] (rundir) (Optional)**
Run directory

**`-awh` [<.xvg>] (awhinit.xvg) (Optional)**
xvgr/xmgr file

**`-plumed` [<.dat>] (plumed.dat) (Optional)**
Generic data file

**`-membed` [<.dat>] (membed.dat) (Optional)**
Generic data file

**`-mp` [<.top>] (membed.top) (Optional)**
Topology file

**-mn [<.ndx>] (membed.ndx) (Optional)**
Index file

Options to specify output files:

**-o [<.trr/.cpt/...>] (traj.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**-x [<.xtc/.tng>] (traj_comp.xtc) (Optional)**
Compressed trajectory (tng format or portable xdr format)

**-cpo [<.cpt>] (state.cpt) (Optional)**
Checkpoint file

**-c [<.gro/.g96/...>] (confout.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

**-e [<.edr>] (ener.edr)**
Energy file

**-g [<.log>] (md.log)**
Log file

**-dhdl [<.xvg>] (dhdl.xvg) (Optional)**
xvgr/xmgr file

**-field [<.xvg>] (field.xvg) (Optional)**
xvgr/xmgr file

**-tpi [<.xvg>] (tpi.xvg) (Optional)**
xvgr/xmgr file

**-tpid [<.xvg>] (tpidist.xvg) (Optional)**
xvgr/xmgr file

**-eo [<.xvg>] (edsam.xvg) (Optional)**
xvgr/xmgr file

**-px [<.xvg>] (pullx.xvg) (Optional)**
xvgr/xmgr file

**-pf [<.xvg>] (pullf.xvg) (Optional)**
xvgr/xmgr file

**-ro [<.xvg>] (rotation.xvg) (Optional)**
xvgr/xmgr file

**-ra [<.log>] (rotangles.log) (Optional)**
Log file

**-rs [<.log>] (rotslabs.log) (Optional)**
Log file

**-rt [<.log>] (rottorque.log) (Optional)**
Log file

**-mtx [<.mtx>] (nm.mtx) (Optional)**
Hessian matrix

**-if [<.xvg>] (imdforces.xvg) (Optional)**
xvgr/xmgr file

**-swap [<.xvg>] (swapions.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-deffnm <string>**
Set the default filename for all file options

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-dd <vector> (0 0 0)**
Domain decomposition grid, 0 is optimize

**-ddorder <enum> (interleave)**
DD rank order: interleave, pp_pme, cartesian

**-npme <int> (-1)**
Number of separate ranks to be used for PME, -1 is guess

**-nt <int> (0)**
Total number of threads to start (0 is guess)

**-ntmpi <int> (0)**
Number of thread-MPI ranks to start (0 is guess)

**-ntomp <int> (0)**
Number of OpenMP threads per MPI rank to start (0 is guess)

**-ntomp_pme <int> (0)**
Number of OpenMP threads per MPI rank to start (0 is -ntomp)

**-pin <enum> (auto)**
Whether mdrun should try to set thread affinities: auto, on, inherit, off

**-pinoffset <int> (0)**
The lowest logical core number to which mdrun should pin the first thread

**-pinstride <int> (0)**
Pinning distance in logical cores for threads, use 0 to minimize the number of threads per physical core

**-gpu_id <string>**
List of unique GPU device IDs available to use

**-gputasks <string>**
List of GPU device IDs, mapping each task on a node to a device. Tasks include PP and PME (if present).

**-[no]ddcheck (yes)**
Check for all bonded interactions with DD

**-rdd <real> (0)**
The maximum distance for bonded interactions with DD (nm), 0 is determine from initial coordinates

**-rcon <real> (0)**
Maximum distance for P-LINCS (nm), 0 is estimate

**-dlb <enum> (auto)**
Dynamic load balancing (with DD): auto, no, yes

**-dds <real> (0.8)**
Fraction in (0,1) by whose reciprocal the initial DD cell size will be increased in order to provide a margin in which dynamic load balancing can act while preserving the minimum cell size.

**-nb <enum> (auto)**
Calculate non-bonded interactions on: auto, cpu, gpu

**-nbfe <enum> (auto)**
Perform non-bonded FE calculations on: auto, cpu, gpu

**-nstlist <int> (0)**
Set nstlist when using a Verlet buffer tolerance (0 is guess)

**-[no]tunepme (yes)**
Optimize PME load between PP/PME ranks or GPU/CPU

**-pme <enum> (auto)**
Perform PME calculations on: auto, cpu, gpu

**-pmefft <enum> (auto)**
Perform PME FFT calculations on: auto, cpu, gpu

**-bonded <enum> (auto)**
Perform bonded calculations on: auto, cpu, gpu

**-update <enum> (auto)**
Perform update and constraints on: auto, cpu, gpu

**-[no]v (no)**
Be loud and noisy

**-pforce <real> (-1)**
Print all forces larger than this (kJ/mol nm)

**-[no]reprod (no)**
Avoid optimizations that affect binary reproducibility; this can significantly reduce performance

**-cpt <real> (15)**
Checkpoint interval (minutes)

**-[no]cpnum (no)**
Keep and number checkpoint files

**-[no]append (yes)**
Append to previous output files when continuing from checkpoint instead of adding the simulation part number to all file names

**-nsteps <int> (-2)**
Run this number of steps (-1 means infinite, -2 means use mdp option, smaller is invalid)

**-maxh <real> (-1)**
Terminate after 0.99 times this time (hours)

**-replex <int> (0)**
Attempt replica exchange periodically with this period (steps)

**-nex <int> (0)**
Number of random exchanges to carry out each exchange interval (N^3 is one suggestion). -nex zero or not specified gives neighbor replica exchange.

**-reseed <int> (-1)**
Seed for replica exchange, -1 is generate a seed

### 3.11.55 gmx mindist

#### Synopsis

```
gmx mindist [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.
↪ndx>]]
            [-od [<.xvg>]] [-on [<.xvg>]] [-o [<.out>]]
            [-ox [<.xtc/.trr/...>]] [-or [<.xvg>]] [-b <time>]
            [-e <time>] [-dt <time>] [-tu <enum>] [-[no]w]
            [-xvg <enum>] [-[no]matrix] [-[no]max] [-d <real>]
            [-[no]group] [-[no]pi] [-[no]split] [-ng <int>]
            [-[no]pbc] [-[no]respertime] [-[no]printresname]
```

#### Description

`gmx mindist` computes the distance between one group and a number of other groups. Both the minimum distance (between any pair of atoms from the respective groups) and the number of contacts within a given distance are written to two separate output files. With the `-group` option a contact of an atom in another group with multiple atoms in the first group is counted as one contact instead of as multiple contacts. With `-or`, minimum distances to each residue in the first group are determined and plotted as a function of residue number.

With option `-pi` the minimum distance of a group to its periodic image is plotted. This is useful for checking if a protein has seen its periodic image during a simulation. Only one shift in each direction is considered, giving a total of 26 shifts. Note that periodicity information is required from the file supplied with with `-s`, either as a .tpr file or a .pdb file with CRYST1 fields. It also plots the maximum distance within the group and the lengths of the three box vectors.

Also *gmx distance* (page 198) and *gmx pairdist* (page 279) calculate distances.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**-od [<.xvg>] (mindist.xvg)**
xvgr/xmgr file

**-on [<.xvg>] (numcont.xvg) (Optional)**
xvgr/xmgr file

**-o [<.out>] (atm-pair.out) (Optional)**
Generic output file

**-ox [<.xtc/.trr/...>] (mindist.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-or [<.xvg>] (mindistres.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]matrix (no)**
Calculate half a matrix of group-group distances

**-[no]max (no)**
Calculate *maximum* distance instead of minimum

**-d <real> (0.6)**
Distance for contacts

**-[no]group (no)**
Count contacts with multiple atoms in the first group as one

**-[no]pi (no)**
Calculate minimum distance with periodic images

**-[no]split (no)**
Split graph where time is zero

**-ng <int> (1)**
Number of secondary groups to compute distance to a central group

**-[no]pbc (yes)**
Take periodic boundary conditions into account

**-[no]respertime (no)**
When writing per-residue distances, write distance for each time point

**-[no]printresname (no)**
Write residue names

### 3.11.56 gmx mk_angndx

#### Synopsis

```
gmx mk_angndx [-s [<.tpr>]] [-n [<.ndx>]] [-type <enum>] [-[no]hyd]
              [-hq <real>]
```

#### Description

`gmx mk_angndx` makes an index file for calculation of angle distributions etc. It uses a run input file (.tpx) for the definitions of the angles, dihedrals etc.

#### Options

Options to specify input files:

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

Options to specify output files:

**-n [<.ndx>] (angle.ndx)**
Index file

Other options:

**-type <enum> (angle)**
Type of angle: angle, dihedral, improper, ryckaert-bellemans

**-[no]hyd (yes)**
Include angles with atoms with mass < 1.5

**-hq <real> (-1)**
Ignore angles with atoms with mass < 1.5 and magnitude of their charge less than this value

### 3.11.57 gmx msd

#### Synopsis

```
gmx msd [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>]]
        [-o [<.xvg>]] [-mol [<.xvg>]] [-b <time>] [-e <time>]
        [-dt <time>] [-tu <enum>] [-fgroup <selection>] [-xvg <enum>
↪]
        [-[no]rmpbc] [-[no]pbc] [-sf <file>] [-selrpos <enum>]
        [-seltype <enum>] [-sel <selection>] [-type <enum>]
        [-lateral <enum>] [-trestart <real>] [-maxtau <real>]
        [-beginfit <real>] [-endfit <real>]
```

## Description

`gmx msd` computes the mean square displacement (MSD) of atoms from a set of initial positions. This provides an easy way to compute the diffusion constant using the Einstein relation. The time between the reference points for the MSD calculation is set with `-trestart`. The diffusion constant is calculated by least squares fitting a straight line (D*t + c) through the MSD(t) from `-beginfit` to `-endfit` (note that t is time from the reference positions, not simulation time). An error estimate given, which is the difference of the diffusion coefficients obtained from fits over the two halves of the fit interval.

There are three, mutually exclusive, options to determine different types of mean square displacement: `-type`, `-lateral` and `-ten`. Option `-ten` writes the full MSD tensor for each group, the order in the output is: trace xx yy zz yx zx zy.

If `-mol` is set, `gmx msd` plots the MSD for individual molecules (including making molecules whole across periodic boundaries): for each individual molecule a diffusion constant is computed for its center of mass. The chosen index group will be split into molecules. With -mol, only one index group can be selected.

The diffusion coefficient is determined by linear regression of the MSD. When `-beginfit` is -1, fitting starts at 10% and when `-endfit` is -1, fitting goes to 90%. Using this option one also gets an accurate error estimate based on the statistics between individual molecules. Note that this diffusion coefficient and error estimate are only accurate when the MSD is completely linear between `-beginfit` and `-endfit`.

By default, `gmx msd` compares all trajectory frames against every frame stored at `-trestart` intervals, so the number of frames stored scales linearly with the number of frames processed. This can lead to long analysis times and out-of-memory errors for long/large trajectories, and often the data at higher time deltas lacks sufficient sampling, often manifesting as a wobbly line on the MSD plot after a straighter region at lower time deltas. The `-maxtau` option can be used to cap the maximum time delta for frame comparison, which may improve performance and can be used to avoid out-of-memory issues.

## Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**`-o` [<.xvg>] (msdout.xvg) (Optional)**
MSD output

**`-mol` [<.xvg>] (diff_mol.xvg) (Optional)**
Report diffusion coefficients for each molecule in selection

Other options:

**`-b` <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-seltype <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-sel <selection>**
Selections to compute MSDs for from the reference

**-type <enum> (unused)**
: x, y, z, unused

**-lateral <enum> (unused)**
: x, y, z, unused

**-trestart <real> (10)**
Time between restarting points in trajectory (ps)

**-maxtau <real> (1.79769e+308)**
Maximum time delta between frames to calculate MSDs for (ps)

**-beginfit <real> (-1)**
Time point at which to start fitting.

**-endfit <real> (-1)**
End time for fitting.

### 3.11.58 gmx nmeig

#### Synopsis

```
gmx nmeig [-f [<.mtx>]] [-s [<.tpr>]] [-of [<.xvg>]] [-ol [<.xvg>]]
          [-os [<.xvg>]] [-qc [<.xvg>]] [-v [<.trr/.cpt/...>]]
          [-xvg <enum>] [-[no]m] [-first <int>] [-last <int>]
          [-maxspec <int>] [-T <real>] [-P <real>] [-sigma <int>]
          [-scale <real>] [-linear_toler <real>] [-[no]constr]
          [-width <real>]
```

#### Description

`gmx nmeig` calculates the eigenvectors/values of a (Hessian) matrix, which can be calculated with *gmx mdrun* (page 258). The eigenvectors are written to a trajectory file (`-v`). The structure is written first with t=0. The eigenvectors are written as frames with the eigenvector number and eigenvalue written as step number and timestamp, respectively. The eigenvectors can be analyzed with *gmx anaeig* (page 151). An ensemble of structures can be generated from the eigenvectors with *gmx nmens* (page 271). When mass weighting is used, the generated eigenvectors will be scaled back to plain Cartesian coordinates before generating the output. In this case, they will no longer be exactly orthogonal in the standard Cartesian norm, but in the mass-weighted norm they would be.

This program can be optionally used to compute quantum corrections to heat capacity and enthalpy by providing an extra file argument `-qcorr`. See the GROMACS manual, Chapter 1, for details. The result includes subtracting a harmonic degree of freedom at the given temperature. The total correction is printed on the terminal screen. The recommended way of getting the corrections out is:

```
gmx nmeig -s topol.tpr -f nm.mtx -first 7 -last 10000 -T 300 -qc
[-constr]
```

The `-constr` option should be used when bond constraints were used during the simulation **for all the covalent bonds**. If this is not the case, you need to analyze the `quant_corr.xvg` file yourself.

To make things more flexible, the program can also take virtual sites into account when computing quantum corrections. When selecting `-constr` and `-qc`, the `-begin` and `-end` options will be set automatically as well.

Based on a harmonic analysis of the normal mode frequencies, thermochemical properties S0 (Standard Entropy), Cv (Heat capacity at constant volume), Zero-point energy and the internal energy are computed, much in the same manner as popular quantum chemistry programs.

#### Options

Options to specify input files:

**`-f` [<.mtx>] (hessian.mtx)**
Hessian matrix

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

Options to specify output files:

**-of [<.xvg>] (eigenfreq.xvg)**
xvgr/xmgr file

**-ol [<.xvg>] (eigenval.xvg)**
xvgr/xmgr file

**-os [<.xvg>] (spectrum.xvg) (Optional)**
xvgr/xmgr file

**-qc [<.xvg>] (quant_corr.xvg) (Optional)**
xvgr/xmgr file

**-v [<.trr/.cpt/...>] (eigenvec.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

Other options:

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]m (yes)**
Divide elements of Hessian by product of sqrt(mass) of involved atoms prior to diagonalization. This should be used for 'Normal Modes' analysis

**-first <int> (1)**
First eigenvector to write away

**-last <int> (50)**
Last eigenvector to write away. -1 is use all dimensions.

**-maxspec <int> (4000)**
Highest frequency (1/cm) to consider in the spectrum

**-T <real> (298.15)**
Temperature for computing entropy, quantum heat capacity and enthalpy when using normal mode calculations to correct classical simulations

**-P <real> (1)**
Pressure (bar) when computing entropy

**-sigma <int> (1)**
Number of symmetric copies used when computing entropy. E.g. for water the number is 2, for NH3 it is 3 and for methane it is 12.

**-scale <real> (1)**
Factor to scale frequencies before computing thermochemistry values

**-linear_toler <real> (1e-05)**
Tolerance for determining whether a compound is linear as determined from the ration of the moments inertion Ix/Iy and Ix/Iz.

**-[no]constr (no)**
If constraints were used in the simulation but not in the normal mode analysis you will need to set this for computing the quantum corrections.

**-width <real> (1)**
Width (sigma) of the gaussian peaks (1/cm) when generating a spectrum

### 3.11.59 gmx nmens

#### Synopsis

```
gmx nmens [-v [<.trr/.cpt/...>]] [-e [<.xvg>]] [-s [<.tpr/.gro/...>
↪]]
          [-n [<.ndx>]] [-o [<.xtc/.trr/...>]] [-xvg <enum>]
          [-temp <real>] [-seed <int>] [-num <int>] [-first <int>]
          [-last <int>]
```

#### Description

`gmx nmens` generates an ensemble around an average structure in a subspace that is defined by a set of normal modes (eigenvectors). The eigenvectors are assumed to be mass-weighted. The position along each eigenvector is randomly taken from a Gaussian distribution with variance kT/eigenvalue.

By default the starting eigenvector is set to 7, since the first six normal modes are the translational and rotational degrees of freedom.

#### Options

Options to specify input files:

**-v [<.trr/.cpt/...>] (eigenvec.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**-e [<.xvg>] (eigenval.xvg)**
xvgr/xmgr file

**-s [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**-o [<.xtc/.trr/...>] (ensemble.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

Other options:

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-temp <real> (300)**
Temperature in Kelvin

**-seed <int> (0)**
Random seed (0 means generate)

**-num <int> (100)**
Number of structures to generate

**-first <int> (7)**
First eigenvector to use (-1 is select)

**-last <int> (-1)**
Last eigenvector to use (-1 is till the last)

### 3.11.60 gmx nmr

#### Synopsis

```
gmx nmr [-f [<.edr>]] [-f2 [<.edr>]] [-s [<.tpr>]] [-viol [<.xvg>]]
        [-pairs [<.xvg>]] [-ora [<.xvg>]] [-ort [<.xvg>]]
        [-oda [<.xvg>]] [-odr [<.xvg>]] [-odt [<.xvg>]]
        [-oten [<.xvg>]] [-b <time>] [-e <time>] [-[no]w]
        [-xvg <enum>] [-[no]dp] [-skip <int>] [-[no]aver]
        [-[no]orinst] [-[no]ovec]
```

#### Description

`gmx nmr` extracts distance or orientation restraint data from an energy file. The user is prompted to interactively select the desired terms.

When the `-viol` option is set, the time averaged violations are plotted and the running time-averaged and instantaneous sum of violations are recalculated. Additionally running time-averaged and instantaneous distances between selected pairs can be plotted with the `-pairs` option.

Options `-ora`, `-ort`, `-oda`, `-odr` and `-odt` are used for analyzing orientation restraint data. The first two options plot the orientation, the last three the deviations of the orientations from the experimental values. The options that end on an 'a' plot the average over time as a function of restraint. The options that end on a 't' prompt the user for restraint label numbers and plot the data as a function of time. Option `-odr` plots the RMS deviation as a function of restraint. When the run used time or ensemble averaged orientation restraints, option `-orinst` can be used to analyse the instantaneous, not ensemble-averaged orientations and deviations instead of the time and ensemble averages.

Option `-oten` plots the eigenvalues of the molecular order tensor for each orientation restraint experiment. With option `-ovec` also the eigenvectors are plotted.

#### Options

Options to specify input files:

**-f [<.edr>] (ener.edr)**
Energy file

**-f2 [<.edr>] (ener.edr) (Optional)**
Energy file

**-s [<.tpr>] (topol.tpr) (Optional)**
Portable xdr run input file

Options to specify output files:

**-viol [<.xvg>] (violaver.xvg) (Optional)**
xvgr/xmgr file

**-pairs [<.xvg>] (pairs.xvg) (Optional)**
xvgr/xmgr file

**-ora [<.xvg>] (orienta.xvg) (Optional)**
xvgr/xmgr file

**-ort [<.xvg>] (orientt.xvg) (Optional)**
xvgr/xmgr file

**-oda [<.xvg>] (orideva.xvg) (Optional)**
xvgr/xmgr file

**-odr [<.xvg>] (oridevr.xvg) (Optional)**
xvgr/xmgr file

**-odt [<.xvg>] (oridevt.xvg) (Optional)**
xvgr/xmgr file

**-oten [<.xvg>] (oriten.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]dp (no)**
Print energies in high precision

**-skip <int> (0)**
Skip number of frames between data points

**-[no]aver (no)**
Also print the exact average and rmsd stored in the energy frames (only when 1 term is requested)

**-[no]orinst (no)**
Analyse instantaneous orientation data

**-[no]ovec (no)**
Also plot the eigenvectors with `-oten`

### 3.11.61 gmx nmtraj

#### Synopsis

```
gmx nmtraj [-s [<.tpr/.gro/...>]] [-v [<.trr/.cpt/...>]]
           [-o [<.xtc/.trr/...>]] [-eignr <string>]
           [-phases <string>] [-temp <real>] [-amplitude <real>]
           [-nframes <int>]
```

#### Description

`gmx nmtraj` generates an virtual trajectory from an eigenvector, corresponding to a harmonic Cartesian oscillation around the average structure. The eigenvectors should normally be mass-weighted, but you can use non-weighted eigenvectors to generate orthogonal motions. The output frames are written as a trajectory file covering an entire period, and the first frame is the average structure. If you write the trajectory in (or convert to) PDB format you can view it directly in PyMol and also render a photorealistic movie. Motion amplitudes are calculated from the eigenvalues and a preset temperature, assuming equipartition of the energy over all modes. To make the motion clearly visible in PyMol you might want to amplify it by setting an unrealistically high temperature. However, be aware that both the linear Cartesian displacements and mass weighting will lead to serious structure deformation for high amplitudes - this is is simply a limitation of the Cartesian normal mode model. By default the selected eigenvector is set to 7, since the first six normal modes are the translational and rotational degrees of freedom.

#### Options

Options to specify input files:

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-v` [<.trr/.cpt/...>] (eigenvec.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

Options to specify output files:

**`-o` [<.xtc/.trr/...>] (nmtraj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

Other options:

**`-eignr` <string> (7)**
String of eigenvectors to use (first is 1)

**`-phases` <string> (0.0)**
String of phases (default is 0.0)

**`-temp` <real> (300)**
Temperature (K)

**`-amplitude` <real> (0.25)**
Amplitude for modes with eigenvalue<=0

**`-nframes` <int> (30)**
Number of frames to generate

### 3.11.62 gmx nonbonded-benchmark

#### Synopsis

```
gmx nonbonded-benchmark [-o [<.csv>]] [-size <int>] [-nt <int>]
             [-simd <enum>] [-coulomb <enum>]
             [-interactmodifier <enum>] [-[no]table]
             [-combrule <enum>] [-[no]halflj] [-[no]energy]
             [-[no]all] [-cutoff <real>] [-iter <int>]
             [-warmup <int>] [-[no]cycles] [-[no]time]
```

#### Description

`gmx nonbonded-benchmark` runs benchmarks for one or more so-called Nbnxm non-bonded pair kernels. The non-bonded pair kernels are the most compute intensive part of MD simulations and usually comprise 60 to 90 percent of the runtime. For this reason they are highly optimized and several different setups are available to compute the same physical interactions. In addition, there are different physical treatments of Coulomb interactions and optimizations for atoms without Lennard-Jones interactions. There are also different physical treatments of Lennard-Jones interactions, but only a plain cut-off is supported in this tool, as that is by far the most common treatment. And finally, while force output is always necessary, energy output is only required at certain steps. In total there are 36 relevant combinations of options. The combinations double to 72 when two different SIMD setups are supported. These combinations can be run with a single invocation using the `-all` option. The behavior of each kernel is affected by caching behavior, which is determined by the hardware used together with the system size and the cut-off radius. The larger the number of atoms per thread, the more L1 cache is needed to avoid L1 cache misses. The cut-off radius mainly affects the data reuse: a larger cut-off results in more data reuse and makes the kernel less sensitive to cache misses.

OpenMP parallelization is used to utilize multiple hardware threads within a compute node. In these benchmarks there is no interaction between threads, apart from starting and closing a single OpenMP parallel region per iteration. Additionally, threads interact through sharing and evicting data from shared caches. The number of threads to use is set with the `-nt` option. Thread affinity is important, especially with SMT and shared caches. Affinities can be set through the OpenMP library using the GOMP_CPU_-AFFINITY environment variable.

The benchmark tool times one or more kernels by running them repeatedly for a number of iterations set by the `-iter` option. An initial kernel call is done to avoid additional initial cache misses. Times are recording in cycles read from efficient, high accuracy counters in the CPU. Note that these often do not correspond to actual clock cycles. For each kernel, the tool reports the total number of cycles, cycles per iteration, and (total and useful) pair interactions per cycle. Because a cluster pair list is used instead of an atom pair list, interactions are also computed for some atom pairs that are beyond the cut-off distance. These pairs are not useful (except for additional buffering, but that is not of interest here), only a side effect of the cluster-pair setup. The SIMD 2xMM kernel has a higher useful pair ratio then the 4xM kernel due to a smaller cluster size, but a lower total pair throughput. It is best to run this, or for that matter any, benchmark with locked CPU clocks, as thermal throttling can significantly affect performance. If that is not an option, the `-warmup` option can be used to run initial, untimed iterations to warm up the processor.

The most relevant regime is between 0.1 to 1 millisecond per iteration. Thus it is useful to run with system sizes that cover both ends of this regime.

The `-simd` and `-table` options select different implementations to compute the same physics. The choice of these options should ideally be optimized for the target hardware. Historically, we only found

tabulated Ewald correction to be useful on 2-wide SIMD or 4-wide SIMD without FMA support. As all modern architectures are wider and support FMA, we do not use tables by default. The only exceptions are kernels without SIMD, which only support tables. Options `-coulomb`, `-combrule`, `-interactmodifier` and `-halflj` depend on the force field and composition of the simulated system. The optimization of computing Lennard-Jones interactions for only half of the atoms in a cluster is useful for water, which does not use Lennard-Jones on hydrogen atoms in most water models. In the MD engine, any clusters where at most half of the atoms have LJ interactions will automatically use this kernel. And finally, the `-energy` option selects the computation of energies, which are usually only needed infrequently.

### Options

Options to specify output files:

**`-o` [<.csv>] (nonbonded-benchmark.csv) (Optional)**
Also output results in csv format

Other options:

**`-size` <int> (1)**
The system size is 3000 atoms times this value

**`-nt` <int> (1)**
The number of OpenMP threads to use

**`-simd` <enum> (auto)**
SIMD type, auto runs all supported SIMD setups or no SIMD when SIMD is not supported: auto, no, 4xm, 2xmm

**`-coulomb` <enum> (ewald)**
The functional form for the Coulomb interactions: ewald, reaction-field

**`-interactmodifier` <enum> (PotShift)**
The Coulomb / VdW interaction modifier. Reported under the 'intmod.' column: PotShift, PotSwitch, ForceSwitch

**`-[no]table` (no)**
Use lookup table for Ewald correction instead of analytical

**`-combrule` <enum> (geometric)**
The LJ combination rule. Reported under the 'comb.' column: geometric, lb, none

**`-[no]halflj` (no)**
Use optimization for LJ on half of the atoms. Reported under the 'LJ' column

**`-[no]energy` (no)**
Compute energies in addition to forces

**`-[no]all` (no)**
Run all 36 combinations of options for coulomb, halflj, combrule, interactmodifier

**`-cutoff` <real> (1)**
Pair-list and interaction cut-off distance

**`-iter` <int> (100)**
The number of iterations for each kernel

**-warmup <int> (0)**
The number of iterations for initial warmup

**-[no]cycles (no)**
Report cycles/pair instead of pairs/cycle

**-[no]time (no)**
Report micro-seconds instead of cycles

### 3.11.63 gmx order

#### Synopsis

```
gmx order [-f [<.xtc/.trr/...>]] [-n [<.ndx>]] [-nr [<.ndx>]]
          [-s [<.tpr>]] [-o [<.xvg>]] [-od [<.xvg>]] [-ob [<.pdb>]]
          [-os [<.xvg>]] [-Sg [<.xvg>]] [-Sk [<.xvg>]]
          [-Sgsl [<.xvg>]] [-Sksl [<.xvg>]] [-b <time>] [-e <time>]
          [-dt <time>] [-[no]w] [-xvg <enum>] [-d <enum>] [-sl <int>
↪]
          [-[no]szonly] [-[no]permolecule] [-[no]radial]
          [-[no]calcdist]
```

#### Description

`gmx order` computes the order parameter per atom for carbon tails. For atom i the vector i-1, i+1 is used together with an axis. The index file should contain only the groups to be used for calculations, with each group of equivalent carbons along the relevant acyl chain in its own group. There should not be any generic groups (like System, Protein) in the index file to avoid confusing the program (this is not relevant to tetrahedral order parameters however, which only work for water anyway).

`gmx order` can also give all diagonal elements of the order tensor and even calculate the deuterium order parameter Scd (default). If the option `-szonly` is given, only one order tensor component (specified by the `-d` option) is given and the order parameter per slice is calculated as well. If `-szonly` is not selected, all diagonal elements and the deuterium order parameter is given.

The tetrahedrality order parameters can be determined around an atom. Both angle an distance order parameters are calculated. See P.-L. Chau and A.J. Hardwick, Mol. Phys., 93, (1998), 511-518. for more details.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-n [<.ndx>] (index.ndx)**
Index file

**-nr [<.ndx>] (index.ndx) (Optional)**
Index file

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

Options to specify output files:

**-o [<.xvg>] (order.xvg)**
xvgr/xmgr file

**-od [<.xvg>] (deuter.xvg)**
xvgr/xmgr file

**-ob [<.pdb>] (eiwit.pdb) (Optional)**
Protein data bank file

**-os [<.xvg>] (sliced.xvg)**
xvgr/xmgr file

**-Sg [<.xvg>] (sg-ang.xvg) (Optional)**
xvgr/xmgr file

**-Sk [<.xvg>] (sk-dist.xvg) (Optional)**
xvgr/xmgr file

**-Sgsl [<.xvg>] (sg-ang-slice.xvg) (Optional)**
xvgr/xmgr file

**-Sksl [<.xvg>] (sk-dist-slice.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-d <enum> (z)**
Direction of the normal on the membrane: z, x, y

**-sl <int> (1)**
Calculate order parameter as function of box length, dividing the box into this number of slices.

**-[no]szonly (no)**
Only give Sz element of order tensor. (axis can be specified with `-d`)

**-[no]permolecule (no)**
Compute per-molecule Scd order parameters

**-[no]radial (no)**
Compute a radial membrane normal

**-[no]calcdist** (no)
Compute distance from a reference

### Known Issues

- This tool only works for saturated carbons and united atom force fields. For anything else, it is highly recommended to use a different analysis method!
- The option `-unsat` claimed to do analysis for unsaturated carbons but this hasn't worked ever since it was added and has thus been removed.

## 3.11.64 gmx pairdist

### Synopsis

```
gmx pairdist [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.
↪ndx>]]
             [-o [<.xvg>]] [-b <time>] [-e <time>] [-dt <time>]
             [-tu <enum>] [-fgroup <selection>] [-xvg <enum>]
             [-[no]rmpbc] [-[no]pbc] [-sf <file>] [-selrpos <enum>]
             [-seltype <enum>] [-cutoff <real>] [-type <enum>]
             [-refgrouping <enum>] [-selgrouping <enum>]
             [-ref <selection>] [-sel <selection>]
```

### Description

`gmx pairdist` calculates pairwise distances between one reference selection (given with `-ref`) and one or more other selections (given with `-sel`). It can calculate either the minimum distance (the default), or the maximum distance (with `-type max`). Distances to each selection provided with `-sel` are computed independently.

By default, the global minimum/maximum distance is computed. To compute more distances (e.g., minimum distances to each residue in `-ref`), use `-refgrouping` and/or `-selgrouping` to specify how the positions within each selection should be grouped.

Computed distances are written to the file specified with `-o`. If there are N groups in `-ref` and M groups in the first selection in `-sel`, then the output contains N*M columns for the first selection. The columns contain distances like this: r1-s1, r2-s1, ..., r1-s2, r2-s2, ..., where rn is the n'th group in `-ref` and sn is the n'th group in the other selection. The distances for the second selection comes as separate columns after the first selection, and so on. If some selections are dynamic, only the selected positions are used in the computation but the same number of columns is always written out. If there are no positions contributing to some group pair, then the cutoff value is written (see below).

`-cutoff` sets a cutoff for the computed distances. If the result would contain a distance over the cutoff, the cutoff value is written to the output file instead. By default, no cutoff is used, but if you are not interested in values beyond a cutoff, or if you know that the minimum distance is smaller than a cutoff, you should set this option to allow the tool to use grid-based searching and be significantly faster.

If you want to compute distances between fixed pairs, *gmx distance* (page 198) may be a more suitable tool.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**`-o` [<.xvg>] (dist.xvg)**
Distances as function of time

Other options:

**`-b` <time> (0)**
First frame (ps) to read from trajectory

**`-e` <time> (0)**
Last frame (ps) to read from trajectory

**`-dt` <time> (0)**
Only use frame if t MOD dt == first time (ps)

**`-tu` <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**`-fgroup` <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**`-xvg` <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**`-[no]rmpbc` (yes)**
Make molecules whole for each frame

**`-[no]pbc` (yes)**
Use periodic boundary conditions for distance calculation

**`-sf` <file>**
Provide selections from files

**`-selrpos` <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**`-seltype` <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**`-cutoff` <real> (0)**
Maximum distance to consider

**-type <enum> (min)**
Type of distances to calculate: min, max

**-refgrouping <enum> (all)**
Grouping of -ref positions to compute the min/max over: all, res, mol, none

**-selgrouping <enum> (all)**
Grouping of -sel positions to compute the min/max over: all, res, mol, none

**-ref <selection>**
Reference positions to calculate distances from

**-sel <selection>**
Positions to calculate distances for

### 3.11.65 gmx pdb2gmx

#### Synopsis

```
gmx pdb2gmx [-f [<.gro/.g96/...>]] [-o [<.gro/.g96/...>]] [-p [<.
↪top>]]
            [-i [<.itp>]] [-n [<.ndx>]] [-q [<.gro/.g96/...>]]
            [-chainsep <enum>] [-merge <enum>] [-ff <string>]
            [-water <enum>] [-[no]inter] [-[no]ss] [-[no]ter]
            [-[no]lys] [-[no]arg] [-[no]asp] [-[no]glu] [-[no]gln]
            [-[no]his] [-angle <real>] [-dist <real>] [-[no]una]
            [-[no]ignh] [-[no]missing] [-[no]v] [-posrefc <real>]
            [-vsite <enum>] [-[no]heavyh] [-[no]deuterate]
            [-[no]cmap] [-[no]renum] [-rtpres <enum>]
```

#### Description

`gmx pdb2gmx` reads a *.pdb* (page 581) (or *.gro* (page 577)) file, reads some database files, adds hydrogens to the molecules and generates coordinates in GROMACS (GROMOS), or optionally *.pdb* (page 581), format and a topology in GROMACS format. These files can subsequently be processed to generate a run input file.

`gmx pdb2gmx` will search for force fields by looking for a `forcefield.itp` file in subdirectories `<forcefield>.ff` of the current working directory and of the GROMACS library directory as inferred from the path of the binary or the `GMXLIB` environment variable. By default the forcefield selection is interactive, but you can use the `-ff` option to specify one of the short names in the list on the command line instead. In that case `gmx pdb2gmx` just looks for the corresponding `<forcefield>.ff` directory.

After choosing a force field, all files will be read only from the corresponding force field directory. If you want to modify or add a residue types, you can copy the force field directory from the GROMACS library directory to your current working directory. If you want to add new protein residue types, you will need to modify `residuetypes.dat` in the library directory or copy the whole library directory to a local directory and set the environment variable `GMXLIB` to the name of that directory. Check Chapter 5 of the manual for more information about file formats.

Note that a *.pdb* (page 581) file is nothing more than a file format, and it need not necessarily contain a protein structure. Every kind of molecule for which there is support in the database can be converted. If

there is no support in the database, you can add it yourself.

The program has limited intelligence, it reads a number of database files, that allow it to make special bonds (Cys-Cys, Heme-His, etc.), if necessary this can be done manually. The program can prompt the user to select which kind of LYS, ASP, GLU, CYS or HIS residue is desired. For Lys the choice is between neutral (two protons on NZ) or protonated (three protons, default), for Asp and Glu unprotonated (default) or protonated, for His the proton can be either on ND1, on NE2 or on both. By default these selections are done automatically. For His, this is based on an optimal hydrogen bonding conformation. Hydrogen bonds are defined based on a simple geometric criterion, specified by the maximum hydrogen-donor-acceptor angle and donor-acceptor distance, which are set by `-angle` and `-dist` respectively.

The protonation state of N- and C-termini can be chosen interactively with the `-ter` flag. Default termini are ionized (NH3+ and COO-), respectively. Some force fields support zwitterionic forms for chains of one residue, but for polypeptides these options should NOT be selected. The AMBER force fields have unique forms for the terminal residues, and these are incompatible with the `-ter` mechanism. You need to prefix your N- or C-terminal residue names with "N" or "C" respectively to use these forms, making sure you preserve the format of the coordinate file. Alternatively, use named terminating residues (e.g. ACE, NME).

The separation of chains is not entirely trivial since the markup in user-generated PDB files frequently varies and sometimes it is desirable to merge entries across a TER record, for instance if you want a disulfide bridge or distance restraints between two protein chains or if you have a HEME group bound to a protein. In such cases multiple chains should be contained in a single `moleculetype` definition. To handle this, `gmx pdb2gmx` uses two separate options. First, `-chainsep` allows you to choose when a new chemical chain should start, and termini added when applicable. This can be done based on the existence of TER records, when the chain id changes, or combinations of either or both of these. You can also do the selection fully interactively. In addition, there is a `-merge` option that controls how multiple chains are merged into one moleculetype, after adding all the chemical termini (or not). This can be turned off (no merging), all non-water chains can be merged into a single molecule, or the selection can be done interactively.

`gmx pdb2gmx` will also check the occupancy field of the *.pdb* (page 581) file. If any of the occupancies are not one, indicating that the atom is not resolved well in the structure, a warning message is issued. When a *.pdb* (page 581) file does not originate from an X-ray structure determination all occupancy fields may be zero. Either way, it is up to the user to verify the correctness of the input data (read the article!).

During processing the atoms will be reordered according to GROMACS conventions. With `-n` an index file can be generated that contains one group reordered in the same way. This allows you to convert a GROMOS trajectory and coordinate file to GROMOS. There is one limitation: reordering is done after the hydrogens are stripped from the input and before new hydrogens are added. This means that you should not use `-ignh`.

The *.gro* (page 577) and `.g96` file formats do not support chain identifiers. Therefore it is useful to enter a *.pdb* (page 581) file name at the `-o` option when you want to convert a multi-chain *.pdb* (page 581) file.

The option `-vsite` removes hydrogen and fast improper dihedral motions. Angular and out-of-plane motions can be removed by changing hydrogens into virtual sites and fixing angles, which fixes their position relative to neighboring atoms. Additionally, all atoms in the aromatic rings of the standard amino acids (i.e. PHE, TRP, TYR and HIS) can be converted into virtual sites, eliminating the fast improper dihedral fluctuations in these rings (but this feature is deprecated). **Note** that in this case all other hydrogen atoms are also converted to virtual sites. The mass of all atoms that are converted into virtual sites, is added to the heavy atoms.

Also slowing down of dihedral motion can be done with `-heavyh` done by increasing the hydrogen-mass by a factor of 4. This is also done for water hydrogens to slow down the rotational motion of water. The increase in mass of the hydrogens is subtracted from the bonded (heavy) atom so that the total mass of the system remains the same. As a special case, ring-closed (or cyclic) molecules are considered. `gmx pdb2gmx` automatically determines if a cyclic molecule is present by evaluating the distance between the terminal atoms of a given chain. If this distance is greater than the `-sb` ("Short bond warning distance", default 0.05 nm) and less than the `-lb` ("Long bond warning distance", default 0.25 nm) the molecule is considered to be ring closed and will be processed as such. Please note that this does not detect cyclic bonds over periodic boundaries.

### Options

Options to specify input files:

**`-f` [<.gro/.g96/...>] (protein.pdb)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

Options to specify output files:

**`-o` [<.gro/.g96/...>] (conf.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

**`-p` [<.top>] (topol.top)**
Topology file

**`-i` [<.itp>] (posre.itp)**
Include file for topology

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

**`-q` [<.gro/.g96/...>] (clean.pdb) (Optional)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

Other options:

**`-chainsep` <enum> (id_or_ter)**
Condition in PDB files when a new chain should be started (adding termini): id_or_ter, id_and_ter, ter, id, interactive

**`-merge` <enum> (no)**
Merge multiple chains into a single [moleculetype]: no, all, interactive

**`-ff` <string> (select)**
Force field, interactive by default. Use `-h` for information.

**`-water` <enum> (select)**
Water model to use: select, none, opc, opc3, spc, spce, tip3p, tip4p, tip4pew, tip5p, tips3p

**`-[no]inter` (no)**
Set the next 8 options to interactive

**`-[no]ss` (no)**
Interactive SS bridge selection

**`-[no]ter` (no)**
Interactive termini selection, instead of charged (default)

**-[no]lys (no)**
Interactive lysine selection, instead of charged

**-[no]arg (no)**
Interactive arginine selection, instead of charged

**-[no]asp (no)**
Interactive aspartic acid selection, instead of charged

**-[no]glu (no)**
Interactive glutamic acid selection, instead of charged

**-[no]gln (no)**
Interactive glutamine selection, instead of charged

**-[no]his (no)**
Interactive histidine selection, instead of checking H-bonds

**-angle <real> (135)**
Minimum hydrogen-donor-acceptor angle for a H-bond (degrees)

**-dist <real> (0.3)**
Maximum donor-acceptor distance for a H-bond (nm)

**-[no]una (no)**
Select aromatic rings with united CH atoms on phenylalanine, tryptophane and tyrosine

**-[no]ignh (no)**
Ignore hydrogen atoms that are in the coordinate file

**-[no]missing (no)**
Continue when atoms are missing and bonds cannot be made, dangerous

**-[no]v (no)**
Be slightly more verbose in messages

**-posrefc <real> (1000)**
Force constant for position restraints

**-vsite <enum> (none)**
Convert atoms to virtual sites: none, hydrogens, aromatics

**-[no]heavyh (no)**
Make hydrogen atoms heavy

**-[no]deuterate (no)**
Change the mass of hydrogens to 2 amu

**-[no]cmap (yes)**
Use cmap torsions (if enabled in the *.rtp* (page 582) file)

**-[no]renum (no)**
Renumber the residues consecutively in the output

**-rtpres <enum> (auto)**
Use *.rtp* (page 582) entry names as residue names: auto, no, yes

### 3.11.66 gmx pme_error

#### Synopsis

```
gmx pme_error [-s [<.tpr>]] [-o [<.out>]] [-so [<.tpr>]] [-beta
↪<real>]
              [-[no]tune] [-self <real>] [-seed <int>] [-[no]v]
```

#### Description

`gmx pme_error` estimates the error of the electrostatic forces if using the sPME algorithm. The flag `-tune` will determine the splitting parameter such that the error is equally distributed over the real and reciprocal space part. The part of the error that stems from self interaction of the particles is computationally demanding. However, a good a approximation is to just use a fraction of the particles for this term which can be indicated by the flag `-self`.

#### Options

Options to specify input files:

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

Options to specify output files:

**`-o` [<.out>] (error.out)**
Generic output file

**`-so` [<.tpr>] (tuned.tpr) (Optional)**
Portable xdr run input file

Other options:

**`-beta` <real> (-1)**
If positive, overwrite ewald_beta from *.tpr* (page 586) file with this value

**`-[no]tune` (no)**
Tune the splitting parameter such that the error is equally distributed between real and reciprocal space

**`-self` <real> (1)**
If between 0.0 and 1.0, determine self interaction error from just this fraction of the charged particles

**`-seed` <int> (0)**
Random number seed used for Monte Carlo algorithm when `-self` is set to a value between 0.0 and 1.0

**`-[no]v` (no)**
Be loud and noisy

### 3.11.67 gmx polystat

#### Synopsis

```
gmx polystat [-s [<.tpr>]] [-f [<.xtc/.trr/...>]] [-n [<.ndx>]]
             [-o [<.xvg>]] [-v [<.xvg>]] [-p [<.xvg>]] [-i [<.xvg>]]
             [-b <time>] [-e <time>] [-dt <time>] [-tu <enum>]
             [-[no]w] [-xvg <enum>] [-[no]mw] [-[no]pc]
```

#### Description

`gmx polystat` plots static properties of polymers as a function of time and prints the average.

By default it determines the average end-to-end distance and radii of gyration of polymers. It asks for an index group and split this into molecules. The end-to-end distance is then determined using the first and the last atom in the index group for each molecules. For the radius of gyration the total and the three principal components for the average gyration tensor are written. With option `-v` the eigenvectors are written. With option `-pc` also the average eigenvalues of the individual gyration tensors are written. With option `-i` the mean square internal distances are written.

With option `-p` the persistence length is determined. The chosen index group should consist of atoms that are consecutively bonded in the polymer mainchains. The persistence length is then determined from the cosine of the angles between bonds with an index difference that is even, the odd pairs are not used, because straight polymer backbones are usually all trans and therefore only every second bond aligns. The persistence length is defined as number of bonds where the average cos reaches a value of 1/e. This point is determined by a linear interpolation of log(<cos>).

#### Options

Options to specify input files:

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-o` [<.xvg>] (polystat.xvg)**
xvgr/xmgr file

**`-v` [<.xvg>] (polyvec.xvg) (Optional)**
xvgr/xmgr file

**`-p` [<.xvg>] (persist.xvg) (Optional)**
xvgr/xmgr file

**`-i` [<.xvg>] (intdist.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]mw (yes)**
Use the mass weighting for radii of gyration

**-[no]pc (no)**
Plot average eigenvalues

### 3.11.68 gmx potential

#### Synopsis

```
gmx potential [-f [<.xtc/.trr/...>]] [-n [<.ndx>]] [-s [<.tpr>]]
              [-o [<.xvg>]] [-oc [<.xvg>]] [-of [<.xvg>]] [-b <time>]
              [-e <time>] [-dt <time>] [-[no]w] [-xvg <enum>]
              [-d <string>] [-sl <int>] [-cb <int>] [-ce <int>]
              [-tz <real>] [-[no]spherical] [-ng <int>] [-[no]center]
              [-[no]symm] [-[no]correct]
```

#### Description

`gmx potential` computes the electrostatical potential across the box. The potential is calculated by first summing the charges per slice and then integrating twice of this charge distribution. Periodic boundaries are not taken into account. Reference of potential is taken to be the left side of the box. It is also possible to calculate the potential in spherical coordinates as function of r by calculating a charge distribution in spherical slices and twice integrating them. epsilon_r is taken as 1, but 2 is more appropriate in many cases.

Option `-center` performs the histogram binning and potential calculation relative to the center of an arbitrary group, in absolute box coordinates. If you are calculating profiles along the Z axis box dimension bZ, output would be from -bZ/2 to bZ/2 if you center based on the entire system. Option `-symm` symmetrizes the output around the center. This will automatically turn on `-center` too.

### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-n [<.ndx>] (index.ndx)**
Index file

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

Options to specify output files:

**-o [<.xvg>] (potential.xvg)**
xvgr/xmgr file

**-oc [<.xvg>] (charge.xvg)**
xvgr/xmgr file

**-of [<.xvg>] (field.xvg)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-d <string> (Z)**
Take the normal on the membrane in direction X, Y or Z.

**-sl <int> (10)**
Calculate potential as function of boxlength, dividing the box in this number of slices.

**-cb <int> (0)**
Discard this number of first slices of box for integration

**-ce <int> (0)**
Discard this number of last slices of box for integration

**-tz <real> (0)**
Translate all coordinates by this distance in the direction of the box

**-[no]spherical (no)**
Calculate in spherical coordinates

**-ng <int> (1)**
Number of groups to consider

**-[no]center (no)**
Perform the binning relative to the center of the (changing) box. Useful for bilayers.

**-[no]symm (no)**
Symmetrize the density along the axis, with respect to the center. Useful for bilayers.

**-[no]correct (no)**
Assume net zero charge of groups to improve accuracy

### Known Issues

- Discarding slices for integration should not be necessary.

## 3.11.69 gmx principal

### Synopsis

```
gmx principal [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]]
              [-n [<.ndx>]] [-a1 [<.xvg>]] [-a2 [<.xvg>]]
              [-a3 [<.xvg>]] [-om [<.xvg>]] [-b <time>] [-e <time>]
              [-dt <time>] [-tu <enum>] [-[no]w] [-xvg <enum>]
              [-[no]foo]
```

### Description

`gmx principal` calculates the three principal axes of inertia for a group of atoms. NOTE: Old versions of GROMACS wrote the output data in a strange transposed way. As of GROMACS 5.0, the output file paxis1.dat contains the x/y/z components of the first (major) principal axis for each frame, and similarly for the middle and minor axes in paxis2.dat and paxis3.dat.

### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**-a1 [<.xvg>] (paxis1.xvg)**
xvgr/xmgr file

**-a2 [<.xvg>] (paxis2.xvg)**
xvgr/xmgr file

**-a3 [<.xvg>] (paxis3.xvg)**
xvgr/xmgr file

**-om [<.xvg>] (moi.xvg)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]foo (no)**
Dummy option to avoid empty array

### 3.11.70 gmx rama

#### Synopsis

```
gmx rama [-f [<.xtc/.trr/...>]] [-s [<.tpr>]] [-o [<.xvg>]] [-b
↪<time>]
         [-e <time>] [-dt <time>] [-[no]w] [-xvg <enum>]
```

#### Description

`gmx rama` selects the phi/psi dihedral combinations from your topology file and computes these as a function of time. Using simple Unix tools such as *grep* you can select out specific residues.

### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

Options to specify output files:

**-o [<.xvg>] (rama.xvg)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

## 3.11.71 gmx rdf

### Synopsis

```
gmx rdf [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>]]
        [-o [<.xvg>]] [-cn [<.xvg>]] [-b <time>] [-e <time>]
        [-dt <time>] [-tu <enum>] [-fgroup <selection>] [-xvg <enum>
↪]
        [-[no]rmpbc] [-[no]pbc] [-sf <file>] [-selrpos <enum>]
        [-seltype <enum>] [-bin <real>] [-norm <enum>] [-[no]xy]
        [-[no]excl] [-cut <real>] [-rmax <real>] [-surf <enum>]
        [-ref <selection>] [-sel <selection>]
```

## Description

`gmx rdf` calculates radial distribution functions from one reference set of position (set with `-ref`) to one or more sets of positions (set with `-sel`). To compute the RDF with respect to the closest position in a set in `-ref` instead, use `-surf`: if set, then `-ref` is partitioned into sets based on the value of `-surf`, and the closest position in each set is used. To compute the RDF around axes parallel to the *z*-axis, i.e., only in the *x*-*y* plane, use `-xy`.

To set the bin width and maximum distance to use in the RDF, use `-bin` and `-rmax`, respectively. The latter can be used to limit the computational cost if the RDF is not of interest up to the default (half of the box size with PBC, three times the box size without PBC).

To use exclusions from the topology (`-s`), set `-excl` and ensure that both `-ref` and `-sel` only select atoms. A rougher alternative to exclude intra-molecular peaks is to set `-cut` to a non-zero value to clear the RDF at small distances.

The RDFs are normalized by 1) average number of positions in `-ref` (the number of groups with `-surf`), 2) volume of the bin, and 3) average particle density of `-sel` positions for that selection. To change the normalization, use `-norm`:

- `rdf`: Use all factors for normalization. This produces a normal RDF.
- `number_density`: Use the first two factors. This produces a number density as a function of distance.
- `none`: Use only the first factor. In this case, the RDF is only scaled with the bin width to make the integral of the curve represent the number of pairs within a range.

Note that exclusions do not affect the normalization: even if `-excl` is set, or `-ref` and `-sel` contain the same selection, the normalization factor is still N*M, not N*(M-excluded).

For `-surf`, the selection provided to `-ref` must select atoms, i.e., centers of mass are not supported. Further, `-nonorm` is implied, as the bins have irregular shapes and the volume of a bin is not easily computable.

Option `-cn` produces the cumulative number RDF, i.e. the average number of particles within a distance r.

## Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**`-o` [<.xvg>] (rdf.xvg)**
Computed RDFs

**-cn [<.xvg>] (rdf_cn.xvg) (Optional)**
Cumulative RDFs

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-seltype <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-bin <real> (0.002)**
Bin width (nm)

**-norm <enum> (rdf)**
Normalization: rdf, number_density, none

**-[no]xy (no)**
Use only the x and y components of the distance

**-[no]excl (no)**
Use exclusions from topology

**-cut <real> (0)**
Shortest distance (nm) to be considered

**-rmax <real> (0)**
Largest distance (nm) to calculate

**-surf <enum> (no)**
RDF with respect to the surface of the reference: no, mol, res

**-ref <selection>**
Reference selection for RDF computation

**-sel <selection>**
Selections to compute RDFs for from the reference

### 3.11.72 gmx report-methods

#### Synopsis

```
gmx report-methods [-s [<.tpr/.gro/...>]] [-m [<.tex>]] [-o [<.out>
↪]]
```

#### Description

`gmx report-methods` reports basic system information for the run input file specified with `-s` either to the terminal, to a LaTeX formatted output file if run with the `-m` option or to an unformatted file with the `-o` option. The functionality has been moved here from its previous place in *gmx check* (page 165).

#### Options

Options to specify input files:

**-s [<.tpr/.gro/...>] (topol.tpr)**
Run input file for report: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

Options to specify output files:

**-m [<.tex>] (report.tex) (Optional)**
LaTeX formatted report output

**-o [<.out>] (report.out) (Optional)**
Unformatted report output to file

### 3.11.73 gmx rms

#### Synopsis

```
gmx rms [-s [<.tpr/.gro/...>]] [-f [<.xtc/.trr/...>]]
        [-f2 [<.xtc/.trr/...>]] [-n [<.ndx>]] [-o [<.xvg>]]
        [-mir [<.xvg>]] [-a [<.xvg>]] [-dist [<.xvg>]] [-m [<.xpm>]]
        [-bin [<.dat>]] [-bm [<.xpm>]] [-b <time>] [-e <time>]
        [-dt <time>] [-tu <enum>] [-[no]w] [-xvg <enum>]
        [-what <enum>] [-[no]pbc] [-fit <enum>] [-prev <int>]
        [-[no]split] [-skip <int>] [-skip2 <int>] [-max <real>]
        [-min <real>] [-bmax <real>] [-bmin <real>] [-[no]mw]
        [-nlevels <int>] [-ng <int>]
```

## Description

`gmx rms` compares two structures by computing the root mean square deviation (RMSD), the size-independent rho similarity parameter (`rho`) or the scaled rho (`rhosc`), see Maiorov & Crippen, Proteins **22**, 273 (1995). This is selected by `-what`.

Each structure from a trajectory (`-f`) is compared to a reference structure. The reference structure is taken from the structure file (`-s`).

With option `-mir` also a comparison with the mirror image of the reference structure is calculated. This is useful as a reference for 'significant' values, see Maiorov & Crippen, Proteins **22**, 273 (1995).

Option `-prev` produces the comparison with a previous frame the specified number of frames ago.

Option `-m` produces a matrix in *.xpm* (page 587) format of comparison values of each structure in the trajectory with respect to each other structure. This file can be visualized with for instance `xv` and can be converted to postscript with *gmx xpm2ps* (page 357).

Option `-fit` controls the least-squares fitting of the structures on top of each other: complete fit (rotation and translation), translation only, or no fitting at all.

Option `-mw` controls whether mass weighting is done or not. If you select the option (default) and supply a valid *.tpr* (page 586) file masses will be taken from there, otherwise the masses will be deduced from the `atommass.dat` file in `GMXLIB` (deprecated). This is fine for proteins, but not necessarily for other molecules. You can check whether this happened by turning on the `-debug` flag and inspecting the log file.

With `-f2`, the 'other structures' are taken from a second trajectory, this generates a comparison matrix of one trajectory versus the other.

Option `-bin` does a binary dump of the comparison matrix.

Option `-bm` produces a matrix of average bond angle deviations analogously to the `-m` option. Only bonds between atoms in the comparison group are considered.

## Options

Options to specify input files:

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-f2` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-o` [<.xvg>] (rmsd.xvg)**
xvgr/xmgr file

**-mir [<.xvg>] (rmsdmir.xvg) (Optional)**
xvgr/xmgr file

**-a [<.xvg>] (avgrp.xvg) (Optional)**
xvgr/xmgr file

**-dist [<.xvg>] (rmsd-dist.xvg) (Optional)**
xvgr/xmgr file

**-m [<.xpm>] (rmsd.xpm) (Optional)**
X PixMap compatible matrix file

**-bin [<.dat>] (rmsd.dat) (Optional)**
Generic data file

**-bm [<.xpm>] (bond.xpm) (Optional)**
X PixMap compatible matrix file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-what <enum> (rmsd)**
Structural difference measure: rmsd, rho, rhosc

**-[no]pbc (yes)**
PBC check

**-fit <enum> (rot+trans)**
Fit to reference structure: rot+trans, translation, none

**-prev <int> (0)**
Compare with previous frame

**-[no]split (no)**
Split graph where time is zero

**-skip <int> (1)**
Only write every nr-th frame to matrix

**-skip2 <int> (1)**
Only write every nr-th frame to matrix

**-max <real> (-1)**
Maximum level in comparison matrix

**-min <real> (-1)**
Minimum level in comparison matrix

**-bmax <real> (-1)**
Maximum level in bond angle matrix

**-bmin <real> (-1)**
Minimum level in bond angle matrix

**-[no]mw (yes)**
Use mass weighting for superposition

**-nlevels <int> (80)**
Number of levels in the matrices

**-ng <int> (1)**
Number of groups to compute RMS between

### 3.11.74 gmx rmsdist

#### Synopsis

```
gmx rmsdist [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.
↪ndx>]]
            [-equiv [<.dat>]] [-o [<.xvg>]] [-rms [<.xpm>]]
            [-scl [<.xpm>]] [-mean [<.xpm>]] [-nmr3 [<.xpm>]]
            [-nmr6 [<.xpm>]] [-noe [<.dat>]] [-b <time>] [-e <time>]
            [-dt <time>] [-[no]w] [-xvg <enum>] [-nlevels <int>]
            [-max <real>] [-[no]sumh] [-[no]pbc]
```

#### Description

`gmx rmsdist` computes the root mean square deviation of atom distances, which has the advantage that no fit is needed like in standard RMS deviation as computed by *gmx rms* (page 294). The reference structure is taken from the structure file. The RMSD at time t is calculated as the RMS of the differences in distance between atom-pairs in the reference structure and the structure at time t.

`gmx rmsdist` can also produce matrices of the rms distances, rms distances scaled with the mean distance and the mean distances and matrices with NMR averaged distances (1/r^3 and 1/r^6 averaging). Finally, lists of atom pairs with 1/r^3 and 1/r^6 averaged distance below the maximum distance (`-max`, which will default to 0.6 in this case) can be generated, by default averaging over equivalent hydrogens (all triplets of hydrogens named *[123]). Additionally a list of equivalent atoms can be supplied (`-equiv`), each line containing a set of equivalent atoms specified as residue number and name and atom name; e.g.:

```
HB* 3 SER HB1 3 SER HB2
```

Residue and atom names must exactly match those in the structure file, including case. Specifying non-sequential atoms is undefined.

### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

**-equiv [<.dat>] (equiv.dat) (Optional)**
Generic data file

Options to specify output files:

**-o [<.xvg>] (distrmsd.xvg)**
xvgr/xmgr file

**-rms [<.xpm>] (rmsdist.xpm) (Optional)**
X PixMap compatible matrix file

**-scl [<.xpm>] (rmsscale.xpm) (Optional)**
X PixMap compatible matrix file

**-mean [<.xpm>] (rmsmean.xpm) (Optional)**
X PixMap compatible matrix file

**-nmr3 [<.xpm>] (nmr3.xpm) (Optional)**
X PixMap compatible matrix file

**-nmr6 [<.xpm>] (nmr6.xpm) (Optional)**
X PixMap compatible matrix file

**-noe [<.dat>] (noe.dat) (Optional)**
Generic data file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-nlevels <int> (40)**
Discretize RMS in this number of levels

**-max <real> (-1)**
Maximum level in matrices

**-[no]sumh (yes)**
Average distance over equivalent hydrogens

**-[no]pbc (yes)**
Use periodic boundary conditions when computing distances

### 3.11.75 gmx rmsf

#### Synopsis

```
gmx rmsf [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>]]
         [-q [<.pdb>]] [-oq [<.pdb>]] [-ox [<.pdb>]] [-o [<.xvg>]]
         [-od [<.xvg>]] [-oc [<.xvg>]] [-dir [<.log>]] [-b <time>]
         [-e <time>] [-dt <time>] [-[no]w] [-xvg <enum>] [-[no]res]
         [-[no]aniso] [-[no]fit]
```

#### Description

`gmx rmsf` computes the root mean square fluctuation (RMSF, i.e. standard deviation) of atomic positions in the trajectory (supplied with `-f`) after (optionally) fitting to a reference frame (supplied with `-s`).

With option `-oq` the RMSF values are converted to B-factor values, which are written to a *.pdb* (page 581) file. By default, the coordinates in this output file are taken from the structure file provided with `-s`,although you can also use coordinates read from a different *.pdb* (page 581) fileprovided with `-q`. There is very little error checking, so in this caseit is your responsibility to make sure all atoms in the structure fileand *.pdb* (page 581) file correspond exactly to each other.

Option `-ox` writes the B-factors to a file with the average coordinates in the trajectory.

With the option `-od` the root mean square deviation with respect to the reference structure is calculated.

With the option `-aniso`, `gmx rmsf` will compute anisotropic temperature factors and then it will also output average coordinates and a *.pdb* (page 581) file with ANISOU records (corresponding to the `-oq` or `-ox` option). Please note that the U values are orientation-dependent, so before comparison with experimental data you should verify that you fit to the experimental coordinates.

When a *.pdb* (page 581) input file is passed to the program and the `-aniso` flag is set a correlation plot of the Uij will be created, if any anisotropic temperature factors are present in the *.pdb* (page 581) file.

With option `-dir` the average MSF (3x3) matrix is diagonalized. This shows the directions in which the atoms fluctuate the most and the least.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

**`-q` [<.pdb>] (eiwit.pdb) (Optional)**
Protein data bank file

Options to specify output files:

**`-oq` [<.pdb>] (bfac.pdb) (Optional)**
Protein data bank file

**`-ox` [<.pdb>] (xaver.pdb) (Optional)**
Protein data bank file

**`-o` [<.xvg>] (rmsf.xvg)**
xvgr/xmgr file

**`-od` [<.xvg>] (rmsdev.xvg) (Optional)**
xvgr/xmgr file

**`-oc` [<.xvg>] (correl.xvg) (Optional)**
xvgr/xmgr file

**`-dir` [<.log>] (rmsf.log) (Optional)**
Log file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-[no]res` (no)**
Calculate averages for each residue

**`-[no]aniso` (no)**
Compute anisotropic temperature factors

**-[no]fit (yes)**
Do a least squares superposition before computing RMSF. Without this you must make sure that the reference structure and the trajectory match.

### 3.11.76 gmx rotacf

#### Synopsis

```
gmx rotacf [-f [<.xtc/.trr/...>]] [-s [<.tpr>]] [-n [<.ndx>]]
           [-o [<.xvg>]] [-b <time>] [-e <time>] [-dt <time>]
           [-[no]w] [-xvg <enum>] [-[no]d] [-[no]aver]
           [-acflen <int>] [-[no]normalize] [-P <enum>]
           [-fitfn <enum>] [-beginfit <real>] [-endfit <real>]
```

#### Description

`gmx rotacf` calculates the rotational correlation function for molecules. Atom triplets (i,j,k) must be given in the index file, defining two vectors ij and jk. The rotational ACF is calculated as the autocorrelation function of the vector n = ij x jk, i.e. the cross product of the two vectors. Since three atoms span a plane, the order of the three atoms does not matter. Optionally, by invoking the `-d` switch, you can calculate the rotational correlation function for linear molecules by specifying atom pairs (i,j) in the index file.

EXAMPLES

```
gmx rotacf -P 1 -nparm 2 -fft -n index -o rotacf-x-P1 -fa
expfit-x-P1 -beginfit 2.5 -endfit 20.0
```

This will calculate the rotational correlation function using a first order Legendre polynomial of the angle of a vector defined by the index file. The correlation function will be fitted from 2.5 ps until 20.0 ps to a two-parameter exponential.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

**-n [<.ndx>] (index.ndx)**
Index file

Options to specify output files:

**-o [<.xvg>] (rotacf.xvg)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]d (no)**
Use index doublets (vectors) for correlation function instead of triplets (planes)

**-[no]aver (yes)**
Average over molecules

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### 3.11.77 gmx rotmat

#### Synopsis

```
gmx rotmat [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>
↪]]
           [-o [<.xvg>]] [-b <time>] [-e <time>] [-dt <time>]
           [-[no]w] [-xvg <enum>] [-ref <enum>] [-skip <int>]
           [-[no]fitxy] [-[no]mw]
```

### Description

`gmx rotmat` plots the rotation matrix required for least squares fitting a conformation onto the reference conformation provided with `-s`. Translation is removed before fitting. The output are the three vectors that give the new directions of the x, y and z directions of the reference conformation, for example: (zx,zy,zz) is the orientation of the reference z-axis in the trajectory frame.

This tool is useful for, for instance, determining the orientation of a molecule at an interface, possibly on a trajectory produced with `gmx trjconv -fit rotxy+transxy` to remove the rotation in the *x*-*y* plane.

Option `-ref` determines a reference structure for fitting, instead of using the structure from `-s`. The structure with the lowest sum of RMSD's to all other structures is used. Since the computational cost of this procedure grows with the square of the number of frames, the `-skip` option can be useful. A full fit or only a fit in the *x*-*y* plane can be performed.

Option `-fitxy` fits in the *x*-*y* plane before determining the rotation matrix.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-o` [<.xvg>] (rotmat.xvg)**
xvgr/xmgr file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-ref` <enum> (none)**
Determine the optimal reference structure: none, xyz, xy

**`-skip` <int> (1)**
Use every nr-th frame for `-ref`

**-[no]fitxy (no)**
Fit the x/y rotation before determining the rotation

**-[no]mw (yes)**
Use mass weighted fitting

### 3.11.78 gmx saltbr

#### Synopsis

```
gmx saltbr [-f [<.xtc/.trr/...>]] [-s [<.tpr>]] [-b <time>] [-e
↪<time>]
           [-dt <time>] [-t <real>] [-[no]sep]
```

#### Description

`gmx saltbr` plots the distance between all combination of charged groups as a function of time. The groups are combined in different ways. A minimum distance can be given (i.e. a cut-off), such that groups that are never closer than that distance will not be plotted.

Output will be in a number of fixed filenames, `min-min.xvg`, `plus-min.xvg` and `plus-plus.xvg`, or files for every individual ion pair if the `-sep` option is selected. In this case, files are named as `sb-(Resname)(Resnr)-(Atomnr)`. There may be **many** such files.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr>] (topol.tpr)**
Portable xdr run input file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-t <real> (1000)**
Groups that are never closer than this distance are not plotted

**-[no]sep (no)**
Use separate files for each interaction (may be MANY)

### 3.11.79 gmx sans-legacy

#### Synopsis

```
gmx sans-legacy [-s [<.tpr>]] [-f [<.xtc/.trr/...>]] [-n [<.ndx>]]
                [-d [<.dat>]] [-pr [<.xvg>]] [-sq [<.xvg>]]
                [-prframe [<.xvg>]] [-sqframe [<.xvg>]] [-b <time>]
                [-e <time>] [-dt <time>] [-tu <enum>] [-xvg <enum>]
                [-mode <enum>] [-mcover <real>] [-[no]pbc]
                [-startq <real>] [-endq <real>] [-qstep <real>]
                [-seed <int>] [-nt <int>]
```

#### Description

`gmx sans-legacy` computes SANS spectra using Debye formula. It currently uses topology file (since it need to assign element for each atom).

Parameters:

`-pr` Computes normalized g(r) function averaged over trajectory

`-prframe` Computes normalized g(r) function for each frame

`-sq` Computes SANS intensity curve averaged over trajectory

`-sqframe` Computes SANS intensity curve for each frame

`-startq` Starting q value in nm

`-endq` Ending q value in nm

`-qstep` Stepping in q space

Note: When using Debye direct method computational cost increases as 1/2 * N * (N - 1) where N is atom number in group of interest.

WARNING: If sq or pr specified this tool can produce large number of files! Up to two times larger than number of frames!

#### Options

Options to specify input files:

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

**`-d` [<.dat>] (nsfactor.dat) (Optional)**
Generic data file

Options to specify output files:

**-pr [<.xvg>] (pr.xvg)**
xvgr/xmgr file

**-sq [<.xvg>] (sq.xvg)**
xvgr/xmgr file

**-prframe [<.xvg>] (prframe.xvg) (Optional)**
xvgr/xmgr file

**-sqframe [<.xvg>] (sqframe.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-mode <enum> (direct)**
Mode for sans spectra calculation: direct, mc

**-mcover <real> (-1)**
Monte-Carlo coverage should be -1(default) or (0,1]

**-[no]pbc (yes)**
Use periodic boundary conditions for computing distances

**-startq <real> (0)**
Starting q (1/nm)

**-endq <real> (2)**
Ending q (1/nm)

**-qstep <real> (0.01)**
Stepping in q (1/nm)

**-seed <int> (0)**
Random seed for Monte-Carlo

**-nt <int> (96)**
Number of threads to start

### 3.11.80 gmx sasa

#### Synopsis

```
gmx sasa [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>]]
         [-o [<.xvg>]] [-odg [<.xvg>]] [-or [<.xvg>]] [-oa [<.xvg>]]
         [-tv [<.xvg>]] [-q [<.pdb>]] [-b <time>] [-e <time>]
         [-dt <time>] [-tu <enum>] [-fgroup <selection>]
         [-xvg <enum>] [-[no]rmpbc] [-[no]pbc] [-sf <file>]
         [-selrpos <enum>] [-probe <real>] [-ndots <int>]
↪[-[no]prot]
         [-dgs <real>] [-surface <selection>] [-output <selection>]
```

#### Description

`gmx sasa` computes solvent accessible surface areas. See Eisenhaber F, Lijnzaad P, Argos P, Sander C, & Scharf M (1995) J. Comput. Chem. 16, 273-284 for the algorithm used. With `-q`, the Connolly surface can be generated as well in a *.pdb* (page 581) file where the nodes are represented as atoms and the edges connecting the nearest nodes as CONECT records. `-odg` allows for estimation of solvation free energies from per-atom solvation energies per exposed surface area.

The program requires a selection for the surface calculation to be specified with `-surface`. This should always consist of all non-solvent atoms in the system. The area of this group is always calculated. Optionally, `-output` can specify additional selections, which should be subsets of the calculation group. The solvent-accessible areas for these groups are also extracted from the full surface.

The average and standard deviation of the area over the trajectory can be calculated per residue and atom (options `-or` and `-oa`).

With the `-tv` option the total volume and density of the molecule can be computed. With `-pbc` (the default), you must ensure that your molecule/surface group is not split across PBC. Otherwise, you will get non-sensical results. Please also consider whether the normal probe radius is appropriate in this case or whether you would rather use, e.g., 0. It is good to keep in mind that the results for volume and density are very approximate. For example, in ice Ih, one can easily fit water molecules in the pores which would yield a volume that is too low, and surface area and density that are both too high.

#### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**`-o` [<.xvg>] (area.xvg)**
Total area as a function of time

**-odg [<.xvg>] (dgsolv.xvg) (Optional)**
Estimated solvation free energy as a function of time

**-or [<.xvg>] (resarea.xvg) (Optional)**
Average area per residue

**-oa [<.xvg>] (atomarea.xvg) (Optional)**
Average area per atom

**-tv [<.xvg>] (volume.xvg) (Optional)**
Total volume and density as a function of time

**-q [<.pdb>] (connolly.pdb) (Optional)**
PDB file for Connolly surface

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-probe <real> (0.14)**
Radius of the solvent probe (nm)

**-ndots <int> (24)**
Number of dots per sphere, more dots means more accuracy

**-[no]prot (yes)**
Output the protein to the Connolly *.pdb* (page 581) file too

**-dgs <real> (0)**
Default value for solvation free energy per area (kJ/mol/nm^2)

**-surface <selection>**
Surface calculation selection

**-output <selection>**
Output selection(s)

### 3.11.81 gmx saxs-legacy

#### Synopsis

```
gmx saxs-legacy [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]]
                [-n [<.ndx>]] [-d [<.dat>]] [-sq [<.xvg>]] [-b <time>]
                [-e <time>] [-dt <time>] [-xvg <enum>] [-ng <int>]
                [-startq <real>] [-endq <real>] [-energy <real>]
```

#### Description

`gmx saxs-legacy` calculates SAXS structure factors for given index groups based on Cromer's method. Both topology and trajectory files are required.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

**-d [<.dat>] (sfactor.dat) (Optional)**
Generic data file

Options to specify output files:

**-sq [<.xvg>] (sq.xvg)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-ng <int> (1)**
Number of groups to compute SAXS

**-startq <real> (0)**
Starting q (1/nm)

**-endq <real> (60)**
Ending q (1/nm)

**-energy <real> (12)**
Energy of the incoming X-ray (keV)

### 3.11.82 gmx scattering

#### Synopsis

```
gmx scattering [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]]
               [-n [<.ndx>]] [-o [<.xvg>]] [-b <time>] [-e <time>]
               [-dt <time>] [-tu <enum>] [-fgroup <selection>]
               [-xvg <enum>] [-[no]rmpbc] [-[no]pbc] [-sf <file>]
               [-selrpos <enum>] [-seltype <enum>] [-sel <selection>]
               [-startq <real>] [-endq <real>] [-qspacing <real>]
               [-binwidth <real>] [-mc-coverage <real>] [-seed <int>]
               [-[no]norm] [-[no]mc] [-scattering-type <enum>]
```

#### Description

`gmx scattering` calculates SANS and SAXS scattering curves using Debye method.

The scattering intensity, I(q), as a function of scattering angle q with averaging over frames.

Note that this is a new implementation of the SANS/SAXS utilities added in GROMACS 2024. If you need the old ones, use `gmx sans-legacy` or `gmx saxs-legacy`.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**-o [<.xvg>] (scattering.xvg) (Optional)**
scattering intensity as a function of q

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-seltype <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-sel <selection>**
Selection for Scattering calculation

**-startq <real> (0)**
smallest q value (1/nm)

**-endq <real> (2)**
largest q value (1/nm)

**-qspacing <real> (0.01)**
spacing of q values (1/nm)

**-binwidth <real> (0.1)**
Bin width (nm) for P(r)

**-mc-coverage <real> (0.2)**
coverage of Monte Carlo (%)

**-seed <int> (2023)**
random seed for Monte Carlo

**-[no]norm (no)**
normalize scattering intensities

**-[no]mc (yes)**
use Monte Carlo to scattering intensities

**-scattering-type <enum> (sans)**
Scattering type: saxs, sans

### 3.11.83 gmx select

#### Synopsis

```
gmx select [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>
↪]]
           [-os [<.xvg>]] [-oc [<.xvg>]] [-oi [<.dat>]]
           [-on [<.ndx>]] [-om [<.xvg>]] [-of [<.xvg>]]
           [-ofpdb [<.pdb>]] [-olt [<.xvg>]] [-b <time>] [-e <time>]
           [-dt <time>] [-tu <enum>] [-fgroup <selection>]
           [-xvg <enum>] [-[no]rmpbc] [-[no]pbc] [-sf <file>]
           [-selrpos <enum>] [-seltype <enum>] [-select <selection>]
           [-[no]norm] [-[no]cfnorm] [-resnr <enum>]
           [-pdbatoms <enum>] [-[no]cumlt]
```

#### Description

`gmx select` writes out basic data about dynamic selections. It can be used for some simple analyses, or the output can be combined with output from other programs and/or external analysis programs to calculate more complex things. For detailed help on the selection syntax, please use `gmx help selections`.

Any combination of the output options is possible, but note that `-om` only operates on the first selection. Also note that if you provide no output options, no output is produced.

With `-os`, calculates the number of positions in each selection for each frame. With `-norm`, the output is between 0 and 1 and describes the fraction from the maximum number of positions (e.g., for selection 'resname RA and x < 5' the maximum number of positions is the number of atoms in RA residues). With `-cfnorm`, the output is divided by the fraction covered by the selection. `-norm` and `-cfnorm` can be specified independently of one another.

With `-oc`, the fraction covered by each selection is written out as a function of time.

With `-oi`, the selected atoms/residues/molecules are written out as a function of time. In the output, the first column contains the frame time, the second contains the number of positions, followed by the atom/residue/molecule numbers. If more than one selection is specified, the size of the second group immediately follows the last number of the first group and so on.

With `-on`, the selected atoms are written as a index file compatible with `make_ndx` and the analyzing tools. Each selection is written as a selection group and for dynamic selections a group is written for each frame.

For residue numbers, the output of `-oi` can be controlled with `-resnr`: `number` (default) prints the residue numbers as they appear in the input file, while `index` prints unique numbers assigned to the residues in the order they appear in the input file, starting with 1. The former is more intuitive, but if the input contains multiple residues with the same number, the output can be less useful.

With `-om`, a mask is printed for the first selection as a function of time. Each line in the output corresponds to one frame, and contains either 0/1 for each atom/residue/molecule possibly selected. 1 stands for the atom/residue/molecule being selected for the current frame, 0 for not selected.

With `-of`, the occupancy fraction of each position (i.e., the fraction of frames where the position is selected) is printed.

With `-ofpdb`, a PDB file is written out where the occupancy column is filled with the occupancy fraction of each atom in the selection. The coordinates in the PDB file will be those from the input topology. `-pdbatoms` can be used to control which atoms appear in the output PDB file: with `all` all atoms are present, with `maxsel` all atoms possibly selected by the selection are present, and with `selected` only atoms that are selected at least in one frame are present.

With `-olt`, a histogram is produced that shows the number of selected positions as a function of the time the position was continuously selected. `-cumlt` can be used to control whether subintervals of longer intervals are included in the histogram.

`-om`, `-of`, and `-olt` only make sense with dynamic selections.

To plot coordinates for selections, use *gmx trajectory* (page 331).

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**`-os` [<.xvg>] (size.xvg) (Optional)**
Number of positions in each selection

**`-oc` [<.xvg>] (cfrac.xvg) (Optional)**
Covered fraction for each selection

**`-oi` [<.dat>] (index.dat) (Optional)**
Indices selected by each selection

**`-on` [<.ndx>] (index.ndx) (Optional)**
Index file from the selection

**`-om` [<.xvg>] (mask.xvg) (Optional)**
Mask for selected positions

**`-of` [<.xvg>] (occupancy.xvg) (Optional)**
Occupied fraction for selected positions

**`-ofpdb` [<.pdb>] (occupancy.pdb) (Optional)**
PDB file with occupied fraction for selected positions

**-olt [<.xvg>] (lifetime.xvg) (Optional)**
Lifetime histogram

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-seltype <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-select <selection>**
Selections to analyze

**-[no]norm (no)**
Normalize by total number of positions with -os

**-[no]cfnorm (no)**
Normalize by covered fraction with -os

**-resnr <enum> (number)**
Residue number output type with -oi and -on: number, index

**-pdbatoms <enum> (all)**
Atoms to write with -ofpdb: all, maxsel, selected

**-[no]cumlt (yes)**
Cumulate subintervals of longer intervals in -olt

### 3.11.84 gmx sham

#### Synopsis

```
gmx sham [-f [<.xvg>]] [-ge [<.xvg>]] [-ene [<.xvg>]] [-dist [<.xvg>
↪]]
         [-histo [<.xvg>]] [-bin [<.ndx>]] [-lp [<.xpm>]]
         [-ls [<.xpm>]] [-lsh [<.xpm>]] [-lss [<.xpm>]]
         [-ls3 [<.pdb>]] [-g [<.log>]] [-[no]w] [-xvg <enum>]
         [-[no]time] [-b <real>] [-e <real>] [-ttol <real>]
         [-n <int>] [-[no]d] [-[no]sham] [-tsham <real>]
         [-pmin <real>] [-dim <vector>] [-ngrid <vector>]
         [-xmin <vector>] [-xmax <vector>] [-pmax <real>]
         [-gmax <real>] [-emin <real>] [-emax <real>]
         [-nlevels <int>]
```

#### Description

`gmx sham` makes multi-dimensional free-energy, enthalpy and entropy plots. `gmx sham` reads one or more *.xvg* (page 589) files and analyzes data sets. The basic purpose of `gmx sham` is to plot Gibbs free energy landscapes (option `-ls`) by Bolzmann inverting multi-dimensional histograms (option `-lp`), but it can also make enthalpy (option `-lsh`) and entropy (option `-lss`) plots. The histograms can be made for any quantities the user supplies. A line in the input file may start with a time (see option `-time`) and any number of *y*-values may follow. Multiple sets can also be read when they are separated by & (option `-n`), in this case only one *y*-value is read from each line. All lines starting with # and @ are skipped.

Option `-ge` can be used to supply a file with free energies when the ensemble is not a Boltzmann ensemble, but needs to be biased by this free energy. One free energy value is required for each (multi-dimensional) data point in the `-f` input.

Option `-ene` can be used to supply a file with energies. These energies are used as a weighting function in the single histogram analysis method by Kumar et al. When temperatures are supplied (as a second column in the file), an experimental weighting scheme is applied. In addition the vales are used for making enthalpy and entropy plots.

With option `-dim`, dimensions can be gives for distances. When a distance is 2- or 3-dimensional, the circumference or surface sampled by two particles increases with increasing distance. Depending on what one would like to show, one can choose to correct the histogram and free-energy for this volume effect. The probability is normalized by r and r^2 for dimensions of 2 and 3, respectively. A value of -1 is used to indicate an angle in degrees between two vectors: a sin(angle) normalization will be applied. **Note** that for angles between vectors the inner-product or cosine is the natural quantity to use, as it will produce bins of the same volume.

### Options

Options to specify input files:

**`-f` [<.xvg>] (graph.xvg)**
xvgr/xmgr file

**`-ge` [<.xvg>] (gibbs.xvg) (Optional)**
xvgr/xmgr file

**`-ene` [<.xvg>] (esham.xvg) (Optional)**
xvgr/xmgr file

Options to specify output files:

**`-dist` [<.xvg>] (ener.xvg) (Optional)**
xvgr/xmgr file

**`-histo` [<.xvg>] (edist.xvg) (Optional)**
xvgr/xmgr file

**`-bin` [<.ndx>] (bindex.ndx) (Optional)**
Index file

**`-lp` [<.xpm>] (prob.xpm) (Optional)**
X PixMap compatible matrix file

**`-ls` [<.xpm>] (gibbs.xpm) (Optional)**
X PixMap compatible matrix file

**`-lsh` [<.xpm>] (enthalpy.xpm) (Optional)**
X PixMap compatible matrix file

**`-lss` [<.xpm>] (entropy.xpm) (Optional)**
X PixMap compatible matrix file

**`-ls3` [<.pdb>] (gibbs3.pdb) (Optional)**
Protein data bank file

**`-g` [<.log>] (shamlog.log) (Optional)**
Log file

Other options:

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-[no]time` (yes)**
Expect a time in the input

**`-b` <real> (-1)**
First time to read from set

**`-e` <real> (-1)**
Last time to read from set

**`-ttol` <real> (0)**
Tolerance on time in appropriate units (usually ps)

**-n <int> (1)**
Read this number of sets separated by lines containing only an ampersand

**-[no]d (no)**
Use the derivative

**-[no]sham (yes)**
Turn off energy weighting even if energies are given

**-tsham <real> (298.15)**
Temperature for single histogram analysis

**-pmin <real> (0)**
Minimum probability. Anything lower than this will be set to zero

**-dim <vector> (1 1 1)**
Dimensions for distances, used for volume correction (max 3 values, dimensions > 3 will get the same value as the last)

**-ngrid <vector> (32 32 32)**
Number of bins for energy landscapes (max 3 values, dimensions > 3 will get the same value as the last)

**-xmin <vector> (0 0 0)**
Minimum for the axes in energy landscape (see above for > 3 dimensions)

**-xmax <vector> (1 1 1)**
Maximum for the axes in energy landscape (see above for > 3 dimensions)

**-pmax <real> (0)**
Maximum probability in output, default is calculate

**-gmax <real> (0)**
Maximum free energy in output, default is calculate

**-emin <real> (0)**
Minimum enthalpy in output, default is calculate

**-emax <real> (0)**
Maximum enthalpy in output, default is calculate

**-nlevels <int> (25)**
Number of levels for energy landscape

### 3.11.85 gmx sigeps

#### Synopsis

```
gmx sigeps [-o [<.xvg>]] [-[no]w] [-xvg <enum>] [-c6 <real>]
           [-cn <real>] [-pow <int>] [-sig <real>] [-eps <real>]
           [-A <real>] [-B <real>] [-C <real>] [-qi <real>]
           [-qj <real>] [-sigfac <real>]
```

### Description

`gmx sigeps` is a simple utility that converts C6/C12 or C6/Cn combinations to sigma and epsilon, or vice versa. It can also plot the potential in file. In addition, it makes an approximation of a Buckingham potential to a Lennard-Jones potential.

### Options

Options to specify output files:

**`-o` [<.xvg>] (potje.xvg)**
xvgr/xmgr file

Other options:

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-c6` <real> (0.001)**
C6

**`-cn` <real> (1e-06)**
Constant for repulsion

**`-pow` <int> (12)**
Power of the repulsion term

**`-sig` <real> (0.3)**
sigma

**`-eps` <real> (1)**
epsilon

**`-A` <real> (100000)**
Buckingham A

**`-B` <real> (32)**
Buckingham B

**`-C` <real> (0.001)**
Buckingham C

**`-qi` <real> (0)**
qi

**`-qj` <real> (0)**
qj

**`-sigfac` <real> (0.7)**
Factor in front of sigma for starting the plot

### 3.11.86 gmx solvate

#### Synopsis

```
gmx solvate [-cp [<.gro/.g96/...>]] [-cs [<.gro/.g96/...>]]
            [-p [<.top>]] [-o [<.gro/.g96/...>]] [-box <vector>]
            [-radius <real>] [-scale <real>] [-shell <real>]
            [-maxsol <int>] [-[no]vel]
```

#### Description

`gmx solvate` can do one of 2 things:

1) Generate a box of solvent. Specify `-cs` and `-box`. Or specify `-cs` and `-cp` with a structure file with a box, but without atoms.

2) Solvate a solute configuration, e.g. a protein, in a bath of solvent molecules. Specify `-cp` (solute) and `-cs` (solvent). The box specified in the solute coordinate file (`-cp`) is used, unless `-box` is set. If you want the solute to be centered in the box, the program *gmx editconf* (page 208) has sophisticated options to change the box dimensions and center the solute. Solvent molecules are removed from the box where the distance between any atom of the solute molecule(s) and any atom of the solvent molecule is less than the sum of the scaled van der Waals radii of both atoms. A database (`vdwradii.dat`) of van der Waals radii is read by the program, and the resulting radii scaled by `-scale`. If radii are not found in the database, those atoms are assigned the (pre-scaled) distance `-radius`. Note that the usefulness of those radii depends on the atom names, and thus varies widely with force field.

The default solvent is Simple Point Charge water (SPC), with coordinates from `$GMXLIB/spc216.gro`. These coordinates can also be used for other 3-site water models, since a short equibilibration will remove the small differences between the models. Other solvents are also supported, as well as mixed solvents. The only restriction to solvent types is that a solvent molecule consists of exactly one residue. The residue information in the coordinate files is used, and should therefore be more or less consistent. In practice this means that two subsequent solvent molecules in the solvent coordinate file should have different residue number. The box of solute is built by stacking the coordinates read from the coordinate file. This means that these coordinates should be equlibrated in periodic boundary conditions to ensure a good alignment of molecules on the stacking interfaces. The `-maxsol` option simply adds only the first `-maxsol` solvent molecules and leaves out the rest that would have fitted into the box. This can create a void that can cause problems later. Choose your volume wisely.

Setting `-shell` larger than zero will place a layer of water of the specified thickness (nm) around the solute. Hint: it is a good idea to put the protein in the center of a box first (using *gmx editconf* (page 208)).

Finally, `gmx solvate` will optionally remove lines from your topology file in which a number of solvent molecules is already added, and adds a line with the total number of solvent molecules in your coordinate file.

### Options

Options to specify input files:

**-cp [<.gro/.g96/...>] (protein.gro) (Optional)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

**-cs [<.gro/.g96/...>] (spc216.gro) (Library)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

Options to specify input/output files:

**-p [<.top>] (topol.top) (Optional)**
Topology file

Options to specify output files:

**-o [<.gro/.g96/...>] (out.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

Other options:

**-box <vector> (0 0 0)**
Box size (in nm)

**-radius <real> (0.105)**
Default van der Waals distance

**-scale <real> (0.57)**
Scale factor to multiply Van der Waals radii from the database in share/gromacs/top/vdwradii.dat. The default value of 0.57 yields density close to 1000 g/l for proteins in water.

**-shell <real> (0)**
Thickness of optional water layer around solute

**-maxsol <int> (0)**
Maximum number of solvent molecules to add if they fit in the box. If zero (default) this is ignored

**-[no]vel (no)**
Keep velocities from input solute and solvent

### Known Issues

- Molecules must be whole in the initial configurations.

## 3.11.87 gmx sorient

### Synopsis

```
gmx sorient [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.
↪ndx>]]
            [-o [<.xvg>]] [-no [<.xvg>]] [-ro [<.xvg>]]
            [-co [<.xvg>]] [-rc [<.xvg>]] [-b <time>] [-e <time>]
            [-dt <time>] [-[no]w] [-xvg <enum>] [-[no]com]␣
↪[-[no]v23]
            [-rmin <real>] [-rmax <real>] [-cbin <real>]
```

```
[-rbin <real>] [-[no]pbc]
```

## Description

`gmx sorient` analyzes solvent orientation around solutes. It calculates two angles between the vector from one or more reference positions to the first atom of each solvent molecule:

- theta_1: the angle with the vector from the first atom of the solvent molecule to the midpoint between atoms 2 and 3.
- theta_2: the angle with the normal of the solvent plane, defined by the same three atoms, or, when the option `-v23` is set, the angle with the vector between atoms 2 and 3.

The reference can be a set of atoms or the center of mass of a set of atoms. The group of solvent atoms should consist of 3 atoms per solvent molecule. Only solvent molecules between `-rmin` and `-rmax` are considered for `-o` and `-no` each frame.

`-o`: distribution of cos(theta_1) for rmin<=r<=rmax.

`-no`: distribution of cos(theta_2) for rmin<=r<=rmax.

`-ro`: <cos(theta_1)> and <3cos(^2theta_2)-1> as a function of the distance.

`-co`: the sum over all solvent molecules within distance r of cos(theta_1) and 3cos(^2(theta_2)-1) as a function of r.

`-rc`: the distribution of the solvent molecules as a function of r

## Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-o` [<.xvg>] (sori.xvg)**
xvgr/xmgr file

**`-no` [<.xvg>] (snor.xvg)**
xvgr/xmgr file

**`-ro` [<.xvg>] (sord.xvg)**
xvgr/xmgr file

**`-co` [<.xvg>] (scum.xvg)**
xvgr/xmgr file

**`-rc` [<.xvg>] (scount.xvg)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]com (no)**
Use the center of mass as the reference position

**-[no]v23 (no)**
Use the vector between atoms 2 and 3

**-rmin <real> (0)**
Minimum distance (nm)

**-rmax <real> (0.5)**
Maximum distance (nm)

**-cbin <real> (0.02)**
Binwidth for the cosine

**-rbin <real> (0.02)**
Binwidth for r (nm)

**-[no]pbc (no)**
Check PBC for the center of mass calculation. Only necessary when your reference group consists of several molecules.

### 3.11.88 gmx spatial

#### Synopsis

```
gmx spatial [-s [<.tpr/.gro/...>]] [-f [<.xtc/.trr/...>]] [-n [<.
↪ndx>]]
            [-b <time>] [-e <time>] [-dt <time>] [-[no]w] [-[no]pbc]
            [-[no]div] [-ign <int>] [-bin <real>] [-nab <int>]
```

## Description

`gmx spatial` calculates the spatial distribution function and outputs it in a form that can be read by VMD as Gaussian98 cube format. For a system of 32,000 atoms and a 50 ns trajectory, the SDF can be generated in about 30 minutes, with most of the time dedicated to the two runs through `trjconv` that are required to center everything properly. This also takes a whole bunch of space (3 copies of the trajectory file). Still, the pictures are pretty and very informative when the fitted selection is properly made. 3-4 atoms in a widely mobile group (like a free amino acid in solution) works well, or select the protein backbone in a stable folded structure to get the SDF of solvent and look at the time-averaged solvation shell. It is also possible using this program to generate the SDF based on some arbitrary Cartesian coordinate. To do that, simply omit the preliminary *gmx trjconv* (page 334) steps.

Usage:

1. Use *gmx make_ndx* (page 255) to create a group containing the atoms around which you want the SDF
2. `gmx trjconv -s a.tpr -f a.tng -o b.tng -boxcenter tric -ur compact -pbc none`
3. `gmx trjconv -s a.tpr -f b.tng -o c.tng -fit rot+trans`
4. run `gmx spatial` on the `c.tng` output of step #3.
5. Load `grid.cube` into VMD and view as an isosurface.

**Note** that systems such as micelles will require `gmx trjconv -pbc cluster` between steps 1 and 2.

## Warnings

The SDF will be generated for a cube that contains all bins that have some non-zero occupancy. However, the preparatory `-fit rot+trans` option to *gmx trjconv* (page 334) implies that your system will be rotating and translating in space (in order that the selected group does not). Therefore the values that are returned will only be valid for some region around your central group/coordinate that has full overlap with system volume throughout the entire translated/rotated system over the course of the trajectory. It is up to the user to ensure that this is the case.

## Risky options

To reduce the amount of space and time required, you can output only the coords that are going to be used in the first and subsequent run through *gmx trjconv* (page 334). However, be sure to set the `-nab` option to a sufficiently high value since memory is allocated for cube bins based on the initial coordinates and the `-nab` option value.

### Options

Options to specify input files:

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-[no]pbc` (no)**
Use periodic boundary conditions for computing distances

**`-[no]div` (yes)**
Calculate and apply the divisor for bin occupancies based on atoms/minimal cube size. Set as TRUE for visualization and as FALSE (`-nodiv`) to get accurate counts per frame

**`-ign` <int> (-1)**
Do not display this number of outer cubes (positive values may reduce boundary speckles; -1 ensures outer surface is visible)

**`-bin` <real> (0.05)**
Width of the bins (nm)

**`-nab` <int> (16)**
Number of additional bins to ensure proper memory allocation

### Known Issues

- When the allocated memory is not large enough, an error may occur suggesting the use of the `-nab` (Number of Additional Bins) option or increasing the `-nab` value.

### 3.11.89 gmx spol

#### Synopsis

```
gmx spol [-f [<.xtc/.trr/...>]] [-s [<.tpr>]] [-n [<.ndx>]]
         [-o [<.xvg>]] [-b <time>] [-e <time>] [-dt <time>] [-[no]w]
         [-xvg <enum>] [-[no]com] [-refat <int>] [-rmin <real>]
         [-rmax <real>] [-dip <real>] [-bw <real>]
```

#### Description

`gmx spol` analyzes dipoles around a solute; it is especially useful for polarizable water. A group of reference atoms, or a center of mass reference (option `-com`) and a group of solvent atoms is required. The program splits the group of solvent atoms into molecules. For each solvent molecule the distance to the closest atom in reference group or to the COM is determined. A cumulative distribution of these distances is plotted. For each distance between `-rmin` and `-rmax` the inner product of the distance vector and the dipole of the solvent molecule is determined. For solvent molecules with net charge (ions), the net charge of the ion is subtracted evenly from all atoms in the selection of each ion. The average of these dipole components is printed. The same is done for the polarization, where the average dipole is subtracted from the instantaneous dipole. The magnitude of the average dipole is set with the option `-dip`, the direction is defined by the vector from the first atom in the selected solvent group to the midpoint between the second and the third atom.

#### Options

Options to specify input files:

**`-f [<.xtc/.trr/...>] (traj.xtc)`**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s [<.tpr>] (topol.tpr)`**
Portable xdr run input file

**`-n [<.ndx>] (index.ndx) (Optional)`**
Index file

Options to specify output files:

**`-o [<.xvg>] (scdist.xvg)`**
xvgr/xmgr file

Other options:

**`-b <time> (0)`**
Time of first frame to read from trajectory (default unit ps)

**`-e <time> (0)`**
Time of last frame to read from trajectory (default unit ps)

**`-dt <time> (0)`**
Only use frame when t MOD dt = first time (default unit ps)

**`-[no]w (no)`**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]com (no)**
Use the center of mass as the reference position

**-refat <int> (1)**
The reference atom of the solvent molecule

**-rmin <real> (0)**
Maximum distance (nm)

**-rmax <real> (0.32)**
Maximum distance (nm)

**-dip <real> (0)**
The average dipole (D)

**-bw <real> (0.01)**
The bin width

### 3.11.90 gmx tcaf

#### Synopsis

```
gmx tcaf [-f [<.trr/.cpt/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>]]
         [-ot [<.xvg>]] [-oa [<.xvg>]] [-o [<.xvg>]] [-of [<.xvg>]]
         [-oc [<.xvg>]] [-ov [<.xvg>]] [-b <time>] [-e <time>]
         [-dt <time>] [-[no]w] [-xvg <enum>] [-[no]mol] [-[no]k34]
         [-wt <real>] [-acflen <int>] [-[no]normalize] [-P <enum>]
         [-fitfn <enum>] [-beginfit <real>] [-endfit <real>]
```

#### Description

`gmx tcaf` computes tranverse current autocorrelations. These are used to estimate the shear viscosity, eta. For details see: Palmer, Phys. Rev. E 49 (1994) pp 359-366.

Transverse currents are calculated using the k-vectors (1,0,0) and (2,0,0) each also in the *y*- and *z*-direction, (1,1,0) and (1,-1,0) each also in the 2 other planes (these vectors are not independent) and (1,1,1) and the 3 other box diagonals (also not independent). For each k-vector the sine and cosine are used, in combination with the velocity in 2 perpendicular directions. This gives a total of 16*2*2=64 transverse currents. One autocorrelation is calculated fitted for each k-vector, which gives 16 TCAFs. Each of these TCAFs is fitted to f(t) = exp(-v)(cosh(Wv) + 1/W sinh(Wv)), v = -t/(2 tau), W = sqrt(1 - 4 tau eta/rho k^2), which gives 16 values of tau and eta. The fit weights decay exponentially with time constant w (given with `-wt`) as exp(-t/w), and the TCAF and fit are calculated up to time 5*w. The eta values should be fitted to 1 - a eta(k) k^2, from which one can estimate the shear viscosity at k=0.

When the box is cubic, one can use the option `-oc`, which averages the TCAFs over all k-vectors with the same length. This results in more accurate TCAFs. Both the cubic TCAFs and fits are written to `-oc` The cubic eta estimates are also written to `-ov`.

With option `-mol`, the transverse current is determined of molecules instead of atoms. In this case, the index group should consist of molecule numbers instead of atom numbers.

The k-dependent viscosities in the `-ov` file should be fitted to eta(k) = eta_0 (1 - a k^2) to obtain the viscosity at infinite wavelength.

**Note:** make sure you write coordinates and velocities often enough. The initial, non-exponential, part of the autocorrelation function is very important for obtaining a good fit.

### Options

Options to specify input files:

**`-f` [<.trr/.cpt/...>] (traj.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-ot` [<.xvg>] (transcur.xvg) (Optional)**
xvgr/xmgr file

**`-oa` [<.xvg>] (tcaf_all.xvg)**
xvgr/xmgr file

**`-o` [<.xvg>] (tcaf.xvg)**
xvgr/xmgr file

**`-of` [<.xvg>] (tcaf_fit.xvg)**
xvgr/xmgr file

**`-oc` [<.xvg>] (tcaf_cub.xvg) (Optional)**
xvgr/xmgr file

**`-ov` [<.xvg>] (visc_k.xvg)**
xvgr/xmgr file

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-dt` <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-[no]mol` (no)**
Calculate TCAF of molecules

**-[no]k34 (no)**
Also use k=(3,0,0) and k=(4,0,0)

**-wt <real> (5)**
Exponential decay time for the TCAF fit weights

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### 3.11.91 gmx traj

#### Synopsis

```
gmx traj [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>]]
         [-ox [<.xvg>]] [-oxt [<.xtc/.trr/...>]] [-ov [<.xvg>]]
         [-of [<.xvg>]] [-ob [<.xvg>]] [-ot [<.xvg>]] [-ekt [<.xvg>
↪]]
         [-ekr [<.xvg>]] [-vd [<.xvg>]] [-cv [<.pdb>]] [-cf [<.pdb>
↪]]
         [-av [<.xvg>]] [-af [<.xvg>]] [-b <time>] [-e <time>]
         [-dt <time>] [-tu <enum>] [-[no]w] [-xvg <enum>] [-[no]com]
         [-[no]pbc] [-[no]mol] [-[no]nojump] [-[no]x] [-[no]y]
         [-[no]z] [-ng <int>] [-[no]len] [-[no]fp] [-bin <real>]
         [-ctime <real>] [-scale <real>]
```

#### Description

`gmx traj` plots coordinates, velocities, forces and/or the box. With `-com` the coordinates, velocities and forces are calculated for the center of mass of each group. When `-mol` is set, the numbers in the index file are interpreted as molecule numbers and the same procedure as with `-com` is used for each molecule.

Option `-ot` plots the temperature of each group, provided velocities are present in the trajectory file. No corrections are made for constrained degrees of freedom! This implies `-com`.

Options `-ekt` and `-ekr` plot the translational and rotational kinetic energy of each group, provided velocities are present in the trajectory file. This implies `-com`.

Options `-cv` and `-cf` write the average velocities and average forces as temperature factors to a *.pdb* (page 581) file with the average coordinates or the coordinates at `-ctime`. The temperature factors

are scaled such that the maximum is 10. The scaling can be changed with the option `-scale`. To get the velocities or forces of one frame set both `-b` and `-e` to the time of desired frame. When averaging over frames you might need to use the `-nojump` option to obtain the correct average coordinates. If you select either of these option the average force and velocity for each atom are written to an *.xvg* (page 589) file as well (specified with `-av` or `-af`).

Option `-vd` computes a velocity distribution, i.e. the norm of the vector is plotted. In addition in the same graph the kinetic energy distribution is given.

See *gmx trajectory* (page 331) for plotting similar data for selections.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**`-ox` [<.xvg>] (coord.xvg) (Optional)**
xvgr/xmgr file

**`-oxt` [<.xtc/.trr/...>] (coord.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-ov` [<.xvg>] (veloc.xvg) (Optional)**
xvgr/xmgr file

**`-of` [<.xvg>] (force.xvg) (Optional)**
xvgr/xmgr file

**`-ob` [<.xvg>] (box.xvg) (Optional)**
xvgr/xmgr file

**`-ot` [<.xvg>] (temp.xvg) (Optional)**
xvgr/xmgr file

**`-ekt` [<.xvg>] (ektrans.xvg) (Optional)**
xvgr/xmgr file

**`-ekr` [<.xvg>] (ekrot.xvg) (Optional)**
xvgr/xmgr file

**`-vd` [<.xvg>] (veldist.xvg) (Optional)**
xvgr/xmgr file

**`-cv` [<.pdb>] (veloc.pdb) (Optional)**
Protein data bank file

**-cf [<.pdb>] (force.pdb) (Optional)**
Protein data bank file

**-av [<.xvg>] (all_veloc.xvg) (Optional)**
xvgr/xmgr file

**-af [<.xvg>] (all_force.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]com (no)**
Plot data for the com of each group

**-[no]pbc (yes)**
Make molecules whole for COM

**-[no]mol (no)**
Index contains molecule numbers instead of atom numbers

**-[no]nojump (no)**
Remove jumps of atoms across the box

**-[no]x (yes)**
Plot X-component

**-[no]y (yes)**
Plot Y-component

**-[no]z (yes)**
Plot Z-component

**-ng <int> (1)**
Number of groups to consider

**-[no]len (no)**
Plot vector length

**-[no]fp (no)**
Full precision output

**-bin <real> (1)**
Binwidth for velocity histogram (nm/ps)

**-ctime <real> (-1)**
Use frame at this time for x in -cv and -cf instead of the average x

**-scale <real> (0)**
Scale factor for *.pdb* (page 581) output, 0 is autoscale

### 3.11.92 gmx trajectory

#### Synopsis

```
gmx trajectory [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]]
               [-n [<.ndx>]] [-ox [<.xvg>]] [-ov [<.xvg>]]
               [-of [<.xvg>]] [-b <time>] [-e <time>] [-dt <time>]
               [-tu <enum>] [-fgroup <selection>] [-xvg <enum>]
               [-[no]rmpbc] [-[no]pbc] [-sf <file>] [-selrpos <enum>]
               [-seltype <enum>] [-select <selection>] [-[no]x]
               [-[no]y] [-[no]z] [-[no]len]
```

#### Description

gmx trajectory plots coordinates, velocities, and/or forces for provided selections. By default, the X, Y, and Z components for the requested vectors are plotted, but specifying one or more of -len, -x, -y, and -z overrides this.

For dynamic selections, currently the values are written out for all positions that the selection could select.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc) (Optional)**
Input trajectory or single configuration: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Input structure: *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Extra index groups

Options to specify output files:

**-ox [<.xvg>] (coord.xvg) (Optional)**
Coordinates for each position as a function of time

**-ov [<.xvg>] (veloc.xvg) (Optional)**
Velocities for each position as a function of time

**-of [<.xvg>] (force.xvg) (Optional)**
Forces for each position as a function of time

Other options:

**-b <time> (0)**
First frame (ps) to read from trajectory

**-e <time> (0)**
Last frame (ps) to read from trajectory

**-dt <time> (0)**
Only use frame if t MOD dt == first time (ps)

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-fgroup <selection>**
Atoms stored in the trajectory file (if not set, assume first N atoms)

**-xvg <enum> (xmgrace)**
Plot formatting: xmgrace, xmgr, none

**-[no]rmpbc (yes)**
Make molecules whole for each frame

**-[no]pbc (yes)**
Use periodic boundary conditions for distance calculation

**-sf <file>**
Provide selections from files

**-selrpos <enum> (atom)**
Selection reference positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-seltype <enum> (atom)**
Default selection output positions: atom, res_com, res_cog, mol_com, mol_cog, whole_res_com, whole_res_cog, whole_mol_com, whole_mol_cog, part_res_com, part_res_cog, part_mol_com, part_mol_cog, dyn_res_com, dyn_res_cog, dyn_mol_com, dyn_mol_cog

**-select <selection>**
Selections to analyze

**-[no]x (yes)**
Plot X component

**-[no]y (yes)**
Plot Y component

**-[no]z (yes)**
Plot Z component

**-[no]len (no)**
Plot vector length

### 3.11.93 gmx trjcat

#### Synopsis

```
gmx trjcat [-f [<.xtc/.trr/...> [...]]] [-n [<.ndx>]] [-demux [<.
↪xvg>]]
           [-o [<.xtc/.trr/...> [...]]] [-tu <enum>] [-xvg <enum>]
           [-b <time>] [-e <time>] [-dt <time>] [-[no]settime]
           [-[no]sort] [-[no]keeplast] [-[no]overwrite] [-[no]cat]
```

#### Description

`gmx trjcat` concatenates several input trajectory files in sorted order. In case of double time frames the one in the later file is used. By specifying `-settime` you will be asked for the start time of each file. The input files are taken from the command line, such that a command like `gmx trjcat -f *.trr -o fixed.trr` should do the trick. Using `-cat`, you can simply paste several files together without removal of frames with identical time stamps.

One important option is inferred when the output file is amongst the input files. In that case that particular file will be appended to which implies you do not need to store double the amount of data. Obviously the file to append to has to be the one with lowest starting time since one can only append at the end of a file.

If the `-demux` option is given, the N trajectories that are read, are written in another order as specified in the *.xvg* (page 589) file. The *.xvg* (page 589) file should contain something like:

```
0   0   1   2   3   4   5
2   1   0   2   3   5   4
```

The first number is the time, and subsequent numbers point to trajectory indices. The frames corresponding to the numbers present at the first line are collected into the output trajectory. If the number of frames in the trajectory does not match that in the *.xvg* (page 589) file then the program tries to be smart. Beware.

#### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...> [...]] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

**`-demux` [<.xvg>] (remd.xvg) (Optional)**
xvgr/xmgr file

Options to specify output files:

**`-o` [<.xtc/.trr/...> [...]] (trajout.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

Other options:

**-tu <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-b <time> (-1)**
First time to use (ps)

**-e <time> (-1)**
Last time to use (ps)

**-dt <time> (0)**
Only write frame when t MOD dt = first time (ps)

**-[no]settime (no)**
Change starting time interactively

**-[no]sort (yes)**
Sort trajectory files (not frames)

**-[no]keeplast (no)**
Keep overlapping frames at end of trajectory

**-[no]overwrite (no)**
Overwrite overlapping frames during appending

**-[no]cat (no)**
Do not discard double time frames

### 3.11.94 gmx trjconv

#### Synopsis

```
gmx trjconv [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.
↪ndx>]]
            [-fr [<.ndx>]] [-sub [<.ndx>]] [-drop [<.xvg>]]
            [-o [<.xtc/.trr/...>]] [-b <time>] [-e <time>]
            [-tu <enum>] [-[no]w] [-xvg <enum>] [-skip <int>]
            [-dt <time>] [-[no]round] [-dump <time>] [-t0 <time>]
            [-timestep <time>] [-pbc <enum>] [-ur <enum>]
            [-[no]center] [-boxcenter <enum>] [-box <vector>]
            [-trans <vector>] [-shift <vector>] [-fit <enum>]
            [-ndec <int>] [-[no]vel] [-[no]force] [-trunc <time>]
            [-exec <string>] [-split <time>] [-[no]sep]
            [-nzero <int>] [-dropunder <real>] [-dropover <real>]
            [-[no]conect]
```

### Description

`gmx trjconv` can convert trajectory files in many ways:

- from one format to another
- select a subset of atoms
- change the periodicity representation
- keep multimeric molecules together
- center atoms in the box
- fit atoms to reference structure
- reduce the number of frames
- change the timestamps of the frames (`-t0` and `-timestep`)
- select frames within a certain range of a quantity given in an *.xvg* (page 589) file.

The option to write subtrajectories (-sub) based on the information obtained from cluster analysis has been removed from `gmx trjconv` and is now part of [gmx extract-cluster]

*gmx trjcat* (page 333) is better suited for concatenating multiple trajectory files.

The following formats are supported for input and output: *.xtc* (page 588), *.trr* (page 586), *.gro* (page 577), `.g96`, *.pdb* (page 581) and *.tng* (page 584). The file formats are detected from the file extension. The precision of the *.xtc* (page 588) output is taken from the input file for *.xtc* (page 588), *.gro* (page 577) and *.pdb* (page 581), and from the `-ndec` option for other input formats. The precision is always taken from `-ndec`, when this option is set. All other formats have fixed precision. *.trr* (page 586) output can be single or double precision, depending on the precision of the `gmx trjconv` binary. Note that velocities are only supported in *.trr* (page 586), *.tng* (page 584), *.gro* (page 577) and `.g96` files.

Option `-sep` can be used to write every frame to a separate `.gro, .g96` or *.pdb* (page 581) file. By default, all frames all written to one file. *.pdb* (page 581) files with all frames concatenated can be viewed with `rasmol -nmrpdb`.

It is possible to select part of your trajectory and write it out to a new trajectory file in order to save disk space, e.g. for leaving out the water from a trajectory of a protein in water. **ALWAYS** put the original trajectory on tape! We recommend to use the portable *.xtc* (page 588) format for your analysis to save disk space and to have portable files. When writing *.tng* (page 584) output the file will contain one molecule type of the correct count if the selection name matches the molecule name and the selected atoms match all atoms of that molecule. Otherwise the whole selection will be treated as one single molecule containing all the selected atoms.

There are two options for fitting the trajectory to a reference either for essential dynamics analysis, etc. The first option is just plain fitting to a reference structure in the structure file. The second option is a progressive fit in which the first timeframe is fitted to the reference structure in the structure file to obtain and each subsequent timeframe is fitted to the previously fitted structure. This way a continuous trajectory is generated, which might not be the case when using the regular fit method, e.g. when your protein undergoes large conformational transitions.

Option `-pbc` sets the type of periodic boundary condition treatment:

- `mol` puts the center of mass of molecules in the box, and requires a run input file to be supplied with `-s`.

- `res` puts the center of mass of residues in the box.
- `atom` puts all the atoms in the box.
- `nojump` checks if atoms jump across the box and then puts them back. This has the effect that all molecules will remain whole (provided they were whole in the initial conformation). **Note** that this ensures a continuous trajectory but molecules may diffuse out of the box. The starting configuration for this procedure is taken from the structure file, if one is supplied, otherwise it is the first frame.
- `cluster` clusters all the atoms in the selected index such that they are all closest to the center of mass of the cluster, which is iteratively updated. **Note** that this will only give meaningful results if you in fact have a cluster. Luckily that can be checked afterwards using a trajectory viewer. Note also that if your molecules are broken this will not work either.
- `whole` only makes broken molecules whole.

Option `-ur` sets the unit cell representation for options `mol`, `res` and `atom` of `-pbc`. All three options give different results for triclinic boxes and identical results for rectangular boxes. `rect` is the ordinary brick shape. `tric` is the triclinic unit cell. `compact` puts all atoms at the closest distance from the center of the box. This can be useful for visualizing e.g. truncated octahedra or rhombic dodecahedra. The center for options `tric` and `compact` is `tric` (see below), unless the option `-boxcenter` is set differently.

Option `-center` centers the system in the box. The user can select the group which is used to determine the geometrical center. Option `-boxcenter` sets the location of the center of the box for options `-pbc` and `-center`. The center options are: `tric`: half of the sum of the box vectors, `rect`: half of the box diagonal, `zero`: zero. Use option `-pbc mol` in addition to `-center` when you want all molecules in the box after the centering.

Option `-box` sets the size of the new box. This option only works for leading dimensions and is thus generally only useful for rectangular boxes. If you want to modify only some of the dimensions, e.g. when reading from a trajectory, you can use -1 for those dimensions that should stay the same It is not always possible to use combinations of `-pbc`, `-fit`, `-ur` and `-center` to do exactly what you want in one call to `gmx trjconv`. Consider using multiple calls, and check out the GROMACS website for suggestions.

With `-dt`, it is possible to reduce the number of frames in the output. This option relies on the accuracy of the times in your input trajectory, so if these are inaccurate use the `-timestep` option to modify the time (this can be done simultaneously). For making smooth movies, the program *gmx filter* (page 220) can reduce the number of frames while using low-pass frequency filtering, this reduces aliasing of high frequency motions.

Using `-trunc` `gmx trjconv` can truncate *.trr* (page 586) in place, i.e. without copying the file. This is useful when a run has crashed during disk I/O (i.e. full disk), or when two contiguous trajectories must be concatenated without having double frames.

Option `-dump` can be used to extract a frame at or near one specific time from your trajectory. If the frames in the trajectory are not in temporal order, the result is unspecified.

Option `-drop` reads an *.xvg* (page 589) file with times and values. When options `-dropunder` and/or `-dropover` are set, frames with a value below and above the value of the respective options will not be written.

### Options

Options to specify input files:

**`-f` [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**`-n` [<.ndx>] (index.ndx) (Optional)**
Index file

**`-fr` [<.ndx>] (frames.ndx) (Optional)**
Index file

**`-sub` [<.ndx>] (cluster.ndx) (Optional)**
Index file

**`-drop` [<.xvg>] (drop.xvg) (Optional)**
xvgr/xmgr file

Options to specify output files:

**`-o` [<.xtc/.trr/...>] (trajout.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

Other options:

**`-b` <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**`-e` <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**`-tu` <enum> (ps)**
Unit for time values: fs, ps, ns, us, ms, s

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-xvg` <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**`-skip` <int> (1)**
Only write every nr-th frame

**`-dt` <time> (0)**
Only write frame when t MOD dt = first time (ps)

**`-[no]round` (no)**
Round measurements to nearest picosecond

**`-dump` <time> (-1)**
Dump frame nearest specified time (ps)

**`-t0` <time> (0)**
Starting time (ps) (default: don't change)

**-timestep <time> (0)**
Change time step between input frames (ps)

**-pbc <enum> (none)**
PBC treatment (see help text for full description): none, mol, res, atom, nojump, cluster, whole

**-ur <enum> (rect)**
Unit-cell representation: rect, tric, compact

**-[no]center (no)**
Center atoms in box

**-boxcenter <enum> (tric)**
Center for -pbc and -center: tric, rect, zero

**-box <vector> (0 0 0)**
Size for new cubic box (default: read from input)

**-trans <vector> (0 0 0)**
All coordinates will be translated by trans. This can advantageously be combined with -pbc mol -ur compact.

**-shift <vector> (0 0 0)**
All coordinates will be shifted by framenr*shift

**-fit <enum> (none)**
Fit molecule to ref structure in the structure file: none, rot+trans, rotxy+transxy, translation, transxy, progressive

**-ndec <int> (3)**
Number of decimal places to write to .xtc output

**-[no]vel (yes)**
Read and write velocities if possible

**-[no]force (no)**
Read and write forces if possible

**-trunc <time> (-1)**
Truncate input trajectory file after this time (ps)

**-exec <string>**
Execute command for every output frame with the frame number as argument

**-split <time> (0)**
Start writing new file when t MOD split = first time (ps)

**-[no]sep (no)**
Write each frame to a separate .gro, .g96 or .pdb file

**-nzero <int> (0)**
If the -sep flag is set, use these many digits for the file numbers and prepend zeros as needed

**-dropunder <real> (0)**
Drop all frames below this value

**-dropover <real> (0)**
Drop all frames above this value

**-[no]conect (no)**
Add CONECT PDB records when writing *.pdb* (page 581) files. Useful for visualization of non-standard molecules, e.g. coarse grained ones. Can only be done when a topology (tpr) file is present

### 3.11.95 gmx trjorder

#### Synopsis

```
gmx trjorder [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.
↪ndx>]]
             [-o [<.xtc/.trr/...>]] [-nshell [<.xvg>]] [-b <time>]
             [-e <time>] [-dt <time>] [-xvg <enum>] [-na <int>]
             [-da <int>] [-[no]com] [-r <real>] [-[no]z]
```

#### Description

`gmx trjorder` orders molecules according to the smallest distance to atoms in a reference group or on z-coordinate (with option `-z`). With distance ordering, it will ask for a group of reference atoms and a group of molecules. For each frame of the trajectory the selected molecules will be reordered according to the shortest distance between atom number `-da` in the molecule and all the atoms in the reference group. The center of mass of the molecules can be used instead of a reference atom by setting `-da` to 0. All atoms in the trajectory are written to the output trajectory.

`gmx trjorder` can be useful for e.g. analyzing the n waters closest to a protein. In that case the reference group would be the protein and the group of molecules would consist of all the water atoms. When an index group of the first n waters is made, the ordered trajectory can be used with any GROMACS program to analyze the n closest waters.

If the output file is a *.pdb* (page 581) file, the distance to the reference target will be stored in the B-factor field in order to color with e.g. Rasmol.

With option `-nshell` the number of molecules within a shell of radius `-r` around the reference group are printed.

#### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**-o [<.xtc/.trr/...>] (ordered.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-nshell [<.xvg>] (nshell.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-na <int> (3)**
Number of atoms in a molecule

**-da <int> (1)**
Atom used for the distance calculation, 0 is COM

**-[no]com (no)**
Use the distance to the center of mass of the reference group

**-r <real> (0)**
Cutoff used for the distance calculation when computing the number of molecules in a shell around e.g. a protein

**-[no]z (no)**
Order molecules on z-coordinate

### 3.11.96 gmx tune_pme

#### Synopsis

```
gmx tune_pme [-s [<.tpr>]] [-cpi [<.cpt>]] [-table [<.xvg>]]
             [-tablep [<.xvg>]] [-tableb [<.xvg>]]
             [-rerun [<.xtc/.trr/...>]] [-ei [<.edi>]] [-p [<.out>]]
             [-err [<.log>]] [-so [<.tpr>]] [-o [<.trr/.cpt/...>]]
             [-x [<.xtc/.tng>]] [-cpo [<.cpt>]]
             [-c [<.gro/.g96/...>]] [-e [<.edr>]] [-g [<.log>]]
             [-dhdl [<.xvg>]] [-field [<.xvg>]] [-tpi [<.xvg>]]
             [-tpid [<.xvg>]] [-eo [<.xvg>]] [-px [<.xvg>]]
             [-pf [<.xvg>]] [-ro [<.xvg>]] [-ra [<.log>]]
             [-rs [<.log>]] [-rt [<.log>]] [-mtx [<.mtx>]]
             [-swap [<.xvg>]] [-bo [<.trr/.cpt/...>]] [-bx [<.xtc>]]
             [-bcpo [<.cpt>]] [-bc [<.gro/.g96/...>]] [-be [<.edr>]]
             [-bg [<.log>]] [-beo [<.xvg>]] [-bdhdl [<.xvg>]]
             [-bfield [<.xvg>]] [-btpi [<.xvg>]] [-btpid [<.xvg>]]
             [-bdevout [<.xvg>]] [-brunav [<.xvg>]] [-bpx [<.xvg>]]
```

```
[-bpf [<.xvg>]] [-bro [<.xvg>]] [-bra [<.log>]]
[-brs [<.log>]] [-brt [<.log>]] [-bmtx [<.mtx>]]
[-bdn [<.ndx>]] [-bswap [<.xvg>]] [-xvg <enum>]
[-mdrun <string>] [-np <int>] [-npstring <enum>]
[-ntmpi <int>] [-r <int>] [-max <real>] [-min <real>]
[-npme <enum>] [-fix <int>] [-rmax <real>]
[-rmin <real>] [-[no]scalevdw] [-ntpr <int>]
[-steps <int>] [-resetstep <int>] [-nsteps <int>]
[-[no]launch] [-[no]bench] [-[no]check]
[-gpu_id <string>] [-[no]append] [-[no]cpnum]
[-deffnm <string>]
```

## Description

For a given number `-np` or `-ntmpi` of ranks, `gmx tune_pme` systematically times *gmx mdrun* (page 258) with various numbers of PME-only ranks and determines which setting is fastest. It will also test whether performance can be enhanced by shifting load from the reciprocal to the real space part of the Ewald sum. Simply pass your *.tpr* (page 586) file to `gmx tune_pme` together with other options for *gmx mdrun* (page 258) as needed.

`gmx tune_pme` needs to call *gmx mdrun* (page 258) and so requires that you specify how to call mdrun with the argument to the `-mdrun` parameter. Depending how you have built GROMACS, values such as 'gmx mdrun', 'gmx_d mdrun', or 'gmx_mpi mdrun' might be needed.

The program that runs MPI programs can be set in the environment variable MPIRUN (defaults to 'mpirun'). Note that for certain MPI frameworks, you need to provide a machine- or hostfile. This can also be passed via the MPIRUN variable, e.g.

`export MPIRUN="/usr/local/mpirun -machinefile hosts"` Note that in such cases it is normally necessary to compile and/or run `gmx tune_pme` without MPI support, so that it can call the MPIRUN program.

Before doing the actual benchmark runs, `gmx tune_pme` will do a quick check whether *gmx mdrun* (page 258) works as expected with the provided parallel settings if the `-check` option is activated (the default). Please call `gmx tune_pme` with the normal options you would pass to *gmx mdrun* (page 258) and add `-np` for the number of ranks to perform the tests on, or `-ntmpi` for the number of threads. You can also add `-r` to repeat each test several times to get better statistics.

`gmx tune_pme` can test various real space / reciprocal space workloads for you. With `-ntpr` you control how many extra *.tpr* (page 586) files will be written with enlarged cutoffs and smaller Fourier grids respectively. Typically, the first test (number 0) will be with the settings from the input *.tpr* (page 586) file; the last test (number `ntpr`) will have the Coulomb cutoff specified by `-rmax` with a somewhat smaller PME grid at the same time. In this last test, the Fourier spacing is multiplied with `rmax`/rcoulomb. The remaining *.tpr* (page 586) files will have equally-spaced Coulomb radii (and Fourier spacings) between these extremes. **Note** that you can set `-ntpr` to 1 if you just seek the optimal number of PME-only ranks; in that case your input *.tpr* (page 586) file will remain unchanged.

For the benchmark runs, the default of 1000 time steps should suffice for most MD systems. The dynamic load balancing needs about 100 time steps to adapt to local load imbalances, therefore the time step counters are by default reset after 100 steps. For large systems (>1M atoms), as well as for a higher accuracy of the measurements, you should set `-resetstep` to a higher value. From the 'DD' load imbalance entries in the md.log output file you can tell after how many steps the load is sufficiently balanced. Example call:

```
gmx tune_pme -np 64 -s protein.tpr -launch
```

After calling *gmx mdrun* (page 258) several times, detailed performance information is available in the output file `perf.out`. **Note** that during the benchmarks, a couple of temporary files are written (options `-b*`), these will be automatically deleted after each test.

If you want the simulation to be started automatically with the optimized parameters, use the command line option `-launch`.

Basic support for GPU-enabled `mdrun` exists. Give a string containing the IDs of the GPUs that you wish to use in the optimization in the `-gpu_id` command-line argument. This works exactly like `mdrun -gpu_id`, does not imply a mapping, and merely declares the eligible set of GPU devices. `gmx-tune_pme` will construct calls to mdrun that use this set appropriately. `gmx-tune_pme` does not support `-gputasks`.

### Options

Options to specify input files:

**`-s` [<.tpr>] (topol.tpr)**
Portable xdr run input file

**`-cpi` [<.cpt>] (state.cpt) (Optional)**
Checkpoint file

**`-table` [<.xvg>] (table.xvg) (Optional)**
xvgr/xmgr file

**`-tablep` [<.xvg>] (tablep.xvg) (Optional)**
xvgr/xmgr file

**`-tableb` [<.xvg>] (table.xvg) (Optional)**
xvgr/xmgr file

**`-rerun` [<.xtc/.trr/...>] (rerun.xtc) (Optional)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**`-ei` [<.edi>] (sam.edi) (Optional)**
ED sampling input

Options to specify output files:

**`-p` [<.out>] (perf.out)**
Generic output file

**`-err` [<.log>] (bencherr.log)**
Log file

**`-so` [<.tpr>] (tuned.tpr)**
Portable xdr run input file

**`-o` [<.trr/.cpt/...>] (traj.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**`-x` [<.xtc/.tng>] (traj_comp.xtc) (Optional)**
Compressed trajectory (tng format or portable xdr format)

**-cpo [<.cpt>] (state.cpt) (Optional)**
Checkpoint file

**-c [<.gro/.g96/...>] (confout.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

**-e [<.edr>] (ener.edr)**
Energy file

**-g [<.log>] (md.log)**
Log file

**-dhdl [<.xvg>] (dhdl.xvg) (Optional)**
xvgr/xmgr file

**-field [<.xvg>] (field.xvg) (Optional)**
xvgr/xmgr file

**-tpi [<.xvg>] (tpi.xvg) (Optional)**
xvgr/xmgr file

**-tpid [<.xvg>] (tpidist.xvg) (Optional)**
xvgr/xmgr file

**-eo [<.xvg>] (edsam.xvg) (Optional)**
xvgr/xmgr file

**-px [<.xvg>] (pullx.xvg) (Optional)**
xvgr/xmgr file

**-pf [<.xvg>] (pullf.xvg) (Optional)**
xvgr/xmgr file

**-ro [<.xvg>] (rotation.xvg) (Optional)**
xvgr/xmgr file

**-ra [<.log>] (rotangles.log) (Optional)**
Log file

**-rs [<.log>] (rotslabs.log) (Optional)**
Log file

**-rt [<.log>] (rottorque.log) (Optional)**
Log file

**-mtx [<.mtx>] (nm.mtx) (Optional)**
Hessian matrix

**-swap [<.xvg>] (swapions.xvg) (Optional)**
xvgr/xmgr file

**-bo [<.trr/.cpt/...>] (bench.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**-bx [<.xtc>] (bench.xtc)**
Compressed trajectory (portable xdr format): xtc

**-bcpo [<.cpt>] (bench.cpt)**
Checkpoint file

**-bc [<.gro/.g96/...>] (bench.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp

**-be [<.edr>] (bench.edr)**
Energy file

**-bg [<.log>] (bench.log)**
Log file

**-beo [<.xvg>] (benchedo.xvg) (Optional)**
xvgr/xmgr file

**-bdhdl [<.xvg>] (benchdhdl.xvg) (Optional)**
xvgr/xmgr file

**-bfield [<.xvg>] (benchfld.xvg) (Optional)**
xvgr/xmgr file

**-btpi [<.xvg>] (benchtpi.xvg) (Optional)**
xvgr/xmgr file

**-btpid [<.xvg>] (benchtpid.xvg) (Optional)**
xvgr/xmgr file

**-bdevout [<.xvg>] (benchdev.xvg) (Optional)**
xvgr/xmgr file

**-brunav [<.xvg>] (benchrnav.xvg) (Optional)**
xvgr/xmgr file

**-bpx [<.xvg>] (benchpx.xvg) (Optional)**
xvgr/xmgr file

**-bpf [<.xvg>] (benchpf.xvg) (Optional)**
xvgr/xmgr file

**-bro [<.xvg>] (benchrot.xvg) (Optional)**
xvgr/xmgr file

**-bra [<.log>] (benchrota.log) (Optional)**
Log file

**-brs [<.log>] (benchrots.log) (Optional)**
Log file

**-brt [<.log>] (benchrott.log) (Optional)**
Log file

**-bmtx [<.mtx>] (benchn.mtx) (Optional)**
Hessian matrix

**-bdn [<.ndx>] (bench.ndx) (Optional)**
Index file

**-bswap [<.xvg>] (benchswp.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-mdrun <string>**
Command line to run a simulation, e.g. 'gmx mdrun' or 'gmx_mpi mdrun'

**-np <int> (1)**
Number of ranks to run the tests on (must be > 2 for separate PME ranks)

**-npstring <enum> (np)**
Name of the `$MPIRUN` option that specifies the number of ranks to use ('np', or 'n'; use 'none' if there is no such option): np, n, none

**-ntmpi <int> (1)**
Number of MPI-threads to run the tests on (turns MPI & mpirun off)

**-r <int> (2)**
Repeat each test this often

**-max <real> (0.5)**
Max fraction of PME ranks to test with

**-min <real> (0.25)**
Min fraction of PME ranks to test with

**-npme <enum> (auto)**
Within -min and -max, benchmark all possible values for `-npme`, or just a reasonable subset. Auto neglects -min and -max and chooses reasonable values around a guess for npme derived from the .tpr: auto, all, subset

**-fix <int> (-2)**
If >= -1, do not vary the number of PME-only ranks, instead use this fixed value and only vary rcoulomb and the PME grid spacing.

**-rmax <real> (0)**
If >0, maximal rcoulomb for -ntpr>1 (rcoulomb upscaling results in fourier grid downscaling)

**-rmin <real> (0)**
If >0, minimal rcoulomb for -ntpr>1

**-[no]scalevdw (yes)**
Scale rvdw along with rcoulomb

**-ntpr <int> (0)**
Number of *.tpr* (page 586) files to benchmark. Create this many files with different rcoulomb scaling factors depending on -rmin and -rmax. If < 1, automatically choose the number of *.tpr* (page 586) files to test

**-steps <int> (1000)**
Take timings for this many steps in the benchmark runs

**-resetstep <int> (1500)**
Let dlb equilibrate this many steps before timings are taken (reset cycle counters after this many steps)

**-nsteps <int> (-1)**
If non-negative, perform this many steps in the real run (overwrites nsteps from *.tpr* (page 586), add *.cpt* (page 575) steps)

**-[no]launch (no)**
Launch the real simulation after optimization

**-[no]bench (yes)**
Run the benchmarks or just create the input *.tpr* (page 586) files?

**-[no]check (yes)**
Before the benchmark runs, check whether mdrun works in parallel

**-gpu_id <string>**
List of unique GPU device IDs that are eligible for use

**-[no]append (yes)**
Append to previous output files when continuing from checkpoint instead of adding the simulation part number to all file names (for launch only)

**-[no]cpnum (no)**
Keep and number checkpoint files (launch only)

**-deffnm <string>**
Set the default filenames (launch only)

### 3.11.97 gmx vanhove

#### Synopsis

```
gmx vanhove [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.
↪ndx>]]
            [-om [<.xpm>]] [-or [<.xvg>]] [-ot [<.xvg>]] [-b <time>]
            [-e <time>] [-dt <time>] [-[no]w] [-xvg <enum>]
            [-sqrt <real>] [-fm <int>] [-rmax <real>] [-rbin <real>]
            [-mmax <real>] [-nlevels <int>] [-nr <int>] [-fr <int>]
            [-rt <real>] [-ft <int>]
```

#### Description

`gmx vanhove` computes the Van Hove correlation function. The Van Hove G(r,t) is the probability that a particle that is at r_0 at time zero can be found at position r_0+r at time t. `gmx vanhove` determines G not for a vector r, but for the length of r. Thus it gives the probability that a particle moves a distance of r in time t. Jumps across the periodic boundaries are removed. Corrections are made for scaling due to isotropic or anisotropic pressure coupling.

With option `-om` the whole matrix can be written as a function of t and r or as a function of sqrt(t) and r (option `-sqrt`).

With option `-or` the Van Hove function is plotted for one or more values of t. Option `-nr` sets the number of times, option `-fr` the number spacing between the times. The binwidth is set with option `-rbin`. The number of bins is determined automatically.

With option `-ot` the integral up to a certain distance (option `-rt`) is plotted as a function of time.

For all frames that are read the coordinates of the selected particles are stored in memory. Therefore the program may use a lot of memory. For options `-om` and `-ot` the program may be slow. This is because the calculation scales as the number of frames times `-fm` or `-ft`. Note that with the `-dt` option the memory usage and calculation time can be reduced.

### Options

Options to specify input files:

**-f [<.xtc/.trr/...>] (traj.xtc)**
Trajectory: *xtc* (page 588) *trr* (page 586) *cpt* (page 575) *gro* (page 577) *g96* (page 576) *pdb* (page 581) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**-om [<.xpm>] (vanhove.xpm) (Optional)**
X PixMap compatible matrix file

**-or [<.xvg>] (vanhove_r.xvg) (Optional)**
xvgr/xmgr file

**-ot [<.xvg>] (vanhove_t.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-sqrt <real> (0)**
Use sqrt(t) on the matrix axis which binspacing # in sqrt(ps)

**-fm <int> (0)**
Number of frames in the matrix, 0 is plot all

**-rmax <real> (2)**
Maximum r in the matrix (nm)

**-rbin <real> (0.01)**
Binwidth in the matrix and for `-or` (nm)

**-mmax <real> (0)**
Maximum density in the matrix, 0 is calculate (1/nm)

**-nlevels <int> (81)**
Number of levels in the matrix

**-nr <int> (1)**
Number of curves for the -or output

**-fr <int> (0)**
Frame spacing for the -or output

**-rt <real> (0)**
Integration limit for the -ot output (nm)

**-ft <int> (0)**
Number of frames in the -ot output, 0 is plot all

### 3.11.98 gmx velacc

#### Synopsis

```
gmx velacc [-f [<.trr/.cpt/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>
↪]]
           [-o [<.xvg>]] [-os [<.xvg>]] [-b <time>] [-e <time>]
           [-dt <time>] [-[no]w] [-xvg <enum>] [-[no]m] [-[no]recip]
           [-[no]mol] [-acflen <int>] [-[no]normalize] [-P <enum>]
           [-fitfn <enum>] [-beginfit <real>] [-endfit <real>]
```

#### Description

gmx velacc computes the velocity autocorrelation function. When the -m option is used, the momentum autocorrelation function is calculated.

With option -mol the velocity autocorrelation function of molecules is calculated. In this case the index group should consist of molecule numbers instead of atom numbers.

By using option -os you can also extract the estimated (vibrational) power spectrum, which is the Fourier transform of the velocity autocorrelation function. Be sure that your trajectory contains frames with velocity information (i.e. nstvout was set in your original *.mdp* (page 579) file), and that the time interval between data collection points is much shorter than the time scale of the autocorrelation.

#### Options

Options to specify input files:

**-f [<.trr/.cpt/...>] (traj.trr)**
Full precision trajectory: *trr* (page 586) *cpt* (page 575) *tng* (page 584) h5md

**-s [<.tpr/.gro/...>] (topol.tpr) (Optional)**
Structure+mass(db): *tpr* (page 586) *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent

**-n [<.ndx>] (index.ndx) (Optional)**
Index file

Options to specify output files:

**-o [<.xvg>] (vac.xvg)**
xvgr/xmgr file

**-os [<.xvg>] (spectrum.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-b <time> (0)**
Time of first frame to read from trajectory (default unit ps)

**-e <time> (0)**
Time of last frame to read from trajectory (default unit ps)

**-dt <time> (0)**
Only use frame when t MOD dt = first time (default unit ps)

**-[no]w (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-[no]m (no)**
Calculate the momentum autocorrelation function

**-[no]recip (yes)**
Use cm^-1 on X-axis instead of 1/ps for spectra.

**-[no]mol (no)**
Calculate the velocity acf of molecules

**-acflen <int> (-1)**
Length of the ACF, default is half the number of frames

**-[no]normalize (yes)**
Normalize ACF

**-P <enum> (0)**
Order of Legendre polynomial for ACF (0 indicates none): 0, 1, 2, 3

**-fitfn <enum> (none)**
Fit function: none, exp, aexp, exp_exp, exp5, exp7, exp9

**-beginfit <real> (0)**
Time where to begin the exponential fit of the correlation function

**-endfit <real> (-1)**
Time where to end the exponential fit of the correlation function, -1 is until the end

### 3.11.99 gmx wham

#### Synopsis

```
gmx wham [-ix [<.dat>]] [-if [<.dat>]] [-it [<.dat>]] [-is [<.dat>]]
         [-iiact [<.dat>]] [-tab [<.dat>]] [-o [<.xvg>]]
         [-hist [<.xvg>]] [-oiact [<.xvg>]] [-bsres [<.xvg>]]
         [-bsprof [<.xvg>]] [-xvg <enum>] [-min <real>] [-max <real>
↪]
         [-[no]auto] [-bins <int>] [-temp <real>] [-tol <real>]
         [-[no]v] [-b <real>] [-e <real>] [-dt <real>]
```

[**-[no]histonly**] [**-[no]boundsonly**] [**-[no]log**] [**-unit** *<enum>*]
[**-zprof0** *<real>*] [**-[no]cycl**] [**-[no]sym**] [**-[no]ac**]
[**-acsig** *<real>*] [**-ac-trestart** *<real>*] [**-nBootstrap** *<int>*]
[**-bs-method** *<enum>*] [**-bs-tau** *<real>*] [**-bs-seed** *<int>*]
[**-histbs-block** *<int>*] [**-[no]vbs**]

## Description

`gmx wham` is an analysis program that implements the Weighted Histogram Analysis Method (WHAM). It is intended to analyze output files generated by umbrella sampling simulations to compute a potential of mean force (PMF).

`gmx wham` is currently not fully up to date. It only supports pull setups where the first pull coordinate(s) is/are umbrella pull coordinates and, if multiple coordinates need to be analyzed, all used the same geometry and dimensions. In most cases this is not an issue.

At present, three input modes are supported.

- With option `-it`, the user provides a file which contains the file names of the umbrella simulation run-input files (*.tpr* (page 586) files), AND, with option `-ix`, a file which contains file names of the pullx `mdrun` output files. The *.tpr* (page 586) and pullx files must be in corresponding order, i.e. the first *.tpr* (page 586) created the first pullx, etc.
- Same as the previous input mode, except that the user provides the pull force output file names (`pullf.xvg`) with option `-if`. From the pull force the position in the umbrella potential is computed. This does not work with tabulated umbrella potentials.

By default, all pull coordinates found in all pullx/pullf files are used in WHAM. If only some of the pull coordinates should be used, a pull coordinate selection file (option `-is`) can be provided. The selection file must contain one line for each tpr file in tpr-files.dat. Each of these lines must contain one digit (0 or 1) for each pull coordinate in the tpr file. Here, 1 indicates that the pull coordinate is used in WHAM, and 0 means it is omitted. Example: If you have three tpr files, each containing 4 pull coordinates, but only pull coordinates 1 and 2 should be used, coordsel.dat looks like this:

```
1 1 0 0
1 1 0 0
1 1 0 0
```

By default, the output files are:

```
``-o``      PMF output file
``-hist``   Histograms output file
```

Always check whether the histograms sufficiently overlap.

The umbrella potential is assumed to be harmonic and the force constants are read from the *.tpr* (page 586) files. If a non-harmonic umbrella force was applied a tabulated potential can be provided with `-tab`.

### WHAM options

- `-bins` Number of bins used in analysis
- `-temp` Temperature in the simulations
- `-tol` Stop iteration if profile (probability) changed less than tolerance
- `-auto` Automatic determination of boundaries
- `-min,-max` Boundaries of the profile

The data points that are used to compute the profile can be restricted with options `-b`, `-e`, and `-dt`. Adjust `-b` to ensure sufficient equilibration in each umbrella window.

With `-log` (default) the profile is written in energy units, otherwise (with `-nolog`) as probability. The unit can be specified with `-unit`. With energy output, the energy in the first bin is defined to be zero. If you want the free energy at a different position to be zero, set `-zprof0` (useful with bootstrapping, see below).

For cyclic or periodic reaction coordinates (dihedral angle, channel PMF without osmotic gradient), the option `-cycl` is useful. `gmx wham` will make use of the periodicity of the system and generate a periodic PMF. The first and the last bin of the reaction coordinate will assumed be be neighbors.

Option `-sym` symmetrizes the profile around z=0 before output, which may be useful for, e.g. membranes.

### Parallelization

If available, the number of OpenMP threads used by gmx wham can be controlled by setting the `OMP_NUM_THREADS` environment variable.

### Autocorrelations

With `-ac`, `gmx wham` estimates the integrated autocorrelation time (IACT) tau for each umbrella window and weights the respective window with 1/[1+2*tau/dt]. The IACTs are written to the file defined with `-oiact`. In verbose mode, all autocorrelation functions (ACFs) are written to `hist_autocorr.xvg`. Because the IACTs can be severely underestimated in case of limited sampling, option `-acsig` allows one to smooth the IACTs along the reaction coordinate with a Gaussian (sigma provided with `-acsig`, see output in `iact.xvg`). Note that the IACTs are estimated by simple integration of the ACFs while the ACFs are larger 0.05. If you prefer to compute the IACTs by a more sophisticated (but possibly less robust) method such as fitting to a double exponential, you can compute the IACTs with *gmx analyze* (page 154) and provide them to `gmx wham` with the file `iact-in.dat` (option `-iiact`), which should contain one line per input file (pullx/pullf file) and one column per pull coordinate in the respective file.

### Error analysis

Statistical errors may be estimated with bootstrap analysis. Use it with care, otherwise the statistical error may be substantially underestimated. More background and examples for the bootstrap technique can be found in Hub, de Groot and Van der Spoel, JCTC (2010) 6: 3713-3720. `-nBootstrap` defines the number of bootstraps (use, e.g., 100). Four bootstrapping methods are supported and selected with `-bs-method`.

- `b-hist` Default: complete histograms are considered as independent data points, and the bootstrap is carried out by assigning random weights to the histograms ("Bayesian bootstrap"). Note that each point along the reaction coordinate must be covered by multiple independent histograms (e.g. 10 histograms), otherwise the statistical error is underestimated.
- `hist` Complete histograms are considered as independent data points. For each bootstrap, N histograms are randomly chosen from the N given histograms (allowing duplication, i.e. sampling with replacement). To avoid gaps without data along the reaction coordinate blocks of histograms (`-histbs-block`) may be defined. In that case, the given histograms are divided into blocks and only histograms within each block are mixed. Note that the histograms within each block must be representative for all possible histograms, otherwise the statistical error is underestimated.
- `traj` The given histograms are used to generate new random trajectories, such that the generated data points are distributed according the given histograms and properly autocorrelated. The autocorrelation time (ACT) for each window must be known, so use `-ac` or provide the ACT with `-iiact`. If the ACT of all windows are identical (and known), you can also provide them with `-bs-tau`. Note that this method may severely underestimate the error in case of limited sampling, that is if individual histograms do not represent the complete phase space at the respective positions.
- `traj-gauss` The same as method `traj`, but the trajectories are not bootstrapped from the umbrella histograms but from Gaussians with the average and width of the umbrella histograms. That method yields similar error estimates like method `traj`.

Bootstrapping output:

- `-bsres` Average profile and standard deviations
- `-bsprof` All bootstrapping profiles

With `-vbs` (verbose bootstrapping), the histograms of each bootstrap are written, and, with bootstrap method `traj`, the cumulative distribution functions of the histograms.

### Options

Options to specify input files:

**`-ix` [<.dat>] (pullx-files.dat) (Optional)**
Generic data file

**`-if` [<.dat>] (pullf-files.dat) (Optional)**
Generic data file

**`-it` [<.dat>] (tpr-files.dat) (Optional)**
Generic data file

**`-is` [<.dat>] (coordsel.dat) (Optional)**
Generic data file

**-iiact [<.dat>] (iact-in.dat) (Optional)**
Generic data file

**-tab [<.dat>] (umb-pot.dat) (Optional)**
Generic data file

Options to specify output files:

**-o [<.xvg>] (profile.xvg)**
xvgr/xmgr file

**-hist [<.xvg>] (histo.xvg)**
xvgr/xmgr file

**-oiact [<.xvg>] (iact.xvg) (Optional)**
xvgr/xmgr file

**-bsres [<.xvg>] (bsResult.xvg) (Optional)**
xvgr/xmgr file

**-bsprof [<.xvg>] (bsProfs.xvg) (Optional)**
xvgr/xmgr file

Other options:

**-xvg <enum> (xmgrace)**
xvg plot formatting: xmgrace, xmgr, none

**-min <real> (0)**
Minimum coordinate in profile

**-max <real> (0)**
Maximum coordinate in profile

**-[no]auto (yes)**
Determine min and max automatically

**-bins <int> (200)**
Number of bins in profile

**-temp <real> (298)**
Temperature

**-tol <real> (1e-06)**
Tolerance

**-[no]v (no)**
Verbose mode

**-b <real> (50)**
First time to analyse (ps)

**-e <real> (1e+20)**
Last time to analyse (ps)

**-dt <real> (0)**
Analyse only every dt ps

**-[no]histonly (no)**
Write histograms and exit

**-[no]boundsonly (no)**
Determine min and max and exit (with -auto)

**-[no]log (yes)**
Calculate the log of the profile before printing

**-unit <enum> (kJ)**
Energy unit in case of log output: kJ, kCal, kT

**-zprof0 <real> (0)**
Define profile to 0.0 at this position (with -log)

**-[no]cycl (no)**
Create cyclic/periodic profile. Assumes min and max are the same point.

**-[no]sym (no)**
Symmetrize profile around z=0

**-[no]ac (no)**
Calculate integrated autocorrelation times and use in wham

**-acsig <real> (0)**
Smooth autocorrelation times along reaction coordinate with Gaussian of this sigma

**-ac-trestart <real> (1)**
When computing autocorrelation functions, restart computing every .. (ps)

**-nBootstrap <int> (0)**
nr of bootstraps to estimate statistical uncertainty (e.g., 200)

**-bs-method <enum> (b-hist)**
Bootstrap method: b-hist, hist, traj, traj-gauss

**-bs-tau <real> (0)**
Autocorrelation time (ACT) assumed for all histograms. Use option -ac if ACT is unknown.

**-bs-seed <int> (-1)**
Seed for bootstrapping. (-1 = use time)

**-histbs-block <int> (8)**
When mixing histograms only mix within blocks of -histbs-block.

**-[no]vbs (no)**
Verbose bootstrapping. Print the CDFs and a histogram file for each bootstrap.

### 3.11.100 gmx wheel

#### Synopsis

```
gmx wheel [-f [<.dat>]] [-o [<.eps>]] [-r0 <int>] [-rot0 <real>]
          [-T <string>] [-[no]nn]
```

### Description

`gmx wheel` plots a helical wheel representation of your sequence. The input sequence is in the *.dat* (page 576) file where the first line contains the number of residues and each consecutive line contains a residue name.

### Options

Options to specify input files:

**`-f` [<.dat>] (nnnice.dat)**
Generic data file

Options to specify output files:

**`-o` [<.eps>] (plot.eps)**
Encapsulated PostScript (tm) file

Other options:

**`-r0` <int> (1)**
The first residue number in the sequence

**`-rot0` <real> (0)**
Rotate around an angle initially (90 degrees makes sense)

**`-T` <string>**
Plot a title in the center of the wheel (must be shorter than 10 characters, or it will overwrite the wheel)

**`-[no]nn` (yes)**
Toggle numbers

## 3.11.101 gmx x2top

### Synopsis

```
gmx x2top [-f [<.gro/.g96/...>]] [-o [<.top>]] [-r [<.rtp>]]
          [-ff <string>] [-[no]v] [-nexcl <int>] [-[no]H14]
          [-[no]alldih] [-[no]remdih] [-[no]pairs] [-name <string>]
          [-[no]pbc] [-[no]pdbq] [-[no]param] [-[no]round]
          [-kb <real>] [-kt <real>] [-kp <real>]
```

### Description

`gmx x2top` generates a primitive topology from a coordinate file. The program assumes all hydrogens are present when defining the hybridization from the atom name and the number of bonds. The program can also make an *.rtp* (page 582) entry, which you can then add to the *.rtp* (page 582) database.

When `-param` is set, equilibrium distances and angles and force constants will be printed in the topology for all interactions. The equilibrium distances and angles are taken from the input coordinates, the force constant are set with command line options. The force fields somewhat supported currently are:

G53a5 GROMOS96 53a5 Forcefield (official distribution)

oplsaa OPLS-AA/L all-atom force field (2001 aminoacid dihedrals)

The corresponding data files can be found in the library directory with name `atomname2type.n2t`. Check Chapter 5 of the manual for more information about file formats. By default, the force field selection is interactive, but you can use the `-ff` option to specify one of the short names above on the command line instead. In that case `gmx x2top` just looks for the corresponding file.

### Options

Options to specify input files:

**`-f` [<.gro/.g96/...>] (conf.gro)**
Structure file: *gro* (page 577) *g96* (page 576) *pdb* (page 581) brk ent esp *tpr* (page 586)

Options to specify output files:

**`-o` [<.top>] (out.top) (Optional)**
Topology file

**`-r` [<.rtp>] (out.rtp) (Optional)**
Residue Type file used by pdb2gmx

Other options:

**`-ff` <string> (oplsaa)**
Force field for your simulation. Type "select" for interactive selection.

**`-[no]v` (no)**
Generate verbose output in the top file.

**`-nexcl` <int> (3)**
Number of exclusions

**`-[no]H14` (yes)**
Use 3rd neighbour interactions for hydrogen atoms

**`-[no]alldih` (no)**
Generate all proper dihedrals

**`-[no]remdih` (no)**
Remove dihedrals on the same bond as an improper

**`-[no]pairs` (yes)**
Output 1-4 interactions (pairs) in topology file

**`-name` <string> (ICE)**
Name of your molecule

**`-[no]pbc` (yes)**
Use periodic boundary conditions.

**`-[no]pdbq` (no)**
Use the B-factor supplied in a *.pdb* (page 581) file for the atomic charges

**`-[no]param` (yes)**
Print parameters in the output

**`-[no]round` (yes)**
Round off measured values

**-kb <real> (400000)**
Bonded force constant (kJ/mol/nm^2)

**-kt <real> (400)**
Angle force constant (kJ/mol/rad^2)

**-kp <real> (5)**
Dihedral angle force constant (kJ/mol/rad^2)

### Known Issues

- The atom type selection is primitive. Virtually no chemical knowledge is used
- Periodic boundary conditions screw up the bonding
- No improper dihedrals are generated
- The atoms to atomtype translation table is incomplete (`atomname2type.n2t` file in the data directory). Please extend it and send the results back to the GROMACS crew.

## 3.11.102 gmx xpm2ps

### Synopsis

```
gmx xpm2ps [-f [<.xpm>]] [-f2 [<.xpm>]] [-di [<.m2p>]] [-do [<.m2p>
↪]]
           [-o [<.eps>]] [-xpm [<.xpm>]] [-[no]w] [-[no]frame]
           [-title <enum>] [-[no]yonce] [-legend <enum>]
           [-diag <enum>] [-size <real>] [-bx <real>] [-by <real>]
           [-rainbow <enum>] [-gradient <vector>] [-skip <int>]
           [-[no]zeroline] [-legoffset <int>] [-combine <enum>]
           [-cmin <real>] [-cmax <real>]
```

### Description

`gmx xpm2ps` makes a beautiful color plot of an XPixelMap file. Labels and axis can be displayed, when they are supplied in the correct matrix format. Matrix data may be generated by programs such as *gmx rms* (page 294) or *gmx mdmat* (page 256).

Parameters are set in the `.m2p` file optionally supplied with `-di`. Reasonable defaults are provided. Settings for the *y*-axis default to those for the *x*-axis. Font names have a defaulting hierarchy: titlefont -> legendfont; titlefont -> (xfont -> yfont -> ytickfont) -> xtickfont, e.g. setting titlefont sets all fonts, setting xfont sets yfont, ytickfont and xtickfont.

When no `.m2p` file is supplied, many settings are taken from command line options. The most important option is `-size`, which sets the size of the whole matrix in postscript units. This option can be overridden with the `-bx` and `-by` options (and the corresponding parameters in the `.m2p` file), which set the size of a single matrix element.

With `-f2` a second matrix file can be supplied. Both matrix files will be read simultaneously and the upper left half of the first one (`-f`) is plotted together with the lower right half of the second one (`-f2`). The diagonal will contain values from the matrix file selected with `-diag`. Plotting of the diagonal values can be suppressed altogether by setting `-diag` to `none`. In this case, a new color map will

be generated with a red gradient for negative numbers and a blue for positive. If the color coding and legend labels of both matrices are identical, only one legend will be displayed, else two separate legends are displayed. With `-combine`, an alternative operation can be selected to combine the matrices. The output range is automatically set to the actual range of the combined matrix. This can be overridden with `-cmin` and `-cmax`.

`-title` can be set to `none` to suppress the title, or to `ylabel` to show the title in the Y-label position (alongside the *y*-axis).

With the `-rainbow` option, dull grayscale matrices can be turned into attractive color pictures.

Merged or rainbowed matrices can be written to an XPixelMap file with the `-xpm` option.

### Options

Options to specify input files:

**`-f` [<.xpm>] (root.xpm)**
X PixMap compatible matrix file

**`-f2` [<.xpm>] (root2.xpm) (Optional)**
X PixMap compatible matrix file

**`-di` [<.m2p>] (ps.m2p) (Optional, Library)**
Input file for mat2ps

Options to specify output files:

**`-do` [<.m2p>] (out.m2p) (Optional)**
Input file for mat2ps

**`-o` [<.eps>] (plot.eps) (Optional)**
Encapsulated PostScript (tm) file

**`-xpm` [<.xpm>] (root.xpm) (Optional)**
X PixMap compatible matrix file

Other options:

**`-[no]w` (no)**
View output *.xvg* (page 589), *.xpm* (page 587), *.eps* (page 576) and *.pdb* (page 581) files

**`-[no]frame` (yes)**
Display frame, ticks, labels, title and legend

**`-title` <enum> (top)**
Show title at: top, once, ylabel, none

**`-[no]yonce` (no)**
Show y-label only once

**`-legend` <enum> (both)**
Show legend: both, first, second, none

**`-diag` <enum> (first)**
Diagonal: first, second, none

**`-size` <real> (400)**
Horizontal size of the matrix in ps units

**-bx <real> (0)**
Element x-size, overrides -size (also y-size when -by is not set)

**-by <real> (0)**
Element y-size

**-rainbow <enum> (no)**
Rainbow colors, convert white to: no, blue, red

**-gradient <vector> (0 0 0)**
Re-scale colormap to a smooth gradient from white {1,1,1} to {r,g,b}

**-skip <int> (1)**
only write out every nr-th row and column

**-[no]zeroline (no)**
insert line in *.xpm* (page 587) matrix where axis label is zero

**-legoffset <int> (0)**
Skip first N colors from *.xpm* (page 587) file for the legend

**-combine <enum> (halves)**
Combine two matrices: halves, add, sub, mult, div

**-cmin <real> (0)**
Minimum for combination output

**-cmax <real> (0)**
Maximum for combination output

### 3.11.103 Command-line interface and conventions

All GROMACS commands require an option before any arguments (i.e., all command-line arguments need to be preceded by an argument starting with a dash, and values not starting with a dash are arguments to the preceding option). Most options, except for boolean flags, expect an argument (or multiple in some cases) after the option name. The argument must be a separate command-line argument, i.e., separated by space, as in `-f traj.xtc`. If more than one argument needs to be given to an option, they should be similarly separated from each other. Some options also have default arguments, i.e., just specifying the option without any argument uses the default argument. If an option is not specified at all, a default value is used; in the case of optional files, the default might be not to use that file (see below).

All GROMACS command options start with a single dash, whether they are single- or multiple-letter options. However, two dashes are also recognized (starting from 5.1).

In addition to command-specific options, some options are handled by the **gmx** wrapper, and can be specified for any command. See *wrapper binary help* (page 143) for the list of such options. These options are recognized both before the command name (e.g., **gmx -quiet grompp**) as well as after the command name (e.g., **gmx grompp -quiet**). There is also a `-hidden` option that can be specified in combination with `-h` to show help for advanced/developer-targeted options.

Most analysis commands can process a trajectory with fewer atoms than the run input or structure file, but only if the trajectory consists of the first *n* atoms of the run input or structure file.

#### Handling specific types of command-line options

**boolean options**
Boolean flags can be specified like `-pbc` and negated like `-nopbc`. It is also possible to use an explicit value like `-pbc no` and `-pbc yes`.

**file name options**
Options that accept files names have features that support using default file names (where the default file name is specific to that option):

- If a required option is not set, the default is used.
- If an option is marked optional, the file is not used unless the option is set (or other conditions make the file required).
- If an option is set, and no file name is provided, the default is used.

All such options will accept file names without a file extension. The extension is automatically appended in such a case. When multiple input formats are accepted, such as a generic structure format, the directory will be searched for files of each type with the supplied or default name. When no file with a recognized extension is found, an error is given. For output files with multiple formats, a default file type will be used.

Some file formats can also be read from compressed (`.Z` or `.gz`) formats.

**enum options**
Enumerated options (enum) should be used with one of the arguments listed in the option description. The argument may be abbreviated, and the first match to the shortest argument in the list will be selected.

**vector options**
Some options accept a vector of values. Either 1 or 3 parameters can be supplied; when only one parameter is supplied the two other values are also set to this value.

**selection options**
See *Selection syntax and usage* (page 371).

### 3.11.104 Commands by name

- *gmx* (page 143) - molecular dynamics simulation suite
- *gmx anaeig* (page 151) - Analyze eigenvectors/normal modes
- *gmx analyze* (page 154) - Analyze data sets
- *gmx angle* (page 158) - Calculate distributions and correlations for angles and dihedrals
- *gmx awh* (page 160) - Extract data from an accelerated weight histogram (AWH) run
- *gmx bar* (page 161) - Calculate free energy difference estimates through Bennett's acceptance ratio
- *gmx bundle* (page 164) - Analyze bundles of axes, e.g., helices
- *gmx check* (page 165) - Check and compare files
- *gmx chi* (page 167) - Calculate everything you want to know about chi and other dihedrals
- *gmx cluster* (page 171) - Cluster structures

- *gmx clustsize* (page 175) - Calculate size distributions of atomic clusters
- *gmx confrms* (page 177) - Fit two structures and calculates the RMSD
- *gmx convert-tpr* (page 178) - Make a modified run-input file
- *gmx convert-trj* (page 179) - Converts between different trajectory types
- *gmx covar* (page 181) - Calculate and diagonalize the covariance matrix
- *gmx current* (page 183) - Calculate dielectric constants and current autocorrelation function
- *gmx density* (page 185) - Calculate the density of the system
- *gmx densmap* (page 187) - Calculate 2D planar or axial-radial density maps
- *gmx densorder* (page 189) - Calculate surface fluctuations
- *gmx dielectric* (page 191) - Calculate frequency dependent dielectric constants
- *gmx dipoles* (page 193) - Compute the total dipole plus fluctuations
- *gmx disre* (page 196) - Analyze distance restraints
- *gmx distance* (page 198) - Calculate distances between pairs of positions
- *gmx dos* (page 200) - Analyze density of states and properties based on that
- *gmx dssp* (page 202) - Calculate protein secondary structure via DSSP algorithm
- *gmx dump* (page 205) - Make binary files human readable
- *gmx dyecoupl* (page 206) - Extract dye dynamics from trajectories
- *gmx editconf* (page 208) - Convert and manipulates structure files
- *gmx eneconv* (page 211) - Convert energy files
- *gmx enemat* (page 212) - Extract an energy matrix from an energy file
- *gmx energy* (page 214) - Writes energies to xvg files and display averages
- *gmx extract-cluster* (page 218) - Allows extracting frames corresponding to clusters from trajectory
- *gmx filter* (page 220) - Frequency filter trajectories, useful for making smooth movies
- *gmx freevolume* (page 222) - Calculate free volume
- *gmx gangle* (page 223) - Calculate angles
- *gmx genconf* (page 226) - Multiply a conformation in 'random' orientations
- *gmx genion* (page 227) - Generate monoatomic ions on energetically favorable positions
- *gmx genrestr* (page 228) - Generate position restraints or distance restraints for index groups
- *gmx grompp* (page 230) - Make a run input file
- *gmx gyrate* (page 233) - Calculate radius of gyration of a molecule
- *gmx gyrate-legacy* (page 234) - Calculate the radius of gyration
- *gmx h2order* (page 236) - Compute the orientation of water molecules
- *gmx hbond* (page 237) - Compute and analyze hydrogen bonds.
- *gmx hbond-legacy* (page 240) - Compute and analyze hydrogen bonds

- *gmx helix* (page 244) - Calculate basic properties of alpha helices
- *gmx helixorient* (page 245) - Calculate local pitch/bending/rotation/orientation inside helices
- *gmx help* (page 247) - Print help information
- *gmx hydorder* (page 247) - Compute tetrahedrality parameters around a given atom
- *gmx insert-molecules* (page 248) - Insert molecules into existing vacancies
- *gmx lie* (page 250) - Estimate free energy from linear combinations
- *gmx make_edi* (page 251) - Generate input files for essential dynamics sampling
- *gmx make_ndx* (page 255) - Make index files
- *gmx mdmat* (page 256) - Calculate residue contact maps
- *gmx mdrun* (page 258) - Perform a simulation, do a normal mode analysis or an energy minimization
- *gmx mindist* (page 264) - Calculate the minimum distance between two groups
- *gmx mk_angndx* (page 266) - Generate index files for 'gmx angle'
- *gmx msd* (page 266) - Compute mean squared displacements
- *gmx nmeig* (page 269) - Diagonalize the Hessian for normal mode analysis
- *gmx nmens* (page 271) - Generate an ensemble of structures from the normal modes
- *gmx nmr* (page 272) - Analyze nuclear magnetic resonance properties from an energy file
- *gmx nmtraj* (page 274) - Generate a virtual oscillating trajectory from an eigenvector
- *gmx nonbonded-benchmark* (page 275) - Benchmarking tool for the non-bonded pair kernels.
- *gmx order* (page 277) - Compute the order parameter per atom for carbon tails
- *gmx pairdist* (page 279) - Calculate pairwise distances between groups of positions
- *gmx pdb2gmx* (page 281) - Convert coordinate files to topology and FF-compliant coordinate files
- *gmx pme_error* (page 285) - Estimate the error of using PME with a given input file
- *gmx polystat* (page 286) - Calculate static properties of polymers
- *gmx potential* (page 287) - Calculate the electrostatic potential across the box
- *gmx principal* (page 289) - Calculate principal axes of inertia for a group of atoms
- *gmx rama* (page 290) - Compute Ramachandran plots
- *gmx rdf* (page 291) - Calculate radial distribution functions
- *gmx report-methods* (page 294) - Write short summary about the simulation setup to a text file and/or to the standard output.
- *gmx rms* (page 294) - Calculate RMSDs with a reference structure and RMSD matrices
- *gmx rmsdist* (page 297) - Calculate atom pair distances averaged with power -2, -3 or -6
- *gmx rmsf* (page 299) - Calculate atomic fluctuations
- *gmx rotacf* (page 301) - Calculate the rotational correlation function for molecules
- *gmx rotmat* (page 302) - Plot the rotation matrix for fitting to a reference structure

- *gmx saltbr* (page 304) - Compute salt bridges
- *gmx sans-legacy* (page 305) - Compute small angle neutron scattering spectra
- *gmx sasa* (page 307) - Compute solvent accessible surface area
- *gmx saxs-legacy* (page 309) - Compute small angle X-ray scattering spectra
- *gmx scattering* (page 310) - Calculate small angle scattering profiles for SANS or SAXS
- *gmx select* (page 312) - Print general information about selections
- *gmx sham* (page 315) - Compute free energies or other histograms from histograms
- *gmx sigeps* (page 317) - Convert c6/12 or c6/cn combinations to and from sigma/epsilon
- *gmx solvate* (page 319) - Solvate a system
- *gmx sorient* (page 320) - Analyze solvent orientation around solutes
- *gmx spatial* (page 322) - Calculate the spatial distribution function
- *gmx spol* (page 325) - Analyze solvent dipole orientation and polarization around solutes
- *gmx tcaf* (page 326) - Calculate viscosities of liquids
- *gmx traj* (page 328) - Plot x, v, f, box, temperature and rotational energy from trajectories
- *gmx trajectory* (page 331) - Print coordinates, velocities, and/or forces for selections
- *gmx trjcat* (page 333) - Concatenate trajectory files
- *gmx trjconv* (page 334) - Convert and manipulates trajectory files
- *gmx trjorder* (page 339) - Order molecules according to their distance to a group
- *gmx tune_pme* (page 340) - Time mdrun as a function of PME ranks to optimize settings
- *gmx vanhove* (page 346) - Compute Van Hove displacement and correlation functions
- *gmx velacc* (page 348) - Calculate velocity autocorrelation functions
- *gmx wham* (page 349) - Perform weighted histogram analysis after umbrella sampling
- *gmx wheel* (page 354) - Plot helical wheels
- *gmx x2top* (page 355) - Generate a primitive topology from coordinates
- *gmx xpm2ps* (page 357) - Convert XPM (XPixelMap) matrices to postscript or XPM

### 3.11.105 Commands by topic

#### Trajectory analysis

***gmx gangle*** **(page 223)**
Calculate angles

***gmx convert-trj*** **(page 179)**
Converts between different trajectory types

***gmx distance*** **(page 198)**
Calculate distances between pairs of positions

*gmx dssp* **(page 202)**
Calculate protein secondary structure via DSSP algorithm

*gmx extract-cluster* **(page 218)**
Allows extracting frames corresponding to clusters from trajectory

*gmx freevolume* **(page 222)**
Calculate free volume

*gmx hbond* **(page 237)**
Compute and analyze hydrogen bonds.

*gmx msd* **(page 266)**
Compute mean squared displacements

*gmx pairdist* **(page 279)**
Calculate pairwise distances between groups of positions

*gmx rdf* **(page 291)**
Calculate radial distribution functions

*gmx sasa* **(page 307)**
Compute solvent accessible surface area

*gmx scattering* **(page 310)**
Calculate small angle scattering profiles for SANS or SAXS

*gmx select* **(page 312)**
Print general information about selections

*gmx trajectory* **(page 331)**
Print coordinates, velocities, and/or forces for selections

*gmx gyrate* **(page 233)**
Calculate radius of gyration of a molecule

### Generating topologies and coordinates

*gmx editconf* **(page 208)**
Edit the box and write subgroups

*gmx x2top* **(page 355)**
Generate a primitive topology from coordinates

*gmx solvate* **(page 319)**
Solvate a system

*gmx insert-molecules* **(page 248)**
Insert molecules into existing vacancies

*gmx genconf* **(page 226)**
Multiply a conformation in 'random' orientations

*gmx genion* **(page 227)**
Generate monoatomic ions on energetically favorable positions

*gmx genrestr* **(page 228)**
Generate position restraints or distance restraints for index groups

*gmx pdb2gmx* **(page 281)**
Convert coordinate files to topology and FF-compliant coordinate files

### Running a simulation

*gmx grompp* **(page 230)**
Make a run input file

*gmx mdrun* **(page 258)**
Perform a simulation, do a normal mode analysis or an energy minimization

*gmx convert-tpr* **(page 178)**
Make a modified run-input file

### Viewing trajectories

*gmx nmtraj* **(page 274)**
Generate a virtual oscillating trajectory from an eigenvector

### Processing energies

*gmx enemat* **(page 212)**
Extract an energy matrix from an energy file

*gmx energy* **(page 214)**
Writes energies to xvg files and display averages

*gmx mdrun* **(page 258)**
(Re)calculate energies for trajectory frames with -rerun

### Converting files

*gmx editconf* **(page 208)**
Convert and manipulates structure files

*gmx eneconv* **(page 211)**
Convert energy files

*gmx sigeps* **(page 317)**
Convert c6/12 or c6/cn combinations to and from sigma/epsilon

*gmx trjcat* **(page 333)**
Concatenate trajectory files

*gmx trjconv* **(page 334)**
Convert and manipulates trajectory files

*gmx xpm2ps* **(page 357)**
Convert XPM (XPixelMap) matrices to postscript or XPM

### Tools

*gmx analyze* **(page 154)**
Analyze data sets

*gmx awh* **(page 160)**
Extract data from an accelerated weight histogram (AWH) run

*gmx filter* **(page 220)**
Frequency filter trajectories, useful for making smooth movies

*gmx lie* **(page 250)**
Estimate free energy from linear combinations

*gmx pme_error* **(page 285)**
Estimate the error of using PME with a given input file

*gmx sham* **(page 315)**
Compute free energies or other histograms from histograms

*gmx spatial* **(page 322)**
Calculate the spatial distribution function

*gmx traj* **(page 328)**
Plot x, v, f, box, temperature and rotational energy from trajectories

*gmx tune_pme* **(page 340)**
Time mdrun as a function of PME ranks to optimize settings

*gmx wham* **(page 349)**
Perform weighted histogram analysis after umbrella sampling

*gmx check* **(page 165)**
Check and compare files

*gmx dump* **(page 205)**
Make binary files human readable

*gmx make_ndx* **(page 255)**
Make index files

*gmx mk_angndx* **(page 266)**
Generate index files for 'gmx angle'

*gmx trjorder* **(page 339)**
Order molecules according to their distance to a group

*gmx xpm2ps* **(page 357)**
Convert XPM (XPixelMap) matrices to postscript or XPM

*gmx report-methods* **(page 294)**
Write short summary about the simulation setup to a text file and/or to the standard output.

### Distances between structures

*gmx cluster* **(page 171)**
Cluster structures

*gmx confrms* **(page 177)**
Fit two structures and calculates the RMSD

*gmx rms* **(page 294)**
Calculate RMSDs with a reference structure and RMSD matrices

*gmx rmsf* **(page 299)**
Calculate atomic fluctuations

### Distances in structures over time

*gmx mindist* **(page 264)**
Calculate the minimum distance between two groups

*gmx mdmat* **(page 256)**
Calculate residue contact maps

*gmx polystat* **(page 286)**
Calculate static properties of polymers

*gmx rmsdist* **(page 297)**
Calculate atom pair distances averaged with power -2, -3 or -6

### Mass distribution properties over time

*gmx gyrate-legacy* **(page 234)**
Calculate the radius of gyration

*gmx polystat* **(page 286)**
Calculate static properties of polymers

*gmx rdf* **(page 291)**
Calculate radial distribution functions

*gmx rotacf* **(page 301)**
Calculate the rotational correlation function for molecules

*gmx rotmat* **(page 302)**
Plot the rotation matrix for fitting to a reference structure

*gmx sans-legacy* **(page 305)**
Compute small angle neutron scattering spectra

*gmx saxs-legacy* **(page 309)**
Compute small angle X-ray scattering spectra

*gmx traj* **(page 328)**
Plot x, v, f, box, temperature and rotational energy from trajectories

*gmx vanhove* **(page 346)**
Compute Van Hove displacement and correlation functions

### Analyzing bonded interactions

*gmx angle* **(page 158)**
Calculate distributions and correlations for angles and dihedrals

*gmx mk_angndx* **(page 266)**
Generate index files for 'gmx angle'

### Structural properties

*gmx bundle* **(page 164)**
Analyze bundles of axes, e.g., helices

*gmx clustsize* **(page 175)**
Calculate size distributions of atomic clusters

*gmx disre* **(page 196)**
Analyze distance restraints

*gmx hbond-legacy* **(page 240)**
Compute and analyze hydrogen bonds

*gmx order* **(page 277)**
Compute the order parameter per atom for carbon tails

*gmx principal* **(page 289)**
Calculate principal axes of inertia for a group of atoms

*gmx rdf* **(page 291)**
Calculate radial distribution functions

*gmx saltbr* **(page 304)**
Compute salt bridges

*gmx sorient* **(page 320)**
Analyze solvent orientation around solutes

*gmx spol* **(page 325)**
Analyze solvent dipole orientation and polarization around solutes

### Kinetic properties

*gmx bar* **(page 161)**
Calculate free energy difference estimates through Bennett's acceptance ratio

*gmx current* **(page 183)**
Calculate dielectric constants and current autocorrelation function

*gmx dos* **(page 200)**
Analyze density of states and properties based on that

*gmx dyecoupl* **(page 206)**
Extract dye dynamics from trajectories

*gmx principal* **(page 289)**
Calculate principal axes of inertia for a group of atoms

*gmx tcaf* **(page 326)**
Calculate viscosities of liquids

*gmx traj* **(page 328)**
Plot x, v, f, box, temperature and rotational energy from trajectories

*gmx vanhove* **(page 346)**
Compute Van Hove displacement and correlation functions

*gmx velacc* **(page 348)**
Calculate velocity autocorrelation functions

### Electrostatic properties

*gmx current* **(page 183)**
Calculate dielectric constants and current autocorrelation function

*gmx dielectric* **(page 191)**
Calculate frequency dependent dielectric constants

*gmx dipoles* **(page 193)**
Compute the total dipole plus fluctuations

*gmx potential* **(page 287)**
Calculate the electrostatic potential across the box

*gmx spol* **(page 325)**
Analyze solvent dipole orientation and polarization around solutes

*gmx genion* **(page 227)**
Generate monoatomic ions on energetically favorable positions

### Protein-specific analysis

*gmx chi* **(page 167)**
Calculate everything you want to know about chi and other dihedrals

*gmx helix* **(page 244)**
Calculate basic properties of alpha helices

*gmx helixorient* **(page 245)**
Calculate local pitch/bending/rotation/orientation inside helices

*gmx rama* **(page 290)**
Compute Ramachandran plots

*gmx wheel* **(page 354)**
Plot helical wheels

### Interfaces

*gmx bundle* **(page 164)**
Analyze bundles of axes, e.g., helices

*gmx density* **(page 185)**
Calculate the density of the system

*gmx densmap* **(page 187)**
Calculate 2D planar or axial-radial density maps

*gmx densorder* **(page 189)**
Calculate surface fluctuations

*gmx h2order* **(page 236)**
Compute the orientation of water molecules

*gmx hydorder* **(page 247)**
Compute tetrahedrality parameters around a given atom

*gmx order* **(page 277)**
Compute the order parameter per atom for carbon tails

*gmx potential* **(page 287)**
Calculate the electrostatic potential across the box

### Covariance analysis

*gmx anaeig* **(page 151)**
Analyze the eigenvectors

*gmx covar* **(page 181)**
Calculate and diagonalize the covariance matrix

*gmx make_edi* **(page 251)**
Generate input files for essential dynamics sampling

### Normal modes

*gmx anaeig* **(page 151)**
Analyze the normal modes

*gmx nmeig* **(page 269)**
Diagonalize the Hessian for normal mode analysis

*gmx nmtraj* **(page 274)**
Generate a virtual oscillating trajectory from an eigenvector

*gmx nmens* **(page 271)**
Generate an ensemble of structures from the normal modes

*gmx grompp* **(page 230)**
Make a run input file

*gmx mdrun* **(page 258)**
Find a potential energy minimum and calculate the Hessian

### 3.11.106 Special topics

The information in these topics is also accessible through `gmx help topic` on the command line.

#### Selection syntax and usage

Selections are used to select atoms/molecules/residues for analysis. In contrast to traditional index files, selections can be dynamic, i.e., select different atoms for different trajectory frames. The GROMACS manual contains a short introductory section to selections in the Analysis chapter, including suggestions on how to get familiar with selections if you are new to the concept. The subtopics listed below provide more details on the technical and syntactic aspects of selections.

Each analysis tool requires a different number of selections and the selections are interpreted differently. The general idea is still the same: each selection evaluates to a set of positions, where a position can be an atom position or center-of-mass or center-of-geometry of a set of atoms. The tool then uses these positions for its analysis to allow very flexible processing. Some analysis tools may have limitations on the types of selections allowed.

#### Specifying selections from command line

If no selections are provided on the command line, you are prompted to type the selections interactively (a pipe can also be used to provide the selections in this case for most tools). While this works well for testing, it is easier to provide the selections from the command line if they are complex or for scripting.

Each tool has different command-line arguments for specifying selections (see the help for the individual tools). You can either pass a single string containing all selections (separated by semicolons), or multiple strings, each containing one selection. Note that you need to quote the selections to protect them from the shell.

If you set a selection command-line argument, but do not provide any selections, you are prompted to type the selections for that argument interactively. This is useful if that selection argument is optional, in which case it is not normally prompted for.

To provide selections from a file, use `-sf file.dat` in the place of the selection for a selection argument (e.g., `-select -sf file.dat`). In general, the `-sf` argument reads selections from the provided file and assigns them to selection arguments that have been specified up to that point, but for which no selections have been provided. As a special case, `-sf` provided on its own, without preceding selection arguments, assigns the selections to all (yet unset) required selections (i.e., those that would be prompted interactively if no selections are provided on the command line).

To use groups from a traditional index file, use argument `-n` to provide a file. See the "syntax" subtopic for how to use them. If this option is not provided, default groups are generated. The default groups are generated with the same logic as for non-selection tools.

Depending on the tool, two additional command-line arguments may be available to control the behavior:

- `-seltype` can be used to specify the default type of positions to calculate for each selection.
- `-selrpos` can be used to specify the default type of positions used in selecting atoms by coordinates.

See the "positions" subtopic for more information on these options.

Tools that take selections apply them to a structure/topology and/or a trajectory file. If the tool takes both (typically as `-s` for structure/topology and `-f` for trajectory), then the trajectory file is only used

for coordinate information, and all other information, such as atom names and residue information, is read from the structure/topology file. If the tool only takes a structure file, or if only that input parameter is provided, then also the coordinates are taken from that file. For example, to select atoms from a `.pdb`/`.gro` file in a tool that provides both options, pass it as `-s` (only). There is no warning if the trajectory file specifies, e.g., different atom names than the structure file. Only the number of atoms is checked. Many selection-enabled tools also provide an `-fgroup` option to specify the atom indices that are present in the trajectory for cases where the trajectory only has a subset of atoms from the topology/structure file.

### Selection syntax

A set of selections consists of one or more selections, separated by semicolons. Each selection defines a set of positions for the analysis. Each selection can also be preceded by a string that gives a name for the selection for use in, e.g., graph legends. If no name is provided, the string used for the selection is used automatically as the name.

For interactive input, the syntax is slightly altered: line breaks can also be used to separate selections. followed by a line break can be used to continue a line if necessary. Notice that the above only applies to real interactive input, not if you provide the selections, e.g., from a pipe.

It is possible to use variables to store selection expressions. A variable is defined with the following syntax:

```
VARNAME = EXPR ;
```

where `EXPR` is any valid selection expression. After this, `VARNAME` can be used anywhere where `EXPR` would be valid.

Selections are composed of three main types of expressions, those that define atoms (`ATOM_EXPR`), those that define positions (`POS_EXPR`), and those that evaluate to numeric values (`NUM_EXPR`). Each selection should be a `POS_EXPR` or a `ATOM_EXPR` (the latter is automatically converted to positions). The basic rules are as follows:

- An expression like `NUM_EXPR1 < NUM_EXPR2` evaluates to an `ATOM_EXPR` that selects all the atoms for which the comparison is true.
- Atom expressions can be combined with boolean operations such as `not ATOM_EXPR`, `ATOM_EXPR and ATOM_EXPR`, or `ATOM_EXPR or ATOM_EXPR`. Parentheses can be used to alter the evaluation order.
- `ATOM_EXPR` expressions can be converted into `POS_EXPR` expressions in various ways, see the "positions" subtopic for more details.
- `POS_EXPR` can be converted into `NUM_EXPR` using syntax like "`x of POS_EXPR`". Currently, this is only supported for single positions like in expression "`x of cog of ATOM_EXPR`".

Some keywords select atoms based on string values such as the atom name. For these keywords, it is possible to use wildcards (`name "C*"`) or regular expressions (e.g., `resname "R[AB]"`). The match type is automatically guessed from the string: if it contains other characters than letters, numbers, '*', or '?', it is interpreted as a regular expression. To force the matching to use literal string matching, use `name = "C*"` to match a literal C*. To force other type of matching, use '?' or '~' in place of '=' to force wildcard or regular expression matching, respectively.

Strings that contain non-alphanumeric characters should be enclosed in double quotes as in the examples. For other strings, the quotes are optional, but if the value conflicts with a reserved keyword, a syntax

error will occur. If your strings contain uppercase letters, this should not happen.

Index groups provided with the `-n` command-line option or generated by default can be accessed with `group NR` or `group NAME`, where `NR` is a zero-based index of the group and `NAME` is part of the name of the desired group. The keyword `group` is optional if the whole selection is provided from an index group. To see a list of available groups in the interactive mode, press enter in the beginning of a line.

### Specifying positions in selections

Possible ways of specifying positions in selections are:

1. A constant position can be defined as `[XX, YY, ZZ]`, where `XX`, `YY` and `ZZ` are real numbers.
2. `com of ATOM_EXPR [pbc]` or `cog of ATOM_EXPR [pbc]` calculate the center of mass/geometry of `ATOM_EXPR`. If `pbc` is specified, the center is calculated iteratively to try to deal with cases where `ATOM_EXPR` wraps around periodic boundary conditions.
3. `POSTYPE of ATOM_EXPR` calculates the specified positions for the atoms in `ATOM_EXPR`. `POSTYPE` can be `atom`, `res_com`, `res_cog`, `mol_com` or `mol_cog`, with an optional prefix `whole_` `part_` or `dyn_`. `whole_` calculates the centers for the whole residue/molecule, even if only part of it is selected. `part_` prefix calculates the centers for the selected atoms, but uses always the same atoms for the same residue/molecule. The used atoms are determined from the largest group allowed by the selection. `dyn_` calculates the centers strictly only for the selected atoms. If no prefix is specified, whole selections default to `part_` and other places default to `whole_`. The latter is often desirable to select the same molecules in different tools, while the first is a compromise between speed (`dyn_` positions can be slower to evaluate than `part_`) and intuitive behavior.
4. `ATOM_EXPR`, when given for whole selections, is handled as 3. above, using the position type from the command-line argument `-seltype`.

Selection keywords that select atoms based on their positions, such as `dist from`, use by default the positions defined by the `-selrpos` command-line option. This can be overridden by prepending a `POSTYPE` specifier to the keyword. For example, `res_com dist from POS` evaluates the residue center of mass distances. In the example, all atoms of a residue are either selected or not, based on the single distance calculated.

### Arithmetic expressions in selections

Basic arithmetic evaluation is supported for numeric expressions. Supported operations are addition, subtraction, negation, multiplication, division, and exponentiation (using ^). Result of a division by zero or other illegal operations is undefined.

### Selection keywords

The following selection keywords are currently available. For keywords marked with a plus, additional help is available through a subtopic KEYWORD, where KEYWORD is the name of the keyword.

- Keywords that select atoms by an integer property:

```
atomnr
mol (synonym for molindex)
molecule (synonym for molindex)
molindex
resid (synonym for resnr)
residue (synonym for resindex)
resindex
resnr
```

(use in expressions or like "atomnr 1 to 5 7 9")

- Keywords that select atoms by a numeric property:

```
beta (synonym for betafactor)
betafactor
charge
distance from POS [cutoff REAL]
distance from POS [cutoff REAL]
mass
mindistance from POS_EXPR [cutoff REAL]
mindistance from POS_EXPR [cutoff REAL]
occupancy
x
y
z
```

(use in expressions or like "occupancy 0.5 to 1")

- Keywords that select atoms by a string property:

```
altloc
atomname
atomtype
chain
insertcode
name (synonym for atomname)
pdbatomname
pdbname (synonym for pdbatomname)
```

(continues on next page)

(continued from previous page)

```
resname
type (synonym for atomtype)
```

(use like "name PATTERN [PATTERN] ...")

- Additional keywords that directly select atoms:

```
all
insolidangle center POS span POS_EXPR [cutoff REAL]
none
same KEYWORD as ATOM_EXPR
within REAL of POS_EXPR
```

- Keywords that directly evaluate to positions:

```
cog of ATOM_EXPR [pbc]
com of ATOM_EXPR [pbc]
```

(see also "positions" subtopic)

- Additional keywords:

```
merge POSEXPR
POSEXPR permute P1 ... PN
plus POSEXPR
```

### Selecting atoms by name - atomname, name, pdbatomname, pdbname

```
name
pdbname
atomname
pdbatomname
```

These keywords select atoms by name. `name` selects atoms using the GROMACS atom naming convention. For input formats other than PDB, the atom names are matched exactly as they appear in the input file. For PDB files, 4 character atom names that start with a digit are matched after moving the digit to the end (e.g., to match 3HG2 from a PDB file, use `name HG23`). `pdbname` can only be used with a PDB input file, and selects atoms based on the exact name given in the input file, without the transformation described above.

`atomname` and `pdbatomname` are synonyms for the above two keywords.

### Selecting based on distance - dist, distance, mindist, mindistance, within

```
distance from POS [cutoff REAL]
mindistance from POS_EXPR [cutoff REAL]
within REAL of POS_EXPR
```

`distance` and `mindistance` calculate the distance from the given position(s), the only difference being in that `distance` only accepts a single position, while any number of positions can be given for `mindistance`, which then calculates the distance to the closest position. `within` directly selects atoms that are within `REAL` of `POS_EXPR`.

For the first two keywords, it is possible to specify a cutoff to speed up the evaluation: all distances above the specified cutoff are returned as equal to the cutoff.

### Selecting atoms in a solid angle - insolidangle

```
insolidangle center POS span POS_EXPR [cutoff REAL]
```

This keyword selects atoms that are within `REAL` degrees (default=5) of any position in `POS_EXPR` as seen from `POS` a position expression that evaluates to a single position), i.e., atoms in the solid angle spanned by the positions in `POS_EXPR` and centered at `POS`.

Technically, the solid angle is constructed as a union of small cones whose tip is at `POS` and the axis goes through a point in `POS_EXPR`. There is such a cone for each position in `POS_EXPR`, and point is in the solid angle if it lies within any of these cones. The cutoff determines the width of the cones.

### Merging selections - merge, plus

```
POSEXPR merge POSEXPR [stride INT]
POSEXPR merge POSEXPR [merge POSEXPR ...]
POSEXPR plus POSEXPR [plus POSEXPR ...]
```

Basic selection keywords can only create selections where each atom occurs at most once. The `merge` and `plus` selection keywords can be used to work around this limitation. Both create a selection that contains the positions from all the given position expressions, even if they contain duplicates. The difference between the two is that `merge` expects two or more selections with the same number of positions, and the output contains the input positions selected from each expression in turn, i.e., the output is like A1 B1 A2 B2 and so on. It is also possible to merge selections of unequal size as long as the size of the first is a multiple of the second one. The `stride` parameter can be used to explicitly provide this multiplicity. `plus` simply concatenates the positions after each other, and can work also with selections of different sizes. These keywords are valid only at the selection level, not in any subexpressions.

### Permuting selections - permute

```
permute P1 ... PN
```

By default, all selections are evaluated such that the atom indices are returned in ascending order. This can be changed by appending `permute P1 P2 ... PN` to an expression. The `Pi` should form a permutation of the numbers 1 to N. This keyword permutes each N-position block in the selection such that the i'th position in the block becomes Pi'th. Note that it is the positions that are permuted, not individual atoms. A fatal error occurs if the size of the selection is not a multiple of n. It is only possible to permute the whole selection expression, not any subexpressions, i.e., the `permute` keyword should appear last in a selection.

### Selecting atoms by residue number - resid, residue, resindex, resnr

```
resnr
resid
resindex
residue
```

`resnr` selects atoms using the residue numbering in the input file. `resid` is synonym for this keyword for VMD compatibility.

`resindex N` selects the `N` th residue starting from the beginning of the input file. This is useful for uniquely identifying residues if there are duplicate numbers in the input file (e.g., in multiple chains). `residue` is a synonym for `resindex`. This allows `same residue as` to work as expected.

### Extending selections - same

```
same KEYWORD as ATOM_EXPR
```

The keyword `same` can be used to select all atoms for which the given `KEYWORD` matches any of the atoms in `ATOM_EXPR`. Keywords that evaluate to integer or string values are supported.

### Selection evaluation and optimization

Boolean evaluation proceeds from left to right and is short-circuiting i.e., as soon as it is known whether an atom will be selected, the remaining expressions are not evaluated at all. This can be used to optimize the selections: you should write the most restrictive and/or the most inexpensive expressions first in boolean expressions. The relative ordering between dynamic and static expressions does not matter: all static expressions are evaluated only once, before the first frame, and the result becomes the leftmost expression.

Another point for optimization is in common subexpressions: they are not automatically recognized, but can be manually optimized by the use of variables. This can have a big impact on the performance of complex selections, in particular if you define several index groups like this:

```
rdist = distance from com of resnr 1 to 5;
resname RES and rdist < 2;
```

(continues on next page)

(continued from previous page)

```
resname RES and rdist < 4;
resname RES and rdist < 6;
```

Without the variable assignment, the distances would be evaluated three times, although they are exactly the same within each selection. Anything assigned into a variable becomes a common subexpression that is evaluated only once during a frame. Currently, in some cases the use of variables can actually lead to a small performance loss because of the checks necessary to determine for which atoms the expression has already been evaluated, but this should not be a major problem.

### Selection limitations

- Some analysis programs may require a special structure for the input selections (e.g., some options of `gmx gangle` require the index group to be made of groups of three or four atoms). For such programs, it is up to the user to provide a proper selection expression that always returns such positions.
- All selection keywords select atoms in increasing order, i.e., you can consider them as set operations that in the end return the atoms in sorted numerical order. For example, the following selections select the same atoms in the same order:

  ```
  resname RA RB RC
  resname RB RC RA
  ```

  ```
  atomnr 10 11 12 13
  atomnr 12 13 10 11
  atomnr 10 to 13
  atomnr 13 to 10
  ```

  If you need atoms/positions in a different order, you can:

  - use external index groups (for some static selections),
  - use the `permute` keyword to change the final order, or
  - use the `merge` or `plus` keywords to compose the final selection from multiple distinct selections.
- Due to technical reasons, having a negative value as the first value in expressions like

  ```
  charge -1 to -0.7
  ```

  result in a syntax error. A workaround is to write

  ```
  charge {-1 to -0.7}
  ```

  instead.
- When `name` selection keyword is used together with PDB input files, the behavior may be unintuitive. When GROMACS reads in a PDB file, 4 character atom names that start with a digit are transformed such that, e.g., 1HG2 becomes HG21, and the latter is what is matched by the `name` keyword. Use `pdbname` to match the atom name as it appears in the input PDB file.

### Selection examples

Below, examples of different types of selections are given.

- Selection of all water oxygens:

```
resname SOL and name OW
```

- Centers of mass of residues 1 to 5 and 10:

```
res_com of resnr 1 to 5 10
```

- All atoms farther than 1 nm of a fixed position:

```
not within 1 of [1.2, 3.1, 2.4]
```

- All atoms of a residue LIG within 0.5 nm of a protein (with a custom name):

```
"Close to protein" resname LIG and within 0.5 of group "Protein"
```

- All protein residues that have at least one atom within 0.5 nm of a residue LIG:

```
group "Protein" and same residue as within 0.5 of resname LIG
```

- All RES residues whose COM is between 2 and 4 nm from the COM of all of them:

```
rdist = res_com distance from com of resname RES
resname RES and rdist >= 2 and rdist <= 4
```

- Selection like with duplicate atoms like C1 C2 C2 C3 C3 C4 ... C8 C9:

```
name "C[1-8]" merge name "C[2-9]"
```

  This can be used with `gmx distance` to compute C1-C2, C2-C3 etc. distances.

- Selection with atoms in order C2 C1:

```
name C1 C2 permute 2 1
```

  This can be used with `gmx gangle` to get C2->C1 vectors instead of C1->C2.

- Selection with COMs of two index groups:

```
com of group 1 plus com of group 2
```

  This can be used with `gmx distance` to compute the distance between these two COMs.

- Fixed vector along x (can be used as a reference with `gmx gangle`):

```
[0, 0, 0] plus [1, 0, 0]
```

- The following examples explain the difference between the various position types. This selection selects a position for each residue where any of the three atoms C[123] has `x < 2`. The positions are computed as the COM of all three atoms. This is the default behavior if you just write `res_com of`.

```
part_res_com of name C1 C2 C3 and x < 2
```

This selection does the same, but the positions are computed as COM positions of whole residues:

```
whole_res_com of name C1 C2 C3 and x < 2
```

Finally, this selection selects the same residues, but the positions are computed as COM of exactly those atoms atoms that match the `x < 2` criterion:

```
dyn_res_com of name C1 C2 C3 and x < 2
```

- Without the `of` keyword, the default behavior is different from above, but otherwise the rules are the same:

```
name C1 C2 C3 and res_com x < 2
```

works as if `whole_res_com` was specified, and selects the three atoms from residues whose COM satisfiex `x < 2`. Using

```
name C1 C2 C3 and part_res_com x < 2
```

instead selects residues based on the COM computed from the C[123] atoms.

### 3.11.107 Command changes between versions

Starting from GROMACS 5.0, some of the analysis commands (and a few other commands as well) have changed significantly.

One main driver for this has been that many new tools mentioned below now accept selections through one or more command-line options instead of prompting for a static index group. To take full advantage of selections, the interface to the commands has changed somewhat, and some previous command-line options are no longer present as the same effect can be achieved with suitable selections. Please see *Selection syntax and usage* (page 371) additional information on how to use selections.

In the process, some old analysis commands have been removed in favor of more powerful functionality that is available through an alternative tool. For removed or replaced commands, this page documents how to perform the same tasks with new tools. For new commands, a brief note on the available features is given. See the linked help for the new commands for a full description.

This section lists only major changes; minor changes like additional/removed options or bug fixes are not typically included.

For more information about changed features, please check out the *Release notes* (page 838).

## Version 2020

### gmx convert-trj

**new**

*gmx convert-trj* (page 179) has been introduced as a selection-enabled alternative for exchanging trajectory file format (previously done in *gmx trjconv* (page 334)).

### gmx extract-cluster

**new**

*gmx extract-cluster* (page 218) has been introduced as a selection-enabled way to write sub-trajectories based on the output from a cluster analysis. The corresponding option **-sub** in *gmx trjconv* (page 334) has been removed.

## Version 2018

### gmx trajectory

**new**

*gmx trajectory* (page 331) has been introduced as a selection-enabled version of *gmx traj* (page 328). It supports output of coordinates, velocities, and/or forces for positions calculated for selections.

## Version 2016

### Analysis on arbitrary subsets of atoms

Tools implemented in the new analysis framework can now operate upon trajectories that match only a subset of the atoms in the input structure file.

### gmx insert-molecules

**improved**

*gmx insert-molecules* (page 248) has gained an option `-replace` that makes it possible to insert molecules into a solvated configuration, replacing any overlapping solvent atoms. In a fully solvated box, it is also possible to insert into a certain region of the solvent only by selecting a subset of the solvent atoms (`-replace` takes a selection that can also contain expressions like `not within 1 of ...`).

### gmx rdf

#### improved

The normalization for the output RDF can now also be the radial number density.

### gmx genconf

#### simplified

Removed `-block`, `-sort` and `-shuffle`.

## Version 5.1

## General

Symbolic links from 5.0 are no longer supported. The only way to invoke a command is through `gmx <command>`.

### gmx pairdist

#### new

*gmx pairdist* (page 279) has been introduced as a selection-enabled replacement for *gmx mindist* (page 264) (`gmx mindist` still exists unchanged). It can calculate min/max pairwise distances between a pair of selections, including, e.g., per-residue minimum distances or distances from a single point to a set of residue-centers-of-mass.

### gmx rdf

#### rewritten

*gmx rdf* (page 291) has been rewritten for 5.1 to use selections for specifying the points from which the RDFs are calculated. The interface is mostly the same, except that there are new command-line options to specify the selections. The following additional changes have been made:

- `-com` and `-rdf` options have been removed. Equivalent functionality is available through selections:
  - `-com` can be replaced with a `com of <selection>` as the reference selection.
  - `-rdf` can be replaced with a suitable set of selections (e.g., `res_com of <selection>`) and/or using `-seltype`.
- `-rmax` option is added to specify a cutoff for the RDFs. If set to a value that is significantly smaller than half the box size, it can speed up the calculation significantly if a grid-based neighborhood search can be used.
- `-hq` and `-fade` options have been removed, as they are simply postprocessing steps on the raw numbers that can be easily done after the analysis.

## Version 5.0

### General

Version 5.0 introduced the **gmx** wrapper binary. For backwards compatibility, this version still creates symbolic links by default for old tools: e.g., `g_order <options>` is equivalent to `gmx order <options>`, and `g_order` is simply a symbolic link on the file system.

### g_bond

#### replaced

This tool has been removed in 5.0. A replacement is *gmx distance* (page 198).

You can provide your existing index file to *gmx distance* (page 198), and it will calculate the same distances. The differences are:

- `-blen` and `-tol` options have different default values.
- You can control the output histogram with `-binw`.
- `-aver` and `-averdist` options are not present. Instead, you can choose between the different things to calculate using `-oav` (corresponds to `-d` with `-averdist`), `-oall` (corresponds to `-d` without `-averdist`), `-oh` (corresponds to `-o` with `-aver`), and `-oallstat` (corresponds to `-l` without `-aver`).

You can produce any combination of output files. Compared to `g_bond`, `gmx distance -oall` is currently missing labels for the output columns.

### g_dist

#### replaced

This tool has been removed in 5.0. A replacement is *gmx distance* (page 198) (for most options) or *gmx select* (page 312) (for `-dist` or `-lt`).

If you had index groups A and B in `index.ndx` for `g_dist`, you can use the following command to compute the same distance with `gmx distance`:

```
gmx distance -n index.ndx -select 'com of group "A" plus com of␣
↪group "B"' -oxyz -oall
```

The `-intra` switch is replaced with `-nopbc`.

If you used `-dist D`, you can do the same calculation with `gmx select`:

```
gmx select -n index.ndx -select 'group "B" and within D of com of␣
↪group "A"' -on/-oi/-os/-olt
```

You can select the output option that best suits your post-processing needs (`-olt` is a replacement for `g_dist -dist -lt`)

### gmx distance

#### new

*gmx distance* (page 198) has been introduced as a selection-enabled replacement for various tools that computed distances between fixed pairs of atoms (or centers-of-mass of groups). It has a combination of the features of `g_bond` and `g_dist`, allowing computation of one or multiple distances, either between atom-atom pairs or centers-of-mass of groups, and providing a combination of output options that were available in one of the tools.

### gmx gangle

#### new

*gmx gangle* (page 223) has been introduced as a selection-enabled replacement for `g_sgangle`. In addition to supporting atom-atom vectors, centers-of-mass can be used as endpoints of the vectors, and there are a few additional angle types that can be calculated. The command also has basic support for calculating normal angles between three atoms and/or centers-of-mass, making it a partial replacement for *gmx angle* (page 158) as well.

### gmx protonate

#### replaced

This was a very old tool originally written for united atom force fields, where it was necessary to generate all hydrogens after running a trajectory in order to calculate e.g. distance restraint violations. The functionality to simply protonate a structure is available in *gmx pdb2gmx* (page 281). If there is significant interest, we might reintroduce it after moving to new topology formats in the future.

### gmx freevolume

#### new

This tool has been introduced in 5.0. It uses a Monte Carlo sampling method to calculate the fraction of free volume within the box (using a probe of a given size).

### g_sas

#### rewritten

This tool has been rewritten in 5.0, and renamed to *gmx sasa* (page 307) (the underlying surface area calculation algorithm is still the same).

The main difference in the new tool is support for selections. Instead of prompting for an index group, a (potentially dynamic) selection for the calculation can be given with `-surface`. Any number of output groups can be given with `-output`, allowing multiple parts of the surface area to be computed in a single run. The total area of the `-surface` group is now always calculated.

The tool no longer automatically divides the surface into hydrophobic and hydrophilic areas, and there is no `-f_index` option. The same effects can be obtained by defining suitable selections for `-output`.

If you want output that contains the same numbers as with the old tool for a calculation group `A` and output group `B`, you can use

```
gmx sasa -surface 'group "A"' -output '"Hydrophobic" group "A" and␣
↪charge {-0.2 to 0.2}; "Hydrophilic" group "B" and not charge {-0.
↪2 to 0.2}; "Total" group "B"'
```

Solvation free energy estimates are now calculated only if separately requested with `-odg`, and are written into a separate file.

Output option `-i` for a position restraint file is not currently implemented in the new tool, but would not be very difficult to add if requested.

### g_sgangle

#### replaced

This tool has been removed in 5.0. A replacement is *gmx gangle* (page 223) (for angle calculation) and *gmx distance* (page 198) (for `-od`, `-od1`, `-od2`).

If you had index groups A and B in index.ndx for `g_sgangle`, you can use the following command to compute the same angle with `gmx gangle`:

```
gmx gangle -n index.ndx -g1 vector/plane -group1 'group "A"' -g2␣
↪vector/plane -group2 'group "B"' -oav
```

You need to select either `vector` or `plane` for the `-g1` and `-g2` options depending on which one your index groups specify.

If you only had a single index group A in index.ndx and you used `g_sgangle -z` or `-one`, you can use:

```
gmx gangle -n index.ndx -g1 vector/plane -group1 'group "A"' -g2 z/
↪t0 -oav
```

For the distances, you can use *gmx distance* (page 198) to compute one or more distances as you want. Both distances between centers of groups or individual atoms are supported using the new selection syntax.

### genbox

This tool has been split to *gmx solvate* (page 319) and *gmx insert-molecules* (page 248).

#### tpbconv

This tool has been renamed *gmx convert-tpr* (page 178).

## 3.12 Terminology

### 3.12.1 Pressure

The pressure in molecular dynamics can be computed from the kinetic energy and the virial.

#### Fluctuation

Whether or not pressure coupling is used within a simulation, the pressure value for the simulation box will oscillate significantly. Instantaneous pressure is meaningless, and not well-defined. Over a picosecond time scale it usually will not be a good indicator of the true pressure. This variation is entirely normal due to the fact that pressure is a macroscopic property and can only be measured properly as time average, while it is being measured and/or adjusted with pressure coupling on the microscopic scale. How much it varies and the speed at which it does depends on the number of atoms in the system, the type of pressure coupling used and the value of the coupling constants. Fluctuations of the order of hundreds of bar are typical. For a box of 216 waters, fluctuations of 500-600 bar are standard. Since the fluctuations go down with the square root of the number of particles, a system of 21600 water molecules (100 times larger) will still have pressure fluctuations of 50-60 bar.

### 3.12.2 Periodic boundary conditions

Periodic boundary conditions (PBC) are used in molecular dynamics simulations to avoid problems with boundary effects caused by finite size, and make the system more like an infinite one, at the cost of possible periodicity effects.

Beginners visualizing a trajectory sometimes think they are observing a problem when

- the molecule(s) does not stay in the centre of the box, or
- it appears that (parts of) the molecule(s) diffuse out of the box, or
- holes are created, or
- broken molecules appear, or
- their unit cell was a rhombic dodecahedron or cubic octahedron but it looks like a slanted cube after the simulation, or
- crazy bonds all across the simulation cell appear.

This is not a problem or error that is occurring, it is what you should expect.

The existence of PBC means that any atom that leaves a simulation box by, say, the right-hand face, then enters the simulation box by the left-hand face. In the example of a large protein, if you look at the face of the simulation box that is opposite to the one from which the protein is protruding, then a hole in the solvent will be visible. The reason why the molecule(s) move from where they were initially located within the box is that they are (for the vast majority of simulations) free to diffuse around. And so they do. They are not held in a magic location of the box. The box is not centered around anything while performing the simulation. Molecules are not made whole as a matter of course. Moreover, any periodic

cell shape can be expressed as a parallelepiped (a.k.a. triclinic cell), and GROMACS does so internally regardless of the initial shape of the box.

These visual issues can be fixed after the conclusion of the simulation by judicious use of the optional inputs to *gmx trjconv* (page 334) to process the trajectory files. Similarly, analyses such as RMSD of atomic positions can be flawed when a reference structure is compared with a structure that needs adjusting for periodicity effects, and the solution with *gmx trjconv* (page 334) follows the same lines. Some complex cases needing more than one operation will require more than one invocation of *gmx trjconv* (page 334) in order to work.

For further information, see the corresponding section in the *Reference Manual* (page 429).

#### Suggested workflow

Fixing periodicity effects with *gmx trjconv* (page 334) to suit visualization or analysis can be tricky. Multiple invocations can be necessary. You may need to create custom index groups (e.g. to keep your ligand with your protein) Following the steps below in order (omitting those not required) should help get a pleasant result. You will need to consult `gmx trjconv -h` to find out the details for each step. That is deliberate – there is no magic "do what I want" recipe. You have to decide what you want, first. :-)

1. First make your molecules whole if you want them whole.
2. Cluster your molecules/particles if you want them clustered.
3. If you want jumps removed, extract the first frame from the trajectory to use as the reference, and then use `-pbc nojump` with that first frame as reference.
4. Center your system using some criterion. Doing so shifts the system, so do not use `-pbc nojump` after this step.
5. Perhaps put everything in some box with the other `-pbc` or `-ur` options.
6. Fit the resulting trajectory to some (other) reference structure (if desired), and do not use any PBC related option afterwards.

With point three, the issue is that *gmx trjconv* (page 334) removes the jumps from the first frame using the reference structure provided with -s. If the reference structure (run input file) is not clustered/whole, using `-pbc nojump` will undo steps 1 and 2.

### 3.12.3 Thermostats

Thermostats are designed to help a simulation sample from the correct ensemble (i.e. NVT or NPT) by modulating the temperature of the system in some fashion. First, we need to establish what we mean by temperature. In simulations, the "instantaneous (kinetic) temperature" is usually computed from the kinetic energy of the system using the equipartition theorem. In other words, the temperature is computed from the system's total kinetic energy.

So, what is the goal of a thermostat? Actually, it turns out the goal is not to keep the temperature constant, as that would mean fixing the total kinetic energy, which would be silly and not the aim of NVT or NPT. Rather, it is to ensure that the average temperature of a system be correct.

To see why this is the case, imagine a glass of water sitting in a room. Suppose you can look very closely at a few molecules in some small region of the glass, and measure their kinetic energies. You would not expect the kinetic energy of this small number of particles to remain precisely constant; rather, you

would expect fluctuations in the kinetic energy due to the small number of particles. As you average over larger and larger numbers of particles, the fluctuations in the average get smaller and smaller, so finally by the time you look at the whole glass, you say it has "constant temperature".

Molecular dynamics simulations are often fairly small compared to a glass of water, so we have bigger fluctuations. So it is really more appropriate here to think of the role of a thermostat as ensuring that we have

(a) the correct average temperature, and

(b) the fluctuations of the correct size.

See the relevant section in the *Reference Manual* (page 447) for details on how temperature coupling is applied and the types currently available.

### What to do

Some hints on practices that generally are a good idea:

- Preferably, use a thermostat that samples the correct distribution of temperatures (for examples, see the corresponding manual section), in addition to giving you the correct average temperature.
- At least: use a thermostat that gives you the correct average temperature, and apply it to components of your system for which they are justified (see the first bullet in *What not to do* (page 388)). In some cases, using `tc-grps = System` may lead to the "hot solvent/cold solute" problem described in the 3rd reference in *Further reading* (page 388).

### What not to do

Some hints on practices that generally not a good idea to use:

- Do not use separate thermostats for every component of your system. Some molecular dynamics thermostats only work well in the thermodynamic limit. A group must be of sufficient size to justify its own thermostat. If you use one thermostat for, say, a small molecule, another for protein, and another for water, you are likely introducing errors and artifacts that are hard to predict. In particular, do not couple ions in aqueous solvent in a separate group from that solvent. For a protein simulation, using `tc-grps = Protein Non-Protein` is usually best.
- Do not use thermostats that work well only in the limit of a large number of degrees of freedom for systems with few degrees of freedom. For example, do not use Nosé-Hoover or Berendsen thermostats for types of free energy calculations where you will have a component of the system with very few degrees of freedom in an end state (i.e. a noninteracting small molecule).

### Further reading

1. Cheng, A. & Merz, K. M. Application of the Nosé-Hoover chain algorithm to the study of protein dynamics. *J. Phys. Chem.* **100** (5), 1927–1937 (1996).
2. Mor, A., Ziv, G. & Levy, Y. Simulations of proteins with inhomogeneous degrees of freedom: the effect of thermostats. *J. Comput. Chem.* **29** (12), 1992–1998 (2008).
3. Lingenheil, M., Denschlag, R., Reichold, R. & Tavan, P. The "hot-solvent/cold-solute" problem revisited. *J. Chem. Theory Comput.* **4** (8), 1293–1306 (2008).

### 3.12.4 Energy conservation

In principle, a molecular dynamics simulation should conserve the total energy, the total momentum and (in a non-periodic system) the total angular momentum. A number of algorithmic and numerical issues make that this is not always the case:

- Cut-off treatment and/or long-range electrostatics treatment (see Van Der Spoel, D. & van Maaren, P. J. The origin of layer structure artifacts in simulations of liquid water. *J. Chem. Theor. Comp.* **2**, 1–11 (2006).)
- Treatment of pair lists,
- Constraint algorithms (see e.g. Hess, B. P-LINCS: A parallel linear constraint solver for molecular simulation. *J. Chem. Theor. Comp.* **4**, 116–122 (2008).).
- The integration timestep.
- *Temperature coupling* (page 387) and *pressure coupling* (page 386).
- Round-off error (in particular in single precision), for example subtracting large numbers (Lippert, R. A. et al. A common, avoidable source of error in molecular dynamics integrators. *J. Chem. Phys.* **126**, 046101 (2007).).
- The choice of the integration algorithm (in GROMACS this is normally leap-frog).
- Removal of center of mass motion: when doing this in more than one group the conservation of energy will be violated.

### 3.12.5 Average structure

Various GROMACS utilities can compute average structures. Presumably the idea for this comes from something like an ensemble-average NMR structure. In some cases, it makes sense to calculate an average structure (as a step on the way to calculating root-mean-squared fluctuations (RMSF), for example, one needs the average position of all of the atoms).

However, it is important to remember that an average structure is not necessarily meaningful. By way of analogy, suppose I alternate holding a ball in my left hand, then in my right hand. What is the average position of the ball? Halfway in between – even though I always have it either in my left hand or my right hand. Similarly, for structures, averages will tend to be meaningless anytime there are separate metastable conformational states. This can happen on a sidechain level, or for some regions of backbone, or even whole helices or components of the secondary structure.

Thus, if you derive an average structure from a molecular dynamics simulation, and find artifacts like unphysical bond lengths, weird structures, etc., this does not necessarily mean something is wrong. It just shows the above: an average structure from a simulation is not necessarily a physically meaningful structure.

### 3.12.6 Blowing up

*Blowing up* is a highly technical term used to describe a common sort of simulation failure. In brief, it describes a failure typically due to an unacceptably large force that ends up resulting in a failure of the integrator.

To give a bit more background, it is important to remember that molecular dynamics numerically integrates Newton's equations of motion by taking small, discrete timesteps, and using these timesteps to determine new velocities and positions from velocities, positions, and forces at the previous timestep. If forces become too large at one timestep, this can result in extremely large changes in velocity/position when going to the next timestep. Typically, this will result in a cascade of errors: one atom experiences a very large force one timestep, and thus goes shooting across the system in an uncontrolled way in the next timestep, overshooting its preferred location or landing on top of another atom or something similar. This then results in even larger forces the next timestep, more uncontrolled motions, and so on. Ultimately, this will cause the simulation package to crash in some way, since it cannot cope with such situations. In simulations with constraints, the first symptom of this will usually be some LINCS or SHAKE warning or error – not because the constraints are the source of the problem, but just because they are the first thing to crash. Likewise for warnings about tabulated or 1-4 interactions being outside the distance supported by the table. This can happen on one computer system while another resulted in a stable simulation because of the impossibility of numerical reproducibility of these calculations on different computer systems.

Possible causes include:

- you did not minimize well enough,
- you have a bad starting structure, perhaps with steric clashes,
- you are using too large a timestep (particularly given your choice of constraints),
- you are doing particle insertion in free energy calculations without using soft core,
- you are using inappropriate pressure coupling (e.g. when you are not in equilibrium, Berendsen can be good while relaxing the volume, but you will need to switch to a more accurate pressure-coupling algorithm later),
- you are using inappropriate temperature coupling, perhaps on inappropriate groups, or
- your position restraints are to coordinates too different from those present in the system, or
- you have a single water molecule somewhere within the system that is isolated from the other water molecules, or
- you are experiencing a bug in *gmx mdrun* (page 258).

Because blowing up is due, typically, to forces that are too large for a particular timestep size, there are a couple of basic solutions:

- make sure the forces do not get that large, or
- use a smaller timestep.

Better system preparation is a way to make sure that forces do not get large, if the problems are occurring near the beginning of a simulation.

### 3.12.7 Diagnosing an unstable system

Troubleshooting a system that is blowing up can be challenging, especially for an inexperienced user. Here are a few general tips that one may find useful when addressing such a scenario:

1. If the crash is happening relatively early (within a few steps), set `nstxout` (or `nstxout-compressed`) to 1, capturing all possible frames. Watch the resulting trajectory to see which atoms/residues/molecules become unstable first.
2. Simplify the problem to try to establish a cause:
   - If you have a new box of solvent, try minimizing and simulating a single molecule to see if the instability is due to some inherent problem with the molecule's topology or if instead there are clashes in your starting configuration.
   - If you have a protein-ligand system, try simulating the protein alone in the desired solvent. If it is stable, simulate the ligand in vacuo to see if its topology gives stable configurations, energies, etc.
   - Remove the use of fancy algorithms, particularly if you have not equilibrated thoroughly first.
3. Monitor various components of the system's energy using *gmx energy* (page 214). If an intramolecular term is spiking, that may indicate improper bonded parameters, for example.
4. Make sure you have not been ignoring error messages (missing atoms when running *gmx pdb2gmx* (page 281), mismatching names when running *gmx grompp* (page 230), etc.) or using workarounds (like using `gmx grompp -maxwarn` when you should not be) to make sure your topology is intact and being interpreted correctly.
5. Make sure you are using appropriate settings in your *mdp* (page 579) file for the force field you have chosen and the type of system you have. Particularly important settings are treatment of cutoffs, proper neighbor searching interval (`nstlist`), and temperature coupling. Improper settings can lead to a breakdown in the model physics, even if the starting configuration of the system is reasonable.

When using no explict solvent, starting your equilibration with a smaller time step than your production run can help energy equipartition more stably.

There are several common situations in which instability frequently arises, usually in the introduction of new species (ligands or other molecules) into the system. To determine the source of the problem, simplify the system (e.g. the case of a protein-ligand complex) in the following way.

1. Does the protein (in water) minimize adequately by itself? This is a test of the integrity of the coordinates and system preparation. If this fails, something probably went wrong when running *gmx pdb2gmx* (page 281) (see below), or maybe *gmx genion* (page 227) placed an ion very close to the protein (it is random, after all).
2. Does the ligand minimize in vacuo? This is a test of the topology. If it does not, check your parameterization of the ligand and any implementation of new parameters in force field files.
3. (If previous item is successful) Does the ligand minimize in water, and/or does a short simulation of the ligand in water succeed?

Other sources of possible problems are in the biomolecule topology itself.

1. Did you use `-missing` when running *gmx pdb2gmx* (page 281)? If so, do not. Reconstruct missing coordinates rather than ignoring them.

2. Did you override long/short bond warnings by changing the lengths? If so, do not. You probably have missing atoms or some terrible input geometry.

### 3.12.8 Molecular dynamics

Molecular dynamics (MD) is computer simulation with atoms and/or molecules interacting using some basic laws of physics. The GROMACS *Reference Manual* (page 433) provides a good general introduction to this area, as well as specific material for use with GROMACS. The first few chapters are mandatory reading for anybody wishing to use GROMACS and not waste time.

- Introduction to molecular modeling (slides, video) - theoretical framework, modeling levels, limitations and possibilities, systems and methods (Erik Lindahl).

#### Books

There are several text books around.

Good introductory books are:

- A. Leach (2001) Molecular Modeling: Principles and Applications.
- T. Schlick (2002) Molecular Modeling and Simulation

With programming background:

- D. Rapaport (1996) The Art of Molecular Dynamics Simulation
- D. Frenkel, B. Smith (2001) Understanding Molecular Simulation

More from the physicist's view:

- M. Allen, D. Tildesley (1989) Computer simulation of liquids
- H.J.C. Berendsen (2007) Simulating the Physical World: Hierarchical Modeling from Quantum Mechanics to Fluid Dynamics

#### Types / Ensembles

- NVE - number of particles (N), system volume (V) and energy (E) are constant / conserved.
- NVT - number of particles (N), system volume (V) and temperature (T) are constant / conserved. (See *thermostats* (page 387) for more on *constant* temperature).
- NPT - number of particles (N), system pressure (P) and temperature (T) are constant / conserved. (See *pressure coupling* (page 386) for more on *constant* pressure).

### 3.12.9 Force field

Force fields are sets of potential functions and parametrized interactions that can be used to study physical systems. A general introduction to their history, function and use is beyond the scope of this guide, and the user is asked to consult either the relevant literature or try to start at the relevant Wikipedia page.

## 3.13 Validation

As with any code, both scientific or commercial, results produced by GROMACS can be incorrect. We provide no guarantees on the correctness of any output. With that said, because of extensive test coverage and its large user base, the results produced by GROMACS are very reliable. However, new code and features have not had as extensive testing as established parts of code and there is therefore a higher risk for bugs, which the user needs to be aware of. We mark such features with two different tags: "experimental" and "validation pending". *gmx mdrun* (page 258) will notify users in the log file and on the standard output when such features are in use.

Note that the responsibility of the GROMACS developers is only the correctness of unmodified GROMACS code distributed as part of an official release, not that of modified versions, external packages or plug-ins. An example is QM/MM simulations. If the results are incorrect because of an issue in the QM-package used, that is not something the GROMACS developers are responsible for. What the GROMACS developers are responsible for in this case is the correctness of the QM/MM interface (indeed there are tests that cover the QM/MM interface alone).

### 3.13.1 Experimental features

Features labeled "experimental" have not received sufficient testing. Such features are not suitable for production simulations. We try to keep code belonging to this category to a minimum. Such features are included in the main codebase primarily because it simplifies the development process. Experimental features always need to be activated using an environment variable and/or by activating a `cmake` configuration option. Users can try such features and are encouraged to provide feedback, in particular in case of issues.

The current features with experimental status are:

- CUDA graph code, activated by the `GMX_CUDA_GRAPH` environment variable
- NVSHMEM communication, enabled at build with `-DGMX_NVSHMEM=ON` CMake variable
- OneAPI graph support, enabled at build with `-DGMX_SYCL_ENABLE_GRAPHS=ON` CMake variable
- Parallel PME over multiple GPUs with the HIP GPU backend targeting AMD GPUs, enabled at build with `-DGMX_GPU=HIP` and `-DGMX_USE_HEFFTE=ON` CMake variables
- Direct halo communication, activated by the `GMX_FILLERS_IN_LOCAL_STATE` environment variable
- NBNxM 1x1 non-bonded kernels, activated by the `GMX_NBNXM_PLAINC_1X1` environment variable
- H5MD trajectory output, activated by selecting the `h5md` type for trajectory file output in mdrun
- Using wave64 execution on wave32 (RDNA) devices with HIP, activated by passing `-mwavefrontsize64` as an additional compilation flag.

- VkFFT support for evaluating 3D Fast Fourier Transforms (on non-AMD and non-Apple platforms), enabled at build time with `-DGMX_GPU_FFT_LIBRARY=vkFFT` CMake variable
- oneMath support for evaluating 3D Fast Fourier Transforms (on any platform), enabled at build time with `-DGMX_GPU_FFT_LIBRARY=oneMath` CMake variable
- Double-batched FFT library support for evaluating 3D Fast Fourier Transforms, enabled at build time with `-DGMX_GPU_FFT_LIBRARY=BBFFT` CMake variable

### 3.13.2 Features with validation pending

Ideally, we would have a validation test suite that covers all combinations of features of GROMACS as well as all possible CPU architectures and GPU backends. Currently we do not have this, but we are working on a collection of validation systems. Still it will be challenging to cover the combinatorial explosion of features and options, as well as all supported hardware. When a new feature or acceleration backend has successfully passed our internal testing and validation, it will in most cases enter in a GROMACS release in the category "validation pending". For the user this means that we expect the feature to give correct results, but there is a possibility of incorrect results with certain combinations of features and/or backends. Such features will never be active by default; the user will have to actively set an *gmx mdrun* (page 258) command line option or an environment variable, and in some cases an `mdp` option. We expect features to be fully validated over the course of one or two years. Users are encouraged to test features with validation pending to aid the validation process. Carefully check your results and please report any issues on the GROMACS user forum or by opening an issue on GitLab issues.

The current features with status validation pending are:

- The modular simulator with an integrator different from Velocity Verlet
- The Colvars interface, activated by the `colvars-active` `mdp` option
- The PLUMED interface, activated by the `-plumed` option of *gmx mdrun* (page 258)
- The neural network potential interface, activated by the `nnpot-active` `mdp` option and configuring with LibTorch
- The SYCL GPU backend for non-AMD and non-Intel GPU platforms, activated by choosing the `SYCL` option for `GMX_GPU` in `cmake`
- HIP GPU backend targeting AMD GPUs, activated by choosing the `HIP` option for `GMX_GPU` in `cmake`
- The Fast Multipole Method interface, enabled at build time by the `-DGMX_USE_EXT_FMM` CMake variable
- AMBER LEaP-compatible dihedral reordering by grompp, activated by the `_FF_AMBER_LEAP_ATOM_REORDERING` preprocessor define
- Non-bonded free-energy calculations on a GPU, activated by the `-nbfe gpu` mdrun option

### 3.13.3 Feature lifecycle stages

1. Development/Experimental: A new feature that may be incomplete, under active development, or not sufficiently tested.
   - Risk: High. May contain bugs, the API or behavior may change, and the feature could be removed without notice.
   - Usage recommendation: Not for production use. Use only for testing and feedback.
2. Validation pending: A feature that is code-complete and has passed initial tests, but has not been validated across the full matrix of scientific cases, hardware, and parallelization options.
   - Risk: Moderate. The feature is expected to be correct, but carries a risk of incorrect results exists in specific, untested scenarios.
   - Usage recommendation: Use with caution. Users are encouraged to test and aid in validation by carefully checking results and reporting any issues.
3. Stable: The default state for a feature, validated, and considered reliable for production simulations.
   - Risk: Low (within the bounds of the standard GROMACS disclaimer).
   - Usage recommendation: Recommended for all users.
4. Legacy: A feature that is functional but has not been extensively used or tested in recent releases, or it has been replaced by a newer alternative.
   - Risk: Low to moderate (for correctness), but it will not receive updates and may be less performant or flexible than alternatives.
   - Usage recommendation: Discouraged for new work. Users should migrate to the recommended modern alternative (if applicable).
5. Deprecated: A formal, end-of-life warning that the feature is no longer recommended and is scheduled for removal in a future major release.
   - Risk: Low to moderate (for correctness), high (for future-proofing/continuity). It will receive no bug fixes or maintenance.
   - Usage recommendation: Do not use.
6. Removed: The feature's code has been permanently deleted from the GROMACS codebase, documentation still may refer to it as a past feature.
   - Risk: N/A
   - Usage recommendation: If required, use an older release which still contains it.

## 3.14 Environment Variables

GROMACS programs may be influenced by the use of environment variables. First of all, the variables set in the `GMXRC` file are essential for running and compiling GROMACS. Some other useful environment variables are listed in the following sections. Most environment variables function by being set in your shell to any non-NULL value. Specific requirements are described below if other values need to be set. You should consult the documentation for your shell for instructions on how to set environment variables in the current shell, or in configuration files for future shells. Note that requirements for exporting environment variables to jobs run under batch control systems vary and you should consult your local documentation for details.

### 3.14.1 Output Control

**GMX_AUTHOR_EMAIL**
Email address of corresponding author in `GMX_AUTHOR_NAME`.

**GMX_AUTHOR_NAME**
Name of corresponding author that simulation data is attributed to.

**GMX_COMPELDUMP**
Applies for computational electrophysiology setups only (see reference manual). The initial structure gets dumped to *pdb* (page 581) file, which allows to check whether multimeric channels have the correct PBC representation.

**GMX_DISABLE_GPU_TIMING**
Disables GPU timings in the log file for OpenCL.

**GMX_ENABLE_GPU_TIMING**
Enables GPU timings in the log file for CUDA and SYCL. Note that CUDA timings are incorrect with multiple streams, as happens with domain decomposition or with both non-bondeds and PME on the GPU (this is also the main reason why they are not turned on by default).

**GMX_LOG_BUFFER**
the size of the buffer for file I/O. When set to 0, all file I/O will be unbuffered and therefore very slow. This can be handy for debugging purposes, because it ensures that all files are always totally up-to-date.

**GMX_MAXBACKUP**
GROMACS automatically backs up old copies of files when trying to write a new file of the same name, and this variable controls the maximum number of backups that will be made, default 99. If set to 0 it fails to run if any output file already exists. And if set to -1 it overwrites any output file without making a backup.

**GMX_NO_QUOTES**
if this is explicitly set, no cool quotes will be printed at the end of a program.

**GMX_PRINT_LONGFORMAT**
use long float format when printing decimal values.

**GMX_SUPPRESS_DUMP**
prevent dumping of step files during (for example) blowing up during failure of constraint algorithms.

**GMX_TPI_DUMP**
dump all configurations to a *pdb* (page 581) file that have an interaction energy less than the value

set in this environment variable.

**GMX_TRAJECTORY_IO_VERBOSITY**
Defaults to 1, which prints frame count e.g. when reading trajectory files. Set to 0 for quiet operation.

**GMX_VIEW_XVG**
`GMX_VIEW_EPS` and `GMX_VIEW_PDB`, commands used to automatically view *xvg* (page 589), *eps* (page 576) and *pdb* (page 581) file types, respectively; they default to `xmgrace`, `ghostview` and `rasmol`. Set to empty to disable automatic viewing of a particular file type. The command will be forked off and run in the background at the same priority as the GROMACS tool (which might not be what you want). Be careful not to use a command which blocks the terminal (e.g. `vi`), since multiple instances might be run.

### 3.14.2 Debugging

**GMX_DD_NPULSE**
over-ride the number of DD pulses used (default 0, meaning no over-ride). Normally 1 or 2.

**GMX_DD_DEBUG**
general debugging trigger for every domain decomposition (default 0, meaning off). Currently only checks global-local atom index mapping for consistency.

**GMX_DD_NST_DUMP**
number of steps that elapse between dumping the current DD to a PDB file (default 0). This only takes effect during domain decomposition, so it should typically be 0 (never), 1 (every DD phase) or a multiple of *nstlist* (page 57).

**GMX_DD_NST_DUMP_GRID**
number of steps that elapse between dumping the current DD grid to a PDB file (default 0). This only takes effect during domain decomposition, so it should typically be 0 (never), 1 (every DD phase) or a multiple of *nstlist* (page 57).

**GMX_DISABLE_ALTERNATING_GPU_WAIT**
disables the specialized polling wait path used to wait for the PME and nonbonded GPU tasks completion to overlap to do the reduction of the resulting forces that arrive first. Setting this variable switches to the generic path with fixed waiting order.

**GMX_TEST_REQUIRED_NUMBER_OF_DEVICES**
sets the number of GPUs required by the test suite. By default, the test suite would fall-back to using CPU if GPUs could not be detected. Set it to a positive integer value to ensure that at least this at least this number of usable GPUs are detected. Default: 0 (not testing GPU availability).

There are a number of extra environment variables like these that are used in debugging - check the code!

### 3.14.3 Performance and Run Control

**GMX_AWH_NO_POINT_LIMIT**
Removes the upper limit on the number of points in an AWH bias grid. By default, an error is raised if the grid is unreasonably large and can cause sampling problems. Setting this variable will only remove this safety check. It is recommended instead to reduce the grid size, e.g., by using lower force constants.

**GMX_BONDED_NTHREAD_UNIFORM**
Value of the number of threads per rank from which to switch from uniform to localized bonded interaction distribution; optimal value dependent on system and hardware, default value is 4.

**GMX_CUDA_GRAPH**
Use CUDA Graphs to schedule a graph on each step rather than multiple activities scheduled to multiple CUDA streams, if the run conditions allow. Experimental.

**GMX_CYCLE_ALL**
times all code during runs. Incompatible with threads.

**GMX_CYCLE_BARRIER**
calls MPI_Barrier before each cycle start/stop call.

**GMX_DD_ORDER_ZYX**
build domain decomposition cells in the order (z, y, x) rather than the default (x, y, z).

**GMX_DD_RECORD_LOAD**
record DD load statistics for reporting at end of the run (default 1, meaning on)

**GMX_DD_SINGLE_RANK**
Controls the use of the domain decomposition machinery when using a single MPI rank. Value 0 turns DD off, 1 turns DD on. Default is automated choice based on heuristics.

**GMX_DD_USE_SENDRECV2**
during constraint and vsite communication, use a pair of `MPI_Sendrecv` calls instead of two simultaneous non-blocking calls (default 0, meaning off). Might be faster on some MPI implementations.

**GMX_DETAILED_PERF_STATS**
when set, print slightly more detailed performance information to the *log* (page 578) file. The resulting output is the way performance summary is reported in versions 4.5.x and thus may be useful for anyone using scripts to parse *log* (page 578) files or standard output.

**GMX_DISABLE_DYNAMICPRUNING**
disables dynamic pair-list pruning. Note that *gmx mdrun* (page 258) will still tune nstlist to the optimal value picked assuming dynamic pruning. Thus for good performance the -nstlist option should be used.

**GMX_DISABLE_GPU_DETECTION**
when set, disables GPU detection even if *gmx mdrun* (page 258) was compiled with GPU support.

**GMX_DISABLE_GPU_TIMING**
timing of asynchronously executed GPU operations can have a non-negligible overhead with short step times. Disabling timing can improve performance in these cases. Timings are disabled by default with CUDA and SYCL.

**GMX_DISABLE_SIMD_KERNELS**
disables architecture-specific SIMD-optimized (SSE2, SSE4.1, AVX, etc.) non-bonded kernels thus forcing the use of plain C kernels.

**GMX_DISABLE_STAGED_GPU_TO_CPU_PMEPP_COMM**
Use direct rather than staged GPU communications for PME force transfers from the PME GPU to the CPU memory of a PP rank. This may have advantages in PCIe-only servers, or for runs with low atom counts (which are more sensitive to latency than bandwidth).

**GMX_DISRE_ENSEMBLE_SIZE**
the number of systems for distance restraint ensemble averaging. Takes an integer value.

**GMX_DLB_BASED_ON_FLOPS**
do domain-decomposition dynamic load balancing based on flop count rather than measured time elapsed (default 0, meaning off). This makes the load balancing reproducible, which can be useful for debugging purposes. A value of 1 uses the flops; a value > 1 adds (value - 1)*5% of noise to the flops to increase the imbalance and the scaling.

**GMX_DLB_MAX_BOX_SCALING**
maximum percentage box scaling permitted per domain-decomposition load-balancing step (default 10)

**GMX_EMULATE_GPU**
emulate GPU runs by using algorithmically equivalent CPU reference code instead of GPU-accelerated functions. As the CPU code is slow, it is intended to be used only for debugging purposes.

**GMX_DISABLE_DIRECT_GPU_COMM**
Disables the direct GPU communication in multi-rank parallel runs even when build and simulation setup support it.

**GMX_ENABLE_NVSHMEM**
Enables GPU kernel-initiated communication using NVSHMEM in multi-rank parallel runs when build and simulation setup support it.

**GMX_ENABLE_NVSHMEM_FORCE_HALO_SYNC**
Inserts a synchronization after the fused force halo exchange kernel when using NVSHMEM. This is a debug/tuning knob.

**GMX_ENABLE_STAGED_GPU_TO_CPU_PMEPP_COMM**
Use a staged implementation of GPU communications for PME force transfers from the PME GPU to the CPU memory of a PP rank for thread-MPI. The staging is done via a GPU buffer on the PP GPU. This is expected to be beneficial for servers with direct communication links between GPUs.

**GMX_ENX_NO_FATAL**
disable exiting upon encountering a corrupted frame in an *edr* (page 576) file, allowing the use of all frames up until the corruption.

**GMX_FILLERS_IN_LOCAL_STATE**
Fillers particles are needed to make the number of particles a multiple of the SIMD or GPU warp/wave-front width for computing non-bonded interactions. These fillers can also be added to the local state, if all algorithms support this, which avoids indexing but increases the size of buffers. This environment variable can be set to 0 or 2 to force this behavior off or on. Setting it to 1 turns the behavior on when supported.

**GMX_FORCE_UPDATE**
update forces when invoking `mdrun -rerun`.

**GMX_FORCE_GPU_AWARE_MPI**
Override the result of build- and runtime GPU-aware MPI detection and force the use of direct

GPU MPI communication. Aimed at cases where the user knows that the MPI library is GPU-aware, but GROMACS is not able to detect this. Note that only CUDA and SYCL builds support such functionality.

**GMX_FORCE_UPDATE_DEFAULT_CPU**
Force update to run on the CPU by default, makes the `mdrun -update auto` behave as `-update cpu`.

**GMX_GPU_DD_COMMS**
Removed.

**GMX_GPU_DISABLE_COMPATIBILITY_CHECK**
Disables the hardware compatibility check in OpenCL and SYCL. Useful for developers and allows testing the OpenCL/SYCL kernels on non-supported platforms without source code modification.

**GMX_GPU_ID**
set in the same way as `mdrun -gpu_id`, `GMX_GPU_ID` allows the user to specify different GPU IDs for different ranks, which can be useful for selecting different devices on different compute nodes in a cluster. Cannot be used in conjunction with `mdrun -gpu_id`.

**GMX_GPU_NB_EWALD_TWINCUT**
force the use of twin-range cutoff kernel even if *rvdw* (page 62) equals *rcoulomb* (page 60) after PP-PME load balancing. The switch to twin-range kernels is automated, so this variable should be used only for benchmarking.

**GMX_GPU_NB_ANA_EWALD**
force the use of analytical Ewald kernels. Should be used only for benchmarking.

**GMX_GPU_NB_TAB_EWALD**
force the use of tabulated Ewald kernels. Should be used only for benchmarking.

**GMX_GPU_PME_DECOMPOSITION**
Enable the support for PME decomposition on GPU. This feature is supported with CUDA and SYCL backends, and allows using multiple PME ranks with GPU offload, which is expected to improve performance when scaling over many GPUs. Note: this feature still lacks substantial testing.

**GMX_GPU_DISABLE_BUFFER_OPS**
Disable coordinate/force transformation to be done on the GPU and execute these tasks on the CPU instead.

**GMX_GPU_PME_PP_COMMS**
Removed.

**GMX_GPUTASKS**
set in the same way as `mdrun -gputasks`, `GMX_GPUTASKS` allows the mapping of GPU tasks to GPU device IDs to be different on different ranks, if e.g. the MPI runtime permits this variable to be different for different ranks. Cannot be used in conjunction with `mdrun -gputasks`. Has all the same requirements as `mdrun -gputasks`.

**GMX_HEFFTE_RESHAPE_ALGORITHM**
Sets `heffte::plan_options::reshape_algorithm` to `p2p` (the default) or `p2p_plined`, `alltoallv`, or `alltoall`. See the HeFFTe docs for details.

**GMX_HEFFTE_USE_GPU_AWARE**
Sets `heffte::plan_options::use_gpu_aware` to `true` (the default) or `false`. See the HeFFTe docs for details.

**GMX_HEFFTE_USE_PENCILS**
Sets `heffte::plan_options::use_pencils` to `true` or `false` (the default). See the HeFFTe docs for details.

**GMX_HEFFTE_USE_REORDER**
Sets `heffte::plan_options::use_reorder` to `true` (the default) or `false`. See the HeFFTe docs for details.

**GMX_IGNORE_FSYNC_FAILURE**
allow *gmx mdrun* (page 258) to continue even if a file is missing.

**GMX_LJCOMB_TOL**
when set to a floating-point value, overrides the default tolerance of 1e-5 for force-field floating-point parameters.

**GMX_MAXCONSTRWARN**
if set to -1, *gmx mdrun* (page 258) will not exit if it produces too many LINCS warnings.

**GMX_NB_MIN_CI**
neighbor list balancing parameter used when running on GPU. Sets the target minimum number pair-lists in order to improve multi-processor load-balance for better performance with small simulation systems. Must be set to a non-negative integer, the 0 value disables list splitting. The default value is optimized for supported GPUs therefore changing it is not necessary for normal usage, but it can be useful on future architectures.

**GMX_NBNXM_CYCLE**
when set, print detailed neighbor search cycle counting.

**GMX_NBNXM_EWALD_ANALYTICAL**
force the use of analytical Ewald non-bonded kernels, mutually exclusive of `GMX_NBNXM_EWALD_TABLE`.

**GMX_NBNXM_EWALD_TABLE**
force the use of tabulated Ewald non-bonded kernels, mutually exclusive of `GMX_NBNXM_EWALD_ANALYTICAL`.

**GMX_NBNXM_PLAINC_1X1**
force the use of the reference 1x1 non-SIMD CPU non-bonded kernel, mutually exclusive of `GMX_NBNXM_PLAINC_4X4`, `GMX_NBNXM_SIMD_2XNN` and `GMX_NBNXM_SIMD_4XN`.

**GMX_NBNXM_PLAINC_4X4**
force the use of the reference 4x4 non-SIMD CPU non-bonded kernel, mutually exclusive of `GMX_NBNXM_PLAINC_1X1`, `GMX_NBNXM_SIMD_2XNN` and `GMX_NBNXM_SIMD_4XN`.

**GMX_NBNXM_SIMD_2XNN**
force the use of 2x(N+N) SIMD CPU non-bonded kernels, mutually exclusive of `GMX_NBNXM_PLAINC_1X1`, `GMX_NBNXM_PLAINC_4X4` and `GMX_NBNXM_SIMD_4XN`.

**GMX_NBNXM_SIMD_4XN**
force the use of 4xN SIMD CPU non-bonded kernels, mutually exclusive of `GMX_NBNXM_PLAINC_1X1`, `GMX_NBNXM_PLAINC_4X4` and `GMX_NBNXM_SIMD_2XNN`.

**GMX_NO_CART_REORDER**
used in initializing domain decomposition communicators. Rank reordering is default, but can be switched off with this environment variable.

**GMX_NO_INT, GMX_NO_TERM, GMX_NO_USR1**
disable signal handlers for SIGINT, SIGTERM, and SIGUSR1, respectively.

**GMX_NO_LJ_COMB_RULE**
force the use of LJ parameter lookup instead of using combination rules in the non-bonded kernels.

**GMX_NO_NODECOMM**
do not use separate inter- and intra-node communicators.

**GMX_NO_NONBONDED**
skip non-bonded calculations; can be used to estimate the possible performance gain from adding a GPU accelerator to the current hardware setup – assuming that this is fast enough to complete the non-bonded calculations while the CPU does bonded force and PME computation. Freezing the particles will be required to stop the system blowing up.

**GMX_NO_UPDATEGROUPS**
turns off update groups. May allow for a decomposition of more domains for small systems at the cost of communication during update.

**GMX_NOOPTIMIZEDKERNELS**
deprecated, use `GMX_DISABLE_SIMD_KERNELS` instead.

**GMX_NOPREDICT**
shell positions are not predicted.

**GMX_NSTLIST_DYNAMICPRUNING**
overrides the dynamic pair-list pruning interval chosen heuristically by mdrun. Values should be between the pruning frequency value (1 for CPU and 2 for GPU) and *`nstlist`* (page 57) `- 1`.

**GMX_OVERRIDE_IDENTICAL_SIMULATION_PARTS**
overrides the check for identical simulation parts when continuing simulations with *gmx mdrun* (page 258) with the `-multidir` option.

**GMX_PME_NUM_THREADS**
set the number of OpenMP or PME threads; overrides the default set by *gmx mdrun* (page 258); can be used instead of the `-npme` command line option, also useful to set heterogeneous per-process/-node thread count.

**GMX_PME_P3M**
use P3M-optimized influence function instead of smooth PME B-spline interpolation.

**GMX_PME_THREAD_DIVISION**
PME thread division in the format "x y z" for all three dimensions. The sum of the threads in each dimension must equal the total number of PME threads (set in `GMX_PME_NTHREADS`).

**GMX_PMEONEDD**
if the number of domain decomposition cells is set to 1 for both x and y, decompose PME in one dimension.

**GMX_PULL_PARTICIPATE_ALL**
disable the default heuristic for when to use a separate pull MPI communicator (at >=32 ranks).

**GMX_REPORT_CPU_AFFINITY**
print to the log file the CPU affinity of each MPI rank at startup. This prints the effective affinity mask of the main thread before OpenMP and GPU offload are initialized (unless they do so on library load, as is the case with `libgomp`).

**GMX_REQUIRE_SHELL_INIT**
require that shell positions are initiated.

**GMX_TPIC_MASSES**
should contain multiple masses used for test particle insertion into a cavity. The center of mass of

the last atoms is used for insertion into the cavity.

**GMX_VERLET_BUFFER_PRESSURE_TOLERANCE**
sets the maximum tolerated error in the pressure in bar for the automated tuning of the Verlet pair-list buffering. Can only be used with system where this tolerance has not been set using the mdp parameter.

**GMX_VERLET_BUFFER_RES**
resolution of buffer size in Verlet cutoff scheme. The default value is 0.001, but can be overridden with this environment variable.

**HWLOC_XMLFILE**
Not strictly a GROMACS environment variable, but on large machines the hwloc detection can take a few seconds if you have lots of MPI processes. If you run the hwloc command **`lstopo out.xml`** and set this environment variable to point to the location of this file, the hwloc library will use the cached information instead, which can be faster.

**MDRUN**
the *gmx mdrun* (page 258) command used by *gmx tune_pme* (page 340).

**MPIRUN**
the `mpirun` command used by *gmx tune_pme* (page 340).

### 3.14.4 OpenCL management

Currently, several environment variables exist that help customize some aspects of the OpenCL version of GROMACS. They are mostly related to the runtime compilation of OpenCL kernels, but they are also used in device selection.

**GMX_OCL_DEBUG**
Use in conjunction with `OCL_FORCE_CPU` or with an AMD device. It adds the debug flag to the compiler options (-g).

**GMX_OCL_DISABLE_FASTMATH**
Prevents the use of `-cl-fast-relaxed-math` compiler option. Note: fast math is always disabled on Intel devices due to instability.

**GMX_OCL_DISABLE_I_PREFETCH**
Disables i-atom data (type or LJ parameter) prefetch allowing testing.

**GMX_OCL_ENABLE_I_PREFETCH**
Enables i-atom data (type or LJ parameter) prefetch allowing testing on platforms where this behavior is not default.

**GMX_OCL_DUMP_INTERM_FILES**
If defined, intermediate language code corresponding to the OpenCL build process is saved to file. Caching has to be turned off in order for this option to take effect.

- NVIDIA GPUs: PTX code is saved in the current directory with the name `device_name.ptx`
- AMD GPUs: `.IL/.ISA` files will be created for each OpenCL kernel built. For details about where these files are created check AMD documentation for `-save-temps` compiler option.

**GMX_OCL_DUMP_LOG**
If defined, the OpenCL build log is always written to the mdrun log file. Otherwise, the build log is written to the log file only when an error occurs.

**GMX_OCL_FILE_PATH**
Use this parameter to force GROMACS to load the OpenCL kernels from a custom location. Use it only if you want to override GROMACS default behavior, or if you want to test your own kernels.

**GMX_OCL_FORCE_AMD_WAVEFRONT64**
Force the use of Wave64 mode on AMD devices. This allows using OpenCL on RDNA-family devices, but is not recommended. For development use only.

**GMX_OCL_FORCE_CPU**
Force the selection of a CPU device instead of a GPU. This exists only for debugging purposes. Do not expect GROMACS to function properly with this option on, it is solely for the simplicity of stepping in a kernel and see what is happening.

**GMX_OCL_GENCACHE**
Enable OpenCL binary caching. Only intended to be used for development and (expert) testing as neither concurrency nor cache invalidation is implemented safely!

**GMX_OCL_NOFASTGEN**
If set, generate and compile all algorithm flavors, otherwise only the flavor required for the simulation is generated and compiled.

**GMX_OCL_NOOPT**
Disable optimisations. Adds the option `cl-opt-disable` to the compiler options.

**GMX_OCL_SHOW_DIAGNOSTICS**
Use Intel OpenCL extension to show additional runtime performance diagnostics.

**GMX_OCL_VERBOSE**
If defined, it enables verbose mode for OpenCL kernel build. Currently available only for NVIDIA GPUs. See `GMX_OCL_DUMP_LOG` for details about how to obtain the OpenCL build log.

### 3.14.5 SYCL management

**GMX_SYCL_ALLOW_ALL_DEVICES**
Include CPUs and Accelerator SYCL devices in addition to GPUs. This exists for debugging purposes and for enabling experimental backends.

### 3.14.6 Analysis and Core Functions

**GMX_AMBER_LEAP_ATOM_REORDERING_VERBOSE**
make *gmx grompp* (page 230) print indices and types of dihedral atoms that were processed for reordering to match AMBER LEaP.

**GMX_DIPOLE_SPACING**
spacing used by *gmx dipoles* (page 193).

**GMX_ENER_VERBOSE**
make *gmx energy* (page 214) and *gmx eneconv* (page 211) loud and noisy.

**GMX_MAXRESRENUM**
sets the maximum number of residues to be renumbered by *gmx grompp* (page 230). A value of -1 indicates all residues should be renumbered.

**GMX_NO_FFRTP_TER_RENAME**
Some force fields (like AMBER) use specific names for N- and C- terminal residues (NXXX and CXXX) as *rtp* (page 582) entries that are normally renamed. Setting this environment variable disables this renaming.

**GMX_USE_XMGR**
sets viewer to `xmgr` (deprecated) instead of `xmgrace`.

**GMXTIMEUNIT**
the time unit used in output files, can be anything in fs, ps, ns, us, ms, s, m or h.

**VMD_PLUGIN_PATH**
where to find VMD plug-ins. Needed to be able to read file formats recognized only by a VMD plug-in.

**VMDDIR**
base path of VMD installation.

### 3.14.7 MDModules

**GMX_NNPOT_SKIP_MODEL_CHECK**
skips the compatibility check between the neural network potential model and its inputs. This can be useful if errors are encountered due to the dummy input, or to prepare *tpr* (page 586) files without a Libtorch-compatible installation.

## 3.15 Floating point arithmetic

GROMACS spends its life doing arithmetic on real numbers, often summing many millions of them. These real numbers are encoded on computers in so-called binary floating-point representation. This representation is somewhat like scientific exponential notation (but uses binary rather than decimal), and is necessary for the fastest possible speed for calculations. Unfortunately the laws of algebra only approximately apply to binary floating-point. In part, this is because some real numbers that are represented simply and exactly in decimal (like 1/5=0.2) have no exact representation in binary floating-point, just as 1/3 cannot be represented in decimal. There are many sources you can find with a search engine that discuss this issue more exhaustively, such as Wikipedia and David Goldberg's 1991 paper *What every computer scientist should know about floating-point arithmetic* (article, addendum). Bruce Dawson also has a written a number of very valuable blog posts on modern floating-point programming at his Random ASCII site that are worth reading.

So, the sum of a large number of binary representations of exact decimal numbers need not equal the expected algebraic or decimal result. Users observe this phenomenon in sums of partial charges expressed to two decimal places that sometimes only approximate the integer total charge to which they contribute (however a deviation in the first decimal place would always be indicative of a badly-formed topology). When GROMACS has to represent such floating-point numbers in output, it sometimes uses a computer form of scientific notation known as E notation. In such notation, a number like -9.999971e-01 is actually -0.9999971, which is close enough to -1 for purposes of assessing the total charge of a system.

It is also not appropriate for GROMACS to guess to round things, because such rounding relies on assumptions about the inputs that need not be true. Instead the user needs to understand how their tools work.

## 3.16 Security when using GROMACS

We advise the users of GROMACS to be careful when using GROMACS with files obtained from an unknown source (e.g. the Internet).

We cannot guarantee that the program will not crash with serious errors that could cause execution of code with the same privileges as GROMACS and e.g. delete the contents of your home directory.

Files that the user has created themselves do not carry those risks, but may still misbehave and crash or consume large amounts of resources upon malformed input.

Run input files obtained from outside sources should be treated with the same caution as an executable file from the same source.

## 3.17 Policy for deprecating GROMACS functionality

Occasionally functionality ceases being useful, is unable to be fixed or maintained, or its user interface needs to be improved. The development team does this sparingly. Broken functionality might be removed without notice if nobody willing to fix it can be found. Working functionality will be changed only after announcing in the previous major release the intent to remove and/or change the form of such functionality. Thus there is typically a year for users and external tool providers to prepare for such changes, and contact the GROMACS developers to see how they might be affected and how best to adapt.

There is a current list of anticipated changes and deprecated functionality in the "Major release" *notes* (page 838).

When environment variables are deprecated, it is up to the user to make sure that their scripts are updated accordingly for the new release. In cases where it is sensible, the development team should do the effort to keep the old environment variables working for one extra release cycle, before fully removing them. The user should be informed about this future deprecation with a warning. If keeping the old environment variable is not possible or highly problematic, setting the removed environment variable should be triggering a warning during one release cycle.

CHAPTER

FOUR

# SHORT HOW-TO GUIDES

A number of short guides are presented here to help users getting started with simulations. Useful third-party tutorials provided by Justin Lemkul are found here http://www.mdtutorials.com/.

## 4.1 Beginners

For those just starting out with GROMACS and / or *Molecular Dynamics Simulations* (page 392) it can be very daunting. It is highly recommended that the various and extensive documentation that has been made available for GROMACS is read first, plus papers published in the area of interest.

### 4.1.1 Resources

- GROMACS *Reference Manual* (page 418) - very detailed document that can also act as a very good introduction for *MD* (page 392) in general.
- *Flow Chart* (page 36)- simple flow chart of a typical GROMACS MD run of a protein in a box of water.
- Molecular dynamics simulations and GROMACS introduction (slides, video) - force fields, integrators, control of temperature and pressure (Berk Hess).

## 4.2 Adding a Residue to a Force Field

### 4.2.1 Adding a new residue

If you have the need to introduce a new residue into an existing force field so that you can use *pdb2gmx* (page 281), or modify an existing one, there are several files you will need to modify. You must consult the *Reference Manual* (page 418) for description of the required format. Follow these steps:

1. Add the residue to the *rtp* (page 582) file for your chosen force field. You might be able to copy an existing residue, rename it and modify it suitably, or you may need to use an external topology generation tool and adapt the results to the *rtp* (page 582) format.
2. If you need hydrogens to be able to be added to your residue, create an entry in the relevant *hdb* (page 578) file.
3. If you are introducing new atom types, add them to the `atomtypes.atp` and `ffnonbonded.itp` files.
4. If you require any new bonded types, add them to `ffbonded.itp`.

5. Add your residue to `residuetypes.dat` with the appropriate specification (Protein, DNA, Ion, etc).
6. If the residue involves special connectivity to other residues, update `specbond.dat`.

Note that if all you are doing is simulating some weird ligand in water, or some weird ligand with a normal protein, then the above is more work than generating a standalone *itp* (page 578) file containing a `[moleculetype]` (for example, by modifying the *top* (page 584) produced by some parameterization server), and inserting an `#include` of that *itp* (page 578) file into a *top* (page 584) generated for the system without that weird ligand.

### 4.2.2 Modifying a force field

Modifying a force field is best done by making a full copy of the installed forcefield directory and `residuetypes.dat` into your local working directory:

```
cp -r $GMXLIB/residuetypes.dat $GMXLIB/amber99sb.ff .
```

Then, modify those local copies as above. *pdb2gmx* (page 281) will then find both the original and modified version and you can choose the modified version interactively from the list, or if you use the *pdb2gmx* (page 281) `-ff` option the local version will override the system version.

## 4.3 Water solvation

When using *solvate* (page 319) to generate a box of solvent, you need to supply a pre-equilibrated box of a suitable solvent for *solvate* (page 319) to stack around your solute(s), and then to truncate to give the simulation volume you desire. When using any 3-point model (e.g. `SPC`, `SPC/E` or `TIP3P`) you should specify `-cs spc216.gro` which will take this file from `the gromacs/share/top` directory. Other water models (e.g. `TIP4P` and `TIP5P`) are available as well. Check the contents of the `/share/top` subdirectory of your GROMACS installation. After solvation, you should then be sure to equilibrate for at least 5-10ps at the desired temperature. You will need to select the right water model in your *top* (page 584) file, either with the `-water` flag to *pdb2gmx* (page 281), or by editing your *top* (page 584) file appropriately by hand.

For information about how to use solvents other than pure water, please see *Non-Water Solvation* (page 408) or *Mixed Solvents* (page 409).

## 4.4 Non water solvent

It is possible to use solvents other than water in GROMACS. The only requirements are that you have a pre-equilibrated box of whatever solvent you need, and suitable parameters for this species in a simulation. One can then pass the solvent box to the -cs switch of *solvate* (page 319) to accomplish solvation.

A series of about 150 different equilibrated liquids validated for use with GROMACS, and for the OPLS/AA and GAFF force fields, can be found at virtualchemistry.

### 4.4.1 Making a non-aqueous solvent box

Choose a box density and box size. The size does not have to be that of your eventual simulation box - a 1nm cube is probably fine. Generate a single molecule of the solvent. Work out how much volume a single molecule would have in the box of your chosen density and size. Use *editconf* (page 208) to place a box of that size around your single molecule. Then use *editconf* (page 208) to move the molecule a little bit off center. Then use *genconf* (page 226) `-rot` to replicate that box into a large one of the right size and density. Then equilibrate thoroughly to remove the residual ordering of the molecules, using NVT and periodic boundary conditions. Now you have a box you can pass to *solvate* (page 319) `-cs`, which will replicate it to fit the size of the actual simulation box.

## 4.5 Mixed solvent

A common question that new users have is how to create a system with mixed solvent (urea or DMSO at a given concentration in water, for example). The simplest procedure for accomplishing this task is as follows:

- Determine the number of co-solvent molecules necessary, given the box dimensions of your system.
- Generate a coordinate file of a single molecule of your co-solvent (i.e., `urea.gro`).
- Use the `-ci -nmol` options of *gmx insert-molecules* (page 248) to add the required number of co-solvent molecules to the box.
- Fill the remainder of the box with water (or whatever your other solvent is) using *gmx solvate* (page 319) or *gmx insert-molecules* (page 248).
- Edit your *topology* (page 584) to `#include` the appropriate *itp* (page 578) files, as well as make changes to the `[ molecules ]` directive to account for all the species in your system.

## 4.6 Making Disulfide Bonds

The easiest way to do this is by using the mechanism implemented with the `specbond.dat` file and *pdb2gmx* (page 281). You may find *pdb2gmx* (page 281) `-ss yes` is useful. The sulfur atoms will need to be in the same unit that *pdb2gmx* (page 281) is converting to a `moleculetype`, so invoking *pdb2gmx* (page 281) `-chainsep` correctly may be required. See *pdb2gmx* (page 281) `-h`. This requires that the two sulfur atoms be within a distance + tolerance (usually 10%) in order to be recognised as a disulfide. If your sulfur atoms are not this close, then either you can

- edit the contents of `specbond.dat` to allow the bond formation and do energy minimization very carefully to allow the bond to relax to a sensible length, or
- run a preliminary EM or MD with a distance restraint (and no disulfide bond) between these sulfur atoms with a large force constant so that they approach within the existing `specbond.dat` range to provide a suitable coordinate file for a second invocation of *pdb2gmx* (page 281).

Otherwise, editing your *top* (page 584) file by hand is the only option.

## 4.7 Running membrane simulations in GROMACS

### 4.7.1 Running Membrane Simulations

Users frequently encounter problems when running simulations of lipid bilayers, especially when a protein is involved. Users seeking to simulate membrane proteins may find this tutorial useful.

One protocol for the simulation of membrane proteins consists of the following steps:

1. Choose a force field for which you have parameters for the protein and lipids.
2. Insert the protein into the membrane. (For instance, use g_membed on a pre-formed bilayer or do a coarse-grained self-assembly simulation and then convert back to the atomistic representation.)
3. Solvate the system and add ions to neutralize excess charges and adjust the final ion concentration.
4. Energy minimize.
5. Let the membrane adjust to the protein. Typically run MD for ~5-10ns with restraints (1000 kJ/(mol nm2) on all protein heavy atoms.
6. Equilibrate without restraints.
7. Run production MD.

### 4.7.2 Adding waters with genbox

When generating waters around a pre-formed lipid membrane with *solvate* (page 319) you may find that water molecules get introduced into interstices in the membrane. There are several approaches to removing these, including

- a short MD run to get the hydrophobic effect to exclude these waters. In general this is sufficient to reach a water-free hydrophobic phase, as the molecules are usually expelled quickly and without disrupting the general structure. If your setup relies on a completely water-free hydrophobic phase at the start, you can try to follow the advice below:
- Set the `-radius` option in *gmx solvate* (page 319) to change the water exclusion radius,
- copy `vdwradii.dat` from your `$GMXLIB` location to the working directory, and edit it to increase the radii of your lipid atoms (between 0.35 and 0.5nm is suggested for carbon) to prevent *solvate* (page 319) from seeing interstices large enough for water insertion,
- editing your structure by hand to delete them (remembering to adjust your atom count for *gro* (page 577) files and to account for any changes in the *topology* (page 584)), or
- use a script someone wrote to remove them.

### 4.7.3 External material

- Membrane simulations slides , membrane simulations video - (Erik Lindahl).
- tutorial for membrane protein simulations - designed to demonstrate what sorts of questions and problems occur when simulating proteins that are embedded within a lipid bilayer.
- Combining the OPLS-AA forcefield with the Berger lipids A detailed description of the motivation, method, and testing.
- Several Topologies for membrane proteins with different force fields gaff, charmm berger Shirley W. I. Siu, Robert Vacha, Pavel Jungwirth, Rainer A. Böckmann: Biomolecular simulations of membranes: Physical properties from different force fields.
- Lipidbook is a public repository for force-field parameters of lipids, detergents and other molecules that are used in the simulation of membranes and membrane proteins. It is described in: J. Domański, P. Stansfeld, M.S.P. Sansom, and O. Beckstein. J. Membrane Biol. 236 (2010), 255—258. doi:10.1007/s00232-010-9296-8.

## 4.8 Parameterization of novel molecules

Most of your parametrization questions/problems can be resolved very simply, by remembering the following two rules:

- **You should not mix and match force fields**. *Force fields* (page 393) are (at best) designed to be self-consistent, and will not typically work well with other force fields. If you simulate part of your system with one force field and another part with a different force field which is not parametrized with the first force field in mind, your results will probably be questionable, and hopefully reviewers will be concerned. Pick a force field. Use that force field.
- If you need to develop new parameters, derive them in a manner consistent with how the rest of the force field was originally derived, which means that you will need to review the original literature. There isn't a single right way to derive force field parameters; what you need is to derive parameters that are consistent with the rest of the force field. How you go about doing this depends on which force field you want to use. For example, with AMBER force fields, deriving parameters for a non-standard amino acid would probably involve doing a number of different quantum calculations, while deriving GROMOS or OPLS parameters might involve more (a) fitting various fluid and liquid-state properties, and (b) adjusting parameters based on experience/chemical intuition/analogy. Some suggestions for automated approaches can be found *here* (page 40).

It would be wise to have a reasonable amount of simulation experience with GROMACS before attempting to parametrize new force fields, or new molecules for existing force fields. These are expert topics, and not suitable for giving to (say) undergraduate students for a research project, unless you like expensive quasi-random number generators. A very thorough knowledge of *Chapter 5: Interaction function and force fields* (page 481) of the GROMACS Reference Manual will be required. If you haven't been warned strongly enough, please read below about parametrization for exotic species.

Another bit of advice: Don't be more haphazard in obtaining parameters than you would be buying fine jewellery. Just because the guy on the street offers to sell you a *diamond* necklace for $10 doesn't mean that's where you should buy one. Similarly, it isn't necessarily the best strategy to just download parameters for your molecule of interest from the website of someone you've never heard of, especially if they don't explain how they got the parameters.

Be forewarned about using PRODRG topologies without verifying their contents: the artifacts of doing so are now published, along with some tips for properly deriving parameters for the GROMOS family of force fields.

### 4.8.1 Exotic Species

So, you want to simulate a protein/nucleic acid system, but it binds various exotic metal ions (ruthenium?), or there is an iron-sulfur cluster essential for its functionality, or similar. But, (unfortunately?) there aren't parameters available for these in the force field you want to use. What should you do? You shoot an e-mail to the GROMACS user discussion forum, and get referred to the FAQs.

If you really insist on simulating these in molecular dynamics, you'll need to obtain parameters for them, either from the literature, or by doing your own parametrization. But before doing so, it's probably important to stop and think, as sometimes there is a reason there may not already be parameters for such atoms/clusters. In particular, here are a couple of basic questions you can ask yourself to see whether it's reasonable to develop/obtain standard parameters for these and use them in molecular dynamics:

- Are quantum effects (i.e. charge transfer) likely to be important? (i.e., if you have a divalent metal ion in an enzyme active site and are interested in studying enzyme functionality, this is probably a huge issue).
- Are standard force field parametrization techniques used for my force field of choice likely to fail for an atom/cluster of this type? (i.e. because Hartree-Fock 6-31G* can't adequately describe transition metals, for example)

If the answer to either of these questions is "Yes", you may want to consider doing your simulations with something other than classical molecular dynamics.

Even if the answer to both of these is "No", you probably want to consult with someone who is an expert on the compounds you're interested in, before attempting your own parametrization. Further, you probably want to try parametrizing something more straightforward before you embark on one of these.

## 4.9 Potential of Mean Force

The potential of mean force (PMF) is defined as the potential that gives an average force over all the configurations of a given system. There are several ways to calculate the PMF in GROMACS, probably the most common of which is to make use of the pull code. The steps for obtaining a PMF using umbrella sampling, which allows for sampling of statistically-improbable states, are:

- Generate a series of configurations along a reaction coordinate (from a steered MD simulation, a normal MD simulation, or from some arbitrarily-created configurations)
- Use umbrella sampling to restrain these configurations within sampling windows.
- Use *gmx wham* (page 349) to make use of the WHAM algorithm to reconstruct a PMF curve.

A more detailed tutorial is linked here for umbrella sampling.

## 4.10 Single-Point Energy

Computing the energy of a single configuration is an operation that is sometimes useful. The best way to do this with GROMACS is with the *mdrun* (page 258) `-rerun` mechanism, which applies the model physics in the *tpr* (page 586) to the configuration in the trajectory or coordinate file supplied to mdrun.

```
mdrun -s input.tpr -rerun configuration.pdb
```

Note that the configuration supplied must match the topology you used when generating the *tpr* (page 586) file with *grompp* (page 230). The configuration you supplied to *grompp* (page 230) is irrelevant, except perhaps for atom names. You can also use this feature with energy groups (see the Reference manual), or with a trajectory of multiple configurations (and in this case, by default *mdrun* (page 258) will do neighbour searching for each configuration, because it can make no assumptions about the inputs being similar).

A zero-step energy minimization does a step before reporting the energy, and a zero-step MD run has (avoidable) complications related to catering to possible restarts in the presence of constraints, so neither of those procedures are recommended.

## 4.11 Carbon Nanotube

### 4.11.1 Robert Johnson's Tips

Taken from Robert Johnson's posts on the gmx-users mailing list archive.

- Be absolutely sure that the "terminal" carbon atoms are sharing a bond in the topology file.
- Use `periodic_molecules = yes` in your *mdp* (page 579) file for input in *gmx grompp* (page 230).
- Even if the topology is correct, crumpling may occur if you place the nanotube in a box of wrong dimension, so use VMD to visualize the nanotube and its periodic images and make sure that the space between images is correct. If the spacing is too small or too big, there will be a large amount of stress induced in the tube which will lead to crumpling or stretching.
- Don't apply pressure coupling along the axis of the nanotube. In fact, for debugging purposes, it might be better to turn off pressure coupling altogether until you figure out if anything is going wrong, and if so, what.
- When using *x2top* (page 355) with a specific force field, things are assumed about the connectivity of the molecule. The terminal carbon atoms of your nanotube will only be bonded to, at most, 2 other carbons, if periodic, or one if non-periodic and capped with hydrogens.
- You can generate an "infinite" nanotube with the `-pbc` option to *x2top* (page 355). Here, *x2top* (page 355) will recognize that the terminal C atoms actually share a chemical bond. Thus, when you use *grompp* (page 230) you won't get an error about a single bonded C.

### 4.11.2 Andrea Minoia's tutorial

Modeling Carbon Nanotubes with GROMACS (also archived as http://chembytes.wikidot.com/grocnt) contains everything to set up simple simulations of a CNT using OPLS-AA parameters. Structures of simple CNTs can be easily generated e.g. by buildCstruct (Python script that also adds terminal hydrogens) or TubeGen Online (just copy and paste the PDB output into a file and name it cnt.pdb).

To make it work with modern GROMACS you'll probably want to do the following:

- make a directory cnt_oplsaa.ff
- In this directory, create the following files, using the data from the tutorial page:
  - forcefield.itp from the file in section *itp* (page 578)
  - atomnames2types.n2t from the file in section *n2t* (page 581)
  - aminoacids.rtp from the file in section *rtp* (page 582)
- generate a topology with the custom forcefield (the cnt_oplsaa.ff directory must be in the same directory as where the *gmx x2top* (page 355) command is run or it must be found on the GMXLIB path), `-noparam` instructs *gmx x2top* (page 355) to not use bond/angle/dihedral force constants from the command line (-kb, -ka, -kd) but rely on the force field files; however, this necessitates the next step (fixing the dihedral functions)

```
gmx x2top -f cnt.gro -o cnt.top -ff cnt_oplsaa -name CNT -noparam
```

The function type for the dihedrals is set to '1' by *gmx x2top* (page 355) but the force field file specifies type '3'. Therefore, replace func type '1' with '3' in the `[ dihedrals ]` section of the topology file. A quick way is to use sed (but you might have to adapt this to your operating system; also manually look at the top file and check that you only changed the dihedral func types):

```
sed -i~ '/\[ dihedrals \]/,/\[ system \]/s/1 *$/3/' cnt.top
```

Once you have the topology you can set up your system. For instance, a simple in-vacuo simulation (using your favourite parameters in em.*mdp* (page 579) and md.*mdp* (page 579)):

Put into a slightly bigger box:

```
gmx editconf -f cnt.gro -o boxed.gro -bt dodecahedron -d 1
```

Energy minimise in vacuuo:

```
gmx grompp -f em.mdp -c boxed.gro -p cnt.top -o em.tpr
gmx mdrun -v -deffnm em
```

MD in vacuuo:

```
gmx grompp -f md.mdp -c em.gro -p cnt.top -o md.tpr
gmx mdrun -v -deffnm md
```

Look at trajectory:

```
gmx trjconv -f md.xtc -s md.tpr -o md_centered.xtc -pbc mol -center
gmx trjconv -s md.tpr -f md_centered.xtc -o md_fit.xtc -fit␣
```

(continues on next page)

(continued from previous page)

```
↪rot+trans
vmd em.gro md_fit.xtc
```

## 4.12 Visualization Software

Some programs that are useful for visualizing either a trajectory file and/or a coordinate file are:

- VMD - a molecular visualization program for displaying, animating, and analyzing large biomolecular systems using 3-D graphics and built-in scripting. Reads GROMACS trajectories.
- PyMOL - capable molecular viewer with support for animations, high-quality rendering, crystallography, and other common molecular graphics activities. Does not read GROMACS trajectories in default configuration, requiring conversion to PDB or similar format. When compiled with VMD plugins, *trr* (page 586) & *xtc* (page 588) files can be loaded.
- Rasmol - the derivative software Protein Explorer (below) might be a better alternative, but the Chime component requires windows. Rasmol works fine on Unix.
- Protein Explorer - a RasMol-derivative, is the easiest-to-use and most powerful software for looking at macromolecular structure and its relation to function. It runs on Windows or Macintosh/PPC computers.
- Chimera - A full featured, Python-based visualization program with all sorts of features for use on any platform. The current version reads GROMACS trajectories.
- Molscript - This is a script-driven program form high-quality display of molecular 3D structures in both schematic and detailed representations. You can get an academic license for free from Avatar.

### 4.12.1 Topology bonds vs Rendered bonds

Remember that each of these visualization tools is only looking at the coordinate file you gave it. Thus it's not using your topology which is described in either your *top* (page 584) file or your *tpr* (page 586) file. Each of these programs makes their own guesses about where the chemical bonds are for rendering purposes, so do not be surprised if the heuristics do not always match your topology.

## 4.13 Extracting Trajectory Information

There are several techniques available for finding information in GROMACS trajectory (*trr* (page 586), *xtc* (page 588), *tng* (page 584)) files.

- use the GROMACS trajectory analysis utilities
- use *gmx traj* (page 328) to write a *xvg* (page 589) file and read that in an external program as above
- write your own C code using `gromacs/share/template/template.cpp` as a template
- use *gmx dump* (page 205) and redirect the shell output to a file and read that in an external program like MATLAB, or Mathematica or other spreadsheet software.

## 4.14 External tools to perform trajectory analysis

In recent years several external tools have matured sufficiently to analyse diverse sets of trajectory data from several simulation packages. Below is a short list of tools (in an alphabetical order) that are known to be able to analyse GROMACS trajectory data.

- LOOS
- MDAnalysis
- MDTraj
- Pteros

## 4.15 Plotting Data

The various GROMACS analysis utilities can generate *xvg* (page 589) files. These are text files that have been specifically formatted for direct use in Grace. You can, however, in all GROMACS analysis programs turn off the Grace specific codes by running the programs with the `-xvg none` option. This circumvents problems with tools like gnuplot and Excel (see below).

Note that Grace uses some embedded backslash codes to indicate superscripts, normal script, etc. in units. So "Area (nmS2N)" is nm squared.

### 4.15.1 Software

Some software packages that can be used to graph data in a *xvg* (page 589) file:

- Grace - WYSIWYG 2D plotting tool for the X Window System and M*tif. Grace runs on practically any version of Unix-like OS, provided that you can satisfy its library dependencies (Lesstif is a valid free alternative to Motif). It is also available for the other common operation systems.
- gnuplot - portable command-line driven interactive data and function plotting utility for UNIX, IBM OS/2, MS Windows, DOS, Macintosh, VMS, Atari and many other platforms. Remember to use:

  ```
  set datafile commentschars "#@&"
  ```

  to avoid gnuplot trying to interpret Grace-specific commands in the *xvg* (page 589) file or use the `-xvg none` option when running the analysis program. For simple usage,:

  ```
  plot "file.xvg" using 1:2 with lines
  ```

  is a hack that will achieve the right result.
- Matplotlib - a popular Python library for visualization. A simple script that will plot the data in `file.xvg` and show the result on the screen

  ```
  import numpy as np
  import matplotlib.pyplot as plt
  x, y = np.loadtxt("file.xvg", comments=["@", "#", "&"],␣
  ↪unpack=True)
  ```

(continues on next page)

(continued from previous page)

```
plt.plot(x, y)
plt.show()
```

- MS Excel - change the file extension to .csv and open the file (when prompted, choose to ignore the first 20 or so rows and select fixed-width columns, if you are using German MS Excel version, you have to change decimal delimiter from "," to ".", or use your favourite *nix tool.
- Sigma Plot A commercial tool for windows with some useful analysis tools in it.
- R - freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc.
- SPSS A commercial tool (Statistical Product and Service Solutions), which can also plot and analyse data.

## 4.16 Micelle Clustering

This is necessary for the *gmx spatial* (page 322) tool if you have a fully-formed single aggregate and want to generate the spatial distribution function for that aggregate or for solvent around that aggregate.

Clustering to ensure that the micelle is not split across a *periodic boundary condition* (page 386) border is an essential step prior to calculating properties such as the radius of gyration and the radial distribution function. Without this step your results will be incorrect (a sign of this error is unexplained huge fluctuations in the calculated value when the visualized trajectory looks fine).

Three steps are required:

- use *trjconv* (page 334) `-pbc cluster` to obtain a single frame that has all of the lipids in the unit cell. This must be the first frame of your trajectory. A similar frame from some previous timepoint will not work.
- use *grompp* (page 230) to make a new *tpr* (page 586) file based on the frame that was output from the step above.
- use *trjconv* (page 334) `-pbc nojump` to produce the desired trajectory using the newly produced *tpr* (page 586) file.

More explicitly, the same steps are:

```
gmx trjconv -f a.xtc -o a_cluster.gro -e 0.001 -pbc cluster
gmx grompp -f a.mdp -c a_cluster.gro -o a_cluster.tpr
gmx trjconv -f a.xtc -o a_cluster.xtc -s a_cluster.tpr -pbc nojump
```

CHAPTER

FIVE

# REFERENCE MANUAL

This part of the documentation covers implementation details of GROMACS.

For quick simulation set-up and short explanations, please refer to the *User guide* (page 34).

Help with the installation of GROMACS can be found in the *Install guide* (page 3).

If you want to help with developing GROMACS, your are most welcome to read up on the *Developer Guide* (page 742) and continue right away with coding for GROMACS.

## 5.1 Preface and Disclaimer

GROMACS - 2027.0-dev

Current Contributors: Mark Abraham, Andrey Alekseenko, Brian Andrews, Paul Bauer, Cathrine Bergh, Hugh Bird, Eliane Briand, Ania Brown, Yiqi Chen, Mahesh Doijade, Giacomo Fiorin, Stefan Fleischmann, Sergey Gorelov, Gilles Gouaillardet, Alan Gray, Farzaneh Jalalypour, Petter Johansson, Carsten Kutzner, Grzegorz Łazarski, Justin A. Lemkul, Magnus Lundborg, Julio Maia, Pascal Merz, Vedran Miletić, Dmitry Morozov, Lukas Müllender, Szilárd Páll, Andrea Pasquadibisceglie, Michele Pellegrino, Nicola Piasentin, Daniele Rapetti, Muhammad Umair Sadiq, Hubert Santuz, Michael Shirts, Tatiana Shugaeva, Alexey Shvetsov, Bálint Soproni, Philip Turner, Alessandra Villa, Yang Zhang

Previous Contributors: Emile Apol, Rossen Apostolov, James Barnett, Vladimir Basov, Herman J.C. Berendsen, Pär Bjelkmar, Christian Blau, Viacheslav Bolnykh, Kevin Boyd, Aldert van Buuren, Carlo Camilloni, Rudi van Drunen, Anton Feenstra, Oliver Fleetwood, Vytas Gapsys, Gaurav Garg, Gerrit Groenhof, Bert de Groot, Anca Hamuraru, Vincent Hindriksen, Victor Holanda, M. Eric Irrgang, Aleksei Iupinov, Joe Jordan, Christoph Junghans, Prashanth Kanduri, Dimitrios Karkoulis, Peter Kasson, Sebastian Kehl, Sebastian Keller, Jiri Kraus, Per Larsson, Viveca Lindahl, Erik Marklund, Pieter Meulenhoff, Teemu Murtola, Julien Nabet, Sander Pronk, Roland Schulz, Alfons Sijbers, David van der Spoel, Peter Tieleman, Carsten Uphoff, Jon Vincent, Teemu Virolainen, Christian Wennberg, Sebastian Wingbermühle, Maarten Wolf, Artem Zhmurov

Project leaders: Berk Hess, Erik Lindahl

© 1991 – 2000:

Department of Biophysical Chemistry, University of Groningen. Nijenborgh 4, 9747 AG Groningen, The Netherlands.

© 2001 – 2026:

The GROMACS development teams at the Royal Institute of Technology and Uppsala University, Sweden.

This manual is not complete and has no pretension to be so, due to lack of time of the contributors – our first priority is to improve the software. It is worked on continuously, which in some cases might mean the information is not entirely correct.

Comments on form and content are welcome, please send them to the user discussion forum or the developer discussion forum (see our webpage or this section on how to *contribute* (page 742)), or open an issue on our issue tracker. Corrections can also be made in the GROMACS git source repository and uploaded to the GROMACS GitLab.

We release an updated version of the manual whenever we release a new version of the software, so in general it is a good idea to use a manual with the same major and minor release number as your GROMACS installation.

### 5.1.1 Citation information

This is not a release build of GROMACS, so please reference one of the GROMACS papers and the base release of the manual.

However, we prefer that you cite (some of) the GROMACS papers:

- *Bekker et al. (1993)* (page 683)
- *Berendsen et al. (1995)* (page 683)
- *Lindahl et al. (2001)* (page 683)
- *van der Spoel at al. (2005)* (page 683)
- *Hess et al. (2008)* (page 683)
- *Pronk et al. (2013)* (page 683)
- *Pall et al. (2015)* (page 683)
- *Abraham et al. (2015)* (page 683)

when you publish your results. Any future development depends on academic research grants, since the package is distributed as free software!

### 5.1.2 GROMACS is *Free Software*

The entire GROMACS package is available under the GNU Lesser General Public License (LGPL), version 2.1. This means it is free as in free speech, not just that you can use it without paying us money. You can redistribute GROMACS and/or modify it under the terms of the LGPL as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version. For details, check the COPYING file in the source code or consult this page.

The GROMACS source code and selected set of binary packages are available on our homepage, www.gromacs.org. Have fun!

## 5.2 Introduction

### 5.2.1 Computational Chemistry and Molecular Modeling

GROMACS is an engine to perform molecular dynamics simulations and energy minimization. These are two of the many techniques that belong to the realm of computational chemistry and molecular modeling. *Computational chemistry* is just a name to indicate the use of computational techniques in chemistry, ranging from quantum mechanics of molecules to dynamics of large complex molecular aggregates. *Molecular modeling* indicates the general process of describing complex chemical systems in terms of a realistic atomic model, with the goal being to understand and predict macroscopic properties based on detailed knowledge on an atomic scale. Often, molecular modeling is used to design new materials, for which the accurate prediction of physical properties of realistic systems is required.

Macroscopic physical properties can be distinguished by

1. *static equilibrium properties*, such as the binding constant of an inhibitor to an enzyme, the average potential energy of a system, or the radial distribution function of a liquid, and
2. *dynamic or non-equilibrium properties*, such as the viscosity of a liquid, diffusion processes in membranes, the dynamics of phase changes, reaction kinetics, or the dynamics of defects in crystals.

The choice of technique depends on the question asked and on the feasibility of the method to yield reliable results at the present state of the art. Ideally, the (relativistic) time-dependent Schrödinger equation describes the properties of molecular systems with high accuracy, but anything more complex than the equilibrium state of a few atoms cannot be handled at this *ab initio* level. Thus, approximations are necessary; the higher the complexity of a system and the longer the time span of the processes of interest is, the more severe the required approximations are. At a certain point (reached very much earlier than one would wish), the *ab initio* approach must be augmented or replaced by *empirical* parameterization of the model used. Where simulations based on physical principles of atomic interactions still fail due to the complexity of the system, molecular modeling is based entirely on a similarity analysis of known structural and chemical data. The QSAR methods (Quantitative Structure-Activity Relations) and many homology-based protein structure predictions belong to the latter category.

Macroscopic properties are always ensemble averages over a representative statistical ensemble (either equilibrium or non-equilibrium) of molecular systems. For molecular modeling, this has two important consequences:

- The knowledge of a single structure, even if it is the structure of the global energy minimum, is not sufficient. It is necessary to generate a representative ensemble at a given temperature, in order to compute macroscopic properties. But this is not enough to compute thermodynamic equilibrium properties that are based on free energies, such as phase equilibria, binding constants, solubilities, relative stability of molecular conformations, etc. The computation of free energies and thermodynamic potentials requires special extensions of molecular simulation techniques.
- While molecular simulations, in principle, provide atomic details of the structures and motions, such details are often not relevant for the macroscopic properties of interest. This opens the way to simplify the description of interactions and average over irrelevant details. The science of statistical mechanics provides the theoretical framework for such simplifications. There is a hierarchy of methods ranging from considering groups of atoms as one unit, describing motion in a reduced number of collective coordinates, averaging over solvent molecules with potentials of mean force combined with stochastic dynamics *9* (page 683), to *mesoscopic dynamics* describing densities rather than atoms and fluxes as response to thermodynamic gradients rather than velocities or accelerations as response to forces *10* (page 683).

For the generation of a representative equilibrium ensemble two methods are available:

1. *Monte Carlo simulations* and
2. *Molecular Dynamics simulations*.

For the generation of non-equilibrium ensembles and for the analysis of dynamic events, only the second method is appropriate. While Monte Carlo simulations are more simple than MD (they do not require the computation of forces), they do not yield significantly better statistics than MD in a given amount of computer time. Therefore, MD is the more universal technique. If a starting configuration is very far from equilibrium, the forces may be excessively large and the MD simulation may fail. In those cases, a robust *energy minimization* is required. Another reason to perform an energy minimization is the removal of all kinetic energy from the system: if several "snapshots" from dynamic simulations must be compared, energy minimization reduces the thermal noise in the structures and potential energies so that they can be compared better.

### 5.2.2 Molecular Dynamics Simulations

MD simulations solve Newton's equations of motion for a system of $N$ interacting atoms:

$$m_i \frac{\partial^2 \mathbf{r}_i}{\partial t^2} = \mathbf{F}_i, \; i = 1 \ldots N. \tag{5.1}$$

The forces are the negative derivatives of a potential function $V(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$:

$$\mathbf{F}_i = -\frac{\partial V}{\partial \mathbf{r}_i} \tag{5.2}$$

The equations are solved simultaneously in small time steps. The system is followed for some time, taking care that the temperature and pressure remain at the required values, and the coordinates are written to an output file at regular intervals. The coordinates as a function of time represent a *trajectory* of the system. After initial changes, the system will usually reach an *equilibrium state*. By averaging over an equilibrium trajectory, many macroscopic properties can be extracted from the output file.

It is useful at this point to consider the limitations of MD simulations. The user should be aware of those limitations and always perform checks on known experimental properties to assess the accuracy of the simulation. We list the approximations below.

**The simulations are classical**

- Using Newton's equation of motion automatically implies the use of *classical mechanics* to describe the motion of atoms. This is all right for most atoms at normal temperatures, but there are exceptions. Hydrogen atoms are quite light and the motion of protons is sometimes of essential quantum mechanical character. For example, a proton may *tunnel* through a potential barrier in the course of a transfer over a hydrogen bond. Such processes cannot be properly treated by classical dynamics! Helium liquid at low temperature is another example where classical mechanics breaks down. While helium may not deeply concern us, the high frequency vibrations of covalent bonds should make us worry! The statistical mechanics of a classical harmonic oscillator differs appreciably from that of a real quantum oscillator when the resonance frequency $\nu$ approaches or exceeds $k_B T/h$. At room temperature the wavenumber $\sigma = 1/\lambda = \nu/c$ at which $h\nu = k_B T$ is approximately 200 cm$^{-1}$. Thus, all frequencies higher than, say, 100 cm$^{-1}$ may misbehave in classical simulations. This means that practically all bond and bond-angle vibrations are suspect, and even hydrogen-bonded motions as translational or librational H-bond vibrations are beyond the classical limit (see Table 5.1) What can we do?

Table 5.1: Typical vibrational frequencies (wavenumbers) in molecules and hydrogen-bonded liquids. Compare $kT/h = 200\ \mathrm{cm}^{-1}$ at 300 K.

| type of bond | type of vibration | wavenumber $\mathrm{cm}^{-1}$ |
|---|---|---|
| C-H, O-H, N-H | stretch | 3000–3500 |
| C=C, C=O | stretch | 1700–2000 |
| HOH | bending | 1600 |
| C-C | stretch | 1400–1600 |
| $H_2CX$ | sciss, rock | 1000–1500 |
| CCC | bending | 800–1000 |
| O-H$\cdots$O | vibration | 400–700 |
| O-H$\cdots$O | stretch | 50–200 |

- Well, apart from real quantum-dynamical simulations, we can do one of two things:

  (a) If we perform MD simulations using harmonic oscillators for bonds, we should make corrections to the total internal energy $U = E_{kin} + E_{pot}$ and specific heat $C_V$ (and to entropy $S$ and free energy $A$ or $G$ if those are calculated). The corrections to the energy and specific heat of a one-dimensional oscillator with frequency $\nu$ are: *11* (page 683)

$$U^{QM} = U^{cl} + kT\left(\frac{1}{2}x - 1 + \frac{x}{e^x - 1}\right) \tag{5.3}$$

$$C_V^{QM} = C_V^{cl} + k\left(\frac{x^2 e^x}{(e^x - 1)^2} - 1\right) \tag{5.4}$$

  where $x = h\nu/kT$. The classical oscillator absorbs too much energy ($kT$), while the high-frequency quantum oscillator is in its ground state at the zero-point energy level of $\frac{1}{2}h\nu$.

  (b) We can treat the bonds (and bond angles) as *constraints* in the equations of motion. The rationale behind this is that a quantum oscillator in its ground state resembles a constrained bond more closely than a classical oscillator. A good practical reason for this choice is that the algorithm can use larger time steps when the highest frequencies are removed. In practice the time step can be made four times as large when bonds are constrained than when they are oscillators *12* (page 683). GROMACS has this option for the bonds and bond angles. The flexibility of the latter is rather essential to allow for the realistic motion and coverage of configurational space *13* (page 683).

**Electrons are in the ground state**

In MD simulations, we use a *conservative* force field that is a function of the positions of atoms only. This means that the electronic motions are not considered: the electrons are supposed to adjust their dynamics instantly when the atomic positions change (the *Born-Oppenheimer* approximation), and remain in their ground state. This is really all right, almost always. But of course, electron transfer processes and electronically excited states can not be treated. Neither can chemical reactions be treated properly, but there are other reasons to shy away from reactions for the time being.

**Force fields are approximate**

Force fields provide the forces. They are not really a part of the simulation method and their parameters can be modified by the user as the need arises or knowledge improves. But the form of the forces that can be used in a particular program is subject to limitations. The force field that is incorporated in GROMACS is described in Chapter 4. In the present version the force field is pair-additive (apart from long-range Coulomb forces), it cannot incorporate polarizabilities, and

it does not contain fine-tuning of bonded interactions. This urges the inclusion of some limitations in this list below. For the rest, it is quite useful and fairly reliable for biologically-relevant macromolecules in aqueous solution!

**The force field is pair-additive**
: This means that all *non-bonded* forces result from the sum of non-bonded pair interactions. Non pair-additive interactions, the most important example of which is interaction through atomic polarizability, are represented by *effective pair potentials*. Only average non pair-additive contributions are incorporated. This also means that the pair interactions are not pure, *i.e.*, they are not valid for isolated pairs or for situations that differ appreciably from the test systems on which the models were parameterized. In fact, the effective pair potentials are not that bad in practice. But the omission of polarizability also means that electrons in atoms do not provide a dielectric constant as they should. For example, real liquid alkanes have a dielectric constant of slightly more than 2, which reduce the long-range electrostatic interaction between (partial) charges. Thus, the simulations will exaggerate the long-range Coulomb terms. Luckily, the next item compensates this effect a bit.

**Long-range interactions are cut off**
: GROMACS commonly uses a cut-off radius for the Lennard-Jones interactions and sometimes for the Coulomb interactions as well. The "minimum-image convention" used by GROMACS requires that only one image of each particle in the periodic boundary conditions is considered for a pair interaction, so the cut-off radius cannot exceed half the box size. That is still pretty big for large systems, and trouble is only expected for systems containing charged particles. But then truly bad things can happen, like accumulation of charges at the cut-off boundary or very wrong energies! For such systems, you should consider using one of the implemented long-range electrostatic algorithms, such as particle-mesh Ewald *14* (page 683), *15* (page 683).

**Boundary conditions are unnatural**
: Since system size is small (even 100,000 particles is small), a cluster of particles will have a lot of unwanted boundary with its environment (vacuum). We must avoid this condition if we wish to simulate a bulk system. As such, we use periodic boundary conditions to avoid real phase boundaries. Since liquids are not crystals, something unnatural remains. This item is mentioned last because it is the least of the evils. For large systems, the errors are small, but for small systems with a lot of internal spatial correlation, the periodic boundaries may enhance internal correlation. In that case, beware of, and test, the influence of system size. This is especially important when using lattice sums for long-range electrostatics, since these are known to sometimes introduce extra ordering.

### 5.2.3 Energy Minimization and Search Methods

As mentioned in *Computational Chemistry and Molecular Modeling* (page 420), in many cases energy minimization is required. GROMACS provides a number of methods for local energy minimization, as detailed in *Energy Minimization* (page 467).

The potential energy function of a (macro)molecular system is a very complex landscape (or *hypersurface*) in a large number of dimensions. It has one deepest point, the *global minimum* and a very large number of *local minima*, where all derivatives of the potential energy function with respect to the coordinates are zero and all second derivatives are non-negative. The matrix of second derivatives, which is called the *Hessian matrix*, has non-negative eigenvalues; only the collective coordinates that correspond to translation and rotation (for an isolated molecule) have zero eigenvalues. In between the local minima there are *saddle points*, where the Hessian matrix has only one negative eigenvalue. These points are the mountain passes through which the system can migrate from one local minimum to another.

Knowledge of all local minima, including the global one, and of all saddle points would enable us to describe the relevant structures and conformations and their free energies, as well as the dynamics of structural transitions. Unfortunately, the dimensionality of the configurational space and the number of local minima is so high that it is impossible to sample the space at a sufficient number of points to obtain a complete survey. In particular, no minimization method exists that guarantees the determination of the global minimum in any practical amount of time. Impractical methods exist, some much faster than others *16* (page 683). However, given a starting configuration, it is possible to find the *nearest local minimum*. "Nearest" in this context does not always imply "nearest" in a geometrical sense (*i.e.*, the least sum of square coordinate differences), but means the minimum that can be reached by systematically moving down the steepest local gradient. Finding this nearest local minimum is all that GROMACS can do for you, sorry! If you want to find other minima and hope to discover the global minimum in the process, the best advice is to experiment with temperature-coupled MD: run your system at a high temperature for a while and then quench it slowly down to the required temperature; do this repeatedly! If something as a melting or glass transition temperature exists, it is wise to stay for some time slightly below that temperature and cool down slowly according to some clever scheme, a process called *simulated annealing*. Since no physical truth is required, you can use your imagination to speed up this process. One trick that often works is to make hydrogen atoms heavier (mass 10 or so): although that will slow down the otherwise very rapid motions of hydrogen atoms, it will hardly influence the slower motions in the system, while enabling you to increase the time step by a factor of 3 or 4. You can also modify the potential energy function during the search procedure, *e.g.* by removing barriers (remove dihedral angle functions or replace repulsive potentials by *soft-core* potentials *17* (page 683)), but always take care to restore the correct functions slowly. The best search method that allows rather drastic structural changes is to allow excursions into four-dimensional space *18* (page 683), but this requires some extra programming beyond the standard capabilities of GROMACS.

Three possible energy minimization methods are:

- Those that require only function evaluations. Examples are the simplex method and its variants. A step is made on the basis of the results of previous evaluations. If derivative information is available, such methods are inferior to those that use this information.
- Those that use derivative information. Since the partial derivatives of the potential energy with respect to all coordinates are known in MD programs (these are equal to minus the forces) this class of methods is very suitable as modification of MD programs.
- Those that use second derivative information as well. These methods are superior in their convergence properties near the minimum: a quadratic potential function is minimized in one step! The problem is that for $N$ particles a $3N \times 3N$ matrix must be computed, stored, and inverted. Apart from the extra programming to obtain second derivatives, for most systems of interest this is beyond the available capacity. There are intermediate methods that build up the Hessian matrix on the fly, but they also suffer from excessive storage requirements. So GROMACS will shy away from this class of methods.

The *steepest descent* method, available in GROMACS, is of the second class. It simply takes a step in the direction of the negative gradient (hence in the direction of the force), without any consideration of the history built up in previous steps. The step size is adjusted such that the search is fast, but the motion is always downhill. This is a simple and sturdy, but somewhat stupid, method: its convergence can be quite slow, especially in the vicinity of the local minimum! The faster-converging *conjugate gradient method* (see *e.g.* *19* (page 684)) uses gradient information from previous steps. In general, steepest descents will bring you close to the nearest local minimum very quickly, while conjugate gradients brings you *very* close to the local minimum, but performs worse far away from the minimum. GROMACS also supports the L-BFGS minimizer, which is mostly comparable to *conjugate gradient method*, but in some cases converges faster.

## 5.3 Definitions and Units

### 5.3.1 Notation

The following conventions for mathematical typesetting are used throughout this document:

| Item | Notation | Example |
|---|---|---|
| Vector | Bold italic | $\mathbf{r}_i$ |
| Vector Length | Italic | $r_i$ |

We define the *lowercase* subscripts $i$, $j$, $k$ and $l$ to denote particles: $\mathbf{r}_i$ is the *position vector* of particle $i$, and using this notation:

$$\begin{aligned} \mathbf{r}_{ij} &= \mathbf{r}_j - \mathbf{r}_i \\ r_{ij} &= |\mathbf{r}_{ij}| \end{aligned} \tag{5.5}$$

The force on particle $i$ is denoted by $\mathbf{F}_i$ and

$$\mathbf{F}_{ij} = \text{force on } i \text{ exerted by } j \tag{5.6}$$

### 5.3.2 MD units

GROMACS uses a consistent set of units that produce values in the vicinity of unity for most relevant molecular quantities. Let us call them *MD units*. The basic units in this system are nm, ps, K, electron charge (e) and atomic mass unit (u), see Table 5.2 The values used in GROMACS are taken from the CODATA Internationally recommended 2010 values of fundamental physical constants (see NIST homepage).

Table 5.2: Basic units used in GROMACS

| Quantity | Symbol | Unit |
|---|---|---|
| length | r | nm $=10^{-9}\ m$ |
| mass | m | u (unified atomic mass unit) = $1.660\,538\,921 \times 10^{-27}\ kg$ |
| time | t | ps $=10^{-12}\ s$ |
| charge | q | $e$ = elementary charge = $1.602\,176\,565 \times 10^{-19}\ C$ |
| temperature | T | K |

Consistent with these units are a set of derived units, given in Table 5.3

Table 5.3: Derived units. Note that an additional conversion factor of $10^{28}$ a.m.u ($\approx 16.6$) is applied to get bar instead of internal MD units in the energy and log files

| Quantity | Symbol | Unit |
|---|---|---|
| energy | $E, V$ | $\mathrm{kJ\ mol^{-1}}$ |
| Force | $\mathbf{F}$ | $\mathrm{kJ\ mol^{-1}\ nm^{-1}}$ |
| pressure | $p$ | bar |
| velocity | $v$ | $\mathrm{nm\ ps^{-1}} = 1000\ \mathrm{m\ s^{-1}}$ |
| dipole moment | $\mu$ | $\mathrm{e\ nm}$ |
| electric potential | $\Phi$ | $\mathrm{kJ\ mol^{-1}\ e^{-1}} = 0.010\,364\,269\,19$ Volt |
| electric field | $E$ | $\mathrm{kJ\ mol^{-1}\ nm^{-1}\ e^{-1}} = 1.036\,426\,919 \times 10^{7}\ \mathrm{Vm^{-1}}$ |

The **electric conversion factor** $f = \frac{1}{4\pi\varepsilon_o} = 138.935\,458\ \mathrm{kJ\ mol^{-1} nm\ e^{-2}}$. It relates the mechanical quantities to the electrical quantities as in

$$V = f\frac{q^2}{r} \quad \text{or} \quad F = f\frac{q^2}{r^2} \tag{5.7}$$

Electric potentials $\Phi$ and electric fields $\mathbf{E}$ are intermediate quantities in the calculation of energies and forces. They do not occur inside GROMACS. If they are used in evaluations, there is a choice of equations and related units. We strongly recommend following the usual practice of including the factor $f$ in expressions that evaluate $\Phi$ and $\mathbf{E}$:

$$\begin{aligned} \Phi(\mathbf{r}) &= f\sum_j \frac{q_j}{|\mathbf{r}-\mathbf{r}_j|} \\ \mathbf{E}(\mathbf{r}) &= f\sum_j q_j \frac{(\mathbf{r}-\mathbf{r}_j)}{|\mathbf{r}-\mathbf{r}_j|^3} \end{aligned} \tag{5.8}$$

With these definitions, $q\Phi$ is an energy and $q\mathbf{E}$ is a force. The units are those given in Table 5.3 about 10 mV for potential. Thus, the potential of an electronic charge at a distance of 1 nm equals $f \approx 140$ units $\approx 1.4$ V. (exact value: $1.439\,964\,5$ V)

**Note** that these units are mutually consistent; changing any of the units is likely to produce inconsistencies and is therefore *strongly discouraged*! In particular: if Å are used instead of nm, the unit of time changes to 0.1 ps. If $\mathrm{kcal\ mol^{-1}}$ (= 4.184 $\mathrm{kJ\ mol^{-1}}$) is used instead of $\mathrm{kJ\ mol^{-1}}$ for energy, the unit of time becomes 0.488882 ps and the unit of temperature changes to 4.184 K. But in both cases all electrical energies go wrong, because they will still be computed in $\mathrm{kJ\ mol^{-1}}$, expecting nm as the unit of length. Although careful rescaling of charges may still yield consistency, it is clear that such confusions must be rigidly avoided.

In terms of the MD units, the usual physical constants take on different values (see Table 5.4). All quantities are per mol rather than per molecule. There is no distinction between Boltzmann's constant $k$ and the gas constant $R$: their value is $0.008\,314\,462\,1\mathrm{kJ\ mol^{-1}K^{-1}}$.

Table 5.4: Some Physical Constants

| Symbol | Name | Value |
|---|---|---|
| $N_{AV}$ | Avogadro's number | $6.022\,141\,29 \times 10^{23}$ mol$^{-1}$ |
| $R$ | gas constant | $8.314\,462\,1 \times 10^{-3}$ kJ mol$^{-1}$ K$^{-1}$ |
| $k_B$ | Boltzmann's constant | *idem* |
| $h$ | Planck's constant | $0.399\,031\,271$ kJ mol$^{-1}$ ps |
| $\hbar$ | Dirac's constant | $0.063\,507\,799\,3$ kJ mol$^{-1}$ ps |
| $c$ | velocity of light | $299\,792.458$ nm ps$^{-1}$ |

### 5.3.3 Reduced units

When simulating Lennard-Jones (LJ) systems, it might be advantageous to use reduced units (*i.e.*, setting $\epsilon_{ii} = \sigma_{ii} = m_i = k_B = 1$ for one type of atoms). This is possible. When specifying the input in reduced units, the output will also be in reduced units. The one exception is the *temperature*, which is expressed in $0.008\,314\,462\,1$ reduced units. This is a consequence of using Boltzmann's constant in the evaluation of temperature in the code. Thus not $T$, but $k_BT$, is the reduced temperature. A GROMACS temperature $T = 1$ means a reduced temperature of $0.008\ldots$ units; if a reduced temperature of 1 is required, the GROMACS temperature should be $120.272\,36$.

In Table 5.5 quantities are given for LJ potentials:

$$V_{LJ} = 4\epsilon \left[ \left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{5.9}$$

Table 5.5: Reduced Lennard-Jones quantities

| Quantity | Symbol | Relation to SI |
|---|---|---|
| Length | r* | $r\sigma^{-1}$ |
| Mass | m* | $m\,M^{-1}$ |
| Time | t* | $t\sigma^{-1}\sqrt{\epsilon/M}$ |
| Temperature | T* | $k_BT\,\epsilon^{-1}$ |
| Energy | E* | $E\epsilon^{-1}$ |
| Force | F* | $F\sigma\,\epsilon^{-1}$ |
| Pressure | P* | $P\sigma^3\epsilon^{-1}$ |
| Velocity | v* | $v\sqrt{M/\epsilon}$ |
| Density | $\rho^*$ | $N\sigma^3\,V^{-1}$ |

### 5.3.4 Mixed or Double precision

GROMACS can be compiled in either mixed or double precision. Documentation of previous GROMACS versions referred to *single precision*, but the implementation has made selective use of double precision for many years. Using single precision for all variables would lead to a significant reduction in accuracy. Although in *mixed precision* all state vectors, i.e. particle coordinates, velocities and forces, are stored in single precision, critical variables are double precision. A typical example of the latter is the virial, which is a sum over all forces in the system, which have varying signs. In addition, in many parts of the code we managed to avoid double precision for arithmetic, by paying attention to summation order or reorganization of mathematical expressions. The default configuration uses mixed precision, but it is easy to turn on double precision by adding the option `-DGMX_DOUBLE=on` to `cmake`. Double

precision will be 20 to 100% slower than mixed precision depending on the architecture you are running on. Double precision will use somewhat more memory and run input, energy and full-precision trajectory files will be almost twice as large.

The energies in mixed precision are accurate up to the last decimal, the last one or two decimals of the forces are non-significant. The virial is less accurate than the forces, since the virial is only one order of magnitude larger than the size of each element in the sum over all atoms (sec. *Virial and pressure* (page 529)). In most cases this is not really a problem, since the fluctuations in the virial can be two orders of magnitude larger than the average. Using cut-offs for the Coulomb interactions cause large errors in the energies, forces, and virial. Even when using a reaction-field or lattice sum method, the errors are larger than, or comparable to, the errors due to the partial use of single precision. Since MD is chaotic, trajectories with very similar starting conditions will diverge rapidly, the divergence is faster in mixed precision than in double precision.

For most simulations, mixed precision is accurate enough. In some cases double precision is required to get reasonable results:

- normal mode analysis, for the conjugate gradient or l-bfgs minimization and the calculation and diagonalization of the Hessian
- long-term energy conservation, especially for large systems

## 5.4 Algorithms

In this chapter we first give describe some general concepts used in GROMACS: *periodic boundary conditions* (sec. *Periodic boundary conditions* (page 429)) and the *group concept* (sec. *The group concept* (page 432)). The MD algorithm is described in sec. *Molecular Dynamics* (page 433): first a global form of the algorithm is given, which is refined in subsequent subsections. The (simple) EM (Energy Minimization) algorithm is described in sec. *Energy Minimization* (page 467). Some other algorithms for special purpose dynamics are described after this.

A few issues are of general interest. In all cases the *system* must be defined, consisting of molecules. Molecules again consist of particles with defined interaction functions. The detailed description of the *topology* of the molecules and of the *force field* and the calculation of forces is given in chapter *Interaction function and force fields* (page 481). In the present chapter we describe other aspects of the algorithm, such as pair list generation, update of velocities and positions, coupling to external temperature and pressure, conservation of constraints. The *analysis* of the data generated by an MD simulation is treated in chapter *Analysis* (page 653).

### 5.4.1 Periodic boundary conditions

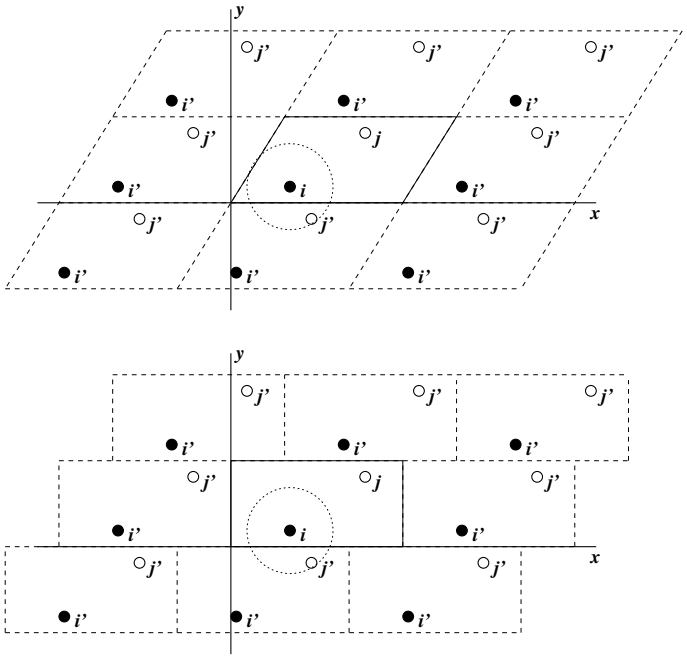



Fig. 5.1: Periodic boundary conditions in two dimensions.

The classical way to minimize edge effects in a finite system is to apply *periodic boundary conditions*. The atoms of the system to be simulated are put into a space-filling box, which is surrounded by translated copies of itself (Fig. 5.1). Thus there are no boundaries of the system; the artifact caused by unwanted boundaries in an isolated cluster is now replaced by the artifact of periodic conditions. If the system is crystalline, such boundary conditions are desired (although motions are naturally restricted to periodic motions with wavelengths fitting into the box). If one wishes to simulate non-periodic systems, such as liquids or solutions, the periodicity by itself causes errors. The errors can be evaluated by comparing various system sizes; they are expected to be less severe than the errors resulting from an unnatural boundary with vacuum.

There are several possible shapes for space-filling unit cells. Some, like the *rhombic dodecahedron* and

the *truncated octahedron 20* (page 684) are closer to being a sphere than a cube is, and are therefore better suited to the study of an approximately spherical macromolecule in solution, since fewer solvent molecules are required to fill the box given a minimum distance between macromolecular images. At the same time, rhombic dodecahedra and truncated octahedra are special cases of *triclinic* unit cells; the most general space-filling unit cells that comprise all possible space-filling shapes *21* (page 684). For this reason, GROMACS is based on the triclinic unit cell.

GROMACS uses periodic boundary conditions, combined with the *minimum image convention*: only one – the nearest – image of each particle is considered for short-range non-bonded interaction terms. For long-range electrostatic interactions this is not always accurate enough, and GROMACS therefore also incorporates lattice sum methods such as Ewald Sum, PME and PPPM.

GROMACS supports triclinic boxes of any shape. The simulation box (unit cell) is defined by the 3 box vectors $\mathbf{a}$,$\mathbf{b}$ and $\mathbf{c}$. The box vectors must satisfy the following conditions:

$$a_y = a_z = b_z = 0 \tag{5.10}$$

$$a_x > 0, \quad b_y > 0, \quad c_z > 0 \tag{5.11}$$

$$|b_x| \leq \frac{1}{2}\, a_x, \quad |c_x| \leq \frac{1}{2}\, a_x, \quad |c_y| \leq \frac{1}{2}\, b_y \tag{5.12}$$

Equations (5.10) can always be satisfied by rotating the box. Inequalities ((5.11)) and ((5.12)) can always be satisfied by adding and subtracting box vectors.

Even when simulating using a triclinic box, GROMACS always keeps the particles in a brick-shaped volume for efficiency, as illustrated in Fig. 5.1 for a 2-dimensional system. Therefore, from the output trajectory it might seem that the simulation was done in a rectangular box. The program *trjconv* (page 334) can be used to convert the trajectory to a different unit-cell representation.

It is also possible to simulate without periodic boundary conditions, but it is usually more efficient to simulate an isolated cluster of molecules in a large periodic box, since fast grid searching can only be used in a periodic system.

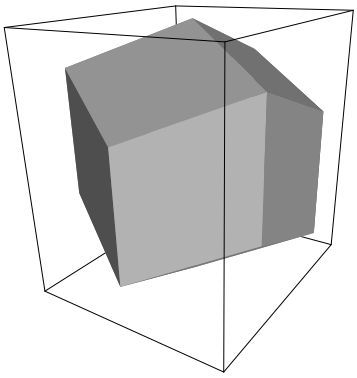


Fig. 5.2: A rhombic dodecahedron (arbitrary orientation).

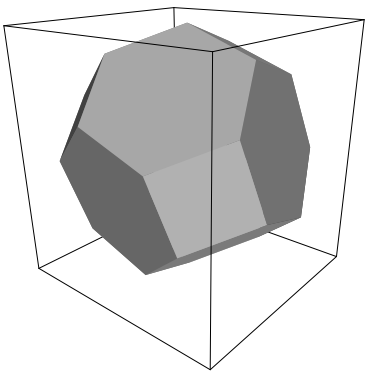


Fig. 5.3: A truncated octahedron (arbitrary orientation).

### Some useful box types

Table 5.6: Overview over different box types

| box type | image distance | box volume | box vectors **a** | **b** | **c** | box vector angles ∠ **bc** | ∠ **ac** | ∠ **ab** |
|---|---|---|---|---|---|---|---|---|
| cubic | $d$ | $d^3$ | $d$ | 0 | 0 | 90° | 90° | 90° |
| | | | 0 | $d$ | 0 | | | |
| | | | 0 | 0 | $d$ | | | |
| rhombic | $d$ | $\frac{1}{2}\sqrt{2}\,d^3$ | $d$ | 0 | $\frac{1}{2}\,d$ | 60° | 60° | 90° |
| dodecahedron | | $0.707\,d^3$ | 0 | $d$ | $\frac{1}{2}\,d$ | | | |
| (xy-square) | | | 0 | 0 | $\frac{1}{2}\sqrt{2}\,d$ | | | |
| rhombic | $d$ | $\frac{1}{2}\sqrt{2}\,d^3$ | $d$ | $\frac{1}{2}\,d$ | $\frac{1}{2}\,d$ | 60° | 60° | 60° |
| dodecahedron | | $0.707\,d^3$ | 0 | $\frac{1}{2}\sqrt{3}\,d$ | $\frac{1}{6}\sqrt{3}\,d$ | | | |
| (xy- hexagon) | | | 0 | 0 | $\frac{1}{3}\sqrt{6}\,d$ | | | |
| truncated | $d$ | $\frac{4}{9}\sqrt{3}\,d^3$ | $d$ | $\frac{1}{3}\,d$ | $-\frac{1}{3}\,d$ | 70.53° | 109.47° | 70.53° |
| octahedron | | $0.770\,d^3$ | 0 | $\frac{2}{3}\sqrt{2}\,d$ | $\frac{1}{3}\sqrt{2}\,d$ | | | |
| | | | 0 | 0 | $\frac{1}{3}\sqrt{6}\,d$ | | | |

The three most useful box types for simulations of solvated systems are described in Table 5.6. The rhombic dodecahedron (Fig. 5.2) is the smallest and most regular space-filling unit cell. Each of the 12 image cells is at the same distance. The volume is 71% of the volume of a cube having the same image distance. This saves about 29% of CPU-time when simulating a spherical or flexible molecule in solvent. There are two different orientations of a rhombic dodecahedron that satisfy equations (5.10), (5.11) and (5.12). The program *editconf* (page 208) produces the orientation which has a square intersection with the xy-plane. This orientation was chosen because the first two box vectors coincide with the x and y-axis, which is easier to comprehend. The other orientation can be useful for simulations of membrane proteins. In this case the cross-section with the xy-plane is a hexagon, which has an area which is 14% smaller than the area of a square with the same image distance. The height of the box ($c_z$) should be changed to obtain an optimal spacing. This box shape not only saves CPU time, it also results in a more uniform arrangement of the proteins.

#### Cut-off restrictions

The minimum image convention implies that the cut-off radius used to truncate non-bonded interactions may not exceed half the shortest box vector:

$$R_c < \frac{1}{2} \min(\|\mathbf{a}\|, \|\mathbf{b}\|, \|\mathbf{c}\|), \tag{5.13}$$

because otherwise more than one image would be within the cut-off distance of the force. When a macromolecule, such as a protein, is studied in solution, this restriction alone is not sufficient: in principle, a single solvent molecule should not be able to 'see' both sides of the macromolecule. This means that the length of each box vector must exceed the length of the macromolecule in the direction of that edge *plus* two times the cut-off radius $R_c$. It is, however, common to compromise in this respect, and make the solvent layer somewhat smaller in order to reduce the computational cost. For efficiency reasons the cut-off with triclinic boxes is more restricted. For grid search the extra restriction is weak:

$$R_c < \min(a_x, b_y, c_z) \tag{5.14}$$

For simple search the extra restriction is stronger:

$$R_c < \frac{1}{2} \min(a_x, b_y, c_z) \tag{5.15}$$

Each unit cell (cubic, rectangular or triclinic) is surrounded by 26 translated images. A particular image can therefore always be identified by an index pointing to one of 27 *translation vectors* and constructed by applying a translation with the indexed vector (see *Compute forces* (page 442)). Restriction (5.14) ensures that only 26 images need to be considered.

### 5.4.2 The group concept

The GROMACS MD and analysis programs use user-defined *groups* of atoms to perform certain actions on. The maximum number of groups is 256, but each atom can only belong to six different groups, one each of the following:

**temperature-coupling group**
: The temperature coupling parameters (reference temperature, time constant, number of degrees of freedom, see *The leap-frog integrator* (page 443)) can be defined for each T-coupling group separately. For example, in a solvated macromolecule the solvent (that tends to generate more heating by force and integration errors) can be coupled with a shorter time constant to a bath than is a macromolecule, or a surface can be kept cooler than an adsorbing molecule. Many different T-coupling groups may be defined. See also center of mass groups below.

**freeze group**
: Atoms that belong to a freeze group are kept stationary in the dynamics. This is useful during equilibration, *e.g.* to avoid badly placed solvent molecules giving unreasonable kicks to protein atoms, although the same effect can also be obtained by putting a restraining potential on the atoms that must be protected. The freeze option can be used, if desired, on just one or two coordinates of an atom, thereby freezing the atoms in a plane or on a line. When an atom is partially frozen, constraints will still be able to move it, even in a frozen direction. A fully frozen atom can not be moved by constraints. Many freeze groups can be defined. Frozen coordinates are unaffected by pressure scaling; in some cases this can produce unwanted results, particularly when constraints are also used (in this case you will get very large pressures). Accordingly, it is recommended to avoid combining freeze groups with constraints and pressure coupling. For the sake of equilibration it could suffice to start with freezing in a constant volume simulation, and afterward use position restraints in conjunction with constant pressure.

**accelerate group**
: On each atom in an "accelerate group" an acceleration $\mathbf{a}^g$ is imposed. This is equivalent to a mass-weighted external force. This feature makes it possible to drive the system into a non-equilibrium state to compute, for example, transport properties.

**energy-monitor group**
: Mutual interactions between all energy-monitor groups are compiled during the simulation. This is done separately for Lennard-Jones and Coulomb terms. In principle up to 256 groups could be defined, but that would lead to 256×256 items! Better use this concept sparingly.

  All non-bonded interactions between pairs of energy-monitor groups can be excluded (see details in the User Guide). Pairs of particles from excluded pairs of energy-monitor groups are not put into the pair list. This can result in a significant speedup for simulations where interactions within or between parts of the system are not required.

**center of mass group**
: In GROMACS, the center of mass (COM) motion can be removed, for either the complete system or for groups of atoms. The latter is useful, *e.g.* for systems where there is limited friction (*e.g.* gas systems) to prevent center of mass motion to occur. It makes sense to use the same groups for temperature coupling and center of mass motion removal.

**Compressed position output group**
: In order to further reduce the size of the compressed trajectory file (*xtc* (page 588) or *tng* (page 584)), it is possible to store only a subset of all particles. All x-compression groups that are specified are saved, the rest are not. If no such groups are specified, than all atoms are saved to the compressed trajectory file.

The use of groups in GROMACS tools is described in sec. *Using Groups* (page 653).

### 5.4.3 Molecular Dynamics

**THE GLOBAL MD ALGORITHM**

---

**1. Input initial conditions**
Potential interaction $V$ as a function of atom positions
Positions $\mathbf{r}$ of all atoms in the system
Velocities $\mathbf{v}$ of all atoms in the system
⇓

---

**repeat 2,3,4** for the required number of steps:

---

**2. Compute forces**

The force on any atom

$\mathbf{F}_i = -\frac{\partial V}{\partial \mathbf{r}_i}$

is computed by calculating the force between non-bonded atom pairs:

$\mathbf{F}_i = \sum_j \mathbf{F}_{ij}$

plus the forces due to bonded interactions (which may depend on 1, 2, 3, or 4 atoms), plus restraining and/or external forces.

The potential and kinetic energies and the pressure tensor may be computed.

⇓

**3. Update configuration**

The movement of the atoms is simulated by numerically solving Newton's equations of motion

$\frac{\mathrm{d}^2\mathbf{r}_i}{\mathrm{d}t^2} = \frac{\mathbf{F}_i}{m_i}$

or

$\frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} = \mathbf{v}_i; \quad \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = \frac{\mathbf{F}_i}{m_i}$

⇓

**4.** if required: **Output step**

write positions, velocities, energies, temperature, pressure, etc.

A global flow scheme for MD is given above. Each MD or EM run requires as input a set of initial coordinates and – optionally – initial velocities of all particles involved. This chapter does not describe how these are obtained; for the setup of an actual MD run check the *User guide* (page 34) in Sections *System preparation* (page 40) and *Getting started* (page 36).

### Initial conditions

### Topology and force field

The system topology, including a description of the force field, must be read in. Force fields and topologies are described in chapter *Interaction function and force fields* (page 481) and *top* (page 584), respectively. All this information is static; it is never modified during the run.

### Coordinates and velocities

Then, before a run starts, the box size and the coordinates and velocities of all particles are required. The box size and shape is determined by three vectors (nine numbers) $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$, which represent the three basis vectors of the periodic box.

If the run starts at $t = t_0$, the coordinates at $t = t_0$ must be known. The *leap-frog algorithm*, the default algorithm used to update the time step with $\Delta t$ (see *The leap-frog integrator* (page 443)), also requires that the velocities at $t = t_0 - \frac{1}{2}\Delta t$ are known. If velocities are not available, the program can generate initial atomic velocities $v_i, i = 1 \ldots 3N$ with a Maxwell-Boltzmann distribution (Fig. 5.4) at a given absolute temperature $T$:

$$p(v_i) = \sqrt{\frac{m_i}{2\pi kT}} \exp\left(-\frac{m_i v_i^2}{2kT}\right) \tag{5.16}$$

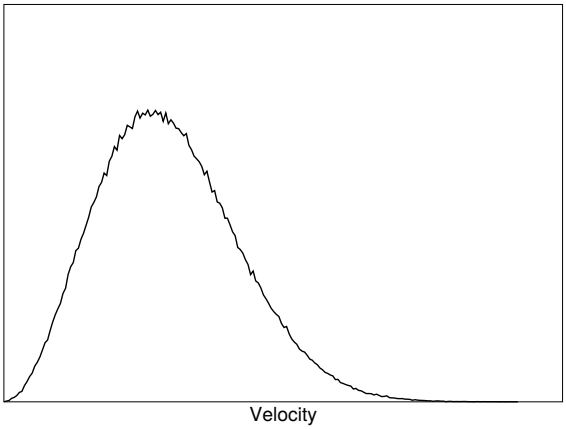



Fig. 5.4: A Maxwell-Boltzmann velocity distribution, generated from random numbers.

where $k$ is Boltzmann's constant (see chapter *Definitions and Units* (page 425)). To accomplish this, normally distributed random numbers are generated by adding twelve random numbers $R_k$ in the range $0 \leq R_k < 1$ and subtracting 6.0 from their sum. The result is then multiplied by the standard deviation of the velocity distribution $\sqrt{kT/m_i}$. Since the resulting total energy will not correspond exactly to the required temperature $T$, a correction is made: first the center-of-mass motion is removed and then all velocities are scaled so that the total energy corresponds exactly to $T$ (see (5.21)).

### Center-of-mass motion

The center-of-mass velocity is normally set to zero at every step; there is (usually) no net external force acting on the system and the center-of-mass velocity should remain constant. In practice, however, the update algorithm introduces a very slow change in the center-of-mass velocity, and therefore in the total kinetic energy of the system – especially when temperature coupling is used. If such changes are not quenched, an appreciable center-of-mass motion can develop in long runs, and the temperature will be significantly misinterpreted. Something similar may happen due to overall rotational motion, but only when an isolated cluster is simulated. In periodic systems with filled boxes, the overall rotational motion is coupled to other degrees of freedom and does not cause such problems.

### Neighbor searching

As mentioned in chapter *Interaction function and force fields* (page 481), internal forces are either generated from fixed (static) lists, or from dynamic lists. The latter consist of non-bonded interactions between any pair of particles. When calculating the non-bonded forces, it is convenient to have all particles in a rectangular box. As shown in Fig. 5.1, it is possible to transform a triclinic box into a rectangular box. The output coordinates are always in a rectangular box, even when a dodecahedron or triclinic box was used for the simulation. (5.10) ensures that we can reset particles in a rectangular box by first shifting them with box vector $\mathbf{c}$, then with $\mathbf{b}$ and finally with $\mathbf{a}$. Equations (5.12), (5.13) and (5.14) ensure that we can find the 14 nearest triclinic images within a linear combination that does not involve multiples of box vectors.

### Pair lists generation

The non-bonded pair forces need to be calculated only for those pairs $i, j$ for which the distance $r_{ij}$ between $i$ and the nearest image of $j$ is less than a given cut-off radius $R_c$. Some of the particle pairs that fulfill this criterion are excluded, when their interaction is already fully accounted for by bonded interactions. But for most electrostatic treatments, correction forces also need to be computed for such excluded atom pairs. GROMACS employs a *pair list* that contains those particle pairs for which non-bonded forces must be calculated. The pair list contains particles $i$, a displacement vector for particle $i$, and all particles $j$ that are within `rlist` of this particular image of particle $i$. The list is updated every `nstlist` steps.

To make the pair list, all atom pairs that are within the pair-list cut-off distance need to be found and stored in a list. Note that such a list generally does not store all neighbors for each atom, since each atom pair should appear only once in the list. This searching, usually called neighbor search (NS) or pair search, involves periodic boundary conditions and determining the *image* (see sec. *Periodic boundary conditions* (page 429)). The search algorithm employed in GROMACS is $O(N)$.

As pair searching is an expensive operation, a generated pair list is retained for a certain number of integration steps. A buffer is needed to account for relative displacements of atoms over the steps where a fixed pair list is retained. GROMACS uses a buffered pair list by default. It also uses clusters of particles, but these are not static as in the old charge group scheme. Rather, the clusters are defined spatially and consist of 4 or 8 particles, which is convenient for streaming-style computing on parallel hardware in CPUs and GPUs. At neighbor search steps, a pair list is created with a Verlet buffer, i.e. the pair-list cut-off is larger than the interaction cut-off. In the non-bonded kernels, interactions are only computed when a particle pair is within the cut-off distance at that particular time step. This ensures that as particles move between pair search steps, forces between nearly all particles within the cut-off distance are calculated. We say *nearly* all particles, because GROMACS uses a fixed pair-list update frequency for efficiency. A particle-pair, whose distance was outside the cut-off, could possibly move enough during this fixed number of steps that its distance is now within the cut-off. This small chance results in a small energy drift, and the size of the chance depends on the temperature. When temperature coupling is used, the buffer size can be determined automatically, given a certain tolerance on the energy drift. The default tolerance is 0.005 kJ/mol/ps per particle, but in practice the energy drift is usually an order of magnitude smaller. Note that in single precision for normal atomistic simulations constraints cause a drift somewhere around 0.0001 kJ/mol/ps per particle, so it does not make sense to go much lower than that.

The pair list is implemented in a very efficient fashion based on clusters of particles. The simplest example is a cluster size of 4 particles. The pair list is then constructed based on cluster pairs. The cluster-pair search is much faster searching based on particle pairs, because $4 \times 4 = 16$ particle pairs are put in the list at once. The non-bonded force calculation kernel can then calculate many particle-pair interactions at once, which maps nicely to SIMD or SIMT units on modern hardware, which can perform multiple floating operations at once. These non-bonded kernels are much faster than the kernels used in the group scheme for most types of systems, particularly on newer hardware. For further information on algorithmic and implementation details of the Verlet cut-off scheme and the NxM kernels, as well as detailed performance analysis, please consult the following article: *182* (page 692).

Additionally, when the list buffer is determined automatically as described below, we also apply dynamic pair list pruning. The pair list can be constructed infrequently, but that can lead to a lot of pairs in the list that are outside the cut-off range for all or most of the life time of this pair list. Such pairs can be pruned out by applying a cluster-pair kernel that only determines which clusters are in range. Because of the way the non-bonded data is regularized in GROMACS, this kernel is an order of magnitude faster than the search and the interaction kernel. On the GPU this pruning can be overlapped with the integration on the CPU, so it is free in some cases. Therefore we can prune every 4-10 integration steps with little

overhead and significantly reduce the number of cluster pairs in the interaction kernel. This procedure is applied automatically, unless the user set the pair-list buffer size manually.

### Energy drift and pair list buffering

For a canonical (NVT) ensemble, the average energy error caused by diffusion of $j$ particles from outside the pair-list cut-off $r_\ell$ to inside the interaction cut-off $r_c$ over the lifetime of the list can be determined from the atomic displacements and the shape of the potential at the cut-off. The displacement distribution along one dimension for a freely moving particle with mass $m$ over time $t$ at temperature $T$ is a Gaussian $G(x)$ of zero mean and variance $\sigma^2 = t^2 k_B T/m$. For the distance between two particles, the variance changes to $\sigma^2 = \sigma_{12}^2 = t^2 k_B T(1/m_1 + 1/m_2)$. Note that in practice particles usually interact with (bump into) other particles over time $t$ and therefore the real displacement distribution is much narrower. Given a non-bonded interaction cut-off distance of $r_c$ and a pair-list cut-off $r_\ell = r_c + r_b$ for the Verlet buffer size $r_b$, we can then write the average energy error after time $t$ for all missing pair interactions between a single $i$ particle of type 1 surrounded by all $j$ particles that are of type 2 with number density $\rho_2$, when the inter-particle distance changes from $r_0$ to $r_t$, as:

$$\langle \Delta V \rangle = \int_0^{r_c} \int_{r_\ell}^{\infty} 4\pi r_0^2 \rho_2 V(r_t) G\left(\frac{r_t - r_0}{\sigma}\right) dr_0\, dr_t \tag{5.17}$$

To evaluate this analytically, we need to make some approximations. First we replace $V(r_t)$ by a Taylor expansion around $r_c$, then we can move the lower bound of the integral over $r_0$ to $-\infty$ which will simplify the result:

$$\begin{aligned}\langle \Delta V \rangle \approx \int_{-\infty}^{r_c} \int_{r_\ell}^{\infty} 4\pi r_0^2 \rho_2 \Big[ & V(r_c) + \\ & V'(r_c)(r_t - r_c) + \\ & V''(r_c)\frac{1}{2}(r_t - r_c)^2 + \\ & V'''(r_c)\frac{1}{6}(r_t - r_c)^3 + \\ & O\big((r_t - r_c)^4\big) \Big] G\left(\frac{r_t - r_0}{\sigma}\right) dr_0\, dr_t\end{aligned}$$

Replacing the factor $r_0^2$ by $(r_\ell + \sigma)^2$, which results in a slight overestimate, allows us to calculate the integrals analytically:

$$\begin{aligned}
\langle \Delta V \rangle \approx & \; 4\pi(r_\ell+\sigma)^2\rho_2 \int_{-\infty}^{r_c}\int_{r_\ell}^{\infty} \Big[V(r_c) + \\
& \qquad V'(r_c)(r_t - r_c) + \\
& \qquad V''(r_c)\frac{1}{2}(r_t - r_c)^2 + \\
& \qquad V'''(r_c)\frac{1}{6}(r_t - r_c)^3\Big] G\Big(\frac{r_t - r_0}{\sigma}\Big)\, dr_0\, dr_t \\
= & \; 4\pi(r_\ell+\sigma)^2\rho_2 \Big\{ V(r_c)\left[\sigma G\Big(\frac{r_b}{\sigma}\Big) - r_b E\Big(\frac{r_b}{\sigma}\Big)\right] + \\
& \qquad \frac{1}{2}V'(r_c)\left[r_b\sigma G\Big(\frac{r_b}{\sigma}\Big) - (r_b^2+\sigma^2)E\Big(\frac{r_b}{\sigma}\Big)\right] + \\
& \qquad \frac{1}{6}V''(r_c)\left[\sigma(r_b^2+2\sigma^2)G\Big(\frac{r_b}{\sigma}\Big) - r_b(r_b^2+3\sigma^2)E\Big(\frac{r_b}{\sigma}\Big)\right] + \\
& \qquad \frac{1}{24}V'''(r_c)\Big[r_b\sigma(r_b^2+5\sigma^2)G\Big(\frac{r_b}{\sigma}\Big) \\
& \qquad - (r_b^4+6r_b^2\sigma^2+3\sigma^4)E\Big(\frac{r_b}{\sigma}\Big)\Big]\Big\}
\end{aligned}$$

where $G(x)$ is a Gaussian distribution with 0 mean and unit variance and $E(x) = \frac{1}{2}\mathrm{erfc}(x/\sqrt{2})$. Note the potential at the cut-off, $V(r_c)$, is zero by definition. But the same formula can be used to estimate errors in the pressure and then the force is used for $V$ in these formulas and this leading term will generally not be zero. We always want to achieve small energy error, so $\sigma$ will be small compared to both $r_c$ and $r_\ell$, thus the approximations in the equations above are good, since the Gaussian distribution decays rapidly. The energy error needs to be averaged over all particle pair types and weighted with the particle counts. In GROMACS we do not allow cancellation of error between pair types, so we average the absolute values. To obtain the average energy error per unit time, it needs to be divided by the neighbor-list life time $t = (\texttt{nstlist} - 1) \times \texttt{dt}$. The function can not be inverted analytically, so we use bisection to obtain the buffer size $r_b$ for a target drift. Again we note that in practice the error we usually be much smaller than this estimate, as in the condensed phase particle displacements will be much smaller than for freely moving particles, which is the assumption used here.

For inhomogeneous systems, using the global atom densities $\rho$ can lead to an underestimate of the energy drift. To avoid that, an effective density is used. This is computed by putting all atoms on a grid where the cells are approximately the size of the cut-off. The densities are then averaged over the cells weighted by the density of each cell. This provides accurate estimates for inhomogeneous systems, e.g. the effective density for a molecule or a droplet in an otherwise empty box does not depend on the size of the box.

When (bond) constraints are present, some particles will have fewer degrees of freedom. This will reduce the energy errors. For simplicity, we only consider one constraint per particle, the heaviest particle in case a particle is involved in multiple constraints. This simplification overestimates the displacement. The motion of a constrained particle is a superposition of the 3D motion of the center of mass of both particles and a 2D rotation around the center of mass. The displacement in an arbitrary direction of a particle with 2 degrees of freedom is not Gaussian, but rather follows the complementary error function:

$$\frac{\sqrt{\pi}}{2\sqrt{2}\sigma}\,\mathrm{erfc}\left(\frac{|r|}{\sqrt{2}\,\sigma}\right) \tag{5.18}$$

where $\sigma^2$ is again $t^2 k_B T/m$. This distribution can no longer be integrated analytically to obtain the energy error. But we can generate a tight upper bound using a scaled and shifted Gaussian distribution (not shown). This Gaussian distribution can then be used to calculate the energy error as described above. The rotation displacement around the center of mass can not be more than the length of the arm. To take this into account, we scale $\sigma$ in (5.18) (details not presented here) to obtain an overestimate of the real displacement. This latter effect significantly reduces the buffer size for longer neighborlist lifetimes in e.g. water, as constrained hydrogens are by far the fastest particles, but they can not move further than 0.1 nm from the heavy atom they are connected to.

There is one important implementation detail that reduces the energy errors caused by the finite Verlet buffer list size. The derivation above assumes a particle pair list. However, the GROMACS implementation uses a cluster pair list for efficiency. The pair list consists of pairs of clusters of 4 particles in most cases, also called a $4 \times 4$ list, but the list can also be $4 \times 8$ (GPU CUDA kernels and AVX 256-bit single precision kernels) or $4 \times 2$ (SSE double-precision kernels). This means that the pair list is effectively much larger than the corresponding $1 \times 1$ list. Thus slightly beyond the pair-list cut-off there will still be a large fraction of particle pairs present in the list. This fraction can be determined in a simulation and accurately estimated under some reasonable assumptions. The fraction decreases with increasing pair-list range, meaning that a smaller buffer can be used. For typical all-atom simulations with a cut-off of 0.9 nm this fraction is around 0.9, which gives a reduction in the energy errors of a factor of 10. This reduction is taken into account during the automatic Verlet buffer calculation and results in a smaller buffer size.

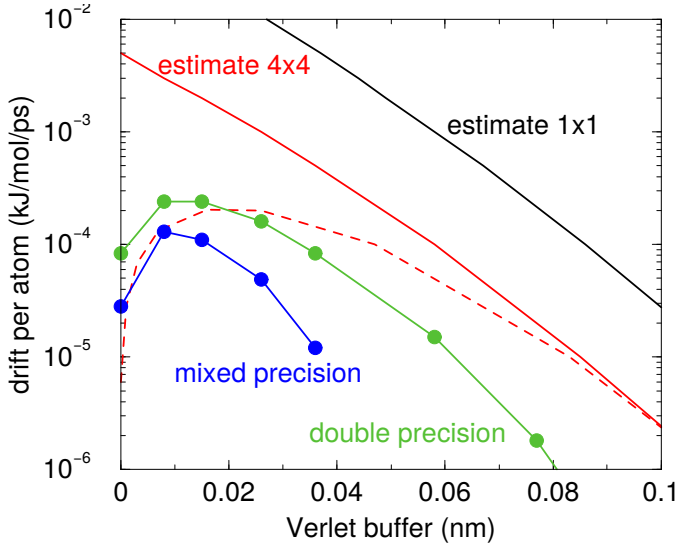



Fig. 5.5: Energy drift per atom for an SPC/E water system at 300K with a time step of 2 fs and a pair-list update period of 10 steps (pair-list life time: 18 fs). PME was used with `ewald-rtol` set to $10^{-5}$; this parameter affects the shape of the potential at the cut-off. Error estimates due to finite Verlet buffer size are shown for a $1 \times 1$ atom pair list and $4 \times 4$ atom pair list without and with (dashed line) cancellation of positive and negative errors. Real energy drift is shown for simulations using double- and mixed-precision settings. Rounding errors in the SETTLE constraint algorithm from the use of single precision causes the drift to become negative at large buffer size. Note that at zero buffer size, the real drift is small because positive (H-H) and negative (O-H) energy errors cancel.

In Fig. 5.5 one can see that for small buffer sizes the drift of the total energy is much smaller than the pair energy error tolerance, due to cancellation of errors. For larger buffer size, the error estimate is a factor of 6 higher than drift of the total energy, or alternatively the buffer estimate is 0.024 nm too large. This is because the protons do not move freely over 18 fs, but rather vibrate.

The only approximation that can lead to an underestimate of the buffer size is that of homogeneous atom

density. This would be particularly problematic for systems with large amount of empty space in the unit cell. This issue is largely mitigated by computing the atom density on a grid with cells of the size of the non-bonded cut-off distance and weighting the density by the atom count in each cell. Thus empty space does not affect the effective atom density. This effective atom density is computed for the starting configuration passed to *mdrun* (page 258). Thus there is only an issue with e.g. phase transitions that start from a gas and end up in a liquid.

### Cut-off artifacts and switched interactions

By default, the pair potentials are shifted to be zero at the cut-off, which makes the potential the integral of the force. However, there can still be energy drift when the forces are non-zero at the cut-off. This effect is extremely small and often not noticeable, as other integration errors (e.g. from constraints) may dominate. To completely avoid cut-off artifacts, the non-bonded forces can be switched exactly to zero at some distance smaller than the neighbor list cut-off (there are several ways to do this in GROMACS, see sec. *Modified non-bonded interactions* (page 484)). One then has a buffer with the size equal to the neighbor list cut-off less the longest interaction cut-off.

### Pressure deviations due to cut-off artifacts

The pressure can be affected more than the energy by missing interactions close to the cut-off, as the force generally has a discontinuity at the cut-off. For Lennard-Jones forces this leads to a consistent increase in pressure as the age of the pair list increases because all missing dispersion interactions have the same sign. The electrostatic forces are much larger at the cut-off, but here the errors tend to cancel out due to (local) electroneutrality. We have not observed errors larger than 0.1 bar due to missing electrostatic interactions in water with PME electrostatics. In practice the Lennard-Jones errors are small when electrostatics interactions are present, as there will be a sufficient buffer to keep the electrostatic energy drift below the tolerance. The only case where there can be significant errors in the pressure is when there are no electrostatic interactions at all or the Ewald relative tolerance parameter is very small, leading to no, or a very small, pair-list buffer. The most common case is coarse-grained systems. In the log file from *mdrun* (page 258) one can find an (over)estimate of the error in the average pressure due to missing Lennard-Jones interactions. The estimate uses (5.18) where we plug in the Lennard-Jones force for $V$. The resulting force error is multiplied by the cut-off distance $r_c$ and divided by the effective box volume to get the (over)estimate for the error in the pressure. The effective box volume ignores empty space to get a better, higher, estimate of the local error in the pressure error in inhomogeneous systems.

When automatically setting the Verlet list life time and buffer, a tolerance on the error in the average pressure due to missing Lennard-Jones interactions can be provided. This uses the estimation formulas described above and puts an upper bound on the error of the pressure averaged over the lifetime of the pair list. The default value for this tolerance is 0.5 bar. For liquid water this corresponds to a maximum relative deviation of the density of $2 \times 10^{-5}$.

### Simple search

Due to (5.10) and (5.15), the vector $\mathbf{r}_{ij}$ connecting images within the cut-off $R_c$ can be found by constructing:

$$\begin{aligned} \mathbf{r}''' &= \mathbf{r}_j - \mathbf{r}_i \\ \mathbf{r}'' &= \mathbf{r}''' - \mathbf{c} * \text{round}(r'''_z/c_z) \\ \mathbf{r}' &= \mathbf{r}'' - \mathbf{b} * \text{round}(r''_y/b_y) \\ \mathbf{r}_{ij} &= \mathbf{r}' - \mathbf{a} * \text{round}(r'_x/a_x) \end{aligned} \tag{5.19}$$

When distances between two particles in a triclinic box are needed that do not obey (5.10), many shifts of combinations of box vectors need to be considered to find the nearest image.

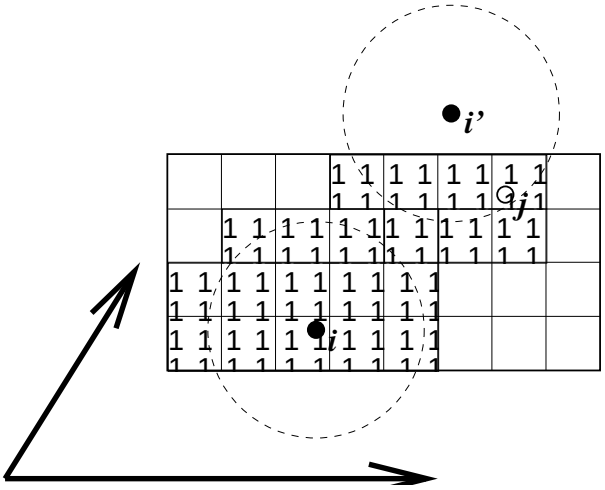



Fig. 5.6: Grid search in two dimensions. The arrows are the box vectors.

### Grid search

The grid search is schematically depicted in Fig. 5.6. All particles are put on the NS grid, with the smallest spacing $\geq R_c/2$ in each of the directions. In the direction of each box vector, a particle $i$ has three images. For each direction the image may be -1,0 or 1, corresponding to a translation over -1, 0 or +1 box vector. We do not search the surrounding NS grid cells for neighbors of $i$ and then calculate the image, but rather construct the images first and then search neighbors corresponding to that image of $i$. As Fig. 5.6 shows, some grid cells may be searched more than once for different images of $i$. This is not a problem, since, due to the minimum image convention, at most one image will "see" the $j$-particle. For every particle, fewer than 125 ($5^3$) neighboring cells are searched. Therefore, the algorithm scales linearly with the number of particles. Although the prefactor is large, the scaling behavior makes the algorithm far superior over the standard $O(N^2)$ algorithm when there are more than a few hundred particles. The grid search is equally fast for rectangular and triclinic boxes. Thus for most protein and peptide simulations the rhombic dodecahedron will be the preferred box shape.

### Charge groups

Charge groups were originally introduced to reduce cut-off artifacts of Coulomb interactions. This concept has been superseded by exact atomistic cut-off treatments. For historical reasons charge groups are still defined in the atoms section for each moleculetype in the topology, but they are no longer used.

### Compute forces

### Potential energy

When forces are computed, the potential energy of each interaction term is computed as well. The total potential energy is summed for various contributions, such as Lennard-Jones, Coulomb, and bonded terms. It is also possible to compute these contributions for *energy-monitor groups* of atoms that are separately defined (see sec. *The group concept* (page 432)).

### Kinetic energy and temperature

The temperature is given by the total kinetic energy of the $N$-particle system:

$$E_{kin} = \frac{1}{2}\sum_{i=1}^{N} m_i v_i^2 \tag{5.20}$$

From this the absolute temperature $T$ can be computed using:

$$\frac{1}{2} N_{\mathrm{df}} kT = E_{\mathrm{kin}} \tag{5.21}$$

where $k$ is Boltzmann's constant and $N_{df}$ is the number of degrees of freedom which can be computed from:

$$N_{\mathrm{df}} \ = \ 3N - N_c - N_{\mathrm{com}} \tag{5.22}$$

Here $N_c$ is the number of *constraints* imposed on the system. When performing molecular dynamics $N_{\mathrm{com}} = 3$ additional degrees of freedom must be removed, because the three center-of-mass velocities are constants of the motion, which are usually set to zero. When simulating in vacuo, the rotation around the center of mass can also be removed, in this case $N_{\mathrm{com}} = 6$. When more than one temperature-coupling group is used, the number of degrees of freedom for group $i$ is:

$$N_{\mathrm{df}}^i \ = \ (3N^i - N_c^i)\frac{3N - N_c - N_{\mathrm{com}}}{3N - N_c} \tag{5.23}$$

The kinetic energy can also be written as a tensor, which is necessary for pressure calculation in a triclinic system, or systems where shear forces are imposed:

$$\mathbf{E}_{\mathrm{kin}} = \frac{1}{2}\sum_{i}^{N} m_i \mathbf{v}_i \otimes \mathbf{v}_i \tag{5.24}$$

### Pressure and virial

The pressure tensor $\mathbf{P}$ is calculated from the difference between kinetic energy $E_{\text{kin}}$ and the virial $\mathbf{\Xi}$:

$$\mathbf{P} = \frac{2}{V}(\mathbf{E}_{\text{kin}} - \mathbf{\Xi}) \tag{5.25}$$

where $V$ is the volume of the computational box. The scalar pressure $P$, which can be used for pressure coupling in the case of isotropic systems, is computed as:

$$P = \text{trace}(\mathbf{P})/3$$

The virial $\mathbf{\Xi}$ tensor is defined as:

$$\mathbf{\Xi} = -\frac{1}{2}\sum_{i<j}\mathbf{r}_{ij} \otimes \mathbf{F}_{ij} \tag{5.26}$$

The GROMACS implementation of the virial computation is described in sec. *Virial and pressure* (page 529)

### The leap-frog integrator

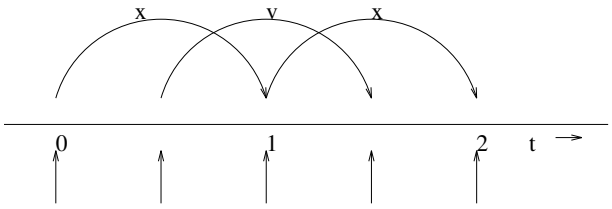



Fig. 5.7: The Leap-Frog integration method. The algorithm is called Leap-Frog because $\mathbf{r}$ and $\mathbf{v}$ are leaping like frogs over each other's backs.

The default MD integrator in GROMACS is the so-called *leap-frog* algorithm *22* (page 684) for the integration of the equations of motion. When extremely accurate integration with temperature and/or pressure coupling is required, the velocity Verlet integrators are also present and may be preferable (see *The velocity Verlet integrator* (page 444)). The leap-frog algorithm uses positions $\mathbf{r}$ at time $t$ and velocities $\mathbf{v}$ at time $t - \frac{1}{2}\Delta t$; it updates positions and velocities using the forces $\mathbf{F}(t)$ determined by the positions at time $t$ using these relations:

$$\begin{aligned} \mathbf{v}(t + \frac{1}{2}\Delta t) &= \mathbf{v}(t - \frac{1}{2}\Delta t) + \frac{\Delta t}{m}\mathbf{F}(t) \\ \mathbf{r}(t + \Delta t) &= \mathbf{r}(t) + \Delta t\mathbf{v}(t + \frac{1}{2}\Delta t) \end{aligned} \tag{5.27}$$

The algorithm is visualized in Fig. 5.7. It produces trajectories that are identical to the Verlet *23* (page 684) algorithm, whose position-update relation is

$$\mathbf{r}(t + \Delta t) = 2\mathbf{r}(t) - \mathbf{r}(t - \Delta t) + \frac{1}{m}\mathbf{F}(t)\Delta t^2 + O(\Delta t^4) \tag{5.28}$$

The algorithm is of third order in $\mathbf{r}$ and is time-reversible. See ref. *24* (page 684) for the merits of this algorithm and comparison with other time integration algorithms.

The equations of motion are modified for temperature coupling and pressure coupling, and extended to include the conservation of constraints, all of which are described below.

### The velocity Verlet integrator

The velocity Verlet algorithm *25* (page 684) is also implemented in GROMACS, though it is not yet fully integrated with all sets of options. In velocity Verlet, positions $\mathbf{r}$ and velocities $\mathbf{v}$ at time $t$ are used to integrate the equations of motion; velocities at the previous half step are not required.

$$\begin{aligned} \mathbf{v}(t+\frac{1}{2}\Delta t) &= \mathbf{v}(t) + \frac{\Delta t}{2m}\mathbf{F}(t) \\ \mathbf{r}(t+\Delta t) &= \mathbf{r}(t) + \Delta t\,\mathbf{v}(t+\frac{1}{2}\Delta t) \\ \mathbf{v}(t+\Delta t) &= \mathbf{v}(t+\frac{1}{2}\Delta t) + \frac{\Delta t}{2m}\mathbf{F}(t+\Delta t) \end{aligned} \tag{5.29}$$

or, equivalently,

$$\begin{aligned} \mathbf{r}(t+\Delta t) &= \mathbf{r}(t) + \Delta t\,\mathbf{v} + \frac{\Delta t^2}{2m}\mathbf{F}(t) \\ \mathbf{v}(t+\Delta t) &= \mathbf{v}(t) + \frac{\Delta t}{2m}\left[\mathbf{F}(t) + \mathbf{F}(t+\Delta t)\right] \end{aligned} \tag{5.30}$$

With no temperature or pressure coupling, and with *corresponding* starting points, leap-frog and velocity Verlet will generate identical trajectories, as can easily be verified by hand from the equations above. Given a single starting file with the *same* starting point $\mathbf{x}(0)$ and $\mathbf{v}(0)$, leap-frog and velocity Verlet will *not* give identical trajectories, as leap-frog will interpret the velocities as corresponding to $t = -\frac{1}{2}\Delta t$, while velocity Verlet will interpret them as corresponding to the timepoint $t = 0$.

### Understanding reversible integrators: The Trotter decomposition

To further understand the relationship between velocity Verlet and leap-frog integration, we introduce the reversible Trotter formulation of dynamics, which is also useful to understanding implementations of thermostats and barostats in GROMACS.

A system of coupled, first-order differential equations can be evolved from time $t = 0$ to time $t$ by applying the evolution operator

$$\begin{aligned} \Gamma(t) &= \exp(iLt)\Gamma(0) \\ iL &= \dot{\Gamma}\cdot\nabla_{\Gamma}, \end{aligned}$$

where $L$ is the Liouville operator, and $\Gamma$ is the multidimensional vector of independent variables (positions and velocities). A short-time approximation to the true operator, accurate at time $\Delta t = t/P$, is applied $P$ times in succession to evolve the system as

$$\Gamma(t) = \prod_{i=1}^{P} \exp(iL\Delta t)\Gamma(0) \tag{5.31}$$

For NVE dynamics, the Liouville operator is

$$iL = \sum_{i=1}^{N} \mathbf{v}_i \cdot \nabla_{\mathbf{r}_i} + \sum_{i=1}^{N} \frac{1}{m_i}\mathbf{F}(r_i)\cdot\nabla_{\mathbf{v}_i}. \tag{5.32}$$

This can be split into two additive operators

$$\begin{aligned} iL_1 &= \sum_{i=1}^{N} \frac{1}{m_i}\mathbf{F}(r_i)\cdot\nabla_{\mathbf{v}_i} \\ iL_2 &= \sum_{i=1}^{N} \mathbf{v}_i\cdot\nabla_{\mathbf{r}_i} \end{aligned}$$

Then a short-time, symmetric, and thus reversible approximation of the true dynamics will be

$$\exp(iL\Delta t) = \exp(iL_2\frac{1}{2}\Delta t)\exp(iL_1\Delta t)\exp(iL_2\frac{1}{2}\Delta t) + \mathcal{O}(\Delta t^3). \tag{5.33}$$

This corresponds to velocity Verlet integration. The first exponential term over $\frac{1}{2}\Delta t$ corresponds to a velocity half-step, the second exponential term over $\Delta t$ corresponds to a full velocity step, and the last exponential term over $\frac{1}{2}\Delta t$ is the final velocity half step. For future times $t = n\Delta t$, this becomes

$$\begin{aligned}\exp(iLn\Delta t) \approx & \left(\exp(iL_2\frac{1}{2}\Delta t)\exp(iL_1\Delta t)\exp(iL_2\frac{1}{2}\Delta t)\right)^n \\ \approx & \exp(iL_2\frac{1}{2}\Delta t)\left(\exp(iL_1\Delta t)\exp(iL_2\Delta t)\right)^{n-1} \\ & \exp(iL_1\Delta t)\exp(iL_2\frac{1}{2}\Delta t)\end{aligned}$$

This formalism allows us to easily see the difference between the different flavors of Verlet integrators. The leap-frog integrator can be seen as starting with (5.33) with the $\exp(iL_1\Delta t)$ term, instead of the half-step velocity term, yielding

$$\exp(iLn\Delta t) = \exp(iL_1\Delta t)\exp(iL_2\Delta t) + \mathcal{O}(\Delta t^3). \tag{5.34}$$

Here, the full step in velocity is between $t - \frac{1}{2}\Delta t$ and $t + \frac{1}{2}\Delta t$, since it is a combination of the velocity half steps in velocity Verlet. For future times $t = n\Delta t$, this becomes

$$\exp(iLn\Delta t) \approx \left(\exp(iL_1\Delta t)\exp(iL_2\Delta t)\right)^n. \tag{5.35}$$

Although at first this does not appear symmetric, as long as the full velocity step is between $t - \frac{1}{2}\Delta t$ and $t + \frac{1}{2}\Delta t$, then this is simply a way of starting velocity Verlet at a different place in the cycle.

Even though the trajectory and thus potential energies are identical between leap-frog and velocity Verlet, the kinetic energy and temperature will not necessarily be the same. Standard velocity Verlet uses the velocities at the $t$ to calculate the kinetic energy and thus the temperature only at time $t$; the kinetic energy is then a sum over all particles

$$\begin{aligned}KE_{\text{full}}(t) = & \sum_i \left(\frac{1}{2m_i}\mathbf{v}_i(t)\right)^2 \\ = & \sum_i \frac{1}{2m_i}\left(\frac{1}{2}\mathbf{v}_i(t-\frac{1}{2}\Delta t) + \frac{1}{2}\mathbf{v}_i(t+\frac{1}{2}\Delta t)\right)^2,\end{aligned}$$

with the square on the *outside* of the average. Standard leap-frog calculates the kinetic energy at time $t$ based on the average kinetic energies at the timesteps $t + \frac{1}{2}\Delta t$ and $t - \frac{1}{2}\Delta t$, or the sum over all particles

$$KE_{\text{average}}(t) = \sum_i \frac{1}{2m_i}\left(\frac{1}{2}\mathbf{v}_i(t-\frac{1}{2}\Delta t)^2 + \frac{1}{2}\mathbf{v}_i(t+\frac{1}{2}\Delta t)^2\right), \tag{5.36}$$

where the square is *inside* the average.

A non-standard variant of velocity Verlet which averages the kinetic energies $KE(t+\frac{1}{2}\Delta t)$ and $KE(t-\frac{1}{2}\Delta t)$, exactly like leap-frog, is also now implemented in GROMACS (as *mdp* (page 579) file option `integrator=md-vv-avek` (page 52)). Without temperature and pressure coupling, velocity Verlet with half-step-averaged kinetic energies and leap-frog will be identical up to numerical precision. For temperature- and pressure-control schemes, however, velocity Verlet with half-step-averaged kinetic energies and leap-frog will be different, as will be discussed in the section in thermostats and barostats.

The half-step-averaged kinetic energy and temperature are slightly more accurate for a given step size; the difference in average kinetic energies using the half-step-averaged kinetic energies ( *integrator=md* (page 51) and *integrator=md-vv-avek* (page 52) ) will be closer to the kinetic energy obtained in the limit of small step size than will the full-step kinetic energy (using *integrator=md-vv* (page 51)). For NVE simulations, this difference is usually not significant, since the positions and velocities of the particles are still identical; it makes a difference in the way the temperature of the simulations are **interpreted**, but **not** in the trajectories that are produced. Although the kinetic energy is more accurate with the half-step-averaged method, meaning that it changes less as the timestep gets large, it is also more noisy. The RMS deviation of the total energy of the system (sum of kinetic plus potential) in the half-step-averaged kinetic energy case will be higher (about twice as high in most cases) than the full-step kinetic energy. The drift will still be the same, however, as again, the trajectories are identical.

For NVT simulations, however, there **will** be a difference, as discussed in the section on temperature control, since the velocities of the particles are adjusted such that kinetic energies of the simulations, which can be calculated either way, reach the distribution corresponding to the set temperature. In this case, the three methods will not give identical results.

Because the velocity and position are both defined at the same time $t$ the velocity Verlet integrator can be used for some methods, especially rigorously correct pressure control methods, that are not actually possible with leap-frog. The integration itself takes negligibly more time than leap-frog, but twice as many communication calls are currently required. In most cases, and especially for large systems where communication speed is important for parallelization and differences between thermodynamic ensembles vanish in the $1/N$ limit, and when only NVT ensembles are required, leap-frog will likely be the preferred integrator. For pressure control simulations where the fine details of the thermodynamics are important, only velocity Verlet allows the true ensemble to be calculated. In either case, simulation with double precision may be required to get fine details of thermodynamics correct.

### Multiple time-stepping

The leap-frog integrator in GROMACS supports a configurable multiple time-stepping scheme. This can be used to improve performance by computing slowly varying forces less frequently. The RESPA scheme *191* (page 692) is used, which is based on a TROTTER decomposition and is therefore reversible and symplectic.

In order to allow tuning this for each system, the integrator makes it possible to specify different types of bonded and non-bonded interactions for multiple-time step integration. To avoid integration errors, it is still imperative that the integration interval used for each force component is short enough, and there is no universal formula that allows the algorithm to detect this. Since the slowly-varying forces are often of smaller magnitude, using time steps that are too large might not result in simulations crashing, so it is recommended to be conservative and only gradually increase intervals while ensuring you get proper sampling and avoid energy drifts. As an initial guidance, many of the most common biomolecular force fields appear to run into stability problems when the period of integrating Lennard-Jones forces is 4 fs or longer, so for now we only recommend computing long-range electrostatics (PME mesh contribution) less frequently than every step when using a base time step of 2 fs. Another, rather different, scenario is to use a base time step of 0.5 fs with non-constrained harmonic bonds, and compute other interactions every second or fourth step. Despite these caveats, we encourage users to test the functionality, assess stability and energy drifts, and either discuss your experience in the GROMACS forums or suggest improvements to the documentation so we can improve this guidance in the future.

For using larger time steps for all interactions, and integration, angle vibrations involving hydrogen atoms can be removed using virtual interaction sites (see sec. *Removing fastest degrees of freedom*

(page 624)), which brings the shortest time step up to PME mesh update frequency of a multiple time stepping scheme. This results in a near doubling of the simulation performance.

### Temperature coupling

While direct use of molecular dynamics gives rise to the NVE (constant number, constant volume, constant energy ensemble), most quantities that we wish to calculate are actually from a constant temperature (NVT) ensemble, also called the canonical ensemble. GROMACS can use the *weak-coupling* scheme of Berendsen *26* (page 684), stochastic randomization through the Andersen thermostat *27* (page 684), the extended ensemble Nosé-Hoover scheme *28* (page 684), *29* (page 684), or a velocity-rescaling scheme *30* (page 684) to simulate constant temperature, with advantages of each of the schemes laid out below.

There are several other reasons why it might be necessary to control the temperature of the system (drift during equilibration, drift as a result of force truncation and integration errors, heating due to external or frictional forces), but this is not entirely correct to do from a thermodynamic standpoint, and in some cases only masks the symptoms (increase in temperature of the system) rather than the underlying problem (deviations from correct physics in the dynamics). For larger systems, errors in ensemble averages and structural properties incurred by using temperature control to remove slow drifts in temperature appear to be negligible, but no completely comprehensive comparisons have been carried out, and some caution must be taking in interpreting the results.

When using temperature and/or pressure coupling the total energy is no longer conserved. Instead there is a conserved energy quantity the formula of which will depend on the combination or temperature and pressure coupling algorithm used. For all coupling algorithms, except for Andersen temperature coupling and Parrinello-Rahman pressure coupling combined with shear stress, the conserved energy quantity is computed and stored in the energy and log file. Note that this quantity will not be conserved when external forces are applied to the system, such as pulling on group with a changing distance or an electric field. Furthermore, how well the energy is conserved depends on the accuracy of all algorithms involved in the simulation. Usually the algorithms that cause most drift are constraints and the pair-list buffer, depending on the parameters used.

### Berendsen temperature coupling

The Berendsen algorithm mimics weak coupling with first-order kinetics to an external heat bath with given temperature $T_0$. See ref. *31* (page 684) for a comparison with the Nosé-Hoover scheme. The effect of this algorithm is that a deviation of the system temperature from $T_0$ is slowly corrected according to:

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{T_0 - T}{\tau} \tag{5.37}$$

which means that a temperature deviation decays exponentially with a time constant $\tau$. This method of coupling has the advantage that the strength of the coupling can be varied and adapted to the user requirement: for equilibration purposes the coupling time can be taken quite short (*e.g.* 0.01 ps), but for reliable equilibrium runs it can be taken much longer (*e.g.* 0.5 ps) in which case it hardly influences the conservative dynamics.

The Berendsen thermostat suppresses the fluctuations of the kinetic energy. This means that one does not generate a proper canonical ensemble, so rigorously, the sampling will be incorrect. This error scales with $1/N$, so for very large systems most ensemble averages will not be affected significantly, except for the distribution of the kinetic energy itself. However, fluctuation properties, such as the heat capacity, will be affected. A similar thermostat which does produce a correct ensemble is the velocity rescaling

thermostat *30* (page 684) described below, so while the Berendsen thermostat is supported for historical reasons, including the ability to reproduce old simulations, we strongly recommend against using it for new simulations.

The heat flow into or out of the system is affected by scaling the velocities of each particle every step, or every $n_{\mathrm{TC}}$ steps, with a time-dependent factor $\lambda$, given by:

$$\lambda = \left[1 + \frac{n_{\mathrm{TC}}\Delta t}{\tau_T}\left\{\frac{T_0}{T(t-\frac{1}{2}\Delta t)} - 1\right\}\right]^{1/2} \tag{5.38}$$

The parameter $\tau_T$ is close, but not exactly equal, to the time constant $\tau$ of the temperature coupling ((5.37)):

$$\tau = 2C_V\tau_T/N_{df}k \tag{5.39}$$

where $C_V$ is the total heat capacity of the system, $k$ is Boltzmann's constant, and $N_{df}$ is the total number of degrees of freedom. The reason that $\tau \neq \tau_T$ is that the kinetic energy change caused by scaling the velocities is partly redistributed between kinetic and potential energy and hence the change in temperature is less than the scaling energy. In practice, the ratio $\tau/\tau_T$ ranges from 1 (gas) to 2 (harmonic solid) to 3 (water). When we use the term *temperature coupling time constant*, we mean the parameter $\tau_T$. **Note** that in practice the scaling factor $\lambda$ is limited to the range of $0.8 <= \lambda <= 1.25$, to avoid scaling by very large numbers which may crash the simulation. In normal use, $\lambda$ will always be much closer to 1.0.

The thermostat modifies the kinetic energy at each scaling step by:

$$\Delta E_k = (\lambda - 1)^2 E_k \tag{5.40}$$

The sum of these changes over the run needs to subtracted from the total energy to obtain the conserved energy quantity.

### Velocity-rescaling temperature coupling

The velocity-rescaling thermostat *30* (page 684) is essentially a Berendsen thermostat (see above) with an additional stochastic term that ensures a correct kinetic energy distribution by modifying it according to

$$\mathrm{d}K = (K_0 - K)\frac{\mathrm{d}t}{\tau_T} + 2\sqrt{\frac{KK_0}{N_f}}\frac{\mathrm{d}W}{\sqrt{\tau_T}}, \tag{5.41}$$

where $K$ is the kinetic energy, $N_f$ the number of degrees of freedom and $\mathrm{d}W$ a Wiener process. There are no additional parameters, except for a random seed. This thermostat produces a correct canonical ensemble and still has the advantage of the Berendsen thermostat: first order decay of temperature deviations and no oscillations.

### Andersen thermostat

One simple way to maintain a thermostatted ensemble is to take an $NVE$ integrator and periodically re-select the velocities of the particles from a Maxwell-Boltzmann distribution *27* (page 684). This can either be done by randomizing all the velocities simultaneously (massive collision) every $\tau_T/\Delta t$ steps (`andersen-massive`), or by randomizing every particle with some small probability every

timestep (`andersen`), equal to $\Delta t/\tau$, where in both cases $\Delta t$ is the timestep and $\tau_T$ is a characteristic coupling time scale. Because of the way constraints operate, all particles in the same constraint group must be randomized simultaneously. Because of parallelization issues, the `andersen` version cannot currently (5.0) be used in systems with constraints. `andersen-massive` can be used regardless of constraints. This thermostat is also currently only possible with velocity Verlet algorithms, because it operates directly on the velocities at each timestep.

This algorithm completely avoids some of the ergodicity issues of other thermostatting algorithms, as energy cannot flow back and forth between energetically decoupled components of the system as in velocity scaling motions. However, it can slow down the kinetics of system by randomizing correlated motions of the system, including slowing sampling when $\tau_T$ is at moderate levels (less than 10 ps). This algorithm should therefore generally not be used when examining kinetics or transport properties of the system *32* (page 684).

### Nosé-Hoover temperature coupling

The Berendsen weak-coupling algorithm is extremely efficient for relaxing a system to the target temperature, but once the system has reached equilibrium it might be more important to probe a correct canonical ensemble. This is unfortunately not the case for the weak-coupling scheme.

To enable canonical ensemble simulations, GROMACS also supports the extended-ensemble approach first proposed by Nosé *28* (page 684) and later modified by Hoover *29* (page 684). The system Hamiltonian is extended by introducing a thermal reservoir and a friction term in the equations of motion. The friction force is proportional to the product of each particle's velocity and a friction parameter, $\xi$. This friction parameter (or *heat bath* variable) is a fully dynamic quantity with its own momentum ($p_\xi$) and equation of motion; the time derivative is calculated from the difference between the current kinetic energy and the reference temperature.

In this formulation, the particles' equations of motion in the global *MD scheme* (page 433) are replaced by:

$$\frac{\mathrm{d}^2\mathbf{r}_i}{\mathrm{d}t^2} = \frac{\mathbf{F}_i}{m_i} - \frac{p_\xi}{Q}\frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t}, \tag{5.42}$$

where the equation of motion for the heat bath parameter $\xi$ is:

$$\frac{\mathrm{d}p_\xi}{\mathrm{d}t} = (T - T_0)\, N_f k. \tag{5.43}$$

The reference temperature is denoted $T_0$, while $T$ is the current instantaneous temperature of the system, $N_f$ is the total number of degrees of freedom and $k$ is Boltzmann's constant (see chapter *Definitions and Units* (page 425)). The strength of the coupling is determined by the constant $Q$ (usually called the *mass parameter* of the reservoir) in combination with the reference temperature.[1]

The conserved quantity for the Nosé-Hoover equations of motion is not the total energy, but rather

$$H = \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m_i} + U\left(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N\right) + \frac{p_\xi^2}{2Q} + N_f k T \xi, \tag{5.44}$$

In our opinion, the mass parameter is a somewhat awkward way of describing coupling strength, especially due to its dependence on reference temperature (and some implementations even include the number of degrees of freedom in your system when defining $Q$). To maintain the coupling strength, one would have to change $Q$ in proportion to the change in reference temperature. For this reason, we prefer

[1] Note that some derivations, an alternative notation $\xi_{\mathrm{alt}} = v_\xi = p_\xi/Q$ is used.

to let the GROMACS user work with the period $\tau_T$ of the oscillations of kinetic energy between the system and the reservoir instead. It is directly related to $Q$ and $T_0$ via:

$$Q = \frac{\tau_T^2 N_f k T_0}{4\pi^2}. \tag{5.45}$$

This provides a much more intuitive way of selecting the Nosé-Hoover coupling strength (similar to the weak-coupling relaxation), and in addition $\tau_T$ is independent of system size and reference temperature.

It is, however, important to keep the difference between the weak-coupling scheme and the Nosé-Hoover algorithm in mind: Using weak coupling you get a strongly damped *exponential relaxation*, while the Nosé-Hoover approach produces an *oscillatory relaxation*. The actual time it takes to relax with Nosé-Hoover coupling is several times larger than the period of the oscillations that you select. These oscillations (in contrast to exponential relaxation) also means that the time constant normally should be 4–5 times larger than the relaxation time used with weak coupling, but your mileage may vary.

Nosé-Hoover dynamics in simple systems such as collections of harmonic oscillators, can be *nonergodic*, meaning that only a subsection of phase space is ever sampled, even if the simulations were to run for infinitely long. For this reason, the Nosé-Hoover chain approach was developed, where each of the Nosé-Hoover thermostats has its own Nosé-Hoover thermostat controlling its temperature. In the limit of an infinite chain of thermostats, the dynamics are guaranteed to be ergodic. Using just a few chains can greatly improve the ergodicity, but recent research has shown that the system will still be nonergodic, and it is still not entirely clear what the practical effect of this is *33* (page 684). Currently, the default number of chains is 10, but this can be controlled by the user. In the case of chains, the equations are modified in the following way to include a chain of thermostatting particles *34* (page 684):

$$\begin{aligned}
\frac{\mathrm{d}^2\mathbf{r}_i}{\mathrm{d}t^2} &= \frac{\mathbf{F}_i}{m_i} - \frac{p_{\xi_1}}{Q_1}\frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} \\
\frac{\mathrm{d}p_{\xi_1}}{\mathrm{d}t} &= N_f k\left(T - T_0\right) - p_{\xi_1}\frac{p_{\xi_2}}{Q_2} \\
\frac{\mathrm{d}p_{\xi_{i=2\ldots M-1}}}{\mathrm{d}t} &= \left(\frac{p_{\xi_{i-1}}^2}{Q_{i-1}} - kT_0\right) - p_{\xi_i}\frac{p_{\xi_{i+1}}}{Q_{i+1}} \\
\frac{\mathrm{d}p_{\xi_M}}{\mathrm{d}t} &= \left(\frac{p_{\xi_{N-1}}^2}{Q_{N-1}} - kT_0\right)
\end{aligned}$$

The conserved quantity for Nosé-Hoover chains is

$$H = \sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{2m_i} + U\left(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N\right) + \sum_{k=1}^{M}\frac{p_{\xi_k}^2}{2Q_k} + N_f k T \xi_1 + kT\sum_{k=2}^{M}\xi_k \tag{5.46}$$

The values and velocities of the Nosé-Hoover thermostat variables are generally not included in the output, as they take up a fair amount of space and are generally not important for analysis of simulations, but by setting an *mdp* (page 579) option the values of all the positions and velocities of all Nosé-Hoover particles in the chain are written to the *edr* (page 576) file. Leap-frog simulations currently can only have Nosé-Hoover chain lengths of 1, but this will likely be updated in a later version.

As described in the integrator section, for temperature coupling, the temperature that the algorithm attempts to match to the reference temperature is calculated differently in velocity Verlet and leap-frog dynamics. Velocity Verlet (*md-vv*) uses the full-step kinetic energy, while leap-frog and *md-vv-avek* use the half-step-averaged kinetic energy.

We can examine the Trotter decomposition again to better understand the differences between these constant-temperature integrators. In the case of Nosé-Hoover dynamics (for simplicity, using a chain

with $N = 1$, with more details in Ref. *35* (page 684)), we split the Liouville operator as

$$iL = iL_1 + iL_2 + iL_{\mathrm{NHC}}, \tag{5.47}$$

where

$$\begin{aligned} iL_1 &= \sum_{i=1}^{N} \left[\frac{\mathbf{p}_i}{m_i}\right] \cdot \frac{\partial}{\partial \mathbf{r}_i} \\ iL_2 &= \sum_{i=1}^{N} \mathbf{F}_i \cdot \frac{\partial}{\partial \mathbf{p}_i} \\ iL_{\mathrm{NHC}} &= \sum_{i=1}^{N} -\frac{p_\xi}{Q}\mathbf{v}_i \cdot \nabla_{\mathbf{v}_i} + \frac{p_\xi}{Q}\frac{\partial}{\partial \xi} + N_f k \left(T - T_0\right) \frac{\partial}{\partial p_\xi} \end{aligned}$$

For standard velocity Verlet with Nosé-Hoover temperature control, this becomes

$$\begin{aligned} \exp(iL\Delta t) = &\exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) \exp\left(iL_2\Delta t/2\right) \\ &\exp\left(iL_1\Delta t\right) \exp\left(iL_2\Delta t/2\right) \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) + \mathcal{O}(\Delta t^3). \end{aligned}$$

For half-step-averaged temperature control using *md-vv-avek*, this decomposition will not work, since we do not have the full step temperature until after the second velocity step. However, we can construct an alternate decomposition that is still reversible, by switching the place of the NHC and velocity portions of the decomposition:

$$\begin{aligned} \exp(iL\Delta t) = &\exp\left(iL_2\Delta t/2\right) \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) \exp\left(iL_1\Delta t\right) \\ &\exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) \exp\left(iL_2\Delta t/2\right) + \mathcal{O}(\Delta t^3) \end{aligned}$$

This formalism allows us to easily see the difference between the different flavors of velocity Verlet integrator. The leap-frog integrator can be seen as starting with (5.48) just before the $\exp\left(iL_1\Delta t\right)$ term, yielding:

$$\begin{aligned} \exp(iL\Delta t) = &\exp\left(iL_1\Delta t\right) \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) \\ &\exp\left(iL_2\Delta t\right) \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) + \mathcal{O}(\Delta t^3) \end{aligned}$$

and then using some algebra tricks to solve for some quantities are required before they are actually calculated *36* (page 684).

### Group temperature coupling

In GROMACS temperature coupling can be performed on groups of atoms, typically a protein and solvent. The reason such algorithms were introduced is that energy exchange between different components is not perfect, due to different effects including cut-offs etc. If now the whole system is coupled to one heat bath, water (which experiences the largest cut-off noise) will tend to heat up and the protein will cool down. Typically 100 K differences can be obtained. With the use of proper electrostatic methods (PME) these difference are much smaller but still not negligible. The parameters for temperature coupling in groups are given in the *mdp* (page 579) file. Recent investigation has shown that small temperature differences between protein and water may actually be an artifact of the way temperature is calculated when there are finite timesteps, and very large differences in temperature are likely a sign of something else seriously going wrong with the system, and should be investigated carefully *37* (page 684).

One special case should be mentioned: it is possible to temperature-couple only part of the system, leaving other parts without temperature coupling. This is done by specifying $-1$ for the time constant

$\tau_T$ for the group that should not be thermostatted. If only part of the system is thermostatted, the system will still eventually converge to an NVT system. In fact, one suggestion for minimizing errors in the temperature caused by discretized timesteps is that if constraints on the water are used, then only the water degrees of freedom should be thermostatted, not protein degrees of freedom, as the higher frequency modes in the protein can cause larger deviations from the *true* temperature, the temperature obtained with small timesteps *37* (page 684).

### Pressure coupling

In the same spirit as the temperature coupling, the system can also be coupled to a *pressure bath.* GROMACS supports both the Berendsen algorithm *26* (page 684) that scales coordinates and box vectors every step (we strongly recommend not to use it), a new stochastic cell rescaling algorithm, the extended-ensemble Parrinello-Rahman approach *38* (page 684), *39* (page 685), and for the velocity Verlet variants, the Martyna-Tuckerman-Tobias-Klein (MTTK) implementation of pressure control *35* (page 684). Parrinello-Rahman and Berendsen can be combined with any of the temperature coupling methods above. MTTK can only be used with Nosé-Hoover temperature control. From version 5.1 onwards, it can only used when the system does not have constraints.

### Berendsen pressure coupling

The Berendsen algorithm rescales the coordinates and box vectors every step, or every $n_{\mathrm{PC}}$ steps, with a matrix $\mu$, which has the effect of a first-order kinetic relaxation of the pressure towards a given reference pressure $\mathbf{P}_0$ according to

$$\frac{\mathrm{d}\mathbf{P}}{\mathrm{d}t} = \frac{\mathbf{P}_0 - \mathbf{P}}{\tau_p}. \tag{5.48}$$

The scaling matrix $\mu$ is given by

$$\mu_{ij} = \delta_{ij} - \frac{n_{\mathrm{PC}}\Delta t}{3\,\tau_p}\beta_{ij}\{P_{0ij} - P_{ij}(t)\}. \tag{5.49}$$

Here, $\beta$ is the isothermal compressibility of the system. In most cases this will be a diagonal matrix, with equal elements on the diagonal, the value of which is generally not known. It suffices to take a rough estimate because the value of $\beta$ only influences the non-critical time constant of the pressure relaxation without affecting the average pressure itself. For water at 1 atm and 300 K $\beta = 4.6 \times 10^{-10}$ Pa$^{-1}$ $= 4.6 \times 10^{-5}$ bar$^{-1}$, which is $7.6 \times 10^{-4}$ MD units (see chapter *Definitions and Units* (page 425)). Most other liquids have similar values. When scaling completely anisotropically, the system has to be rotated in order to obey (5.10). This rotation is approximated in first order in the scaling, which is usually less than $10^{-4}$. The actual scaling matrix $\mu'$ is

$$\mu' = \begin{pmatrix} \mu_{xx} & \mu_{xy} + \mu_{yx} & \mu_{xz} + \mu_{zx} \\ 0 & \mu_{yy} & \mu_{yz} + \mu_{zy} \\ 0 & 0 & \mu_{zz} \end{pmatrix}. \tag{5.50}$$

The velocities are neither scaled nor rotated. Since the equations of motion are modified by pressure coupling, the conserved energy quantity also needs to be modified. For first order pressure coupling, the work the barostat applies to the system every step needs to be subtracted from the total energy to obtain the conserved energy quantity:

$$-\sum_{i,j}(\mu_{ij} - \delta_{ij})P_{ij}V = \sum_{i,j} 2(\mu_{ij} - \delta_{ij})\Xi_{ij} \tag{5.51}$$

where $\delta_{ij}$ is the Kronecker delta and $\boldsymbol{\Xi}$ is the virial. Note that the factor 2 originates from the factor $\frac{1}{2}$ in the virial definition (5.26).

In GROMACS, the Berendsen scaling can also be done isotropically, which means that instead of $\mathbf{P}$ a diagonal matrix with elements of size trace$(\mathbf{P})/3$ is used. For systems with interfaces, semi-isotropic scaling can be useful. In this case, the $x/y$-directions are scaled isotropically and the $z$ direction is scaled independently. The compressibility in the $x/y$ or $z$-direction can be set to zero, to scale only in the other direction(s).

If you allow full anisotropic deformations and use constraints you might have to scale more slowly or decrease your timestep to avoid errors from the constraint algorithms.

It is important to note that although the Berendsen pressure control algorithm yields a simulation with the correct average pressure, it does not yield the exact NPT ensemble, and does not compute the correct fluctuations in pressure or volume. We strongly advise against using it for new simulations. The only useful role it has had recently is to ensure fast relaxation without oscillations, e.g. at the start of a simulation for from equilibrium, but this is now provided by the stochastic cell rescaling, which should be used instead. For full anisotropic simulations you need to use the Parrinello-Rahman barostat (for now). This does have the same oscillation problems as many other correct-ensemble barostats, so if you cannot get your initial system stable you might need to use Berendsen briefly - but the warnings/errors you get are a reminder it should not be used for production runs.

### Stochastic cell rescaling

The stochastic cell rescaling algorithm is a variant of the Berendsen algorithm that allows correct fluctuations to be sampled. Similarly to the Berendsen algorithm, it rescales the coordinates and box vectors every step, or every $n_{\mathrm{PC}}$ steps with the effect of a first-order kinetic relaxation of the pressure towards a given reference pressure $P_0$. At variance with the Berendsen algorithm, the rescaling matrix is calculated including a stochastic term that makes volume fluctuations correct.

The isotropic version can be easily written in term of the strain $\epsilon = \log(V/V_0)$ that is evolved according to the following equation of motion

$$\mathrm{d}\epsilon = -\frac{\beta}{\tau_p}(P_0 - P)\mathrm{d}t + \sqrt{\frac{2k_BT\beta}{V\tau_p}}\mathrm{d}W \tag{5.52}$$

Here, $\beta$ is the isothermal compressibility of the system. It suffices to take a rough estimate because the value of $\beta$ only influences the non-critical time constant of the pressure relaxation without affecting the volume distribution itself. For water at 1 atm and 300 K $\beta = 4.6 \times 10^{-10}$ Pa$^{-1}$ $= 4.6 \times 10^{-5}$ bar$^{-1}$, which is $7.6 \times 10^{-4}$ MD units (see chapter *Definitions and Units* (page 425)). Most other liquids have similar values.

Another difference with respect to the Berendsen algorithm is that velocities are scaled with a factor that is the reciprocal of the scaling factor for positions.

A semi-isotropic implementation is also provided. By defining the variables $\epsilon_{xy} = \log(A/A_0)$ and $\epsilon_z = \log(L/L_0)$, where $A$ and $L$ are the area of the simulation box in the $xy$ plane and its height, respectively, the following equations can be obtained:

$$\mathrm{d}\epsilon_{xy} = -\frac{2\beta}{3\tau_p}(P_0 - \frac{\gamma}{L} - \frac{P_{xx} + P_{yy}}{2})\mathrm{d}t + \sqrt{\frac{4k_BT\beta}{3V\tau_p}}\mathrm{d}W_{xy} \tag{5.53}$$

$$\mathrm{d}\epsilon_z = -\frac{\beta}{3\tau_p}(P_0 - P_{zz})\mathrm{d}t + \sqrt{\frac{2k_BT\beta}{3V\tau_p}}\mathrm{d}W_z \tag{5.54}$$

Here $\gamma$ is the external surface tension and $P_{xx}$, $P_{yy}$, and $P_{zz}$ the components of the internal pressure.

More detailed explanations can be found in the original reference *184* (page 692).

### Parrinello-Rahman pressure coupling

GROMACS also supports constant-pressure simulations using the Parrinello-Rahman approach *38* (page 684), *39* (page 685), which is similar to the Nosé-Hoover temperature coupling, and in theory gives the true NPT ensemble. With the Parrinello-Rahman barostat, the box vectors as represented by the matrix $\mathbf{b}$ obey the matrix equation of motion[2]

$$\frac{\mathrm{d}\mathbf{b}^2}{\mathrm{d}t^2} = V\mathbf{W}^{-1}\mathbf{b}'^{-1}\left(\mathbf{P}-\mathbf{P}_{ref}\right). \tag{5.55}$$

The volume of the box is denoted $V$, and $\mathbf{W}$ is a matrix parameter that determines the strength of the coupling (see below). The matrices $\mathbf{P}$ and $\mathbf{P}_{ref}$ are the current and reference pressures, respectively. The prime notation denotes transposition of the matrix.

The equations of motion for the particles are also changed, just as for the Nosé-Hoover coupling. In most cases you would combine the Parrinello-Rahman barostat with the Nosé-Hoover thermostat, but to keep it simple we only show the Parrinello-Rahman modification here. The modified Hamiltonian, which will be conserved, is:

$$E_{\mathrm{pot}} + E_{\mathrm{kin}} + \sum_i P_{ii}V + \sum_{i,j}\frac{1}{2}W_{ij}\left(\frac{\mathrm{d}b_{ij}}{\mathrm{d}t}\right)^2 \tag{5.56}$$

The equations of motion for the atoms obtained from the Hamiltonian are:

$$\begin{aligned}\frac{\mathrm{d}^2\mathbf{r}_i}{\mathrm{d}t^2} &= \frac{\mathbf{F}_i}{m_i} - \mathbf{M}\frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t},\\ \mathbf{M} &= \mathbf{b}^{-1}\left[\mathbf{b}\frac{\mathrm{d}\mathbf{b}'}{\mathrm{d}t} + \frac{\mathrm{d}\mathbf{b}}{\mathrm{d}t}\mathbf{b}'\right]\mathbf{b}'^{-1}.\end{aligned} \tag{5.57}$$

This extra term has the appearance of a friction, but it should be noted that it is fictitious, and rather an effect of the Parrinello-Rahman equations of motion being defined with all particle coordinates represented relative to the box vectors, while GROMACS uses normal Cartesian coordinates for positions, velocities and forces. It is worth noting that the kinetic energy too should formally be calculated based on velocities relative to the box vectors. This can have an effect e.g. for external constant stress, but for now we only support coupling to constant external pressures, and for any normal simulation the velocities of box vectors should be extremely small compared to particle velocities. Gang Liu has done some work on deriving this for Cartesian coordinates *40* (page 685) but it is not implemented in GROMACS.

The (inverse) mass parameter matrix $\mathbf{W}^{-1}$ determines the strength of the coupling, and how the box can be deformed. The box restriction (5.10) will be fulfilled automatically if the corresponding elements of $\mathbf{W}^{-1}$ are zero. Since the coupling strength also depends on the size of your box, we prefer to calculate it automatically in GROMACS. You only have to provide the approximate isothermal compressibilities $\beta$ and the pressure time constant $\tau_p$ in the input file ($L$ is the largest box matrix element):

$$\left(\mathbf{W}^{-1}\right)_{ij} = \frac{4\pi^2\beta_{ij}}{3\tau_p^2 L}. \tag{5.58}$$

Just as for the Nosé-Hoover thermostat, you should realize that the Parrinello-Rahman time constant is *not* equivalent to the relaxation time used in the Berendsen pressure coupling algorithm. In most

[2] The box matrix representation in corresponds to the transpose of the box matrix representation in the paper by Nosé and Klein. Because of this, some of our equations will look slightly different.

cases you will need to use a 4–5 times larger time constant with Parrinello-Rahman coupling. If your pressure is very far from equilibrium, the Parrinello-Rahman coupling may result in very large box oscillations that could even crash your run. In that case you would have to increase the time constant, or (better) use the weak-coupling or stochastic cell rescaling schemes to reach the target pressure, and then switch to Parrinello-Rahman coupling once the system is in equilibrium. Additionally, using the leap-frog algorithm, the pressure at time $t$ is not available until after the time step has completed, and so the pressure from the previous step must be used, which makes the algorithm not directly reversible, and may not be appropriate for high-precision thermodynamic calculations.

### Surface-tension coupling

When a periodic system consists of more than one phase, separated by surfaces which are parallel to the $xy$-plane, the surface tension and the $z$-component of the pressure can be coupled to a pressure bath. Presently, this only works with the Berendsen pressure coupling algorithm in GROMACS. The average surface tension $\gamma(t)$ can be calculated from the difference between the normal and the lateral pressure

$$\begin{aligned} \gamma(t) = & \frac{1}{n}\int_0^{L_z}\left\{P_{zz}(z,t) - \frac{P_{xx}(z,t)+P_{yy}(z,t)}{2}\right\}\mathrm{d}z \\ = & \frac{L_z}{n}\left\{P_{zz}(t) - \frac{P_{xx}(t)+P_{yy}(t)}{2}\right\}, \end{aligned} \tag{5.59}$$

where $L_z$ is the height of the box and $n$ is the number of surfaces. The pressure in the z-direction is corrected by scaling the height of the box with $\mu_{zz}$

$$\Delta P_{zz} = \frac{\Delta t}{\tau_p}\{P_{0zz} - P_{zz}(t)\} \tag{5.60}$$

$$\mu_{zz} = 1 + \beta_{zz}\Delta P_{zz} \tag{5.61}$$

This is similar to normal pressure coupling, except that the factor of $1/3$ is missing. The pressure correction in the $z$-direction is then used to get the correct convergence for the surface tension to the reference value $\gamma_0$. The correction factor for the box length in the $x$/$y$-direction is

$$\mu_{x/y} = 1 + \frac{\Delta t}{2\,\tau_p}\beta_{x/y}\left(\frac{n\gamma_0}{\mu_{zz}L_z} - \left\{P_{zz}(t) + \Delta P_{zz} - \frac{P_{xx}(t)+P_{yy}(t)}{2}\right\}\right) \tag{5.62}$$

The value of $\beta_{zz}$ is more critical than with normal pressure coupling. Normally an incorrect compressibility will just scale $\tau_p$, but with surface tension coupling it affects the convergence of the surface tension. When $\beta_{zz}$ is set to zero (constant box height), $\Delta P_{zz}$ is also set to zero, which is necessary for obtaining the correct surface tension.

### MTTK pressure control algorithms

As mentioned in the previous section, one weakness of leap-frog integration is in constant pressure simulations, since the pressure requires a calculation of both the virial and the kinetic energy at the full time step; for leap-frog, this information is not available until *after* the full timestep. Velocity Verlet does allow the calculation, at the cost of an extra round of global communication, and can compute, mod any integration errors, the true NPT ensemble.

The full equations, combining both pressure coupling and temperature coupling, are taken from Martyna *et al.* *35* (page 684) and Tuckerman *41* (page 685) and are referred to here as MTTK equations (Martyna-Tuckerman-Tobias-Klein). We introduce for convenience $\epsilon = (1/3)\ln(V/V_0)$, where $V_0$ is a

reference volume. The momentum of $\epsilon$ is $v_\epsilon = p_\epsilon/W = \dot{\epsilon} = \dot{V}/3V$, and define $\alpha = 1 + 3/N_{dof}$ (see Ref *41* (page 685))

The isobaric equations are

$$\begin{aligned}
\dot{\mathbf{r}}_i &= \frac{\mathbf{p}_i}{m_i} + \frac{p_\epsilon}{W}\mathbf{r}_i \\
\frac{\dot{\mathbf{p}}_i}{m_i} &= \frac{1}{m_i}\mathbf{F}_i - \alpha\frac{p_\epsilon}{W}\frac{\mathbf{p}_i}{m_i} \\
\dot{\epsilon} &= \frac{p_\epsilon}{W} \\
\frac{\dot{p_\epsilon}}{W} &= \frac{3V}{W}(P_{\text{int}} - P) + (\alpha - 1)\left(\sum_{n=1}^{N}\frac{\mathbf{p}_i^2}{m_i}\right),
\end{aligned}$$

where

$$P_{\text{int}} = P_{\text{kin}} - P_{\text{vir}} = \frac{1}{3V}\left[\sum_{i=1}^{N}\left(\frac{\mathbf{p}_i^2}{2m_i} - \mathbf{r}_i \cdot \mathbf{F}_i\right)\right]. \tag{5.63}$$

The terms including $\alpha$ are required to make phase space incompressible *41* (page 685). The $\epsilon$ acceleration term can be rewritten as

$$\frac{\dot{p_\epsilon}}{W} = \frac{3V}{W}\left(\alpha P_{\text{kin}} - P_{\text{vir}} - P\right) \tag{5.64}$$

In terms of velocities, these equations become

$$\begin{aligned}
\dot{\mathbf{r}}_i &= \mathbf{v}_i + v_\epsilon\mathbf{r}_i \\
\dot{\mathbf{v}}_i &= \frac{1}{m_i}\mathbf{F}_i - \alpha v_\epsilon\mathbf{v}_i \\
\dot{\epsilon} &= v_\epsilon \\
\dot{v_\epsilon} &= \frac{3V}{W}(P_{\text{int}} - P) + (\alpha - 1)\left(\sum_{n=1}^{N}\frac{1}{2}m_i\mathbf{v}_i^2\right) \\
P_{\text{int}} &= P_{\text{kin}} - P_{\text{vir}} = \frac{1}{3V}\left[\sum_{i=1}^{N}\left(\frac{1}{2}m_i\mathbf{v}_i^2 - \mathbf{r}_i \cdot \mathbf{F}_i\right)\right]
\end{aligned}$$

For these equations, the conserved quantity is

$$H = \sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{2m_i} + U\left(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N\right) + \frac{p_\epsilon}{2W} + PV \tag{5.65}$$

The next step is to add temperature control. Adding Nosé-Hoover chains, including to the barostat degree of freedom, where we use $\eta$ for the barostat Nosé-Hoover variables, and $Q'$ for the coupling

constants of the thermostats of the barostats, we get

$$\begin{aligned}
\dot{\mathbf{r}}_i &= \frac{\mathbf{p}_i}{m_i} + \frac{p_\epsilon}{W}\mathbf{r}_i \\
\frac{\dot{\mathbf{p}}_i}{m_i} &= \frac{1}{m_i}\mathbf{F}_i - \alpha\frac{p_\epsilon}{W}\frac{\mathbf{p}_i}{m_i} - \frac{p_{\xi_1}}{Q_1}\frac{\mathbf{p}_i}{m_i} \\
\dot{\epsilon} &= \frac{p_\epsilon}{W} \\
\frac{\dot{p_\epsilon}}{W} &= \frac{3V}{W}(\alpha P_{\text{kin}} - P_{\text{vir}} - P) - \frac{p_{\eta_1}}{Q'_1}p_\epsilon \\
\dot{\xi}_k &= \frac{p_{\xi_k}}{Q_k} \\
\dot{\eta}_k &= \frac{p_{\eta_k}}{Q'_k} \\
\dot{p}_{\xi_k} &= G_k - \frac{p_{\xi_{k+1}}}{Q_{k+1}} \quad k = 1, \ldots, M-1 \\
\dot{p}_{\eta_k} &= G'_k - \frac{p_{\eta_{k+1}}}{Q'_{k+1}} \quad k = 1, \ldots, M-1 \\
\dot{p}_{\xi_M} &= G_M \\
\dot{p}_{\eta_M} &= G'_M,
\end{aligned}$$

where

$$\begin{aligned}
P_{\text{int}} &= P_{\text{kin}} - P_{\text{vir}} = \frac{1}{3V}\left[\sum_{i=1}^{N}\left(\frac{\mathbf{p}_i^2}{2m_i} - \mathbf{r}_i \cdot \mathbf{F}_i\right)\right] \\
G_1 &= \sum_{i=1}^{N}\frac{\mathbf{p}_i^2}{m_i} - N_f kT \\
G_k &= \frac{p_{\xi_{k-1}}^2}{2Q_{k-1}} - kT \quad k = 2, \ldots, M \\
G'_1 &= \frac{{p_\epsilon}^2}{2W} - kT \\
G'_k &= \frac{p_{\eta_{k-1}}^2}{2Q'_{k-1}} - kT \quad k = 2, \ldots, M
\end{aligned}$$

The conserved quantity is now

$$H = \sum_{i=1}^{N}\frac{\mathbf{p}_i}{2m_i} + U\left(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N\right) + \frac{p_\epsilon^2}{2W} + PV +$$
$$\sum_{k=1}^{M}\frac{p_{\xi_k}^2}{2Q_k} + \sum_{k=1}^{M}\frac{p_{\eta_k}^2}{2Q'_k} + N_f kT\xi_1 + kT\sum_{i=2}^{M}\xi_k + kT\sum_{k=1}^{M}\eta_k$$

Returning to the Trotter decomposition formalism, for pressure control and temperature control *35* (page 684) we get:

$$iL = iL_1 + iL_2 + iL_{\epsilon,1} + iL_{\epsilon,2} + iL_{\text{NHC}-\text{baro}} + iL_{\text{NHC}} \tag{5.66}$$

where "NHC-baro" corresponds to the Nosè-Hoover chain of the barostat, and NHC corresponds to the

NHC of the particles,

$$\begin{aligned} iL_1 &= \sum_{i=1}^{N} \left[ \frac{\mathbf{p}_i}{m_i} + \frac{p_\epsilon}{W}\mathbf{r}_i \right] \cdot \frac{\partial}{\partial \mathbf{r}_i} \\ iL_2 &= \sum_{i=1}^{N} \mathbf{F}_i - \alpha \frac{p_\epsilon}{W}\mathbf{p}_i \cdot \frac{\partial}{\partial \mathbf{p}_i} \\ iL_{\epsilon,1} &= \frac{p_\epsilon}{W}\frac{\partial}{\partial \epsilon} \\ iL_{\epsilon,2} &= G_\epsilon \frac{\partial}{\partial p_\epsilon} \end{aligned} \tag{5.67}$$

and where

$$G_\epsilon = 3V\left(\alpha P_{\text{kin}} - P_{\text{vir}} - P\right) \tag{5.68}$$

Using the Trotter decomposition, we get

$$\begin{aligned} \exp(iL\Delta t) = & \exp\left(iL_{\text{NHC}-\text{baro}}\Delta t/2\right)\exp\left(iL_{\text{NHC}}\Delta t/2\right) \\ & \exp\left(iL_{\epsilon,2}\Delta t/2\right)\exp\left(iL_2\Delta t/2\right) \\ & \exp\left(iL_{\epsilon,1}\Delta t\right)\exp\left(iL_1\Delta t\right) \\ & \exp\left(iL_2\Delta t/2\right)\exp\left(iL_{\epsilon,2}\Delta t/2\right) \\ & \exp\left(iL_{\text{NHC}}\Delta t/2\right)\exp\left(iL_{\text{NHC}-\text{baro}}\Delta t/2\right) + \mathcal{O}(\Delta t^3) \end{aligned}$$

The action of $\exp\left(iL_1\Delta t\right)$ comes from the solution of the differential equation $\dot{\mathbf{r}}_i = \mathbf{v}_i + v_\epsilon \mathbf{r}_i$ with $\mathbf{v}_i = \mathbf{p}_i/m_i$ and $v_\epsilon$ constant with initial condition $\mathbf{r}_i(0)$, evaluate at $t = \Delta t$. This yields the evolution

$$\mathbf{r}_i(\Delta t) = \mathbf{r}_i(0)e^{v_\epsilon \Delta t} + \Delta t \mathbf{v}_i(0)e^{v_\epsilon \Delta t/2}\frac{\sinh\left(v_\epsilon \Delta t/2\right)}{v_\epsilon \Delta t/2}. \tag{5.69}$$

The action of $\exp\left(iL_2\Delta t/2\right)$ comes from the solution of the differential equation $\dot{\mathbf{v}}_i = \frac{\mathbf{F}_i}{m_i} - \alpha v_\epsilon \mathbf{v}_i$, yielding

$$\mathbf{v}_i(\Delta t/2) = \mathbf{v}_i(0)e^{-\alpha v_\epsilon \Delta t/2} + \frac{\Delta t}{2m_i}\mathbf{F}_i(0)e^{-\alpha v_\epsilon \Delta t/4}\frac{\sinh\left(\alpha v_\epsilon \Delta t/4\right)}{\alpha v_\epsilon \Delta t/4}. \tag{5.70}$$

*md-vv-avek* uses the full step kinetic energies for determining the pressure with the pressure control, but the half-step-averaged kinetic energy for the temperatures, which can be written as a Trotter decomposition as

$$\begin{aligned} \exp(iL\Delta t) = & \exp\left(iL_{\text{NHC}-\text{baro}}\Delta t/2\right)\exp\left(iL_{\epsilon,2}\Delta t/2\right)\exp\left(iL_2\Delta t/2\right) \\ & \exp\left(iL_{\text{NHC}}\Delta t/2\right)\exp\left(iL_{\epsilon,1}\Delta t\right)\exp\left(iL_1\Delta t\right)\exp\left(iL_{\text{NHC}}\Delta t/2\right) \\ & \exp\left(iL_2\Delta t/2\right)\exp\left(iL_{\epsilon,2}\Delta t/2\right)\exp\left(iL_{\text{NHC}-\text{baro}}\Delta t/2\right) + \mathcal{O}(\Delta t^3) \end{aligned}$$

With constraints, the equations become significantly more complicated, in that each of these equations need to be solved iteratively for the constraint forces. Before GROMACS 5.1, these iterative constraints were solved as described in *42* (page 685). From GROMACS 5.1 onward, MTTK with constraints has been removed because of numerical stability issues with the iterations.

### Infrequent evaluation of temperature and pressure coupling

Temperature and pressure control require global communication to compute the kinetic energy and virial, which can become costly if performed every step for large systems. We can rearrange the Trotter decomposition to give alternate symplectic, reversible integrator with the coupling steps every $n$ steps instead of every steps. These new integrators will diverge if the coupling time step is too large, as the auxiliary variable integrations will not converge. However, in most cases, long coupling times are more appropriate, as they disturb the dynamics less *35* (page 684).

Standard velocity Verlet with Nosé-Hoover temperature control has a Trotter expansion

$$\begin{aligned}\exp(iL\Delta t) \approx \quad & \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right)\exp\left(iL_2\Delta t/2\right) \\ & \exp\left(iL_1\Delta t\right)\exp\left(iL_2\Delta t/2\right)\exp\left(iL_{\mathrm{NHC}}\Delta t/2\right).\end{aligned}$$

If the Nosé-Hoover chain is sufficiently slow with respect to the motions of the system, we can write an alternate integrator over $n$ steps for velocity Verlet as

$$\begin{aligned}\exp(iL\Delta t) \approx \quad & \left(\exp\left(iL_{\mathrm{NHC}}(n\Delta t/2)\right)\left[\exp\left(iL_2\Delta t/2\right)\right.\right. \\ & \left.\left.\exp\left(iL_1\Delta t\right)\exp\left(iL_2\Delta t/2\right)\right]^n\exp\left(iL_{\mathrm{NHC}}(n\Delta t/2)\right)\right.\end{aligned}$$

For pressure control, this becomes

$$\begin{aligned}\exp(iL\Delta t) \approx \quad & \exp\left(iL_{\mathrm{NHC-baro}}(n\Delta t/2)\right)\exp\left(iL_{\mathrm{NHC}}(n\Delta t/2)\right) \\ & \exp\left(iL_{\epsilon,2}(n\Delta t/2)\right)\left[\exp\left(iL_2\Delta t/2\right)\right. \\ & \exp\left(iL_{\epsilon,1}\Delta t\right)\exp\left(iL_1\Delta t\right) \\ & \left.\exp\left(iL_2\Delta t/2\right)\right]^n\exp\left(iL_{\epsilon,2}(n\Delta t/2)\right) \\ & \exp\left(iL_{\mathrm{NHC}}(n\Delta t/2)\right)\exp\left(iL_{\mathrm{NHC-baro}}(n\Delta t/2)\right),\end{aligned}$$

where the box volume integration occurs every step, but the auxiliary variable integrations happen every $n$ steps.

### The complete update algorithm

**THE UPDATE ALGORITHM**

Given: Positions $\mathbf{r}$ of all atoms at time $t$ Velocities $\mathbf{v}$ of all atoms at time $t - \frac{1}{2}\Delta t$ Accelerations $\mathbf{F}/m$ on all atoms at time $t$. (Forces are computed disregarding any constraints) Total kinetic energy and virial at $t - \Delta t$ $\Downarrow$

1. Compute the scaling factors $\lambda$ and $\mu$ according to (5.38) and (5.49) $\Downarrow$
2. Update and scale velocities: $\mathbf{v}' = \lambda(\mathbf{v} + \mathbf{a}\Delta t)$ $\Downarrow$
3. Compute new unconstrained coordinates: $\mathbf{r}' = \mathbf{r} + \mathbf{v}'\Delta t$ $\Downarrow$
4. Apply constraint algorithm to coordinates: constrain($\mathbf{r}' \rightarrow \mathbf{r}''$; $\mathbf{r}$) $\Downarrow$
5. Correct velocities for constraints: $\mathbf{v} = (\mathbf{r}'' - \mathbf{r})/\Delta t$ $\Downarrow$
6. Scale coordinates and box: $\mathbf{r} = \mu\mathbf{r}''$; $\mathbf{b} = \mu\mathbf{b}$

The complete algorithm for the update of velocities and coordinates is given using leap-frog in *the outline above* (page 459) The SHAKE algorithm of step 4 is explained below.

GROMACS has a provision to *freeze* (prevent motion of) selected particles, which must be defined as a *freeze group*. This is implemented using a *freeze factor* $\mathbf{f}_g$, which is a vector, and differs for each freeze group (see sec. *The group concept* (page 432)). This vector contains only zero (freeze) or one (do not freeze). When we take this freeze factor and the external acceleration $\mathbf{a}_h$ into account the update algorithm for the velocities becomes

$$\mathbf{v}(t+\frac{\Delta t}{2}) \;=\; \mathbf{f}_g * \lambda * \left[\mathbf{v}(t-\frac{\Delta t}{2}) + \frac{\mathbf{F}(t)}{m}\Delta t + \mathbf{a}_h \Delta t\right], \tag{5.71}$$

where $g$ and $h$ are group indices which differ per atom.

### Output step

The most important output of the MD run is the *trajectory file*, which contains particle coordinates and (optionally) velocities at regular intervals. The trajectory file contains frames that could include positions, velocities and/or forces, as well as information about the dimensions of the simulation volume, integration step, integration time, etc. The interpretation of the time varies with the integrator chosen, as described above. For Velocity Verlet integrators, velocities labeled at time $t$ are for that time. For other integrators (e.g. leap-frog, stochastic dynamics), the velocities labeled at time $t$ are for time $t - \frac{1}{2}\Delta t$.

Since the trajectory files are lengthy, one should not save every step! To retain all information it suffices to write a frame every 15 steps, since at least 30 steps are made per period of the highest frequency in the system, and Shannon's sampling theorem states that two samples per period of the highest frequency in a band-limited signal contain all available information. But that still gives very long files! So, if the highest frequencies are not of interest, 10 or 20 samples per ps may suffice. Also, it can be awkward to down-sample long trajectories because of numerical difficulties in evaluating the difference in time between frames when that difference is close to the precision of the float-point data type used.

Be aware of the distortion of high-frequency motions by the *stroboscopic effect*, called *aliasing*: higher frequencies are mirrored with respect to the sampling frequency and appear as lower frequencies. When the simulated system is very large and/or the simulation times very long, it is often sufficient to write in intervals ranging from 10 ps to 1 ns, depending on what the trajectory will be used for.

GROMACS can also write reduced-precision coordinates for a subset of the simulation system to a special compressed trajectory file format. All the other tools can read and write this format. See the User Guide for details on how to set up your *mdp* (page 579) file to have *mdrun* (page 258) use this feature.

## 5.4.4 Shell molecular dynamics

GROMACS can simulate polarizability using the shell model of Dick and Overhauser *43* (page 685). In such models a shell particle representing the electronic degrees of freedom is attached to a nucleus by a spring. The potential energy is minimized with respect to the shell position at every step of the simulation (see below). Successful applications of shell models in GROMACS have been published for $N_2$ *44* (page 685) and water *45* (page 685).

### Optimization of the shell positions

The force $\mathbf{F}_S$ on a shell particle $S$ can be decomposed into two components

$$\mathbf{F}_S \;=\; \mathbf{F}_{bond} + \mathbf{F}_{nb} \tag{5.72}$$

where $\mathbf{F}_{bond}$ denotes the component representing the polarization energy, usually represented by a harmonic potential and $\mathbf{F}_{nb}$ is the sum of Coulomb and van der Waals interactions. If we assume that $\mathbf{F}_{nb}$ is almost constant we can analytically derive the optimal position of the shell, i.e. where $\mathbf{F}_S = 0$. If we have the shell S connected to atom A we have

$$\mathbf{F}_{bond} \;=\; k_b \left(\mathbf{x}_S - \mathbf{x}_A\right). \tag{5.73}$$

In an iterative solver, we have positions $\mathbf{x}_S(n)$ where $n$ is the iteration count. We now have at iteration $n$

$$\mathbf{F}_{nb} \;=\; \mathbf{F}_S - k_b \left(\mathbf{x}_S(n) - \mathbf{x}_A\right) \tag{5.74}$$

and the optimal position for the shells $x_S(n+1)$ thus follows from

$$\mathbf{F}_S - k_b \left(\mathbf{x}_S(n) - \mathbf{x}_A\right) + k_b \left(\mathbf{x}_S(n+1) - \mathbf{x}_A\right) = 0 \tag{5.75}$$

if we write

$$\Delta\mathbf{x}_S = \mathbf{x}_S(n+1) - \mathbf{x}_S(n) \tag{5.76}$$

we finally obtain

$$\Delta\mathbf{x}_S = \mathbf{F}_S/k_b \tag{5.77}$$

which then yields the algorithm to compute the next trial in the optimization of shell positions

$$\mathbf{x}_S(n+1) \;=\; \mathbf{x}_S(n) + \mathbf{F}_S/k_b. \tag{5.78}$$

## 5.4.5 Constraint algorithms

Constraints can be imposed in GROMACS using LINCS (default) or the traditional SHAKE method.

### SHAKE

The SHAKE *46* (page 685) algorithm changes a set of unconstrained coordinates $\mathbf{r}'$ to a set of coordinates $\mathbf{r}''$ that fulfill a list of distance constraints, using a set $\mathbf{r}$ reference, as

$$\mathrm{SHAKE}(\mathbf{r}' \rightarrow \mathbf{r}''; \mathbf{r}) \tag{5.79}$$

This action is consistent with solving a set of Lagrange multipliers in the constrained equations of motion. SHAKE needs a *relative tolerance*; it will continue until all constraints are satisfied within that relative tolerance. An error message is given if SHAKE cannot reset the coordinates because the deviation is too large, or if a given number of iterations is surpassed.

Assume the equations of motion must fulfill $K$ holonomic constraints, expressed as

$$\sigma_k(\mathbf{r}_1 \ldots \mathbf{r}_N) = 0; \;\; k = 1 \ldots K. \tag{5.80}$$

For example, $(\mathbf{r}_1 - \mathbf{r}_2)^2 - b^2 = 0$. Then the forces are defined as

$$-\frac{\partial}{\partial \mathbf{r}_i}\left(V + \sum_{k=1}^{K} \lambda_k \sigma_k\right), \tag{5.81}$$

where $\lambda_k$ are Lagrange multipliers which must be solved to fulfill the constraint equations. The second part of this sum determines the *constraint forces* $\mathbf{G}_i$, defined by

$$\mathbf{G}_i = -\sum_{k=1}^{K} \lambda_k \frac{\partial \sigma_k}{\partial \mathbf{r}_i} \tag{5.82}$$

The displacement due to the constraint forces in the leap-frog or Verlet algorithm is equal to $(\mathbf{G}_i/m_i)(\Delta t)^2$. Solving the Lagrange multipliers (and hence the displacements) requires the solution of a set of coupled equations of the second degree. These are solved iteratively by SHAKE. *SETTLE* (page 462)

### SETTLE

For the special case of rigid water molecules, that often make up more than 80% of the simulation system we have implemented the SETTLE algorithm *47* (page 685) (sec. *Constraint algorithms* (page 545)). The implementation of SETTLE in GROMACS is a slight modification of the original algorithm, in that it completely avoids the calculation of the center of mass of the water molecule. Apart from saving a few operations, the main gain of this is a reduction in rounding errors. For large coordinates, the floating pointing precision of constrained distances is reduced, which leads to an energy drift which usually depends quadratically on the coordinate. For SETTLE this dependence is now linear, which enables accurate integration of systems in single precision up to 1000 nm in size. But note that the drift due to SHAKE and LINCS still has a quadratic dependence, which limits the size of systems with normal constraints in single precision to 100 to 200 nm.

For velocity Verlet, an additional round of constraining must be done, to constrain the velocities of the second velocity half step, removing any component of the velocity parallel to the bond vector. This step is called RATTLE, and is covered in more detail in the original Andersen paper *48* (page 685).

### LINCS

#### The LINCS algorithm

LINCS is an algorithm that resets bonds to their correct lengths after an unconstrained update *49* (page 685). The method is non-iterative, as it always uses two steps. Although LINCS is based on matrices, no matrix-matrix multiplications are needed. The method is more stable and faster than SHAKE, but it can only be used with bond constraints and isolated angle constraints, such as the proton angle in OH. Because of its stability, LINCS is especially useful for Brownian dynamics. LINCS has two parameters, which are explained in the subsection parameters. The parallel version of LINCS, P-LINCS, is described in subsection *Constraints in parallel* (page 476).

### The LINCS formulas

We consider a system of $N$ particles, with positions given by a $3N$ vector $\mathbf{r}(t)$. For molecular dynamics the equations of motion are given by Newton's Law

$$\frac{\mathrm{d}^2\mathbf{r}}{\mathrm{d}t^2} = \mathbf{M}^{-1}\mathbf{F}, \tag{5.83}$$

where $\mathbf{F}$ is the $3N$ force vector and $\mathbf{M}$ is a $3N \times 3N$ diagonal matrix, containing the masses of the particles. The system is constrained by $K$ time-independent constraint equations

$$g_i(\mathbf{r}) = |\mathbf{r}_{i_1} - \mathbf{r}_{i_2}| - d_i = 0 \qquad i = 1, \ldots, K. \tag{5.84}$$

In a numerical integration scheme, LINCS is applied after an unconstrained update, just like SHAKE. The algorithm works in two steps (see figure Fig. 5.8). In the first step, the projections of the new bonds on the old bonds are set to zero. In the second step, a correction is applied for the lengthening of the bonds due to rotation. The numerics for the first step and the second step are very similar. A complete derivation of the algorithm can be found in *49* (page 685). Only a short description of the first step is given here.

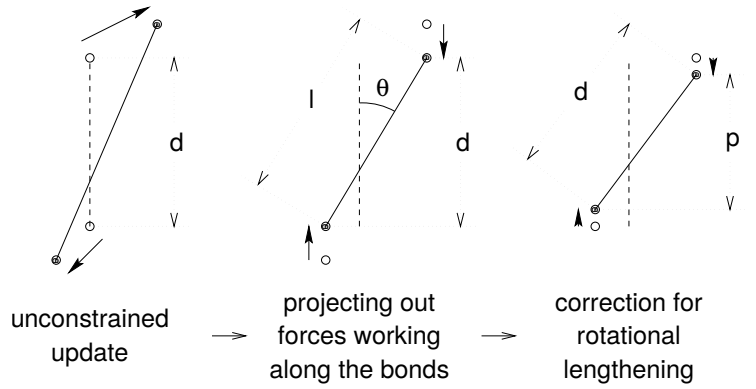



Fig. 5.8: The three position updates needed for one time step. The dashed line is the old bond of length $d$, the solid lines are the new bonds. $l = d\cos\theta$ and $p = (2d^2 - l^2)^{\frac{1}{2}}$.

A new notation is introduced for the gradient matrix of the constraint equations which appears on the right hand side of this equation:

$$B_{hi} = \frac{\partial g_h}{\partial r_i} \tag{5.85}$$

Notice that $\mathbf{B}$ is a $K \times 3N$ matrix, it contains the directions of the constraints. The following equation shows how the new constrained coordinates $\mathbf{r}_{n+1}$ are related to the unconstrained coordinates $\mathbf{r}_{n+1}^{unc}$ by

$$\begin{gathered}\mathbf{r}_{n+1} = (\mathbf{I} - \mathbf{T}_n\mathbf{B}_n)\mathbf{r}_{n+1}^{unc} + \mathbf{T}_n\mathbf{d} = \\ \mathbf{r}_{n+1}^{unc} - \mathbf{M}^{-1}\mathbf{B}_n(\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T)^{-1}(\mathbf{B}_n\mathbf{r}_{n+1}^{unc} - \mathbf{d})\end{gathered} \tag{5.86}$$

where

$$\mathbf{T} = \mathbf{M}^{-1}\mathbf{B}^T(\mathbf{B}\mathbf{M}^{-1}\mathbf{B}^T)^{-1} \tag{5.87}$$

The derivation of this equation from (5.83) and (5.84) can be found in *49* (page 685).

This first step does not set the real bond lengths to the prescribed lengths, but the projection of the new bonds onto the old directions of the bonds. To correct for the rotation of bond $i$, the projection of the bond, $p_i$, on the old direction is set to

$$p_i = \sqrt{2d_i^2 - l_i^2}, \tag{5.88}$$

where $l_i$ is the bond length after the first projection. The corrected positions are

$$\mathbf{r}^*_{n+1} = (\mathbf{I} - \mathbf{T}_n\mathbf{B}_n)\mathbf{r}_{n+1} + \mathbf{T}_n\mathbf{p}. \tag{5.89}$$

This correction for rotational effects is actually an iterative process, but during MD only one iteration is applied. The relative constraint deviation after this procedure will be less than 0.0001 for every constraint. In energy minimization, this might not be accurate enough, so the number of iterations is equal to the order of the expansion (see below).

Half of the CPU time goes to inverting the constraint coupling matrix $\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T$, which has to be done every time step. This $K \times K$ matrix has $1/m_{i_1} + 1/m_{i_2}$ on the diagonal. The off-diagonal elements are only non-zero when two bonds are connected, then the element is $\cos\phi/m_c$, where $m_c$ is the mass of the atom connecting the two bonds and $\phi$ is the angle between the bonds.

The matrix $\mathbf{T}$ is inverted through a power expansion. A $K \times K$ matrix $\mathbf{S}$ is introduced which is the inverse square root of the diagonal of $\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T$. This matrix is used to convert the diagonal elements of the coupling matrix to one:

$$\begin{aligned}(\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T)^{-1} &= \mathbf{S}\mathbf{S}^{-1}(\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T)^{-1}\mathbf{S}^{-1}\mathbf{S} \\ &= \mathbf{S}(\mathbf{S}\mathbf{B}_n\mathbf{M}^{-1}\mathbf{B}_n^T\mathbf{S})^{-1}\mathbf{S} = \mathbf{S}(\mathbf{I} - \mathbf{A}_n)^{-1}\mathbf{S}\end{aligned} \tag{5.90}$$

The matrix $\mathbf{A}_n$ is symmetric and sparse and has zeros on the diagonal. Thus a simple trick can be used to calculate the inverse:

$$(\mathbf{I} - \mathbf{A}_n)^{-1} = \mathbf{I} + \mathbf{A}_n + \mathbf{A}_n^2 + \mathbf{A}_n^3 + \ldots \tag{5.91}$$

This inversion method is only valid if the absolute values of all the eigenvalues of $\mathbf{A}_n$ are smaller than one. In molecules with only bond constraints, the connectivity is so low that this will always be true, even if ring structures are present. Problems can arise in angle-constrained molecules. By constraining angles with additional distance constraints, multiple small ring structures are introduced. This gives a high connectivity, leading to large eigenvalues. Therefore LINCS should NOT be used with coupled angle-constraints.

For molecules with all bonds constrained the eigenvalues of $A$ are around 0.4. This means that with each additional order in the expansion (5.91) the deviations decrease by a factor 0.4. But for relatively isolated triangles of constraints the largest eigenvalue is around 0.7. Such triangles can occur when removing hydrogen angle vibrations with an additional angle constraint in alcohol groups or when constraining water molecules with LINCS, for instance with flexible constraints. The constraints in such triangles converge twice as slow as the other constraints. Therefore, starting with GROMACS 4, additional terms are added to the expansion for such triangles

$$(\mathbf{I} - \mathbf{A}_n)^{-1} \approx \mathbf{I} + \mathbf{A}_n + \ldots + \mathbf{A}_n^{N_i} + \left(\mathbf{A}_n^* + \ldots + \mathbf{A}_n^{*\,N_i}\right)\mathbf{A}_n^{N_i} \tag{5.92}$$

where $N_i$ is the normal order of the expansion and $\mathbf{A}^*$ only contains the elements of $\mathbf{A}$ that couple constraints within rigid triangles, all other elements are zero. In this manner, the accuracy of angle constraints comes close to that of the other constraints, while the series of matrix vector multiplications required for determining the expansion only needs to be extended for a few constraint couplings. This procedure is described in the P-LINCS paper *50* (page 685).

#### The LINCS Parameters

The accuracy of LINCS depends on the number of matrices used in the expansion (5.91). For MD calculations a fourth order expansion is enough. For Brownian dynamics with large time steps an eighth order expansion may be necessary. The order is a parameter in the *mdp* (page 579) file. The implementation of LINCS is done in such a way that the algorithm will never crash. Even when it is impossible to to reset the constraints LINCS will generate a conformation which fulfills the constraints as well as possible. However, LINCS will generate a warning when in one step a bond rotates over more than a predefined angle. This angle is set by the user in the *mdp* (page 579) file.

### 5.4.6 Simulated Annealing

The well known simulated annealing (SA) protocol is supported in GROMACS, and you can even couple multiple groups of atoms separately with an arbitrary number of reference temperatures that change during the simulation. The annealing is implemented by simply changing the current reference temperature for each group in the temperature coupling, so the actual relaxation and coupling properties depends on the type of thermostat you use and how hard you are coupling it. Since we are changing the reference temperature it is important to remember that the system will NOT instantaneously reach this value - you need to allow for the inherent relaxation time in the coupling algorithm too. If you are changing the annealing reference temperature faster than the temperature relaxation you will probably end up with a crash when the difference becomes too large.

The annealing protocol is specified as a series of corresponding times and reference temperatures for each group, and you can also choose whether you only want a single sequence (after which the temperature will be coupled to the last reference value), or if the annealing should be periodic and restart at the first reference point once the sequence is completed. You can mix and match both types of annealing and non-annealed groups in your simulation.

### 5.4.7 Stochastic Dynamics

Stochastic or velocity Langevin dynamics adds a friction and a noise term to Newton's equations of motion, as

$$m_i \frac{\mathrm{d}^2 \mathbf{r}_i}{\mathrm{d}t^2} = -m_i \gamma_i \frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} + \mathbf{F}_i(\mathbf{r}) + \mathring{\mathbf{r}}_i, \tag{5.93}$$

where $\gamma_i$ is the friction constant $[1/\mathrm{ps}]$ and $\mathring{\mathbf{r}}_i(t)$ is a noise process with $\langle \mathring{r}_i(t) \mathring{r}_j(t+s) \rangle = 2 m_i \gamma_i k_B T \delta(s) \delta_{ij}$. When $1/\gamma_i$ is large compared to the time scales present in the system, one could see stochastic dynamics as molecular dynamics with stochastic temperature-coupling. But any processes that take longer than $1/\gamma_i$, e.g. hydrodynamics, will be dampened. Since each degree of freedom is coupled independently to a heat bath, equilibration of fast modes occurs rapidly. For simulating a system in vacuum there is the additional advantage that there is no accumulation of errors for the overall translational and rotational degrees of freedom. When $1/\gamma_i$ is small compared to the time scales present in the system, the dynamics will be completely different from MD, but the sampling is still correct.

In GROMACS there is one simple and efficient implementation. Its accuracy is equivalent to the normal MD leap-frog and Velocity Verlet integrator. It is nearly identical to the common way of discretizing the Langevin equation, but the friction and velocity term are applied in an impulse fashion *51* (page 685). It

can be described as:

$$\begin{aligned} \mathbf{v}' &= \mathbf{v}(t-\frac{1}{2}\Delta t) + \frac{1}{m}\mathbf{F}(t)\Delta t \\ \Delta\mathbf{v} &= -\alpha\,\mathbf{v}'(t+\frac{1}{2}\Delta t) + \sqrt{\frac{k_B T}{m}\alpha(2-\alpha)}\,\mathbf{r}^G{}_i \\ \mathbf{r}(t+\Delta t) &= \mathbf{r}(t) + \left(\mathbf{v}' + \frac{1}{2}\Delta\mathbf{v}\right)\Delta t \end{aligned} \tag{5.94}$$

$$\begin{aligned} \mathbf{v}(t+\frac{1}{2}\Delta t) &= \mathbf{v}' + \Delta\mathbf{v} \\ \alpha &= 1 - e^{-\gamma\Delta t} \end{aligned} \tag{5.95}$$

where $\mathbf{r}^G{}_i$ is Gaussian distributed noise with $\mu = 0$, $\sigma = 1$. The velocity is first updated a full time step without friction and noise to get $\mathbf{v}'$, identical to the normal update in leap-frog. The friction and noise are then applied as an impulse at step $t + \Delta t$. The advantage of this scheme is that the velocity-dependent terms act at the full time step, which makes the correct integration of forces that depend on both coordinates and velocities, such as constraints and dissipative particle dynamics (DPD, not implemented yet), straightforward. With constraints, the coordinate update (5.95) is split into a normal leap-frog update and a $\Delta\mathbf{v}$. After both of these updates the constraints are applied to coordinates and velocities.

SD can be chosen as an integrator by `integrator=sd` (page 52). The simulations are performed using the *mdrun* (page 258) program. When using SD as a thermostat, an appropriate value for $\gamma$ is e.g. 0.5 ps$^{-1}$, since this results in a friction that is lower than the internal friction of water, while it still provides efficient thermostatting.

### 5.4.8 Brownian Dynamics

In the limit of high friction, stochastic dynamics reduces to Brownian dynamics, also called position Langevin dynamics. This applies to over-damped systems, *i.e.* systems in which the inertia effects are negligible. The equation is

$$\frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} = \frac{1}{\gamma_i}\mathbf{F}_i(\mathbf{r}) + \mathring{\mathbf{r}}_i \tag{5.96}$$

where $\gamma_i$ is the friction coefficient [amu/ps] and $\mathring{\mathbf{r}}_i(t)$ is a noise process with $\langle \mathring{r}_i(t)\mathring{r}_j(t+s)\rangle = 2\delta(s)\delta_{ij}k_B T/\gamma_i$. In GROMACS the equations are integrated with a simple, explicit scheme

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \frac{\Delta t}{\gamma_i}\mathbf{F}_i(\mathbf{r}(t)) + \sqrt{2k_B T\frac{\Delta t}{\gamma_i}}\,\mathbf{r}^G{}_i, \tag{5.97}$$

where $\mathbf{r}^G{}_i$ is Gaussian distributed noise with $\mu = 0$, $\sigma = 1$. The friction coefficients $\gamma_i$ can be chosen the same for all particles or as $\gamma_i = m_i\,\gamma_i$, where the friction constants $\gamma_i$ can be different for different groups of atoms. Because the system is assumed to be over-damped, large timesteps can be used. LINCS should be used for the constraints since SHAKE will not converge for large atomic displacements. BD can be activated by using `integrator=bd` (page 52) and the simulations are run using the *mdrun* (page 258) program.

In BD there are no velocities, so there is also no kinetic energy. Still *gmx mdrun* (page 258) will report a kinetic energy and temperature based on atom displacements per step $\Delta x$. This can be used to judge the quality of the integration. A too high temperature is an indication that the time step chosen is too large. The formula for the kinetic energy term reported is:

$$\frac{1}{2}\sum_i \frac{\gamma_i \Delta x_i^2}{2\,\Delta t} \tag{5.98}$$

### 5.4.9 Energy Minimization

Energy minimization in GROMACS can be done using steepest descent, conjugate gradients, or l-bfgs (limited-memory Broyden-Fletcher-Goldfarb-Shanno quasi-Newtonian minimizer... we prefer the abbreviation). Whether to use EM, and which algorithm to use, is specified via the `integrator` (page 51) setting of the *mdrun* (page 258) program.

#### Steepest Descent

Although steepest descent is certainly not the most efficient algorithm for searching, it is robust and easy to implement.

We define the vector $\mathbf{r}$ as the vector of all $3N$ coordinates. Initially a maximum displacement $h_0$ (*e.g.* 0.01 nm) must be given.

First the forces $\mathbf{F}$ and potential energy are calculated. New positions are calculated by

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \frac{\mathbf{F}_n}{\max(|\mathbf{F}_n|)} h_n, \tag{5.99}$$

where $h_n$ is the maximum displacement and $\mathbf{F}_n$ is the force, or the negative gradient of the potential $V$. The notation $\max(|\mathbf{F}_n|)$ means the largest scalar force on any atom. The forces and energy are again computed for the new positions

If ($V_{n+1} < V_n$) the new positions are accepted and $h_{n+1} = 1.2h_n$.
If ($V_{n+1} \geq V_n$) the new positions are rejected and $h_n = 0.2h_n$.

The algorithm stops when either a user-specified number of force evaluations has been performed (*e.g.* 100), or when the maximum of the absolute values of the force (gradient) components is smaller than a specified value $\epsilon$. Since force truncation produces some noise in the energy evaluation, the stopping criterion should not be made too tight to avoid endless iterations. A reasonable value for $\epsilon$ can be estimated from the root mean square force $f$ a harmonic oscillator would exhibit at a temperature $T$. This value is

$$f = 2\pi\nu\sqrt{2mkT}, \tag{5.100}$$

where $\nu$ is the oscillator frequency, $m$ the (reduced) mass, and $k$ Boltzmann's constant. For a weak oscillator with a wave number of 100 cm$^{-1}$ and a mass of 10 atomic units, at a temperature of 1 K, $f = 7.7$ kJ mol$^{-1}$ nm$^{-1}$. A value for $\epsilon$ between 1 and 10 is acceptable.

#### Conjugate Gradient

Conjugate gradient is slower than steepest descent in the early stages of the minimization, but becomes more efficient closer to the energy minimum. The parameters and stop criterion are the same as for steepest descent. The most common use case for conjugate gradient is minimization prior to a normal-mode analysis, which requires very small forces. For most other purposes steepest descent is efficient enough.

### L-BFGS

The original BFGS algorithm works by successively creating better approximations of the inverse Hessian matrix, and moving the system to the currently estimated minimum. The memory requirements for this are proportional to the square of the number of particles, so it is not practical for large systems like biomolecules. Instead, we use the L-BFGS algorithm of Nocedal *52* (page 685), *53* (page 685), which approximates the inverse Hessian by a fixed number of corrections from previous steps. This sliding-window technique is almost as efficient as the original method, but the memory requirements are much lower - proportional to the number of particles multiplied with the correction steps. In practice we have found it to converge faster than conjugate gradients, but due to the correction steps it is not yet parallelized. It is also noteworthy that switched or shifted interactions usually improve the convergence, since sharp cut-offs mean the potential function at the current coordinates is slightly different from the previous steps used to build the inverse Hessian approximation.

## 5.4.10 Normal-Mode Analysis

Normal-mode analysis *54* (page 685)*56* (page 685) can be performed using GROMACS, by diagonalization of the mass-weighted Hessian $H$:

$$\begin{aligned} R^T M^{-1/2} H M^{-1/2} R = &\quad \mathrm{diag}(\lambda_1, \ldots, \lambda_{3N}) \\ \lambda_i = &\quad (2\pi\omega_i)^2 \end{aligned} \tag{5.101}$$

where $M$ contains the atomic masses, $R$ is a matrix that contains the eigenvectors as columns, $\lambda_i$ are the eigenvalues and $\omega_i$ are the corresponding frequencies.

First the Hessian matrix, which is a $3N \times 3N$ matrix where $N$ is the number of atoms, needs to be calculated:

$$H_{ij} = \frac{\partial^2 V}{\partial x_i \partial x_j} \tag{5.102}$$

where $x_i$ and $x_j$ denote the atomic x, y or z coordinates. In practice, this equation is not used, but the Hessian is calculated numerically from the force as:

$$\begin{aligned} H_{ij} = &\quad -\frac{f_i(\mathbf{x} + h\mathbf{e}_j) - f_i(\mathbf{x} - h\mathbf{e}_j)}{2h} \\ f_i = &\quad -\frac{\partial V}{\partial x_i} \end{aligned} \tag{5.103}$$

where $\mathbf{e}_j$ is the unit vector in direction $j$. It should be noted that for a usual normal-mode calculation, it is necessary to completely minimize the energy prior to computation of the Hessian. The tolerance required depends on the type of system, but a rough indication is 0.001 kJ mol$^{-1}$. Minimization should be done with conjugate gradients or L-BFGS in double precision.

A number of GROMACS programs are involved in these calculations. First, the energy should be minimized using *mdrun* (page 258). Then, *mdrun* (page 258) computes the Hessian. **Note** that for generating the run input file, one should use the minimized conformation from the full precision trajectory file, as the structure file is not accurate enough. *gmx nmeig* (page 269) does the diagonalization and the sorting of the normal modes according to their frequencies. Both *mdrun* (page 258) and *gmx nmeig* (page 269) should be run in double precision. The normal modes can be analyzed with the program *gmx anaeig* (page 151). Ensembles of structures at any temperature and for any subset of normal modes can be generated with *gmx nmens* (page 271). An overview of normal-mode analysis and the related principal component analysis (see sec. *Covariance analysis* (page 666)) can be found in *57* (page 685).

### 5.4.11 Free energy calculations

#### Slow-growth methods

Free energy calculations can be performed in GROMACS using a number of methods, including "slow-growth." An example problem might be calculating the difference in free energy of binding of an inhibitor **I** to an enzyme **E** and to a mutated enzyme **E**′. It is not feasible with computer simulations to perform a docking calculation for such a large complex, or even releasing the inhibitor from the enzyme in a reasonable amount of computer time with reasonable accuracy. However, if we consider the free energy cycle in Fig. 5.9 A we can write:

$$\Delta G_1 - \Delta G_2 = \Delta G_3 - \Delta G_4 \tag{5.104}$$

If we are interested in the left-hand term we can equally well compute the right-hand term.

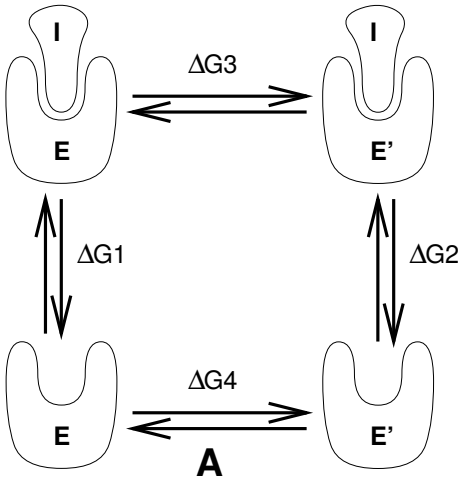



Fig. 5.9: Free energy cycles. **A:** to calculate $\Delta G_{12}$, the free energy difference between the binding of inhibitor **I** to enzymes **E** respectively **E**′.

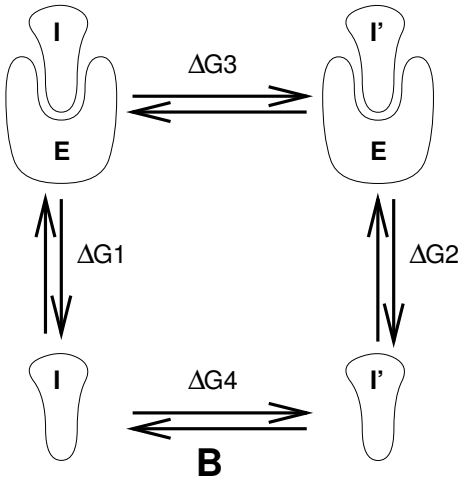



Fig. 5.10: Free energy cycles. **B:** to calculate $\Delta G_{12}$, the free energy difference for binding of inhibitors **I** respectively **I**′ to enzyme **E**.

If we want to compute the difference in free energy of binding of two inhibitors **I** and **I**′ to an enzyme **E** (Fig. 5.10) we can again use (5.104) to compute the desired property.

Free energy differences between two molecular species can be calculated in GROMACS using the "slow-growth" method. Such free energy differences between different molecular species are physically meaningless, but they can be used to obtain meaningful quantities employing a thermodynamic cycle. The method requires a simulation during which the Hamiltonian of the system changes slowly from that describing one system (A) to that describing the other system (B). The change must be so slow that the system remains in equilibrium during the process; if that requirement is fulfilled, the change is reversible and a slow-growth simulation from B to A will yield the same results (but with a different sign) as a slow-growth simulation from A to B. This is a useful check, but the user should be aware of the danger that equality of forward and backward growth results does not guarantee correctness of the results.

The required modification of the Hamiltonian $H$ is realized by making $H$ a function of a *coupling parameter* $\lambda : H = H(p, q; \lambda)$ in such a way that $\lambda = 0$ describes system A and $\lambda = 1$ describes system B:

$$H(p, q; 0) = H^{\mathrm{A}}(p, q); \quad H(p, q; 1) = H^{\mathrm{B}}(p, q). \tag{5.105}$$

In GROMACS, the functional form of the $\lambda$-dependence is different for the various force-field contributions and is described in section sec. *Free energy interactions* (page 511).

The Helmholtz free energy $A$ is related to the partition function $Q$ of an $N, V, T$ ensemble, which is assumed to be the equilibrium ensemble generated by a MD simulation at constant volume and temperature. The generally more useful Gibbs free energy $G$ is related to the partition function $\Delta$ of an $N, p, T$ ensemble, which is assumed to be the equilibrium ensemble generated by a MD simulation at constant pressure and temperature:

$$\begin{aligned} A(\lambda) &= -k_B T \ln Q \\ Q &= c \iint \exp[-\beta H(p, q; \lambda)]\, dp\, dq \\ G(\lambda) &= -k_B T \ln \Delta \\ \Delta &= c \iiint \exp[-\beta H(p, q; \lambda) - \beta p V]\, dp\, dq\, dV \\ G &= A + pV, \end{aligned} \tag{5.106}$$

where $\beta = 1/(k_B T)$ and $c = (N! h^{3N})^{-1}$. These integrals over phase space cannot be evaluated from a simulation, but it is possible to evaluate the derivative with respect to $\lambda$ as an ensemble average:

$$\frac{dA}{d\lambda} = \frac{\iint (\partial H/\partial\lambda) \exp[-\beta H(p, q; \lambda)]\, dp\, dq}{\iint \exp[-\beta H(p, q; \lambda)]\, dp\, dq} = \left\langle \frac{\partial H}{\partial \lambda} \right\rangle_{NVT;\lambda}, \tag{5.107}$$

with a similar relation for $dG/d\lambda$ in the $N, p, T$ ensemble. The difference in free energy between A and B can be found by integrating the derivative over $\lambda$:

$$A^{\mathrm{B}}(V, T) - A^{\mathrm{A}}(V, T) = \int_0^1 \left\langle \frac{\partial H}{\partial \lambda} \right\rangle_{NVT;\lambda} d\lambda \tag{5.108}$$

$$G^{\mathrm{B}}(p, T) - G^{\mathrm{A}}(p, T) = \int_0^1 \left\langle \frac{\partial H}{\partial \lambda} \right\rangle_{NpT;\lambda} d\lambda. \tag{5.109}$$

If one wishes to evaluate $G^{\mathrm{B}}(p, T) - G^{\mathrm{A}}(p, T)$, the natural choice is a constant-pressure simulation. However, this quantity can also be obtained from a slow-growth simulation at constant volume, starting with system A at pressure $p$ and volume $V$ and ending with system B at pressure $p_B$, by applying the following small (but, in principle, exact) correction:

$$G^{\mathrm{B}}(p) - G^{\mathrm{A}}(p) = A^{\mathrm{B}}(V) - A^{\mathrm{A}}(V) - \int_p^{p^{\mathrm{B}}} [V^{\mathrm{B}}(p') - V]\, dp' \tag{5.110}$$

Here we omitted the constant $T$ from the notation. This correction is roughly equal to $-\frac{1}{2}(p^{\mathrm{B}}-p)\Delta V = (\Delta V)^2/(2\kappa V)$, where $\Delta V$ is the volume change at $p$ and $\kappa$ is the isothermal compressibility. This is usually small; for example, the growth of a water molecule from nothing in a bath of 1000 water molecules at constant volume would produce an additional pressure of as much as 22 bar, but a correction to the Helmholtz free energy of just -1 kJ mol$^{-1}$. In Cartesian coordinates, the kinetic energy term in the Hamiltonian depends only on the momenta, and can be separately integrated and, in fact, removed from the equations. When masses do not change, there is no contribution from the kinetic energy at all; otherwise the integrated contribution to the free energy is $-\frac{3}{2}k_BT\ln(m^{\mathrm{B}}/m^{\mathrm{A}})$. **Note** that this is only true in the absence of constraints.

#### Thermodynamic integration

GROMACS offers the possibility to integrate (5.108) or eq. (5.109) in one simulation over the full range from A to B. However, if the change is large and insufficient sampling can be expected, the user may prefer to determine the value of $\langle dG/d\lambda\rangle$ accurately at a number of well-chosen intermediate values of $\lambda$. This can easily be done by setting the stepsize `delta_lambda` to zero. Each simulation can be equilibrated first, and a proper error estimate can be made for each value of $dG/d\lambda$ from the fluctuation of $\partial H/\partial\lambda$. The total free energy change is then determined afterward by an appropriate numerical integration procedure.

GROMACS now also supports the use of Bennett's Acceptance Ratio *58* (page 686) for calculating values of $\Delta$G for transformations from state A to state B using the program *gmx bar* (page 161). The same data can also be used to calculate free energies using MBAR *59* (page 686), though the analysis currently requires external tools from the external pymbar package.

The $\lambda$-dependence for the force-field contributions is described in detail in section sec. *Free energy interactions* (page 511).

### 5.4.12 Replica exchange

Replica exchange molecular dynamics (REMD) is a method that can be used to speed up the sampling of any type of simulation, especially if conformations are separated by relatively high energy barriers. It involves simulating multiple replicas of the same system at different temperatures and randomly exchanging the complete state of two replicas at regular intervals with the probability:

$$P(1 \leftrightarrow 2) = \min\left(1, \exp\left[\left(\frac{1}{k_BT_1} - \frac{1}{k_BT_2}\right)(U_1 - U_2)\right]\right) \tag{5.111}$$

where $T_1$ and $T_2$ are the reference temperatures and $U_1$ and $U_2$ are the instantaneous potential energies of replicas 1 and 2 respectively. After exchange the velocities are scaled by $(T_1/T_2)^{\pm 0.5}$ and a neighbor search is performed the next step. This combines the fast sampling and frequent barrier-crossing of the highest temperature with correct Boltzmann sampling at all the different temperatures *60* (page 686), *61* (page 686). We only attempt exchanges for neighboring temperatures as the probability decreases very rapidly with the temperature difference. One should not attempt exchanges for all possible pairs in one step. If, for instance, replicas 1 and 2 would exchange, the chance of exchange for replicas 2 and 3 not only depends on the energies of replicas 2 and 3, but also on the energy of replica 1. In GROMACS this is solved by attempting exchange for all *odd* pairs on *odd* attempts and for all *even* pairs on *even* attempts. If we have four replicas: 0, 1, 2 and 3, ordered in temperature and we attempt exchange every 1000 steps, pairs 0-1 and 2-3 will be tried at steps 1000, 3000 etc. and pair 1-2 at steps 2000, 4000 etc.

How should one choose the temperatures? The energy difference can be written as:

$$U_1 - U_2 = N_{df}\frac{c}{2}k_B(T_1 - T_2) \tag{5.112}$$

where $N_{df}$ is the total number of degrees of freedom of one replica and $c$ is 1 for harmonic potentials and around 2 for protein/water systems. If $T_2 = (1+\epsilon)T_1$ the probability becomes:

$$P(1 \leftrightarrow 2) = \exp\left(-\frac{\epsilon^2 c\, N_{df}}{2(1+\epsilon)}\right) \approx \exp\left(-\epsilon^2 \frac{c}{2} N_{df}\right) \tag{5.113}$$

Thus for a probability of $e^{-2} \approx 0.135$ one obtains $\epsilon \approx 2/\sqrt{c\, N_{df}}$. With all bonds constrained one has $N_{df} \approx 2\, N_{atoms}$ and thus for $c$ = 2 one should choose $\epsilon$ as $1/\sqrt{N_{atoms}}$. However there is one problem when using pressure coupling. The density at higher temperatures will decrease, leading to higher energy *62* (page 686), which should be taken into account. Using a so-called REMD calculator, you can type in the temperature range and the number of atoms. The tool then proposes a set of temperatures.

An extension to the REMD for the isobaric-isothermal ensemble was proposed by Okabe et al. *63* (page 686). In this work the exchange probability is modified to:

$$P(1 \leftrightarrow 2) = \min\left(1, \exp\left[\left(\frac{1}{k_B T_1} - \frac{1}{k_B T_2}\right)(U_1 - U_2) + \left(\frac{P_1}{k_B T_1} - \frac{P_2}{k_B T_2}\right)(V_1 - V_2)\right]\right) \tag{5.114}$$

where $P_1$ and $P_2$ are the respective reference pressures and $V_1$ and $V_2$ are the respective instantaneous volumes in the simulations. In most cases the differences in volume are so small that the second term is negligible. It only plays a role when the difference between $P_1$ and $P_2$ is large or in phase transitions.

Hamiltonian replica exchange is also supported in GROMACS. In Hamiltonian replica exchange, each replica has a different Hamiltonian, defined by the free energy pathway specified for the simulation. The exchange probability to maintain the correct ensemble probabilities is:

$$P(1 \leftrightarrow 2) = \min\left(1, \exp\left[\frac{1}{k_B T}(U_1(x_1) - U_1(x_2) + U_2(x_2) - U_2(x_1))\right]\right) \tag{5.115}$$

The separate Hamiltonians are defined by the free energy functionality of GROMACS, with swaps made between the different values of $\lambda$ defined in the mdp file.

Hamiltonian and temperature replica exchange can also be performed simultaneously *64* (page 686), using the acceptance criteria:

$$P(1 \leftrightarrow 2) = \min\left(1, \exp\left[\frac{U_1(x_1) - U_1(x_2)}{k_B T_1} + \frac{U_2(x_2) - U_2(x_1)}{k_B T_2}\right]\right) \tag{5.116}$$

Gibbs sampling replica exchange has also been implemented in GROMACS *64* (page 686). In Gibbs sampling replica exchange, all possible pairs are tested for exchange, allowing swaps between replicas that are not neighbors.

Gibbs sampling replica exchange requires no additional potential energy calculations. However there is an additional communication cost in Gibbs sampling replica exchange, as for some permutations, more than one round of swaps must take place. In some cases, this extra communication cost might affect the efficiency.

All replica exchange variants are set using *mdp* (page 579) options and performed using the *mdrun* (page 258) program. It will only work when MPI is installed, due to the inherent parallelism in the algorithm. For efficiency each replica can run on a separate rank. See the manual page of *mdrun* (page 258) on how to use these multinode features.

### 5.4.13 Essential Dynamics sampling

The results from Essential Dynamics (see sec. *Covariance analysis* (page 666)) of a protein can be used to guide MD simulations. The idea is that from an initial MD simulation (or from other sources) a definition of the collective fluctuations with largest amplitude is obtained. The position along one or more of these collective modes can be constrained in a (second) MD simulation in a number of ways for several purposes. For example, the position along a certain mode may be kept fixed to monitor the average force (free-energy gradient) on that coordinate in that position. Another application is to enhance sampling efficiency with respect to usual MD *65* (page 686), *66* (page 686). In this case, the system is encouraged to sample its available configuration space more systematically than in a diffusion-like path that proteins usually take.

Another possibility to enhance sampling is flooding. Here a flooding potential is added to certain (collective) degrees of freedom to expel the system out of a region of phase space *67* (page 686).

The procedure for essential dynamics sampling or flooding is as follows. First, the eigenvectors and eigenvalues need to be determined using covariance analysis (*gmx covar* (page 181)) or normal-mode analysis (*gmx nmeig* (page 269)). Then, this information is fed into *make_edi* (page 251), which has many options for selecting vectors and setting parameters, see `gmx make_edi -h`. The generated *edi* (page 576) input file is then passed to *mdrun* (page 258).

### 5.4.14 Expanded Ensemble

In an expanded ensemble simulation *68* (page 686), both the coordinates and the thermodynamic ensemble are treated as configuration variables that can be sampled over. The probability of any given state can be written as:

$$P(\vec{x}, k) \propto \exp\left(-\beta_k U_k + g_k\right), \tag{5.117}$$

where $\beta_k = \frac{1}{k_B T_k}$ is the $\beta$ corresponding to the $k$th thermodynamic state, and $g_k$ is a user-specified weight factor corresponding to the $k$th state. This space is therefore a *mixed*, *generalized*, or *expanded* ensemble which samples from multiple thermodynamic ensembles simultaneously. $g_k$ is chosen to give a specific weighting of each subensemble in the expanded ensemble, and can either be fixed, or determined by an iterative procedure. The set of $g_k$ is frequently chosen to give each thermodynamic ensemble equal probability, in which case $g_k$ is equal to the free energy in non-dimensional units, but they can be set to arbitrary values as desired. Several different algorithms can be used to equilibrate these weights, described in the mdp option listings.

In GROMACS, this space is sampled by alternating sampling in the $k$ and $\vec{x}$ directions. Sampling in the $\vec{x}$ direction is done by standard molecular dynamics sampling; sampling between the different thermodynamics states is done by Monte Carlo, with several different Monte Carlo moves supported. The $k$ states can be defined by different temperatures, or choices of the free energy $\lambda$ variable, or both. Expanded ensemble simulations thus represent a serialization of the replica exchange formalism, allowing a single simulation to explore many thermodynamic states.

### 5.4.15 Parallelization

The CPU time required for a simulation can be reduced by running the simulation in parallel over more than one core. Ideally, one would want to have linear scaling: running on $N$ cores makes the simulation $N$ times faster. In practice this can only be achieved for a small number of cores. The scaling will depend a lot on the algorithms used. Also, different algorithms can have different restrictions on the interaction ranges between atoms.

### 5.4.16 Domain decomposition

Since most interactions in molecular simulations are local, domain decomposition is a natural way to decompose the system. In domain decomposition, a spatial domain is assigned to each rank, which will then integrate the equations of motion for the particles that currently reside in its local domain. With domain decomposition, there are two choices that have to be made: the division of the unit cell into domains and the assignment of the forces to domains. Most molecular simulation packages use the half-shell method for assigning the forces. But there are two methods that always require less communication: the eighth shell *69* (page 686) and the midpoint *70* (page 686) method. GROMACS currently uses the eighth shell method, but for certain systems or hardware architectures it might be advantageous to use the midpoint method. Therefore, we might implement the midpoint method in the future. Most of the details of the domain decomposition can be found in the GROMACS 4 paper *5* (page 683).

#### Coordinate and force communication

In the most general case of a triclinic unit cell, the space in divided with a 1-, 2-, or 3-D grid in parallelepipeds that we call domain decomposition cells. Each cell is assigned to a particle-particle rank. The system is partitioned over the ranks at the beginning of each MD step in which neighbor searching is performed. The minimum unit of partitioning can be an atom, or a charge group with the (deprecated) group cut-off scheme or an update group. An update group is a group of atoms that has dependencies during update, which occurs when using constraints and/or virtual sites. Thus different update groups can be updated independently. Currently update groups can only be used with at most two sequential constraints, which is the case when only constraining bonds involving hydrogen atoms. The advantages of update groups are that no communication is required in the update and that this allows updating part of the system while computing forces for other parts. Atom groups are assigned to the cell where their center of geometry resides. Before the forces can be calculated, the coordinates from some neighboring cells need to be communicated, and after the forces are calculated, the forces need to be communicated in the other direction. The communication and force assignment is based on zones that can cover one or multiple cells. An example of a zone setup is shown in Fig. 5.11.

The coordinates are communicated by moving data along the "negative" direction in $x$, $y$ or $z$ to the next neighbor. This can be done in one or multiple pulses. In Fig. 5.11 two pulses in $x$ are required, then one in $y$ and then one in $z$. The forces are communicated by reversing this procedure. See the GROMACS 4 paper *5* (page 683) for details on determining which non-bonded and bonded forces should be calculated on which rank.

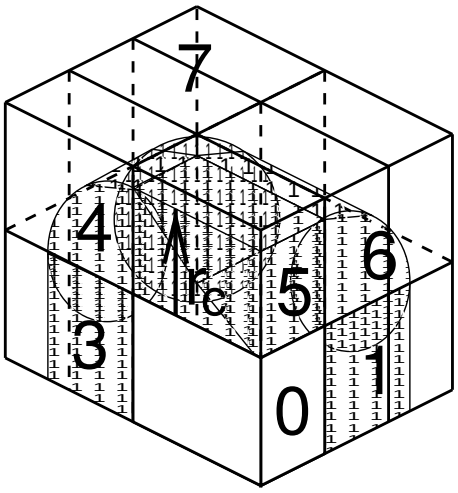



Fig. 5.11: A non-staggered domain decomposition grid of 3×2×2 cells. Coordinates in zones 1 to 7 are communicated to the corner cell that has its home particles in zone 0. $r_c$ is the cut-off radius.

### Dynamic load balancing

When different ranks have a different computational load (load imbalance), all ranks will have to wait for the one that takes the most time. One would like to avoid such a situation. Load imbalance can occur due to four reasons:

- inhomogeneous particle distribution
- inhomogeneous interaction cost distribution (charged/uncharged, water/non-water due to GROMACS water innerloops)
- statistical fluctuation (only with small particle numbers)
- differences in communication time, due to network topology and/or other jobs on the machine interfering with our communication

So we need a dynamic load balancing algorithm where the volume of each domain decomposition cell can be adjusted *independently*. To achieve this, the 2- or 3-D domain decomposition grids need to be staggered. Fig. 5.12 shows the most general case in 2-D. Due to the staggering, one might require two distance checks for deciding if an update group needs to be communicated: a non-bonded distance and a bonded distance check.

By default, *mdrun* (page 258) automatically turns on the dynamic load balancing during a simulation when the total performance loss due to the force calculation imbalance is 2% or more. **Note** that the reported force load imbalance numbers might be higher, since the force calculation is only part of work that needs to be done during an integration step. The load imbalance is reported in the log file at log output steps and when the `-v` option is used also on screen. The average load imbalance and the total performance loss due to load imbalance are reported at the end of the log file.

There is one important parameter for the dynamic load balancing, which is the minimum allowed scaling. By default, each dimension of the domain decomposition cell can scale down by at least a factor of 0.8. For 3-D domain decomposition this allows cells to change their volume by about a factor of 0.5, which should allow for compensation of a load imbalance of 100%. The minimum allowed scaling can be changed with the `-dds` option of *mdrun* (page 258).

The load imbalance is measured by timing a single region of the MD step on each MPI rank. This region can not include MPI communication, as timing of MPI calls does not allow separating wait due to imbal-

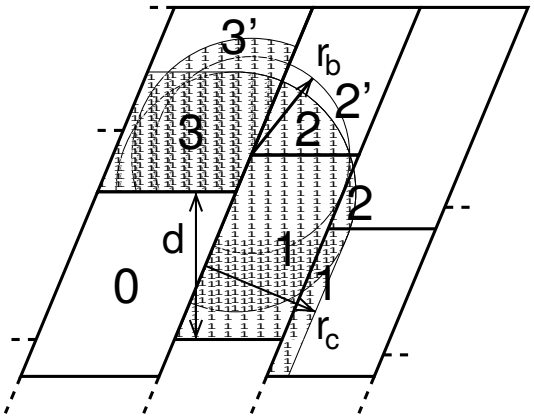



Fig. 5.12: The zones to communicate to the rank of zone 0, see the text for details. $r_c$ and $r_b$ are the non-bonded and bonded cut-off radii respectively, $d$ is an example of a distance between following, staggered boundaries of cells.

ance from actual communication. The domain volumes are then scaled, with under-relaxation, inversely proportional with the measured time. This procedure will decrease the load imbalance when the change in load in the measured region correlates with the change in domain volume and the load outside the measured region does not depend strongly on the domain volume. In CPU-only simulations, the load is measured between the coordinate and the force communication. In simulations with non-bonded work on GPUs, we overlap communication and work on the CPU with calculation on the GPU. Therefore we measure from the last communication before the force calculation to when the CPU or GPU is finished, whichever is last. When not using PME ranks, we subtract the time in PME from the CPU time, as this includes MPI calls and the PME load is independent of domain size. This generally works well, unless the non-bonded load is low and there is imbalance in the bonded interactions. Then two issues can arise. Dynamic load balancing can increase the imbalance in update and constraints and with PME the coordinate and force redistribution time can go up significantly. Although dynamic load balancing can significantly improve performance in cases where there is imbalance in the bonded interactions on the CPU, there are many situations in which some domains continue decreasing in size and the load imbalance increases and/or PME coordinate and force redistribution cost increases significantly. As of version 2016.1, *mdrun* (page 258) disables the dynamic load balancing when measurement indicates that it deteriorates performance. This means that in most cases the user will get good performance with the default, automated dynamic load balancing setting.

### Constraints in parallel

Since with domain decomposition parts of molecules can reside on different ranks, bond constraints can cross cell boundaries. This will not happen in GROMACS when update groups are used, which happens when only bonds involving hydrogens are constrained. Then atoms connected by constraints are assigned to the same domain. But without update groups a parallel constraint algorithm is required. GROMACS uses the P-LINCS algorithm *50* (page 685), which is the parallel version of the LINCS algorithm *49* (page 685) (see *The LINCS algorithm* (page 462)). The P-LINCS procedure is illustrated in Fig. 5.13. When molecules cross the cell boundaries, atoms in such molecules up to (`lincs_order + 1`) bonds away are communicated over the cell boundaries. Then, the normal LINCS algorithm can be applied to the local bonds plus the communicated ones. After this procedure, the local bonds are correctly constrained, even though the extra communicated ones are not. One coordinate communication step is required for the initial LINCS step and one for each iteration. Forces do not need to be communicated.

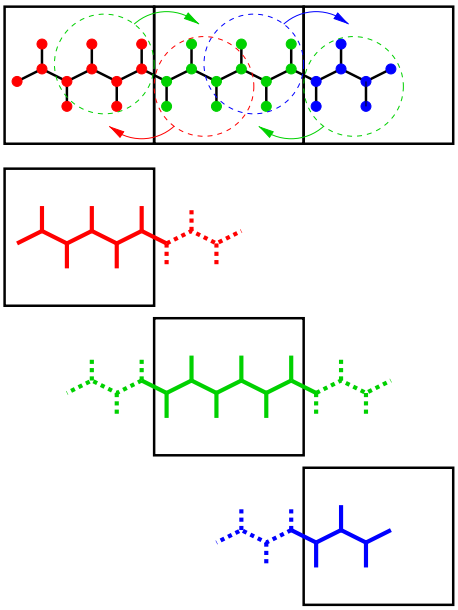


Fig. 5.13: Example of the parallel setup of P-LINCS with one molecule split over three domain decomposition cells, using a matrix expansion order of 3. The top part shows which atom coordinates need to be communicated to which cells. The bottom parts show the local constraints (solid) and the non-local constraints (dashed) for each of the three cells.

### Interaction ranges

Domain decomposition takes advantage of the locality of interactions. This means that there will be limitations on the range of interactions. By default, *mdrun* (page 258) tries to find the optimal balance between interaction range and efficiency. But it can happen that a simulation stops with an error message about missing interactions, or that a simulation might run slightly faster with shorter interaction ranges. A list of interaction ranges and their default values is given in Table 5.7

Table 5.7: The interaction ranges with domain decomposition.

| interaction | range | option | default |
|---|---|---|---|
| non-bonded | $r_c$=max($r_{\text{list}}$,$r_{\text{VdW}}$,$r_{\text{Coul}}$) | *mdp* (page 579) file | |
| two-body bonded | max($r_{\text{mb}}$,$r_c$) | *mdrun* (page 258) `-rdd` | starting conf. + 10% |
| multi-body bonded | $r_{\text{mb}}$ | *mdrun* (page 258) `-rdd` | starting conf. + 10% |
| constraints | $r_{\text{con}}$ | *mdrun* (page 258) `-rcon` | est. from bond lengths |
| virtual sites | $r_{\text{con}}$ | *mdrun* (page 258) `-rcon` | 0 |

In most cases the defaults of *mdrun* (page 258) should not cause the simulation to stop with an error message of missing interactions. The range for the bonded interactions is determined from the distance between bonded update groups in the starting configuration, with 10% added for headroom. For the constraints, the value of $r_{\text{con}}$ is determined by taking the maximum distance that (`lincs_order` + 1) bonds can cover when they all connect at angles of 120 degrees. The actual constraint communication is not limited by $r_{\text{con}}$, but by the minimum cell size $L_C$, which has the following lower limit:

$$L_C \geq \max(r_{\text{mb}}, r_{\text{con}}) \tag{5.118}$$

Without dynamic load balancing the system is actually allowed to scale beyond this limit when pressure scaling is used. **Note** that for triclinic boxes, $L_C$ is not simply the box diagonal component divided by

the number of cells in that direction, rather it is the shortest distance between the triclinic cells borders. For rhombic dodecahedra this is a factor of $\sqrt{3/2}$ shorter along $x$ and $y$.

When $r_{\mathrm{mb}} > r_c$, *mdrun* (page 258) employs a smart algorithm to reduce the communication. Simply communicating all update groups within $r_{\mathrm{mb}}$ would increase the amount of communication enormously. Therefore only update groups that are connected by bonded interactions to update groups which are not locally present are communicated. This leads to little extra communication, but also to a slightly increased cost for the domain decomposition setup. In some cases, *e.g.* coarse-grained simulations with a very short cut-off, one might want to set $r_{\mathrm{mb}}$ by hand to reduce this cost.

### Multiple-Program, Multiple-Data PME parallelization

Electrostatics interactions are long-range, therefore special algorithms are used to avoid summation over many atom pairs. In GROMACS this is usually PME (sec. *PME* (page 526)). Since with PME all particles interact with each other, global communication is required. This will usually be the limiting factor for scaling with domain decomposition. To reduce the effect of this problem, we have come up with a Multiple-Program, Multiple-Data approach *5* (page 683). Here, some ranks are selected to do only the PME mesh calculation, while the other ranks, called particle-particle (PP) ranks, do all the rest of the work. For rectangular boxes the optimal PP to PME rank ratio is usually 3:1, for rhombic dodecahedra usually 2:1. When the number of PME ranks is reduced by a factor of 4, the number of communication calls is reduced by about a factor of 16. Or put differently, we can now scale to 4 times more ranks. In addition, for modern 4 or 8 core machines in a network, the effective network bandwidth for PME is quadrupled, since only a quarter of the cores will be using the network connection on each machine during the PME calculations.

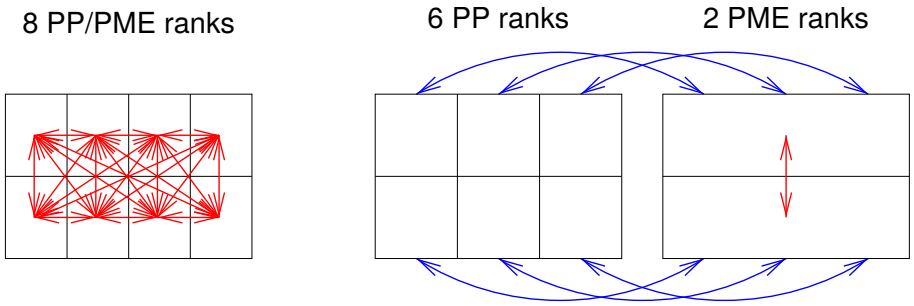



Fig. 5.14: Example of 8 ranks without (left) and with (right) MPMD. The PME communication (red arrows) is much higher on the left than on the right. For MPMD additional PP - PME coordinate and force communication (blue arrows) is required, but the total communication complexity is lower.

*mdrun* (page 258) will by default interleave the PP and PME ranks. If the ranks are not number consecutively inside the machines, one might want to use *mdrun* (page 258) `-ddorder pp_pme`. For machines with a real 3-D torus and proper communication software that assigns the ranks accordingly one should use *mdrun* (page 258) `-ddorder cartesian`.

To optimize the performance one should usually set up the cut-offs and the PME grid such that the PME load is 25 to 33% of the total calculation load. *grompp* (page 230) will print an estimate for this load at the end and also *mdrun* (page 258) calculates the same estimate to determine the optimal number of PME ranks to use. For high parallelization it might be worthwhile to optimize the PME load with the *mdp* (page 579) settings and/or the number of PME ranks with the `-npme` option of *mdrun* (page 258). For changing the electrostatics settings it is useful to know the accuracy of the electrostatics remains nearly constant when the Coulomb cut-off and the PME grid spacing are scaled by the same factor. **Note** that it is usually better to overestimate than to underestimate the number of PME ranks, since the number of PME ranks is smaller than the number of PP ranks, which leads to less total waiting time.

The PME domain decomposition can be 1-D or 2-D along the $x$ and/or $y$ axis. 2-D decomposition is also known as pencil decomposition because of the shape of the domains at high parallelization. 1-D decomposition along the $y$ axis can only be used when the PP decomposition has only 1 domain along $x$. 2-D PME decomposition has to have the number of domains along $x$ equal to the number of the PP decomposition. *mdrun* (page 258) automatically chooses 1-D or 2-D PME decomposition (when possible with the total given number of ranks), based on the minimum amount of communication for the coordinate redistribution in PME plus the communication for the grid overlap and transposes. To avoid superfluous communication of coordinates and forces between the PP and PME ranks, the number of DD cells in the $x$ direction should ideally be the same or a multiple of the number of PME ranks. By default, *mdrun* (page 258) takes care of this issue.

### Domain decomposition flow chart

In Fig. 5.15 a flow chart is shown for domain decomposition with all possible communication for different algorithms. For simpler simulations, the same flow chart applies, without the algorithms and communication for the algorithms that are not used.

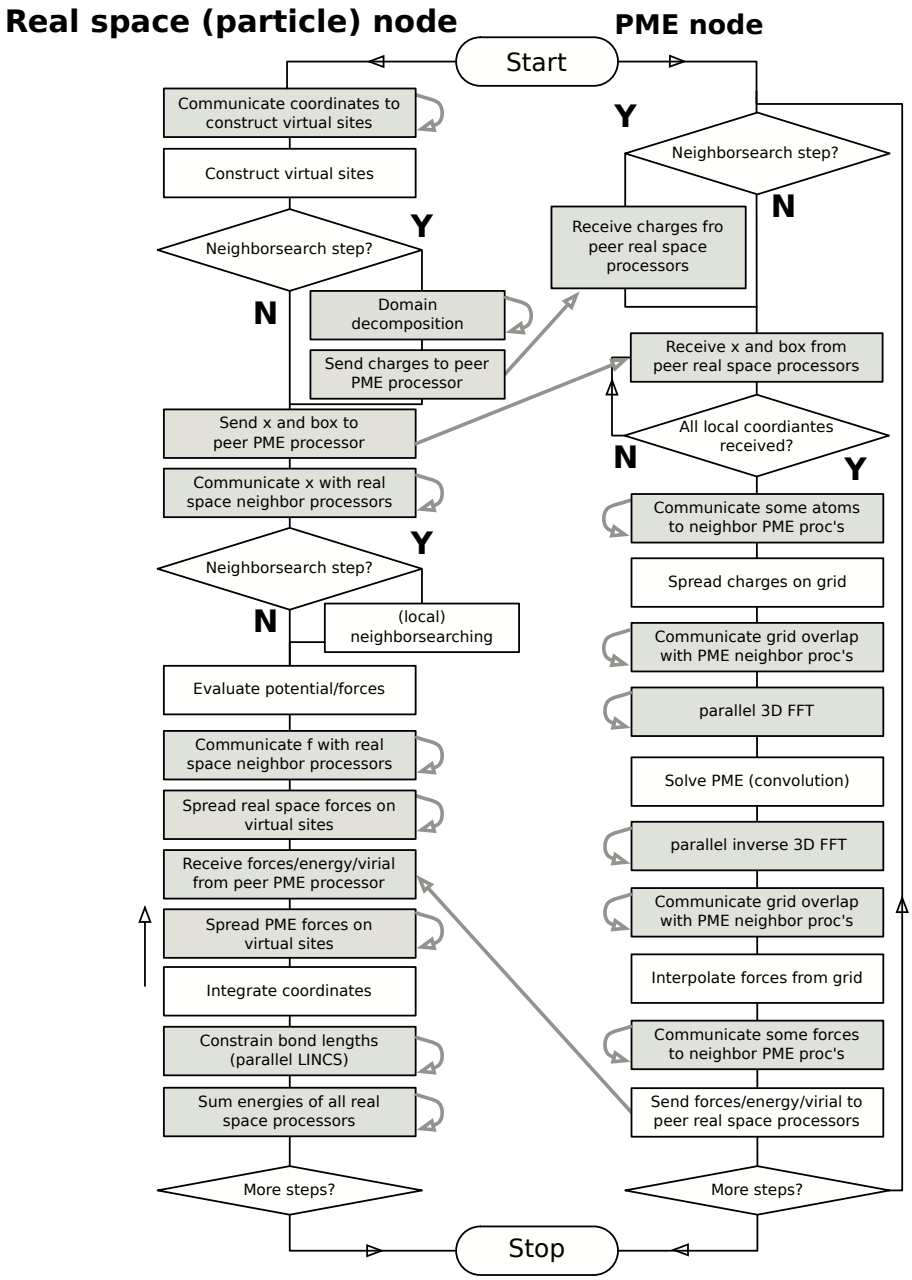



Fig. 5.15: Flow chart showing the algorithms and communication (arrows) for a standard MD simulation with virtual sites, constraints and separate PME-mesh ranks.

## 5.5 Interaction function and force fields

To accommodate the potential functions used in some popular force fields (see *Interaction function and force fields* (page 481)), GROMACS offers a choice of functions, both for non-bonded interaction and for dihedral interactions. They are described in the appropriate subsections.

The potential functions can be subdivided into three parts

1. *Non-bonded*: Lennard-Jones or Buckingham, and Coulomb or modified Coulomb. The non-bonded interactions are computed on the basis of a neighbor list (a list of non-bonded atoms within a certain radius), in which exclusions are already removed.
2. *Bonded*: covalent bond-stretching, angle-bending, improper dihedrals, and proper dihedrals. These are computed on the basis of fixed lists.
3. *Restraints*: position restraints, angle restraints, distance restraints, orientation restraints and dihedral restraints, all based on fixed lists.
4. *Applied Forces*: externally applied forces, see chapter *Special Topics* (page 590).

### 5.5.1 Non-bonded interactions

Non-bonded interactions in GROMACS are pair-additive:

$$V(\mathbf{r}_1, \ldots \mathbf{r}_N) = \sum_{i<j} V_{ij}(\mathbf{r}_{ij}); \tag{5.119}$$

$$\mathbf{F}_i = -\sum_j \frac{dV_{ij}(r_{ij})}{dr_{ij}} \frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.120}$$

Since the potential only depends on the scalar distance, interactions will be centro-symmetric, i.e. the vectorial partial force on particle $i$ from the pairwise interaction $V_{ij}(r_{ij})$ has the opposite direction of the partial force on particle $j$. For efficiency reasons, interactions are calculated by loops over interactions and updating both partial forces rather than summing one complete nonbonded force at a time. The non-bonded interactions contain a repulsion term, a dispersion term, and a Coulomb term. The repulsion and dispersion term are combined in either the Lennard-Jones (or 6-12 interaction), or the Buckingham (or exp-6 potential). In addition, (partially) charged atoms act through the Coulomb term.

#### The Lennard-Jones interaction

The Lennard-Jones potential $V_{LJ}$ between two atoms equals:

$$V_{LJ}(r_{ij}) = \frac{C_{ij}^{(12)}}{{r_{ij}}^{12}} - \frac{C_{ij}^{(6)}}{{r_{ij}}^{6}} \tag{5.121}$$

See also Fig. 5.16 The parameters $C_{ij}^{(12)}$ and $C_{ij}^{(6)}$ depend on pairs of *atom types*; consequently they are taken from a matrix of LJ-parameters. In the Verlet cut-off scheme, the potential is shifted by a constant such that it is zero at the cut-off distance.

The force derived from this potential is:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = -\left(12\,\frac{C_{ij}^{(12)}}{{r_{ij}}^{13}} - 6\,\frac{C_{ij}^{(6)}}{{r_{ij}}^{7}}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.122}$$

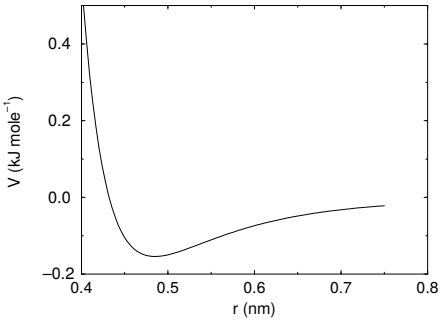



Fig. 5.16: The Lennard-Jones interaction.

The LJ potential may also be written in the following form:

$$V_{LJ}(\mathbf{r}_{ij}) = 4\epsilon_{ij}\left(\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right) \tag{5.123}$$

In constructing the parameter matrix for the non-bonded LJ-parameters, two types of combination rules can be used within GROMACS, only geometric averages (type 1 in the input section of the force-field file):

$$\begin{aligned} C_{ij}^{(6)} &= \left(C_{ii}^{(6)}\, C_{jj}^{(6)}\right)^{1/2} \\ C_{ij}^{(12)} &= \left(C_{ii}^{(12)}\, C_{jj}^{(12)}\right)^{1/2} \end{aligned} \tag{5.124}$$

or, alternatively the Lorentz-Berthelot rules can be used. An arithmetic average is used to calculate $\sigma_{ij}$, while a geometric average is used to calculate $\epsilon_{ij}$ (type 2):

$$\begin{aligned} \sigma_{ij} &= \tfrac{1}{2}(\sigma_{ii} + \sigma_{jj}) \\ \epsilon_{ij} &= (\epsilon_{ii}\,\epsilon_{jj})^{1/2} \end{aligned} \tag{5.125}$$

finally an geometric average for both parameters can be used (type 3):

$$\begin{aligned} \sigma_{ij} &= (\sigma_{ii}\,\sigma_{jj})^{1/2} \\ \epsilon_{ij} &= (\epsilon_{ii}\,\epsilon_{jj})^{1/2} \end{aligned} \tag{5.126}$$

This last rule is used by the OPLS force field.

### Buckingham potential

The Buckingham potential has a more flexible and realistic repulsion term than the Lennard-Jones interaction, but is also more expensive to compute. Note that the Buckingham potential is no longer supported by *gmx mdrun* (page 258). The potential form is:

$$V_{bh}(r_{ij}) = A_{ij}\exp(-B_{ij}r_{ij}) - \frac{C_{ij}}{{r_{ij}}^6} \tag{5.127}$$

See also Fig. 5.17. The force derived from this is:

$$\mathbf{F}_i(r_{ij}) = \left[A_{ij}B_{ij}\exp(-B_{ij}r_{ij}) - 6\frac{C_{ij}}{{r_{ij}}^7}\right]\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.128}$$

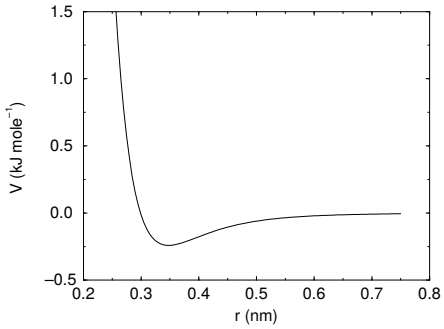



Fig. 5.17: The Buckingham interaction.

### Coulomb interaction

The Coulomb interaction between two charge particles is given by:

$$V_c(r_{ij}) = f\frac{q_i q_j}{\varepsilon_r r_{ij}} \tag{5.129}$$

See also Fig. 5.18, where $f = \frac{1}{4\pi\varepsilon_0} = 138.935\,458$ (see chapter *Definitions and Units* (page 425))

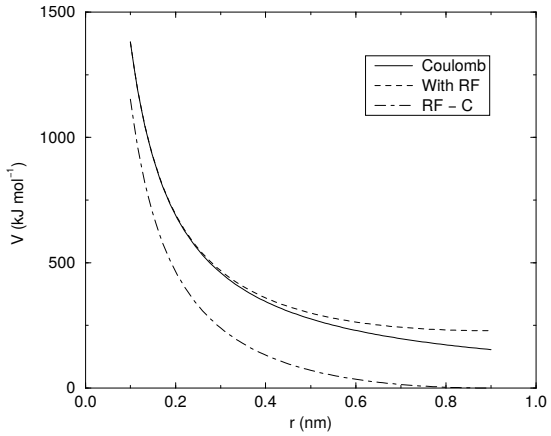



Fig. 5.18: The Coulomb interaction (for particles with equal signed charge) with and without reaction field. In the latter case $\varepsilon_r$ was 1, $\varepsilon_{rf}$ was 78, and $r_c$ was 0.9 nm. The dot-dashed line is the same as the dashed line, except for a constant.

The force derived from this potential is:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = -f\frac{q_i q_j}{\varepsilon_r {r_{ij}}^2}\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.130}$$

A plain Coulomb interaction should only be used without cut-off or when all pairs fall within the cut-off, since there is an abrupt, large change in the force at the cut-off. In case you do want to use a cut-off, the potential can be shifted by a constant to make the potential the integral of the force. With the group cut-off scheme, this shift is only applied to non-excluded pairs. With the Verlet cut-off scheme, the shift is also applied to excluded pairs and self interactions, which makes the potential equivalent to a reaction field with $\varepsilon_{rf} = 1$ (see below).

In GROMACS the relative dielectric constant $\varepsilon_r$ may be set in the in the input for *grompp* (page 230).

### Coulomb interaction with reaction field

The Coulomb interaction can be modified for homogeneous systems by assuming a constant dielectric environment beyond the cut-off $r_c$ with a dielectric constant of $\varepsilon_{rf}$. The interaction then reads:

$$V_{crf} = f\frac{q_i q_j}{\varepsilon_r r_{ij}}\left[1 + \frac{\varepsilon_{rf}-\varepsilon_r}{2\varepsilon_{rf}+\varepsilon_r}\frac{{r_{ij}}^3}{r_c^3}\right] - f\frac{q_i q_j}{\varepsilon_r r_c}\frac{3\varepsilon_{rf}}{2\varepsilon_{rf}+\varepsilon_r} \tag{5.131}$$

in which the constant expression on the right makes the potential zero at the cut-off $r_c$. For charged cut-off spheres this corresponds to neutralization with a homogeneous background charge. We can rewrite (5.131) for simplicity as

$$V_{crf} = f\frac{q_i q_j}{\varepsilon_r}\left[\frac{1}{r_{ij}} + k_{rf}\,{r_{ij}}^2 - c_{rf}\right] \tag{5.132}$$

with

$$k_{rf} = \frac{1}{r_c^3}\frac{\varepsilon_{rf}-\varepsilon_r}{(2\varepsilon_{rf}+\varepsilon_r)} \tag{5.133}$$

$$c_{rf} = \frac{1}{r_c} + k_{rf}\,r_c^2 = \frac{1}{r_c}\frac{3\varepsilon_{rf}}{(2\varepsilon_{rf}+\varepsilon_r)} \tag{5.134}$$

For large $\varepsilon_{rf}$ the $k_{rf}$ goes to $r_c^{-3}/2$, while for $\varepsilon_{rf} = \varepsilon_r$ the correction vanishes. In Fig. 5.18 the modified interaction is plotted, and it is clear that the derivative with respect to $r_{ij}$ (= -force) goes to zero at the cut-off distance. The force derived from this potential reads:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = -f\frac{q_i q_j}{\varepsilon_r}\left[\frac{1}{{r_{ij}}^2} - 2k_{rf}r_{ij}\right]\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.135}$$

The reaction-field correction should also be applied to all excluded atoms pairs, including self interactions, in which case the normal Coulomb term in (5.131) and (5.135) is absent. For the self interactions the constant is halved, leading to this constant potential term:

$$V_{self} = -f\frac{q_i^2}{2\varepsilon_r r_c}\frac{3\varepsilon_{rf}}{2\varepsilon_{rf}+\varepsilon_r}$$

### Modified non-bonded interactions

All physical forces are conservative, meaning that it is possible to assign a numerical value for the potential at any point (which thus does not depend on the path taken), and the force is the negative gradient of this potential. Based on the definitions of the potentials above, this derivative (i.e., the force) is always zero at infinite separation, and in the context of pair potentials this means the potential for each pair contribution must be the integral of the force out from infinity back to the current interaction distance. While it is perfectly valid to have an arbitrary constant factor in the potential, a natural choice is to define the pair interaction to be zero at infinite separation when particles are not really interacting. However, when these definitions using infinite-range potentials are combined with a cutoff for pair interactions we violate their consistency, and the force would no longer be conservative - which in particular means the total energy will no longer be conserved. One way to circumvent this is to instead modify the non-bonded interaction potentials such that they only have finite range, after which the cutoff can be applied. This can either be done as a switching function that changes the shape of the potential and force over a small range, or by shifting the entire potential by a constant factor such that it becomes

zero at the cutoff. The advantage of the shifted interaction modification is that it does not influence the force at all, and since only forces enter the equations of motion it will not influence the dynamics of the system. The drawback is that the total change in the potential is larger. Presently GROMACS only supports this shifted modification, and it is even applied by default (but possible to turn off). Note that we also shift the direct-space component of the PME interaction; the potential difference will be negligible since it has already decayed to the specified PME tolerance at the cutoff, but this improves energy conservation.

When used with reaction-field electrostatics ((5.131)), the self-energy term will effectively make the electrostatic potential constant (but non-zero) outside the cutoff.

For implementation reasons, GROMACS presently uses the reaction-field kernel for normal Coulomb interactions too (with $\varepsilon_{rf} = \varepsilon_r$). Note that this will give the appearance of a similar constant potential outside the cutoff for plain Coulomb electrostatics too. We will try to fix this in a future kernel, but since there are very few (if any) cases where plain Coulomb is a good choice for electrostatics it has not been a high priority.

Although the present kernels only support shifting the potential, we do plan to bring back complete functionality for switch functions, so for completeness in the interface we have retained that documentation below.

While the shift modifier will yield conservative forces, the forces will still have an abrupt change at the cutoff, which among other things can make it difficult to efficiently minimize the energy of a system prior to normal mode calculation. The force-switch function replaces the truncated forces by forces that are continuous and have continuous derivatives at the cut-off radius. With such forces the time integration produces smaller errors, although for Lennard-Jones interactions other errors tend to dominate, such as integration errors at the repulsive part of the potential. For Coulomb interactions we advise against using switch modifiers since it can lead to large peaks in the force close to the cutoff; we strongly recommend considering reaction-field or a proper long-range method such as PME instead.

We apply the switch function to the force $F(r)$ describing either the electrostatic or van der Waals force acting on particle $i$ by particle $j$ as:

$$\mathbf{F}_i = c\, F(r_{ij}) \frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.136}$$

For pure Coulomb or Lennard-Jones interactions $F(r) = F_\alpha(r) = \alpha\, r^{-(\alpha+1)}$. The switched force $F_s(r)$ can generally be written as:

$$\begin{array}{rcll} F_s(r) & = & F_\alpha(r) & r < r_1 \\ F_s(r) & = & F_\alpha(r) + S(r) & r_1 \le r < r_c \\ F_s(r) & = & 0 & r_c \le r \end{array} \tag{5.137}$$

When $r_1 = 0$ this is a traditional shift function, otherwise it acts as a switch function. The corresponding shifted potential function then reads:

$$V_s(r) = \int_r^\infty F_s(x)\, dx \tag{5.138}$$

The GROMACS **force switch** function $S_F(r)$ should be smooth at the boundaries, therefore the following boundary conditions are imposed on the switch function:

$$\begin{array}{rcl} S_F(r_1) & = & 0 \\ S'_F(r_1) & = & 0 \\ S_F(r_c) & = & -F_\alpha(r_c) \\ S'_F(r_c) & = & -F'_\alpha(r_c) \end{array} \tag{5.139}$$

A $3^{rd}$ degree polynomial of the form

$$S_F(r) = A(r - r_1)^2 + B(r - r_1)^3 \tag{5.140}$$

fulfills these requirements. The constants A and B are given by the boundary condition at $r_c$:

$$\begin{aligned} A &= -\alpha \frac{(\alpha+4)r_c - (\alpha+1)r_1}{r_c^{\alpha+2}(r_c - r_1)^2} \\ B &= \alpha \frac{(\alpha+3)r_c - (\alpha+1)r_1}{r_c^{\alpha+2}(r_c - r_1)^3} \end{aligned} \tag{5.141}$$

Thus the total force function is:

$$F_s(r) = \frac{\alpha}{r^{\alpha+1}} + A(r - r_1)^2 + B(r - r_1)^3 \tag{5.142}$$

and the potential function reads:

$$V_s(r) = \frac{1}{r^\alpha} - \frac{A}{3}(r - r_1)^3 - \frac{B}{4}(r - r_1)^4 - C \tag{5.143}$$

where

$$C = \frac{1}{r_c^\alpha} - \frac{A}{3}(r_c - r_1)^3 - \frac{B}{4}(r_c - r_1)^4 \tag{5.144}$$

The GROMACS **potential-switch** function $S_V(r)$ scales the potential between $r_1$ and $r_c$, and has similar boundary conditions, intended to produce smoothly-varying potential and forces:

$$\begin{aligned} S_V(r_1) &= 1 \\ S'_V(r_1) &= 0 \\ S''_V(r_1) &= 0 \\ S_V(r_c) &= 0 \\ S'_V(r_c) &= 0 \\ S''_V(r_c) &= 0 \end{aligned} \tag{5.145}$$

The fifth-degree polynomial that has these properties is

$$S_V(r; r_1, r_c) = 1 - 10\left(\frac{r - r_1}{r_c - r_1}\right)^3 + 15\left(\frac{r - r_1}{r_c - r_1}\right)^4 - 6\left(\frac{r - r_1}{r_c - r_1}\right)^5 \tag{5.146}$$

This implementation is found in several other simulation packages,[73] (page 686)[75] (page 686) but differs from that in CHARMM.[76] (page 686) Switching the potential leads to artificially large forces in the switching region, therefore it is not recommended to switch Coulomb interactions using this function,[72] (page 686) but switching Lennard-Jones interactions using this function produces acceptable results.

### Modified short-range interactions with Ewald summation

When Ewald summation or particle-mesh Ewald is used to calculate the long-range interactions, the short-range Coulomb potential must also be modified. Here the potential is switched to (nearly) zero at the cut-off, instead of the force. In this case the short range potential is given by:

$$V(r) = f\frac{\text{erfc}(\beta r_{ij})}{r_{ij}} q_i q_j, \tag{5.147}$$

where $\beta$ is a parameter that determines the relative weight between the direct space sum and the reciprocal space sum and $\text{erfc}(x)$ is the complementary error function. Note that GROMACS by default shifts this potential by a constant to ensure that the potential is zero at the cut-off. For further details on long-range electrostatics, see sec. *Long Range Electrostatics* (page 525).

### 5.5.2 Bonded interactions

Bonded interactions are based on a fixed list of atoms. They are not exclusively pair interactions, but include 3- and 4-body interactions as well. There are *bond stretching* (2-body), *bond angle* (3-body), and *dihedral angle* (4-body) interactions. A special type of dihedral interaction (called *improper dihedral*) is used to force atoms to remain in a plane or to prevent transition to a configuration of opposite chirality (a mirror image).

#### Bond stretching

##### Harmonic potential

The bond stretching between two covalently bonded atoms $i$ and $j$ is represented by a harmonic potential (see also Fig. 5.19):

$$V_b\left(r_{ij}\right) = \frac{1}{2}k^b_{ij}(r_{ij} - b_{ij})^2 \tag{5.148}$$

With the corresponding force given by:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = k^b_{ij}(r_{ij} - b_{ij})\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.149}$$

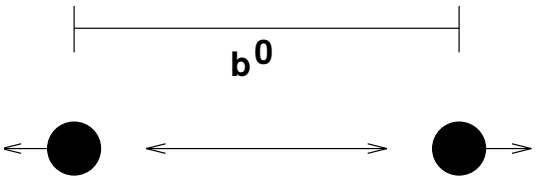



Fig. 5.19: Principle of bond stretching.

##### Fourth power potential

In the GROMOS-96 force field *77* (page 686), the covalent bond potential is, for reasons of computational efficiency, written as:

$$V_b\left(r_{ij}\right) = \frac{1}{4}k^b_{ij}\left({r_{ij}}^2 - b^2_{ij}\right)^2 \tag{5.150}$$

The corresponding force is:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = k^b_{ij}({r_{ij}}^2 - b^2_{ij})\,\mathbf{r}_{ij} \tag{5.151}$$

The force constants for this form of the potential are related to the usual harmonic force constant $k^{b,\mathrm{harm}}$ (sec. *Bond stretching* (page 487)) as

$$2k^b b^2_{ij} = k^{b,\mathrm{harm}} \tag{5.152}$$

The force constants are mostly derived from the harmonic ones used in GROMOS-87 *78* (page 687). Although this form is computationally more efficient (because no square root has to be evaluated), it is conceptually more complex. One particular disadvantage is that since the form is not harmonic, the average energy of a single bond is not equal to $\frac{1}{2}kT$ as it is for the normal harmonic potential.

### Morse potential bond stretching

For some systems that require an anharmonic bond stretching potential, the Morse potential *79* (page 687) between two atoms *i* and *j* is available in GROMACS. This potential differs from the harmonic potential in that it has an asymmetric potential well and a zero force at infinite distance. The functional form is:

$$V_{morse}(r_{ij}) = D_{ij}[1 - \exp(-\beta_{ij}(r_{ij} - b_{ij}))]^2, \tag{5.153}$$

See also Fig. 5.20, and the corresponding force is:

$$\begin{aligned} \mathbf{F}_{morse}(\mathbf{r}_{ij}) &= 2D_{ij}\beta_{ij}\exp(-\beta_{ij}(r_{ij} - b_{ij}))* \\ &\quad [1 - \exp(-\beta_{ij}(r_{ij} - b_{ij}))]\frac{\mathbf{r}_{ij}}{r_{ij}}, \end{aligned} \tag{5.154}$$

where $D_{ij}$ is the depth of the well in kJ/mol, $\beta_{ij}$ defines the steepness of the well (in nm$^{-1}$), and $b_{ij}$ is the equilibrium distance in nm. The steepness parameter $\beta_{ij}$ can be expressed in terms of the reduced mass of the atoms *i* and *j*, the fundamental vibration frequency $\omega_{ij}$ and the well depth $D_{ij}$:

$$\beta_{ij} = \omega_{ij}\sqrt{\frac{\mu_{ij}}{2D_{ij}}} \tag{5.155}$$

and because $\omega = \sqrt{k/\mu}$, one can rewrite $\beta_{ij}$ in terms of the harmonic force constant $k_{ij}$:

$$\beta_{ij} = \sqrt{\frac{k_{ij}}{2D_{ij}}} \tag{5.156}$$

For small deviations $(r_{ij} - b_{ij})$, one can approximate the exp-term to first-order using a Taylor expansion:

$$\exp(-x) \approx 1 - x \tag{5.157}$$

and substituting (5.156) and (5.157) in the functional form:

$$\begin{aligned} V_{morse}(r_{ij}) &= D_{ij}[1 - \exp(-\beta_{ij}(r_{ij} - b_{ij}))]^2 \\ &= D_{ij}[1 - (1 - \sqrt{\tfrac{k_{ij}}{2D_{ij}}}(r_{ij} - b_{ij}))]^2 \\ &= \tfrac{1}{2}k_{ij}(r_{ij} - b_{ij}))^2 \end{aligned} \tag{5.158}$$

we recover the harmonic bond stretching potential.

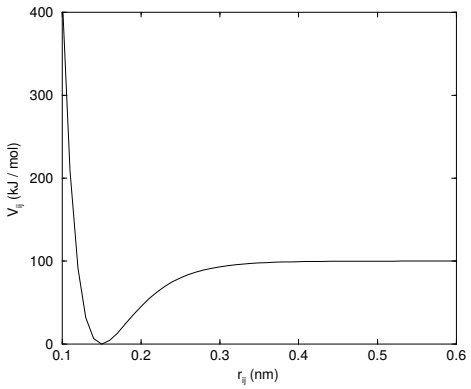



Fig. 5.20: The Morse potential well, with bond length 0.15 nm.

### Cubic bond stretching potential

Another anharmonic bond stretching potential that is slightly simpler than the Morse potential adds a cubic term in the distance to the simple harmonic form:

$$V_b\left(r_{ij}\right) = k^b_{ij}(r_{ij} - b_{ij})^2 + k^b_{ij}k^{cub}_{ij}(r_{ij} - b_{ij})^3 \tag{5.159}$$

A flexible water model (based on the SPC water model *80* (page 687)) including a cubic bond stretching potential for the O-H bond was developed by Ferguson *81* (page 687). This model was found to yield a reasonable infrared spectrum. The Ferguson water model is available in the GROMACS library (`flexwat-ferguson.itp`). It should be noted that the potential is asymmetric: overstretching leads to infinitely low energies. The integration timestep is therefore limited to 1 fs.

The force corresponding to this potential is:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = 2k^b_{ij}(r_{ij} - b_{ij})\,\frac{\mathbf{r}_{ij}}{r_{ij}} + 3k^b_{ij}k^{cub}_{ij}(r_{ij} - b_{ij})^2\,\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.160}$$

### FENE bond stretching potential

In coarse-grained polymer simulations the beads are often connected by a FENE (finitely extensible nonlinear elastic) potential *82* (page 687):

$$V_{\mathrm{FENE}}(r_{ij}) = -\frac{1}{2}k^b_{ij}b^2_{ij}\log\left(1 - \frac{{r_{ij}}^2}{b^2_{ij}}\right) \tag{5.161}$$

The potential looks complicated, but the expression for the force is simpler:

$$F_{\mathrm{FENE}}(\mathbf{r}_{ij}) = -k^b_{ij}\left(1 - \frac{{r_{ij}}^2}{b^2_{ij}}\right)^{-1}\mathbf{r}_{ij} \tag{5.162}$$

At short distances the potential asymptotically goes to a harmonic potential with force constant $k^b$, while it diverges at distance $b$.

### Harmonic angle potential

The bond-angle vibration between a triplet of atoms $i$ - $j$ - $k$ (Fig. 5.19) is also represented by a harmonic potential on the angle $\theta_{ijk}$

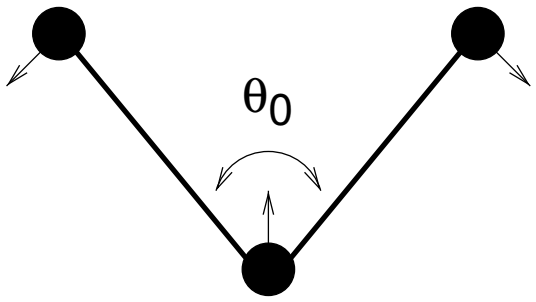



Fig. 5.21: Principle of angle vibration.

$$V_a(\theta_{ijk}) = \frac{1}{2}k^\theta_{ijk}(\theta_{ijk} - {\theta_{ijk}}^0)^2 \tag{5.163}$$

As the bond-angle vibration is represented by a harmonic potential, the form is the same as the bond stretching.

The force equations are given by the chain rule:

$$\begin{aligned} \mathbf{F}_i &= -\frac{dV_a(\theta_{ijk})}{d\mathbf{r}_i} \\ \mathbf{F}_k &= -\frac{dV_a(\theta_{ijk})}{d\mathbf{r}_k} \\ \mathbf{F}_j &= -\mathbf{F}_i - \mathbf{F}_k \end{aligned} \quad \text{where} \quad \theta_{ijk} = \arccos \frac{(\mathbf{r}_{ij} \cdot \mathbf{r}_{kj})}{r_{ij} r_{kj}} \tag{5.164}$$

The numbering $i, j, k$ is in sequence of covalently bonded atoms. Atom $j$ is in the middle; atoms $i$ and $k$ are at the ends (see Fig. 5.21). **Note** that in the input in topology files, angles are given in degrees and force constants in kJ/mol/rad$^2$.

### Cosine based angle potential

In the GROMOS-96 force field a simplified function is used to represent angle vibrations:

$$V_a(\theta_{ijk}) = \frac{1}{2} k^{\theta}_{ijk} \left( \cos(\theta_{ijk}) - \cos({\theta_{ijk}}^0) \right)^2 \tag{5.165}$$

where

$$\cos(\theta_{ijk}) = \frac{\mathbf{r}_{ij} \cdot \mathbf{r}_{kj}}{r_{ij} r_{kj}} \tag{5.166}$$

The corresponding force can be derived by partial differentiation with respect to the atomic positions. The force constants in this function are related to the force constants in the harmonic form $k^{\theta,\mathrm{harm}}$ (*Harmonic angle potential* (page 489)) by:

$$k^{\theta} \sin^2({\theta_{ijk}}^0) = k^{\theta,\mathrm{harm}} \tag{5.167}$$

In the GROMOS-96 manual there is a much more complicated conversion formula which is temperature dependent. The formulas are equivalent at 0 K and the differences at 300 K are on the order of 0.1 to 0.2%. **Note** that in the input in topology files, angles are given in degrees and force constants in kJ/mol.

### Restricted bending potential

The restricted bending (ReB) potential *83* (page 687) prevents the bending angle $\theta$ from reaching the $180^\circ$ value. In this way, the numerical instabilities due to the calculation of the torsion angle and potential are eliminated when performing coarse-grained molecular dynamics simulations.

To systematically hinder the bending angles from reaching the $180^\circ$ value, the bending potential (5.165) is divided by a $\sin^2\theta$ factor:

$$V_{\mathrm{ReB}}(\theta_i) = \frac{1}{2} k_\theta \frac{(\cos\theta_i - \cos\theta_0)^2}{\sin^2\theta_i}. \tag{5.168}$$

Figure 5.22 shows the comparison between the ReB potential, (5.168), and the standard one (5.165).

The wall of the ReB potential is very repulsive in the region close to $180^\circ$ and, as a result, the bending angles are kept within a safe interval, far from instabilities. The power 2 of $\sin\theta_i$ in the denominator has been chosen to guarantee this behavior and allows an elegant differentiation:

$$F_{\mathrm{ReB}}(\theta_i) = -\frac{k_\theta}{\sin^4\theta_i} (\cos\theta_i - \cos\theta_0)(1 - \cos\theta_i \cos\theta_0) \frac{\partial \cos\theta_i}{\partial \vec{r}_k}. \tag{5.169}$$

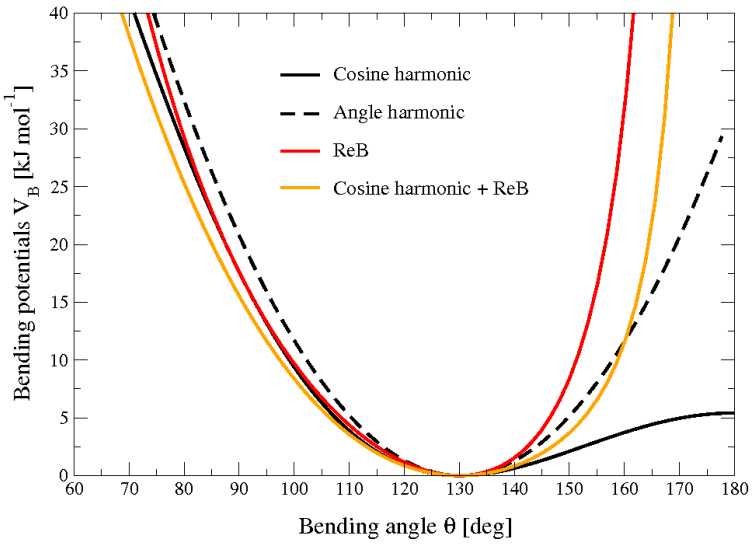



Fig. 5.22: Bending angle potentials: cosine harmonic (solid black line), angle harmonic (dashed black line) and restricted bending (red) with the same bending constant $k_\theta = 85$ kJ mol$^{-1}$ and equilibrium angle $\theta_0 = 130°$. The orange line represents the sum of a cosine harmonic ($k = 50$ kJ mol$^{-1}$) with a restricted bending ($k = 25$ kJ mol$^{-1}$) potential, both with $\theta_0 = 130°$.

Due to its construction, the restricted bending potential cannot be used for equilibrium $\theta_0$ values too close to $0°$ or $180°$ (from experience, at least $10°$ difference is recommended). It is very important that, in the starting configuration, all the bending angles are in the safe interval to avoid initial instabilities. This bending potential can be used in combination with any form of torsion potential. It will always prevent three consecutive particles from becoming collinear and, as a result, any torsion potential will remain free of singularities. It can be also added to a standard bending potential to affect the angle around $180°$, but to keep its original form around the minimum (see the orange curve in Fig. 5.22).

### Urey-Bradley potential

The Urey-Bradley bond-angle vibration between a triplet of atoms $i$ - $j$ - $k$ is represented by a harmonic potential on the angle $\theta_{ijk}$ and a harmonic correction term on the distance between the atoms $i$ and $k$. Although this can be easily written as a simple sum of two terms, it is convenient to have it as a single entry in the topology file and in the output as a separate energy term. It is used mainly in the CHARMM force field *84* (page 687). The energy is given by:

$$V_a(\theta_{ijk}) = \frac{1}{2}k^{\theta}_{ijk}(\theta_{ijk} - {\theta_{ijk}}^0)^2 + \frac{1}{2}k^{UB}_{ijk}(r_{ik} - r^0_{ik})^2 \tag{5.170}$$

The force equations can be deduced from sections *Harmonic potential* (page 487) and *Harmonic angle potential* (page 489).

### Linear Angle potential

The linear angle potential was designed especially for linear compounds such as nitriles and for carbon dioxide *190* (page 692). It avoids the calculation of the angle per se, since the angle force is not well-defined if the angle is 180 degrees. Rather, it computes the deviation of a central atom in a triplet *i,j,k* from a reference position

$$\mathbf{x}_j^0 = a\mathbf{x}_i + (1-a)\mathbf{x}_k$$

where a is defined by the bond-length *i-j* and *j-k*, in a symmetric molecule such as carbon dioxide $a$ = *1/2*. If the compound has different bond lengths $b_{ij}$ and $b_{jk}$ respectively, we instead have

$$a = \frac{b_{jk}}{b_{ij} + b_{jk}}.$$

If the order of atoms is changed to *k,j,i*, $a$ needs to be replaced by *1-a*. The energy is now given by

$$V_{lin} = \frac{k_{lin}}{2}\left(\mathbf{x}_j - \mathbf{x}_j^0\right)^2$$

with $k_{lin}$ the force constant. For examples, and a derivation of the forces from the energy function, see ref. *190* (page 692).

### Bond-Bond cross term

The bond-bond cross term for three particles $i, j, k$ forming bonds $i-j$ and $k-j$ is given by *85* (page 687):

$$V_{rr'} = k_{rr'}\left(|\mathbf{r}_i - \mathbf{r}_j| - r_{1e}\right)\left(|\mathbf{r}_k - \mathbf{r}_j| - r_{2e}\right) \tag{5.171}$$

where $k_{rr'}$ is the force constant, and $r_{1e}$ and $r_{2e}$ are the equilibrium bond lengths of the $i-j$ and $k-j$ bonds respectively. The force associated with this potential on particle $i$ is:

$$\mathbf{F}_i = -k_{rr'}\left(|\mathbf{r}_k - \mathbf{r}_j| - r_{2e}\right)\frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} \tag{5.172}$$

The force on atom $k$ can be obtained by swapping $i$ and $k$ in the above equation. Finally, the force on atom $j$ follows from the fact that the sum of internal forces should be zero: $\mathbf{F}_j = -\mathbf{F}_i - \mathbf{F}_k$.

### Bond-Angle cross term

The bond-angle cross term for three particles $i, j, k$ forming bonds $i-j$ and $k-j$ is given by *85* (page 687):

$$V_{r\theta} = k_{r\theta}\left(|\mathbf{r}_i - \mathbf{r}_k| - r_{3e}\right)\left(|\mathbf{r}_i - \mathbf{r}_j| - r_{1e} + |\mathbf{r}_k - \mathbf{r}_j| - r_{2e}\right) \tag{5.173}$$

where $k_{r\theta}$ is the force constant, $r_{3e}$ is the $i-k$ distance, and the other constants are the same as in (5.171). The force associated with the potential on atom $i$ is:

$$\mathbf{F}_i = -k_{r\theta}\left[\left(|\mathbf{r}_i - \mathbf{r}_k| - r_{3e}\right)\frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} + \left(|\mathbf{r}_i - \mathbf{r}_j| - r_{1e} + |\mathbf{r}_k - \mathbf{r}_j| - r_{2e}\right)\frac{\mathbf{r}_i - \mathbf{r}_k}{|\mathbf{r}_i - \mathbf{r}_k|}\right] \tag{5.174}$$

### Quartic angle potential

For special purposes there is an angle potential that uses a fourth order polynomial:

$$V_q(\theta_{ijk}) \;=\; \sum_{n=0}^{4} C_n(\theta_{ijk} - {\theta_{ijk}}^0)^n \tag{5.175}$$

### Improper dihedrals

Improper dihedrals are meant to keep planar groups (*e.g.* aromatic rings) planar, or to prevent molecules from flipping over to their mirror images, see Fig. 5.23.

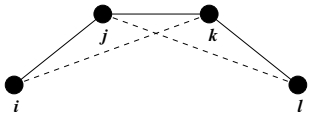



Fig. 5.23: Principle of improper dihedral angles. Out of plane bending for rings. The improper dihedral angle $\xi$ is defined as the angle between planes (i,j,k) and (j,k,l).

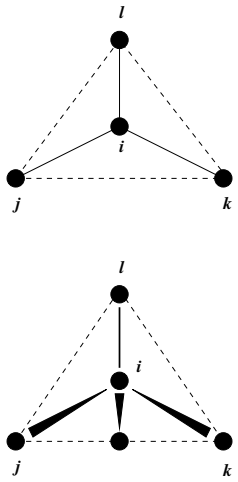



Fig. 5.24: Principle of improper dihedral angles. Out of tetrahedral angle. The improper dihedral angle $\xi$ is defined as the angle between planes (i,j,k) and (j,k,l).

### Improper dihedrals: harmonic type

The simplest improper dihedral potential is a harmonic potential; it is plotted in Fig. 5.25.

$$V_{id}(\xi_{ijkl}) = \frac{1}{2} k_\xi (\xi_{ijkl} - \xi_0)^2 \tag{5.176}$$

Since the potential is harmonic it is discontinuous, but since the discontinuity is chosen at 180° distance from $\xi_0$ this will never cause problems. **Note** that in the input in topology files, angles are given in degrees and force constants in kJ/mol/rad$^2$.

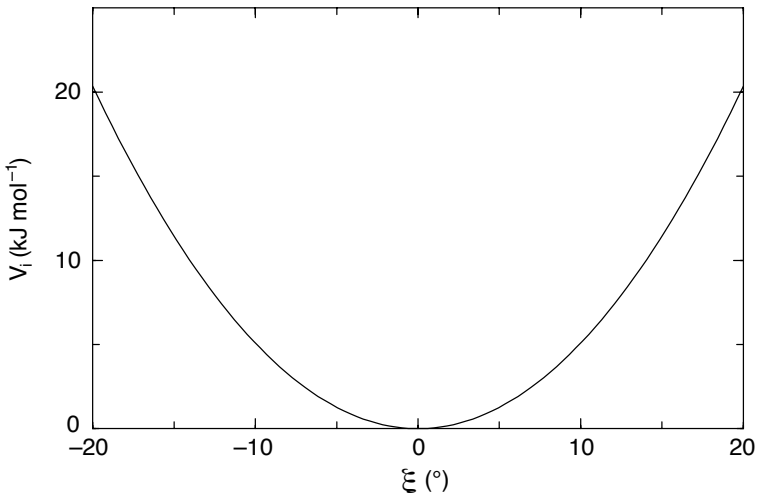



Fig. 5.25: Improper dihedral potential.

### Improper dihedrals: periodic type

This potential is identical to the periodic proper dihedral (see below). There is a separate dihedral type for this (type 4) only to be able to distinguish improper from proper dihedrals in the parameter section and the output.

### Proper dihedrals

For the normal dihedral interaction there is a choice of either the GROMOS periodic function or a function based on expansion in powers of $\cos\phi$ (the so-called Ryckaert-Bellemans potential). This choice has consequences for the inclusion of special interactions between the first and the fourth atom of the dihedral quadruple. With the periodic GROMOS potential a special 1-4 LJ-interaction must be included; with the Ryckaert-Bellemans potential *for alkanes* the 1-4 interactions must be excluded from the non-bonded list. **Note:** Ryckaert-Bellemans potentials are also used in *e.g.* the OPLS force field in combination with 1-4 interactions. You should therefore not modify topologies generated by *pdb2gmx* (page 281) in this case.

### Proper dihedrals: periodic type

Proper dihedral angles are defined according to the IUPAC/IUB convention, where $\phi$ is the angle between the $ijk$ and the $jkl$ planes, with **zero** corresponding to the *cis* configuration ($i$ and $l$ on the same side). There are two dihedral function types in GROMACS topology files. There is the standard type 1 which behaves like any other bonded interactions. For certain force fields, type 9 is useful. Type 9 allows multiple potential functions to be applied automatically to a single dihedral in the `[ dihedral ]` section when multiple parameters are defined for the same atomtypes in the `[ dihedraltypes ]` section.

$$V_d(\phi_{ijkl}) = k_\phi(1 + \cos(n\phi - \phi_s)) \tag{5.177}$$

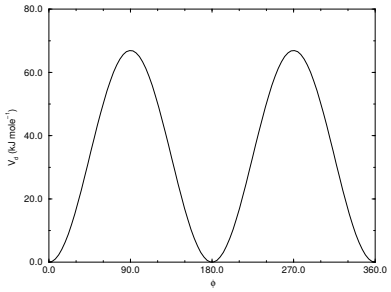



Fig. 5.26: Principle of proper dihedral angle (left, in *trans* form) and the dihedral angle potential (right).

### Proper dihedrals: Ryckaert-Bellemans function

For alkanes, the following proper dihedral potential is often used (see Fig. 5.27):

$$V_{rb}(\phi_{ijkl}) = \sum_{n=0}^{5} C_n(\cos(\psi))^n, \tag{5.178}$$

where $\psi = \phi - 180°$.

**Note:** A conversion from one convention to another can be achieved by multiplying every coefficient $C_n$ by $(-1)^n$.

**Note:** In a force field, the $C_n$ coefficients for each Ryckaert-Bellemans dihedral should sum to the same value, ideally zero, to ensure consistency in the zero point of the potential energy. Failing that, energy values and free energy determination (through $\partial H/\partial\lambda$) might have an undesirable offset, though the dynamics itself will be unaffected.

An example of constants for $C$ is given in Table 5.8.

Table 5.8: Constants for Ryckaert-Bellemans potential ($\mathrm{kJmol}^{-1}$).

| | | | | | |
|---|---|---|---|---|---|
| $C_0$ | 9.28 | $C_2$ | -13.12 | $C_4$ | 26.24 |
| $C_1$ | 12.16 | $C_3$ | -3.06 | $C_5$ | -31.5 |

(**Note:** The use of this potential implies exclusion of LJ interactions between the first and the last atom of the dihedral, and $\psi$ is defined according to the "polymer convention" ($\psi_{trans} = 0$).)

The RB dihedral function can also be used to include Fourier dihedrals (see below):

$$V_{rb}(\phi_{ijkl}) \;=\; \frac{1}{2}\left[F_1(1+\cos(\phi)) + F_2(1-\cos(2\phi)) + F_3(1+\cos(3\phi)) + F_4(1-\cos(4\phi))\right] \tag{5.179}$$

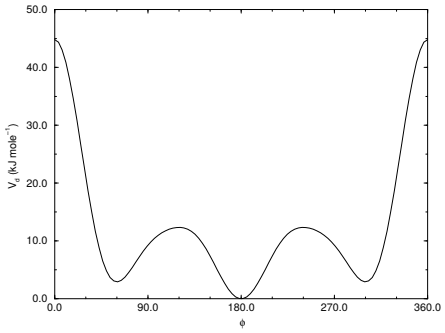



Fig. 5.27: Ryckaert-Bellemans dihedral potential.

Because of the equalities $\cos(2\phi) = 2\cos^2(\phi) - 1$, $\cos(3\phi) = 4\cos^3(\phi) - 3\cos(\phi)$ and $\cos(4\phi) = 8\cos^4(\phi) - 8\cos^2(\phi) + 1$ one can translate the OPLS parameters to Ryckaert-Bellemans parameters as follows:

$$\begin{aligned} C_0 &= F_2 + \tfrac{1}{2}(F_1 + F_3) \\ C_1 &= \tfrac{1}{2}(-F_1 + 3\,F_3) \\ C_2 &= -F_2 + 4\,F_4 \\ C_3 &= -2\,F_3 \\ C_4 &= -4\,F_4 \\ C_5 &= 0 \end{aligned} \tag{5.180}$$

with OPLS parameters in protein convention and RB parameters in polymer convention (this yields a minus sign for the odd powers of $\cos(\phi)$).

**Note:** Mind the conversion from **kcal mol**$^{-1}$ for literature OPLS and RB parameters to **kJ mol**$^{-1}$ in GROMACS.

### Proper dihedrals: Fourier function

The OPLS potential function is given as the first three *86* (page 687) or four *87* (page 687) cosine terms of a Fourier series. In GROMACS the four term function is implemented:

$$V_F(\phi_{ijkl}) \;=\; \frac{1}{2}\left[C_1(1+\cos(\phi)) + C_2(1-\cos(2\phi)) + C_3(1+\cos(3\phi)) + C_4(1-\cos(4\phi))\right], \tag{5.181}$$

Internally, GROMACS uses the Ryckaert-Bellemans code to compute Fourier dihedrals (see above), because this is more efficient.

**Note:** Mind the conversion from **kcal mol**$^{-1}$ for literature OPLS parameters to **kJ mol**$^{-1}$ in GROMACS.

### Proper dihedrals: Restricted torsion potential

In a manner very similar to the restricted bending potential (see *Restricted bending potential* (page 490)), a restricted torsion/dihedral potential is introduced:

$$V_{\mathrm{ReT}}(\phi_i) = \frac{1}{2} k_\phi \frac{(\cos\phi_i - \cos\phi_0)^2}{\sin^2\phi_i} \tag{5.182}$$

with the advantages of being a function of $\cos\phi$ (no problems taking the derivative of $\sin\phi$) and of keeping the torsion angle at only one minimum value. In this case, the factor $\sin^2\phi$ does not allow the dihedral angle to move from the $[-180°{:}0]$ to $[0{:}180°]$ interval, i.e. it cannot have maxima both at $-\phi_0$ and $+\phi_0$ maxima, but only one of them. For this reason, all the dihedral angles of the starting configuration should have their values in the desired angles interval and the equilibrium $\phi_0$ value should not be too close to the interval limits (as for the restricted bending potential, described in *Restricted bending potential* (page 490), at least $10°$ difference is recommended).

### Proper dihedrals: Combined bending-torsion potential

When the four particles forming the dihedral angle become collinear (this situation will never happen in atomistic simulations, but it can occur in coarse-grained simulations) the calculation of the torsion angle and potential leads to numerical instabilities. One way to avoid this is to use the restricted bending potential (see *Restricted bending potential* (page 490)) that prevents the dihedral from reaching the $180°$ value.

Another way is to disregard any effects of the dihedral becoming ill-defined, keeping the dihedral force and potential calculation continuous in entire angle range by coupling the torsion potential (in a cosine form) with the bending potentials of the adjacent bending angles in a unique expression:

$$V_{\mathrm{CBT}}(\theta_{i-1}, \theta_i, \phi_i) = k_\phi \sin^3\theta_{i-1} \sin^3\theta_i \sum_{n=0}^{4} a_n \cos^n\phi_i. \tag{5.183}$$

This combined bending-torsion (CBT) potential has been proposed by *88* (page 687) for polymer melt simulations and is extensively described in *83* (page 687).

This potential has two main advantages:

- it does not only depend on the dihedral angle $\phi_i$ (between the $i-2$, $i-1$, $i$ and $i+1$ beads) but also on the bending angles $\theta_{i-1}$ and $\theta_i$ defined from three adjacent beads ($i-2$, $i-1$ and $i$, and $i-1$, $i$ and $i+1$, respectively). The two $\sin^3\theta$ pre-factors, tentatively suggested by *89* (page 687) and theoretically discussed by *90* (page 687), cancel the torsion potential and force when either of the two bending angles approaches the value of $180°$.
- its dependence on $\phi_i$ is expressed through a polynomial in $\cos\phi_i$ that avoids the singularities in $\phi = 0°$ or $180°$ in calculating the torsional force.

These two properties make the CBT potential well-behaved for MD simulations with weak constraints on the bending angles or even for steered / non-equilibrium MD in which the bending and torsion angles suffer major modifications. When using the CBT potential, the bending potentials for the adjacent $\theta_{i-1}$ and $\theta_i$ may have any form. It is also possible to leave out the two angle bending terms ($\theta_{i-1}$ and $\theta_i$) completely. Fig. 5.28 illustrates the difference between a torsion potential with and without the $\sin^3\theta$ factors (blue and gray curves, respectively).

Additionally, the derivative of $V_{CBT}$ with respect to the Cartesian variables is straightforward:

$$\frac{\partial V_{\mathrm{CBT}}(\theta_{i-1}, \theta_i, \phi_i)}{\partial \vec{r}_l} = \frac{\partial V_{\mathrm{CBT}}}{\partial \theta_{i-1}} \frac{\partial \theta_{i-1}}{\partial \vec{r}_l} + \frac{\partial V_{\mathrm{CBT}}}{\partial \theta_i} \frac{\partial \theta_i}{\partial \vec{r}_l} + \frac{\partial V_{\mathrm{CBT}}}{\partial \phi_i} \frac{\partial \phi_i}{\partial \vec{r}_l} \tag{5.184}$$

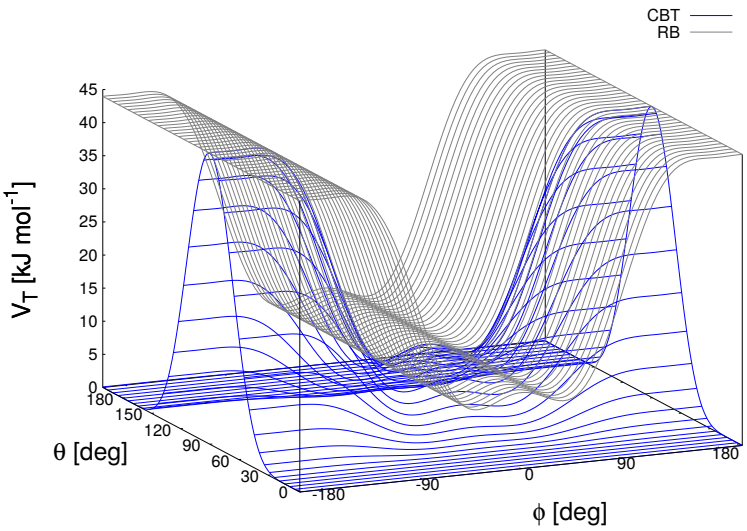



Fig. 5.28: Blue: surface plot of the combined bending-torsion potential ((5.183) with $k = 10$ kJ mol$^{-1}$, $a_0 = 2.41$, $a_1 = -2.95$, $a_2 = 0.36$, $a_3 = 1.33$) when, for simplicity, the bending angles behave the same ($\theta_1 = \theta_2 = \theta$). Gray: the same torsion potential without the $\sin^3\theta$ terms (Ryckaert-Bellemans type). $\phi$ is the dihedral angle.

The CBT is based on a cosine form without multiplicity, so it can only be symmetrical around $0°$. To obtain an asymmetrical dihedral angle distribution (e.g. only one maximum in $[-180°{:}180°]$ interval), a standard torsion potential such as harmonic angle or periodic cosine potentials should be used instead of a CBT potential. However, these two forms have the inconveniences of the force derivation ($1/\sin\phi$) and of the alignment of beads ($\theta_i$ or $\theta_{i-1} = 0°, 180°$). Coupling such non-$\cos\phi$ potentials with $\sin^3\theta$ factors does not improve simulation stability since there are cases in which $\theta$ and $\phi$ are simultaneously $180°$. The integration at this step would be possible (due to the cancelling of the torsion potential) but the next step would be singular ($\theta$ is not $180°$ and $\phi$ is very close to $180°$).

### Bonded pair and 1-4 interactions

Most force fields do not use normal Lennard-Jones and Coulomb interactions for atoms separated by three bonds, the so-called 1-4 interactions. These interactions are still affected by the modified electronic distributions due to the chemical bonds and they are modified in the force field by the dihedral terms. For this reason the Lennard-Jones and Coulomb 1-4 interactions are often scaled down, by a fixed factor given by the force field. These factors can be supplied in the topology and the parameters can also be overriden per 1-4 interaction or atom type pair. The pair interactions can be used for any atom pair in a molecule, not only 1-4 pairs. The non-bonded interactions between such pairs should be excluded to avoid double interactions. Plain Lennard-Jones and Coulomb interactions are used which are not affected by the non-bonded interaction treatment and potential modifiers.

#### Tabulated bonded interaction functions

Tabulated bonded interactions are currently (since version 2020) disabled in GROMACS. The aim is to re-enable this functionality in the future. The section below is kept for reference.

For full flexibility, any functional shape can be used for bonds, angles and dihedrals through user-supplied tabulated functions. The functional shapes are:

$$\begin{aligned} V_b(r_{ij}) &= k\, f_n^b(r_{ij}) \\ V_a(\theta_{ijk}) &= k\, f_n^a(\theta_{ijk}) \\ V_d(\phi_{ijkl}) &= k\, f_n^d(\phi_{ijkl}) \end{aligned} \tag{5.185}$$

where $k$ is a force constant in units of energy and $f$ is a cubic spline function; for details see *Cubic splines for potentials* (page 630). For each interaction, the force constant $k$ and the table number $n$ are specified in the topology. There are two different types of bonds, one that generates exclusions (type 8) and one that does not (type 9). For details see Table 5.14. The table files are supplied to the *mdrun* (page 258) program. After the table file name an underscore, the letter "b" for bonds, "a" for angles or "d" for dihedrals and the table number must be appended. For example, a tabulated bond with $n = 0$ can be read from the file table_b0.xvg. Multiple tables can be supplied simply by adding files with different values of $n$, and are applied to the appropriate bonds, as specified in the topology (Table 5.14). The format for the table files is three fixed-format columns of any suitable width. These columns must contain $x$, $f(x)$, $-f'(x)$, and the values of $x$ should be uniformly spaced. Requirements for entries in the topology are given in Table 5.14. The setup of the tables is as follows:

**bonds**: $x$ is the distance in nm. For distances beyond the table length, *mdrun* (page 258) will quit with an error message.

**angles**: $x$ is the angle in degrees. The table should go from 0 up to and including 180 degrees; the derivative is taken in degrees.

**dihedrals**: $x$ is the dihedral angle in degrees. The table should go from -180 up to and including 180 degrees; the IUPAC/IUB convention is used, *i.e.* zero is cis, the derivative is taken in degrees.

### 5.5.3 Restraints

Special potentials are used for imposing restraints on the motion of the system, either to avoid disastrous deviations, or to include knowledge from experimental data. In either case they are not really part of the force field and the reliability of the parameters is not important. The potential forms, as implemented in GROMACS, are mentioned just for the sake of completeness. Restraints and constraints refer to quite different algorithms in GROMACS.

### Position restraints

These are used to restrain particles to fixed reference positions $\mathbf{R}_i$. They can be used during equilibration in order to avoid drastic rearrangements of critical parts (*e.g.* to restrain motion in a protein that is subjected to large solvent forces when the solvent is not yet equilibrated). Another application is the restraining of particles in a shell around a region that is simulated in detail, while the shell is only approximated because it lacks proper interaction from missing particles outside the shell. Restraining will then maintain the integrity of the inner part. For spherical shells, it is a wise procedure to make the force constant depend on the radius, increasing from zero at the inner boundary to a large value at the outer boundary. This feature has not, however, been implemented in GROMACS.

The following form is used:

$$V_{pr}(\mathbf{r}_i) = \frac{1}{2} k_{pr} |\mathbf{r}_i - \mathbf{R}_i|^2 \tag{5.186}$$

The potential is plotted in Fig. 5.29.

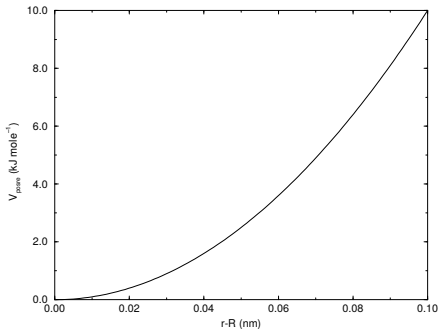



Fig. 5.29: Position restraint potential.

The potential form can be rewritten without loss of generality as:

$$V_{pr}(\mathbf{r}_i) = \frac{1}{2} \left[ k_{pr}^x (x_i - X_i)^2 \, \hat{\mathbf{x}} + k_{pr}^y (y_i - Y_i)^2 \, \hat{\mathbf{y}} + k_{pr}^z (z_i - Z_i)^2 \, \hat{\mathbf{z}} \right] \tag{5.187}$$

Now the forces are:

$$\begin{aligned} F_i^x &= -k_{pr}^x \, (x_i - X_i) \\ F_i^y &= -k_{pr}^y \, (y_i - Y_i) \\ F_i^z &= -k_{pr}^z \, (z_i - Z_i) \end{aligned} \tag{5.188}$$

Using three different force constants the position restraints can be turned on or off in each spatial dimension; this means that atoms can be harmonically restrained to a plane or a line. Position restraints are applied to a special fixed list of atoms. Such a list is usually generated by the *pdb2gmx* (page 281) program.

Note that position restraints make the potential dependent on absolute coordinates in space. Therefore, in general the pressure (and virial) is not well defined, as the pressure is the derivative of the free-energy of the system with respect to the volume. When the reference coordinates are scaled along with the system, which can be selected with the mdp option `refcoord-scaling=all` (page 67), the pressure and virial are well defined.

### Flat-bottomed position restraints

Flat-bottomed position restraints can be used to restrain particles to part of the simulation volume. No force acts on the restrained particle within the flat-bottomed region of the potential, however a harmonic force acts to move the particle to the flat-bottomed region if it is outside it. It is possible to apply normal and flat-bottomed position restraints on the same particle (however, only with the same reference position $\mathbf{R}_i$). The following general potential is used (Figure 5.30 A):

$$V_{\mathrm{fb}}(\mathbf{r}_i) = \frac{1}{2} k_{\mathrm{fb}} [d_g(\mathbf{r}_i; \mathbf{R}_i) - r_{\mathrm{fb}}]^2 \, H[d_g(\mathbf{r}_i; \mathbf{R}_i) - r_{\mathrm{fb}}], \tag{5.189}$$

where $\mathbf{R}_i$ is the reference position, $r_{\mathrm{fb}}$ is the distance from the center with a flat potential, $k_{\mathrm{fb}}$ the force constant, and $H$ is the Heaviside step function. The distance $d_g(\mathbf{r}_i; \mathbf{R}_i)$ from the reference position depends on the geometry $g$ of the flat-bottomed potential.

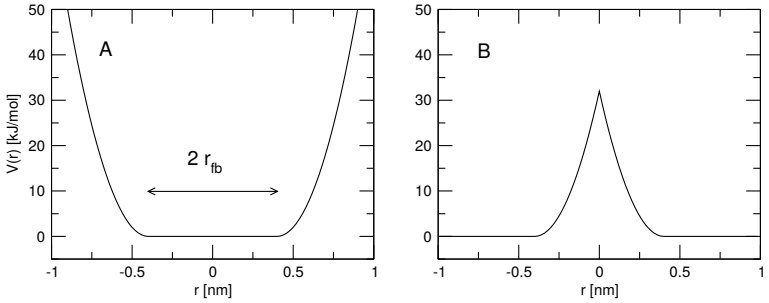



Fig. 5.30: Flat-bottomed position restraint potential. (A) Not inverted, (B) inverted.

The following geometries for the flat-bottomed potential are supported:

**Sphere** ($g = 1$): The particle is kept in a sphere of given radius. The force acts towards the center of the sphere. The following distance calculation is used:

$$d_g(\mathbf{r}_i; \mathbf{R}_i) = |\mathbf{r}_i - \mathbf{R}_i| \tag{5.190}$$

**Cylinder** ($g = 6, 7, 8$): The particle is kept in a cylinder of given radius parallel to the $x$ ($g = 6$), $y$ ($g = 7$), or $z$-axis ($g = 8$). For backwards compatibility, setting $g = 2$ is mapped to $g = 8$ in the code so that old *tpr* (page 586) files and topologies work. The force from the flat-bottomed potential acts towards the axis of the cylinder. The component of the force parallel to the cylinder axis is zero. For a cylinder aligned along the $z$-axis:

$$d_g(\mathbf{r}_i; \mathbf{R}_i) = \sqrt{(x_i - X_i)^2 + (y_i - Y_i)^2} \tag{5.191}$$

**Layer** ($g = 3, 4, 5$): The particle is kept in a layer defined by the thickness and the normal of the layer. The layer normal can be parallel to the $x$, $y$, or $z$-axis. The force acts parallel to the layer normal.

$$d_g(\mathbf{r}_i;\mathbf{R}_i) = |x_i - X_i|, \quad \text{or} \quad d_g(\mathbf{r}_i;\mathbf{R}_i) = |y_i - Y_i|, \quad \text{or} \quad d_g(\mathbf{r}_i;\mathbf{R}_i) = |z_i - Z_i|. \tag{5.192}$$

It is possible to apply multiple independent flat-bottomed position restraints of different geometry on one particle. For example, applying a cylinder and a layer in $z$ keeps a particle within a disk. Applying three layers in $x$, $y$, and $z$ keeps the particle within a cuboid.

In addition, it is possible to invert the restrained region with the unrestrained region, leading to a potential that acts to keep the particle *outside* of the volume defined by $\mathbf{R}_i$, $g$, and $r_{\mathrm{fb}}$. That feature is switched on by defining a negative $r_{\mathrm{fb}}$ in the topology. The following potential is used (Figure 5.30 B):

$$V_{\mathrm{fb}}^{\mathrm{inv}}(\mathbf{r}_i) = \frac{1}{2} k_{\mathrm{fb}} [d_g(\mathbf{r}_i;\mathbf{R}_i) - |r_{\mathrm{fb}}|]^2 \, H[-(d_g(\mathbf{r}_i;\mathbf{R}_i) - |r_{\mathrm{fb}}|)]. \tag{5.193}$$

### Angle restraints

These are used to restrain the angle between two pairs of particles or between one pair of particles and the $z$-axis. The functional form is similar to that of a proper dihedral. For two pairs of atoms:

$$V_{ar}(\mathbf{r}_i,\mathbf{r}_j,\mathbf{r}_k,\mathbf{r}_l) = k_{ar}(1 - \cos(n(\theta - \theta_0))), \quad \text{where } \theta = \arccos\left(\frac{\mathbf{r}_j - \mathbf{r}_i}{\|\mathbf{r}_j - \mathbf{r}_i\|} \cdot \frac{\mathbf{r}_l - \mathbf{r}_k}{\|\mathbf{r}_l - \mathbf{r}_k\|}\right) \tag{5.194}$$

For one pair of atoms and the $z$-axis:

$$V_{ar}(\mathbf{r}_i,\mathbf{r}_j) = k_{ar}(1 - \cos(n(\theta - \theta_0))), \quad \text{where } \theta = \arccos\left(\frac{\mathbf{r}_j - \mathbf{r}_i}{\|\mathbf{r}_j - \mathbf{r}_i\|} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\right) \tag{5.195}$$

A multiplicity ($n$) of 2 is useful when you do not want to distinguish between parallel and anti-parallel vectors. The equilibrium angle $\theta$ should be between 0 and 180 degrees for multiplicity 1 and between 0 and 90 degrees for multiplicity 2.

### Dihedral restraints

These are used to restrain the dihedral angle $\phi$ defined by four particles as in an improper dihedral (sec. *Improper dihedrals* (page 493)) but with a slightly modified potential. Using:

$$\phi' = (\phi - \phi_0) \ \mathrm{MOD}\ 2\pi \tag{5.196}$$

where $\phi_0$ is the reference angle, the potential is defined as:

$$V_{dihr}(\phi') \ = \ \begin{cases} \frac{1}{2} k_{dihr}(\phi' - \Delta\phi)^2 & \text{for} \quad \|\phi'\| \ > \ \Delta\phi \\ 0 & \text{for} \quad \|\phi'\| \ \leq \ \Delta\phi \end{cases} \tag{5.197}$$

where $\Delta\phi$ is a user defined angle and $k_{dihr}$ is the force constant. **Note** that in the input in topology files, angles are given in degrees and force constants in kJ/mol/rad$^2$.

### Distance restraints

Distance restraints add a penalty to the potential when the distance between specified pairs of atoms exceeds a threshold value. They are normally used to impose experimental restraints from, for instance, experiments in nuclear magnetic resonance (NMR), on the motion of the system. Thus, MD can be used for structure refinement using NMR data. In GROMACS there are three ways to impose restraints on pairs of atoms:

- Simple harmonic restraints: use `[ bonds ]` type 6 (see sec. *Exclusions* (page 544)).
- Piecewise linear/harmonic restraints: `[ bonds ]` type 10.
- Complex NMR distance restraints, optionally with pair, time and/or ensemble averaging.

The last two options will be detailed now.

The potential form for distance restraints is quadratic below a specified lower bound and between two specified upper bounds, and linear beyond the largest bound (see Fig. 5.31).

$$V_{dr}(r_{ij}) = \begin{cases} \frac{1}{2}k_{dr}(r_{ij}-r_0)^2 & \text{for } r_{ij} < r_0 \\ 0 & \text{for } r_0 \le r_{ij} < r_1 \\ \frac{1}{2}k_{dr}(r_{ij}-r_1)^2 & \text{for } r_1 \le r_{ij} < r_2 \\ \frac{1}{2}k_{dr}(r_2-r_1)(2r_{ij}-r_2-r_1) & \text{for } r_2 \le r_{ij} \end{cases} \tag{5.198}$$

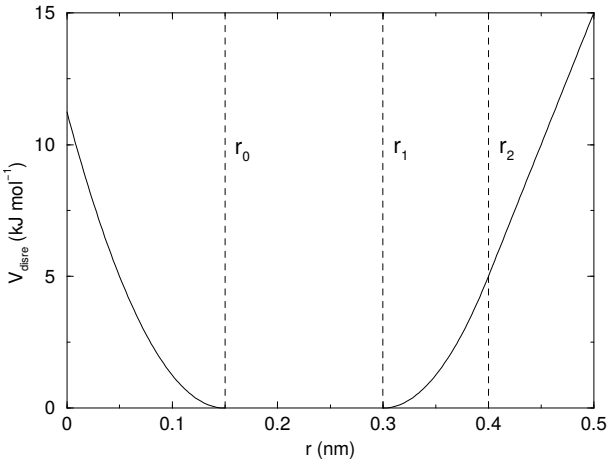



Fig. 5.31: Distance Restraint potential.

The forces are

$$\mathbf{F}_i = \begin{cases} -k_{dr}(r_{ij}-r_0)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for } r_{ij} < r_0 \\ 0 & \text{for } r_0 \le r_{ij} < r_1 \\ -k_{dr}(r_{ij}-r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for } r_1 \le r_{ij} < r_2 \\ -k_{dr}(r_2-r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for } r_2 \le r_{ij} \end{cases} \tag{5.199}$$

For restraints not derived from NMR data, this functionality will usually suffice and a section of `[ bonds ]` type 10 can be used to apply individual restraints between pairs of atoms, see *Topology file* (page 554). For applying restraints derived from NMR measurements, more complex functionality might be required, which is provided through the `[ distance_restraints ]` section and is described below.

### Time averaging

Distance restraints based on instantaneous distances can potentially reduce the fluctuations in a molecule significantly. This problem can be overcome by restraining to a *time averaged* distance *91* (page 687). The forces with time averaging are:

$$\mathbf{F}_i = \begin{cases} -k^a_{dr}(\bar{r}_{ij} - r_0)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad \bar{r}_{ij} < r_0 \\ 0 & \text{for} \quad r_0 \leq \bar{r}_{ij} < r_1 \\ -k^a_{dr}(\bar{r}_{ij} - r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_1 \leq \bar{r}_{ij} < r_2 \\ -k^a_{dr}(r_2 - r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_2 \leq \bar{r}_{ij} \end{cases} \tag{5.200}$$

where $\bar{r}_{ij}$ is given by an exponential running average with decay time $\tau$:

$$\bar{r}_{ij} = < r_{ij}^{-3} >^{-1/3} \tag{5.201}$$

The force constant $k^a_{dr}$ is switched on slowly to compensate for the lack of history at the beginning of the simulation:

$$k^a_{dr} = k_{dr}\left(1 - \exp\left(-\frac{t}{\tau}\right)\right) \tag{5.202}$$

Because of the time averaging, we can no longer speak of a distance restraint potential.

This way an atom can satisfy two incompatible distance restraints *on average* by moving between two positions. An example would be an amino acid side-chain that is rotating around its $\chi$ dihedral angle, thereby coming close to various other groups. Such a mobile side chain can give rise to multiple NOEs that can not be fulfilled by a single structure.

The computation of the time averaged distance in the *mdrun* (page 258) program is done in the following fashion:

$$\begin{aligned} \overline{r^{-3}}_{ij}(0) &= r_{ij}(0)^{-3} \\ \overline{r^{-3}}_{ij}(t) &= \overline{r^{-3}}_{ij}(t - \Delta t)\, \exp\left(-\tfrac{\Delta t}{\tau}\right) + r_{ij}(t)^{-3}\left[1 - \exp\left(-\tfrac{\Delta t}{\tau}\right)\right] \end{aligned} \tag{5.203}$$

When a pair is within the bounds, it can still feel a force because the time averaged distance can still be beyond a bound. To prevent the protons from being pulled too close together, a mixed approach can be used. In this approach, the penalty is zero when the instantaneous distance is within the bounds, otherwise the violation is the square root of the product of the instantaneous violation and the time averaged violation:

$$\mathbf{F}_i = \begin{cases} k^a_{dr}\sqrt{(r_{ij} - r_0)(\bar{r}_{ij} - r_0)}\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_{ij} < r_0 \quad \text{and} \quad \bar{r}_{ij} < r_0 \\ -k^a_{dr}\min\left(\sqrt{(r_{ij} - r_1)(\bar{r}_{ij} - r_1)}, r_2 - r_1\right)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_{ij} > r_1 \quad \text{and} \quad \bar{r}_{ij} > r_1 \\ 0 & \text{otherwise} \end{cases} \tag{5.204}$$

### Averaging over multiple pairs

Sometimes it is unclear from experimental data which atom pair gives rise to a single NOE, in other occasions it can be obvious that more than one pair contributes due to the symmetry of the system, *e.g.* a methyl group with three protons. For such a group, it is not possible to distinguish between the protons, therefore they should all be taken into account when calculating the distance between this methyl group and another proton (or group of protons). Due to the physical nature of magnetic resonance, the intensity of the NOE signal is inversely proportional to the sixth power of the inter-atomic distance. Thus, when combining atom pairs, a fixed list of $N$ restraints may be taken together, where the apparent "distance" is given by:

$$r_N(t) = \left[ \sum_{n=1}^{N} \bar{r}_n(t)^{-6} \right]^{-1/6} \tag{5.205}$$

where we use $r_{ij}$ or (5.201) for the $\bar{r}_n$. The $r_N$ of the instantaneous and time-averaged distances can be combined to do a mixed restraining, as indicated above. As more pairs of protons contribute to the same NOE signal, the intensity will increase, and the summed "distance" will be shorter than any of its components due to the reciprocal summation.

There are two options for distributing the forces over the atom pairs. In the conservative option, the force is defined as the derivative of the restraint potential with respect to the coordinates. This results in a conservative potential when time averaging is not used. The force distribution over the pairs is proportional to $r^{-6}$. This means that a close pair feels a much larger force than a distant pair, which might lead to a molecule that is "too rigid." The other option is an equal force distribution. In this case each pair feels $1/N$ of the derivative of the restraint potential with respect to $r_N$. The advantage of this method is that more conformations might be sampled, but the non-conservative nature of the forces can lead to local heating of the protons.

It is also possible to use *ensemble averaging* using multiple (protein) molecules. In this case the bounds should be lowered as in:

$$\begin{aligned} r_1 &= r_1 * M^{-1/6} \\ r_2 &= r_2 * M^{-1/6} \end{aligned} \tag{5.206}$$

where $M$ is the number of molecules. The GROMACS preprocessor *grompp* (page 230) can do this automatically when the appropriate option is given. The resulting "distance" is then used to calculate the scalar force according to:

$$\mathbf{F}_i = \begin{cases} 0 & r_N < r_1 \\ k_{dr}(r_N - r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & r_1 \le r_N < r_2 \\ k_{dr}(r_2 - r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & r_N \ge r_2 \end{cases} \tag{5.207}$$

where $i$ and $j$ denote the atoms of all the pairs that contribute to the NOE signal.

### Using distance restraints

A list of distance restrains based on NOE data can be added to a molecule definition in your topology file, like in the following example:

```
[ distance_restraints ]
; ai    aj    type    index    type'      low      up1      up2      fac
10      16       1        0        1      0.0      0.3      0.4      1.0
```

(continues on next page)

(continued from previous page)

```
10      28      1       1       1       0.0     0.3     0.4     1.0
10      46      1       1       1       0.0     0.3     0.4     1.0
16      22      1       2       1       0.0     0.3     0.4     2.5
16      34      1       3       1       0.0     0.5     0.6     1.0
```

In this example a number of features can be found. In columns ai and aj you find the atom numbers of the particles to be restrained. The type column should always be 1. As explained in *Distance restraints* (page 503), multiple distances can contribute to a single NOE signal. In the topology this can be set using the index column. In our example, the restraints 10-28 and 10-46 both have index 1, therefore they are treated simultaneously. An extra requirement for treating restraints together is that the restraints must be on successive lines, without any other intervening restraint. The type' column will usually be 1, but can be set to 2 to obtain a distance restraint that will never be time- and ensemble-averaged; this can be useful for restraining hydrogen bonds. The columns `low`, `up1`, and `up2` hold the values of $r_0$, $r_1$, and $r_2$ from (5.198). In some cases it can be useful to have different force constants for some restraints; this is controlled by the column `fac`. The force constant in the parameter file is multiplied by the value in the column `fac` for each restraint. Information for each restraint is stored in the energy file and can be processed and plotted with *gmx nmr* (page 272).

### Orientation restraints

This section describes how orientations between vectors, as measured in certain NMR experiments, can be calculated and restrained in MD simulations. The presented refinement methodology and a comparison of results with and without time and ensemble averaging have been published *92* (page 687).

### Theory

In an NMR experiment, orientations of vectors can be measured when a molecule does not tumble completely isotropically in the solvent. Two examples of such orientation measurements are residual dipolar couplings (between two nuclei) or chemical shift anisotropies. An observable for a vector $\mathbf{r}_i$ can be written as follows:

$$\delta_i = \frac{2}{3}\mathrm{tr}(\mathbf{S}\mathbf{D}_i) \tag{5.208}$$

where $\mathbf{S}$ is the dimensionless order tensor of the molecule. The tensor $\mathbf{D}_i$ is given by:

$$\mathbf{D}_i = \frac{c_i}{\|\mathbf{r}_i\|^\alpha}\begin{pmatrix} 3xx-1 & 3xy & 3xz \\ 3xy & 3yy-1 & 3yz \\ 3xz & 3yz & 3zz-1 \end{pmatrix} \tag{5.209}$$

$$\text{with:}\quad x = \frac{r_{i,x}}{\|\mathbf{r}_i\|},\quad y = \frac{r_{i,y}}{\|\mathbf{r}_i\|},\quad z = \frac{r_{i,z}}{\|\mathbf{r}_i\|} \tag{5.210}$$

For a dipolar coupling $\mathbf{r}_i$ is the vector connecting the two nuclei, $\alpha = 3$ and the constant $c_i$ is given by:

$$c_i = \frac{\mu_0}{4\pi}\gamma_1^i\gamma_2^i\frac{\hbar}{4\pi} \tag{5.211}$$

where $\gamma_1^i$ and $\gamma_2^i$ are the gyromagnetic ratios of the two nuclei.

The order tensor is symmetric and has trace zero. Using a rotation matrix $\mathbf{T}$ it can be transformed into the following form:

$$\mathbf{T}^T\mathbf{S}\mathbf{T} = s\begin{pmatrix} -\frac{1}{2}(1-\eta) & 0 & 0 \\ 0 & -\frac{1}{2}(1+\eta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.212}$$

where $-1 \leq s \leq 1$ and $0 \leq \eta \leq 1$. $s$ is called the order parameter and $\eta$ the asymmetry of the order tensor $\mathbf{S}$. When the molecule tumbles isotropically in the solvent, $s$ is zero, and no orientational effects can be observed because all $\delta_i$ are zero.

### Calculating orientations in a simulation

For reasons which are explained below, the $\mathbf{D}$ matrices are calculated which respect to a reference orientation of the molecule. The orientation is defined by a rotation matrix $\mathbf{R}$, which is needed to least-squares fit the current coordinates of a selected set of atoms onto a reference conformation. The reference conformation is the starting conformation of the simulation. In case of ensemble averaging, which will be treated later, the structure is taken from the first subsystem. The calculated $\mathbf{D}_i^c$ matrix is given by:

$$\mathbf{D}_i^c(t) = \mathbf{R}(t)\mathbf{D}_i(t)\mathbf{R}^T(t) \tag{5.213}$$

The calculated orientation for vector $i$ is given by:

$$\delta_i^c(t) = \frac{2}{3}\text{tr}(\mathbf{S}(t)\mathbf{D}_i^c(t)) \tag{5.214}$$

The order tensor $\mathbf{S}(t)$ is usually unknown. A reasonable choice for the order tensor is the tensor which minimizes the (weighted) mean square difference between the calculated and the observed orientations:

$$MSD(t) = \left(\sum_{i=1}^{N} w_i\right)^{-1} \sum_{i=1}^{N} w_i(\delta_i^c(t) - \delta_i^{exp})^2 \tag{5.215}$$

To properly combine different types of measurements, the unit of $w_i$ should be such that all terms are dimensionless. This means the unit of $w_i$ is the unit of $\delta_i$ to the power $-2$. **Note** that scaling all $w_i$ with a constant factor does not influence the order tensor.

### Time averaging

Since the tensors $\mathbf{D}_i$ fluctuate rapidly in time, much faster than can be observed in an experiment, they should be averaged over time in the simulation. However, in a simulation the time and the number of copies of a molecule are limited. Usually one can not obtain a converged average of the $\mathbf{D}_i$ tensors over all orientations of the molecule. If one assumes that the average orientations of the $\mathbf{r}_i$ vectors within the molecule converge much faster than the tumbling time of the molecule, the tensor can be averaged in an axis system that rotates with the molecule, as expressed by (5.213)). The time-averaged tensors are calculated using an exponentially decaying memory function:

$$\mathbf{D}_i^a(t) = \frac{\displaystyle\int_{u=t_0}^{t} \mathbf{D}_i^c(u) \exp\left(-\frac{t-u}{\tau}\right) \mathrm{d}u}{\displaystyle\int_{u=t_0}^{t} \exp\left(-\frac{t-u}{\tau}\right) \mathrm{d}u} \tag{5.216}$$

Assuming that the order tensor $\mathbf{S}$ fluctuates slower than the $\mathbf{D}_i$, the time-averaged orientation can be calculated as:

$$\delta_i^a(t) = \frac{2}{3}\text{tr}(\mathbf{S}(t)\mathbf{D}_i^a(t)) \tag{5.217}$$

where the order tensor $\mathbf{S}(t)$ is calculated using expression (5.215) with $\delta_i^c(t)$ replaced by $\delta_i^a(t)$.

### Restraining

The simulated structure can be restrained by applying a force proportional to the difference between the calculated and the experimental orientations. When no time averaging is applied, a proper potential can be defined as:

$$V = \frac{1}{2} k \sum_{i=1}^{N} w_i (\delta_i^c(t) - \delta_i^{exp})^2 \tag{5.218}$$

where the unit of $k$ is the unit of energy. Thus the effective force constant for restraint $i$ is $kw_i$. The forces are given by minus the gradient of $V$. The force $\mathbf{F}_i$ working on vector $\mathbf{r}_i$ is:

$$\begin{aligned} \mathbf{F}_i(t) &= -\frac{\mathrm{d}V}{\mathrm{d}\mathbf{r}_i} \\ &= -kw_i(\delta_i^c(t) - \delta_i^{exp}) \frac{\mathrm{d}\delta_i(t)}{\mathrm{d}\mathbf{r}_i} \\ &= -kw_i(\delta_i^c(t) - \delta_i^{exp}) \frac{2c_i}{\|\mathbf{r}\|^{2+\alpha}} \left( 2\mathbf{R}^T\mathbf{S}\mathbf{R}\mathbf{r}_i - \frac{2+\alpha}{\|\mathbf{r}\|^2} \mathrm{tr}(\mathbf{R}^T\mathbf{S}\mathbf{R}\mathbf{r}_i\mathbf{r}_i^T)\mathbf{r}_i \right) \end{aligned} \tag{5.219}$$

### Ensemble averaging

Ensemble averaging can be applied by simulating a system of $M$ subsystems that each contain an identical set of orientation restraints. The systems only interact via the orientation restraint potential which is defined as:

$$V = M \frac{1}{2} k \sum_{i=1}^{N} w_i \langle \delta_i^c(t) - \delta_i^{exp} \rangle^2 \tag{5.220}$$

The force on vector $\mathbf{r}_{i,m}$ in subsystem $m$ is given by:

$$\mathbf{F}_{i,m}(t) = -\frac{\mathrm{d}V}{\mathrm{d}\mathbf{r}_{i,m}} = -kw_i \langle \delta_i^c(t) - \delta_i^{exp} \rangle \frac{\mathrm{d}\delta_{i,m}^c(t)}{\mathrm{d}\mathbf{r}_{i,m}} \tag{5.221}$$

### Time averaging

When using time averaging it is not possible to define a potential. We can still define a quantity that gives a rough idea of the energy stored in the restraints:

$$V = M \frac{1}{2} k^a \sum_{i=1}^{N} w_i \langle \delta_i^a(t) - \delta_i^{exp} \rangle^2 \tag{5.222}$$

The force constant $k_a$ is switched on slowly to compensate for the lack of history at times close to $t_0$. It is exactly proportional to the amount of average that has been accumulated:

$$k^a = k \frac{1}{\tau} \int_{u=t_0}^{t} \exp\left( -\frac{t-u}{\tau} \right) \mathrm{d}u \tag{5.223}$$

What really matters is the definition of the force. It is chosen to be proportional to the square root of the product of the time-averaged and the instantaneous deviation. Using only the time-averaged deviation induces large oscillations. The force is given by:

$$\mathbf{F}_{i,m}(t) = \begin{cases} 0 & \text{for} \quad a\,b \leq 0 \\ k^a w_i \dfrac{a}{|a|} \sqrt{a\,b} \dfrac{\mathrm{d}\delta_{i,m}^c(t)}{\mathrm{d}\mathbf{r}_{i,m}} & \text{for} \quad a\,b > 0 \end{cases} \tag{5.224}$$

$$\begin{aligned} a = & \langle \delta_i^a(t) - \delta_i^{exp} \rangle \\ b = & \langle \delta_i^c(t) - \delta_i^{exp} \rangle \end{aligned} \tag{5.225}$$

### Using orientation restraints

Orientation restraints can be added to a molecule definition in the topology file in the section `[ orientation_restraints ]`. Here we give an example section containing five N-H residual dipolar coupling restraints:

```
[ orientation_restraints ]
; ai    aj  type  exp.  label  alpha     const.      obs.    weight
;                              Hz        nm^3        Hz     Hz^-2
  31    32     1     1      3      3     6.083    -6.73       1.0
  43    44     1     1      4      3     6.083    -7.87       1.0
  55    56     1     1      5      3     6.083    -7.13       1.0
  65    66     1     1      6      3     6.083    -2.57       1.0
  73    74     1     1      7      3     6.083    -2.10       1.0
```

The unit of the observable is Hz, but one can choose any other unit. In columns `ai` and `aj` you find the atom numbers of the particles to be restrained. The `type` column should always be 1. The `exp.` column denotes the experiment number, starting at 1. For each experiment a separate order tensor $\mathbf{S}$ is optimized. The label should be a unique number larger than zero for each restraint. The `alpha` column contains the power $\alpha$ that is used in (5.209)) to calculate the orientation. The `const.` column contains the constant $c_i$ used in the same equation. The constant should have the unit of the observable times nm$^\alpha$. The column `obs.` contains the observable, in any unit you like. The last column contains the weights $w_i$; the unit should be the inverse of the square of the unit of the observable.

Some parameters for orientation restraints can be specified in the *grompp* (page 230) *mdp* (page 579) file, for a study of the effect of different force constants and averaging times and ensemble averaging see *92* (page 687). Information for each restraint is stored in the energy file and can be processed and plotted with *gmx nmr* (page 272).

### 5.5.4 Polarization

Polarization can be treated by GROMACS by attaching shell (Drude) particles to atoms and/or virtual sites. The energy of the shell particle is then minimized at each time step in order to remain on the Born-Oppenheimer surface.

#### Simple polarization

This is implemented as a harmonic potential with equilibrium distance 0. The input given in the topology file is the polarizability $\alpha$ (in GROMACS units) as follows:

```
[ polarization ]
; Atom i   j  type   alpha
1          2  1      0.001
```

in this case the polarizability volume is 0.001 nm$^3$ (or 1 Å$^3$). In order to compute the harmonic force constant $k_{cs}$ (where $cs$ stands for core-shell), the following is used *45* (page 685):

$$k_{cs} = \frac{q_s^2}{\alpha} \tag{5.226}$$

where $q_s$ is the charge on the shell particle.

#### Anharmonic polarization

For the development of the Drude force field by Roux and McKerell *93* (page 687) it was found that some particles can overpolarize and this was fixed by introducing a higher order term in the polarization energy:

$$\begin{aligned} V_{pol} &= \frac{k_{cs}}{2} r_{cs}^2 && r_{cs} \leq \delta \\ &= \frac{k_{cs}}{2} r_{cs}^2 + k_{hyp}(r_{cs} - \delta)^4 && r_{cs} > \delta \end{aligned} \tag{5.227}$$

where $\delta$ is a user-defined constant that is set to 0.02 nm for anions in the Drude force field *94* (page 687). Since this original introduction it has also been used in other atom types *93* (page 687).

```
[ polarization ]
;Atom i j    type   alpha (nm^3)    delta  khyp
1       2       2       0.001786     0.02  16.736e8
```

The above force constant $k_{hyp}$ corresponds to 4·10$^8$ kcal/mol/nm$^4$, hence the strange number.

#### Water polarization

A special potential for water that allows anisotropic polarization of a single shell particle *45* (page 685).

### Thole polarization

Based on early work by Thole *95* (page 687), Roux and coworkers have implemented potentials for molecules like ethanol *96* (page 688), *98* (page 688). Within such molecules, there are intra-molecular interactions between shell particles, however these must be screened because full Coulomb would be too strong. The potential between two shell particles $i$ and $j$ is:

$$V_{thole} = \frac{q_i q_j}{r_{ij}} \left[ 1 - \left( 1 + \frac{\bar{r}_{ij}}{2} \right) \exp^{-\bar{r}_{ij}} \right] \tag{5.228}$$

**Note** that there is a sign error in Equation 1 of Noskov *et al.* *98* (page 688):

$$\bar{r}_{ij} = a \frac{r_{ij}}{(\alpha_i \alpha_j)^{1/6}} \tag{5.229}$$

where $a$ is a magic (dimensionless) constant, usually chosen to be 2.6 *98* (page 688); $\alpha_i$ and $\alpha_j$ are the polarizabilities of the respective shell particles.

## 5.5.5 Free energy interactions

This section describes the $\lambda$-dependence of the potentials used for free energy calculations (see sec. *Free energy calculations* (page 469)). All common types of potentials and constraints can be interpolated smoothly from state A ($\lambda = 0$) to state B ($\lambda = 1$) and vice versa. All bonded interactions are interpolated by linear interpolation of the interaction parameters. Non-bonded interactions can be interpolated linearly or via soft-core interactions.

Starting in GROMACS 4.6, $\lambda$ is a vector, allowing different components of the free energy transformation to be carried out at different rates. Coulomb, Lennard-Jones, bonded, and restraint terms can all be controlled independently, as described in the *mdp* (page 579) options.

### Harmonic potentials

The example given here is for the bond potential, which is harmonic in GROMACS. However, these equations apply to the angle potential and the improper dihedral potential as well.

$$\begin{aligned} V_b = & \quad \frac{1}{2} \left[ (1-\lambda) k_b^A + \lambda k_b^B \right] \left[ b - (1-\lambda) b_0^A - \lambda b_0^B \right]^2 \\ \frac{\partial V_b}{\partial \lambda} = & \quad \frac{1}{2} (k_b^B - k_b^A) \left[ b - (1-\lambda) b_0^A + \lambda b_0^B \right]^2 + \\ & (b_0^A - b_0^B) \left[ b - (1-\lambda) b_0^A - \lambda b_0^B \right] \left[ (1-\lambda) k_b^A + \lambda k_b^B \right] \end{aligned}$$

### GROMOS-96 bonds and angles

Fourth-power bond stretching and cosine-based angle potentials are interpolated by linear interpolation of the force constant and the equilibrium position. Formulas are not given here.

### Proper dihedrals

For the proper dihedrals, the equations are somewhat more complicated:

$$\begin{aligned} V_d = & \left[(1-\lambda)k_d^A + \lambda k_d^B\right]\left(1+\cos\left[n_\phi\phi - (1-\lambda)\phi_s^A - \lambda\phi_s^B\right]\right) \\ \frac{\partial V_d}{\partial\lambda} = & \left(k_d^B - k_d^A\right)\left(1+\cos\left[n_\phi\phi - (1-\lambda)\phi_s^A - \lambda\phi_s^B\right]\right) + \\ & \left(\phi_s^B - \phi_s^A\right)\left[(1-\lambda)k_d^A - \lambda k_d^B\right]\sin\left[n_\phi\phi - (1-\lambda)\phi_s^A - \lambda\phi_s^B\right] \end{aligned}$$

**Note:** that the multiplicity $n_\phi$ can not be parameterized because the function should remain periodic on the interval $[0, 2\pi]$.

### Tabulated bonded interactions

For tabulated bonded interactions only the force constant can interpolated:

$$\begin{aligned} V = & \left((1-\lambda)k^A + \lambda k^B\right) f \\ \frac{\partial V}{\partial\lambda} = & \left(k^B - k^A\right) f \end{aligned} \tag{5.230}$$

### Coulomb interaction

The Coulomb interaction between two particles of which the charge varies with $\lambda$ is:

$$\begin{aligned} V_c = & \frac{f}{\varepsilon_{rf} r_{ij}}\left[(1-\lambda)q_i^A q_j^A + \lambda\, q_i^B q_j^B\right] \\ \frac{\partial V_c}{\partial\lambda} = & \frac{f}{\varepsilon_{rf} r_{ij}}\left[-q_i^A q_j^A + q_i^B q_j^B\right] \end{aligned} \tag{5.231}$$

where $f = \frac{1}{4\pi\varepsilon_0} = 138.935\,458$ (see chapter *Definitions and Units* (page 425)).

### Coulomb interaction with reaction field

The Coulomb interaction including a reaction field, between two particles of which the charge varies with $\lambda$ is:

$$\begin{aligned} V_c = & f\left[\frac{1}{r_{ij}} + k_{rf}\, {r_{ij}}^2 - c_{rf}\right]\left[(1-\lambda)q_i^A q_j^A + \lambda\, q_i^B q_j^B\right] \\ \frac{\partial V_c}{\partial\lambda} = & f\left[\frac{1}{r_{ij}} + k_{rf}\, {r_{ij}}^2 - c_{rf}\right]\left[-q_i^A q_j^A + q_i^B q_j^B\right] \end{aligned} \tag{5.232}$$

**Note** that the constants $k_{rf}$ and $c_{rf}$ are defined using the dielectric constant $\varepsilon_{rf}$ of the medium (see sec. *Coulomb interaction with reaction field* (page 484)).

### Lennard-Jones interaction

For the Lennard-Jones interaction between two particles of which the *atom type* varies with $\lambda$ we can write:

$$\begin{aligned} V_{LJ} &= \frac{(1-\lambda)C_{12}^A + \lambda\, C_{12}^B}{{r_{ij}}^{12}} - \frac{(1-\lambda)C_6^A + \lambda\, C_6^B}{{r_{ij}}^6} \\ \frac{\partial V_{LJ}}{\partial \lambda} &= \frac{C_{12}^B - C_{12}^A}{{r_{ij}}^{12}} - \frac{C_6^B - C_6^A}{{r_{ij}}^6} \end{aligned} \tag{5.233}$$

It should be noted that it is also possible to express a pathway from state A to state B using $\sigma$ and $\epsilon$ (see (5.123)). It may seem to make sense physically to vary the force field parameters $\sigma$ and $\epsilon$ rather than the derived parameters $C_{12}$ and $C_6$. However, the difference between the pathways in parameter space is not large, and the free energy itself does not depend on the pathway, so we use the simple formulation presented above.

### Kinetic Energy

When the mass of a particle changes, there is also a contribution of the kinetic energy to the free energy (note that we can not write the momentum $\mathbf{p}$ as $m\mathbf{v}$, since that would result in the sign of $\frac{\partial E_k}{\partial \lambda}$ being incorrect *99* (page 688)):

$$\begin{aligned} E_k &= \frac{1}{2}\frac{\mathbf{p}^2}{(1-\lambda)m^A + \lambda m^B} \\ \frac{\partial E_k}{\partial \lambda} &= -\frac{1}{2}\frac{\mathbf{p}^2(m^B - m^A)}{((1-\lambda)m^A + \lambda m^B)^2} \end{aligned} \tag{5.234}$$

after taking the derivative, we *can* insert $\mathbf{p} = m\mathbf{v}$, such that:

$$\frac{\partial E_k}{\partial \lambda} = -\frac{1}{2}\mathbf{v}^2(m^B - m^A) \tag{5.235}$$

### Constraints

The constraints are formally part of the Hamiltonian, and therefore they give a contribution to the free energy. In GROMACS this can be calculated using the LINCS or the SHAKE algorithm. If we have $k = 1 \ldots K$ constraint equations $g_k$ for LINCS, then

$$g_k = |\mathbf{r}_k| - d_k \tag{5.236}$$

where $\mathbf{r}_k$ is the displacement vector between two particles and $d_k$ is the constraint distance between the two particles. We can express the fact that the constraint distance has a $\lambda$ dependency by

$$d_k = (1-\lambda)d_k^A + \lambda d_k^B \tag{5.237}$$

Thus the $\lambda$-dependent constraint equation is

$$g_k = |\mathbf{r}_k| - \left((1-\lambda)d_k^A + \lambda d_k^B\right). \tag{5.238}$$

The (zero) contribution $G$ to the Hamiltonian from the constraints (using Lagrange multipliers $\lambda_k$, which are logically distinct from the free-energy $\lambda$) is

$$\begin{aligned} G &= \sum_k^K \lambda_k g_k \\ \frac{\partial G}{\partial \lambda} &= \frac{\partial G}{\partial d_k}\frac{\partial d_k}{\partial \lambda} \\ &= -\sum_k^K \lambda_k \left(d_k^B - d_k^A\right) \end{aligned} \tag{5.239}$$

For SHAKE, the constraint equations are

$$g_k = \mathbf{r}_k^2 - d_k^2 \tag{5.240}$$

with $d_k$ as before, so

$$\frac{\partial G}{\partial \lambda} = -2\sum_k^K \lambda_k \left(d_k^B - d_k^A\right) \tag{5.241}$$

### Soft-core interactions: Beutler *et al.*

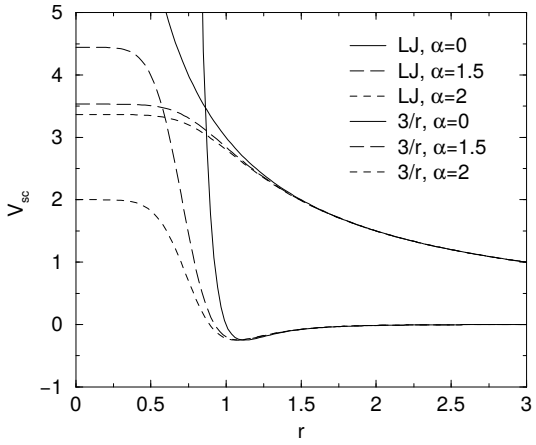



Fig. 5.32: Soft-core interactions at $\lambda = 0.5$, with $p = 2$ and $C_6^A = C_{12}^A = C_6^B = C_{12}^B = 1$.

In a free-energy calculation where particles grow out of nothing, or particles disappear, using the simple linear interpolation of the Lennard-Jones and Coulomb potentials as described in (5.233) and (5.232) may lead to poor convergence. When the particles have nearly disappeared, or are close to appearing (at $\lambda$ close to 0 or 1), the interaction energy will be weak enough for particles to get very close to each other, leading to large fluctuations in the measured values of $\partial V/\partial\lambda$ (which, because of the simple linear interpolation, depends on the potentials at both the endpoints of $\lambda$).

To circumvent these problems, the singularities in the potentials need to be removed. This can be done by modifying the regular Lennard-Jones and Coulomb potentials with "soft-core" potentials that limit the energies and forces involved at $\lambda$ values between 0 and 1, but not *at* $\lambda = 0$ or 1.

In GROMACS the soft-core potentials $V_{sc}$ are shifted versions of the regular potentials, so that the singularity in the potential and its derivatives at $r = 0$ is never reached. This formulation was introduced

by Beutler *et al.*[100] (page 688):

$$\begin{aligned} V_{sc}(r) = &\quad (1-\lambda)V^A(r_A) + \lambda V^B(r_B) \\ r_A = &\quad \left(\alpha\sigma_A^6\lambda^p + r^6\right)^{\frac{1}{6}} \\ r_B = &\quad \left(\alpha\sigma_B^6(1-\lambda)^p + r^6\right)^{\frac{1}{6}} \end{aligned} \tag{5.242}$$

where $V^A$ and $V^B$ are the normal "hard core" Van der Waals or electrostatic potentials in state A ($\lambda = 0$) and state B ($\lambda = 1$) respectively, $\alpha$ is the soft-core parameter (set with `sc_alpha` in the *mdp* (page 579) file), $p$ is the soft-core $\lambda$ power (set with `sc_power`), $\sigma$ is the radius of the interaction, which is $(C_{12}/C_6)^{1/6}$ or an input parameter (`sc_sigma`) when $C_6$ or $C_{12}$ is zero. Beutler *et al.*[100] (page 688) probed various combinations of the $r$ power values for the Lennard-Jones and Coulombic interactions. GROMACS uses $r^6$ for both, van der Waals and electrostatic interactions.

For intermediate $\lambda$, $r_A$ and $r_B$ alter the interactions very little for $r > \alpha^{1/6}\sigma$ and quickly switch the soft-core interaction to an almost constant value for smaller $r$ (Fig. 5.32). The force is:

$$F_{sc}(r) = -\frac{\partial V_{sc}(r)}{\partial r} = (1-\lambda)F^A(r_A)\left(\frac{r}{r_A}\right)^5 + \lambda F^B(r_B)\left(\frac{r}{r_B}\right)^5 \tag{5.243}$$

where $F^A$ and $F^B$ are the "hard core" forces. The contribution to the derivative of the free energy is:

$$\begin{aligned} \frac{\partial V_{sc}(r)}{\partial\lambda} = &\quad V^B(r_B) - V^A(r_A) + (1-\lambda)\frac{\partial V^A(r_A)}{\partial r_A}\frac{\partial r_A}{\partial\lambda} + \lambda\frac{\partial V^B(r_B)}{\partial r_B}\frac{\partial r_B}{\partial\lambda} \\ = &\quad V^B(r_B) - V^A(r_A) + \\ &\quad \frac{p\alpha}{6}\left[\lambda F^B(r_B)r_B^{-5}\sigma_B^6(1-\lambda)^{p-1} - (1-\lambda)F^A(r_A)r_A^{-5}\sigma_A^6\lambda^{p-1}\right] \end{aligned}$$

The original GROMOS Lennard-Jones soft-core function *100* (page 688) uses $p = 2$, but $p = 1$ gives a smoother $\partial H/\partial\lambda$ curve. Another issue that should be considered is the soft-core effect of hydrogens without Lennard-Jones interaction. Their soft-core $\sigma$ is set with `sc_sigma`. These hydrogens produce peaks in $\partial H/\partial\lambda$ at $\lambda$ is 0 and/or 1 for $p = 1$ and close to 0 and/or 1 with $p = 2$. Lowering `sc_sigma` will decrease this effect, but it will also increase the interactions with hydrogens relative to the other interactions in the soft-core state.

When soft-core potentials are selected (by setting `sc_alpha >0`), and the Coulomb and Lennard-Jones potentials are turned on or off sequentially, then the Coulombic interaction is turned off linearly, rather than using soft-core interactions, which should be less statistically noisy in most cases. This behavior can be overwritten by setting `sc-coul=yes`. Note that *`sc-coul`* (page 86) is only taken into account when lambda states are used, and you can still turn off soft-core interactions by setting `sc-alpha=0`. Additionally, the soft-core interaction potential is only applied when either the A or B state has zero interaction potential. If both A and B states have nonzero interaction potential, default linear scaling described above is used. When both Coulombic and Lennard-Jones interactions are turned off simultaneously, a soft-core potential is used, and a hydrogen is being introduced or deleted, the sigma is set to `sc-sigma-min`, which itself defaults to `sc-sigma-default`.

### Soft-core interactions: Gapsys *et al.*

In this section we describe the functional form and parameters for the soft-cored non-bonded interactions using the formalism by Gapsys *et al.* *183* (page 692).

The Gapsys *et al.* soft-core is formulated to act on the level of van der Waals and electrostatic forces: the non-bonded interactions are linearized at a point defined as, $r_{scLJ}$ or $r_{scQ}$, respectively. The linearization point depends on the state of the system as controlled by the $\lambda$ parameter and two parameters $\alpha_Q$ (set with `sc-gapsys-scale-linpoint-q` (page 87)) and $\alpha_{LJ}$ (set with `sc-gapsys-scale-linpoint-lj` (page 87)). The dependence on $\lambda$ guarantees that the end-states are properly represented by their hard-core potentials. Fig. 5.33 illustrates the behaviour of the linearization point, forces and integrated potential energies with respect to the parameters $\alpha_Q$ and $\alpha_{LJ}$. The optimal choices of the parameter values have been systematically explored in *183* (page 692). These recommended values are set by default when `sc-function=gapsys` is selected: `sc-gapsys-scale-linpoint-q=0.3` and `sc-gapsys-scale-linpoint-lj=0.85`.

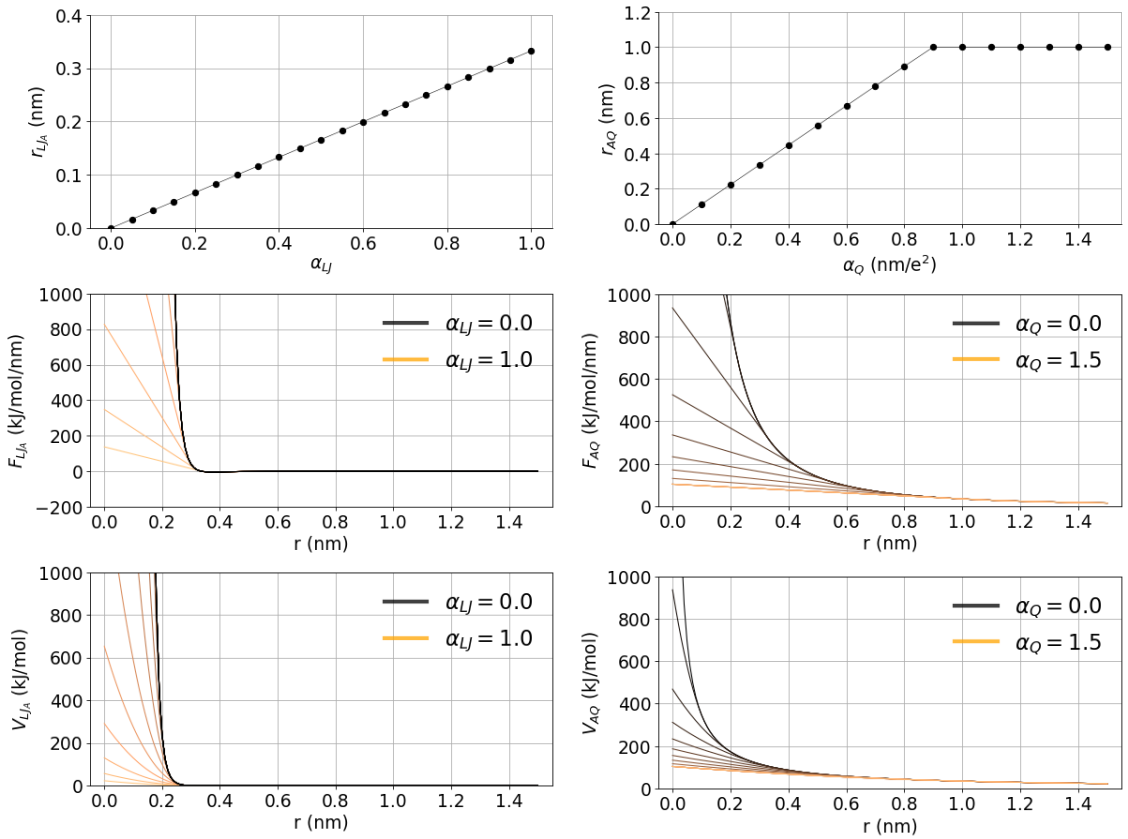



Fig. 5.33: Illustration of the soft-core parameter influence on the linearization point (top row), forces (middle row) and energies (bottom row) for van der Waals (left column) and electrostatic interactions (right column). The case of two interacting atoms is considered. In state A both atoms have charges of 0.5 and $\sigma = 0.3$ nm, $\epsilon = 0.5$ kJ/mol. In state B all the non-bonded interactions are set to zero. The parameter $\lambda$ is set to 0.5 and electrostatic interaction cutoff is 1 nm.

The parameter $\alpha_{LJ}$ is a unitless scaling factor in the range $[0, 1)$. It scales the position of the point from which the van der Waals force will be linearized. The linearization of the force is allowed in the range $[0, F^{LJ}_{min})$, where setting $\alpha_{LJ} = 0$ results in a standard hard-core van der Waals interaction. Setting $\alpha_{LJ}$ closer to 1 brings the force linearization point towards the minimum in the Lennard-Jones force curve ($F^{LJ}_{min}$). This construct allows retaining the repulsion between two particles with non-zero C12 parameter at any $\lambda$ value.

The parameter $\alpha_Q$ has a unit of $\frac{nm}{e^2}$ and is defined in the range $[0, \inf)$. It scales the position of the point from which the Coulombic force will be linearized. Even though in theory $\alpha_Q$ can be set to an arbitrarily large value, algorithmically the linearization point for the force is bound in the range $[0, F^Q_{rcoul})$, where setting $\alpha_Q = 0$ results in a standard hard-core Coulombic interaction. Setting $\alpha_Q$ to a larger value softens the Coulombic force.

In all the notations below, for simplicity, the distance between two atoms $i$ and $j$ is noted as $r$, i.e. $r = r_{ij}$.

### Forces: van der Waals interactions

$$\mathbf{F}^{LJ}_{ij}(\mathbf{r}) = \begin{cases} (\frac{12C^{(12)}_{ij}}{r^{13}} - \frac{6C^{(6)}_{ij}}{r^7})\frac{\mathbf{r}}{r}, & \text{if } r \geq r_{scLJ} \\ \frac{d\mathbf{F}^{LJ}_{ij}}{dr}_{r=r_{scLJ}} r + \mathbf{F}^{LJ}_{ij}(r_{scLJ}), & \text{if } r < r_{scLJ} \end{cases} \tag{5.244}$$

where the switching point between the soft and hard-core Lennard-Jones forces $r_{scLJ} = \alpha_{LJ}(\frac{26}{7}\sigma^6\lambda)^{\frac{1}{6}}$ for state A, and $r_{scLJ} = \alpha_{LJ}(\frac{26}{7}\sigma^6(1-\lambda))^{\frac{1}{6}}$ for state B. In analogy to the Beutler *et al.* soft core version, $\sigma$ is the radius of the interaction, which is $(C_{12}/C_6)^{1/6}$ or an input parameter (set with `sc-sigma-LJ-gapsys`) when C6 or C12 is zero. The default value for this parameter is `sc-sigma-LJ-gapsys=0.3`.

Explicit expression:

$$\mathbf{F}_{LJ}(\mathbf{r}) = \begin{cases} \left(\frac{12C^{(12)}}{r^{13}} - \frac{6C^{(6)}}{r^7}\right)\frac{\mathbf{r}}{r}, & \text{if } r \geq r_{scLJ} \\ \left(-\frac{156C^{(12)}}{r^{14}_{scLJ}} + \frac{42C^{(6)}}{r^8_{scLJ}}\right)\mathbf{r} + \frac{168C^{(12)}}{r^{13}_{scLJ}} - \frac{48C^{(6)}}{r^7_{scLJ}}, & \text{if } r < r_{scLJ} \end{cases} \tag{5.245}$$

### Forces: Coulomb interactions

$$\mathbf{F}^{Q}_{ij}(\mathbf{r}) = \begin{cases} \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon_r r^2}\frac{\mathbf{r}}{r}, & \text{if } r \geq r_{scQ} < r_{cutoffQ} \\ \frac{d\mathbf{F}^{Q}_{ij}}{dr}_{r=r_{scQ}} r + \mathbf{F}^{Q}_{ij}(r_{scQ}), & \text{if } r < r_{scQ} < r_{cutoffQ} \\ \frac{d\mathbf{F}^{Q}_{ij}}{dr}_{r=r_{cutoffQ}} r + \mathbf{F}^{Q}_{ij}(r_{cutoffQ}), & \text{if } r < r^{scQ} \geq r_{cutoffQ} \end{cases} \tag{5.246}$$

where the switching point $r^{sc}$ between the soft and hard-core electrostatic forces is $r_{scQ} = \alpha_Q(1 + |q_i q_j|)\lambda^{\frac{1}{6}}$ for state A, and $r_{scQ} = \alpha_Q(1+|q_i q_j|)(1-\lambda)^{\frac{1}{6}}$ for state B. The $\lambda$ dependence of the linearization point for both van der Waals and Coulombic interactions is of the same power $1/6$.

Explicit expression:

$$\mathbf{F}_{Q}(\mathbf{r}) = \begin{cases} \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon_r r^2}\frac{\mathbf{r}}{r}, & \text{if } r \geq r_{scQ} < r_{cutoffQ} \\ \frac{1}{4\pi\varepsilon_0\varepsilon_r}\left(-\frac{2q_i q_j}{r^3_{sc}}\mathbf{r} + \frac{3q_i q_j}{r^2_{sc}}\right), & \text{if } r < r_{scQ} < r_{cutoffQ} \\ \frac{1}{4\pi\varepsilon_0\varepsilon_r}\left(-\frac{2q_i q_j}{r^3_{cutoffQ}}\mathbf{r} + \frac{3q_i q_j}{r^2_{cutoffQ}}\right), & \text{if } r < r_{scQ} \geq r_{cutoffQ} \end{cases} \tag{5.247}$$

### Energies: van der Waals interactions

Explicit definition of energies:

$$V_{LJ}(r) = \begin{cases} \frac{C^{(12)}}{r^{12}} - \frac{C^{(6)}}{r^6}, & \text{if } r \geq r_{scLJ} \\ \left(\frac{78C^{(12)}}{r_{scLJ}^{14}} - \frac{21C^{(6)}}{r_{scLJ}^{8}}\right) r^2 - \left(\frac{168C^{(12)}}{r_{scLJ}^{13}} - \frac{48C^{(6)}}{r_{scLJ}^{7}}\right) r + \frac{91C^{(12)}}{r_{scLJ}^{12}} - \frac{28C^{(6)}}{r_{scLJ}^{6}}, & \text{if } r < r_{scLJ} \end{cases} \tag{5.248}$$

### Energies: Coulomb interactions

$$V_Q(r) = \begin{cases} \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon_r r}, & \text{if } r \geq r_{scQ} < r_{cutoffQ} \\ \frac{q_i q_j}{r_{scQ}^3} r^2 - \frac{3q_i q_j}{r_{scQ}^2} r + \frac{3q_i q_j}{r_{scQ}}, & \text{if } r < r_{scQ} < r_{cutoffQ} \\ \frac{q_i q_j}{r_{cutoffQ}^3} r^2 - \frac{3q_i q_j}{r_{cutoffQ}^2} r + \frac{3q_i q_j}{r_{cutoffQ}}, & \text{if } r < r_{scQ} \geq r_{cutoffQ} \end{cases} \tag{5.249}$$

### $\partial H/\partial\lambda$: van der Waals interactions

Here we provide the explicit expressions of $\partial H/\partial\lambda$ for Lennard-Jones potential, when $r < r_{scLJ}$. For simplicity, in the expression below we use the notation $r_{scLJ_A} = r_{scA}$ and $r_{scLJ_B} = r_{scB}$.

$$\begin{aligned} \frac{\partial H}{\partial\lambda} &= V_{LJ}^B(r) - V_{LJ}^A(r) + (1-\lambda)\frac{\partial V_{LJ}^A(r)}{\partial\lambda} + \lambda\frac{\partial V_{LJ}^B(r)}{\partial\lambda} \\ &= \left(\frac{78C_B^{(12)}}{r_{scB}^{14}} - \frac{21C_B^{(6)}}{r_{scB}^{8}}\right) r^2 - \left(\frac{168C_B^{(12)}}{r_{scB}^{13}} - \frac{48C_B^{(6)}}{r_{scB}^{7}}\right) r + \frac{91C_B^{(12)}}{r_{scB}^{12}} - \frac{28C_B^{(6)}}{r_{scB}^{6}} \\ &- \left[\left(\frac{78C_A^{(12)}}{r_{scA}^{14}} - \frac{21C_A^{(6)}}{r_{scA}^{8}}\right) r^2 - \left(\frac{168C_A^{(12)}}{r_{scA}^{13}} - \frac{48C_A^{(6)}}{r_{scA}^{7}}\right) r + \frac{91C_A^{(12)}}{r_{scA}^{12}} - \frac{28C_A^{(6)}}{r_{scA}^{6}}\right] \\ &+ \frac{14(\lambda-1)}{\lambda}\left[\left(\frac{13C_A^{(12)}}{r_{scA}^{14}} - \frac{2C_A^{(6)}}{r_{scA}^{8}}\right) r^2 - \left(\frac{26C_A^{(12)}}{r_{scA}^{13}} - \frac{4C_A^{(6)}}{r_{scA}^{7}}\right) r + \frac{13C_A^{(12)}}{r_{scA}^{12}} - \frac{2C_A^{(6)}}{r_{scA}^{6}}\right] \\ &+ \frac{14\lambda}{1-\lambda}\left[\left(\frac{13C_B^{(12)}}{r_{scB}^{14}} - \frac{2C_B^{(6)}}{r_{scB}^{8}}\right) r^2 - \left(\frac{26C_B^{(12)}}{r_{scB}^{13}} - \frac{4C_B^{(6)}}{r_{scB}^{7}}\right) r + \frac{13C_B^{(12)}}{r_{scB}^{12}} - \frac{2C_B^{(6)}}{r_{scB}^{6}}\right] \end{aligned} \tag{5.250}$$

$\partial H/\partial\lambda$ for Lennard-Jones potential, when $r \geq r_{scLJ}$ is calculated as a standard hard-core contribution to $\partial H/\partial\lambda$: $\frac{\partial H}{\partial\lambda} = V_{LJ}^B(r) - V_{LJ}^A(r)$.

#### $\partial H/\partial\lambda$ for Coulomb interactions

Here we provide the explicit expressions of $\partial H/\partial\lambda$ for Coulomb potential, when $r < r_{scQ} < r_{cutoffQ}$. For simplicity, in the expression below we use the notation $r_{scQ_A} = r_{scA}$ and $r_{scQ_B} = r_{scB}$.

$$\begin{aligned}\frac{\partial H}{\partial\lambda} &= V_Q^B(r) - V_Q^A(r) + (1-\lambda)\frac{\partial V_Q^A(r)}{\partial\lambda} + \lambda\frac{\partial V_Q^B(r)}{\partial\lambda} \\ &= \frac{q_i^B q_j^B}{r_{scB}^3}r^2 - \frac{3q_i^B q_j^B}{r_{scB}^2}r + \frac{3q_i^B q_j^B}{r_{scB}} \\ &- \left[\frac{q_i^A q_j^A}{r_{scA}^3}r^2 - \frac{3q_i^A q_j^A}{r_{scA}^2}r + \frac{3q_i^A q_j^A}{r_{scA}}\right] \\ &+ \frac{\lambda-1}{2\lambda}\left[\frac{q_i^A q_j^A}{r_{scA}^3}r^2 - \frac{2q_i^A q_j^A}{r_{scA}^2}r + \frac{q_i^A q_j^A}{r_{scA}}\right] \\ &+ \frac{\lambda}{2(1-\lambda)}\left[\frac{q_i^B q_j^B}{r_{scB}^3}r^2 - \frac{2q_i^B q_j^B}{r_{scB}^2}r + \frac{q_i^B q_j^B}{r_{scB}}\right]\end{aligned} \tag{5.251}$$

$\partial H/\partial\lambda$ for Coulomb potential, when $r < r_{scQ} \geq r_{cutoffQ}$ is calculated using the same expression above by setting $r_{scA} = r_{cutoffQ}$ and $r_{scB} = r_{cutoffQ}$.

$\partial H/\partial\lambda$ for Coulomb potential, when $r \geq r_{scQ} < r_{cutoffQ}$ is calculated as a standard hard-core contribution to $\partial H/\partial\lambda$: $\frac{\partial H}{\partial\lambda} = V_Q^B(r) - V_Q^A(r)$.

### 5.5.6 Methods

#### Exclusions and 1-4 Interactions.

Atoms within a molecule that are close by in the chain, *i.e.* atoms that are covalently bonded, or linked by one or two atoms are called *first neighbors, second neighbors* and *third neighbors*, respectively (see Fig. 5.34). Since the interactions of atom **i** with atoms **i+1** and **i+2** are mainly quantum mechanical, they can not be modeled by a Lennard-Jones potential. Instead it is assumed that these interactions are adequately modeled by a harmonic bond term or constraint (**i, i+1**) and a harmonic angle term (**i, i+2**). The first and second neighbors (atoms **i+1** and **i+2**) are therefore *excluded* from the Lennard-Jones interaction list of atom **i**; atoms **i+1** and **i+2** are called *exclusions* of atom **i**.

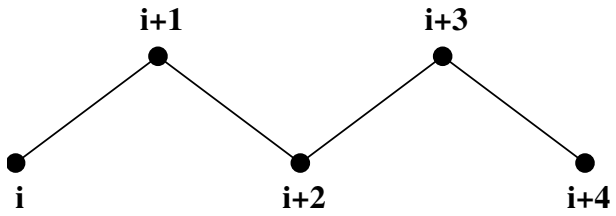



Fig. 5.34: Atoms along an alkane chain.

For third neighbors, the normal Lennard-Jones repulsion is sometimes still too strong, which means that when applied to a molecule, the molecule would deform or break due to the internal strain. This is especially the case for carbon-carbon interactions in a *cis*-conformation (*e.g. cis*-butane). Therefore, for some of these interactions, the Lennard-Jones repulsion has been reduced in the GROMOS force field, which is implemented by keeping a separate list of 1-4 and normal Lennard-Jones parameters. In other force fields, such as OPLS *103* (page 688), the standard Lennard-Jones parameters are reduced

by a factor of two, but in that case also the dispersion ($r^{-6}$) and the Coulomb interaction are scaled. GROMACS can use either of these methods.

### Treatment of Cut-offs

GROMACS is quite flexible in treating cut-offs, which implies there can be quite a number of parameters to set. These parameters are set in the input file for grompp. There are two sort of parameters that affect the cut-off interactions; you can select which type of interaction to use in each case, and which cut-offs should be used in the neighbor searching.

For both Coulomb and van der Waals interactions there are interaction type selectors (termed vdwtype and coulombtype) and two parameters, for a total of six non-bonded interaction parameters. See the User Guide for a complete description of these parameters.

Table 5.9 lists the parameters for the available functional forms and cut-off modifications. See *Pair lists generation* (page 436) for more details about pair-list generation regarding cut-offs.

Table 5.9: Parameters for the different functional forms of the non-bonded interactions.

| Type | | Parameters |
|---|---|---|
| Coulomb | Plain cut-off | $r_c$, $\varepsilon_r$ |
| | Reaction field | $r_c$, $\varepsilon_{rf}$ |
| | Shift function | $r_1$, $r_c$, $\varepsilon_r$ |
| | Switch function | $r_1$, $r_c$, $\varepsilon_r$ |
| VdW | Plain cut-off | $r_c$ |
| | Shift function | $r_1$, $r_c$ |
| | Switch function | $r_1$, $r_c$ |

## 5.5.7 Virtual interaction sites

Virtual interaction sites (called dummy atoms in GROMACS versions before 3.3) can be used in GROMACS in a number of ways. We write the position of the virtual site $\mathbf{r}_s$ as a function of the positions of other particles $\mathbf{r}_i$: $\mathbf{r}_s = f(\mathbf{r}_1..\mathbf{r}_n)$. The virtual site, which may carry charge or be involved in other interactions, can now be used in the force calculation. The force acting on the virtual site must be redistributed over the particles with mass in a consistent way. A good way to do this can be found in ref. *104* (page 688). We can write the potential energy as:

$$V = V(\mathbf{r}_s, \mathbf{r}_1, \ldots, \mathbf{r}_n) = V^*(\mathbf{r}_1, \ldots, \mathbf{r}_n) \tag{5.252}$$

The force on the particle $i$ is then:

$$\mathbf{F}_i = -\frac{\partial V^*}{\partial \mathbf{r}_i} = -\frac{\partial V}{\partial \mathbf{r}_i} - \frac{\partial V}{\partial \mathbf{r}_s}\frac{\partial \mathbf{r}_s}{\partial \mathbf{r}_i} = \mathbf{F}_i^{direct} + \mathbf{F}_i \tag{5.253}$$

The first term is the normal force. The second term is the force on particle $i$ due to the virtual site, which can be written in tensor notation:

$$\mathbf{F}_i = \begin{bmatrix} \frac{\partial x_s}{\partial x_i} & \frac{\partial y_s}{\partial x_i} & \frac{\partial z_s}{\partial x_i} \\ \frac{\partial x_s}{\partial y_i} & \frac{\partial y_s}{\partial y_i} & \frac{\partial z_s}{\partial y_i} \\ \frac{\partial x_s}{\partial z_i} & \frac{\partial y_s}{\partial z_i} & \frac{\partial z_s}{\partial z_i} \end{bmatrix} \mathbf{F}_s \tag{5.254}$$

where $\mathbf{F}_s$ is the force on the virtual site and $x_s$, $y_s$ and $z_s$ are the coordinates of the virtual site. In this way, the total force and the total torque are conserved *104* (page 688).

The computation of the virial ((5.26)) is non-trivial when virtual sites are used. Since the virial involves a summation over all the atoms (rather than virtual sites), the forces must be redistributed from the virtual sites to the atoms (using (5.254)) *before* computation of the virial. In some special cases where the forces on the atoms can be written as a linear combination of the forces on the virtual sites (types 2 and 3 below) there is no difference between computing the virial before and after the redistribution of forces. However, in the general case redistribution should be done first.

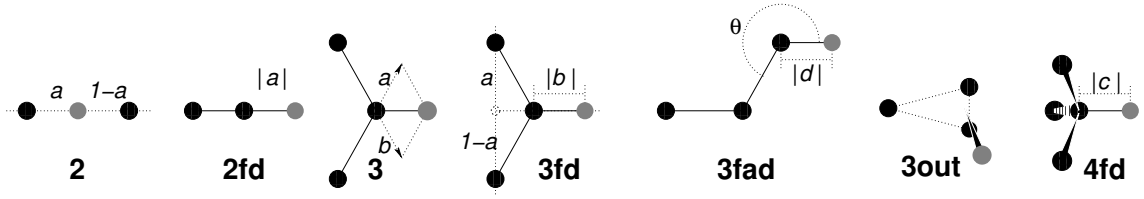



Fig. 5.35: The seven different types of virtual site construction. The constructing atoms are shown as black circles, the virtual sites in gray.

There are six ways to construct virtual sites from surrounding atoms in GROMACS, which we classify by the number of constructing atoms. **Note** that all site types mentioned can be constructed from types 3fd (normalized, in-plane) and 3out (non-normalized, out of plane). However, the amount of computation involved increases sharply along this list, so we strongly recommended using the first adequate virtual site type that will be sufficient for a certain purpose. Fig. 5.35 depicts 6 of the available virtual site constructions. The conceptually simplest construction types are linear combinations:

$$\mathbf{r}_s = \sum_{i=1}^{N} w_i \, \mathbf{r}_i \tag{5.255}$$

The force is then redistributed using the same weights:

$$\mathbf{F}_i = w_i \, \mathbf{F}_s \tag{5.256}$$

The types of virtual sites supported in GROMACS are given in the list below. Constructing atoms in virtual sites can be virtual sites themselves, but only if they are higher in the list, i.e. virtual sites can be constructed from "particles" that are simpler virtual sites. The virtual site velocities are reported, but not used in the integration of the virtual site positions.

### On top of an atom

- This allows giving an atom multiple atom types and with that also assigned multiple, different bonded interactions. This can especially be of use in free-energy calculations.
- The coordinates of the virtual site equal that of the constructing atom:

$$\mathbf{r}_s \; = \; \mathbf{r}_i \tag{5.257}$$

- The force is moved to the constructing atom:

$$\mathbf{F}_i \; = \; \mathbf{F}_s \tag{5.258}$$

- The velocity of the virtual site equals that of the constructing atom:

$$\mathbf{v}_s \; = \; \mathbf{v}_i \tag{5.259}$$

### As a linear combination of two atoms (Fig. 5.35 2)

- The weights are calculated as

$$w_i = 1 - a \, , \;\; w_j = a \tag{5.260}$$

- In this case the virtual site is on the line through atoms $i$ and $j$.
- The velocity of the virtual site is a linear combination of the velocities of the constructing atoms

### On the line through two atoms, with a fixed distance (Fig. 5.35 2fd)

- The position is calculated as:

$$\mathbf{r}_s \;=\; \mathbf{r}_i + a\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} \tag{5.261}$$

- In this case the virtual site is on the line through the other two particles at a distance of $|a|$ from $i$. The force on particles $i$ and $j$ due to the force on the virtual site can be computed as:

$$\begin{array}{rcl} \mathbf{F}_i &=& \mathbf{F}_s - \gamma(\mathbf{F}_{is} - \mathbf{p}) \\ \mathbf{F}_j &=& \gamma(\mathbf{F}_s - \mathbf{p}) \end{array} \quad \text{where} \quad \begin{array}{c} \gamma = \dfrac{a}{|\mathbf{r}_{ij}|} \\ \mathbf{p} = \dfrac{\mathbf{r}_{is} \cdot \mathbf{F}_s}{\mathbf{r}_{is} \cdot \mathbf{r}_{is}} \mathbf{r}_{is} \end{array} \tag{5.262}$$

- The velocity is calculated as:

$$\mathbf{v}_s = \mathbf{v}_i + \frac{a}{|\mathbf{r}_{ij}|}\left(\mathbf{v}_{ij} - \mathbf{r}_{ij}\frac{\mathbf{v}_{ij} \cdot \mathbf{r}_{ij}}{|\mathbf{r}_{ij}|^2}\right) \tag{5.263}$$

### As a linear combination of three atoms (Fig. 5.35 3)

- The weights are calculated as:

$$w_i = 1 - a - b \, , \;\; w_j = a \, , \;\; w_k = b \tag{5.264}$$

- In this case the virtual site is in the plane of the other three particles.

### In the plane of three atoms, with a fixed distance (Fig. 5.35 3fd)

- The position is calculated as:

$$\mathbf{r}_s \;=\; \mathbf{r}_i + b\frac{\mathbf{r}_{ijk}}{|\mathbf{r}_{ijk}|} \;\text{ where }\; \mathbf{r}_{ijk} \;=\; \mathbf{r}_{ij} + a\mathbf{r}_{jk} \tag{5.265}$$

- In this case the virtual site is in the plane of the other three particles at a distance of $|b|$ from $i$. The force on particles $i$, $j$ and $k$ due to the force on the virtual site can be computed as:

$$\begin{array}{rcl} \mathbf{F}_i &=& \mathbf{F}_s - \gamma(\mathbf{F}_{is} - \mathbf{p}) \\ \mathbf{F}_j &=& (1-a)\gamma(\mathbf{F}_s - \mathbf{p}) \\ \mathbf{F}_k &=& a\gamma(\mathbf{F}_s - \mathbf{p}) \end{array} \quad \text{where} \quad \begin{array}{c} \gamma = \dfrac{b}{|\mathbf{r}_{ij} + a\mathbf{r}_{jk}|} \\ \mathbf{p} = \dfrac{\mathbf{r}_{is} \cdot \mathbf{F}_s}{\mathbf{r}_{is} \cdot \mathbf{r}_{is}} \mathbf{r}_{is} \end{array} \tag{5.266}$$

- The velocity is calculated as:

$$\mathbf{v}_s \;=\; \mathbf{v}_i + \frac{b}{|\mathbf{r}_{ijk}|}\left(\dot{\mathbf{r}}_{ijk} - \mathbf{r}_{ijk}\frac{\dot{\mathbf{r}}_{ijk} \cdot \mathbf{r}_{ijk}}{|\mathbf{r}_{ijk}|^2}\right) \tag{5.267}$$

### In the plane of three atoms, with a fixed angle and distance (Fig. 5.35 3fad)

- The position is calculated as:

$$\mathbf{r}_s \;=\; \mathbf{r}_i + d\cos\theta\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} + d\sin\theta\frac{\mathbf{r}_\perp}{|\mathbf{r}_\perp|} \;\text{ where }\; \mathbf{r}_\perp \;=\; \mathbf{r}_{jk} - \frac{\mathbf{r}_{ij}\cdot\mathbf{r}_{jk}}{\mathbf{r}_{ij}\cdot\mathbf{r}_{ij}}\mathbf{r}_{ij} \tag{5.268}$$

- In this case the virtual site is in the plane of the other three particles at a distance of $|d|$ from $i$ at an angle of $\alpha$ with $\mathbf{r}_{ij}$. Atom $k$ defines the plane and the direction of the angle. **Note** that in this case $b$ and $\alpha$ must be specified, instead of $a$ and $b$ (see also sec. *Virtual sites* (page 538)). The force on particles $i$, $j$ and $k$ due to the force on the virtual site can be computed as (with $\mathbf{r}_\perp$ as defined in (5.268)):

$$\begin{aligned}
\mathbf{F}_i &= \mathbf{F}_s - \frac{d\cos\theta}{|\mathbf{r}_{ij}|}\mathbf{F}_1 + \frac{d\sin\theta}{|\mathbf{r}_\perp|}\left(\frac{\mathbf{r}_{ij}\cdot\mathbf{r}_{jk}}{\mathbf{r}_{ij}\cdot\mathbf{r}_{ij}}\mathbf{F}_2 + \mathbf{F}_3\right)\\
\mathbf{F}_j &= \frac{d\cos\theta}{|\mathbf{r}_{ij}|}\mathbf{F}_1 - \frac{d\sin\theta}{|\mathbf{r}_\perp|}\left(\mathbf{F}_2 + \frac{\mathbf{r}_{ij}\cdot\mathbf{r}_{jk}}{\mathbf{r}_{ij}\cdot\mathbf{r}_{ij}}\mathbf{F}_2 + \mathbf{F}_3\right)\\
\mathbf{F}_k &= \frac{d\sin\theta}{|\mathbf{r}_\perp|}\mathbf{F}_2
\end{aligned} \tag{5.269}$$

$$\text{where }\; \mathbf{F}_1 = \mathbf{F}_s - \frac{\mathbf{r}_{ij}\cdot\mathbf{F}_s}{\mathbf{r}_{ij}\cdot\mathbf{r}_{ij}}\mathbf{r}_{ij}\;,\;\; \mathbf{F}_2 = \mathbf{F}_1 - \frac{\mathbf{r}_\perp\cdot\mathbf{F}_s}{\mathbf{r}_\perp\cdot\mathbf{r}_\perp}\mathbf{r}_\perp \text{ and } \mathbf{F}_3 = \frac{\mathbf{r}_{ij}\cdot\mathbf{F}_s}{\mathbf{r}_{ij}\cdot\mathbf{r}_{ij}}\mathbf{r}_\perp$$

- The velocity is calculated as:

$$\mathbf{v}_s = \mathbf{v}_i + d\cos\theta\;\frac{\delta}{\delta t}\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} + d\sin\theta\;\frac{\delta}{\delta t}\frac{\mathbf{r}_\perp}{|\mathbf{r}_\perp|}$$

where

$$\begin{aligned}
\frac{\delta}{\delta t}\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} &= \frac{1}{|\mathbf{r}_{ij}|}\left(\mathbf{v}_{ij} - \mathbf{r}_{ij}\frac{\mathbf{v}_{ij}\cdot\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|^2}\right)\\
\frac{\delta}{\delta t}\frac{\mathbf{r}_\perp}{|\mathbf{r}_\perp|} &= \frac{1}{|\mathbf{r}_\perp|}\left(\dot{\mathbf{r}}_\perp - \mathbf{r}_\perp\frac{\dot{\mathbf{r}}_\perp\cdot\mathbf{r}_\perp}{|\mathbf{r}_\perp|^2}\right)\\
\dot{\mathbf{r}}_\perp &= \mathbf{v}_{jk} - \mathbf{r}_{ij}\frac{|\mathbf{r}_{ij}|^2(\mathbf{v}_{ij}\cdot\mathbf{r}_{jk} + \mathbf{r}_{ij}\cdot\mathbf{v}_{jk}) - (\mathbf{r}_{ij}\cdot\mathbf{r}_{jk})(2\mathbf{r}_{ij}\cdot\mathbf{v}_{ij})}{|\mathbf{r}_{ij}|^4} - \frac{\mathbf{r}_{ij}\cdot\mathbf{r}_{jk}}{|\mathbf{r}_{ij}|^2}\;\mathbf{v}_{ij}
\end{aligned} \tag{5.270}$$

### As a non-linear combination of three atoms, out of plane (Fig. 5.35 3out)

- The position is calculated as:

$$\mathbf{r}_s \;=\; \mathbf{r}_i + a\mathbf{r}_{ij} + b\mathbf{r}_{ik} + c(\mathbf{r}_{ij}\times\mathbf{r}_{ik}) \tag{5.271}$$

- This enables the construction of virtual sites out of the plane of the other atoms. The force on particles $i, j$ and $k$ due to the force on the virtual site can be computed as:

$$\begin{aligned}
\mathbf{F}_j &= \begin{bmatrix} a & -c\,z_{ik} & c\,y_{ik} \\ c\,z_{ik} & a & -c\,x_{ik} \\ -c\,y_{ik} & c\,x_{ik} & a \end{bmatrix}\mathbf{F}_s\\
\mathbf{F}_k &= \begin{bmatrix} b & c\,z_{ij} & -c\,y_{ij} \\ -c\,z_{ij} & b & c\,x_{ij} \\ c\,y_{ij} & -c\,x_{ij} & b \end{bmatrix}\mathbf{F}_s\\
\mathbf{F}_i &= \mathbf{F}_s - \mathbf{F}_j - \mathbf{F}_k
\end{aligned} \tag{5.272}$$

- The velocity is calculated as:

$$\mathbf{v}_s \;=\; \mathbf{v}_i + \frac{c}{|\mathbf{r}_m|}\left(\dot{\mathbf{r}}_m - \mathbf{r}_m\frac{\dot{\mathbf{r}}_m\cdot\mathbf{r}_m}{|\mathbf{r}_m|^2}\right) \tag{5.273}$$

### From four atoms, with a fixed distance, see separate Fig. 5.36

- This construction is a bit complex, in particular since the previous type (4fd) could be unstable which forced us to introduce a more elaborate construction:

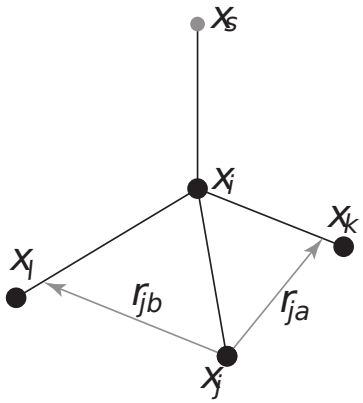



Fig. 5.36: The new 4fdn virtual site construction, which is stable even when all constructing atoms are in the same plane.

- The position is calculated as

$$\begin{aligned}
\mathbf{r}_{ja} &= a\,\mathbf{r}_{ik} - \mathbf{r}_{ij} = a\,(\mathbf{x}_k - \mathbf{x}_i) - (\mathbf{x}_j - \mathbf{x}_i)\\
\mathbf{r}_{jb} &= b\,\mathbf{r}_{il} - \mathbf{r}_{ij} = b\,(\mathbf{x}_l - \mathbf{x}_i) - (\mathbf{x}_j - \mathbf{x}_i)\\
\mathbf{r}_m &= \mathbf{r}_{ja}\times\mathbf{r}_{jb}\\
\mathbf{r}_s &= \mathbf{r}_i + c\frac{\mathbf{r}_m}{|\mathbf{r}_m|}
\end{aligned}$$

- The velocity is calculated as:

$$\mathbf{v}_s = \mathbf{v}_i + \frac{c}{|\mathbf{r}_m|}\left(\dot{\mathbf{r}}_m - \mathbf{r}_m\frac{\dot{\mathbf{r}}_m\cdot\mathbf{r}_m}{|\mathbf{r}_m|^2}\right) \tag{5.274}$$

where

$$\dot{\mathbf{r}}_m = \dot{\mathbf{r}}_{ja}\times\mathbf{r}_{jb} + \mathbf{r}_{ja}\times\dot{\mathbf{r}}_{jb}$$

- In this case the virtual site is at a distance of $|c|$ from $i$, while $a$ and $b$ are parameters. **Note** that the vectors $\mathbf{r}_{ik}$ and $\mathbf{r}_{ij}$ are not normalized to save floating-point operations. The force on particles $i$, $j$, $k$ and $l$ due to the force on the virtual site are computed through chain rule derivatives of the construction expression. This is exact and conserves energy, but it does lead to relatively lengthy expressions that we do not include here (over 200 floating-point operations). The interested reader can look at the source code in `vsite.c`. Fortunately, this vsite type is normally only used for chiral centers such as $C_\alpha$ atoms in proteins.

  The new 4fdn construct is identified with a 'type' value of 2 in the topology. The earlier 4fd type is still supported internally ('type' value 1), but it should not be used for new simulations. All current GROMACS tools will automatically generate type 4fdn instead.

#### A linear combination of $N$ atoms with relative weights $a_i$

- The weight for atom $i$ is:

$$w_i = a_i \left( \sum_{j=1}^{N} a_j \right)^{-1} \tag{5.275}$$

- There are three options for setting the weights:
- center of geometry: equal weights
- center of mass: $a_i$ is the mass of atom $i$; when in free-energy simulations the mass of the atom is changed, only the mass of the A-state is used for the weight
- center of weights: $a_i$ is defined by the user

### 5.5.8 Long Range Electrostatics

#### Ewald summation

The total electrostatic energy of $N$ particles and their periodic images is given by

$$V = \frac{f}{2} \sum_{n_x} \sum_{n_y} \sum_{n_z *} \sum_{i}^{N} \sum_{j}^{N} \frac{q_i q_j}{\mathbf{r}_{ij,\mathbf{n}}}. \tag{5.276}$$

$(n_x, n_y, n_z) = \mathbf{n}$ is the box index vector, and the star indicates that terms with $i = j$ should be omitted when $(n_x, n_y, n_z) = (0, 0, 0)$. The distance $\mathbf{r}_{ij,\mathbf{n}}$ is the real distance between the charges and not the minimum-image. This sum is conditionally convergent, but very slow.

Ewald summation was first introduced as a method to calculate long-range interactions of the periodic images in crystals *105* (page 688). The idea is to convert the single slowly-converging sum (5.276) into two quickly-converging terms and a constant term:

$$\begin{aligned} V &= V_{\text{dir}} + V_{\text{rec}} + V_0 \\ V_{\text{dir}} &= \frac{f}{2} \sum_{i,j}^{N} \sum_{n_x} \sum_{n_y} \sum_{n_z *} q_i q_j \frac{\operatorname{erfc}(\beta r_{ij,\mathbf{n}})}{r_{ij,\mathbf{n}}} \\ V_{\text{rec}} &= \frac{f}{2\pi V} \sum_{i,j}^{N} q_i q_j \sum_{m_x} \sum_{m_y} \sum_{m_z *} \frac{\exp\left(-(\pi \mathbf{m}/\beta)^2 + 2\pi i \mathbf{m} \cdot (\mathbf{r}_i - \mathbf{r}_j)\right)}{\mathbf{m}^2} \\ V_0 &= -\frac{f\beta}{\sqrt{\pi}} \sum_{i}^{N} q_i^2, \end{aligned} \tag{5.277}$$

where $\beta$ is a parameter that determines the relative weight of the direct and reciprocal sums and $\mathbf{m} = (m_x, m_y, m_z)$. In this way we can use a short cut-off (of the order of 1 nm) in the direct space sum and a short cut-off in the reciprocal space sum (*e.g.* 10 wave vectors in each direction). Unfortunately, the computational cost of the reciprocal part of the sum increases as $N^2$ (or $N^{3/2}$ with a slightly better algorithm) and it is therefore not realistic for use in large systems.

There are two sources of error in the Ewald method. One is due to the truncation of the Fourier sum and another is due to the truncation of the direct space interaction at the cut-off. If one would use the

potential as given by (5.276), energy conservation would be violated due to the jump in energy at a direct space cut-off distance. To maintain energy conservation, GROMACS by default shifts the pair potential in $V_{\mathrm{dir}}$ by a constant such that the potential is zero at the cut-off. This shift is also applied with PME.

In the log file and energy file that *gmx mdrun* (page 258) outputs the EWALD/PME energy contributions are reported in two terms. The term `Coulomb (SR)` contains the $V_{\mathrm{dir}}$ with the constant shift applied minus the reciprocal contribution for excluded atom pairs, as well as the charge correction $V_0$. The term `Coul. recip.` contains the reciprocal sum $V_{\mathrm{rec}}$, including contributions of excluded atoms pairs, plus an optional surface correction term that depends on the system dipole.

### Using Ewald

Do not use Ewald unless you are absolutely sure this is what you want - for almost all cases the PME method below will perform much better. If you still want to employ classical Ewald summation enter this in your *mdp* (page 579) file, if the side of your box is about 3 nm:

```
coulombtype      = Ewald
rvdw             = 0.9
rlist            = 0.9
rcoulomb         = 0.9
fourierspacing   = 0.6
ewald-rtol       = 1e-5
```

The ratio of the box dimensions and the fourierspacing parameter determines the highest magnitude of wave vectors $m_x, m_y, m_z$ to use in each direction. With a 3-nm cubic box this example would use 11 wave vectors (from $-5$ to 5) in each direction. The `ewald-rtol` parameter is the relative strength of the electrostatic interaction at the cut-off. Decreasing this gives you a more accurate direct sum, but a less accurate reciprocal sum.

### PME

Particle-mesh Ewald is a method proposed by Tom Darden *14* (page 683) to improve the performance of the reciprocal sum. Instead of directly summing wave vectors, the charges are assigned to a grid using interpolation. The implementation in GROMACS uses cardinal B-spline interpolation *15* (page 683), which is referred to as smooth PME (SPME). The grid is then Fourier transformed with a 3D FFT algorithm and the reciprocal energy term obtained by a single sum over the grid in k-space.

The potential at the grid points is calculated by inverse transformation, and by using the interpolation factors we get the forces on each atom.

The PME algorithm scales as $N \log(N)$, and is substantially faster than ordinary Ewald summation on medium to large systems. On very small systems it might still be better to use Ewald to avoid the overhead in setting up grids and transforms. For the parallelization of PME see the section on MPMD PME (*Multiple-Program, Multiple-Data PME parallelization* (page 478)).

With the Verlet cut-off scheme, the PME direct space potential is shifted by a constant such that the potential is zero at the cut-off. This shift is small and since the net system charge is close to zero, the total shift is very small, unlike in the case of the Lennard-Jones potential where all shifts add up. We apply the shift anyhow, such that the potential is the exact integral of the force.

### Using PME

As an example for using Particle-mesh Ewald summation in GROMACS, specify the following lines in your *mdp* (page 579) file:

```
coulombtype      = PME
rvdw             = 0.9
rlist            = 0.9
rcoulomb         = 0.9
fourierspacing   = 0.12
pme-order        = 4
ewald-rtol       = 1e-5
```

In this case the `fourierspacing` parameter determines the maximum spacing for the FFT grid (i.e. minimum number of grid points), and `pme-order` controls the interpolation order. Using fourth-order (cubic) interpolation and this spacing should give electrostatic energies accurate to about $5 \cdot 10^{-3}$. Since the Lennard-Jones energies are not this accurate it might even be possible to increase this spacing slightly.

Pressure scaling works with PME, but be aware of the fact that anisotropic scaling can introduce artificial ordering in some systems.

### P3M-AD

The Particle-Particle Particle-Mesh methods of Hockney & Eastwood can also be applied in GROMACS for the treatment of long range electrostatic interactions *106* (page 688). Although the P3M method was the first efficient long-range electrostatics method for molecular simulation, the smooth PME (SPME) method has largely replaced P3M as the method of choice in atomistic simulations. One performance disadvantage of the original P3M method was that it required 3 3D-FFT back transforms to obtain the forces on the particles. But this is not required for P3M and the forces can be derived through analytical differentiation of the potential, as done in PME. The resulting method is termed P3M-AD. The only remaining difference between P3M-AD and PME is the optimization of the lattice Green influence function for error minimization that P3M uses. However, in 2012 it has been shown that the SPME influence function can be modified to obtain P3M *107* (page 688). This means that the advantage of error minimization in P3M-AD can be used at the same computational cost and with the same code as PME, just by adding a few lines to modify the influence function. However, at optimal parameter setting the effect of error minimization in P3M-AD is less than 10%. P3M-AD does show large accuracy gains with interlaced (also known as staggered) grids, but that is not supported in GROMACS (yet).

P3M is used in GROMACS with exactly the same options as used with PME by selecting the electrostatics type:

```
coulombtype      = P3M-AD
```

#### Optimizing Fourier transforms and PME calculations

It is recommended to optimize the parameters for calculation of electrostatic interaction such as PME grid dimensions and cut-off radii. This is particularly relevant to do before launching long production runs.

*gmx mdrun* (page 258) will automatically do a lot of PME optimization, and GROMACS also includes a special tool, *gmx tune_pme* (page 340), which automates the process of selecting the optimal number of PME-only ranks.

#### Boundary conditions

Ewald and PME use so-called tin-foil boundary conditions at infinity. This means that it is assumed that a perfectly conducting sphere surrounds the system as the limit to infinite periodic images is taken. This effectively means that there is no penalty to creating a net dipole in the unit cell. This usually does not create issues, unless an electric field is used, as this interacts favourably with the net dipole. GROMACS supports the option to set the dielectric permitivity of the boundary at infinity. This adds a cost to creating a net dipole.

In certain cases one might want to simulate with 2D periodicity, for example when interfaces are involved. Ewald summation and PME are 3D periodic in nature, so strictly speaking not suitable for 2D periodicity. However, GROMACS supports a simple correction which removes the net dipole-dipole interaction along the z-dimension *200* (page 693). It is recommended to surround the system by double its size of vacuum space along z. This is done by default when selecting this correction.

### 5.5.9 Long Range Van der Waals interactions

#### Dispersion correction

In this section, we derive long-range corrections due to the use of a cut-off for Lennard-Jones or Buckingham interactions. We assume that the cut-off is so long that the repulsion term can safely be neglected, and therefore only the dispersion term is taken into account. Due to the nature of the dispersion interaction (we are truncating a potential proportional to $-r^{-6}$), energy and pressure corrections are both negative. While the energy correction is usually small, it may be important for free energy calculations where differences between two different Hamiltonians are considered. In contrast, the pressure correction is very large and can not be neglected under any circumstances where a correct pressure is required, especially for any NPT simulations. Although it is, in principle, possible to parameterize a force field such that the pressure is close to the desired experimental value without correction, such a method makes the parameterization dependent on the cut-off and is therefore undesirable.

#### Energy

The long-range contribution of the dispersion interaction to the virial can be derived analytically, if we assume a homogeneous system beyond the cut-off distance $r_c$. The dispersion energy between two particles is written as:

$$V(r_{ij}) \;=\; -C_6\, {r_{ij}}^{-6} \tag{5.278}$$

and the corresponding force is:

$$\mathbf{F}_{ij} \;=\; -6\, C_6\, r_{ij}^{-8} \mathbf{r}_{ij} \tag{5.279}$$

In a periodic system it is not easy to calculate the full potentials, so usually a cut-off is applied, which can be abrupt or smooth. We will call the potential and force with cut-off $V_c$ and $\mathbf{F}_c$. The long-range contribution to the dispersion energy in a system with $N$ particles and particle density $\rho = N/V$ is:

$$V_{lr} \;=\; \frac{1}{2} N \rho \int_0^\infty 4\pi r^2 g(r) \left(V(r) - V_c(r)\right) \mathrm{d}r \tag{5.280}$$

We will integrate this for the shift function, which is the most general form of van der Waals interaction available in GROMACS. The shift function has a constant difference $S$ from 0 to $r_1$ and is 0 beyond the cut-off distance $r_c$. We can integrate (5.280), assuming that the density in the sphere within $r_1$ is equal to the global density and the radial distribution function $g(r)$ is 1 beyond $r_1$:

$$\begin{aligned} V_{lr} = \;& \frac{1}{2} N \left( \rho \int_0^{r_1} 4\pi r^2 g(r)\, C_6\, S \,\mathrm{d}r + \rho \int_{r_1}^{r_c} 4\pi r^2 \left(V(r) - V_c(r)\right) \mathrm{d}r + \rho \int_{r_c}^{\infty} 4\pi r^2 V(r) \,\mathrm{d}r \right) \\ = \;& \frac{1}{2} N \left( \left( \frac{4}{3}\pi\rho r_1^3 - 1 \right) C_6\, S + \rho \int_{r_1}^{r_c} 4\pi r^2 \left(V(r) - V_c(r)\right) \mathrm{d}r - \frac{4}{3}\pi N \rho\, C_6\, r_c^{-3} \right) \end{aligned} \tag{5.281}$$

where the term $-1$ corrects for the self-interaction. For a plain cut-off we only need to assume that $g(r)$ is 1 beyond $r_c$ and the correction reduces to *108* (page 688):

$$V_{lr} = \;\; -\frac{2}{3}\pi N \rho\, C_6\, r_c^{-3} \tag{5.282}$$

If we consider, for example, a box of pure water, simulated with a cut-off of 0.9 nm and a density of 1 g cm$^{-3}$ this correction is $-0.75$ kJ mol$^{-1}$ per molecule.

For a homogeneous mixture we need to define an *average dispersion constant*:

$$\langle C_6 \rangle = \frac{2}{N(N-1)} \sum_i^N \sum_{j>i}^N C_6(i,j) \tag{5.283}$$

In GROMACS, excluded pairs of atoms do not contribute to the average.

In the case of inhomogeneous simulation systems, *e.g.* a system with a lipid interface, the energy correction can be applied if $\langle C_6 \rangle$ for both components is comparable.

### Virial and pressure

The scalar virial of the system due to the dispersion interaction between two particles $i$ and $j$ is given by:

$$\Xi \;=\; -\frac{1}{2} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij} \;=\; 3\, C_6\, r_{ij}^{-6} \tag{5.284}$$

The pressure is given by:

$$P \;=\; \frac{2}{3\,V} \left( E_{kin} - \Xi \right) \tag{5.285}$$

The long-range correction to the virial is given by:

$$\Xi_{lr} \;=\; \frac{1}{2} N \rho \int_0^\infty 4\pi r^2 g(r) (\Xi - \Xi_c) \,\mathrm{d}r \tag{5.286}$$

We can again integrate the long-range contribution to the virial assuming $g(r)$ is 1 beyond $r_1$:

$$\begin{aligned}\Xi_{lr} = & \frac{1}{2}N\rho\left(\int_{r_1}^{r_c} 4\pi r^2(\Xi-\Xi_c)\,\mathrm{d}r + \int_{r_c}^{\infty} 4\pi r^2 3\,C_6\,{r_{ij}}^{-6}\,\mathrm{d}r\right)\\ = & \frac{1}{2}N\rho\left(\int_{r_1}^{r_c} 4\pi r^2(\Xi-\Xi_c)\,\mathrm{d}r + 4\pi C_6\,r_c^{-3}\right)\end{aligned}$$

For a plain cut-off the correction to the pressure is *108* (page 688):

$$P_{lr} = -\frac{4}{3}\pi C_6\,\rho^2 r_c^{-3} \tag{5.287}$$

Using the same example of a water box, the correction to the virial is 0.75 kJ mol$^{-1}$ per molecule, the corresponding correction to the pressure for SPC water is approximately $-280$ bar.

For homogeneous mixtures, we can again use the average dispersion constant $\langle C_6\rangle$ ((5.283)):

$$P_{lr} = -\frac{4}{3}\pi\langle C_6\rangle\rho^2 r_c^{-3} \tag{5.288}$$

For inhomogeneous systems, (5.288) can be applied under the same restriction as holds for the energy (see sec. *Energy* (page 528)).

### Lennard-Jones PME

In order to treat systems, using Lennard-Jones potentials, that are non-homogeneous outside of the cut-off distance, we can instead use the Particle-mesh Ewald method as discussed for electrostatics above. In this case the modified Ewald equations become

$$\begin{aligned}V = & V_{\mathrm{dir}} + V_{\mathrm{rec}} + V_0\\ V_{\mathrm{dir}} = & -\frac{1}{2}\sum_{i,j}^{N}\sum_{n_x}\sum_{n_y}\sum_{n_z*}\frac{C_6^{ij}\,g(\beta r_{ij,\mathbf{n}})}{{r_{ij,\mathbf{n}}}^6}\end{aligned} \tag{5.289}$$

$$\begin{aligned}V_{\mathrm{rec}} = & \frac{\pi^{\frac{3}{2}}\beta^3}{2V}\sum_{m_x}\sum_{m_y}\sum_{m_z*} f(\pi|\mathbf{m}|/\beta)\times\sum_{i,j}^{N} C_6^{ij}\exp\left[-2\pi i\mathbf{m}\cdot(\mathbf{r_i}-\mathbf{r_j})\right]\\ V_0 = & -\frac{\beta^6}{12}\sum_{i}^{N} C_6^{ii}\end{aligned} \tag{5.290}$$

where $\mathbf{m} = (m_x, m_y, m_z)$, $\beta$ is the parameter determining the weight between direct and reciprocal space, and $C_6^{ij}$ is the combined dispersion parameter for particle $i$ and $j$. The star indicates that terms with $i = j$ should be omitted when $((n_x, n_y, n_z) = (0, 0, 0))$, and $\mathbf{r}_{ij,\mathbf{n}}$ is the real distance between the particles. Following the derivation by Essmann *15* (page 683), the functions $f$ and $g$ introduced above are defined as

$$\begin{aligned}f(x) = & 1/3\left[(1-2x^2)\exp(-x^2) + 2x^3\sqrt{\pi}\,\mathrm{erfc}(x)\right]\\ g(x) = & \exp(-x^2)(1+x^2+\frac{x^4}{2}).\end{aligned} \tag{5.291}$$

The above methodology works fine as long as the dispersion parameters can be combined geometrically ((5.124)) in the same way as the charges for electrostatics

$$C_{6,\mathrm{geom}}^{ij} = \left(C_6^{ii}\,C_6^{jj}\right)^{1/2} \tag{5.292}$$

For Lorentz-Berthelot combination rules ((5.125)), the reciprocal part of this sum has to be calculated seven times due to the splitting of the dispersion parameter according to

$$C_{6,\mathrm{L-B}}^{ij} = (\sigma_i + \sigma_j)^6 = \sum_{n=0}^{6} P_n \sigma_i^n \sigma_j^{(6-n)}, \tag{5.293}$$

for $P_n$ the Pascal triangle coefficients. This introduces a non-negligible cost to the reciprocal part, requiring seven separate FFTs, and therefore this has been the limiting factor in previous attempts to implement LJ-PME. A solution to this problem is to use geometrical combination rules in order to calculate an approximate interaction parameter for the reciprocal part of the potential, yielding a total interaction of

$$\begin{aligned} V(r < r_c) = & \underbrace{C_6^{\mathrm{dir}} g(\beta r) r^{-6}}_{\text{Direct space}} + \underbrace{C_{6,\mathrm{geom}}^{\mathrm{recip}} [1 - g(\beta r)] r^{-6}}_{\text{Reciprocal space}} \\ = & \; C_{6,\mathrm{geom}}^{\mathrm{recip}} r^{-6} + \left( C_6^{\mathrm{dir}} - C_{6,\mathrm{geom}}^{\mathrm{recip}} \right) g(\beta r) r^{-6} \\ V(r > r_c) = & \underbrace{C_{6,\mathrm{geom}}^{\mathrm{recip}} [1 - g(\beta r)] r^{-6}}_{\text{Reciprocal space}}. \end{aligned}$$

This will preserve a well-defined Hamiltonian and significantly increase the performance of the simulations. The approximation does introduce some errors, but since the difference is located in the interactions calculated in reciprocal space, the effect will be very small compared to the total interaction energy. In a simulation of a lipid bilayer, using a cut-off of 1.0 nm, the relative error in total dispersion energy was below 0.5%. A more thorough discussion of this can be found in *109* (page 688).

In GROMACS we now perform the proper calculation of this interaction by subtracting, from the direct-space interactions, the contribution made by the approximate potential that is used in the reciprocal part

$$V_{\mathrm{dir}} = C_6^{\mathrm{dir}} r^{-6} - C_6^{\mathrm{recip}} [1 - g(\beta r)] r^{-6}. \tag{5.294}$$

This potential will reduce to the expression in (5.289) when $C_6^{\mathrm{dir}} = C_6^{\mathrm{recip}}$, and the total interaction is given by

$$\begin{aligned} V(r < r_c) = & \underbrace{C_6^{\mathrm{dir}} r^{-6} - C_6^{\mathrm{recip}} [1 - g(\beta r)] r^{-6}}_{\text{Direct space}} + \underbrace{C_6^{\mathrm{recip}} [1 - g(\beta r)] r^{-6}}_{\text{Reciprocal space}} \\ = & \; C_6^{\mathrm{dir}} r^{-6} \\ V(r > r_c) = & \; C_6^{\mathrm{recip}} [1 - g(\beta r)] r^{-6}. \end{aligned} \tag{5.295}$$

For the case when $C_6^{\mathrm{dir}} \neq C_6^{\mathrm{recip}}$ this will retain an unmodified LJ force up to the cut-off, and the error is an order of magnitude smaller than in simulations where the direct-space interactions do not account for the approximation used in reciprocal space. When using a VdW interaction modifier of potential-shift, the constant

$$\left( -C_6^{\mathrm{dir}} + C_6^{\mathrm{recip}} [1 - g(\beta r_c)] \right) r_c^{-6} \tag{5.296}$$

is added to (5.295) in order to ensure that the potential is continuous at the cutoff. Note that, in the same way as (5.294), this degenerates into the expected $-C_6 g(\beta r_c) r_c^{-6}$ when $C_6^{\mathrm{dir}} = C_6^{\mathrm{recip}}$. In addition to this, a long-range dispersion correction can be applied to correct for the approximation using a combination rule in reciprocal space. This correction assumes, as for the cut-off LJ potential, a uniform particle distribution. But since the error of the combination rule approximation is very small this long-range correction is not necessary in most cases. Also note that this homogenous correction does not correct the surface tension, which is an inhomogeneous property.

### Using LJ-PME

As an example for using Particle-mesh Ewald summation for Lennard-Jones interactions in GROMACS, specify the following lines in your *mdp* (page 579) file:

```
vdwtype           = PME
rvdw              = 0.9
vdw-modifier      = Potential-Shift
rlist             = 0.9
rcoulomb          = 0.9
fourierspacing    = 0.12
pme-order         = 4
ewald-rtol-lj     = 0.001
lj-pme-comb-rule = geometric
```

The same Fourier grid and interpolation order are used if both LJ-PME and electrostatic PME are active, so the settings for `fourierspacing` and `pme-order` are common to both. `ewald-rtol-lj` controls the splitting between direct and reciprocal space in the same way as `ewald-rtol`. In addition to this, the combination rule to be used in reciprocal space is determined by `lj-pme-comb-rule`. If the current force field uses Lorentz-Berthelot combination rules, it is possible to set `lj-pme-comb-rule = geometric` in order to gain a significant increase in performance for a small loss in accuracy. The details of this approximation can be found in the section above.

Note that the use of a complete long-range dispersion correction means that as with Coulomb PME, `rvdw` is now a free parameter in the method, rather than being necessarily restricted by the force-field parameterization scheme. Thus it is now possible to optimize the cutoff, spacing, order and tolerance terms for accuracy and best performance.

Naturally, the use of LJ-PME rather than LJ cut-off adds computation and communication done for the reciprocal-space part, so for best performance in balancing the load of parallel simulations using PME-only ranks, more such ranks should be used. It may be possible to improve upon the automatic load-balancing used by *mdrun* (page 258).

## 5.5.10 Force field

A force field is built up from two distinct components:

- The set of equations (called the *potential functions*) used to generate the potential energies and their derivatives, the forces. These are described in detail in the previous chapter.
- The parameters used in this set of equations. These are not given in this manual, but in the data files corresponding to your GROMACS distribution.

Within one set of equations various sets of parameters can be used. Care must be taken that the combination of equations and parameters form a consistent set. It is in general dangerous to make *ad hoc* changes in a subset of parameters, because the various contributions to the total force are usually interdependent. This means in principle that every change should be documented, verified by comparison to experimental data and published in a peer-reviewed journal before it can be used.

GROMACS 2027.0-dev includes several force fields, and additional ones are available on the website. If you do not know which one to select we recommend GROMOS-96 for united-atom setups and OPLS-AA/L for all-atom parameters. That said, we describe the available options in some detail.

### GROMOS-96

> **Warning**
>
> The GROMOS force fields have been parametrized with a physically incorrect multiple-time-stepping scheme for a twin-range cut-off. When used with a single-range cut-off (or a correct Trotter multiple-time-stepping scheme), physical properties, such as the density, might differ from the intended values. Since there are researchers actively working on validating GROMOS with modern integrators we have not yet removed the GROMOS force fields, but you should be aware of these issues and check if molecules in your system are affected before proceeding. Further information is available in GitLab Issue 2884 , and a longer explanation of our decision to remove physically incorrect algorithms can be found at DOI:10.26434/chemrxiv.11474583.v1 .

GROMACS supports the GROMOS-96 force fields *77* (page 686). All parameters for the 43A1, 43A2 (development, improved alkane dihedrals), 45A3, 53A5, and 53A6 parameter sets are included. All standard building blocks are included and topologies can be built automatically by *pdb2gmx* (page 281).

The GROMOS-96 force field is a further development of the GROMOS-87 force field. It has improvements over the GROMOS-87 force field for proteins and small molecules. **Note** that the sugar parameters present in 53A6 do correspond to those published in 2004*110* (page 688), which are different from those present in 45A4, which is not included in GROMACS at this time. The 45A4 parameter set corresponds to a later revision of these parameters. The GROMOS-96 force field is not, however, recommended for use with long alkanes and lipids. The GROMOS-96 force field differs from the GROMOS-87 force field in a few respects:

- the force field parameters
- the parameters for the bonded interactions are not linked to atom types
- a fourth power bond stretching potential (*Fourth power potential* (page 487))
- an angle potential based on the cosine of the angle (*Cosine based angle potential* (page 490))

There are two differences in implementation between GROMACS and GROMOS-96 which can lead to slightly different results when simulating the same system with both packages:

- in GROMOS-96 neighbor searching for solvents is performed on the first atom of the solvent molecule. This is not implemented in GROMACS, but the difference with searching by centers of charge groups is very small
- the virial in GROMOS-96 is molecule-based. This is not implemented in GROMACS, which uses atomic virials

The GROMOS-96 force field was parameterized with a Lennard-Jones cut-off of 1.4 nm, so be sure to use a Lennard-Jones cut-off (`rvdw`) of at least 1.4. A larger cut-off is possible because the Lennard-Jones potential and forces are almost zero beyond 1.4 nm.

### GROMOS-96 files

GROMACS can read and write GROMOS-96 coordinate and trajectory files. These files should have the extension *g96* (page 576). Such a file can be a GROMOS-96 initial/final configuration file, a coordinate trajectory file, or a combination of both. The file is fixed format; all floats are written as 15.9, and as such, files can get huge. GROMACS supports the following data blocks in the given order:

- Header block:

```
TITLE (mandatory)
```

- Frame blocks:

```
TIMESTEP (optional)
POSITION/POSITIONRED (mandatory)
VELOCITY/VELOCITYRED (optional)
BOX (optional)
```

See the GROMOS-96 manual *77* (page 686) for a complete description of the blocks. **Note** that all GROMACS programs can read compressed (.Z) or gzipped (.gz) files.

### OPLS/AA

### AMBER

GROMACS provides native support for the following AMBER force fields:

- AMBER94 *111* (page 688)
- AMBER96 *112* (page 688)
- AMBER99 *113* (page 688)
- AMBER99SB *114* (page 688)
- AMBER99SB-ILDN *115* (page 689)
- AMBER03 *116* (page 689)
- AMBERGS *117* (page 689)
- AMBER14SB *197* (page 693), *199* (page 693)
- AMBER19SB *198* (page 693), *199* (page 693)

### Dihedrals

GROMACS supports reordering atoms in select dihedrals to match topologies produced by AMBER LEaP.

When this atom reordering is enabled, dihedral atoms $a_i$, $a_j$, $a_k$, and $a_l$ are reordered to $(a_l, a_k, a_j, a_i)$ instead of $(a_i, a_j, a_k, a_l)$ if $at_i > at_l$ or $at_i = at_l$ and $at_j > at_k$, where operator greater than is considered in lexicographical ordering, $at_i$, $at_j$, $at_k$, and $at_l$ are atom type strings from the matching dihedral type, except in case of the wildcard type X, which is replaced by space. This reordering doesn't affect the resulting energies, but it is a necessity for what follows.

Additionally, but only for improper dihedrals, the third atom ($a_k$ or $a_j$) is taken to be central, while the remaining atoms ($a_i$, $a_j$, and $a_l$ or $a_l$, $a_k$, and $a_i$) are then reordered by atom type from the matching improper dihedral type. This can affect the resulting energies.

The reordering is performed by *gmx grompp* (page 230) only when the preprocessor define `_FF_AMBER_LEAP_ATOM_REORDERING` is encountered, which enables this functionality only for select force fields (such as AMBER19SB). In addition, preprocessor define-based enablement allows fine-grained control in the topology file by combining `#define` and `#undef` statements, which can be useful for further force field research and experimentation.

### CMAPs

AMBER19SB and newer versions provide support for amino-acid-specific energy correction maps (CMAPs). When these force fields are used with *pdb2gmx* (page 281), the default option is to enable CMAPs. If that is not the desired behavior, they can be disabled with `-nocmap` parameters.

CMAP types are specified using the following format:

```
[ cmaptypes ]
C-* N-GLY XC-GLY C-GLY N-* 1 24 24\
3.44619344 4.59474328 4.73235504 5.80145072 8.84401368 ...
```

where:

- first five columns are pairs of atom and residue types,
- sixth coulumn is function type (presently only `1` is supported),
- seventh and eighth coulumns are grid extent in `x` and `y` dimensions,
- the following `grid extent x * grid extent y` values constitute the CMAP grid.

Note that AMBER19SB and newer versions require residue names to express the residue type so that CMAP angle terms can be correctly assigned in *grompp* (page 230). This is handled by the default of `-rtpres auto` for that option of *pdb2gmx* (page 281) in concert with the `[ bondedtypes ]` for the residue types for such forcefields.

Note also that in the current implementation the grid extent must be identical for `x` and `y` dimensions and the same for all CMAP type entries.

### CHARMM

GROMACS supports the CHARMM force field for proteins *118* (page 689), *119* (page 689), lipids *120* (page 689) and nucleic acids *121* (page 689), *122* (page 689). The protein parameters (and to some extent the lipid and nucleic acid parameters) were thoroughly tested – both by comparing potential energies between the port and the standard parameter set in the CHARMM molecular simulation package, as well by how the protein force field behaves together with GROMACS-specific techniques such as virtual sites (enabling long time steps) recently implemented *123* (page 689) – and the details and results are presented in the paper by Bjelkmar et al. *124* (page 689). The nucleic acid parameters, as well as the ones for HEME, were converted and tested by Michel Cuendet.

When selecting the CHARMM force field in *pdb2gmx* (page 281) the default option is to use CMAP (for torsional correction map). To exclude CMAP, use `-nocmap`. The basic form of the CMAP term implemented in GROMACS is a function of the $\phi$ and $\psi$ backbone torsion angles. This term is defined

in the `rtp` file by a `[ cmap ]` statement at the end of each residue supporting CMAP. The following five atom names define the two torsional angles. Atoms 1-4 define $\phi$, and atoms 2-5 define $\psi$. The corresponding atom types are then matched to the correct CMAP type in the `cmap.itp` file that contains the correction maps.

A port of the CHARMM36 force field for use with GROMACS is also available at the MacKerell lab webpage.

For branched polymers or other topologies not supported by *pdb2gmx* (page 281), it is possible to use TopoTools *125* (page 689) to generate a GROMACS top file.

### Coarse-grained force fields

Coarse-graining is a systematic way of reducing the number of degrees of freedom representing a system of interest. To achieve this, typically whole groups of atoms are represented by single beads and the coarse-grained force fields describes their effective interactions. Depending on the choice of parameterization, the functional form of such an interaction can be complicated and often tabulated potentials are used.

Coarse-grained models are designed to reproduce certain properties of a reference system. This can be either a full atomistic model or even experimental data. Depending on the properties to reproduce there are different methods to derive such force fields. An incomplete list of methods is given below:

- Conserving free energies
  - Simplex method
  - MARTINI force field (see next section)
- Conserving distributions (like the radial distribution function), so-called structure-based coarse-graining
  - (iterative) Boltzmann inversion
  - Inverse Monte Carlo
- Conversing forces
  - Force matching

Note that coarse-grained potentials are state dependent (e.g. temperature, density,...) and should be reparametrized depending on the system of interest and the simulation conditions. This can for example be done using the Versatile Object-oriented Toolkit for Coarse-Graining Applications (VOTCA) (**???**). The package was designed to assists in systematic coarse-graining, provides implementations for most of the algorithms mentioned above and has a well tested interface to GROMACS. It is available as open source and further information can be found at www.votca.org.

#### MARTINI

The MARTINI force field is a coarse-grain parameter set that allows for the construction of many systems, including proteins and membranes.

#### PLUM

The PLUM force field *126* (page 689) is an example of a solvent-free protein-membrane model for which the membrane was derived from structure-based coarse-graining *127* (page 689). A GROMACS implementation can be found at github.com/tbereau/plumx.

## 5.6 Topologies

GROMACS must know on which atoms and combinations of atoms the various contributions to the potential functions (see chapter *Interaction function and force fields* (page 481)) must act. It must also know what parameters must be applied to the various functions. All this is described in the *topology* file *top* (page 584), which lists the *constant attributes* of each atom. There are many more atom types than elements, but only atom types present in biological systems are parameterized in the force field, plus some metals, ions and silicon. The bonded and special interactions are determined by fixed lists that are included in the topology file. Certain non-bonded interactions must be excluded (first and second neighbors), as these are already treated in bonded interactions. In addition, there are *dynamic attributes* of atoms - their positions, velocities and forces. These do not strictly belong to the molecular topology, and are stored in the coordinate file *gro* (page 577) (positions and velocities), or trajectory file *trr* (page 586) (positions, velocities, forces).

This chapter describes the setup of the topology file, the *top* (page 584) file and the database files: what the parameters stand for and how/where to change them if needed. First, all file formats are explained. Section *Force-field files* (page 570) describes the organization of the files in each force field.

**Note:** if you construct your own topologies, we encourage you to upload them to our topology archive at our webpage! Just imagine how thankful you would have been if your topology had been available there before you started. The same goes for new force fields or modified versions of the standard force fields - contribute them to the force field archive!

### 5.6.1 Particle type

In GROMACS, there are three types of particles , see Table 5.10. Only regular atoms and virtual interaction sites are used in GROMACS; shells are necessary for polarizable models like the Shell-Water models *45* (page 685).

Table 5.10: Particle types in GROMACS

| Particle | Symbol |
|---|---|
| atom | A |
| shell | S |
| virtual site | V (or D) |

### Atom types

Each force field defines a set of atom types, which have a characteristic name or number, and mass (in a.m.u.). These listings are found in the `atomtypes.atp` file (*atp* (page 575) = **a**tom **t**ype **p**arameter file). Therefore, it is in this file that you can begin to change and/or add an atom type. This file is only used by *gmx pdb2gmx* (page 281). A sample from the `gromos43a1.ff` force field is listed below.

```
|  O  15.99940 ;     carbonyl oxygen (C=O)
| OM  15.99940 ;     carboxyl oxygen (CO-)
| OA  15.99940 ;     hydroxyl, sugar or ester oxygen
| OW  15.99940 ;     water oxygen
|  N  14.00670 ;     peptide nitrogen (N or NH)
| NT  14.00670 ;     terminal nitrogen (NH2)
| NL  14.00670 ;     terminal nitrogen (NH3)
| NR  14.00670 ;     aromatic nitrogen
| NZ  14.00670 ;     Arg NH (NH2)
| NE  14.00670 ;     Arg NE (NH)
|  C  12.01100 ;     bare carbon
|CH1  13.01900 ;     aliphatic or sugar CH-group
|CH2  14.02700 ;     aliphatic or sugar CH2-group
|CH3  15.03500 ;     aliphatic CH3-group
```

**Note:** GROMACS makes use of the atom types as a name, *not* as a number (as *e.g.* in GROMOS).

The interaction parameters for the atom types are set through the `[ atomtypes ]` section in the topology file, often obtained through including a force field parameter file. The atomtypes listed in the `atomtypes.atp` file and the `[ atomtypes ]` section are non-bonded atom types. These are used to look up the non-bonded Van der Waals interaction parameters. Some force fields use these same atom types to look up parameters for bonded interactions. Other force fields additionally use bonded atom types to look up parameters for bonded interactions. This is because there are often far fewer bonded atom types needed than non-bonded atom types. In this case, the set of parameters for each non-bonded atom type includes a bonded atom type. Another optional parameter for non-bonded atom types is the atomic number. This is only used in hybrid QM/MM simulations.

### Virtual sites

Some force fields use virtual interaction sites (interaction sites that are constructed from other particle positions) on which certain interactions are located (*e.g.* on benzene rings, to reproduce the correct quadrupole). This is described in sec. *Virtual interaction sites* (page 520).

To make virtual sites in your system, you should include a section `[ virtual_sites? ]` (for backward compatibility the old name `[ dummies? ]` can also be used) in your topology file, where the `?` stands for the number constructing particles for the virtual site. This will be `2` for type 2, `3` for types 3, 3fd, 3fad and 3out and `4` for type 4fdn. The last of these replace an older 4fd type (with the 'type' value 1) that could occasionally be unstable; while it is still supported internally in the code, the old 4fd type should not be used in new input files. The different types are explained in sec. *Virtual interaction sites* (page 520).

Parameters for type 1 should look like this:

```
[ virtual_sites1 ]
; Site  from         funct
5       1            1
```

for type 2 like this:

```
[ virtual_sites2 ]
; Site  from         funct  a
5       1     2      1      0.7439756
```

for type 2fd like this:

```
[ virtual_sites2 ]
; Site  from         funct  d
5       1     2      2      -0.105
```

for type 3 like this:

```
[ virtual_sites3 ]
; Site  from               funct   a          b
5       1     2     3      1       0.7439756  0.128012
```

for type 3fd like this:

```
[ virtual_sites3 ]
; Site  from               funct   a          d
5       1     2     3      2       0.5        -0.105
```

for type 3fad like this:

```
[ virtual_sites3 ]
; Site  from               funct   theta      d
5       1     2     3      3       120        0.5
```

for type 3out like this:

```
[ virtual_sites3 ]
; Site  from               funct   a          b          c
5       1     2     3      4       -0.4       -0.4       6.9281
```

for type 4fdn like this:

```
[ virtual_sites4 ]
; Site  from                     funct   a          b          c
5       1     2     3     4      2       1.0        0.9        0.105
```

This will result in the construction of a virtual site, number 5 (first column `Site`), based on the positions of the atoms whose indices are 1 and 2 or 1, 2 and 3 or 1, 2, 3 and 4 (next two, three or four columns `from`) following the rules determined by the function number (next column `funct`) with the parameters specified (last one, two or three columns `a b . .`). Obviously, the atom numbers (including virtual site number) depend on the molecule. It may be instructive to study the topologies for TIP4P or TIP5P water models that are included with the GROMACS distribution.

**Note** that if any constant bonded interactions are defined between virtual sites and/or normal atoms, they will be removed by *grompp* (page 230) (unless the option `-normvsbds` is used). This removal of bonded interactions is done after generating exclusions, as the generation of exclusions is based on "chemically" bonded interactions.

Virtual sites can be constructed in a more generic way using basic geometric parameters. The directive that can be used is `[ virtual_sitesn ]`. Required parameters are listed in Table 5.14. An example entry for defining a virtual site at the center of geometry of a given set of atoms might be:

```
[ virtual_sitesn ]
; Site   funct    from
5        1        1     2     3     4
```

### 5.6.2 Parameter files

#### Atoms

The *static* properties (see Table 5.11) assigned to the atom types are assigned based on data in several places. The mass is listed in `atomtypes.atp` (see *Atom types* (page 538)), whereas the charge is listed in *rtp* (page 582) (*rtp* (page 582) = **r**esidue **t**opology **p**arameter file, see *rtp* (page 582)). This implies that the charges are only defined in the building blocks of amino acids, nucleic acids or otherwise, as defined by the user. When generating a *topology* (page 584) using the *pdb2gmx* (page 281) program, the information from these files is combined.

Table 5.11: Static atom type properties in GROMACS

| Property | Symbol | Unit |
|---|---|---|
| Type | • | • |
| Mass | m | a.m.u. |
| Charge | q | electron |
| epsilon | $\epsilon$ | kJ/mol |
| sigma | $\sigma$ | nm |

#### Non-bonded parameters

The non-bonded parameters consist of the van der Waals parameters V (`c6` or $\sigma$, depending on the combination rule, see *The Lennard-Jones interaction* (page 481)) and W (`c12` or $\epsilon$), as listed in the file `ffnonbonded.itp`, where `ptype` is the particle type (see Table 5.10). As with the bonded parameters, entries in `[ *type ]` directives are applied to their counterparts in the topology file. Missing parameters generate warnings, except as noted below in section *Intramolecular pair interactions* (page 543).

```
[ atomtypes ]
;name   at.num      mass      charge   ptype         V(c6)          ␣
↪W(c12)
   O         8  15.99940       0.000       A   0.22617E-02   0.
↪74158E-06
  OM         8  15.99940       0.000       A   0.22617E-02   0.
```

(continues on next page)

(continued from previous page)

```
↪74158E-06
   .....

[ nonbond_params ]
  ; i    j func       V(c6)         W(c12)
    O    O    1 0.22617E-02   0.74158E-06
    O   OA    1 0.22617E-02   0.13807E-05
    .....
```

**Note** that most of the included force fields also include the `at.num.` column, but this same information is implied in the OPLS-AA `bond_type` column. The interpretation of the parameters V and W depends on the combination rule that was chosen in the `[ defaults ]` section of the topology file (see *Topology file* (page 554)):

$$\begin{aligned} \text{for combination rule 1 :} \quad & \begin{aligned} \mathrm{V}_{ii} &= C_i^{(6)} &&= 4\,\epsilon_i\sigma_i^6 && [\text{ kJ mol}^{-1}\text{ nm}^6\text{ }] \\ \mathrm{W}_{ii} &= C_i^{(12)} &&= 4\,\epsilon_i\sigma_i^{12} && [\text{ kJ mol}^{-1}\text{ nm}^{12}\text{ }] \end{aligned} \\ \text{for combination rules 2 and 3 :} \quad & \begin{aligned} \mathrm{V}_{ii} &= \sigma_i && [\text{ nm }] \\ \mathrm{W}_{ii} &= \epsilon_i && [\text{ kJ mol}^{-1}\text{ }] \end{aligned} \end{aligned} \tag{5.297}$$

Some or all combinations for different atom types can be given in the `[ nonbond_params ]` section, again with parameters V and W as defined above. Any combination that is not given will be computed from the parameters for the corresponding atom types, according to the combination rule:

$$\begin{aligned} \text{for combination rules 1 and 3 :} \quad & \begin{aligned} C_{ij}^{(6)} &= \left(C_i^{(6)}\,C_j^{(6)}\right)^{\frac{1}{2}} \\ C_{ij}^{(12)} &= \left(C_i^{(12)}\,C_j^{(12)}\right)^{\frac{1}{2}} \end{aligned} \\ \text{for combination rule 2 :} \quad & \begin{aligned} \sigma_{ij} &= \tfrac{1}{2}(\sigma_i + \sigma_j) \\ \epsilon_{ij} &= \sqrt{\epsilon_i\,\epsilon_j} \end{aligned} \end{aligned} \tag{5.298}$$

When $\sigma$ and $\epsilon$ need to be supplied (rules 2 and 3), it would seem it is impossible to have a non-zero $C^{12}$ combined with a zero $C^6$ parameter. However, providing a negative $\sigma$ will do exactly that, such that $C^6$ is set to zero and $C^{12}$ is calculated normally. This situation represents a special case in reading the value of $\sigma$, and nothing more.

There is only one set of combination rules for Buckingham potentials:

$$\begin{aligned} A_{ij} &= (A_{ii}\,A_{jj})^{1/2} \\ B_{ij} &= 2/\left(\tfrac{1}{B_{ii}} + \tfrac{1}{B_{jj}}\right) \\ C_{ij} &= (C_{ii}\,C_{jj})^{1/2} \end{aligned} \tag{5.299}$$

### Bonded parameters

The bonded parameters (*i.e.* bonds, bond angles, improper and proper dihedrals) are listed in `ffbonded.itp`. The entries in this database describe, respectively, the atom types in the interactions, the type of the interaction, and the parameters associated with that interaction. These parameters are then read by *grompp* (page 230) when processing a topology and applied to the relevant bonded parameters, *i.e.* `bondtypes` are applied to entries in the `[ bonds ]` directive, etc. Any bonded parameter that is missing from the relevant :`[ *type ]` directive generates a fatal error. The types of interactions are listed in Table 5.14. Example excerpts from such files follow:

```
[ bondtypes ]
  ; i    j  func       b0          kb
    C    O     1    0.12300      502080.
    C   OM     1    0.12500      418400.
    ......

[ angletypes ]
  ; i    j    k  func       th0         cth
   HO   OA    C     1    109.500      397.480
   HO   OA  CH1     1    109.500      397.480
   ......

[ dihedraltypes ]
  ; i    l  func       q0          cq
 NR5*  NR5     2      0.000      167.360
 NR5* NR5*     2      0.000      167.360
 ......

[ dihedraltypes ]
  ; j    k  func      phi0          cp   mult
    C   OA     1   180.000       16.736      2
    C    N     1   180.000       33.472      2
    ......

[ dihedraltypes ]
;
; Ryckaert-Bellemans Dihedrals
;
; aj    ak      funct
CP2     CP2     3       9.2789  12.156  -13.120 -3.0597 26.240  -31.
↪495
```

In the `ffbonded.itp` file, you can add bonded parameters. If you want to include parameters for new atom types, make sure you define them in `atomtypes.atp` as well.

For most interaction types, bonded parameters are searched and assigned using an exact match for all type names and allowing only a single set of parameters. The exception to this rule are dihedral parameters. For `[ dihedraltypes ]` wildcard atom type names can be specified with the letter `X` in one or more of the four positions. Thus one can for example assign proper dihedral parameters based on the types of the middle two atoms. The parameters for the entry with the most exact matches, i.e. the least wildcard matches, will be used. Note that GROMACS versions older than 5.1.3 used the first match, which means that a full match would be ignored if it is preceded by an entry that matches on wildcards. Thus it is suggested to put wildcard entries at the end, in case someone might use a forcefield with older versions of GROMACS. In addition there is a dihedral type 9 which adds the possibility of assigning multiple dihedral potentials, useful for combining terms with different multiplicities. The different dihedral potential parameter sets should be on directly adjacent lines in the `[ dihedraltypes ]` section.

### 5.6.3 Molecule definition

#### Moleculetype entries

An organizational structure that usually corresponds to molecules is the `[ moleculetype ]` entry. This entry serves two main purposes. One is to give structure to the topology file(s), usually corresponding to real molecules. This makes the topology easier to read and writing it less labor intensive. A second purpose is computational efficiency. The system definition that is kept in memory is proportional in size of the `moleculetype` definitions. If a molecule is present in 100000 copies, this saves a factor of 100000 in memory, which means the system usually fits in cache, which can improve performance tremendously. Interactions that correspond to chemical bonds, that generate exclusions, can only be defined between atoms within a `moleculetype`. It is allowed to have multiple molecules which are not covalently bonded in one `moleculetype` definition. Molecules can be made infinitely long by connecting to themselves over periodic boundaries. When such periodic molecules are present, an option in the *mdp* (page 579) file needs to be set to tell GROMACS not to attempt to make molecules that are broken over periodic boundaries whole again.

#### Intermolecular interactions

In some cases, one would like atoms in different molecules to also interact with other interactions than the usual non-bonded interactions. This is often the case in binding studies. When the molecules are covalently bound, e.g. a ligand binding covalently to a protein, they are effectively one molecule and they should be defined in one `[ moleculetype ]` entry. Note that *pdb2gmx* (page 281) has an option to put two or more molecules in one `[ moleculetype ]` entry. When molecules are not covalently bound, it is much more convenient to use separate `moleculetype` definitions and specify the intermolecular interactions in the `[ intermolecular_interactions]` section. In this section, which is placed at the end of the topology (see Table 5.13), normal bonded interactions can be specified using global atom indices. The only restrictions are that no interactions can be used that generates exclusions and no constraints can be used.

#### Intramolecular pair interactions

Extra Lennard-Jones and electrostatic interactions between pairs of atoms in a molecule can be added in the `[ pairs ]` section of a molecule definition. The parameters for these interactions can be set independently from the non-bonded interaction parameters. In the GROMOS force fields, pairs are only used to modify the 1-4 interactions (interactions of atoms separated by three bonds). In these force fields the 1-4 interactions are excluded from the non-bonded interactions (see sec. *Exclusions* (page 544)).

```
[ pairtypes ]
  ; i    j func         cs6           cs12 ; THESE ARE 1-4␣
↪INTERACTIONS
    O    O    1 0.22617E-02   0.74158E-06
    O   OM    1 0.22617E-02   0.74158E-06
    .....
```

The pair interaction parameters for the atom types in `ffnonbonded.itp` are listed in the `[ pairtypes ]` section. The GROMOS force fields list all these interaction parameters explicitly, but this section might be empty for force fields like OPLS that calculate the 1-4 interactions by uniformly scaling the parameters. Pair parameters that are not present in the `[ pairtypes ]` section are only generated when `gen-pairs` is set to `yes` in the `[ defaults ]` directive of `forcefield.itp`

(see *Topology file* (page 554)). When `gen-pairs` is set to `no`, *grompp* (page 230) will give a warning for each pair type for which no parameters are given.

The normal pair interactions, intended for 1-4 interactions, have function type 1. Function type 2 and the `[ pairs_nb ]` are intended for free-energy simulations. When determining hydration free energies, the solute needs to be decoupled from the solvent. This can be done by adding a B-state topology (see sec. *Free energy calculations* (page 469)) that uses zero for all solute non-bonded parameters, *i.e.* charges and LJ parameters. However, the free energy difference between the A and B states is not the total hydration free energy. One has to add the free energy for reintroducing the internal Coulomb and LJ interactions in the solute when in vacuum. This second step can be combined with the first step when the Coulomb and LJ interactions within the solute are not modified. For this purpose, there is a pairs function type 2, which is identical to function type 1, except that the B-state parameters are always identical to the A-state parameters. For searching the parameters in the `[ pairtypes ]` section, no distinction is made between function type 1 and 2. The pairs section `[ pairs_nb ]` is intended to replace the non-bonded interaction. It uses the unscaled charges and the non-bonded LJ parameters; it also only uses the A-state parameters. **Note** that one should add exclusions for all atom pairs listed in `[ pairs_nb ]`, otherwise such pairs will also end up in the normal neighbor lists.

Alternatively, this same behavior can be achieved without ever touching the topology, by using the `couple-moltype`, `couple-lambda0`, `couple-lambda1`, and `couple-intramol` keywords. See sections sec. *Free energy calculations* (page 469) and sec. *Free energy implementation* (page 590) for more information.

All three pair types always use plain Coulomb interactions, even when Reaction-field, PME, Ewald or shifted Coulomb interactions are selected for the non-bonded interactions. Energies for types 1 and 2 are written to the energy and log file in separate "LJ-14" and "Coulomb-14" entries per energy group pair. Energies for `[ pairs_nb ]` are added to the "LJ-(SR)" and "Coulomb-(SR)" terms.

### Exclusions

The exclusions for non-bonded interactions are generated by *grompp* (page 230) for neighboring atoms up to a certain number of bonds away, as defined in the `[ moleculetype ]` section in the topology file (see *Topology file* (page 554)). Particles are considered bonded when they are connected by "chemical" bonds (`[ bonds ]` types 1 to 5, 7 or 8) or constraints (`[ constraints ]` type 1). Type 5 `[ bonds ]` can be used to create a connection between two atoms without creating an interaction. There is a harmonic interaction (`[ bonds ]` type 6) that does not connect the atoms by a chemical bond. There is also a second constraint type (`[ constraints ]` type 2) that fixes the distance, but does not connect the atoms by a chemical bond. For a complete list of all these interactions, see Table 5.14.

Extra exclusions within a molecule can be added manually in a `[ exclusions ]` section. Each line should start with one atom index, followed by one or more atom indices. All non-bonded interactions between the first atom and the other atoms will be excluded.

### 5.6.4 Constraint algorithms

Constraints are defined in the `[ constraints ]` section. The format is two atom numbers followed by the function type, which can be 1 or 2, and the constraint distance. The only difference between the two types is that type 1 is used for generating exclusions and type 2 is not (see sec. *Exclusions* (page 544)). The distances are constrained using the LINCS or the SHAKE algorithm, which can be selected in the *mdp* (page 579) file. Both types of constraints can be perturbed in free-energy calculations by adding a second constraint distance (see *Constraint forces* (page 569)). Several types of bonds and angles (see Table 5.14) can be converted automatically to constraints by *grompp* (page 230). There are several options for this in the *mdp* (page 579) file.

We have also implemented the SETTLE algorithm *47* (page 685), which is an analytical solution of SHAKE, specifically for water. SETTLE can be selected in the topology file. See, for instance, the SPC molecule definition:

```
[ moleculetype ]
; molname       nrexcl
SOL             1

[ atoms ]
; nr    at type res nr  ren nm  at nm   cg nr   charge
1       OW      1       SOL     OW1     1       -0.82
2       HW      1       SOL     HW2     1        0.41
3       HW      1       SOL     HW3     1        0.41

[ settles ]
; OW    funct   doh     dhh
1       1       0.1     0.16333

[ exclusions ]
1       2       3
2       1       3
3       1       2
```

The `[ settles ]` directive defines the first atom of the water molecule. The settle funct is always 1, and the distance between O-H and H-H distances must be given. **Note** that the algorithm can also be used for TIP3P and TIP4P *128* (page 689). TIP3P just has another geometry. TIP4P has a virtual site, but since that is generated it does not need to be shaken (nor stirred).

### 5.6.5 pdb2gmx input files

The GROMACS program *pdb2gmx* (page 281) generates a topology for the input coordinate file. Several formats are supported for that coordinate file, but *pdb* (page 581) is the most commonly-used format (hence the name *pdb2gmx* (page 281)). *pdb2gmx* (page 281) searches for force fields in sub-directories of the GROMACS `share/top` directory and your working directory. Force fields are recognized from the file `forcefield.itp` in a directory with the extension `.ff`. The file `forcefield.doc` may be present, and if so, its first line will be used by *pdb2gmx* (page 281) to present a short description to the user to help in choosing a force field. Otherwise, the user can choose a force field with the `-ff xxx` command-line argument to *pdb2gmx* (page 281), which indicates that a force field in a `xxx.ff` directory is desired. *pdb2gmx* (page 281) will search first in the working directory, then in the GROMACS `share/top` directory, and use the first matching `xxx.ff` directory found.

Two general files are read by *pdb2gmx* (page 281): an atom type file (extension *atp* (page 575), see *Atom types* (page 538)) from the force-field directory, and a file called `residuetypes.dat` from either the working directory, or the GROMACS `share/top` directory. `residuetypes.dat` determines which residue names are considered protein, DNA, RNA, water, and ions.

*pdb2gmx* (page 281) can read one or multiple databases with topological information for different types of molecules. A set of files belonging to one database should have the same basename, preferably telling something about the type of molecules (*e.g.* aminoacids, rna, dna). The possible files are:

- `<basename>.rtp`
- `<basename>.r2b (optional)`
- `<basename>.arn (optional)`
- `<basename>.hdb (optional)`
- `<basename>.n.tdb (optional)`
- `<basename>.c.tdb (optional)`

Only the *rtp* (page 582) file, which contains the topologies of the building blocks, is mandatory. Information from other files will only be used for building blocks that come from an *rtp* (page 582) file with the same base name. The user can add building blocks to a force field by having additional files with the same base name in their working directory. By default, only extra building blocks can be defined, but calling *pdb2gmx* (page 281) with the `-rtpo` option will allow building blocks in a local file to replace the default ones in the force field.

### Residue database

The files holding the residue databases have the extension *rtp* (page 582). Originally this file contained building blocks (amino acids) for proteins, and is the GROMACS interpretation of the `rt37c4.dat` file of GROMOS. So the residue database file contains information (bonds, charges, charge groups, and improper dihedrals) for a frequently-used building block. It is better *not* to change this file because it is standard input for *pdb2gmx* (page 281), but if changes are needed make them in the *top* (page 584) file (see *Topology file* (page 554)), or in a *rtp* (page 582) file in the working directory as explained in sec. *pdb2gmx input files* (page 545). Defining topologies of new small molecules is probably easier by writing an include topology file *itp* (page 578) directly. This will be discussed in section *Molecule.itp file* (page 564). When adding a new protein residue to the database, do not forget to add the residue name to the residuetypes.dat file, so that *grompp* (page 230), *make_ndx* (page 255) and analysis tools can recognize the residue as a protein residue (see *Default Groups* (page 654)).

The *rtp* (page 582) files are only used by *pdb2gmx* (page 281). As mentioned before, the only extra information this program needs from the *rtp* (page 582) database is bonds, charges of atoms, charge groups, and improper dihedrals, because the rest is read from the coordinate input file. Some proteins contain residues that are not standard, but are listed in the coordinate file. You have to construct a building block for this "strange" residue, otherwise you will not obtain a *top* (page 584) file. This also holds for molecules in the coordinate file such as ligands, polyatomic ions, crystallization co-solvents, etc. The residue database is constructed in the following way:

```
[ bondedtypes ]  ; mandatory
; bonds  angles  dihedrals  impropers
     1       1          1          2  ; mandatory

[ GLY ]  ; mandatory
```

(continues on next page)

(continued from previous page)

```
 [ atoms ]  ; mandatory
; name  type  charge  chargegroup
     N     N  -0.280     0
     H     H   0.280     0
    CA   CH2   0.000     1
     C     C   0.380     2
     O     O  -0.380     2

 [ bonds ]  ; optional
;atom1 atom2      b0      kb
     N     H
     N    CA
    CA     C
     C     O
    -C     N

 [ exclusions ]  ; optional
;atom1 atom2

 [ angles ]  ; optional
;atom1 atom2 atom3    th0    cth

 [ dihedrals ]  ; optional
;atom1 atom2 atom3 atom4   phi0     cp   mult

 [ impropers ]  ; optional
;atom1 atom2 atom3 atom4     q0     cq
     N    -C    CA     H
    -C   -CA     N    -O

[ ZN ]

 [ atoms ]
    ZN    ZN   2.000     0
```

The file is free format; the only restriction is that there can be at most one entry on a line. The first field in the file is the `[ bondedtypes ]` field, which is followed by four numbers, indicating the interaction type for bonds, angles, dihedrals, and improper dihedrals. The file contains residue entries, which consist of atoms and (optionally) bonds, angles, dihedrals, and impropers. The charge group codes denote the charge group numbers. Atoms in the same charge group should always be ordered consecutively. When using the hydrogen database with *pdb2gmx* (page 281) for adding missing hydrogens (see *hdb* (page 578)), the atom names defined in the *rtp* (page 582) entry should correspond exactly to the naming convention used in the hydrogen database. The atom names in the bonded interaction can be preceded by a minus or a plus, indicating that the atom is in the preceding or following residue respectively. Explicit parameters added to bonds, angles, dihedrals, and impropers override the standard parameters in the *itp* (page 578) files. This should only be used in special cases. Instead of parameters, a string can be added for each bonded interaction. This is used in GROMOS-96 *rtp* (page 582) files. These strings are copied to the topology file and can be replaced by force-field parameters by the C-preprocessor in *grompp* (page 230) using `#define` statements.

*pdb2gmx* (page 281) automatically generates all angles. This means that for most force fields the `[ angles ]` field is only useful for overriding *itp* (page 578) parameters. For the GROMOS-96 force field the interaction number of all angles needs to be specified.

*pdb2gmx* (page 281) automatically generates one proper dihedral for every rotatable bond, preferably on heavy atoms. When the `[ dihedrals ]` field is used, no other dihedrals will be generated for the bonds corresponding to the specified dihedrals. It is possible to put more than one dihedral function on a rotatable bond. In the case of CHARMM27 FF *pdb2gmx* (page 281) can add correction maps to the dihedrals using the default `-cmap` option. Please refer to *CHARMM* (page 535) for more information.

*pdb2gmx* (page 281) sets the number of exclusions to 3, which means that interactions between atoms connected by at most 3 bonds are excluded. Pair interactions are generated for all pairs of atoms that are separated by 3 bonds (except pairs of hydrogens). When more interactions need to be excluded, or some pair interactions should not be generated, an `[ exclusions ]` field can be added, followed by pairs of atom names on separate lines. All non-bonded and pair interactions between these atoms will be excluded.

### Residue to building block database

Each force field has its own naming convention for residues. Most residues have consistent naming, but some, especially those with different protonation states, can have many different names. The *r2b* (page 583) files are used to convert standard residue names to the force-field build block names. If no *r2b* (page 583) is present in the force-field directory or a residue is not listed, the building block name is assumed to be identical to the residue name. The *r2b* (page 583) can contain 2 or 5 columns. The 2-column format has the residue name in the first column and the building block name in the second. The 5-column format has 3 additional columns with the building block for the residue occurring in the N-terminus, C-terminus and both termini at the same time (single residue molecule). This is useful for, for instance, the AMBER force fields. If one or more of the terminal versions are not present, a dash should be entered in the corresponding column.

There is a GROMACS naming convention for residues which is only apparent (except for the *pdb2gmx* (page 281) code) through the *r2b* (page 583) file and `specbond.dat` files. This convention is only of importance when you are adding residue types to an *rtp* (page 582) file. The convention is listed in Table 5.12. For special bonds with, for instance, a heme group, the GROMACS naming convention is introduced through `specbond.dat` (see *Special bonds* (page 553)), which can subsequently be translated by the *r2b* (page 583) file, if required.

Table 5.12: Internal GROMACS residue naming convention.

| GROMACS ID | Residue |
|---|---|
| ARG | protonated arginine |
| ARGN | neutral arginine |
| ASP | negatively charged aspartic acid |
| ASPH | neutral aspartic acid |
| CYS | neutral cysteine |
| CYS2 | cysteine with sulfur bound to another cysteine or a heme |
| GLU | negatively charged glutamic acid |
| GLUH | neutral glutamic acid |
| HISD | neutral histidine with $N_\delta$ protonated |
| HISE | neutral histidine with $N_\epsilon$ protonated |
| HISH | positive histidine with both $N_\delta$ and $N_\epsilon$ protonated |
| HIS1 | histidine bound to a heme |
| LYSN | neutral lysine |
| LYS | protonated lysine |
| HEME | heme |

### Atom renaming database

Force fields often use atom names that do not follow IUPAC or PDB convention. The *arn* (page 575) database is used to translate the atom names in the coordinate file to the force-field names. Atoms that are not listed keep their names. The file has three columns: the building block name, the old atom name, and the new atom name, respectively. The residue name supports question-mark wildcards that match a single character.

An additional general atom renaming file called `xlateat.dat` is present in the `share/top` directory, which translates common non-standard atom names in the coordinate file to IUPAC/PDB convention. Thus, when writing force-field files, you can assume standard atom names and no further atom name translation is required, except for translating from standard atom names to the force-field ones.

### Hydrogen database

The hydrogen database is stored in *hdb* (page 578) files. It contains information for the *pdb2gmx* (page 281) program on how to connect hydrogen atoms to existing atoms. In versions of the database before GROMACS 3.3, hydrogen atoms were named after the atom they are connected to: the first letter of the atom name was replaced by an 'H.' In the versions from 3.3 onwards, the H atom has to be listed explicitly, because the old behavior was protein-specific and hence could not be generalized to other molecules. If more than one hydrogen atom is connected to the same atom, a number will be added to the end of the hydrogen atom name. For example, adding two hydrogen atoms to `ND2` (in asparagine), the hydrogen atoms will be named `HD21` and `HD22`. This is important since atom naming in the *rtp* (page 582) file must be the same. The format of the hydrogen database is as follows:

```
; res    # additions
         # H add type    H       i       j       k
ALA      1
         1       1       H       N       -C      CA
ARG      4
```

(continues on next page)

(continued from previous page)

```
        1       2       H       N       CA      C
        1       1       HE      NE      CD      CZ
        2       3       HH1     NH1     CZ      NE
        2       3       HH2     NH2     CZ      NE
```

On the first line we see the residue name (ALA or ARG) and the number of kinds of hydrogen atoms that may be added to this residue by the hydrogen database. After that follows one line for each addition, on which we see:

- The number of H atoms added
- The method for adding H atoms, which can be any of:

  1. *one planar hydrogen, e.g. rings or peptide bond*

     One hydrogen atom (n) is generated, lying in the plane of atoms (i,j,k) on the plane bisecting angle (j-i-k) at a distance of 0.1 nm from atom i, such that the angles (n-i-j) and (n-i-k) are $> 90^o$.

  2. *one single hydrogen, e.g. hydroxyl*

     One hydrogen atom (n) is generated at a distance of 0.1 nm from atom i, such that angle (n-i-j)=109.5 degrees and dihedral (n-i-j-k)=trans.

  3. *two planar hydrogens, e.g. ethylene -C=$CH_2$, or amide -C(=O)$NH_2$*

     Two hydrogens (n1,n2) are generated at a distance of 0.1 nm from atom i, such that angle (n1-i-j)=(n2-i-j)=120 degrees and dihedral (n1-i-j-k)=cis and (n2-i-j-k)=trans, such that names are according to IUPAC standards *129* (page 689).

  4. *two or three tetrahedral hydrogens, e.g. -$CH_3$*

     Three (n1,n2,n3) or two (n1,n2) hydrogens are generated at a distance of 0.1 nm from atom i, such that angle (n1-i-j)=(n2-i-j)=(n3-i-j)=109.47$^o$, dihedral (n1-i-j-k)=trans, (n2-i-j-k)=trans+120 and (n3-i-j-k)=trans+240$^o$.

  5. *one tetrahedral hydrogen, e.g. $C_3CH$*

     One hydrogen atom (n′) is generated at a distance of 0.1 nm from atom i in tetrahedral conformation such that angle (n′-i-j)=(n′-i-k)=(n′-i-l)=109.47$^o$.

  6. *two tetrahedral hydrogens, e.g. C-$CH_2$-C*

     Two hydrogen atoms (n1,n2) are generated at a distance of 0.1 nm from atom i in tetrahedral conformation on the plane bisecting angle j-i-k with angle (n1-i-n2)=(n1-i-j)=(n1-i-k)=109.47$^o$.

  7. *two water hydrogens*

     Two hydrogens are generated around atom i according to SPC *80* (page 687) water geometry. The symmetry axis will alternate between three coordinate axes in both directions.

  8. *three water "hydrogens"*

     Two hydrogens are generated around atom i according to SPC *80* (page 687) water geometry. The symmetry axis will alternate between three coordinate axes in both directions. In addition, an extra particle is generated on the position of the oxygen with the first letter of the name replaced by 'M'. This is for use with four-atom water models such as TIP4P *128* (page 689).

  9. *four water "hydrogens"*

Same as above, except that two additional particles are generated on the position of the oxygen, with names 'LP1' and 'LP2.' This is for use with five-atom water models such as TIP5P *130* (page 689).

- The name of the new H atom (or its prefix, *e.g.* `HD2` for the asparagine example given earlier).
- Three or four control atoms (i,j,k,l), where the first always is the atom to which the H atoms are connected. The other two or three depend on the code selected. For water, there is only one control atom.

Some more exotic cases can be approximately constructed from the above tools, and with suitable use of energy minimization are good enough for beginning MD simulations. For example secondary amine hydrogen, nitrenyl hydrogen (C = NH) and even ethynyl hydrogen could be approximately constructed using method 2 above for hydroxyl hydrogen.

### Termini database

The termini databases are stored in `aminoacids.n.tdb` and `aminoacids.c.tdb` for the N- and C-termini respectively. They contain information for the *pdb2gmx* (page 281) program on how to connect new atoms to existing ones, which atoms should be removed or changed, and which bonded interactions should be added. Their format is as follows (from `gromos43a1.ff/aminoacids.c.tdb`):

```
[ None ]

[ COO- ]
[ replace ]
C   C       C       12.011  0.27
O   O1      OM      15.9994 -0.635
OXT O2      OM      15.9994 -0.635
[ add ]
2   8       O       C       CA      N
    OM      15.9994 -0.635
[ bonds ]
C   O1      gb_5
C   O2      gb_5
[ angles ]
O1  C       O2      ga_37
CA  C       O1      ga_21
CA  C       O2      ga_21
[ dihedrals ]
N   CA      C       O2      gd_20
[ impropers ]
C   CA      O2      O1      gi_1
```

The file is organized in blocks, each with a header specifying the name of the block. These blocks correspond to different types of termini that can be added to a molecule. In this example `[ COO- ]` is the first block, corresponding to changing the terminal carbon atom into a deprotonated carboxyl group. `[ None ]` is the second terminus type, corresponding to a terminus that leaves the molecule as it is. Block names cannot be any of the following: `replace`, `add`, `delete`, `bonds`, `angles`, `dihedrals`, `impropers`. Doing so would interfere with the parameters of the block, and would probably also be very confusing to human readers.

For each block the following options are present:

- `[ replace ]`
  Replace an existing atom by one with a different atom type, atom name, charge, and/or mass. This entry can be used to replace an atom that is present both in the input coordinates and in the *rtp* (page 582) database, but also to only rename an atom in the input coordinates such that it matches the name in the force field. In the latter case, there should also be a corresponding `[ add ]` section present that gives instructions to add the same atom, such that the position in the sequence and the bonding is known. Such an atom can be present in the input coordinates and kept, or not present and constructed by *pdb2gmx* (page 281). For each atom to be replaced on line should be entered with the following fields:
  - name of the atom to be replaced
  - new atom name (optional)
  - new atom type
  - new mass
  - new charge
- `[ add ]`
  Add new atoms. For each (group of) added atom(s), a two-line entry is necessary. The first line contains the same fields as an entry in the hydrogen database (name of the new atom, number of atoms, type of addition, control atoms, see *hdb* (page 578)), but the possible types of addition are extended by two more, specifically for C-terminal additions:
  1. *two carboxyl oxygens, -COO$^-$*
     Two oxygens (n1,n2) are generated according to rule 3, at a distance of 0.136 nm from atom i and an angle (n1-i-j)=(n2-i-j)=117 degrees
  2. *carboxyl oxygens and hydrogen, -COOH*
     Two oxygens (n1,n2) are generated according to rule 3, at distances of 0.123 nm and 0.125 nm from atom i for n1 and n2, respectively, and angles (n1-i-j)=121 and (n2-i-j)=115 degrees. One hydrogen (n$'$) is generated around n2 according to rule 2, where n-i-j and n-i-j-k should be read as n$'$-n2-i and n$'$-n2-i-j, respectively.

  After this line, another line follows that specifies the details of the added atom(s), in the same way as for replacing atoms, *i.e.*:
  - atom type
  - mass
  - charge
  - charge group (optional)

  Like in the hydrogen database (see *rtp* (page 582)), when more than one atom is connected to an existing one, a number will be appended to the end of the atom name. **Note** that, like in the hydrogen database, the atom name is now on the same line as the control atoms, whereas it was at the beginning of the second line prior to GROMACS version 3.3. When the charge group field is left out, the added atom will have the same charge group number as the atom that it is bonded to.
- `[ delete ]`
  Delete existing atoms. One atom name per line.
- `[ bonds ]`, `[ angles ]`, `[ dihedrals ]` and `[ impropers ]`

Add additional bonded parameters. The format is identical to that used in the *rtp* (page 582) file, see *rtp* (page 582).

### Virtual site database

Since we cannot rely on the positions of hydrogens in input files, we need a special input file to decide the geometries and parameters with which to add virtual site hydrogens. For more complex virtual site constructs (*e.g.* when entire aromatic side chains are made rigid) we also need information about the equilibrium bond lengths and angles for all atoms in the side chain. This information is specified in the *vsd* (page 586) file for each force field. Just as for the termini, there is one such file for each class of residues in the *rtp* (page 582) file.

The virtual site database is a simple list of information. The first couple of sections specify which mass centers (typically called $MCH_3$/$MNH_3$) to use for $CH_3$, $NH_3$, and $NH_2$ groups. Depending on the equilibrium bond lengths and angles between the hydrogens and heavy atoms we need to apply slightly different constraint distances between these mass centers. **Note** that we do *not* have to specify the actual parameters (that is automatic), just the type of mass center to use. To accomplish this, there are three sections names `[ CH3 ]`, `[ NH3 ]`, and `[ NH2 ]`. For each of these we expect three columns. The first column is the atom type bound to the 2/3 hydrogens, the second column is the next heavy atom type which this is bound, and the third column the type of mass center to use. As a special case, in the `[ NH2 ]` section it is also possible to specify `planar` in the second column, which will use a different construction without mass center. There are currently different opinions in some force fields whether an $NH_2$ group should be planar or not, but we try hard to stick to the default equilibrium parameters of the force field.

The second part of the virtual site database contains explicit equilibrium bond lengths and angles for pairs/triplets of atoms in aromatic side chains. These entries are currently read by specific routines in the virtual site generation code, so if you would like to extend it *e.g.* to nucleic acids you would also need to write new code there. These sections are named after the short amino acid names (`[ PHE ]`, `[ TYR ]`, `[ TRP ]`, `[ HID ]`, `[ HIE ]`, `[ HIP ]`), and simply contain 2 or 3 columns with atom names, followed by a number specifying the bond length (in nm) or angle (in degrees). **Note** that these are approximations of the equilibrated geometry for the entire molecule, which might not be identical to the equilibrium value for a single bond/angle if the molecule is strained.

### Special bonds

The primary mechanism used by *pdb2gmx* (page 281) to generate inter-residue bonds relies on head-to-tail linking of backbone atoms in different residues to build a macromolecule. In some cases (*e.g.* disulfide bonds, a heme group, branched polymers), it is necessary to create inter-residue bonds that do not lie on the backbone. The file `specbond.dat` takes care of this function. It is necessary that the residues belong to the same `[ moleculetype ]`. The `-merge` and `-chainsep` functions of *pdb2gmx* (page 281) can be useful when managing special inter-residue bonds between different chains.

The first line of `specbond.dat` indicates the number of entries that are in the file. If you add a new entry, be sure to increment this number. The remaining lines in the file provide the specifications for creating bonds. For these bonds, you can also optionally specify a custom improper dihedral associated with the new bond. The format of the lines, with optional entries in [], is as follows:

```
resA atomA nbondsA resB atomB nbondsB length newresA newresB [atomI
atomJ atomK atomL]
```

The columns indicate:

1. `resA` The name of residue A that participates in the bond.
2. `atomA` The name of the atom in residue A that forms the bond.
3. `nbondsA` The total number of bonds `atomA` can form.
4. `resB` The name of residue B that participates in the bond.
5. `atomB` The name of the atom in residue B that forms the bond.
6. `nbondsB` The total number of bonds `atomB` can form.
7. `length` The reference length for the bond. If `atomA` and `atomB` are not within `length` ± 10% in the coordinate file supplied to *pdb2gmx* (page 281), no bond will be formed.
8. `newresA` The new name of residue A, if necessary. Some force fields use *e.g.* CYS2 for a cysteine in a disulfide or heme linkage.
9. `newresB` The new name of residue B, likewise.
10. **`atomI` Custom improper dihedral atom i of i-j-k-l. Has format** [specbond residue]-[atom name] (e.g. B-SG). The letter is either A or B corresponding to resA or resB, respectively.
11. `atomJ` Custom improper dihedral atom j of i-j-k-l.
12. `atomK` Custom improper dihedral atom k of i-j-k-l.
13. `atomL` Custom improper dihedral atom l of i-j-k-l.

### 5.6.6 File formats

#### Topology file

The topology file is built following the GROMACS specification for a molecular topology. A *top* (page 584) file can be generated by *pdb2gmx* (page 281). All possible entries in the topology file are listed in Tables 5.13 and 5.14. Also tabulated are: all the units of the parameters, which interactions can be perturbed for free energy calculations, which bonded interactions are used by *grompp* (page 230) for generating exclusions, and which bonded interactions can be converted to constraints by *grompp* (page 230).

Table 5.13: The topology file.

| Parameters | | | | |
|---|---|---|---|---|
| interaction type | directive | # at. | f. tp | parameters |
| *mandatory* | `defaults` | non-bonded function type; combination rule$^{(cr)}$; generate pairs (no/yes); fudge LJ (); fudge QQ () | | |
| *mandatory* | `atomtypes` | atom type; bonded type; atomic number; m (u); q (e); particle type; V$^{(cr)}$ ; W$^{(cr)}$ (bonded type and atomic number are optional) | | |
| | `bondtypes` | (see Table 5.14, directive `bonds`) | | |
| | `pairtypes` | (see Table 5.14, directive `pairs`) | | |
| | `angletypes` | (see Table 5.14, directive `angles`) | | |
| | `dihedraltypes` | (see Table 5.14, directive `dihedrals`) | | |
| | `constrainttyp` | (see Table 5.14, directive `constraints`) | | |
| LJ | `nonbond_params` | 2 | 1 | V$^{(cr)}$ ; W$^{(cr)}$ |
| Buckingham | `nonbond_params` | 2 | 2 | $a$ kJ mol$^{-1}$ ; $b$ nm$^{-1}$; $c_6$ (kJ mol$^{-1}$ nm$^{-6}$) |

| Molecule definition(s) | | | | F. E. |
|---|---|---|---|---|
| *mandatory* | `moleculetype` | | molecule name; $n_{ex}^{(nrexcl)}$ | |
| *mandatory* | `atoms` | 1 | atom type; residue number; residue name; atom name; charge group number; $q$ (e); $m$ (u) | type $q, m$ |
| intra-molecular interaction and geometry definitions as described in Table 5.14 | | | | |

| System | | |
|---|---|---|
| *mandatory* | `system` | system name |
| *mandatory* | `molecules` | molecule name; number of molecules |

| Inter-molecular interactions | |
|---|---|
| *optional* | `intermolecular_interactions` |
| one or more bonded interactions as described in Table 5.14, with two or more atoms, no interactions that generate exclusions, no constraints, use global atom numbers | |

- `# at` is the required number of atom type indices for this directive
- `f. tp` is the value used to select this function type
- `F. E.` indicates which of the parameters can be interpolated in free energy calculations
- $^{(cr)}$ the combination rule determines the type of LJ parameters, see *Non-bonded parameters* (page 540)
- $^{(*)}$ for `dihedraltypes` one can specify 4 atoms or the inner (outer for improper) 2 atoms
- $n_{ex}^{(nrexcl)}$ exclude neighbors $n_{ex}$ bonds away for non-bonded interactions
- For free energy calculations, type, $q$ and $m$ or no parameters should be added for topology `B` ($\lambda = 1$) on the same line, after the normal parameters.

Table 5.14: Details of `[ moleculetype ]` directives

| Name of interaction | Topology file directive | num. atoms[Page 560, 1] | func. type[Page 560, 2] | Order of parameters and their units | use in F.E.?[Page 560, 3] |
|---|---|---|---|---|---|
| bond | `bonds`[4],[5] | 2 | 1 | $b_0$ (nm); $k_b$ (kJ mol$^{-1}$nm$^{-2}$ | all |
| G96 bond | `bonds`[Page 560, 4] | 2 | 2 | $b_0$ (nm); $k_b$ (kJ mol$^{-1}$nm$^{-4}$ | all |
| Morse | `bonds`[Page 560, 4] | 2 | 3 | $b_0$ (nm); $D$ (kJ mol$^{-1}$; $\beta$ (nm$^{-1}$ | all |
| cubic bond | `bonds`[Page 560, 4] | 2 | 4 | $b_0$ (nm); $C_{i=2,3}$ (kJ mol$^{-1}$nm$^{-i}$ | |
| connection | `bonds`[Page 560, 4] | 2 | 5 | | |
| harmonic potential | `bonds` | 2 | 6 | $b_0$ (nm); $k_b$ (kJ mol$^{-1}$nm$^{-2}$ | all |
| FENE bond | `bonds`[Page 560, 4] | 2 | 7 | $b_m$ (nm); $k_b$ (kJ mol$^{-1}$nm$^{-2}$ | |
| tabulated bond | `bonds`[Page 560, 4] | 2 | 8 | table number ($\geq$ 0); $k$ kJ mol$^{-1}$ | $k$ |
| tabulated bond[6] | `bonds` | 2 | 9 | table number ($\geq$ 0); $k$ kJ mol$^{-1}$ | $k$ |
| restraint potential | `bonds` | 2 | 10 | low, up$_{1,2}$ (nm); $k_{dr}$ ((kJ mol$^{-1}$nm$^{-2}$) | all |

continues on next page

Table 5.14 – continued from previous page

| Name of interaction | Topology file directive | num. atoms[Page 560, 1] | func. type[Page 560, 2] | Order of parameters and their units | use in F.E.?[Page 560, 3] |
|---|---|---|---|---|---|
| extra LJ or Coulomb | `pairs` | 2 | 1 | $V$[7]; $W$[Page 560, 7] | all |
| extra LJ or Coulomb | `pairs` | 2 | 2 | fudge QQ (); $q_i$; $q_j$ (e), $V$[Page 560, 7]; $W$[Page 560, 7] | |
| extra LJ or Coulomb | `pairs_nb` | 2 | 1 | $q_i$; $q_j$ (e); $V$[Page 560, 7]; $W$[Page 560, 7] | |
| angle | `angles`[Page 560] | 3 | 1 | $\theta_0$ (deg); $k_\theta$ (kJ mol$^{-1}$rad$^{-2}$) | all |
| G96 angle | `angles`[Page 560] | 3 | 2 | $\theta_0$ (deg); $k_\theta$ (kJ mol$^{-1}$) | all |
| cross bond-bond | `angles` | 3 | 3 | $r_{1e}$, $r_{2e}$ (nm); $k_{rr'}$ ((kJ mol$^{-1}$nm$^{-2}$) | |
| cross bond-angle | `angles` | 3 | 4 | $r_{1e}$, $r_{2e}$, $r_{3e}$ (nm); $k_{r\theta}$ ((kJ mol$^{-1}$nm$^{-2}$) | |
| Urey-Bradley | `angles`[Page 560] | 3 | 5 | $\theta_0$ (deg); $k_\theta$ (kJ mol$^{-1}$rad$^{-2}$); $r_{13}$ (nm); $k_{UB}$ ((kJ mol$^{-1}$nm$^{-2}$) | all |
| quartic angle | `angles`[Page 560] | 3 | 6 | $\theta_0$ (deg); $C_{i=0,1,2,3,4}$ (kJ mol$^{-1}$rad$^{-i}$) | |
| tabulated angle | `angles` | 3 | 8 | table number ($\geq$ 0); $k$ (kJ mol$^{-1}$) | $k$ |
| linear angle | `angles` | 3 | 9 | $a_0$; $k_{lin}$ ((kJ mol$^{-1}$nm$^{-2}$) | all |
| restricted bending potential | `angles` | 3 | 10 | $\theta_0$ (deg); $k_\theta$ (kJ mol$^{-1}$) | |
| proper dihedral | `dihedrals` | 4 | 1 | $\phi_s$ (deg); $k_\phi$ (kJ mol$^{-1}$); multiplicity | $\phi, k$ |

continues on next page

Table 5.14 – continued from previous page

| Name of interaction | Topology file directive | num. atoms[Page 560, 1] | func. type[Page 560, 2] | Order of parameters and their units | use in F.E.?[Page 560, 3] |
|---|---|---|---|---|---|
| improper dihedral | dihedrals | 4 | 2 | $\xi_0$ (deg); $k_\xi$ (kJ mol$^{-1}$rad$^{-2}$) | all |
| Ryckaert-Bellemans dihedral | dihedrals | 4 | 3 | $C_0$, $C_1$, $C_2$, $C_3$, $C_4$, $C_5$ (kJ mol$^{-1}$) | all |
| periodic improper dihedral | dihedrals | 4 | 4 | $\phi_s$ (deg); $k_\phi$ (kJ mol$^{-1}$); multiplicity | $\phi, k$ |
| Fourier dihedral | dihedrals | 4 | 5 | $C_1$, $C_2$, $C_3$, $C_4$, $C_5$ (kJ mol$^{-1}$) | all |
| tabulated dihedral | dihedrals | 4 | 8 | table number ($\geq$ 0); $k$ (kJ mol$^{-1}$) | $k$ |
| proper dihedral (multiple)[8] | dihedrals | 4 | 9 | $\phi_s$ (deg); $k_\phi$ (kJ mol$^{-1}$); multiplicity | $\phi, k$ |
| restricted dihedral | dihedrals | 4 | 10 | $\phi_0$ (deg); $k_\phi$ (kJ mol$^{-1}$) | |
| combined bending-torsion potential | dihedrals | 4 | 11 | $k_\phi$ (kJ mol$^{-1}$); $a_0$, $a_1$, $a_2$, $a_3$, $a_4$ | |
| exclusions | exclusions | 1 | | one or more atom indices | |
| constraint | constraints | 2 | 1 | $b_0$ (nm) | all |
| constraint[Page 560, 6] | constraints | 2 | 2 | $b_0$ (nm) | all |
| SETTLE | settles | 1 | 1 | $d_{\mathrm{oh}}$, $d_{\mathrm{hh}}$ (nm) | |
| 1-body virtual site | virtual_sites1 | 2 | 1 | | |
| 2-body virtual site | virtual_sites2 | 3 | 1 | $a$ () | |
| 2-body virtual site (fd) | virtual_sites2 | 3 | 2 | $d$ (nm) | |
| 3-body virtual site | virtual_sites3 | 4 | 1 | $a$, $b$ () | |
| 3-body virtual site (fd) | virtual_sites3 | 4 | 2 | $a$ (); $d$ (nm) | |
| 3-body virtual site (fad) | virtual_sites3 | 4 | 3 | $\theta$ (deg); $d$ (nm) | |
| 3-body virtual site (out) | virtual_sites3 | 4 | 4 | $a$, $b$ (); $c$ (nm$^{-1}$) | |

continues on next page

Table 5.14 – continued from previous page

| Name of interaction | Topology file directive | num. atoms[Page 560, 1] | func. type[Page 560, 2] | Order of parameters and their units | use in F.E.?[Page 560, 3] |
|---|---|---|---|---|---|
| 4-body virtual site (fdn) | virtual_sites4 | 5 | 2 | $a$, $b$ (); $c$ (nm) | |
| N-body virtual site (COG) | virtual_sitesn | 1 | 1 | one or more constructing atom indices | |
| N-body virtual site (COM) | virtual_sitesn | 1 | 2 | one or more constructing atom indices | |
| N-body virtual site (COW) | virtual_sitesn | 1 | 3 | one or more pairs consisting of constructing atom index and weight | |
| position restraint | position_restraints | 1 | 1 | $k_x$, $k_y$, $k_z$ ((kJ mol$^{-1}$nm$^{-2}$) | all |
| flat-bottomed position restraint | position_restraints | 1 | 2 | $g$, $r$ (nm), $k$ ((kJ mol$^{-1}$nm$^{-2}$) | |
| distance restraint | distance_restraints | 2 | 1 | type; label; low, up$_{1,2}$ (nm); weight () | |
| dihedral restraint | dihedral_restraints | 4 | 1 | $\phi_0$ (deg); $\Delta\phi$ (deg); $k_{\mathrm{dihr}}$ (kJ mol$^{-1}$rad$^{-2}$) | all |
| orientation restraint | orientation restraints | 2 | 1 | exp.; label; $\alpha$; $c$ (U nm$^{\alpha}$; obs. (U); weight (U$^{-1}$) | |
| angle restraint | angle_restraints | 4 | 1 | $\theta_0$ (deg); $k_c$ (kJ mol$^{-1}$); multiplicity | $\theta, k$ |
| angle restraint (z) | angle_restraints_z | 2 | 1 | $\theta_0$ (deg); $k_c$ (kJ mol$^{-1}$); multiplicity | $\theta, k$ |

Description of the file layout:

- Semicolon (;) and newline characters surround comments
- On a line ending with \ the newline character is ignored.
- Directives are surrounded by `[` and `]`
- The topology hierarchy (which must be followed) consists of three levels:
  - the parameter level, which defines certain force-field specifications (see Table 5.13)
  - the molecule level, which should contain one or more molecule definitions (see Table 5.14)
  - the system level, containing only system-specific information (`[ system ]` and `[ molecules ]`)
- Items should be separated by spaces or tabs, not commas
- Atoms in molecules should be numbered consecutively starting at 1
- Atoms in the same charge group must be listed consecutively
- Bonded atom type name must contain at least one non-digit character.
- The file is parsed only once, which implies that no forward references can be treated: items must be defined before they can be used
- Exclusions can be generated from the bonds or overridden manually
- The bonded force types can be generated from the atom types or overridden per bond
- It is possible to apply multiple bonded interactions of the same type on the same atoms
- Descriptive comment lines and empty lines are highly recommended
- All directives at the parameter level can be used multiple times and there are no restrictions on the order, except that an atom type needs to be defined before it can be used in other parameter definitions
- If parameters for a certain interaction are defined multiple times for the same combination of atom types the last definition is used; *grompp* (page 230) generates a warning for parameter redefinitions with different values, with the exception of `[dihedraltype]` 9
- Using one of the `[ atoms ]`, `[ bonds ]`, `[ pairs ]`, `[ angles ]`, etc. without having used `[ moleculetype ]` before is meaningless and generates a warning
- Using `[ molecules ]` without having used `[ system ]` before is meaningless and generates a warning.
- After `[ system ]` the only allowed directive is `[ molecules ]`
- Using an unknown string in `[ ]` causes all the data until the next directive to be ignored and generates a warning

[1] The required number of atom indices for this directive
[2] The index to use to select this function type
[3] Indicates which of the parameters can be interpolated in free energy calculations
[4] This interaction type will be used by *grompp* (page 230) for generating exclusions
[5] This interaction type can be converted to constraints by *grompp* (page 230)
[6] No connection, and so no exclusions, are generated for this interaction
[7] The combination rule determines the type of LJ parameters, see *Non-bonded parameters* (page 540)
[8] Multiple, consecutive parameter entries will generate multiple dihedral potentials

Here is an example of a topology file, `urea.top`:

```
;
;       Example topology file
;
; The force-field files to be included
#include "amber99.ff/forcefield.itp"

[ moleculetype ]
; name  nrexcl
Urea         3

[ atoms ]
   1   C   1   URE      C      1     0.880229   12.01000   ; amber C ␣
↪type
   2   O   1   URE      O      2    -0.613359   16.00000   ; amber O ␣
↪type
   3   N   1   URE     N1      3    -0.923545   14.01000   ; amber N ␣
↪type
   4   H   1   URE    H11      4     0.395055    1.00800   ; amber H ␣
↪type
   5   H   1   URE    H12      5     0.395055    1.00800   ; amber H ␣
↪type
   6   N   1   URE     N2      6    -0.923545   14.01000   ; amber N ␣
↪type
   7   H   1   URE    H21      7     0.395055    1.00800   ; amber H ␣
↪type
   8   H   1   URE    H22      8     0.395055    1.00800   ; amber H ␣
↪type

[ bonds ]
    1    2
    1    3
    1    6
    3    4
    3    5
    6    7
    6    8

[ dihedrals ]
;   ai    aj    ak    al funct  definition
     2     1     3     4   9
     2     1     3     5   9
     2     1     6     7   9
     2     1     6     8   9
     3     1     6     7   9
     3     1     6     8   9
     6     1     3     4   9
     6     1     3     5   9

[ dihedrals ]
```

(continues on next page)

(continued from previous page)

```
     3      6      1      2    4
     1      4      3      5    4
     1      7      6      8    4

[ position_restraints ]
; you wouldn't normally use this for a molecule like Urea,
; but we include it here for didactic purposes
; ai   funct    fc
   1      1     1000    1000    1000 ; Restrain to a point
   2      1     1000       0    1000 ; Restrain to a line (Y-axis)
   3      1     1000       0       0 ; Restrain to a plane (Y-Z-
↪plane)

[ dihedral_restraints ]
; ai   aj    ak    al  type  phi   dphi   fc
    3    6     1    2     1  180      0   10
    1    4     3    5     1  180      0   10

; Include TIP3P water topology
#include "amber99.ff/tip3p.itp"

[ system ]
Urea in Water

[ molecules ]
;molecule name   nr.
Urea             1
SOL              1000
```

Here follows the explanatory text.

**#include "amber99.ff/forcefield.itp" :** this includes the information for the force field you are using, including bonded and non-bonded parameters. This example uses the AMBER99 force field, but your simulation may use a different force field. *grompp* (page 230) will automatically go and find this file and copy-and-paste its content. That content can be seen in `share/top/amber99.ff/forcefield.itp`}, and it is

```
#define _FF_AMBER
#define _FF_AMBER99

[ defaults ]
; nbfunc        comb-rule       gen-pairs       fudgeLJ fudgeQQ
1               2               yes             0.5     0.8333

#include "ffnonbonded.itp"
#include "ffbonded.itp"
```

The two `#define` statements set up the conditions so that future parts of the topology can know that the AMBER 99 force field is in use.

**[ defaults ] :**

- `nbfunc` is the non-bonded function type. Use 1 (Lennard-Jones) or 2 (Buckingham, no longer supported by *gmx mdrun* (page 258))
- `comb-rule` is the number of the combination rule (see *Non-bonded parameters* (page 540)).
- `gen-pairs` is for pair generation. The default is 'no', *i.e.* get 1-4 parameters from the pairtypes list. When parameters are not present in the list, stop with a fatal error. Setting 'yes' generates 1-4 parameters that are not present in the pair list from normal Lennard-Jones parameters using `fudgeLJ`
- `fudgeLJ` is the factor by which to multiply Lennard-Jones 1-4 interactions, default 1
- `fudgeQQ` is the factor by which to multiply electrostatic 1-4 interactions, default 1
- $N$ is the power for the repulsion term in a 6-$N$ potential (with nonbonded-type Lennard-Jones only), starting with GROMACS version 4.5, *grompp* (page 258) also reads and applies $N$, for values not equal to 12 tabulated interaction functions are used (in older version you would have to use user tabulated interactions).

**Note** that `gen-pairs`, `fudgeLJ`, `fudgeQQ`, and $N$ are optional. `fudgeLJ` is only used when generate pairs is set to 'yes', and `fudgeQQ` is always used. However, if you want to specify $N$ you need to give a value for the other parameters as well.

Then some other `#include` statements add in the large amount of data needed to describe the rest of the force field. We will skip these and return to `urea.top`. There we will see

**[ moleculetype ] :** defines the name of your molecule in this *top* (page 584) and nrexcl = 3 stands for excluding non-bonded interactions between atoms that are no further than 3 bonds away.

**[ atoms ] :** defines the molecule, where `nr` and `type` are fixed, the rest is user defined. So `atom` can be named as you like, `cgnr` made larger or smaller (if possible, the total charge of a charge group should be zero), and charges can be changed here too.

**[ bonds ] :** no comment.

**[ pairs ] :** LJ and Coulomb 1-4 interactions

**[ angles ] :** no comment

**[ dihedrals ] :** in this case there are 9 proper dihedrals (funct = 1), 3 improper (funct = 4) and no Ryckaert-Bellemans type dihedrals. If you want to include Ryckaert-Bellemans type dihedrals in a topology, do the following (in case of *e.g.* decane):

```
[ dihedrals ]
;  ai    aj    ak    al funct       c0       c1       c2
    1     2     3     4     3
    2     3     4     5     3
```

In the original implementation of the potential for alkanes *131* (page 689) no 1-4 interactions were used, which means that in order to implement that particular force field you need to remove the 1-4 interactions from the `[ pairs ]` section of your topology. In most modern force fields, like OPLS/AA or Amber the rules are different, and the Ryckaert-Bellemans potential is used as a cosine series in combination with 1-4 interactions.

**[ position_restraints ] :** harmonically restrain the selected particles to reference positions (*Position restraints* (page 500)). The reference positions are read from a separate coordinate file by *grompp* (page 230).

**[ dihedral_restraints ] :** restrain selected dihedrals to a reference value. The implementation of dihedral restraints is described in section *Dihedral restraints* (page 502) of the manual. The parameters specified in the `[dihedral_restraints]` directive are as follows:

- `type` has only one possible value which is 1
- `phi` is the value of $\phi_0$ in (5.196) and (5.197) of the manual.
- `dphi` is the value of $\Delta\phi$ in (5.197) of the manual.
- `fc` is the force constant $k_{dihr}$ in (5.197) of the manual.

**#include "tip3p.itp" :** includes a topology file that was already constructed (see section *Molecule.itp file* (page 564)).

**[ system ] :** title of your system, user-defined

**[ molecules ] :** this defines the total number of (sub)molecules in your system that are defined in this *top* (page 584). In this example file, it stands for 1 urea molecule dissolved in 1000 water molecules. The molecule type `SOL` is defined in the `tip3p.itp` file. Each name here must correspond to a name given with `[ moleculetype ]` earlier in the topology. The order of the blocks of molecule types and the numbers of such molecules must match the coordinate file that accompanies the topology when supplied to *grompp* (page 230). The blocks of molecules do not need to be contiguous, but some tools (e.g. *genion* (page 227)) may act only on the first or last such block of a particular molecule type. Also, these blocks have nothing to do with the definition of groups (see sec. *The group concept* (page 432) and sec. *Using Groups* (page 653)).

### Molecule.itp file

If you construct a topology file you will use frequently (like the water molecule, `tip3p.itp`, which is already constructed for you) it is good to make a `molecule.itp` file. This only lists the information of one particular molecule and allows you to re-use the `[ moleculetype ]` in multiple systems without re-invoking *pdb2gmx* (page 281) or manually copying and pasting. An example `urea.itp` follows:

```
[ moleculetype ]
; molname   nrexcl
URE     3

[ atoms ]
   1  C  1  URE      C      1     0.880229  12.01000   ; amber C ␣
↪type
...
   8  H  1  URE    H22      8     0.395055   1.00800   ; amber H ␣
↪type

[ bonds ]
    1   2
...
    6   8
[ dihedrals ]
;   ai    aj    ak    al funct  definition
     2     1     3     4    9
...
```

(continues on next page)

(continued from previous page)

```
     6      1      3      5    9
[ dihedrals ]
     3      6      1      2    4
     1      4      3      5    4
     1      7      6      8    4
```

Using *itp* (page 578) files results in a very short *top* (page 584) file:

```
;
;       Example topology file
;
; The force field files to be included
#include "amber99.ff/forcefield.itp"

#include "urea.itp"

; Include TIP3P water topology
#include "amber99/tip3p.itp"

[ system ]
Urea in Water

[ molecules ]
;molecule name   nr.
Urea             1
SOL              1000
```

### Ifdef statements

A very powerful feature in GROMACS is the use of `#ifdef` statements in your *top* (page 584) file. By making use of this statement, and associated `#define` statements like were seen in `amber99.ff/forcefield.itp` earlier, different parameters for one molecule can be used in the same *top* (page 584) file. An example is given for TFE, where there is an option to use different charges on the atoms: charges derived by De Loof et al. *132* (page 689) or by Van Buuren and Berendsen *133* (page 690). In fact, you can use much of the functionality of the C preprocessor, `cpp`, because *grompp* (page 230) contains similar pre-processing functions to scan the file. The way to make use of the `#ifdef` option is as follows:

- either use the option `define = -DDeLoof` in the *mdp* (page 579) file (containing *grompp* (page 230) input parameters), or use the line `#define DeLoof` early in your *top* (page 584) or *itp* (page 578) file; and
- put the `#ifdef` statements in your *top* (page 584), as shown below:

```
...


[ atoms ]
; nr      type      resnr     residu      atom       cgnr       charge    ␣
↪     mass
```

(continues on next page)

(continued from previous page)

```
#ifdef DeLoof
; Use Charges from DeLoof
   1         C         1         TFE         C         1         0.74
   2         F         1         TFE         F         1        -0.25
   3         F         1         TFE         F         1        -0.25
   4         F         1         TFE         F         1        -0.25
   5       CH2         1         TFE       CH2         1         0.25
   6        OA         1         TFE        OA         1        -0.65
   7        HO         1         TFE        HO         1         0.41
#else
; Use Charges from VanBuuren
   1         C         1         TFE         C         1         0.59
   2         F         1         TFE         F         1        -0.2
   3         F         1         TFE         F         1        -0.2
   4         F         1         TFE         F         1        -0.2
   5       CH2         1         TFE       CH2         1         0.26
   6        OA         1         TFE        OA         1        -0.55
   7        HO         1         TFE        HO         1         0.3
#endif

[ bonds ]
;  ai    aj funct            c0            c1
    6     7     1 1.000000e-01 3.138000e+05
    1     2     1 1.360000e-01 4.184000e+05
    1     3     1 1.360000e-01 4.184000e+05
    1     4     1 1.360000e-01 4.184000e+05
    1     5     1 1.530000e-01 3.347000e+05
    5     6     1 1.430000e-01 3.347000e+05
...
```

This mechanism is used by *pdb2gmx* (page 281) to implement optional position restraints (*Position restraints* (page 500)) by `#include`-ing an *itp* (page 578) file whose contents will be meaningful only if a particular `#define` is set (and spelled correctly!)

### Topologies for free energy calculations

Free energy differences between two systems, A and B, can be calculated as described in sec. *Free energy calculations* (page 469). Systems A and B are described by topologies consisting of the same number of molecules with the same number of atoms. Masses and non-bonded interactions can be perturbed by adding B parameters under the `[ atoms ]` directive. Bonded interactions can be perturbed by adding B parameters to the bonded types or the bonded interactions. The parameters that can be perturbed are listed in Tables 5.13 and 5.14. The $\lambda$-dependence of the interactions is described in section sec. *Free energy interactions* (page 511). The bonded parameters that are used (on the line of the bonded interaction definition, or the ones looked up on atom types in the bonded type lists) is explained in Table 5.15. In most cases, things should work intuitively. When the A and B atom types in a bonded interaction are not all identical and parameters are not present for the B-state, either on the line or in the bonded types, *grompp* (page 230) uses the A-state parameters and issues a warning. For free energy calculations, all or no parameters for topology B ($\lambda = 1$) should be added on the same line, after the normal parameters, in the same order as the normal parameters. From GROMACS 4.6 onward, if $\lambda$ is treated as a vector,

then the `bonded-lambdas` component controls all bonded terms that are not explicitly labeled as restraints. Restrain terms are controlled by the `restraint-lambdas` component.

Table 5.15: The bonded parameters that are used for free energy topologies, on the line of the bonded interaction definition or looked up in the bond types section based on atom types. A and B indicate the parameters used for state A and B respectively, + and − indicate the (non-)presence of parameters in the topology, x indicates that the presence has no influence.

| B-state atom types all identical to A-state atom types | parameters on line | | parameters in \| parameters in bonded types \| bonded types of A atoms \| of B atoms | | | | expected message |
|---|---|---|---|---|---|---|---|
| | A | B | A | B | A | B | |
| yes | +AB | − | x | x | | | |
| yes | +A | +B | x | x | | | |
| yes | − | − | − | − | | | error |
| yes | − | − | +AB | − | | | |
| yes | − | − | +A | +B | | | |
| no | +AB | − | x | x | x | x | warning |
| no | +A | +B | x | x | x | x | |
| no | − | − | − | − | x | x | error |
| no | − | − | +AB | − | − | − | warning |
| no | − | − | +A | +B | − | − | warning |
| no | − | − | +A | x | +B | − | |
| no | − | − | +A | x | | +B | |

Below is an example of a topology which changes from 200 propanols to 200 pentanes using the GROMOS-96 force field.

```
; Include force field parameters
#include "gromos43a1.ff/forcefield.itp"

[ moleculetype ]
; Name            nrexcl
PropPent          3

[ atoms ]
; nr type resnr residue atom cgnr  charge    mass  typeB chargeB ␣
↪massB
  1    H    1     PROP    PH    1   0.398    1.008  CH3      0.0  15.
↪035
  2   OA    1     PROP    PO    1  -0.548  15.9994  CH2      0.0  14.
↪027
  3  CH2    1     PROP   PC1    1   0.150   14.027  CH2      0.0  14.
```

(continues on next page)

(continued from previous page)

```
↪027
  4   CH2     1      PROP    PC2     2    0.000    14.027
  5   CH3     1      PROP    PC3     2    0.000    15.035

[ bonds ]
;   ai     aj  funct     par_A   par_B
     1      2      2     gb_1    gb_26
     2      3      2     gb_17   gb_26
     3      4      2     gb_26   gb_26
     4      5      2     gb_26

[ pairs ]
;   ai     aj  funct
     1      4      1
     2      5      1

[ angles ]
;   ai     aj     ak  funct     par_A    par_B
     1      2      3      2     ga_11    ga_14
     2      3      4      2     ga_14    ga_14
     3      4      5      2     ga_14    ga_14

[ dihedrals ]
;   ai     aj     ak     al  funct     par_A    par_B
     1      2      3      4      1     gd_12    gd_17
     2      3      4      5      1     gd_17    gd_17

[ system ]
; Name
Propanol to Pentane

[ molecules ]
; Compound         #mols
PropPent           200
```

Atoms that are not perturbed, `PC2` and `PC3`, do not need B-state parameter specifications, since the B parameters will be copied from the A parameters. Bonded interactions between atoms that are not perturbed do not need B parameter specifications, as is the case for the last bond in the example topology. Topologies using the OPLS/AA force field need no bonded parameters at all, since both the A and B parameters are determined by the atom types. Non-bonded interactions involving one or two perturbed atoms use the free-energy perturbation functional forms. Non-bonded interactions between two non-perturbed atoms use the normal functional forms. This means that when, for instance, only the charge of a particle is perturbed, its Lennard-Jones interactions will also be affected when lambda is not equal to zero or one.

**Note** that this topology uses the GROMOS-96 force field, in which the bonded interactions are not determined by the atom types. The bonded interaction strings are converted by the C-preprocessor. The force-field parameter files contain lines like:

```
#define gb_26        0.1530   7.1500e+06
```

(continues on next page)

(continued from previous page)

```
#define gd_17     0.000       5.86          3
```

### Constraint forces

The constraint force between two atoms in one molecule can be calculated with the free energy perturbation code by adding a constraint between the two atoms, with a different length in the A and B topology. When the B length is 1 nm longer than the A length and lambda is kept constant at zero, the derivative of the Hamiltonian with respect to lambda is the constraint force. For constraints between molecules, the pull code can be used, see sec. *Collective variables: the pull code* (page 592). Below is an example for calculating the constraint force at 0.7 nm between two methanes in water, by combining the two methanes into one "molecule." **Note** that the definition of a "molecule" in GROMACS does not necessarily correspond to the chemical definition of a molecule. In GROMACS, a "molecule" can be defined as any group of atoms that one wishes to consider simultaneously. The added constraint is of function type 2, which means that it is not used for generating exclusions (see sec. *Exclusions* (page 544)). Note that the constraint free energy term is included in the derivative term, and is specifically included in the `bonded-lambdas` component. However, the free energy for changing constraints is *not* included in the potential energy differences used for BAR and MBAR, as this requires reevaluating the energy at each of the constraint components. This functionality is planned for later versions.

```
; Include force-field parameters
#include "gromos43a1.ff/forcefield.itp"

[ moleculetype ]
; Name            nrexcl
Methanes               1

[ atoms ]
; nr   type   resnr  residu   atom    cgnr     charge     mass
   1    CH4     1     CH4      C1       1          0    16.043
   2    CH4     1     CH4      C2       2          0    16.043
[ constraints ]
;  ai    aj funct   length_A  length_B
    1     2     2        0.7       1.7

#include "gromos43a1.ff/spc.itp"

[ system ]
; Name
Methanes in Water

[ molecules ]
; Compound        #mols
Methanes              1
SOL                2002
```

#### Coordinate file

Files with the *gro* (page 577) file extension contain a molecular structure in GROMOS-87 format. A sample piece is included below:

```
MD of 2 waters, reformat step, PA aug-91
    6
    1WATER  OW1    1   0.126   1.624   1.679  0.1227 -0.0580  0.0434
    1WATER  HW2    2   0.190   1.661   1.747  0.8085  0.3191 -0.7791
    1WATER  HW3    3   0.177   1.568   1.613 -0.9045 -2.6469  1.3180
    2WATER  OW1    4   1.275   0.053   0.622  0.2519  0.3140 -0.1734
    2WATER  HW2    5   1.337   0.002   0.680 -1.0641 -1.1349  0.0257
    2WATER  HW3    6   1.326   0.120   0.568  1.9427 -0.8216 -0.0244
   1.82060   1.82060   1.82060
```

This format is fixed, *i.e.* all columns are in a fixed position. If you want to read such a file in your own program without using the GROMACS libraries you can use the following formats:

**C-format:** `"%5i%5s%5s%5i%8.3f%8.3f%8.3f%8.4f%8.4f%8.4f"`

Or to be more precise, with title *etc.* it looks like this:

```
"%s\n", Title
"%5d\n", natoms
for (i=0; (i<natoms); i++) {
  "%5d%-5s%5s%5d%8.3f%8.3f%8.3f%8.4f%8.4f%8.4f\n",
    residuenr,residuename,atomname,atomnr,x,y,z,vx,vy,vz
}
"%10.5f%10.5f%10.5f%10.5f%10.5f%10.5f%10.5f%10.5f%10.5f\n",
  box[X][X],box[Y][Y],box[Z][Z],
  box[X][Y],box[X][Z],box[Y][X],box[Y][Z],box[Z][X],box[Z][Y]
```

**Fortran format:** `(i5,2a5,i5,3f8.3,3f8.4)`

So `confin.gro` is the GROMACS coordinate file and is almost the same as the GROMOS-87 file (for GROMOS users: when used with `ntx=7`). The only difference is the box for which GROMACS uses a tensor, not a vector.

### 5.6.7 Force field organization

#### Force-field files

Many force fields are available by default. Force fields are detected by the presence of `<name>.ff` directories in the `$GMXLIB/share/gromacs/top` sub-directory and/or the working directory. The information regarding the location of the force field files is printed by *pdb2gmx* (page 281) so you can easily keep track of which version of a force field is being called, in case you have made modifications in one location or another. The force fields included with GROMACS are:

- AMBER03 protein, nucleic AMBER94 (Duan et al., J. Comp. Chem. 24, 1999-2012, 2003)
- AMBER94 force field (Cornell et al., JACS 117, 5179-5197, 1995)
- AMBER96 protein, nucleic AMBER94 (Kollman et al., Acc. Chem. Res. 29, 461-469, 1996)

- AMBER99 protein, nucleic AMBER94 (Wang et al., J. Comp. Chem. 21, 1049-1074, 2000)
- AMBER99SB protein, nucleic AMBER94 (Hornak et al., Proteins 65, 712-725, 2006)
- AMBER99SB-ILDN protein, nucleic AMBER94 (Lindorff-Larsen et al., Proteins 78, 1950-58, 2010)
- AMBER14SB protein (Maier, J. A. et al., J. Chem. Theory Comput. 11, 3696–3713, 2015))
- AMBER19SB protein (Tian, C. et al., J. Chem. Theory Comput. 16, 528–552, 2020)
- AMBERGS force field (Garcia & Sanbonmatsu, PNAS 99, 2782-2787, 2002)
- CHARMM27 all-atom force field (CHARM22 plus CMAP for proteins)
- GROMOS96 43a1 force field
- GROMOS96 43a2 force field (improved alkane dihedrals)
- GROMOS96 45a3 force field (Schuler JCC 2001 22 1205)
- GROMOS96 53a5 force field (JCC 2004 vol 25 pag 1656)
- GROMOS96 53a6 force field (JCC 2004 vol 25 pag 1656)
- GROMOS96 54a7 force field (Eur. Biophys. J. (2011), 40,, 843-856, DOI: 10.1007/s00249-011-0700-9)
- OPLS-AA/L all-atom force field (2001 aminoacid dihedrals)

A force field is included at the beginning of a topology file with an `#include` statement followed by `<name>.ff/forcefield.itp`. This statement includes the force-field file, which, in turn, may include other force-field files. All the force fields are organized in the same way. An example of the `amber99.ff/forcefield.itp` was shown in *Topology file* (page 554).

For each force field, there several files which are only used by *pdb2gmx* (page 281). These are: residue databases (*rtp* (page 582)) the hydrogen database (*hdb* (page 578)), two termini databases (`.n.tdb` and `.c.tdb`, see ) and the atom type database (*atp* (page 575)), which contains only the masses. Other optional files are described in sec. *pdb2gmx input files* (page 545).

### Changing force-field parameters

If one wants to change the parameters of few bonded interactions in a molecule, this is most easily accomplished by typing the parameters behind the definition of the bonded interaction directly in the *top* (page 584) file under the `[ moleculetype ]` section (see *Topology file* (page 554) for the format and units). If one wants to change the parameters for all instances of a certain interaction one can change them in the force-field file or add a new `[ ???types ]` section after including the force field. When parameters for a certain interaction are defined multiple times, the last definition is used. As of GROMACS version 3.1.3, a warning is generated when parameters are redefined with a different value. Changing the Lennard-Jones parameters of an atom type is not recommended, because in the GROMOS force fields the Lennard-Jones parameters for several combinations of atom types are not generated according to the standard combination rules. Such combinations (and possibly others that do follow the combination rules) are defined in the `[ nonbond_params ]` section, and changing the Lennard-Jones parameters of an atom type has no effect on these combinations.

### Adding atom types

As of GROMACS version 3.1.3, atom types can be added in an extra `[ atomtypes ]` section after the inclusion of the normal force field. After the definition of the new atom type(s), additional non-bonded and pair parameters can be defined. In pre-3.1.3 versions of GROMACS, the new atom types needed to be added in the `[ atomtypes ]` section of the force-field files, because all non-bonded parameters above the last `[ atomtypes ]` section would be overwritten using the standard combination rules.

## 5.7 File formats

### 5.7.1 Summary of file formats

#### Parameter files

***mdp*** **(page 579)**
: run parameters, input for *gmx grompp* (page 230) and *gmx convert-tpr* (page 178)

***m2p*** **(page 578)**
: input for *gmx xpm2ps* (page 357)

#### Structure files

***gro*** **(page 577)**
: GROMACS format

***g96*** **(page 576)**
: GROMOS-96 format

***pdb*** **(page 581)**
: brookhaven Protein DataBank format

**Structure+mass(db):** ***tpr*** **(page 586),** ***gro*** **(page 577),** ***g96*** **(page 576), or** ***pdb*** **(page 581)**
: Structure and mass input for analysis tools. When gro or pdb is used approximate masses will be read from the mass database.

#### Topology files

***top*** **(page 584)**
: system topology (ascii)

***itp*** **(page 578)**
: include topology (ascii)

***rtp*** **(page 582)**
: residue topology (ascii)

***ndx*** **(page 580)**
: index file (ascii)

***n2t*** **(page 581)**
: atom naming definition (ascii)

***atp*** **(page 575)**
: atom type library (ascii)

***r2b*** **(page 583)**
: residue to building block mapping (ascii)

***arn*** **(page 575)**
: atom renaming database (ascii)

***hdb*** **(page 578)**
: hydrogen atom database (ascii)

*vsd* **(page 586)**
: virtual site database (ascii)

*tdb* **(page 583)**
: termini database (ascii)

### Run Input files

*tpr* **(page 586)**
: system topology, parameters, coordinates and velocities (binary, portable)

### Trajectory files

*tng* **(page 584)**
: Any kind of data (compressed, portable, any precision)

*trr* **(page 586)**
: x, v and f (binary, full precision, portable)

*xtc* **(page 588)**
: x only (compressed, portable, any precision)

*gro* **(page 577)**
: x and v (ascii, any precision)

*g96* **(page 576)**
: x only (ascii, fixed high precision)

*pdb* **(page 581)**
: x only (ascii, reduced precision)

**Formats for full-precision data:**
: *tng* (page 584) or *trr* (page 586)

**Generic trajectory formats:**
: *tng* (page 584), *xtc* (page 588), *trr* (page 586), *gro* (page 577), *g96* (page 576), or *pdb* (page 581)

### Energy files

*ene* **(page 576)**
: energies, temperature, pressure, box size, density and virials (binary)

*edr* **(page 576)**
: energies, temperature, pressure, box size, density and virials (binary, portable)

**Generic energy formats:**
: *edr* (page 576) or *ene* (page 576)

#### Other files

***dat*** **(page 576)**
generic, preferred for input

***edi*** **(page 576)**
essential dynamics constraints input for *gmx mdrun* (page 258)

***eps*** **(page 576)**
Encapsulated Postscript

***log*** **(page 578)**
log file

***mtx*** **(page 580)**
binary matrix data

***out*** **(page 581)**
generic, preferred for output

***tex*** **(page 583)**
LaTeX input

***xpm*** **(page 587)**
ascii matrix data, use *gmx xpm2ps* (page 357) to convert to *eps* (page 576)

***xvg*** **(page 589)**
xvgr input

### 5.7.2 File format details

#### atp

The atp file contains general information about atom types, like the atom number and the mass in atomic mass units.

#### arn

The arn file allows the renaming of atoms from their force field names to the names as defined by IUPAC/PDB, to allow easier visualization and identification.

#### cpt

The cpt file extension stands for portable checkpoint file. The complete state of the simulation is stored in the checkpoint file, including extended thermostat/barostat variables, random number states and NMR time averaged data. With domain decomposition also the some decomposition setup information is stored.

See also *gmx mdrun* (page 258).

### dat

Files with the dat file extension contain generic input or output. As it is not possible to categorize all data file formats, GROMACS has a generic file format called dat of which no format is given.

### edi

Files with the edi file extension contain information for *gmx mdrun* (page 258) to run Molecular Dynamics with Essential Dynamics constraints. It used to be possible to generate those through the options provided in the WHAT IF program.

### edr

The edr file extension stands for portable energy file. The energies are stored using the xdr protocol.

See also *gmx energy* (page 214).

### ene

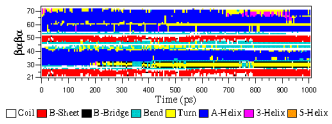
The ene file extension stands for binary energy file. It holds the energies as generated during your *gmx mdrun* (page 258).

The file can be transformed to a portable energy file (portable across hardware platforms), the *edr* (page 576) file using the program *gmx eneconv* (page 211).

See also *gmx energy* (page 214).

### eps

The eps file format is not a special GROMACS format, but just a variant of the standard PostScript(tm). A sample eps file as generated by the *gmx xpm2ps* (page 357) program is included below. It shows the secondary structure of a peptide as a function of time.




### g96

A file with the g96 extension can be a GROMOS-96 initial/final configuration file or a coordinate trajectory file or a combination of both. The file is fixed format, all floats are written as 15.9 (files can get huge). GROMACS supports the following data blocks in the given order:

- Header block:
  - `TITLE` (mandatory)
- Frame blocks:
  - `TIMESTEP` (optional)
  - `POSITION/POSITIONRED` (mandatory)

- VELOCITY/VELOCITYRED (optional)
- BOX (optional)

See the GROMOS-96 manual for a complete description of the blocks.

Note that all GROMACS programs can read compressed or g-zipped files.

### gro

Files with the gro file extension contain a molecular structure in Gromos87 format. gro files can be used as trajectory by simply concatenating files. An attempt will be made to read a time value from the title string in each frame, which should be preceded by 't=', as in the sample below.

A sample piece is included below:

```
MD of 2 waters, t= 0.0
    6
    1WATER  OW1    1   0.126   1.624   1.679  0.1227 -0.0580  0.0434
    1WATER  HW2    2   0.190   1.661   1.747  0.8085  0.3191 -0.7791
    1WATER  HW3    3   0.177   1.568   1.613 -0.9045 -2.6469  1.3180
    2WATER  OW1    4   1.275   0.053   0.622  0.2519  0.3140 -0.1734
    2WATER  HW2    5   1.337   0.002   0.680 -1.0641 -1.1349  0.0257
    2WATER  HW3    6   1.326   0.120   0.568  1.9427 -0.8216 -0.0244
   1.82060   1.82060   1.82060
```

Lines contain the following information (top to bottom):

- title string (free format string, optional time in ps after 't=')
- number of atoms (free format integer)
- one line for each atom (fixed format, see below)
- box vectors (free format, space separated reals), values: v1(x) v2(y) v3(z) v1(y) v1(z) v2(x) v2(z) v3(x) v3(y), the last 6 values may be omitted (they will be set to zero). GROMACS only supports boxes with v1(y)=v1(z)=v2(z)=0.

This format is fixed, ie. all columns are in a fixed position. Optionally (for now only yet with trjconv) you can write gro files with any number of decimal places, the format will then be n+5 positions with n decimal places (n+1 for velocities) in stead of 8 with 3 (with 4 for velocities). Upon reading, the precision will be inferred from the distance between the decimal points (which will be n+5). Columns contain the following information (from left to right):

- residue number (5 positions, integer)
- residue name (5 characters)
- atom name (5 characters)
- atom number (5 positions, integer)
- position (in nm, x y z in 3 columns, each 8 positions with 3 decimal places)
- velocity (in nm/ps (or km/s), x y z in 3 columns, each 8 positions with 4 decimal places)

Note that separate molecules or ions (e.g. water or Cl-) are regarded as residues. If you want to write such a file in your own program without using the GROMACS libraries you can use the following formats:

**C format**

`"%5d%-5s%5s%5d%8.3f%8.3f%8.3f%8.4f%8.4f%8.4f"`

**Fortran format**

`(i5,2a5,i5,3f8.3,3f8.4)`

**Pascal format**

This is left as an exercise for the user

Note that this is the format for writing, as in the above example fields may be written without spaces, and therefore can not be read with the same format statement in C.

### hdb

The hdb file extension stands for hydrogen database Such a file is needed by *gmx pdb2gmx* (page 281) when building hydrogen atoms that were either originally missing, or that were removed with `-ignh`.

### itp

The itp file extension stands for include topology. These files are included in topology files (with the *top* (page 584) extension).

### log

Logfiles are generated by some GROMACS programs and are usually in human-readable format. Use `more logfile`.

### m2p

The m2p file format contains input options for the *gmx xpm2ps* (page 357) program. All of these options are very easy to comprehend when you look at the PostScript(tm) output from *gmx xpm2ps* (page 357).

```
; Command line options of xpm2ps override the parameters in this␣
 ↪file
black&white             = no            ; Obsolete
titlefont               = Times-Roman   ; A PostScript Font
titlefontsize           = 20            ; Font size (pt)
legend                  = yes           ; Show the legend
legendfont              = Times-Roman   ; A PostScript Font
legendlabel             =               ; Used when there is none␣
 ↪in the .xpm
legend2label            =               ; Used when merging two xpm
 ↪'s
legendfontsize          = 14            ; Font size (pt)
xbox                    = 2.0           ; x-size of a matrix element
ybox                    = 2.0           ; y-size of a matrix element
matrixspacing           = 20.0          ; Space between 2 matrices
xoffset                 = 0.0           ; Between matrix and␣
 ↪bounding box
yoffset                 = 0.0           ; Between matrix and␣
```

(continues on next page)

(continued from previous page)

```
 ↪bounding box
x-major                  = 20            ; Major ticks on x axis␣
 ↪every .. frames
x-minor                  = 5             ; Id. Minor ticks
x-firstmajor             = 0             ; First frame for major tick
x-majorat0               = no            ; Major tick at first frame
x-majorticklen           = 8.0           ; x-majorticklength
x-minorticklen           = 4.0           ; x-minorticklength
x-label                  =               ; Used when there is none␣
 ↪in the .xpm
x-fontsize               = 16            ; Font size (pt)
x-font                   = Times-Roman   ; A PostScript Font
x-tickfontsize           = 10            ; Font size (pt)
x-tickfont               = Helvetica     ; A PostScript Font
y-major                  = 20
y-minor                  = 5
y-firstmajor             = 0
y-majorat0               = no
y-majorticklen           = 8.0
y-minorticklen           = 4.0
y-label                  =
y-fontsize               = 16
y-font                   = Times-Roman
y-tickfontsize           = 10
y-tickfont               = Helvetica
```

### mdp

See the user guide for a detailed description of the options.

Below is a sample mdp file. The ordering of the items is not important, but if you enter the same thing twice, the **last** is used (*gmx grompp* (page 230) gives you a note when overriding values). Dashes and underscores on the left hand side are ignored.

The values of the options are values for a 1 nanosecond MD run of a protein in a box of water.

**Note:** The parameters chosen (*e.g.,* short-range cutoffs) depend on the force field being used.

```
integrator               = md
dt                       = 0.002
nsteps                   = 500000

nstlog                   = 5000
nstenergy                = 5000
nstxout-compressed       = 5000

continuation             = yes
constraints              = all-bonds
constraint-algorithm     = lincs
```

(continues on next page)

(continued from previous page)

```
cutoff-scheme            = Verlet

coulombtype              = PME
rcoulomb                 = 1.0

vdwtype                  = Cut-off
rvdw                     = 1.0
DispCorr                 = EnerPres

tcoupl                   = V-rescale
tc-grps                  = Protein  SOL
tau-t                    = 0.1      0.1
ref-t                    = 300      300

pcoupl                   = Parrinello-Rahman
tau-p                    = 2.0
compressibility          = 4.5e-5
ref-p                    = 1.0
```

With this input *gmx grompp* (page 230) will produce a commented file with the default name `mdout.mdp`. That file will contain the above options, as well as all other options not explicitly set, showing their default values.

The mdp sample above was generated using mdpeditor. That tool can help generating mdp files according to best practices based on simple instructions.

### mtx

Files with the mtx file extension contain a matrix. The file format is identical to the *trr* (page 586) format. Currently this file format is only used for hessian matrices, which are produced with *gmx mdrun* (page 258) and read by *gmx nmeig* (page 269).

### ndx

The GROMACS index file (usually called index.ndx) contains some user definable sets of atoms. The file can be read by most analysis programs and by the preprocessor (*gmx grompp* (page 230)). Most of these programs create default index groups when no index file is supplied, so you only need to make an index file when you need special groups.

First the group name is written between square brackets. The following atom numbers may be spread out over as many lines as you like. The atom numbering starts at 1.

An example file is here:

```
[ Oxygen ]
1  4  7
[ Hydrogen ]
2  3  5  6
8  9
```

There are two groups, and total nine atoms. The first group **Oxygen** has 3 elements. The second group **Hydrogen** has 6 elements.

An index file generation tool is available: *gmx make_ndx* (page 255).

### n2t

This GROMACS file can be used to perform primitive translations between atom names found in structure files and the corresponding atom types. This is mostly useful for using utilities such as *gmx x2top* (page 355), but users should be aware that the knowledge in this file is extremely limited.

An example file (`share/top/gromos53a5.ff/atomname2type.n2t`) is here:

```
H        H     0.408   1.008   1   O      0.1
O        OA   -0.674  15.9994  2   C      0.14 H 0.1
C        CH3   0.000  15.035   1   C      0.15
C        CH0   0.266  12.011   4   C      0.15 C 0.15     C 0.15     O 0.
↪14
```

A short description of the file format follows:

- Column 1: Elemental symbol of the atom/first character in the atom name.
- Column 2: The atom type to be assigned.
- Column 3: The charge to be assigned.
- Column 4: The mass of the atom.
- Column 5: The number N of other atoms to which this atom is bonded. The number of fields that follow are related to this number; for each atom, an elemental symbol and the reference distance for its bond length.
- Columns 6-onward: The elemental symbols and reference bond lengths for N connections (column 5) to the atom being assigned parameters (column 1). The reference bond lengths have a tolerance of +/- 10% from the value specified in this file. Any bond outside this tolerance will not be recognized as being connected to the atom being assigned parameters.

### out

Files with the out file extension contain generic output. As it is not possible to categorize all data file formats, GROMACS has a generic file format called out of which no format is given.

### pdb

Files with the *pdb* (page 581) extension are molecular structure files in the protein databank file format. The protein databank file format describes the positions of atoms in a molecular structure. Coordinates are read from the ATOM and HETATM records, until the file ends or an ENDMDL record is encountered. GROMACS programs can read and write a simulation box in the CRYST1 entry. The pdb format can also be used as a trajectory format: several structures, separated by ENDMDL, can be read from or written to one file.

### Example

A pdb file should look like this:

```
ATOM      1  H1  LYS     1      14.260   6.590  34.480  1.00  0.00
ATOM      2  H2  LYS     1      13.760   5.000  34.340  1.00  0.00
ATOM      3  N   LYS     1      14.090   5.850  33.800  1.00  0.00
ATOM      4  H3  LYS     1      14.920   5.560  33.270  1.00  0.00
...
...
```

### rtp

The rtp file extension stands for residue topology. Such a file is needed by *gmx pdb2gmx* (page 281) to make a GROMACS topology for a protein contained in a *pdb* (page 581) file. The file contains the default interaction type for the 4 bonded interactions and residue entries, which consist of atoms and optionally bonds, angles dihedrals and impropers. Parameters can be added to bonds, angles, dihedrals and impropers, these parameters override the standard parameters in the *itp* (page 578) files. This should only be used in special cases. Instead of parameters a string can be added for each bonded interaction, the string is copied to the *top* (page 584) file, this is used for the GROMOS96 forcefield.

*gmx pdb2gmx* (page 281) automatically generates all angles, this means that the `[angles]` field is only useful for overriding *itp* (page 578) parameters.

*gmx pdb2gmx* (page 281) automatically generates one proper dihedral for every rotatable bond, preferably on heavy atoms. When the `[dihedrals]` field is used, no other dihedrals will be generated for the bonds corresponding to the specified dihedrals. It is possible to put more than one dihedral on a rotatable bond.

*gmx pdb2gmx* (page 281) sets the number exclusions to 3, which means that interactions between atoms connected by at most 3 bonds are excluded. Pair interactions are generated for all pairs of atoms which are separated by 3 bonds (except pairs of hydrogens). When more interactions need to be excluded, or some pair interactions should not be generated, an `[exclusions]` field can be added, followed by pairs of atom names on separate lines. All non-bonded and pair interactions between these atoms will be excluded.

A sample is included below.

```
[ bondedtypes ]  ; mandatory
; bonds  angles  dihedrals  impropers
     1       1          1          2  ; mandatory

[ GLY ]  ; mandatory

 [ atoms ]  ; mandatory
; name  type  charge  chargegroup
     N     N  -0.280     0
     H     H   0.280     0
    CA   CH2   0.000     1
     C     C   0.380     2
     O     O  -0.380     2
```

(continues on next page)

(continued from previous page)

```
 [ bonds ]  ; optional
;atom1 atom2      b0      kb
     N     H
     N    CA
    CA     C
     C     O
    -C     N

 [ exclusions ]  ; optional
;atom1 atom2

 [ angles ]  ; optional
;atom1 atom2 atom3    th0    cth

 [ dihedrals ]  ; optional
;atom1 atom2 atom3 atom4   phi0     cp   mult

 [ impropers ]  ; optional
;atom1 atom2 atom3 atom4     q0     cq
     N    -C    CA     H
    -C   -CA     N    -O


[ ZN ]
 [ atoms ]
    ZN    ZN   2.000     0
```

### r2b

The r2b file translates the residue names for residues that have different names in different force fields, or have different names depending on their protonation states.

### tdb

tdb files contain the information about amino acid termini that can be placed at the end of a polypeptide chain.

### tex

We use **LaTeX** for *document* processing. Although the input is not so user friendly, it has some advantages over *word* processors.

- **LaTeX** knows a lot about formatting, probably much more than you.
- The input is clear, you always know what you are doing
- It makes anything from letters to a thesis
- Much more...

### tng

Files with the `.tng` file extension can contain all kinds of data related to the trajectory of a simulation. For example, it might contain coordinates, velocities, forces and/or energies. Various *mdp* (page 579) file options control which of these are written by *gmx mdrun* (page 258), whether data is written with compression, and how lossy that compression can be. This file is in portable binary format and can be read with *gmx dump* (page 205).

*gmx dump* (page 205) -f traj.tng

or if you're not such a fast reader:

```
gmx dump -f traj.tng | less
```

You can also get a quick look in the contents of the file (number of frames etc.) using:

*gmx check* (page 165) -f traj.tng

### top

The top file extension stands for topology. It is an ascii file which is read by *gmx grompp* (page 230) which processes it and creates a binary topology (*tpr* (page 586) file).

A sample file is included below:

```
;
; Example topology file
;
[ defaults ]
; nbfunc        comb-rule       gen-pairs       fudgeLJ fudgeQQ
  1             1               no              1.0     1.0

; The force field files to be included
#include "rt41c5.itp"

[ moleculetype ]
; name  nrexcl
Urea         3

[ atoms ]
;   nr    type   resnr  residu    atom    cgnr  charge
     1       C       1    UREA      C1       1   0.683
     2       O       1    UREA      O2       1  -0.683
     3      NT       1    UREA      N3       2  -0.622
     4       H       1    UREA      H4       2   0.346
     5       H       1    UREA      H5       2   0.276
     6      NT       1    UREA      N6       3  -0.622
     7       H       1    UREA      H7       3   0.346
     8       H       1    UREA      H8       3   0.276

[ bonds ]
;  ai    aj funct           c0           c1
```

(continues on next page)

(continued from previous page)

```
    3      4      1 1.000000e-01 3.744680e+05
    3      5      1 1.000000e-01 3.744680e+05
    6      7      1 1.000000e-01 3.744680e+05
    6      8      1 1.000000e-01 3.744680e+05
    1      2      1 1.230000e-01 5.020800e+05
    1      3      1 1.330000e-01 3.765600e+05
    1      6      1 1.330000e-01 3.765600e+05

[ pairs ]
;  ai    aj funct           c0           c1
    2      4      1 0.000000e+00 0.000000e+00
    2      5      1 0.000000e+00 0.000000e+00
    2      7      1 0.000000e+00 0.000000e+00
    2      8      1 0.000000e+00 0.000000e+00
    3      7      1 0.000000e+00 0.000000e+00
    3      8      1 0.000000e+00 0.000000e+00
    4      6      1 0.000000e+00 0.000000e+00
    5      6      1 0.000000e+00 0.000000e+00

[ angles ]
;  ai    aj    ak funct           c0           c1
    1      3      4      1 1.200000e+02 2.928800e+02
    1      3      5      1 1.200000e+02 2.928800e+02
    4      3      5      1 1.200000e+02 3.347200e+02
    1      6      7      1 1.200000e+02 2.928800e+02
    1      6      8      1 1.200000e+02 2.928800e+02
    7      6      8      1 1.200000e+02 3.347200e+02
    2      1      3      1 1.215000e+02 5.020800e+02
    2      1      6      1 1.215000e+02 5.020800e+02
    3      1      6      1 1.170000e+02 5.020800e+02

[ dihedrals ]
;  ai    aj    ak    al funct           c0           c1           c2
    2      1      3      4      1 1.800000e+02 3.347200e+01 2.000000e+00
    6      1      3      4      1 1.800000e+02 3.347200e+01 2.000000e+00
    2      1      3      5      1 1.800000e+02 3.347200e+01 2.000000e+00
    6      1      3      5      1 1.800000e+02 3.347200e+01 2.000000e+00
    2      1      6      7      1 1.800000e+02 3.347200e+01 2.000000e+00
    3      1      6      7      1 1.800000e+02 3.347200e+01 2.000000e+00
    2      1      6      8      1 1.800000e+02 3.347200e+01 2.000000e+00
    3      1      6      8      1 1.800000e+02 3.347200e+01 2.000000e+00

[ dihedrals ]
;  ai    aj    ak    al funct           c0           c1
    3      4      5      1      2 0.000000e+00 1.673600e+02
    6      7      8      1      2 0.000000e+00 1.673600e+02
    1      3      6      2      2 0.000000e+00 1.673600e+02

; Include SPC water topology
```

(continues on next page)

(continued from previous page)

```
#include "spc.itp"

[ system ]
Urea in Water

[ molecules ]
Urea    1
SOL     1000
```

### tpr

The tpr file extension stands for portable binary run input file. This file contains the starting structure of your simulation, the molecular topology and all the simulation parameters. Because this file is in binary format it cannot be read with a normal editor. To read a portable binary run input file type:

*gmx dump* (page 205) -s topol.tpr

or if you're not such a fast reader:

```
gmx dump -s topol.tpr | less
```

You can also compare two tpr files using:

*gmx check* (page 165) -s1 top1 -s2 top2 | less

### trr

Files with the trr file extension contain the trajectory of a simulation. In this file all the coordinates, velocities, forces and energies are printed as you told GROMACS in your mdp file. This file is in portable binary format and can be read with *gmx dump* (page 205):

```
gmx dump -f traj.trr
```

or if you're not such a fast reader:

```
gmx dump -f traj.trr | less
```

You can also get a quick look in the contents of the file (number of frames etc.) using:

% *gmx check* (page 165) -f traj.trr

### vsd

The vsd file contains the information on how to place virtual sites on a number of different molecules in a force field.

### xdr

GROMACS uses the XDR file format to store things like coordinate files internally.

### xpm

The GROMACS xpm file format is compatible with the XPixMap format and is used for storing matrix data. Thus GROMACS xpm files can be viewed directly with programs like XV. Alternatively, they can be imported into GIMP and scaled to 300 DPI, using strong antialiasing for font and graphics. The first matrix data line in an xpm file corresponds to the last matrix row. In addition to the XPixMap format, GROMACS xpm files may contain extra fields. The information in these fields is used when converting an xpm file to EPS with *gmx xpm2ps* (page 357). The optional extra field are:

- Before the `gv_xpm` declaration: `title`, `legend`, `x-label`, `y-label` and `type`, all followed by a string. The `legend` field determines the legend title. The `type` field must be followed by `"continuous"` or `"discrete"`, this determines which type of legend will be drawn in an EPS file, the default type is continuous.
- The xpm colormap entries may be followed by a string, which is a label for that color.
- Between the colormap and the matrix data, the fields `x-axis` and/or `y-axis` may be present followed by the tick-marks for that axis.

The example GROMACS xpm file below contains all the extra fields. The C-comment delimiters and the colon in the extra fields are optional.

```
/* XPM */
/* This matrix is generated by gmx rms. */
/* title:   "Backbone RMSD matrix" */
/* legend:  "RMSD (nm)" */
/* x-label: "Time (ps)" */
/* y-label: "Time (ps)" */
/* type:    "Continuous" */
static char * gv_xpm[] = {
"13 13   6 1",
"A  c #FFFFFF " /* "0" */,
"B  c #CCCCCC " /* "0.0399" */,
"C  c #999999 " /* "0.0798" */,
"D  c #666666 " /* "0.12" */,
"E  c #333333 " /* "0.16" */,
"F  c #000000 " /* "0.2" */,
/* x-axis:  0 40 80 120 160 200 240 280 320 360 400 440 480 */
/* y-axis:  0 40 80 120 160 200 240 280 320 360 400 440 480 */
"FEDDDDCCCCCBA",
"FEDDDCCCCBBAB",
"FEDDDCCCCBABC",
"FDDDDCCCCABBC",
"EDDCCCCBACCCC",
"EDCCCCBABCCCC",
"EDCCCBABCCCCC",
"EDCCBABCCCCCD",
"EDCCABCCCDDDD",
```

(continues on next page)

(continued from previous page)

```
"ECCACCCCCDDDD",
"ECACCCCCDDDDD",
"DACCDDDDDDEEE",
"ADEEEEEEEFFFF"
```

### xtc

The xtc format is a **portable** format for trajectories. It uses the *xdr* routines for writing and reading data which was created for the Unix NFS system. The trajectories are written using a reduced precision algorithm which works in the following way: the coordinates (in nm) are multiplied by a scale factor, typically 1000, so that you have coordinates in pm. These are rounded to integer values. Then several other tricks are performed, for instance making use of the fact that atoms close in sequence are usually close in space too (e.g. a water molecule). To this end, the *xdr* library is extended with a special routine to write 3-D float coordinates. The routine was originally written by Frans van Hoesel as part of an Europort project. An updated version of it can be obtained through this link.

All the data is stored using calls to *xdr* routines.

**int magic**
: A magic number, for the current file version its value is 1995.

**int natoms**
: The number of atoms in the trajectory.

**int step**
: The simulation step.

**float time**
: The simulation time.

**float box[3][3]**
: The computational box which is stored as a set of three basis vectors, to allow for triclinic PBC. For a rectangular box the box edges are stored on the diagonal of the matrix.

**3dfcoord x[natoms]**
: The coordinates themselves stored in reduced precision. Please note that when the number of atoms is smaller than 9 no reduced precision is used.

### Using xtc in your C++ programs

It is possible to write your own analysis tools to take advantage of the compressed .xtc format files: see the `template.cpp` file in the `share/gromacs/template` directory of your installation for an example and https://manual.gromacs.org/current/doxygen/html-full/page_analysistemplate.xhtml for documentation.

To read and write xtc files the following routines are available via `xtcio.h`:

```
/* All functions return 1 if successful, 0 otherwise */

struct t_fileio* open_xtc(const char* filename, const char* mode);
/* Open a file for xdr I/O */
```

(continues on next page)

(continued from previous page)

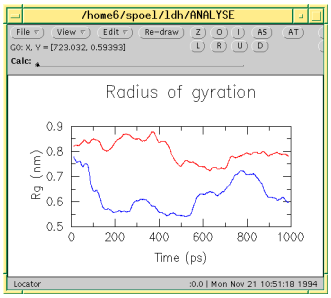
```
void close_xtc(struct t_fileio* fio);
/* Close the file for xdr I/O */

int read_first_xtc(struct t_fileio* fio,
                   int*             natoms,
                   int64_t*         step,
                   real*            time,
                   matrix           box,
                   rvec**           x,
                   real*            prec,
                   gmx_bool*        bOK);
/* Open xtc file, read xtc file first time, allocate memory for x */

int read_next_xtc(struct t_fileio* fio, int natoms, int64_t* step,␣
↪real* time, matrix box, rvec* x, real* prec, gmx_bool* bOK);
/* Read subsequent frames */

int write_xtc(struct t_fileio* fio, int natoms, int64_t step, real␣
↪time, const rvec* box, const rvec* x, real prec);
/* Write a frame to xtc file */
```

To use the library function include `"gromacs/fileio/xtcio.h"` in your file and link with `-lgromacs`.

### xvg

Almost all output from GROMACS analysis tools is ready as input for Grace, formerly known as Xmgr. We use Grace, because it is very flexible, and it is also free software. It produces PostScript(tm) output, which is very suitable for inclusion in eg. LaTeX documents, but also for other word processors.

A sample Grace session with GROMACS data is shown below:



## 5.8 Special Topics

This section covers some of the more specialized topics concerning the use of GROMACS for specific scientific problems.

### 5.8.1 Free energy implementation

For free energy calculations, there are two things that must be specified; the end states, and the pathway connecting the end states. The end states can be specified in two ways. The most straightforward is through the specification of end states in the topology file. Most potential forms support both an $A$ state and a $B$ state. Whenever both states are specified, the $A$ state corresponds to the initial free energy state, and the $B$ state corresponds to the final state.

In some cases, the end state can also be defined in some cases without altering the topology, solely through the *mdp* (page 579) file, through the use of the `couple-moltype`, `couple-lambda0`, `couple-lambda1`, and `couple-intramol` *mdp* (page 579) keywords. Any molecule type selected in `couple-moltype` will automatically have a $B$ state implicitly constructed (and the $A$ state redefined) according to the `couple-lambda` keywords. `couple-lambda0` and `couple-lambda1` define the non-bonded parameters that are present in the $A$ state (`couple-lambda0`) and the $B$ state (`couple-lambda1`). The choices are `q`, `vdw`, and `vdw-q`; these indicate the Coulombic, van der Waals, or both parameters that are turned on in the respective state.

Once the end states are defined, then the path between the end states has to be defined. This path is defined solely in the .mdp file. Starting in 4.6, $\lambda$ is a vector of components, with Coulombic, van der Waals, bonded, restraint, and mass components all able to be adjusted independently. This makes it possible to turn off the Coulombic term linearly, and then the van der Waals using soft core, all in the same simulation. This is especially useful for replica exchange or expanded ensemble simulations, where it is important to sample all the way from interacting to non-interacting states in the same simulation to improve sampling.

`fep-lambdas` is the default array of $\lambda$ values ranging from 0 to 1. All of the other lambda arrays use the values in this array if they are not specified. The previous behavior, where the pathway is controlled by a single $\lambda$ variable, can be preserved by using only `fep-lambdas` to define the pathway.

Fig. 5.37 shows an example of different lambda arrays. There, first the Coulombic terms are reduced, then the van der Waals terms, changing bonded at the same time rate as the van der Waals, but changing the restraints throughout the first two-thirds of the simulation. The corresponding $\lambda$ vector is given here:

```
coul-lambdas            = 0.0 0.2 0.5 1.0 1.0 1.0 1.0 1.0 1.0 1.0
vdw-lambdas             = 0.0 0.0 0.0 0.0 0.4 0.5 0.6 0.7 0.8 1.0
bonded-lambdas          = 0.0 0.0 0.0 0.0 0.4 0.5 0.6 0.7 0.8 1.0
restraint-lambdas       = 0.0 0.0 0.1 0.2 0.3 0.5 0.7 1.0 1.0 1.0
```

This is also equivalent to:

```
fep-lambdas             = 0.0 0.0 0.0 0.0 0.4 0.5 0.6 0.7 0.8 1.0
coul-lambdas            = 0.0 0.2 0.5 1.0 1.0 1.0 1.0 1.0 1.0 1.0
restraint-lambdas       = 0.0 0.0 0.1 0.2 0.3 0.5 0.7 1.0 1.0 1.0
```

The `fep-lambda` array, in this case, is being used as the default to fill in the bonded and van der Waals $\lambda$ arrays. Usually, it's best to fill in all arrays explicitly, just to make sure things are properly assigned.

If you want to turn on only restraints going from $A$ to $B$, then it would be:

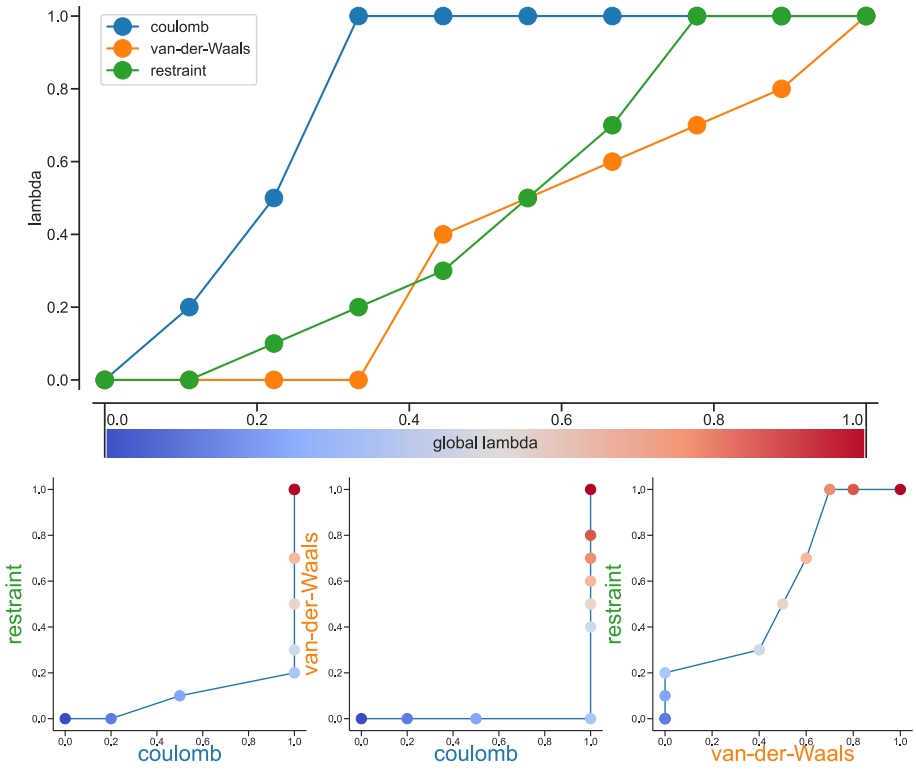



Fig. 5.37: Separate $\lambda$ values for Coulomb, van-der-Waals and restraint interactions.

```
restraint-lambdas       = 0.0 0.1 0.2 0.4 0.6 1.0
```

and all of the other components of the $\lambda$ vector would be left in the $A$ state.

To compute free energies with a vector $\lambda$ using thermodynamic integration, then the TI equation becomes vector equation:

$$\Delta F = \int \langle \nabla H \rangle \cdot d\vec{\lambda} \tag{5.300}$$

or for finite differences:

$$\Delta F \approx \int \sum \langle \nabla H \rangle \cdot \Delta\lambda \tag{5.301}$$

The external pymbar script can compute this integral automatically from the GROMACS `dhdl.xvg` output.

### 5.8.2 Potential of mean force

A potential of mean force (PMF) is a potential that is obtained by integrating the mean force from an ensemble of configurations. In GROMACS, there are several different methods to calculate the mean force. Each method has its limitations, which are listed below.

- **pull code:** between the centers of mass of molecules or groups of molecules.
- **AWH code:** currently acts on coordinates provided by the pull code or the free-energy lambda parameter.
- **free-energy code with harmonic bonds or constraints:** between single atoms.

- **free-energy code with position restraints:** changing the conformation of a relatively immobile group of atoms.
- **pull code in limited cases:** between groups of atoms that are part of a larger molecule for which the bonds are constrained with SHAKE or LINCS. If the pull group if relatively large, the pull code can be used.

The pull and free-energy code a described in more detail in the following two sections.

#### Entropic effects

When a distance between two atoms or the centers of mass of two groups is constrained or restrained, there will be a purely entropic contribution to the PMF due to the rotation of the two groups *134* (page 690). For a system of two non-interacting masses the potential of mean force is:

$$V_{pmf}(r) = -(n_c - 1)k_B T \log(r) \tag{5.302}$$

where $n_c$ is the number of dimensions in which the constraint works (i.e. $n_c = 3$ for a normal constraint and $n_c = 1$ when only the $z$-direction is constrained). Whether one needs to correct for this contribution depends on what the PMF should represent. When one wants to pull a substrate into a protein, this entropic term indeed contributes to the work to get the substrate into the protein. But when calculating a PMF between two solutes in a solvent, for the purpose of simulating without solvent, the entropic contribution should be removed. **Note** that this term can be significant; when at 300K the distance is halved, the contribution is 3.5 kJ mol$^{-1}$.

### 5.8.3 Non-equilibrium pulling

When the distance between two groups is changed continuously, work is applied to the system, which means that the system is no longer in equilibrium. Although in the limit of very slow pulling the system is again in equilibrium, for many systems this limit is not reachable within reasonable computational time. However, one can use the Jarzynski relation *135* (page 690) to obtain the equilibrium free-energy difference $\Delta G$ between two distances from many non-equilibrium simulations:

$$\Delta G_{AB} = -k_B T \log \left\langle e^{-\beta W_{AB}} \right\rangle_A \tag{5.303}$$

where $W_{AB}$ is the work performed to force the system along one path from state A to B, the angular bracket denotes averaging over a canonical ensemble of the initial state A and $\beta = 1/k_B T$.

### 5.8.4 Collective variables: the pull code

The pull code applies forces or constraints on collective variables (sometimes referred to as reaction coordinates). The basic collective pull coordinates are a distance, angle and dihedral angle between centers of mass of groups atoms, the so-called "pull groups". More complex collective variables can be defined using *The transformation pull coordinate* (page 596). A pull group can be part of one or more pull coordinates. Furthermore, a coordinate can also operate on a single group and an absolute reference position in space. The distance between a pair of groups can be determined in 1, 2 or 3 dimensions, or can be along a user-defined vector. The reference distance can be constant or can change linearly with time. Normally all atoms are weighted by their mass, but an additional weighting factor can also be used.

Several different pull types, i.e. ways to apply the pull force, are supported, and in all cases the reference distance can be constant or linearly changing with time.

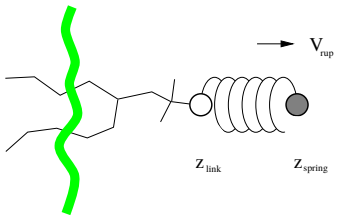



Fig. 5.38: Schematic picture of pulling a lipid out of a lipid bilayer with umbrella pulling. $V_{rup}$ is the velocity at which the spring is retracted, $Z_{link}$ is the atom to which the spring is attached and $Z_{spring}$ is the location of the spring.

1. **Umbrella pulling** A harmonic potential is applied between the centers of mass of two groups. Thus, the force is proportional to the displacement.
2. **Constraint pulling** The distance between the centers of mass of two groups is constrained. The constraint force can be written to a file. This method uses the SHAKE algorithm but only needs 1 iteration to be exact if only two groups are constrained.
3. **Constant force pulling** A constant force is applied between the centers of mass of two groups. Thus, the potential is linear. In this case there is no reference distance of pull rate.
4. **Flat bottom pulling** Like umbrella pulling, but the potential and force are zero for coordinate values below (`pull-coord?-type = flat-bottom`) or above (`pull-coord?-type = flat-bottom-high`) a reference value. This is useful for restraining e.g. the distance between two molecules to a certain region.
5. **External potential** This takes the potential acting on the reaction coordinate from another module. Current only the Accelerated Weight Histogram method (see sec. *Adaptive biasing with AWH* (page 598)) is supported, which provides adaptive biasing of pull coordinates.

In addition, there are different types of reaction coordinates, so-called pull geometries. These are set with the *mdp* (page 579) option `pull-coord?-geometry`.

### Definition of the center of mass

In GROMACS, there are three ways to define the center of mass of a group. The standard way is a "plain" center of mass, possibly with additional weighting factors. With periodic boundary conditions it is no longer possible to uniquely define the center of mass of a group of atoms. Therefore, a reference atom is used. For determining the center of mass, for all other atoms in the group, the closest periodic image to the reference atom is used. This uniquely defines the center of mass. By default, the middle (determined by the order in the topology) atom is used as a reference atom, but the user can also select any other atom if it would be closer to center of the group.

When there are large pull groups, such as a lipid bilayer, `pull-pbc-ref-prev-step-com` can be used to avoid potential large movements of the center of mass in case that atoms in the pull group move so much that the reference atom is too far from the intended center of mass. With this option enabled the center of mass from the previous step is used, instead of the position of the reference atom, to determine the reference position. The position of the reference atom is still used for the first step. For large pull groups it is important to select a reference atom that is close to the intended center of mass, i.e. do not use `pull-group?-pbcatom = 0`.

For a layered system, for instance a lipid bilayer, it may be of interest to calculate the PMF of a lipid as function of its distance from the whole bilayer. The whole bilayer can be taken as reference group in that case, but it might also be of interest to define the reaction coordinate for the PMF more locally. The *mdp* (page 579) option `pull-coord?-geometry = cylinder` does not use all the atoms of the reference group, but instead dynamically only those within a cylinder with radius `pull-cylinder-r` around the pull vector going through the pull group. This only works for distances defined in one dimension, and the cylinder is oriented with its long axis along this one dimension. To avoid jumps in the pull force, contributions of atoms are weighted as a function of distance (in addition to the mass weighting), for atom $i$:

$$\begin{aligned} w_i(r_i < r_{\mathrm{cyl}}) = &\quad 1 - 2\left(\frac{r_i}{r_{\mathrm{cyl}}}\right)^2 + \left(\frac{r_i}{r_{\mathrm{cyl}}}\right)^4 \\ w_i(r_i \geq r_{\mathrm{cyl}}) = &\quad 0 \end{aligned} \tag{5.304}$$

Note that the radial dependence on the weight causes a radial force on both cylinder group and the other pull group:

$$\begin{aligned} F_i^{\mathrm{radial}}(r_i < r_{\mathrm{cyl}}) = &\quad F^{\mathrm{pull}} a_i \frac{1}{\sum_i w_i} \frac{4}{r_{\mathrm{cyl}}^4} r_i (r_i^2 - r_{\mathrm{cyl}}^2) \\ F_i^{\mathrm{radial}}(r_i \geq r_{\mathrm{cyl}}) = &\quad 0 \end{aligned} \tag{5.305}$$

where $F^{\mathrm{pull}}$ is the pull force working between the groups and $a_i$ is the axial distance of atom $i$ to the center of mass of the cylinder group. This is an undesirable, but unavoidable effect. To minimize this effect, the cylinder radius should be chosen sufficiently large. The effective mass is 0.47 times that of a cylinder with uniform weights and equal to the mass of uniform cylinder of 0.79 times the radius.

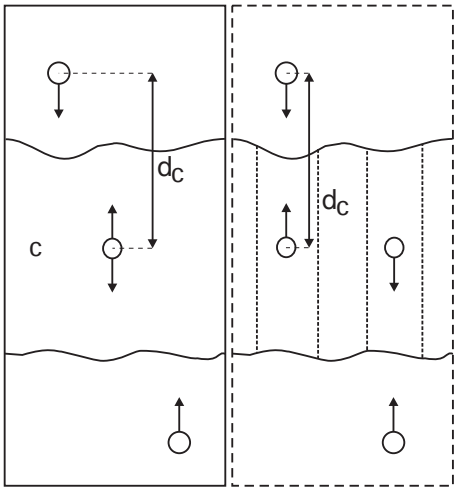



Fig. 5.39: Comparison of a plain center of mass reference group versus a cylinder reference group applied to interface systems. C is the reference group. The circles represent the center of mass of two groups plus the reference group, $d_c$ is the reference distance.

For a group of molecules in a periodic system, a plain reference group might not be well-defined. An example is a water slab that is connected periodically in $x$ and $y$, but has two liquid-vapor interfaces along $z$. In such a setup, water molecules can evaporate from the liquid and they will move through the vapor, through the periodic boundary, to the other interface. Such a system is inherently periodic and there is no proper way of defining a "plain" center of mass along $z$. A proper solution is to using a cosine shaped weighting profile for all atoms in the reference group. The profile is a cosine with a single period in the unit cell. Its phase is optimized to give the maximum sum of weights, including mass weighting. This provides a unique and continuous reference position that is nearly identical to the plain center of

mass position in case all atoms are all within a half of the unit-cell length. See ref *136* (page 690) for details.

When relative weights $w_i$ are used during the calculations, either by supplying weights in the input or due to cylinder geometry or due to cosine weighting, the weights need to be scaled to conserve momentum:

$$w_i' = w_i \sum_{j=1}^{N} w_j \, m_j \Big/ \sum_{j=1}^{N} w_j^2 \, m_j \tag{5.306}$$

where $m_j$ is the mass of atom $j$ of the group. The mass of the group, required for calculating the constraint force, is:

$$M = \sum_{i=1}^{N} w_i' \, m_i \tag{5.307}$$

The definition of the weighted center of mass is:

$$\mathbf{r}_{com} = \sum_{i=1}^{N} w_i' \, m_i \, \mathbf{r}_i \Big/ M \tag{5.308}$$

From the centers of mass the AFM, constraint, or umbrella force $\mathbf{F}_{com}$ on each group can be calculated. The force on the center of mass of a group is redistributed to the atoms as follows:

$$\mathbf{F}_i = \frac{w_i' \, m_i}{M} \, \mathbf{F}_{com} \tag{5.309}$$

### Definition of the pull direction

The most common setup is to pull along the direction of the vector containing the two pull groups, this is selected with `pull-coord?-geometry = distance`. You might want to pull along a certain vector instead, which is selected with `pull-coord?-geometry = direction`. But this can cause unwanted torque forces in the system, unless you pull against a reference group with (nearly) fixed orientation, e.g. a membrane protein embedded in a membrane along x/y while pulling along z. If your reference group does not have a fixed orientation, you should probably use `pull-coord?-geometry = direction-relative`, see Fig. 5.40. Since the potential now depends on the coordinates of two additional groups defining the orientation, the torque forces will work on these two groups.

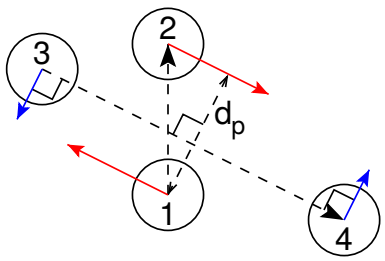



Fig. 5.40: The pull setup for geometry `direction-relative`. The "normal" pull groups are 1 and 2. Groups 3 and 4 define the pull direction and thus the direction of the normal pull forces (red). This leads to reaction forces (blue) on groups 3 and 4, which are perpendicular to the pull direction. Their magnitude is given by the "normal" pull force times the ratio of $d_p$ and the distance between groups 3 and 4.

### Definition of the angle and dihedral pull geometries

Four pull groups are required for `pull-coord?-geometry = angle`. In the same way as for geometries with two groups, each consecutive pair of groups $i$ and $i+1$ define a vector connecting the COMs of groups $i$ and $i+1$. The angle is defined as the angle between the two resulting vectors. E.g., the *mdp* (page 579) option `pull-coord?-groups = 1 2 2 4` defines the angle between the vector from the COM of group 1 to the COM of group 2 and the vector from the COM of group 2 to the COM of group 4. The angle takes values in the closed interval [0, 180] deg. For `pull-coord?-geometry = angle-axis` the angle is defined with respect to a reference axis given by `pull-coord?-vec` and only two groups need to be given. The dihedral geometry requires six pull groups. These pair up in the same way as described above and so define three vectors. The dihedral angle is defined as the angle between the two planes spanned by the two first and the two last vectors. Equivalently, the dihedral angle can be seen as the angle between the first and the third vector when these vectors are projected onto a plane normal to the second vector (the axis vector). As an example, consider a dihedral angle involving four groups: 1, 5, 8 and 9. Here, the *mdp* (page 579) option `pull-coord?-groups = 8 1 1 5 5 9` specifies the three vectors that define the dihedral angle: the first vector is the COM distance vector from group 8 to 1, the second vector is the COM distance vector from group 1 to 5, and the third vector is the COM distance vector from group 5 to 9. The dihedral angle takes values in the interval (-180, 180] deg and has periodic boundaries.

### The transformation pull coordinate

The transformation pull coordinate is a "meta" pull coordinate that can be used to define more complex collective variables. It can transform one or more other pull coordinates, as well as time, using an arbitrary mathematical expression. This is a powerful tool for generating complex collective variables. A simple example is a contact coordinate using a non-linear transformation of a distance. More complex examples are a (non-)linear combination of two or more pull coordinates or a sum of contacts.

Typically, the force constant for pull coordinate(s) the transformation coordinates acts on should be zero. This avoids unintended addition of direct forces on the pull coordinate(s) to the indirect forces from the transition pull coordinate. This is not a requirement, but having both a direct and indirect, from the tranformation coordinate, force working on them is almost never desirable. If the transformation is a linear combination of multiple distances, it is useful to normalize the coefficients such that the transformation coordinate also has units of nanometer. That makes both the choice of the force constant and the interpretation easier.

Here are two examples of pull sections of the *mdp* (page 579) input that use a tranformation coordinate setups. The first is a contact reaction coordinate that is 1 at contact and 0 at larger distances:

```
pull                    = yes
pull-ngroups            = 2
pull-ncoords            = 2

pull-group1-name        = groupA
pull-group2-name        = groupB

pull-coord1-type        = umbrella
pull-coord1-geometry    = distance
pull-coord1-groups      = 1 2
pull-coord1-dim         = Y Y Y
pull-coord1-k           = 0      ; avoid forces working directly␣
```

(continues on next page)

(continued from previous page)

```
 ↪on this distance

pull-coord2-type         = umbrella
pull-coord2-geometry     = transformation
pull-coord2-expression   = 1/(1 + exp(50*(x1 - 1.8*0.3)))  ; x1␣
 ↪refers to the value of coord1
pull-coord2-init         = 1      ; this restrains the distance to␣
 ↪having the contact
pull-coord2-k            = 100
```

The second example is an average of two distances:

```
pull                     = yes
pull-ngroups             = 4
pull-ncoords             = 3

pull-group1-name         = groupA
pull-group2-name         = groupB
pull-group3-name         = groupC
pull-group4-name         = groupD

pull-coord1-type         = umbrella
pull-coord1-geometry     = distance
pull-coord1-groups       = 1 2
pull-coord1-dim          = Y Y Y
pull-coord1-k            = 0      ; avoid forces working directly␣
 ↪on this distance

pull-coord2-type         = umbrella
pull-coord2-geometry     = distance
pull-coord2-groups       = 3 4
pull-coord2-dim          = Y Y Y
pull-coord2-k            = 0      ; avoid forces working directly␣
 ↪on this distance

pull-coord3-type         = umbrella
pull-coord3-geometry     = transformation
pull-coord3-expression   = 0.5*(x1 + x2)  ; x1 and x2 refer to the␣
 ↪value of coord1 and coord2
pull-coord3-init         = 0.8    ; restrains the average distance␣
 ↪to 0.8 nm
pull-coord3-k            = 1000
```

### Limitations

There is one theoretical limitation: strictly speaking, constraint forces can only be calculated between groups that are not connected by constraints to the rest of the system. If a group contains part of a molecule of which the bond lengths are constrained, the pull constraint and LINCS or SHAKE bond constraint algorithms should be iterated simultaneously. This is not done in GROMACS. This means that for simulations with `constraints = all-bonds` in the *mdp* (page 579) file pulling is, strictly speaking, limited to whole molecules or groups of molecules. In some cases this limitation can be avoided by using the free energy code, see sec. *Calculating a PMF using the free-energy code* (page 624). In practice, the errors caused by not iterating the two constraint algorithms can be negligible when the pull group consists of a large amount of atoms and/or the pull force is small. In such cases, the constraint correction displacement of the pull group is small compared to the bond lengths.

## 5.8.5 Adaptive biasing with AWH

The accelerated weight histogram method *185* (page 692) *137* (page 690) calculates the PMF along a reaction coordinate by adding an adaptively determined biasing potential. AWH flattens free energy barriers along the reaction coordinate by applying a history-dependent potential to the system that "fills up" free energy minima. This is similar in spirit to other adaptive biasing potential methods, e.g. the Wang-Landau *138* (page 690), local elevation *139* (page 690) and metadynamics *140* (page 690) methods. The initial sampling stage of AWH makes the method robust against the choice of input parameters. Furthermore, the target distribution along the reaction coordinate may be chosen freely.

### Basics of the method

The AWH method can act on two different types of reaction coordinates. It can work directly on a discrete reaction coordinate $\lambda$ in case this is the free-energy coupling parameter *187* (page 692). And it can act on reaction coordinates that are (continuous) functions of the coordinates: $\xi(x)$. In this case AWH acts on a *reference coordinate* $\lambda$ which takes discrete values and is coupled to $\xi(x)$ using an umbrella function $Q$. We will now describe the method for the most general case. When acting directly on $\lambda$, the function $Q$ is zero. The fundamentals of the method are based on the connection between atom coordinates and $\lambda$ and are established through the extended ensemble *68* (page 686),

$$P(x,\lambda) = \frac{1}{\mathcal{Z}} e^{g(\lambda) - Q(\xi(x),\lambda) - V(x)}, \tag{5.310}$$

where $g(\lambda)$ is a bias function (a free variable) and $V(x)$ is the unbiased potential energy of the system. The distribution along $\lambda$ can be tuned to be any predefined *target distribution* $\rho(\lambda)$ (often chosen to be flat) by choosing $g(\lambda)$ wisely. This is evident from

$$P(\lambda) = \int P(x,\lambda) dx = \frac{1}{\mathcal{Z}} e^{g(\lambda)} \int e^{-Q(\xi(x),\lambda) - V(x)} dx \equiv \frac{1}{\mathcal{Z}} e^{g(\lambda) - F(\lambda)}, \tag{5.311}$$

where $F(\lambda)$ is the free energy

$$F(\lambda) = -\ln \int e^{-Q(\xi(x),\lambda) - V(x)} dx. \tag{5.312}$$

The reaction coordinate $\xi(x)$ is commonly coupled to $\lambda$ with a harmonic potential

$$Q(\xi,\lambda) = \frac{1}{2}\beta k (\xi - \lambda)^2, \tag{5.313}$$

so that for large force constants $k$, $\xi \approx \lambda$. Note the use of dimensionless energies for compatibility with previously published work. Units of energy are obtained by multiplication with $k_B T = 1/\beta$. In the simulation, $\lambda$ samples the user-defined sampling interval $I$.

Being the convolution of the PMF with the Gaussian defined by the harmonic potential, $F(\lambda)$ is a smoothened version of the PMF. (5.311) shows that in order to obtain $P(\lambda) = \rho(\lambda)$, $F(\lambda)$ needs to be determined accurately. Thus, AWH adaptively calculates $F(\lambda)$ and simultaneously converges $P(\lambda)$ toward $\rho(\lambda)$.

For a multidimensional reaction coordinate $\xi$, the sampling interval is the Cartesian product $I = \Pi_\mu I_\mu$ (a rectangular domain).

N.b., it is not yet possible to use AWH for alchemical transformations that involve perturbed masses or constraints.

### The free energy update

AWH is initialized with an estimate of the free energy $F_0(\lambda)$. At regular time intervals this estimate is updated using data collected in between the updates. At update $n$, the applied bias $g_n(\lambda)$ is a function of the current free energy estimate $F_n(\lambda)$ and target distribution $\rho_n(\lambda)$,

$$g_n(\lambda) = \ln \rho_n(\lambda) + F_n(\lambda), \tag{5.314}$$

which is consistent with (5.311). Note that also the target distribution may be updated during the simulation (see examples in section *Choice of target distribution* (page 604)). Substituting this choice of $g = g_n$ back into (5.311) yields the simple free energy update

$$\Delta F_n(\lambda) = F(\lambda) - F_n(\lambda) = -\ln \frac{P_n(\lambda)}{\rho_n(\lambda)}, \tag{5.315}$$

which would yield a better estimate $F_{n+1} = F_n + \Delta F_n$, assuming $P_n(\lambda)$ can be measured accurately. AWH estimates $P_n(\lambda)$ by regularly calculating the conditional distribution

$$\omega_n(\lambda|x) \equiv P_n(\lambda|x) = \frac{e^{g_n(\lambda) - Q(\xi(x),\lambda)}}{\sum_{\lambda'} e^{g_n(\lambda') - Q(\xi(x),\lambda')}}. \tag{5.316}$$

Accumulating these probability weights yields $\sum_t \omega(\lambda|x(t)) \sim P_n(\lambda)$, where $\int P_n(\lambda|x) P_n(x) dx = P_n(\lambda)$ has been used. The $\omega_n(\lambda|x)$ weights are thus the samples of the AWH method. With the limited amount of sampling one has in practice, update scheme (5.315) yields very noisy results. AWH instead applies a free energy update that has the same form but which can be applied repeatedly with limited and localized sampling,

$$\Delta F_n = -\ln \frac{W_n(\lambda) + \sum_t \omega_n(\lambda|x(t))}{W_n(\lambda) + \sum_t \rho_n(\lambda))}. \tag{5.317}$$

Here $W_n(\lambda)$ is the *reference weight histogram* representing prior sampling. The update for $W(\lambda)$, disregarding the initial stage (see section *The initial stage* (page 600)), is

$$W_{n+1}(\lambda) = W_n(\lambda) + \sum_t \rho_n(\lambda). \tag{5.318}$$

Thus, the weight histogram equals the targeted, "ideal" history of samples. There are two important things to note about the free energy update. First, sampling is driven away from oversampled, currently local regions. For such $\lambda$ values, $\omega_n(\lambda) > \rho_n(\lambda)$ and $\Delta F_n(\lambda) < 0$, which by (5.314) implies $\Delta g_n(\lambda) < 0$ (assuming $\Delta \rho_n \equiv 0$). Thus, the probability to sample $\lambda$ decreases after the update (see (5.311)).

Secondly, the normalization of the histogram $N_n = \sum_\lambda W_n(\lambda)$, determines the update size $|\Delta F(\lambda)|$. For instance, for a single sample $\omega(\lambda|x)$, and using a harmonic potential (:see (5.313)), the shape of the update is approximately a Gaussian function of width $\sigma = 1/\sqrt{\beta k}$ and height $\propto 1/N_n$ *137* (page 690),

$$|\Delta F_n(\lambda)| \propto \frac{1}{N_n} e^{-\frac{1}{2}\beta k(\xi(x)-\lambda)^2}. \tag{5.319}$$

When directly controlling the lambda state of the system, the shape of the update is instead

$$|\Delta F_n(\lambda)| \propto \frac{1}{N_n} P_n(\lambda|x). \tag{5.320}$$

Therefore, in both cases, as samples accumulate in $W(\lambda)$ and $N_n$ grows, the updates get smaller, allowing for the free energy to converge.

Note that quantity of interest to the user is not $F(\lambda)$ but the PMF $\Phi(\xi)$. $\Phi(\xi)$ is extracted by reweighting samples $\xi(t)$ on the fly *137* (page 690) (see also section *Reweighting and combining biased data* (page 606)) and will converge at the same rate as $F(\lambda)$, see Fig. 5.41. The PMF will be written to output (see section *Limitations* (page 607)).

### Applying the bias to the system

The bias potential can be applied to the system in two ways. Either by applying a harmonic potential centered at $\lambda(t)$, which is sampled using (rejection-free) Monte-Carlo sampling from the conditional distribution $\omega_n(\lambda|x(t)) = P_n(\lambda|x(t))$, see (5.316). This is also called Gibbs sampling or independence sampling. Alternatively, and by default in the code, the following *convolved bias potential* can be applied,

$$U_n(\xi) = -\ln \int e^{g_n(\lambda) - Q(\xi,\lambda)} d\lambda. \tag{5.321}$$

These two approaches are equivalent in the sense that they give rise to the same biased probabilities $P_n(x)$ (cf. (5.310)) while the dynamics are clearly different in the two cases. This choice does not affect the internals of the AWH algorithm, only what force and potential AWH returns to the MD engine.

Along a bias dimension directly controlling the $\lambda$ state of the system, such as when controlling free energy perturbations, the Monte-Carlo sampling alternative is always used, even if a convolved bias potential is chosen to be used along the other dimensions (if there are more than one).

### The initial stage

Initially, when the bias potential is far from optimal, samples will be highly correlated. In such cases, letting $W(\lambda)$ accumulate samples as prescribed by (5.318), entails a too rapid decay of the free energy update size. This motivates splitting the simulation into an *initial stage* where the weight histogram grows according to a more restrictive and robust protocol, and a *final stage* where the weight histogram grows linearly at the sampling rate ((5.318)). The AWH initial stage takes inspiration from the well-known Wang-Landau algorithm *138* (page 690), although there are differences in the details.

In the initial stage the update size is kept constant (by keeping $N_n$ constant) until a transition across the sampling interval has been detected, a "covering". For the definition of a covering, see (5.322) below. After a covering has occurred, $N_n$ is scaled up by a constant "growth factor" $\gamma$, which by default has the value of 2. Thus, in the initial stage $N_n$ is set dynamically as $N_n = \gamma^m N_0$, where $m$ is the number of coverings. Since the update size scales as $1/N$ ( (5.319)) this leads to a close to exponential decay of the update size in the initial stage, see Fig. 5.41.

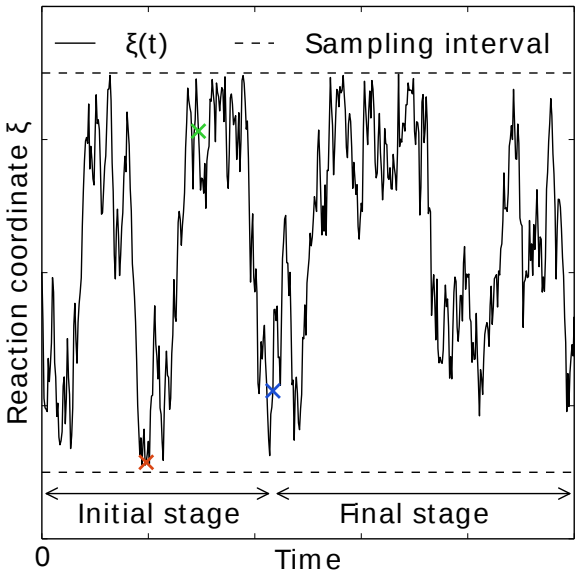



Fig. 5.41: AWH evolution in time for a Brownian particle in a double-well potential. The reaction coordinate $\xi(t)$ traverses the sampling interval multiple times in the initial stage before exiting and entering the final stage. In the final stage, the dynamics of $\xi$ becomes increasingly diffusive.

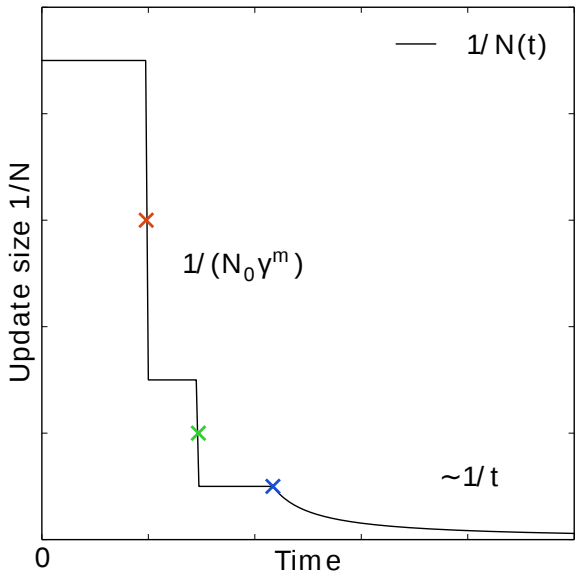



Fig. 5.42: In the final stage, the dynamics of $\xi$ becomes increasingly diffusive. The times of covering are shown as $\times$-markers of different colors. At these times the free energy update size $\sim 1/N$, where $N$ is the size of the weight histogram, is decreased by scaling $N$ by a factor of $\gamma = 3$ (note that the default value of $\gamma$ is 2 since GROMACS 2024).

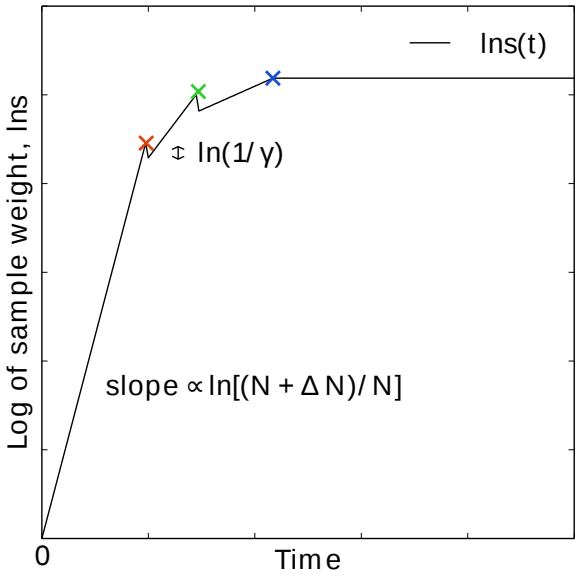



Fig. 5.43: In the final stage, $N$ grows at the sampling rate and thus $1/N \sim 1/t$. The exit from the final stage is determined on the fly by ensuring that the effective sample weight $s$ of data collected in the final stage exceeds that of initial stage data (note that $\ln s(t)$ is plotted).

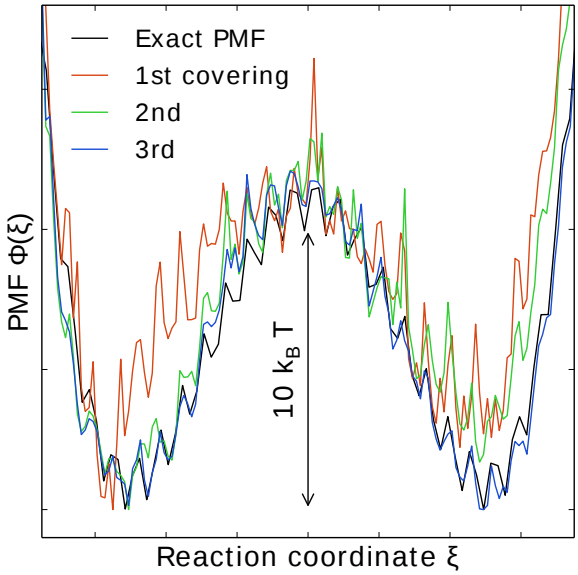



Fig. 5.44: An estimate of the PMF is also extracted from the simulation (bottom right), which after exiting the initial stage should estimate global free energy differences fairly accurately.

The update size directly determines the rate of change of $F_n(\lambda)$ and hence, from (5.314), also the rate of change of the bias funcion $g_n(\lambda)$ Thus initially, when $N_n$ is kept small and updates large, the system will be driven along the reaction coordinate by the constantly fluctuating bias. If $N_0$ is set small enough, the first transition will typically be fast because of the large update size and will quickly give a first rough estimate of the free energy. The second transition, using $N_1 = \gamma N_0$ refines this estimate further. Thus, rather than very carefully filling free energy minima using a small initial update size, the sampling interval is sweeped back-and-forth multiple times, using a wide range of update sizes, see Fig. 5.41. This way, the initial stage also makes AWH robust against the choice of $N_0$.

### The covering criterion

In the general case of a multidimensional reaction coordinate $\lambda = (\lambda_\mu)$, the sampling interval $I$ is considered covered when all dimensions have been covered. A dimension $d$ is covered if all points $\lambda_\mu$ in the one-dimensional sampling interval $I_\mu$ have been "visited". Finally, a point $\lambda_\mu \in I_\mu$ has been visited if there is at least one point $\lambda^* \in I$ with $\lambda^*_\mu = \lambda_\mu$ that since the last covering has accumulated probability weight corresponding to the peak of a multidimensional Gaussian distribution

$$\Delta W(\lambda^*) \geq w_{\text{peak}} \equiv \prod_\mu \frac{\Delta\lambda_\mu}{\sqrt{2\pi}\sigma_k}. \tag{5.322}$$

Here, $\Delta\lambda_\mu$ is the point spacing of the discretized $I_\mu$ and $\sigma_k = 1/\sqrt{\beta k_\mu}$ (where $k_\mu$ is the force constant) is the Gaussian width.

### Exit from the initial stage

For longer times, when major free energy barriers have largely been flattened by the converging bias potential, the histogram $W(\lambda)$ should grow at the actual sampling rate and the initial stage needs to be exited *141* (page 690). There are multiple reasonable (heuristic) ways of determining when this transition should take place. One option is to postulate that the number of samples in the weight histogram $N_n$ should never exceed the actual number of collected samples, and exit the initial stage when this condition breaks *137* (page 690). In the initial stage, $N$ grows close to exponentially while the collected number of samples grows linearly, so an exit will surely occur eventually. Here we instead apply an exit criterion based on the observation that "artificially" keeping $N$ constant while continuing to collect samples corresponds to scaling down the relative weight of old samples relative to new ones. Similarly, the subsequent scaling up of $N$ by a factor $\gamma$ corresponds to scaling up the weight of old data. Briefly, the exit criterion is devised such that the weight of a sample collected *after* the initial stage is always larger or equal to the weight of a sample collected *during* the initial stage, see Fig. 5.41. This is consistent with scaling down early, noisy data.

The initial stage exit criterion will now be described in detail. We start out at the beginning of a covering stage, so that $N$ has just been scaled by $\gamma$ and is now kept constant. Thus, the first sample of this stage has the weight $s = 1/\gamma$ relative to the last sample of the previous covering stage. We assume that $\Delta N$ samples are collected and added to $W$ for each update . To keep $N$ constant, $W$ needs to be scaled down by a factor $N/(N + \Delta N)$ after every update. Equivalently, this means that new data is scaled up relative to old data by the inverse factor. Thus, after $\Delta n$ updates a new sample has the relative weight $s = (1/\gamma)[(N_n + \Delta N)/N_n]^{\Delta n}$. Now assume covering occurs at this time. To continue to the next covering stage, $N$ should be scaled by $\gamma$, which corresponds to again multiplying $s$ by $1/\gamma$. If at this point $s \geq \gamma$, then after rescaling $s \geq 1$; i.e. overall the relative weight of a new sample relative to an old sample is still growing fast. If on the contrary $s < \gamma$, and this defines the exit from the initial stage, then the initial stage is over and from now $N$ simply grows at the sampling rate (see (5.318)). To really ensure that $s \geq 1$ holds before exiting, so that samples after the exit have at least the sample weight of older samples, the last covering stage is extended by a sufficient number of updates.

### Choice of target distribution

The target distribution $\rho(\lambda)$ is traditionally chosen to be uniform

$$\rho_{\text{const}}(\lambda) = \text{const.} \tag{5.323}$$

This choice exactly flattens $F(\lambda)$ in user-defined sampling interval $I$. Generally, $\rho(\lambda) = 0, \lambda \notin I$. In certain cases other choices may be preferable. For instance, in the multidimensional case the rectangular sampling interval is likely to contain regions of very high free energy, e.g. where atoms are clashing. To exclude such regions, $\rho(\lambda)$ can specified by the following function of the free energy

$$\rho_{\text{cut}}(\lambda) \propto \frac{1}{1 + e^{F(\lambda) - F_{\text{cut}}}}, \tag{5.324}$$

where $F_{\text{cut}}$ is a free energy cutoff (relative to $\min_\lambda F(\lambda)$). Thus, regions of the sampling interval where $F(\lambda) > F_{\text{cut}}$ will be exponentially suppressed (in a smooth fashion). Alternatively, very high free energy regions could be avoided while still flattening more moderate free energy barriers by targeting a Boltzmann distribution corresponding to scaling $\beta = 1/k_B T$ by a factor $0 < s_\beta < 1$,

$$\rho_{\text{Boltz}}(\lambda) \propto e^{-s_\beta F(\lambda)}, \tag{5.325}$$

The parameter $s_\beta$ determines to what degree the free energy landscape is flattened; the lower $s_\beta$, the flatter. Note that both $\rho_{\text{cut}}(\lambda)$ and $\rho_{\text{Boltz}}(\lambda)$ depend on $F(\lambda)$, which needs to be substituted by the current best estimate $F_n(\lambda)$. Thus, the target distribution is also updated (consistently with (5.314)).

There is in fact an alternative approach to obtaining $\rho_{\text{Boltz}}(\lambda)$ as the limiting target distribution in AWH, which is particular in the way the weight histogram $W(\lambda)$ and the target distribution $\rho$ are updated and coupled to each other. This yields an evolution of the bias potential which is very similar to that of well-tempered metadynamics *142* (page 690), see *137* (page 690) for details. Because of the popularity and success of well-tempered metadynamics, this is a special case worth considering. In this case $\rho$ is a function of the reference weight histogram

$$\rho_{\text{Boltz,loc}}(\lambda) \propto W(\lambda), \tag{5.326}$$

and the update of the weight histogram is modified (cf. (5.318))

$$W_{n+1}(\lambda) = W_n(\lambda) + s_\beta \sum_t \omega(\lambda | x(t)). \tag{5.327}$$

Thus, here the weight histogram equals the real history of samples, but scaled by $s_\beta$. This target distribution is called *local* Boltzmann since $W$ is only modified locally, where sampling has taken place. We see that when $s_\beta \approx 0$ the histogram essentially does not grow and the size of the free energy update will stay at a constant value (as in the original formulation of metadynamics). Thus, the free energy estimate will not converge, but continue to fluctuate around the correct value. This illustrates the inherent coupling between the convergence and choice of target distribution for this special choice of target. Furthermore note that when using $\rho = \rho_{\text{Boltz,loc}}$ there is no initial stage (section *The initial stage* (page 600)). The rescaling of the weight histogram applied in the initial stage is a global operation, which is incompatible $\rho_{\text{Boltz,loc}}$ only depending locally on the sampling history.

The target distribution can also be modulated by arbitrary probability weights

$$\rho(\lambda) = \rho_0(\lambda) w_{\text{user}}(\lambda). \tag{5.328}$$

where $w_{\text{user}}(\lambda)$ is provided by user data and in principle $\rho_0(\lambda)$ can be any of the target distributions mentioned above.

Lastly, it is possible to automatically scale the target distribution ($\rho_0(\lambda)$) based on the AWH friction metric (see section *The friction metric* (page 607)). This implies scaling the target distribution by the square root of the friction metric (see (5.333)),

$$\rho(\lambda) = \rho_0(\lambda) w_{\text{user}}(\lambda) \sqrt{\det \eta_{\mu\nu}(\lambda)}, \tag{5.329}$$

where $w_{\text{user}}(\lambda)$ can be uniform and sqrt{deteta_{munu}(lambda)} is the square root of the friction metric. This scaling of the target distribution, increasing the relative sampling of regions with slower diffusion, should generally lower the statistical error of the estimated free energy landscape.

This modification is only applied after leaving the initial stage (section *The initial stage* (page 600)), if applicable, and is performed when updating the target distribution, typically when also updating the free energy. The scaling is based on the relative difference of the local friction metric compared to the average friction metric (of points that have a non-zero friction metric).

If any histograms have not been sampled enough to have a friction metric they will be scaled by the average friction metric, i.e., practically unscaled. Insufficient sampling can result in a too low, but still non-zero, friction metric. To address that, the scaling down of the target distribution (relative scaling < 1) is based on the local sampling of each point, so that the target distribution of points that have not been sampled much yet will be almost unaffected. Furthermore, the scaling can be limited by a maximum scaling factor (`awh1-target-metric-scaling-limit` (page 80)). The lower limit of the scaling is the inverse of the maximum scaling factor.

More information about this scaling can be found in *194* (page 693).

Scaling the target distribution based on the friction metric can be combined with Boltzmann or Local-Boltzmann target distributions. However, this is generally not recommended, due to the risk of feedback loops between the two adaptive update mechanisms.

### Multiple independent or sharing biases

Multiple independent bias potentials may be applied within one simulation. This only makes sense if the biased coordinates $\xi^{(1)}$, $\xi^{(2)}$, ... evolve essentially independently from one another. A typical example of this would be when applying an independent bias to each monomer of a protein. Furthermore, multiple AWH simulations can be launched in parallel, each with a (set of) independent biases.

If the defined sampling interval is large relative to the diffusion time of the reaction coordinate, traversing the sampling interval multiple times as is required by the initial stage (section *The initial stage* (page 600)) may take an infeasible mount of simulation time. In these cases it could be advantageous to parallelize the work and have a group of multiple "walkers" $\xi^{(i)}(t)$ share a single bias potential. This can be achieved by collecting samples from all $\xi^{(i)}$ of the same sharing group into a single histogram and update a common free energy estimate. Samples can be shared between walkers within the simulation and/or between multiple simulations. However, currently only sharing between simulations is supported in the code while all biases within a simulation are independent.

Note that when attempting to shorten the simulation time by using bias-sharing walkers, care must be taken to ensure the simulations are still long enough to properly explore and equilibrate all regions of the sampling interval. To begin, the walkers in a group should be decorrelated and distributed approximately according to the target distribution before starting to refine the free energy. This can be achieved e.g. by "equilibrating" the shared weight histogram before letting it grow; for instance, $W(\lambda)/N \approx \rho(\lambda)$ with some tolerance. Such a check is turned on by default (mdp parameter `awh1-equilibrate-histogram` (page 78)). The tolerance is set by `awh1-histogram-tolerance` (page 79), the default value of 0.3 is expected to work well in all cases, both with single and multiple walkers.

Furthermore, the "covering" or transition criterion of the initial stage should to be generalized to detect when the sampling interval has been collectively traversed. One alternative is to just use the same criterion as for a single walker (but now with more samples), see (5.322). However, in contrast to the single walker case this does not ensure that any real transitions across the sampling interval has taken place; in principle all walkers could be sampling only very locally and still cover the whole interval. Just as with a standard umbrella sampling procedure, the free energy may appear to be converged while in reality simulations sampling closeby $\lambda$ values are sampling disconnected regions of phase space. A stricter criterion, which helps avoid such issues, is to require that before a simulation marks a point $\lambda_\mu$ along dimension $\mu$ as visited, and shares this with the other walkers, also all points within a certain diameter $D_{\text{cover}}$ should have been visited (i.e. fulfill (5.322)). Increasing $D_{\text{cover}}$ increases robustness, but may slow down convergence. For the maximum value of $D_{\text{cover}}$, equal to the length of the sampling interval, the sampling interval is considered covered when at least one walker has independently traversed the sampling interval.

In practice biases are shared by setting *awh-share-multisim* (page 77) to true and *awh1-share-group* (page 80) (for bias 1) to a non-zero value. Here, bias 1 will be shared between simulations that have the same share group value. Sharing can be different for bias 1, 2, etc. (although there are few use cases where this is useful). Technically there are no restrictions on sharing, apart from that biases that are shared need to have the same number of grid points and the update intervals should match. Note that biases can not be shared within a simulation. The latter could be useful, especially for multimeric proteins, but this is more difficult to implement.

### Reweighting and combining biased data

Often one may want to, post-simulation, calculate the unbiased PMF $\Phi(u)$ of another variable $u(x)$. $\Phi(u)$ can be estimated using $\xi$-biased data by reweighting ("unbiasing") the trajectory using the bias potential $U_{n(t)}$, see (5.321). Essentially, one bins the biased data along $u$ and removes the effect of $U_{n(t)}$ by dividing the weight of samples $u(t)$ by $e^{-U_{n(t)}(\xi(t))}$,

$$\hat{\Phi}(u) = -\ln \sum_t 1_u(u(t)) e^{U_{n(t)}(\xi(t)} \mathcal{Z}_{n(t)}. \tag{5.330}$$

Here the indicator function $1_u$ denotes the binning procedure: $1_u(u') = 1$ if $u'$ falls into the bin labeled by $u$ and 0 otherwise. The normalization factor $\mathcal{Z}_n = \int e^{-\Phi(\xi)-U_n(\xi)} d\xi$ is the partition function of the extended ensemble. As can be seen $\mathcal{Z}_n$ depends on $\Phi(\xi)$, the PMF of the (biased) reaction coordinate $\xi$ (which is calculated and written to file by the AWH simulation). It is advisable to use only final stage data in the reweighting procedure due to the rapid change of the bias potential during the initial stage. If one would include initial stage data, one should use the sample weights that are inferred by the repeated rescaling of the histogram in the initial stage, for the sake of consistency. Initial stage samples would then in any case be heavily scaled down relative to final stage samples. Note that (5.330) can also be used to combine data from multiple simulations (by adding another sum also over the trajectory set). Furthermore, when multiple independent AWH biases have generated a set of PMF estimates $\{\hat{\Phi}^{(i)}(\xi)\}$, a combined best estimate $\hat{\Phi}(\xi)$ can be obtained by applying self-consistent exponential averaging. More details on this procedure and a derivation of (5.330) (using slightly different notation) can be found in *143* (page 690).

### The friction metric

During the AWH simulation, the following time-integrated force correlation function is calculated,

$$\eta_{\mu\nu}(\lambda) = \beta \int_0^\infty \frac{\langle \delta\mathcal{F}_\mu(x(t),\lambda)\delta\mathcal{F}_\nu(x(0),\lambda)\omega(\lambda|x(t))\omega(\lambda|x(0))\rangle}{\langle\omega^2(\lambda|x)\rangle} dt. \tag{5.331}$$

Here $\mathcal{F}_\mu(x,\lambda) = k_\mu(\xi_\mu(x) - \lambda_\mu)$ is the force along dimension $\mu$ from an harmonic potential centered at $\lambda$ and $\delta\mathcal{F}_\mu(x,\lambda) = \mathcal{F}_\mu(x,\lambda) - \langle\mathcal{F}_\mu(x,\lambda)\rangle$ is the deviation of the force. The factors $\omega(\lambda|x(t))$, see (5.316), reweight the samples. $\eta_{\mu\nu}(\lambda)$ is a friction tensor *186* (page 692) and *144* (page 690). The diffusion matrix, on the flattened landscape, is equal to $k_BT$ times the inverse of the friction metrix tensor:

$$\mathbf{D}(\lambda) = k_BT\eta^{-1}(\lambda). \tag{5.332}$$

A measure of sampling (in)efficiency at each $\lambda$ is given by

$$\eta^{\frac{1}{2}}(\lambda) = \sqrt{\det \eta_{\mu\nu}(\lambda)}. \tag{5.333}$$

A large value of $\eta^{\frac{1}{2}}(\lambda)$ indicates slow dynamics and long correlation times, which may require more sampling.

### Limitations

The only real limitation of the AWH implementation, apart from the not uncommon practical issue that the method might not converge sufficiently fast, is a limit on the maximum free energy difference. This limit is set to $700k_BT$, because $e^{700}$ is close to the maximum value that can be accurately represented by a double-precision floating-point value. For physical reaction coordinates, this is not a limit in practice. This does limit the range of applications for alchemical coordinates. For instance, hydration free-energies of divalent cations with a pair of monovalent anions can exceed this limit. The limit can also be exceeded when decoupling large molecules from solvent, but this often coincides with the limit where the sampling becomes problematic.

### Usage

AWH stores data in the energy file (*edr* (page 576)) with a frequency set by the user. The data – the PMF, the convolved bias, distributions of the $\lambda$ and $\xi$ coordinates, etc. – can be extracted after the simulation using the *gmx awh* (page 160) tool. Furthermore, the trajectory of the reaction coordinate $\xi(t)$ is printed to the pull output file `pullx.xvg`. The log file (*log* (page 578)) also contains information; check for messages starting with "awh", they will tell you about covering and potential sampling issues.

### Setting the initial update size

The initial value of the weight histogram size $N$ sets the initial update size (and the rate of change of the bias). When $N$ is kept constant, like in the initial stage, the average variance of the free energy scales as $\varepsilon^2 \sim 1/(ND)$ *137* (page 690), for a simple model system with constant diffusion $D$ along the reaction coordinate. This provides a ballpark estimate used by AWH to initialize $N$ in terms of more meaningful quantities

$$\frac{1}{N_0} = \frac{1}{N_0(\varepsilon_0, D)} = \Delta t_{\text{sample}} \max_d \frac{2D_d}{L_d^2}\varepsilon_0^2 \tag{5.334}$$

where $L_d$ is the length of the interval and $D_d$ is the diffusion constant along dimension $d$ of the AWH bias. For one dimension, $L^2/2D$ is the average time to diffuse over a distance of $L$. We then takes the maximum crossing time over all dimensions involved in the bias. Essentially, this formula tells us that a slower system (small $D$) requires more samples (larger $N^0$) to attain the same level of accuracy ($\varepsilon_0$) at a given sampling rate. Conversely, for a system of given diffusion, how to choose the initial biasing rate depends on how good the initial accuracy is. Both the initial error $\varepsilon_0$ and the diffusion $D$ only need to be roughly estimated or guessed. In the typical case, one would only tweak the $D$ parameter, and use a default value for $\varepsilon_0$. For good convergence, $D$ should be chosen as large as possible (while maintaining a stable system) giving large initial bias updates and fast initial transitions. Choosing $D$ too small can lead to slow initial convergence. It may be a good idea to run a short trial simulation and after the first covering check the maximum free energy difference of the PMF estimate. If this is much larger than the expected magnitude of the free energy barriers that should be crossed, then the system is probably being pulled too hard and $D$ should be decreased. An accurate estimate of the diffusion can be obtained from an AWH simulation with the *gmx awh* (page 160) tool. $\varepsilon_0$ on the other hand, should be a rough estimate of the initial error.

### Estimating errors

As with any adaptive method, estimating errors for AWH is difficult from data of a single simulation only. We are looking into methods to do this. For now, the only safe way to estimate errors is to run multiple completely independent simulations and compute a standard error estimate. Note that for the simulations to be really independent, they should start from different, equilibrated states along the reaction coordinate(s). In practice, this is often difficult to achieve, in particular in the common case that you only know the starting state along the reaction coordinate. The exit from the initial phase of AWH is designed such that, in most cases, such systematic errors are as small as the noise when exiting the initial phase, but it cannot be excluded that some effects are still present.

### Tips for efficient sampling

The force constant $k$ should be larger than the curvature of the PMF landscape. If this is not the case, the distributions of the reaction coordinate $\xi$ and the reference coordinate $\lambda$, will differ significantly and warnings will be printed in the log file. One can choose $k$ as large as the time step supports. This will necessarily increase the number of points of the discretized sampling interval $I$. In general however, it should not affect the performance of the simulation noticeably because the AWH update is implemented such that only sampled points are accessed at free energy update time.

For an alchemical free-energy dimension, AWH accesses all $\lambda$ points at every sampling step. Because the number of $\lambda$ points is usually far below 100, there is no significant cost to this in the AWH method itself. However, foreign energy differences need to be computed for every $\lambda$ value used, which can become somewhat costly.

As with any method, the choice of reaction coordinate(s) is critical. If a single reaction coordinate does not suffice, identifying a second reaction coordinate and sampling the two-dimensional landscape may help. In this case, using a target distribution with a free energy cutoff (see (5.324)) might be required to avoid sampling uninteresting regions of very high free energy. Obtaining accurate free energies for reaction coordinates of much higher dimensionality than 3 or possibly 4 is generally not feasible.

Monitoring the transition rate of $\xi(t)$, across the sampling interval is also advisable. For reliable statistics (e.g. when reweighting the trajectory as described in section *Reweighting and combining biased data* (page 606)), one would generally want to observe at least a few transitions after having exited the initial stage. Furthermore, if the dynamics of the reaction coordinate suddenly changes, this may be a sign of e.g. a reaction coordinate problem.

Difficult regions of sampling may also be detected by calculating the friction tensor $\eta_{\mu\nu}(\lambda)$ in the sampling interval, see section *The friction metric* (page 607). $\eta_{\mu\nu}(\lambda)$ as well as the sampling efficiency measure $\eta^{\frac{1}{2}}(\lambda)$ ((5.333)) are written to the energy file and can be extracted with *gmx awh* (page 160). A high peak in $\eta^{\frac{1}{2}}(\lambda)$ indicates that this region requires longer time to sample properly.

### 5.8.6 Enforced Rotation

This module can be used to enforce the rotation of a group of atoms, as *e.g.* a protein subunit. There are a variety of rotation potentials, among them complex ones that allow flexible adaptations of both the rotated subunit as well as the local rotation axis during the simulation. An example application can be found in ref. *145* (page 690).

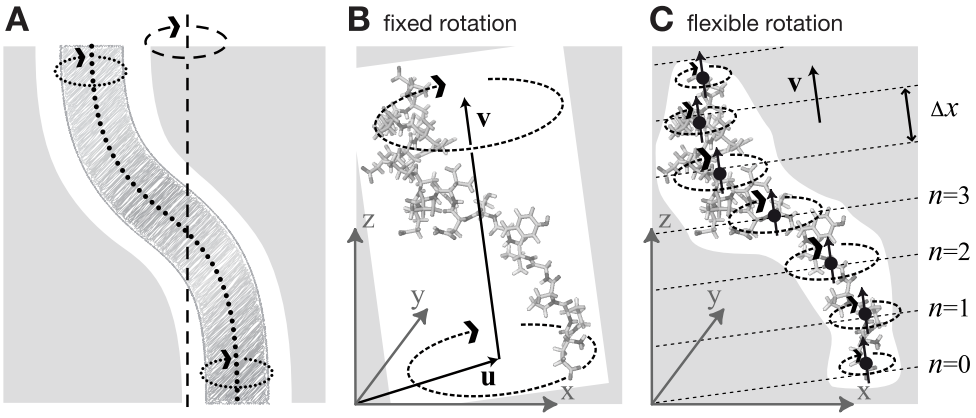



Fig. 5.45: Comparison of fixed and flexible axis rotation. A: Rotating the sketched shape inside the white tubular cavity can create artifacts when a fixed rotation axis (dashed) is used. More realistically, the shape would revolve like a flexible pipe-cleaner (dotted) inside the bearing (gray). B: Fixed rotation around an axis $\mathbf{v}$ with a pivot point specified by the vector $\mathbf{u}$. C: Subdividing the rotating fragment into slabs with separate rotation axes (↑) and pivot points (•) for each slab allows for flexibility. The distance between two slabs with indices $n$ and $n+1$ is $\Delta x$.

#### Fixed Axis Rotation

##### Stationary Axis with an Isotropic Potential

In the fixed axis approach (see Fig. 5.45 B), torque on a group of $N$ atoms with positions $\mathbf{x}_i$ (denoted "rotation group") is applied by rotating a reference set of atomic positions – usually their initial positions $\mathbf{y}_i^0$ – at a constant angular velocity $\omega$ around an axis defined by a direction vector $\hat{\mathbf{v}}$ and a pivot point $\mathbf{u}$. To that aim, each atom with position $\mathbf{x}_i$ is attracted by a "virtual spring" potential to its moving reference position $\mathbf{y}_i = \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u})$, where $\mathbf{\Omega}(t)$ is a matrix that describes the rotation around the axis. In the simplest case, the "springs" are described by a harmonic potential,

$$V^{\text{iso}} = \frac{k}{2}\sum_{i=1}^{N} w_i \left[\mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u}) - (\mathbf{x}_i - \mathbf{u})\right]^2 \tag{5.335}$$

with optional mass-weighted prefactors $w_i = N\,m_i/M$ with total mass $M = \sum_{i=1}^{N} m_i$. The rotation matrix $\mathbf{\Omega}(t)$ is

$$\mathbf{\Omega}(t) = \begin{pmatrix} \cos\omega t + v_x^2\,\xi & v_x v_y\,\xi - v_z \sin\omega t & v_x v_z\,\xi + v_y \sin\omega t \\ v_x v_y\,\xi + v_z \sin\omega t & \cos\omega t + v_y^2\,\xi & v_y v_z\,\xi - v_x \sin\omega t \\ v_x v_z\,\xi - v_y \sin\omega t & v_y v_z\,\xi + v_x \sin\omega t & \cos\omega t + v_z^2\,\xi \end{pmatrix} \tag{5.336}$$

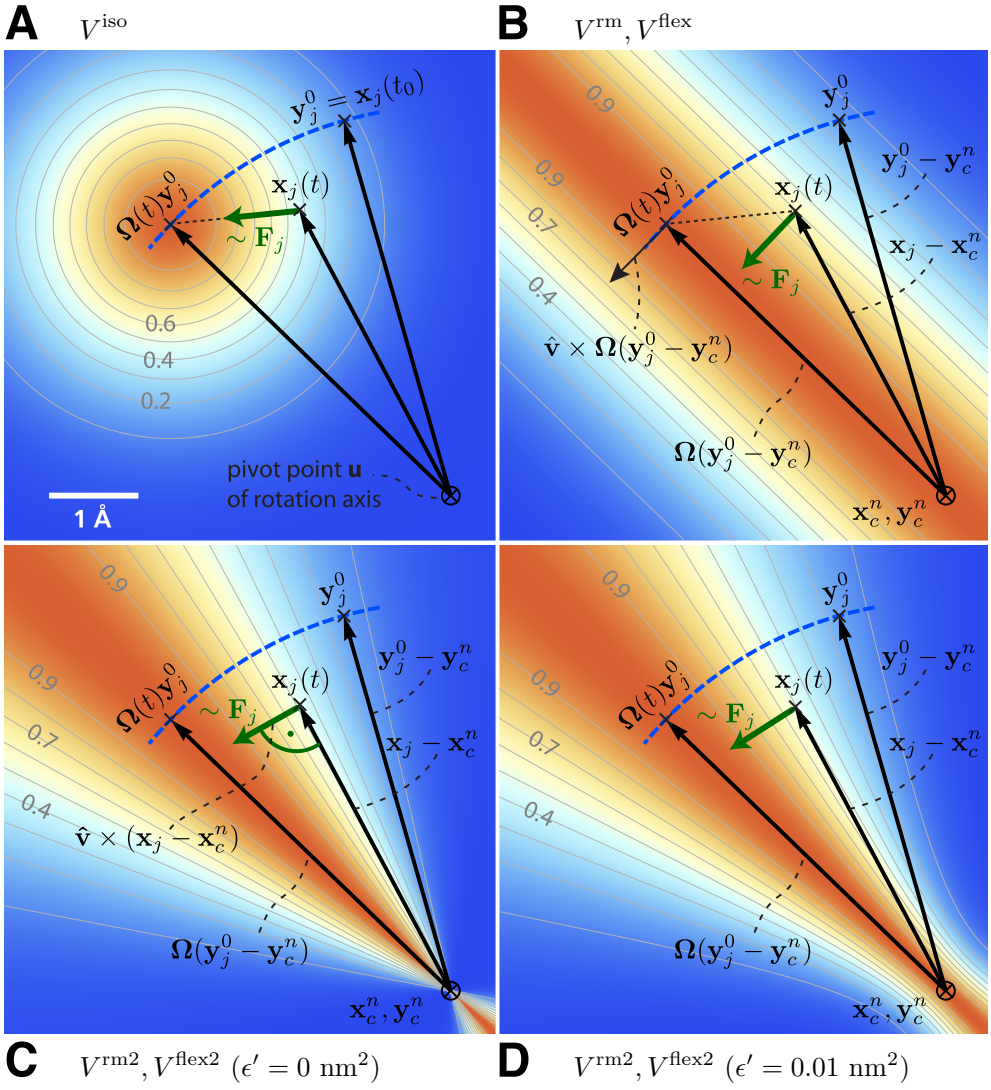



Fig. 5.46: Selection of different rotation potentials and definition of notation. All four potentials $V$ (color coded) are shown for a single atom at position $\mathbf{x}_j(t)$. A: Isotropic potential $V^{\mathrm{iso}}$, B: radial motion potential $V^{\mathrm{rm}}$ and flexible potential $V^{\mathrm{flex}}$, C–D: radial motion2 potential $V^{\mathrm{rm2}}$ and flexible2 potential $V^{\mathrm{flex2}}$ for $\epsilon'$=0nm$^2$ (C) and $\epsilon'$=0.01nm$^2$ (D). The rotation axis is perpendicular to the plane and marked by $\otimes$. The light gray contours indicate Boltzmann factors $e^{-V/(k_B T)}$ in the $\mathbf{x}_j$-plane for $T = 300$K and $k$=200kJ/(mol $\cdot$ nm$^2$). The green arrow shows the direction of the force $\mathbf{F}_j$ acting on atom $j$; the blue dashed line indicates the motion of the reference position.

where $v_x$, $v_y$, and $v_z$ are the components of the normalized rotation vector $\hat{\mathbf{v}}$, and $\xi := 1 - \cos(\omega t)$. As illustrated in Fig. 5.46 A for a single atom $j$, the rotation matrix $\mathbf{\Omega}(t)$ operates on the initial reference positions $\mathbf{y}_j^0 = \mathbf{x}_j(t_0)$ of atom $j$ at $t = t_0$. At a later time $t$, the reference position has rotated away from its initial place (along the blue dashed line), resulting in the force

$$\mathbf{F}_j^{\text{iso}} = -\nabla_j V^{\text{iso}} = k\, w_j \left[\mathbf{\Omega}(t)(\mathbf{y}_j^0 - \mathbf{u}) - (\mathbf{x}_j - \mathbf{u})\right] \tag{5.337}$$

which is directed towards the reference position.

### Pivot-Free Isotropic Potential

Instead of a fixed pivot vector $\mathbf{u}$ this potential uses the center of mass $\mathbf{x}_c$ of the rotation group as pivot for the rotation axis,

$$\mathbf{x}_c = \frac{1}{M}\sum_{i=1}^{N} m_i \mathbf{x}_i \text{and} \mathbf{y}_c^0 = \frac{1}{M}\sum_{i=1}^{N} m_i \mathbf{y}_i^0\,, \tag{5.338}$$

which yields the "pivot-free" isotropic potential

$$V^{\text{iso}-\text{pf}} = \frac{k}{2}\sum_{i=1}^{N} w_i \left[\mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^0) - (\mathbf{x}_i - \mathbf{x}_c)\right]^2, \tag{5.339}$$

with forces

$$\mathbf{F}_j^{\text{iso}-\text{pf}} = k\, w_j \left[\mathbf{\Omega}(t)(\mathbf{y}_j^0 - \mathbf{y}_c^0) - (\mathbf{x}_j - \mathbf{x}_c)\right]. \tag{5.340}$$

Without mass-weighting, the pivot $\mathbf{x}_c$ is the geometrical center of the group.

### Parallel Motion Potential Variant

The forces generated by the isotropic potentials (eqns. (5.335) and (5.339)) also contain components parallel to the rotation axis and thereby restrain motions along the axis of either the whole rotation group (in case of $V^{\text{iso}}$) or within the rotation group, in case of $V^{\text{iso}-\text{pf}}$.

For cases where unrestrained motion along the axis is preferred, we have implemented a "parallel motion" variant by eliminating all components parallel to the rotation axis for the potential. This is achieved by projecting the distance vectors between reference and actual positions

$$\mathbf{r}_i = \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u}) - (\mathbf{x}_i - \mathbf{u}) \tag{5.341}$$

onto the plane perpendicular to the rotation vector,

$$\mathbf{r}_i^{\perp} := \mathbf{r}_i - (\mathbf{r}_i \cdot \hat{\mathbf{v}})\hat{\mathbf{v}} \tag{5.342}$$

yielding

$$\begin{aligned} V^{\text{pm}} &= \frac{k}{2}\sum_{i=1}^{N} w_i (\mathbf{r}_i^{\perp})^2 \\ &= \frac{k}{2}\sum_{i=1}^{N} w_i \left\{\mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u}) - (\mathbf{x}_i - \mathbf{u})\right. \\ &\quad \left. - \left\{\left[\mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u}) - (\mathbf{x}_i - \mathbf{u})\right] \cdot \hat{\mathbf{v}}\right\}\hat{\mathbf{v}}\right\}^2 \end{aligned}$$

and similarly

$$\mathbf{F}_j^{\text{pm}} = k\, w_j\, \mathbf{r}_j^{\perp} \tag{5.343}$$

### Pivot-Free Parallel Motion Potential

Replacing in eqn. (5.343) the fixed pivot $\mathbf{u}$ by the center of mass $\mathbf{x_c}$ yields the pivot-free variant of the parallel motion potential. With

$$\mathbf{s}_i = \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^0) - (\mathbf{x}_i - \mathbf{x}_c) \tag{5.344}$$

the respective potential and forces are

$$V^{\mathrm{pm-pf}} = \frac{k}{2}\sum_{i=1}^{N} w_i (\mathbf{s}_i^{\perp})^2 \tag{5.345}$$

$$\mathbf{F}_j^{\mathrm{pm-pf}} = k\, w_j\, \mathbf{s}_j^{\perp} \tag{5.346}$$

### Radial Motion Potential

In the above variants, the minimum of the rotation potential is either a single point at the reference position $\mathbf{y}_i$ (for the isotropic potentials) or a single line through $\mathbf{y}_i$ parallel to the rotation axis (for the parallel motion potentials). As a result, radial forces restrict radial motions of the atoms. The two subsequent types of rotation potentials, $V^{\mathrm{rm}}$ and $V^{\mathrm{rm2}}$, drastically reduce or even eliminate this effect. The first variant, $V^{\mathrm{rm}}$ (Fig. 5.46 B), eliminates all force components parallel to the vector connecting the reference atom and the rotation axis,

$$V^{\mathrm{rm}} = \frac{k}{2}\sum_{i=1}^{N} w_i \left[\mathbf{p}_i \cdot (\mathbf{x}_i - \mathbf{u})\right]^2 , \tag{5.347}$$

with

$$\mathbf{p}_i := \frac{\hat{\mathbf{v}} \times \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u})}{\|\hat{\mathbf{v}} \times \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u})\|} . \tag{5.348}$$

This variant depends only on the distance $\mathbf{p}_i \cdot (\mathbf{x}_i - \mathbf{u})$ of atom $i$ from the plane spanned by $\hat{\mathbf{v}}$ and $\mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u})$. The resulting force is

$$\mathbf{F}_j^{\mathrm{rm}} = -k\, w_j \left[\mathbf{p}_j \cdot (\mathbf{x}_j - \mathbf{u})\right] \mathbf{p}_j . \tag{5.349}$$

### Pivot-Free Radial Motion Potential

Proceeding similar to the pivot-free isotropic potential yields a pivot-free version of the above potential. With

$$\mathbf{q}_i := \frac{\hat{\mathbf{v}} \times \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^0)}{\|\hat{\mathbf{v}} \times \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^0)\|} , \tag{5.350}$$

the potential and force for the pivot-free variant of the radial motion potential read

$$V^{\mathrm{rm-pf}} = \frac{k}{2}\sum_{i=1}^{N} w_i \left[\mathbf{q}_i \cdot (\mathbf{x}_i - \mathbf{x}_c)\right]^2 , \tag{5.351}$$

$$\mathbf{F}_j^{\mathrm{rm-pf}} = -k\, w_j \left[\mathbf{q}_j \cdot (\mathbf{x}_j - \mathbf{x}_c)\right] \mathbf{q}_j + k\frac{m_j}{M}\sum_{i=1}^{N} w_i \left[\mathbf{q}_i \cdot (\mathbf{x}_i - \mathbf{x}_c)\right] \mathbf{q}_i . \tag{5.352}$$

### Radial Motion 2 Alternative Potential

As seen in Fig. 5.46 B, the force resulting from $V^{\mathrm{rm}}$ still contains a small, second-order radial component. In most cases, this perturbation is tolerable; if not, the following alternative, $V^{\mathrm{rm2}}$, fully eliminates the radial contribution to the force, as depicted in Fig. 5.46 C,

$$V^{\mathrm{rm2}} = \frac{k}{2}\sum_{i=1}^{N} w_i \frac{\left[(\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{u}))\cdot\boldsymbol{\Omega}(t)(\mathbf{y}_i^0-\mathbf{u})\right]^2}{\|\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{u})\|^2+\epsilon'}\,, \tag{5.353}$$

where a small parameter $\epsilon'$ has been introduced to avoid singularities. For $\epsilon'$=0nm$^2$, the equipotential planes are spanned by $\mathbf{x}_i-\mathbf{u}$ and $\hat{\mathbf{v}}$, yielding a force perpendicular to $\mathbf{x}_i-\mathbf{u}$, thus not contracting or expanding structural parts that moved away from or toward the rotation axis.

Choosing a small positive $\epsilon'$ (*e.g.*, $\epsilon'$=0.01nm$^2$, Fig. 5.46 D) in the denominator of eqn. (5.353) yields a well-defined potential and continuous forces also close to the rotation axis, which is not the case for $\epsilon'$=0nm$^2$ (Fig. 5.46 C). With

$$\begin{aligned}
\mathbf{r}_i &:= && \boldsymbol{\Omega}(t)(\mathbf{y}_i^0-\mathbf{u}) \\
\mathbf{s}_i &:= && \frac{\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{u})}{\|\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{u})\|} \equiv \Psi_i\ \hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{u}) \\
\Psi_i^* &:= && \frac{1}{\|\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{u})\|^2+\epsilon'}
\end{aligned} \tag{5.354}$$

the force on atom $j$ reads

$$\mathbf{F}_j^{\mathrm{rm2}} = -k\left\{w_j\ (\mathbf{s}_j\cdot\mathbf{r}_j)\ \left[\frac{\Psi_j^*}{\Psi_j}\mathbf{r}_j - \frac{\Psi_j^{*2}}{\Psi_j^3}(\mathbf{s}_j\cdot\mathbf{r}_j)\mathbf{s}_j\right]\right\}\times\hat{\mathbf{v}}. \tag{5.355}$$

### Pivot-Free Radial Motion 2 Potential

The pivot-free variant of the above potential is

$$V^{\mathrm{rm2-pf}} = \frac{k}{2}\sum_{i=1}^{N} w_i \frac{\left[(\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c))\cdot\boldsymbol{\Omega}(t)(\mathbf{y}_i^0-\mathbf{y}_c)\right]^2}{\|\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c)\|^2+\epsilon'}\,. \tag{5.356}$$

With

$$\begin{aligned}
\mathbf{r}_i &:= && \boldsymbol{\Omega}(t)(\mathbf{y}_i^0-\mathbf{y}_c) \\
\mathbf{s}_i &:= && \frac{\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c)}{\|\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c)\|} \equiv \Psi_i\ \hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c) \\
\Psi_i^* &:= && \frac{1}{\|\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c)\|^2+\epsilon'}
\end{aligned} \tag{5.357}$$

the force on atom $j$ reads

$$\begin{aligned}
\mathbf{F}_j^{\mathrm{rm2-pf}} = & -k\left\{w_j\ (\mathbf{s}_j\cdot\mathbf{r}_j)\ \left[\frac{\Psi_j^*}{\Psi_j}\mathbf{r}_j - \frac{\Psi_j^{*2}}{\Psi_j^3}(\mathbf{s}_j\cdot\mathbf{r}_j)\mathbf{s}_j\right]\right\}\times\hat{\mathbf{v}} \\
& +k\,\frac{m_j}{M}\left\{\sum_{i=1}^{N} w_i\ (\mathbf{s}_i\cdot\mathbf{r}_i)\ \left[\frac{\Psi_i^*}{\Psi_i}\mathbf{r}_i - \frac{\Psi_i^{*2}}{\Psi_i^3}(\mathbf{s}_i\cdot\mathbf{r}_i)\ \mathbf{s}_i\right]\right\}\times\hat{\mathbf{v}}\,.
\end{aligned}$$

### Flexible Axis Rotation

As sketched in Fig. 5.45 A–B, the rigid body behavior of the fixed axis rotation scheme is a drawback for many applications. In particular, deformations of the rotation group are suppressed when the equilibrium atom positions directly depend on the reference positions. To avoid this limitation, eqns. (5.351) and (5.356) will now be generalized towards a "flexible axis" as sketched in Fig. 5.45 C. This will be achieved by subdividing the rotation group into a set of equidistant slabs perpendicular to the rotation vector, and by applying a separate rotation potential to each of these slabs. Fig. 5.45 C shows the midplanes of the slabs as dotted straight lines and the centers as thick black dots.

To avoid discontinuities in the potential and in the forces, we define "soft slabs" by weighing the contributions of each slab $n$ to the total potential function $V^{\text{flex}}$ by a Gaussian function

$$g_n(\mathbf{x}_i) = \Gamma \exp\left(-\frac{\beta_n^2(\mathbf{x}_i)}{2\sigma^2}\right), \tag{5.358}$$

centered at the midplane of the $n$th slab. Here $\sigma$ is the width of the Gaussian function, $\Delta x$ the distance between adjacent slabs, and

$$\beta_n(\mathbf{x}_i) := \mathbf{x}_i \cdot \hat{\mathbf{v}} - n\,\Delta x\,. \tag{5.359}$$

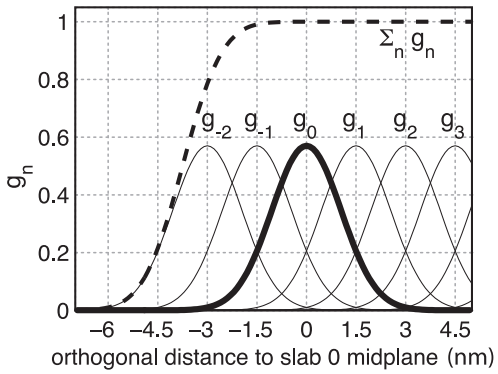



Fig. 5.47: Gaussian functions $g_n$ centered at $n\,\Delta x$ for a slab distance $\Delta x = 1.5$ nm and $n \geq -2$. Gaussian function $g_0$ is highlighted in bold; the dashed line depicts the sum of the shown Gaussian functions.

A most convenient choice is $\sigma = 0.7\Delta x$ and

$$1/\Gamma = \sum_{n\in Z} \exp\left(-\frac{(n-\frac{1}{4})^2}{2\cdot 0.7^2}\right) \approx 1.75464\,, \tag{5.360}$$

which yields a nearly constant sum, essentially independent of $\mathbf{x}_i$ (dashed line in Fig. 5.47), *i.e.*,

$$\sum_{n\in Z} g_n(\mathbf{x}_i) = 1 + \epsilon(\mathbf{x}_i)\,, \tag{5.361}$$

with $|\epsilon(\mathbf{x}_i)| < 1.3\cdot 10^{-4}$. This choice also implies that the individual contributions to the force from the slabs add up to unity such that no further normalization is required.

To each slab center $\mathbf{x}_c^n$, all atoms contribute by their Gaussian-weighted (optionally also mass-weighted) position vectors $g_n(\mathbf{x}_i)\,\mathbf{x}_i$. The instantaneous slab centers $\mathbf{x}_c^n$ are calculated from the current positions $\mathbf{x}_i$,

$$\mathbf{x}_c^n = \frac{\sum_{i=1}^N g_n(\mathbf{x}_i)\,m_i\,\mathbf{x}_i}{\sum_{i=1}^N g_n(\mathbf{x}_i)\,m_i}\,, \tag{5.362}$$

while the reference centers $\mathbf{y}_c^n$ are calculated from the reference positions $\mathbf{y}_i^0$,

$$\mathbf{y}_c^n = \frac{\sum_{i=1}^N g_n(\mathbf{y}_i^0)\, m_i\, \mathbf{y}_i^0}{\sum_{i=1}^N g_n(\mathbf{y}_i^0)\, m_i}\,. \tag{5.363}$$

Due to the rapid decay of $g_n$, each slab will essentially involve contributions from atoms located within $\approx 3\Delta x$ from the slab center only.

### Flexible Axis Potential

We consider two flexible axis variants. For the first variant, the slab segmentation procedure with Gaussian weighting is applied to the radial motion potential (eqn. (5.351) / Fig. 5.46 B), yielding as the contribution of slab $n$

$$V^n = \frac{k}{2}\sum_{i=1}^N w_i\, g_n(\mathbf{x}_i)\,[\mathbf{q}_i^n \cdot (\mathbf{x}_i - \mathbf{x}_c^n)]^2\,, \tag{5.364}$$

and a total potential function

$$V^{\mathrm{flex}} = \sum_n V^n\,. \tag{5.365}$$

Note that the global center of mass $\mathbf{x}_c$ used in eqn. (5.351) is now replaced by $\mathbf{x}_c^n$, the center of mass of the slab. With

$$\begin{aligned} \mathbf{q}_i^n &:= \frac{\hat{\mathbf{v}} \times \boldsymbol{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^n)}{\|\hat{\mathbf{v}} \times \boldsymbol{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^n)\|} \\ b_i^n &:= \mathbf{q}_i^n \cdot (\mathbf{x}_i - \mathbf{x}_c^n)\,, \end{aligned} \tag{5.366}$$

the resulting force on atom $j$ reads

$$\begin{aligned} \mathbf{F}_j^{\mathrm{flex}} = & - k\, w_j \sum_n g_n(\mathbf{x}_j)\, b_j^n \left\{ \mathbf{q}_j^n - b_j^n \frac{\beta_n(\mathbf{x}_j)}{2\sigma^2}\hat{\mathbf{v}} \right\} \\ & + k\, m_j \sum_n \frac{g_n(\mathbf{x}_j)}{\sum_h g_n(\mathbf{x}_h)} \sum_{i=1}^N w_i\, g_n(\mathbf{x}_i)\, b_i^n \left\{ \mathbf{q}_i^n - \frac{\beta_n(\mathbf{x}_j)}{\sigma^2}\,[\mathbf{q}_i^n \cdot (\mathbf{x}_j - \mathbf{x}_c^n)]\,\hat{\mathbf{v}} \right\}. \end{aligned}$$

Note that for $V^{\mathrm{flex}}$, as defined, the slabs are fixed in space and so are the reference centers $\mathbf{y}_c^n$. If during the simulation the rotation group moves too far in $\mathbf{v}$ direction, it may enter a region where – due to the lack of nearby reference positions – no reference slab centers are defined, rendering the potential evaluation impossible. We therefore have included a slightly modified version of this potential that avoids this problem by attaching the midplane of slab $n = 0$ to the center of mass of the rotation group, yielding slabs that move with the rotation group. This is achieved by subtracting the center of mass $\mathbf{x}_c$ of the group from the positions,

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{x}_c\,, \quad \text{and} \quad \tilde{\mathbf{y}}_i^0 = \mathbf{y}_i^0 - \mathbf{y}_c^0\,, \tag{5.367}$$

such that

$$V^{\mathrm{flex-t}} = \frac{k}{2}\sum_n \sum_{i=1}^N w_i\, g_n(\tilde{\mathbf{x}}_i) \left[ \frac{\hat{\mathbf{v}} \times \boldsymbol{\Omega}(t)(\tilde{\mathbf{y}}_i^0 - \tilde{\mathbf{y}}_c^n)}{\|\hat{\mathbf{v}} \times \boldsymbol{\Omega}(t)(\tilde{\mathbf{y}}_i^0 - \tilde{\mathbf{y}}_c^n)\|} \cdot (\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_c^n) \right]^2. \tag{5.368}$$

To simplify the force derivation, and for efficiency reasons, we here assume $\mathbf{x}_c$ to be constant, and thus $\partial\mathbf{x}_c/\partial x = \partial\mathbf{x}_c/\partial y = \partial\mathbf{x}_c/\partial z = 0$. The resulting force error is small (of order $O(1/N)$ or $O(m_j/M)$ if mass-weighting is applied) and can therefore be tolerated. With this assumption, the forces $\mathbf{F}^{\mathrm{flex-t}}$ have the same form as eqn. (5.367).

### Flexible Axis 2 Alternative Potential

In this second variant, slab segmentation is applied to $V^{\mathrm{rm2}}$ (eqn. (5.356)), resulting in a flexible axis potential without radial force contributions (Fig. 5.46 C),

$$V^{\mathrm{flex2}} = \frac{k}{2}\sum_{i=1}^{N}\sum_{n} w_i\, g_n(\mathbf{x}_i)\frac{\left[(\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c^n))\cdot\boldsymbol{\Omega}(t)(\mathbf{y}_i^0-\mathbf{y}_c^n)\right]^2}{\|\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c^n)\|^2+\epsilon'} \tag{5.369}$$

With

$$\begin{aligned}
\mathbf{r}_i^n &:= \boldsymbol{\Omega}(t)(\mathbf{y}_i^0-\mathbf{y}_c^n)\\
\mathbf{s}_i^n &:= \frac{\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c^n)}{\|\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c^n)\|} \equiv \psi_i\ \hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c^n)\\
\psi_i^* &:= \frac{1}{\|\hat{\mathbf{v}}\times(\mathbf{x}_i-\mathbf{x}_c^n)\|^2+\epsilon'}\\
W_j^n &:= \frac{g_n(\mathbf{x}_j)\,m_j}{\sum_h g_n(\mathbf{x}_h)\,m_h}\\
\mathbf{S}^n &:= \sum_{i=1}^{N} w_i\ g_n(\mathbf{x}_i)\ (\mathbf{s}_i^n\cdot\mathbf{r}_i^n)\left[\frac{\psi_i^*}{\psi_i}\mathbf{r}_i^n-\frac{\psi_i^{*2}}{\psi_i^3}(\mathbf{s}_i^n\cdot\mathbf{r}_i^n)\,\mathbf{s}_i^n\right]
\end{aligned} \tag{5.370}$$

the force on atom $j$ reads

$$\begin{aligned}
\mathbf{F}_j{}^{\mathrm{flex2}} = \quad & -k\left\{\sum_n w_j\ g_n(\mathbf{x}_j)\ (\mathbf{s}_j^n\cdot\mathbf{r}_j^n)\ \left[\frac{\psi_j^*}{\psi_j}\mathbf{r}_j^n-\frac{\psi_j^{*2}}{\psi_j^3}(\mathbf{s}_j^n\cdot\mathbf{r}_j^n)\,\mathbf{s}_j^n\right]\right\}\times\hat{\mathbf{v}}\\
& +k\left\{\sum_n W_j^n\,\mathbf{S}^n\right\}\times\hat{\mathbf{v}}-k\left\{\sum_n W_j^n\ \frac{\beta_n(\mathbf{x}_j)}{\sigma^2}\frac{1}{\psi_j}\ \mathbf{s}_j^n\cdot\mathbf{S}^n\right\}\hat{\mathbf{v}}\\
& +\frac{k}{2}\left\{\sum_n w_j\ g_n(\mathbf{x}_j)\frac{\beta_n(\mathbf{x}_j)}{\sigma^2}\frac{\psi_j^*}{\psi_j^2}(\mathbf{s}_j^n\cdot\mathbf{r}_j^n)^2\right\}\hat{\mathbf{v}}.
\end{aligned}$$

Applying transformation (5.367) yields a "translation-tolerant" version of the flexible2 potential, $V^{\mathrm{flex2-t}}$. Again, assuming that $\partial\mathbf{x}_c/\partial x$, $\partial\mathbf{x}_c/\partial y$, $\partial\mathbf{x}_c/\partial z$ are small, the resulting equations for $V^{\mathrm{flex2-t}}$ and $\mathbf{F}^{\mathrm{flex2-t}}$ are similar to those of $V^{\mathrm{flex2}}$ and $\mathbf{F}^{\mathrm{flex2}}$.

### Usage

To apply enforced rotation, the particles $i$ that are to be subjected to one of the rotation potentials are defined via index groups `rot-group0`, `rot-group1`, etc., in the *mdp* (page 579) input file. The reference positions $\mathbf{y}_i^0$ are read from a special *trr* (page 586) file provided to *grompp* (page 230). If no such file is found, $\mathbf{x}_i(t=0)$ are used as reference positions and written to *trr* (page 586) such that they can be used for subsequent setups. All parameters of the potentials such as $k$, $\epsilon'$, etc. (Table 5.16) are provided as *mdp* (page 579) parameters; `rot-type` selects the type of the potential. The option `rot-massw` allows to choose whether or not to use mass-weighted averaging. For the flexible potentials, a cutoff value $g_n^{\min}$ (typically $g_n^{\min}=0.001$) makes sure that only significant contributions to $V$ and $\mathbf{F}$ are evaluated, *i.e.* terms with $g_n(\mathbf{x})<g_n^{\min}$ are omitted. Table 5.17 summarizes observables that are written to additional output files and which are described below.

Table 5.16: Parameters used by the various rotation potentials. **x** indicate which parameter is actually used for a given potential

| parameter<br>*mdp* (page 579) input variable name<br>unit | | | $k$<br>k<br>$\frac{\text{kJ}}{\text{mol}\cdot\text{nm}^2}$ | $\hat{\mathbf{v}}$<br>vec<br>– | **u**<br>pivot<br>nm | $\omega$<br>rate<br>°/ps | $\epsilon'$<br>eps<br>$\text{nm}^2$ | $\Delta x$<br>slab-dist<br>nm | $g_n^{\min}$<br>min-gauss<br>– |
|---|---|---|---|---|---|---|---|---|---|
| fixed axis potentials: | | eqn. | | | | | | | |
| isotropic | $V^{\text{iso}}$ | (5.335) | **x** | **x** | **x** | **x** | – | – | – |
| — pivot-free | $V^{\text{iso-pf}}$ | (5.339) | **x** | **x** | – | **x** | – | – | – |
| parallel motion | $V^{\text{pm}}$ | (5.343) | **x** | **x** | **x** | **x** | – | – | – |
| — pivot-free | $V^{\text{pm-pf}}$ | (5.345) | **x** | **x** | – | **x** | – | – | – |
| radial motion | $V^{\text{rm}}$ | (5.347) | **x** | **x** | **x** | **x** | – | – | – |
| — pivot-free | $V^{\text{rm-pf}}$ | (5.351) | **x** | **x** | – | **x** | – | – | – |
| radial motion 2 | $V^{\text{rm2}}$ | (5.353) | **x** | **x** | **x** | **x** | **x** | – | – |
| — pivot-free | $V^{\text{rm2-pf}}$ | (5.356) | **x** | **x** | – | **x** | **x** | – | – |
| flexible axis potentials: | | eqn. | | | | | | | |
| flexible | $V^{\text{flex}}$ | (5.365) | **x** | **x** | – | **x** | – | **x** | **x** |
| — transl. tol | $V^{\text{flex-t}}$ | (5.368) | **x** | **x** | – | **x** | – | **x** | **x** |
| flexible 2 | $V^{\text{flex2}}$ | (5.369) | **x** | **x** | – | **x** | **x** | **x** | **x** |
| — transl. tol | $V^{\text{flex2-t}}$ | – | **x** | **x** | – | **x** | **x** | **x** | **x** |

Table 5.17: Quantities recorded in output files during enforced rotation simulations. All slab-wise data is written every `nstsout` steps, other rotation data every `nstrout` steps.

| quantity | unit | equation | output file | fixed | flexible |
|---|---|---|---|---|---|
| $V(t)$ | kJ/mol | see Table 5.16 | `rotation` | **x** | **x** |
| $\theta_{\text{ref}}(t)$ | degrees | $\theta_{\text{ref}}(t) = \omega t$ | `rotation` | **x** | **x** |
| $\theta_{\text{av}}(t)$ | degrees | (5.371) | `rotation` | **x** | – |
| $\theta_{\text{fit}}(t)$, $\theta_{\text{fit}}(t, n)$ | degrees | (5.373) | `rotangles` | – | **x** |
| $\mathbf{y}_0(n)$, $\mathbf{x}_0(t, n)$ | nm | (5.362),(5.363) | `rotslabs` | – | **x** |
| $\tau(t)$ | kJ/mol | (5.374) | `rotation` | **x** | – |
| $\tau(t, n)$ | kJ/mol | (5.374) | `rottorque` | – | **x** |

### Angle of Rotation Groups: Fixed Axis

For fixed axis rotation, the average angle $\theta_{\mathrm{av}}(t)$ of the group relative to the reference group is determined via the distance-weighted angular deviation of all rotation group atoms from their reference positions,

$$\theta_{\mathrm{av}} = \sum_{i=1}^{N} r_i \, \theta_i \Bigg/ \sum_{i=1}^{N} r_i \; . \tag{5.371}$$

Here, $r_i$ is the distance of the reference position to the rotation axis, and the difference angles $\theta_i$ are determined from the atomic positions, projected onto a plane perpendicular to the rotation axis through pivot point $\mathbf{u}$ (see eqn. (5.342) for the definition of $\perp$),

$$\cos\theta_i = \frac{(\mathbf{y}_i - \mathbf{u})^{\perp} \cdot (\mathbf{x}_i - \mathbf{u})^{\perp}}{\|(\mathbf{y}_i - \mathbf{u})^{\perp} \cdot (\mathbf{x}_i - \mathbf{u})^{\perp}\|} \; . \tag{5.372}$$

The sign of $\theta_{\mathrm{av}}$ is chosen such that $\theta_{\mathrm{av}} > 0$ if the actual structure rotates ahead of the reference.

### Angle of Rotation Groups: Flexible Axis

For flexible axis rotation, two outputs are provided, the angle of the entire rotation group, and separate angles for the segments in the slabs. The angle of the entire rotation group is determined by an RMSD fit of $\mathbf{x}_i$ to the reference positions $\mathbf{y}_i^0$ at $t = 0$, yielding $\theta_{\mathrm{fit}}$ as the angle by which the reference has to be rotated around $\hat{\mathbf{v}}$ for the optimal fit,

$$\mathrm{RMSD}\big(\mathbf{x}_i,\ \boldsymbol{\Omega}(\theta_{\mathrm{fit}})\mathbf{y}_i^0\big) \stackrel{!}{=} \min \, . \tag{5.373}$$

To determine the local angle for each slab $n$, both reference and actual positions are weighted with the Gaussian function of slab $n$, and $\theta_{\mathrm{fit}}(t, n)$ is calculated as in eqn. (5.373) from the Gaussian-weighted positions.

For all angles, the *mdp* (page 579) input option `rot-fit-method` controls whether a normal RMSD fit is performed or whether for the fit each position $\mathbf{x}_i$ is put at the same distance to the rotation axis as its reference counterpart $\mathbf{y}_i^0$. In the latter case, the RMSD measures only angular differences, not radial ones.

### Angle Determination by Searching the Energy Minimum

Alternatively, for `rot-fit-method = potential`, the angle of the rotation group is determined as the angle for which the rotation potential energy is minimal. Therefore, the used rotation potential is additionally evaluated for a set of angles around the current reference angle. In this case, the `rotangles.log` output file contains the values of the rotation potential at the chosen set of angles, while `rotation.xvg` lists the angle with minimal potential energy.

### Torque

The torque $\tau(t)$ exerted by the rotation potential is calculated for fixed axis rotation via

$$\tau(t) = \sum_{i=1}^{N} \mathbf{r}_i(t) \times \mathbf{f}_i^{\perp}(t), \tag{5.374}$$

where $\mathbf{r}_i(t)$ is the distance vector from the rotation axis to $\mathbf{x}_i(t)$ and $\mathbf{f}_i^{\perp}(t)$ is the force component perpendicular to $\mathbf{r}_i(t)$ and $\hat{\mathbf{v}}$. For flexible axis rotation, torques $\tau_n$ are calculated for each slab using the local rotation axis of the slab and the Gaussian-weighted positions.

### 5.8.7 Electric fields

A pulsed and oscillating electric field can be applied according to:

$$E(t) = E_0 \exp\left[-\frac{(t-t_0)^2}{2\sigma^2}\right] \cos\left[\omega(t-t_0)\right] \tag{5.375}$$

where $E_0$ is the field strength, the angular frequency $\omega = 2\pi c/\lambda$, $t_0$ is the time at of the peak in the field strength and $\sigma$ is the width of the pulse. Special cases occur when $\sigma$ = 0 (non-pulsed field) and for $\omega$ is 0 (static field). See `electric-field-x` (page 94) for more details.

This simulated laser-pulse was applied to simulations of melting ice *146* (page 690). A pulsed electric field may look like Fig. 5.48. In the supporting information of that paper the impact of an applied electric field on a system under periodic boundary conditions is analyzed. It is described that the effective electric field under PBC is larger than the applied field, by a factor depending on the size of the box and the dielectric properties of molecules in the box. For a system with static dielectric properties this factor can be corrected for. But for a system where the dielectric varies over time, for example a membrane protein with a pore that opens and closes during the simulation, this way of applying an electric field is not useful. In such cases one can use the computational electrophysiology protocol described in the next section (sec. *Computational Electrophysiology* (page 620)).

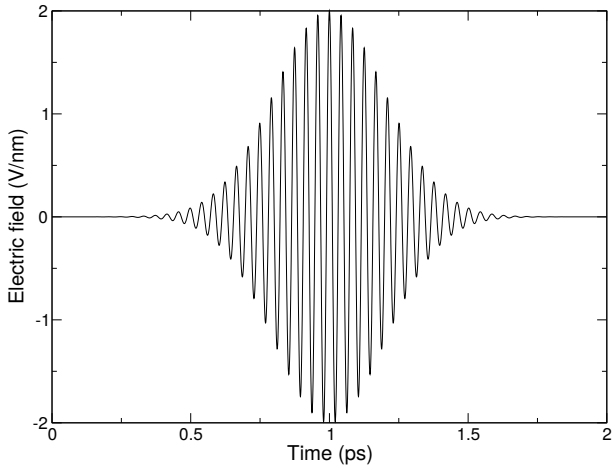



Fig. 5.48: A simulated laser pulse in GROMACS.

Electric fields are applied when the following options are specified in the *grompp* (page 230) *mdp* (page 579) file. You specify, in order, $E_0$, $\omega$, $t_0$ and $\sigma$:

```
electric-field-x = 0.04 0       0     0
```

yields a static field with $E_0$ = 0.04 V/nm in the X-direction. In contrast,

```
electric-field-x = 2.0  150     5     0
```

yields an oscillating electric field with $E_0$ = 2 V/nm, $\omega$ = 150/ps and $t_0$ = 5 ps. Finally

```
electric-field-x = 2.0  150     5     1
```

yields an pulsed-oscillating electric field with $E_0$ = 2 V/nm, $\omega$ = 150/ps and $t_0$ = 5 ps and $\sigma$ = 1 ps. Read more in ref. *146* (page 690). Note that the input file format is changed from the undocumented older version. A figure like Fig. 5.48 may be produced by passing the `-field` option to *gmx mdrun* (page 258).

#### Boundary conditions

In a finite, non-periodic system with plain Coulomb interactions, the application of an electric field is straightforward and one could define a potential energy. But in most cases periodic systems are used. This is problematic, as dipoles will align with the field and build a net dipole in one periodic image. The interactions between this dipole and all its periodic images is a conditionally convergent sum. This leads to the, somewhat strange, effect that the boundary condition at infinity affects the energy of the system and the sampled conformations.

By default, Ewald type electrostatics methods will give a conducting boundary condition. This means that there is no penalty to building up a net dipole. This does not correspond to the situation of putting an electric field on a finite amount of material in an experiment. In fact, the electric field applied in the simulation is larger than that applied to a finite amount of material by a factor of the dielectric constant of the system, which can be rather large. One can correct for this by lowering the applied electric field by the dielectric constant.

When using Ewald type electrostatics, one can directly obtain the correct average polarization in an electric field by using insulating boundary conditions by setting `epsilon-surface` to 1. A disadvantage of this is that the fluctuations of the polarization are suppressed by a factor corresponding to the dielectric constant, at least when the simulated system is supposed to represent a small part of the total system. In practice, insulating boundary conditions can usually not be used, as this is only supported when each molecule is a single update group so molecules are not broken over periodic boundary conditions.

Another issue of periodic boundary conditions is that one can not define a potential energy when charged molecules are present. It would be possible when all molecules are neutral, but in GROMACS this is not done as this would require keeping track of periodic images of parts of molecules. When there are charged molecules in a liquid, a constant electric field will lead to non-equilibrium simulation where the charged molecules move along the field.

It might seem that one can avoid part of these issues by avoiding full-range electrostatics and using reaction-field electrostatics instead. But, apart from the issues with ignoring long-range interactions, there are still similar issues in that the response to the electric field depends on the dielectric permittivity used for the reaction field.

### 5.8.8 Computational Electrophysiology

The Computational Electrophysiology (CompEL) protocol *147* (page 690) allows the simulation of ion flux through membrane channels, driven by transmembrane potentials or ion concentration gradients. Just as in real cells, CompEL establishes transmembrane potentials by sustaining a small imbalance of charges $\Delta q$ across the membrane, which gives rise to a potential difference $\Delta U$ according to the membrane capacitance:

$$\Delta U = \Delta q / C_{membrane} \tag{5.376}$$

The transmembrane electric field and concentration gradients are controlled by *mdp* (page 579) options, which allow the user to set reference counts for the ions on either side of the membrane. If a difference between the actual and the reference numbers persists over a certain time span, specified by the user, a number of ion/water pairs are exchanged between the compartments until the reference numbers are restored. Alongside the calculation of channel conductance and ion selectivity, CompEL simulations also enable determination of the channel reversal potential, an important characteristic obtained in electrophysiology experiments.

In a CompEL setup, the simulation system is divided into two compartments **A** and **B** with independent ion concentrations. This is best achieved by using double bilayer systems with a copy (or copies) of the

channel/pore of interest in each bilayer (Fig. 5.49 A, B). If the channel axes point in the same direction, channel flux is observed simultaneously at positive and negative potentials in this way, which is for instance important for studying channel rectification.

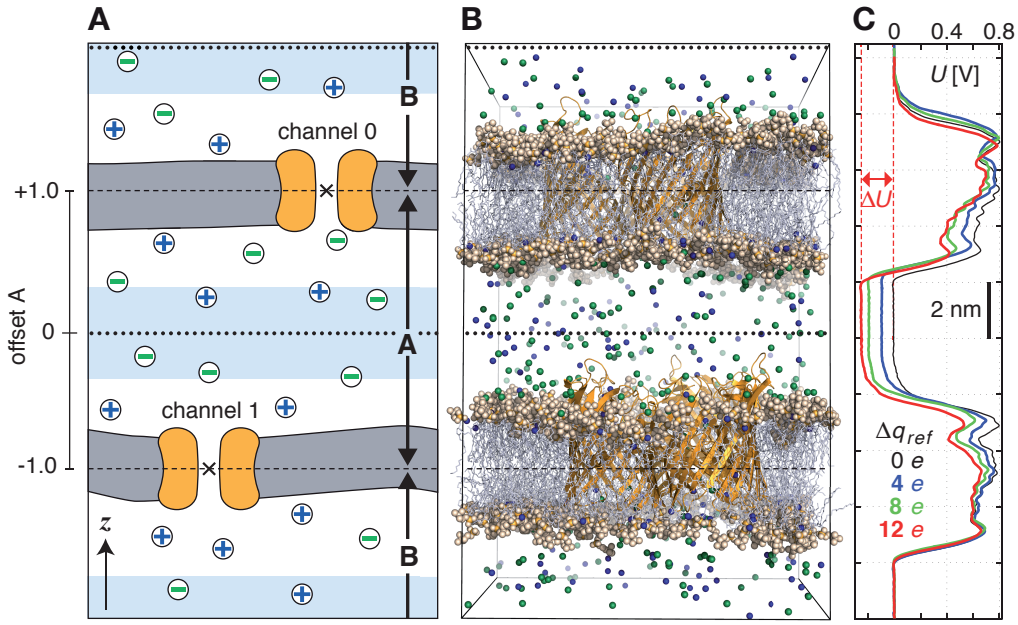



Fig. 5.49: Typical double-membrane setup for CompEL simulations (A, B). Ion/water molecule exchanges will be performed as needed between the two light blue volumes around the dotted black lines (A). Plot (C) shows the potential difference $\Delta U$ resulting from the selected charge imbalance $\Delta q_{ref}$ between the compartments.

The potential difference $\Delta U$ across the membrane is easily calculated with the *gmx potential* (page 287) utility. By this, the potential drop along $z$ or the pore axis is exactly known in each time interval of the simulation (Fig. 5.49 C). Type and number of ions $n_i$ of charge $q_i$, traversing the channel in the simulation, are written to the swapions.xvg output file, from which the average channel conductance $G$ in each interval $\Delta t$ is determined by:

$$G = \frac{\sum_i n_i q_i}{\Delta t \, \Delta U} \, . \tag{5.377}$$

The ion selectivity is calculated as the number flux ratio of different species. Best results are obtained by averaging these values over several overlapping time intervals.

The calculation of reversal potentials is best achieved using a small set of simulations in which a given transmembrane concentration gradient is complemented with small ion imbalances of varying magnitude. For example, if one compartment contains 1M salt and the other 0.1M, and given charge neutrality otherwise, a set of simulations with $\Delta q = 0\,e$, $\Delta q = 2\,e$, $\Delta q = 4\,e$ could be used. Fitting a straight line through the current-voltage relationship of all obtained $I$-$U$ pairs near zero current will then yield $U_{rev}$.

### Usage

The following *mdp* (page 579) options control the CompEL protocol:

```
swapcoords     = Z          ; Swap positions: no, X, Y, Z
swap-frequency = 100        ; Swap attempt frequency
```

Choose `Z` if your membrane is in the $xy$-plane (Fig. 5.49). Ions will be exchanged between compartments depending on their $z$-positions alone. `swap-frequency` determines how often a swap attempt will be made. This step requires that the positions of the split groups, the ions, and possibly the solvent molecules are communicated between the parallel processes, so if chosen too small it can decrease the simulation performance. The `Position swapping` entry in the cycle and time accounting table at the end of the `md.log` file summarizes the amount of runtime spent in the swap module.

```
split-group0   = channel0 ; Defines compartment boundary
split-group1   = channel1 ; Defines other compartment boundary
massw-split0   = no       ; use mass-weighted center?
massw-split1   = no
```

`split-group0` and `split-group1` are two index groups that define the boundaries between the two compartments, which are usually the centers of the channels. If `massw-split0` or `massw-split1` are set to `yes`, the center of mass of each index group is used as boundary, here in $z$-direction. Otherwise, the geometrical centers will be used ($\times$ in Fig. 5.49 A). If, such as here, a membrane channel is selected as split group, the center of the channel will define the dividing plane between the compartments (dashed horizontal lines). All index groups must be defined in the index file.

If, to restore the requested ion counts, an ion from one compartment has to be exchanged with a water molecule from the other compartment, then those molecules are swapped which have the largest distance to the compartment-defining boundaries (dashed horizontal lines). Depending on the ion concentration, this effectively results in exchanges of molecules between the light blue volumes. If a channel is very asymmetric in $z$-direction and would extend into one of the swap volumes, one can offset the swap exchange plane with the `bulk-offset` parameter. A value of 0.0 means no offset $b$, values $-1.0 < b < 0$ move the swap exchange plane closer to the lower, values $0 < b < 1.0$ closer to the upper membrane. Fig. 5.49 A (left) depicts that for the **A** compartment.

```
solvent-group  = SOL      ; Group containing the solvent molecules
iontypes       = 3        ; Number of different ion types to control
iontype0-name  = NA       ; Group name of the ion type
iontype0-in-A  = 51       ; Reference count of ions of type 0 in A
iontype0-in-B  = 35       ; Reference count of ions of type 0 in B
iontype1-name  = K
iontype1-in-A  = 10
iontype1-in-B  = 38
iontype2-name  = CL
iontype2-in-A  = -1
iontype2-in-B  = -1
```

The group name of solvent molecules acting as exchange partners for the ions has to be set with `solvent-group`. The number of different ionic species under control of the CompEL protocol is given by the `iontypes` parameter, while `iontype0-name` gives the name of the index group containing the atoms of this ionic species. The reference number of ions of this type can be set with the `iontype0-in-A` and `iontype0-in-B` options for compartments **A** and **B**, respectively. Obviously, the sum of `iontype0-in-A` and `iontype0-in-B` needs to equal the number of ions in the group

defined by `iontype0-name`. A reference number of `-1` means: use the number of ions as found at the beginning of the simulation as the reference value.

```
coupl-steps     = 10        ; Average over these many swap steps
threshold       = 1         ; Do not swap if < threshold
```

If `coupl-steps` is set to 1, then the momentary ion distribution determines whether ions are exchanged. `coupl-steps > 1` will use the time-average of ion distributions over the selected number of attempt steps instead. This can be useful, for example, when ions diffuse near compartment boundaries, which would lead to numerous unproductive ion exchanges. A `threshold` of 1 means that a swap is performed if the average ion count in a compartment differs by at least 1 from the requested values. Higher thresholds will lead to toleration of larger differences. Ions are exchanged until the requested number $\pm$ the threshold is reached.

```
cyl0-r          = 5.0       ; Split cylinder 0 radius (nm)
cyl0-up         = 0.75      ; Split cylinder 0 upper extension (nm)
cyl0-down       = 0.75      ; Split cylinder 0 lower extension (nm)
cyl1-r          = 5.0       ; same for other channel
cyl1-up         = 0.75
cyl1-down       = 0.75
```

The cylinder options are used to define virtual geometric cylinders around the channel's pore to track how many ions of which type have passed each channel. Ions will be counted as having traveled through a channel according to the definition of the channel's cylinder radius, upper and lower extension, relative to the location of the respective split group. This will not affect the actual flux or exchange, but will provide you with the ion permeation numbers across each of the channels. Note that an ion can only be counted as passing through a particular channel if it is detected *within* the defined split cylinder in a swap step. If `swap-frequency` is chosen too high, a particular ion may be detected in compartment **A** in one swap step, and in compartment **B** in the following swap step, so it will be unclear through which of the channels it has passed.

A double-layered system for CompEL simulations can be easily prepared by duplicating an existing membrane/channel MD system in the direction of the membrane normal (typically $z$) with *gmx editconf* (page 208) `-translate 0 0 <l_z>`, where `l_z` is the box length in that direction. If you have already defined index groups for the channel for the single-layered system, *gmx make_ndx* (page 255) `-n index.ndx -twin` will provide you with the groups for the double-layered system.

To suppress large fluctuations of the membranes along the swap direction, it may be useful to apply a harmonic potential (acting only in the swap dimension) between each of the two channel and/or bilayer centers using umbrella pulling (see section *Collective variables: the pull code* (page 592)).

### Multimeric channels

If a split group consists of more than one molecule, the correct PBC image of all molecules with respect to each other has to be chosen such that the channel center can be correctly determined. GROMACS assumes that the starting structure in the *tpr* (page 586) file has the correct PBC representation. Set the following environment variable to check whether that is the case:

- `GMX_COMPELDUMP`: output the starting structure after it has been made whole to *pdb* (page 581) file.

### 5.8.9 Calculating a PMF using the free-energy code

The free-energy coupling-parameter approach (see sec. *Free energy calculations* (page 469)) provides several ways to calculate potentials of mean force. A potential of mean force between two atoms can be calculated by connecting them with a harmonic potential or a constraint. For this purpose there are special potentials that avoid the generation of extra exclusions, see sec. *Exclusions* (page 544). When the position of the minimum or the constraint length is 1 nm more in state B than in state A, the restraint or constraint force is given by $\partial H/\partial\lambda$. The distance between the atoms can be changed as a function of $\lambda$ and time by setting delta-lambda in the *mdp* (page 579) file. The results should be identical (although not numerically due to the different implementations) to the results of the pull code with umbrella sampling and constraint pulling. Unlike the pull code, the free energy code can also handle atoms that are connected by constraints.

Potentials of mean force can also be calculated using position restraints. With position restraints, atoms can be linked to a position in space with a harmonic potential (see *Position restraints* (page 500)). These positions can be made a function of the coupling parameter $\lambda$. The positions for the A and the B states are supplied to *grompp* (page 230) with the `-r` and `-rb` options, respectively. One could use this approach to do targeted MD; note that we do not encourage the use of targeted MD for proteins. A protein can be forced from one conformation to another by using these conformations as position restraint coordinates for state A and B. One can then slowly change $\lambda$ from 0 to 1. The main drawback of this approach is that the conformational freedom of the protein is severely limited by the position restraints, independent of the change from state A to B. Also, the protein is forced from state A to B in an almost straight line, whereas the real pathway might be very different. An example of a more fruitful application is a solid system or a liquid confined between walls where one wants to measure the force required to change the separation between the boundaries or walls. Because the boundaries (or walls) already need to be fixed, the position restraints do not limit the system in its sampling.

### 5.8.10 Removing fastest degrees of freedom

The maximum time step in MD simulations is limited by the smallest oscillation period that can be found in the simulated system. Bond-stretching vibrations are in their quantum-mechanical ground state and are therefore better represented by a constraint instead of a harmonic potential.

For the remaining degrees of freedom, the shortest oscillation period (as measured from a simulation) is 13 fs for bond-angle vibrations involving hydrogen atoms. Taking as a guideline that with a Verlet (leap-frog) integration scheme a minimum of 5 numerical integration steps should be performed per period of a harmonic oscillation in order to integrate it with reasonable accuracy, the maximum time step will be about 3 fs. Disregarding these very fast oscillations of period 13 fs, the next shortest periods are around 20 fs, which will allow a maximum time step of about 4 fs.

Removing the bond-angle degrees of freedom from hydrogen atoms can best be done by defining them as virtual interaction sites instead of normal atoms. Whereas a normal atom is connected to the molecule with bonds, angles and dihedrals, a virtual site's position is calculated from the position of three nearby heavy atoms in a predefined manner (see also sec. *Virtual interaction sites* (page 520)). For the hydrogens in water and in hydroxyl, sulfhydryl, or amine groups, no degrees of freedom can be removed, because rotational freedom should be preserved. The only other option available to slow down these motions is to increase the mass of the hydrogen atoms at the expense of the mass of the connected heavy atom. This will increase the moment of inertia of the water molecules and the hydroxyl, sulfhydryl, or amine groups, without affecting the equilibrium properties of the system and without affecting the dynamical properties too much. These constructions will shortly be described in sec. *Hydrogen bond-angle vibrations* (page 625) and have previously been described in full detail *148* (page 690).

Using both virtual sites and modified masses, the next bottleneck is likely to be formed by the improper dihedrals (which are used to preserve planarity or chirality of molecular groups) and the peptide dihedrals. The peptide dihedral cannot be changed without affecting the physical behavior of the protein. The improper dihedrals that preserve planarity mostly deal with aromatic residues. Bonds, angles, and dihedrals in these residues can also be replaced with somewhat elaborate virtual site constructions.

All modifications described in this section can be performed using the GROMACS topology building tool *pdb2gmx* (page 281). Separate options exist to increase hydrogen masses, virtualize all hydrogen atoms, or also virtualize the aromatic rings in standard residues. **Note** that when all hydrogen atoms are virtualized, those inside the aromatic residues will be virtualized as well, *i.e.* hydrogens in the aromatic residues are treated differently depending on the treatment of the aromatic residues. Note further that the virtualization of aromatic rings is deprecated.

Parameters for the virtual site constructions for the hydrogen atoms are inferred from the force-field parameters (*vis.* bond lengths and angles) directly by *grompp* (page 230) while processing the topology file. The constructions for the aromatic residues are based on the bond lengths and angles for the geometry as described in the force fields, but these parameters are hard-coded into *pdb2gmx* (page 281) due to the complex nature of the construction needed for a whole aromatic group.

### Hydrogen bond-angle vibrations

### Construction of virtual sites

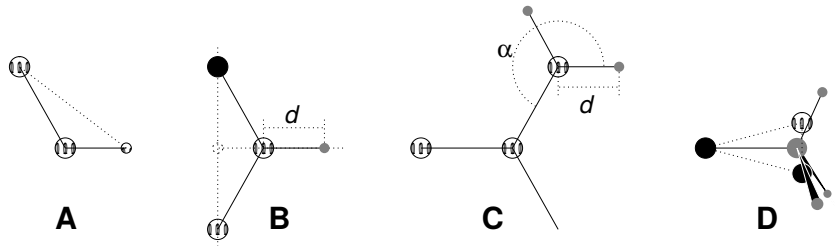



Fig. 5.50: The different types of virtual site constructions used for hydrogen atoms. The atoms used in the construction of the virtual site(s) are depicted as black circles, virtual sites as gray ones. Hydrogens are smaller than heavy atoms. A: fixed bond angle, note that here the hydrogen is not a virtual site; B: in the plane of three atoms, with fixed distance; C: in the plane of three atoms, with fixed angle and distance; D: construction for amine groups (-$NH_2$ or -$NH_3^+$), see text for details.

The goal of defining hydrogen atoms as virtual sites is to remove all high-frequency degrees of freedom from them. In some cases, not all degrees of freedom of a hydrogen atom should be removed, *e.g.* in the case of hydroxyl or amine groups the rotational freedom of the hydrogen atom(s) should be preserved. Care should be taken that no unwanted correlations are introduced by the construction of virtual sites, *e.g.* bond-angle vibration between the constructing atoms could translate into hydrogen bond-length vibration. Additionally, since virtual sites are by definition massless, in order to preserve total system mass, the mass of each hydrogen atom that is treated as virtual site should be added to the bonded heavy atom.

Taking into account these considerations, the hydrogen atoms in a protein naturally fall into several categories, each requiring a different approach (see also Fig. 5.50).

- *hydroxyl (-OH) or sulfhydryl (-SH) hydrogen:* The only internal degree of freedom in a hydroxyl group that can be constrained is the bending of the C-O-H angle. This angle is fixed by defining an additional bond of appropriate length, see Fig. 5.50 A. Doing so removes the high-frequency

angle bending, but leaves the dihedral rotational freedom. The same goes for a sulfhydryl group. **Note** that in these cases the hydrogen is not treated as a virtual site.

- *single amine or amide (-NH-) and aromatic hydrogens (-CH-):* The position of these hydrogens cannot be constructed from a linear combination of bond vectors, because of the flexibility of the angle between the heavy atoms. Instead, the hydrogen atom is positioned at a fixed distance from the bonded heavy atom on a line going through the bonded heavy atom and a point on the line through both second bonded atoms, see Fig. 5.50 B.
- *planar amine (-$NH_2$) hydrogens:* The method used for the single amide hydrogen is not well suited for planar amine groups, because no suitable two heavy atoms can be found to define the direction of the hydrogen atoms. Instead, the hydrogen is constructed at a fixed distance from the nitrogen atom, with a fixed angle to the carbon atom, in the plane defined by one of the other heavy atoms, see Fig. 5.50 C.
- *amine group (umbrella -$NH_2$ or -$NH_3^+$)* hydrogens:** Amine hydrogens with rotational freedom cannot be constructed as virtual sites from the heavy atoms they are connected to, since this would result in loss of the rotational freedom of the amine group. To preserve the rotational freedom while removing the hydrogen bond-angle degrees of freedom, two "dummy masses" are constructed with the same total mass, moment of inertia (for rotation around the C-N bond) and center of mass as the amine group. These dummy masses have no interaction with any other atom, except for the fact that they are connected to the carbon and to each other, resulting in a rigid triangle. From these three particles, the positions of the nitrogen and hydrogen atoms are constructed as linear combinations of the two carbon-mass vectors and their outer product, resulting in an amine group with rotational freedom intact, but without other internal degrees of freedom. See Fig. 5.50 D.

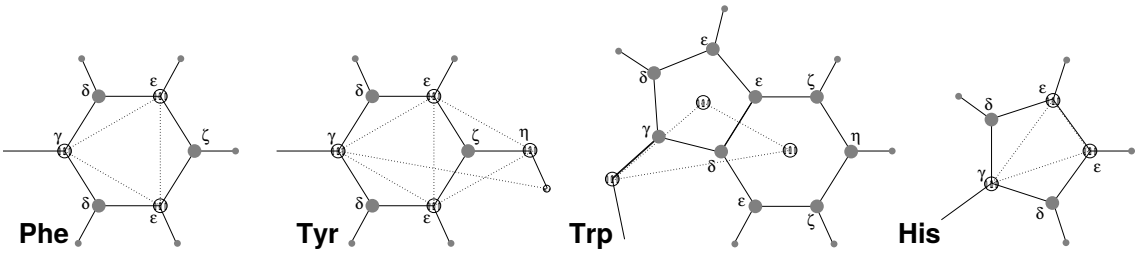



Fig. 5.51: The different types of virtual site constructions used for aromatic residues. The atoms used in the construction of the virtual site(s) are depicted as black circles, virtual sites as gray ones. Hydrogens are smaller than heavy atoms. A: phenylalanine; B: tyrosine (note that the hydroxyl hydrogen is *not* a virtual site); C: tryptophan; D: histidine.

### Out-of-plane vibrations in aromatic groups

The planar arrangements in the side chains of the aromatic residues lends itself perfectly to a virtual-site construction, giving a perfectly planar group without the inherently unstable constraints that are necessary to keep normal atoms in a plane. The basic approach is to define three atoms or dummy masses with constraints between them to fix the geometry and create the rest of the atoms as simple virtual sites type (see sec. *Virtual interaction sites* (page 520)) from these three. Each of the aromatic residues require a different approach:

- *Phenylalanine:* $C_\gamma$, $C_{\epsilon 1}$, and $C_{\epsilon 2}$ are kept as normal atoms, but with each a mass of one third the total mass of the phenyl group. See Fig. 5.50 A.
- *Tyrosine:* The ring is treated identically to the phenylalanine ring. Additionally, constraints are

defined between $C_{\epsilon 1}$, $C_{\epsilon 2}$, and $O_\eta$. The original improper dihedral angles will keep both triangles (one for the ring and one with $O_\eta$) in a plane, but due to the larger moments of inertia this construction will be much more stable. The bond-angle in the hydroxyl group will be constrained by a constraint between $C_\gamma$ and $H_\eta$. **Note** that the hydrogen is not treated as a virtual site. See Fig. 5.50 B.

- *Tryptophan:* $C_\beta$ is kept as a normal atom and two dummy masses are created at the center of mass of each of the rings, each with a mass equal to the total mass of the respective ring ($C_{\delta 2}$ and $C_{\epsilon 2}$ are each counted half for each ring). This keeps the overall center of mass and the moment of inertia almost (but not quite) equal to what it was. See Fig. 5.50 C.
- *Histidine:* $C_\gamma$, $C_{\epsilon 1}$ and $N_{\epsilon 2}$ are kept as normal atoms, but with masses redistributed such that the center of mass of the ring is preserved. See Fig. 5.50 D.

### 5.8.11 Viscosity calculation

The shear viscosity is a property of liquids that can be determined easily by experiment. It is useful for parameterizing a force field because it is a kinetic property, while most other properties which are used for parameterization are thermodynamic. The viscosity is also an important property, since it influences the rates of conformational changes of molecules solvated in the liquid.

The viscosity can be calculated from an equilibrium simulation using an Einstein relation:

$$\eta = \frac{1}{2}\frac{V}{k_B T}\lim_{t\to\infty}\frac{\mathrm{d}}{\mathrm{d}t}\left\langle\left(\int_{t_0}^{t_0+t} P_{xz}(t')\mathrm{d}t'\right)^2\right\rangle_{t_0} \tag{5.378}$$

This can be done with *gmx energy* (page 214). This method converges rather slowly *149* (page 690), and usually hundreds of nanoseconds are needed for an accurate determination of the viscosity. The result is very dependent on the treatment of the electrostatics. Using a (short) cut-off results in large noise on the off-diagonal pressure elements, which can increase the calculated viscosity by an order of magnitude. It is most convenient to use the Einstein relation. Because *gmx mdrun* (page 258) stores averages of quantities computed every `nstcalcenergy` steps, the Einstein relation can use these averages and thus writing to energy file (`nstenergy`) can be done infrequently. This avoids the overhead of very large energy files that are needed with the autocorrelation function approach.

GROMACS also has two non-equilibrium methods for determining the viscosity. The recommended method is to apply a shear to the system, see the next section. The viscosity can the be measured as the stress of the corresponding off-diagonal element of the pressure tensor divided by the shear rate. This is a straightforward procedure. The only disadvantage is that one needs to balance the cost of long simulatiosn at low shear rate due to low signal to noise ratio to the risk of shear thinning appearing at higher shear rates. Running at multiple shear rates might be necessary to ensure that one is in the linear regime.

The second non-equilibrium method is called "cosine acceleration". This makes use of the fact that energy, which is fed into system by external forces, is dissipated through viscous friction. The generated heat is removed by coupling to a heat bath. For a Newtonian liquid adding a small force will result in a velocity gradient according to the following equation:

$$a_x(z) + \frac{\eta}{\rho}\frac{\partial^2 v_x(z)}{\partial z^2} = 0 \tag{5.379}$$

Here we have applied an acceleration $a_x(z)$ in the $x$-direction, which is a function of the $z$-coordinate. In GROMACS the acceleration profile is:

$$a_x(z) = A\cos\left(\frac{2\pi z}{l_z}\right) \tag{5.380}$$

where $l_z$ is the height of the box. The generated velocity profile is:

$$v_x(z) = V \cos\left(\frac{2\pi z}{l_z}\right) \tag{5.381}$$

$$V = A\frac{\rho}{\eta}\left(\frac{l_z}{2\pi}\right)^2 \tag{5.382}$$

The viscosity can be calculated from $A$ and $V$:

$$\eta = \frac{A}{V}\rho\left(\frac{l_z}{2\pi}\right)^2 \tag{5.383}$$

In the simulation $V$ is defined as:

$$V = \frac{\sum_{i=1}^{N} m_i v_{i,x} 2\cos\left(\frac{2\pi z}{l_z}\right)}{\sum_{i=1}^{N} m_i} \tag{5.384}$$

The generated velocity profile is not coupled to the heat bath. Moreover, the velocity profile is excluded from the kinetic energy. One would like $V$ to be as large as possible to get good statistics. However, the shear rate should not be so high that the system gets too far from equilibrium. The maximum shear rate occurs where the cosine is zero, the rate being:

$$\mathrm{sh}_{\max} = \max_z \left|\frac{\partial v_x(z)}{\partial z}\right| = A\frac{\rho}{\eta}\frac{l_z}{2\pi} \tag{5.385}$$

For a simulation with: $\eta = 10^{-3}$ [kgm$^{-1}$s$^{-1}$], $\rho = 10^3$[kgm$^{-3}$] and $l_z = 2\pi$[nm], $\mathrm{sh}_{\max} = 1$[psnm$^{-1}$] $A$. This shear rate should be smaller than one over the longest correlation time in the system. For most liquids, this will be the rotation correlation time, which is around 10 ps. In this case, $A$ should be smaller than 0.1[nmps$^{-2}$]. When the shear rate is too high, the observed viscosity will be too low. Because $V$ is proportional to the square of the box height, the optimal box is elongated in the $z$-direction. In general, a simulation length of 100 ps is enough to obtain an accurate value for the viscosity.

The heat generated by the viscous friction is removed by coupling to a heat bath. Because this coupling is not instantaneous the real temperature of the liquid will be slightly lower than the observed temperature. Berendsen derived this temperature shift *31* (page 684), which can be written in terms of the shear rate as:

$$T_s = \frac{\eta\,\tau}{2\rho\,C_v}\mathrm{sh}_{\max}^2 \tag{5.386}$$

where $\tau$ is the coupling time for the Berendsen thermostat and $C_v$ is the heat capacity. Using the values of the example above, $\tau = 10^{-13}$ [s] and $C_v = 2\cdot 10^3$[J kg$^{-1}$K$^{-1}$], we get: $T_s = 25$[Kps$^{-2}$]$\mathrm{sh}_{\max}^2$. When we want the shear rate to be smaller than $1/10$[ps$^{-1}$], $T_s$ is smaller than 0.25[K], which is negligible.

**Note** that the system has to build up the velocity profile when starting from an equilibrium state. This build-up time is of the order of the correlation time of the liquid.

Two quantities are written to the energy file, along with their averages and fluctuations: $V$ and $1/\eta$, as obtained from ((5.383)).

### 5.8.12 Shear simulations

A common type of non-equilibrium simulations in fluid dynamics and rheology are shearing simulations. These are non-equilibrium simulations where work is performed on the simulation system to achieve a shear flow. This can be used to compute viscosities and friction and to study the effect of shear stress on conformations. In GROMACS there are four different ways to achieve shear flow.

Groups of atoms can be given a constant acceleration, which is effectively a mass-weighted force. This will cause such groups to move with respect to the rest of the system. Care needs to be taken to control the velocity of the center of mass of the system. Normal center of mass motion removal can not be used, as that would affect the flow in the system.

As GROMACS supports general triclinic unit-cell shapes, the unit cell can be deformed to set up a shear flow. This can be achieved either by deforming the unit cell directly using the `deform` option in the *mdp* (page 579) file, or this can be driven by applying an off-diagonal stress through pressure coupling. In the former case, one can measure the viscosity through the stress, in the latter case through measuring the shear rate. Note that the with the `deform` option, *gmx mdrun* (page 258) corrects the velocities of particles that are moved by a periodic vector for the difference in flow velocity. Such a correction is not applied when using a stress to generate a flow.

For measuring the viscosity of simple liquids one can use a cosine-shaped acceleration profile, which can be specified using the `cos-acceleration` option in the *mdp* (page 579) file. As the unit-cell does not deform, this avoids some complications of the other methods. The viscosity is computed on the fly and reported in the energy file.

And finally, there is the case where one wants to study the effect of walls on the flow. In particular, structured walls are of interest, consisting of atoms that can be of any kind. In this case one wants to have walls on two sides of the system, typically in the xy-plane close to z=0 and the box height. The flow is then driven by moving the walls at constant speed by using a constant force. A constant force can be achieved by use of acceleration groups, but that will not allow position restraining atoms in the walls along the direction of the shear, which is needed for some types of walls. For the case of walls where (part of) the atoms are position restrained, a constant speed can be set by using the free-energy lambda-coupling code. To achieve this, you need to supply a second, B-state, position restraint file with the `-r` option of *gmx grompp* (page 230). If you shift the coordinates in this file by 1 nm in the direction of shear, you can set the speed of the walls with the `delta-lambda` option in the *mdp* (page 579) file. Note that this makes lambda increase proportionally with simulation time. There is no limit on magnitude of lambda and periodic shifts of walls are handled correctly. When the position restraint coordinates are shifted by 1 nm, the force on the walls is given directly by $dV/d\lambda$.

A Poiseuille flow is a popular setup in experiments. Unfortunately this is difficult to achieve in simulations. The best would be to, as in experiment, apply a pressure difference over (part of) the simulation box. But that is not easy to set up. One can accelerate all liquid atoms, but this does not guarantee that atoms that interact directly with the wall experience the same forces as they would in an experiment. A slightly better setup would be accelerating only the atoms in the middle of the flow, but spatially defined acceleration groups are currently not supported in GROMACS.

### 5.8.13 Tabulated interaction functions

#### Cubic splines for potentials

In some of the inner loops of GROMACS, look-up tables are used for computation of potential and forces. The tables are interpolated using a cubic spline algorithm. There are separate tables for electrostatic, dispersion, and repulsion interactions, but for the sake of caching performance these have been combined into a single array. The cubic spline interpolation for $x_i \leq x < x_{i+1}$ looks like this:

$$V_s(x) = A_0 + A_1\,\epsilon + A_2\,\epsilon^2 + A_3\,\epsilon^3 \tag{5.387}$$

where the table spacing $h$ and fraction $\epsilon$ are given by:

$$\begin{aligned} h = &\quad x_{i+1} - x_i \\ \epsilon = &\quad (x - x_i)/h \end{aligned} \tag{5.388}$$

so that $0 \leq \epsilon < 1$. From this, we can calculate the derivative in order to determine the forces:

$$-V_s'(x) \;=\; -\frac{\mathrm{d}V_s(x)}{\mathrm{d}\epsilon}\frac{\mathrm{d}\epsilon}{\mathrm{d}x} \;=\; -(A_1 + 2A_2\,\epsilon + 3A_3\,\epsilon^2)/h \tag{5.389}$$

The four coefficients are determined from the four conditions that $V_s$ and $-V_s'$ at both ends of each interval should match the exact potential $V$ and force $-V'$. This results in the following errors for each interval:

$$\begin{aligned} |V_s - V|_{max} = &\quad V'''' \frac{h^4}{384} + O(h^5) \\ |V_s' - V'|_{max} = &\quad V'''' \frac{h^3}{72\sqrt{3}} + O(h^4) \\ |V_s'' - V''|_{max} = &\quad V'''' \frac{h^2}{12} + O(h^3) \end{aligned} \tag{5.390}$$

V and V' are continuous, while V" is the first discontinuous derivative. The number of points per nanometer is 500 and 2000 for mixed- and double-precision versions of GROMACS, respectively. This means that the errors in the potential and force will usually be smaller than the mixed precision accuracy.

GROMACS stores $A_0$, $A_1$, $A_2$ and $A_3$. The force routines get a table with these four parameters and a scaling factor $s$ that is equal to the number of points per nm. (**Note** that $h$ is $s^{-1}$). The algorithm goes a little something like this:

1. Calculate distance vector ($\mathbf{r}_{ij}$) and distance $r_{ij}$
2. Multiply $r_{ij}$ by $s$ and truncate to an integer value $n_0$ to get a table index
3. Calculate fractional component ($\epsilon = sr_{ij} - n_0$) and $\epsilon^2$
4. Do the interpolation to calculate the potential $V$ and the scalar force $f$
5. Calculate the vector force $\mathbf{F}$ by multiplying $f$ with $\mathbf{r}_{ij}$

**Note** that table look-up is significantly *slower* than computation of the most simple Lennard-Jones and Coulomb interaction. However, it is much faster than the shifted Coulomb function used in conjunction with the PPPM method. Finally, it is much easier to modify a table for the potential (and get a graphical representation of it) than to modify the inner loops of the MD program.

### User-specified potential functions

You can also use your own potential functions without editing the GROMACS code. The potential function should be according to the following equation

$$V(r_{ij}) = \frac{q_i q_j}{4\pi\epsilon_0} f(r_{ij}) + C_6\, g(r_{ij}) + C_{12}\, h(r_{ij}) \tag{5.391}$$

where $f$, $g$, and $h$ are user defined functions. **Note** that if $g(r)$ represents a normal dispersion interaction, $g(r)$ should be $< 0$. $C_6$, $C_{12}$ and the charges are read from the topology. Also note that combination rules are only supported for Lennard-Jones and Buckingham, and that your tables should match the parameters in the binary topology.

When you add the following lines in your *mdp* (page 579) file:

```
rlist            = 1.0
coulombtype      = User
rcoulomb         = 1.0
vdwtype          = User
rvdw             = 1.0
```

*mdrun* (page 258) will read a single non-bonded table file, or multiple when `energygrp-table` is set (see below). The name of the file(s) can be set with the *mdrun* (page 258) option `-table`. The table file should contain seven columns of table look-up data in the order: $x$, $f(x)$, $-f'(x)$, $g(x)$, $-g'(x)$, $h(x)$, $-h'(x)$. The $x$ should run from 0 to $r_c+1$ (the value of `table_extension` can be changed in the *mdp* (page 579) file). You can choose the spacing you like; for the standard tables GROMACS uses a spacing of 0.002 and 0.0005 nm when you run in mixed and double precision, respectively. In this context, $r_c$ denotes the maximum of the two cut-offs `rvdw` and `rcoulomb` (see above). These variables need not be the same (and need not be 1.0 either). Some functions used for potentials contain a singularity at $x = 0$, but since atoms are normally not closer to each other than 0.1 nm, the function value at $x = 0$ is not important. Finally, it is also possible to combine a standard Coulomb with a modified LJ potential (or vice versa). One then specifies *e.g.* `coulombtype = Cut-off` or `coulombtype = PME`, combined with `vdwtype = User`. The table file must always contain the 7 columns however, and meaningful data (i.e. not zeroes) must be entered in all columns. A number of pre-built table files can be found in the `GMXLIB` directory for 6-8, 6-9, 6-10, 6-11, and 6-12 Lennard-Jones potentials combined with a normal Coulomb.

If you want to have different functional forms between different groups of atoms, this can be set through energy groups. Different tables can be used for non-bonded interactions between different energy groups pairs through the *mdp* (page 579) option `energygrp-table` (see details in the User Guide). Atoms that should interact with a different potential should be put into different energy groups. Between group pairs which are not listed in `energygrp-table`, the normal user tables will be used. This makes it easy to use a different functional form between a few types of atoms.

## 5.8.14 Hybrid Quantum-Classical simulations (QM/MM) with CP2K interface

In a molecular mechanics (MM) force field, the influence of electrons is expressed by empirical parameters that are assigned on the basis of experimental data, or on the basis of results from high-level quantum chemistry calculations. These are valid for the ground state of a given covalent structure, and the MM approximation is usually sufficiently accurate for ground-state processes in which the overall connectivity between the atoms in the system remains unchanged. However, for processes in which the connectivity does change, such as chemical reactions, or processes that involve multiple electronic

states, such as photochemical conversions, electrons can no longer be ignored, and a quantum mechanical description is required for at least those parts of the system in which the reaction takes place.

One approach to the simulation of chemical reactions in solution, or in enzymes, is to use a combination of quantum mechanics (QM) and molecular mechanics (MM). The reacting parts of the system are treated quantum mechanically, with the remainder being modeled using the force field. The current version of GROMACS provides an interface to the popular Quantum Chemistry package CP2K *188* (page 692).

When using the GROMACS-CP2K interface for QM/MM simulations, please cite Morozov et al., "Seamless QM/MM Simulations via a GROMACS-CP2K Interface" *201* (page 693).

### Overview

GROMACS interactions between the QM and the MM subsystems are handled using the GEEP approach as described by Laino et al. *189* (page 692). This method of evaluating interactions between the QM and MM subsystems is a variant of the "electrostatic embedding" scheme. The electrostatic interactions between the electrons of the QM region and the MM atoms and between the QM nuclei and the MM atoms are explicitly included into the Hamiltonian for the QM subsystem:

$$H^{QM/MM} = H_e^{QM} - \sum_i^n \sum_J^M \frac{e^2 Q_J}{4\pi\epsilon_0 r_{iJ}} + \sum_A^N \sum_J^M \frac{e^2 Z_A Q_J}{4\pi\epsilon_0 R_{AJ}},$$

where $n$ and $N$ are the number of electrons and nuclei in the QM region, respectively, and $M$ is the number of charged MM atoms. The first term on the right hand side is the original electronic Hamiltonian of an isolated QM system. The first of the double sums is the total electrostatic interaction between the QM electrons and the MM atoms. The total electrostatic interaction of the QM nuclei with the MM atoms is given by the second double sum. An important advantage of using the CP2K/GEEP combination is that it allows evaluation of forces for both QM-QM and QM-MM interactions, in the case of systems with periodic boundary conditions (PBC). To avoid double accounting for electrostatic interactions and LJ, classical MM charge on the QM atoms are zeroed out as well as LJ interactions between QM-QM atoms are excluded. It should be noted that LJ interactions between QM-MM atoms are kept and still calculated by GROMACS. Bonded interactions between QM and MM atoms are described at the MM level by the appropriate force-field terms. All bonds, consisting of 2 QM atoms, angles and settles containing 2 or 3 QM atoms, dihedrals containing 3 or 4 QM atoms are excluded from the forcefield evaluation. Broken chemical bonds between QM and MM subsystems needs to be capped in the QM calculation. This is done within CP2K by adding a hydrogen atom to complete the valence of the QM region. The force on this atom, which is present in the QM region only, is distributed over the two atoms of the bond. The cap atom is usually referred to as a link atom. Within the interface all described topology modifications are performed automatically during *gmx grompp* (page 230) pre-processing.

### Software prerequisites

CP2K version 8.1 (or later) should be linked into GROMACS as libcp2k. For a specific installation instructions please follow the *Building with CP2K QM/MM support* (page 26) guide.

### Limitations in simulation techniques

The QM/MM interface limits simulations in two ways. First, no topology modifications are possible during the simulations in the QM region. Second, interface completely ignores "B" state parameters in the topology, making double topology setups impossible, e.g. free-energy perturbation simulations (*Free energy implementation* (page 590)).

In addition it should be noted that the contribution of forces from QM/MM to the system virial are not accounted for. The size of the effect on the pressure-coupling algorithm grows with the total summed force due to QM-MM interactions and might produce artifacts in simulations with the NPT ensemble.

### Usage

QM/MM simulations with CP2K interface are controlled by setting *mdp* (page 579) file options and, in some cases, providing an additional input file for *gmx grompp* (page 230) with the `-qmi` command-line option. All options that are related to QM/MM simulations with CP2K are prefixed with `qmmm-cp2k`.

Setting `qmmm-cp2k-active=true` will trigger a QM/MM simulation using the whole system as QM part and default parameters for all other options.

### Choosing atoms for QM calculation

The QM part of your system is chosen with a name that corresponds to an atom group in the index file of GROMACS to the *qmmm-cp2k-qmgroup* (page 98) option in *mdp* (page 579) file. The typical QM part should consist of atoms that are interesting from the chemical point of view, i.e. part of the system where reaction happens. To make computation of the QM part feasible, it should be small and as compact as possible in a space. DFT simulations often scale as 3rd order of the number of atoms in the QM part. This means increasing number of atoms in the QM part by a factor of 2 will slow down the simulation by a factor of 8.

In addition user should provide total charge of your QM subsystem with *qmmm-cp2k-qmcharge* (page 99) option and spin-state (multiplicity) with *qmmm-cp2k-qmmultiplicity* (page 99) option.

### Supported QM methods

The QM method is chosen with *qmmm-cp2k-qmmethod* (page 98) in the *mdp* (page 579) file. Currently the following QM methods are supported:

1. DFT using the PBE, BLYP, PBE0, B3LYP, CAM-B3LYP, or wB97X functional, with optional D3 dispersion corrections where available.
2. *qmmm-cp2k-qmmethod=SCC-DFTB* (page 99) using CP2K's bundled `DFTB/scc` parameters for C, H, N, O, P, S, and Zn, except for the P–Zn pair.
3. *qmmm-cp2k-qmmethod=GFN1-xTB* (page 99) and *qmmm-cp2k-qmmethod=GFN2-xTB* (page 99) through CP2K's tblite interface.

The generated periodic SCC-DFTB and xTB inputs use the short-range regularized `POINT_CHARGE` coupling in CP2K by default. The *qmmm-cp2k-dftb-electrostatic-coupling* (page 99) option can select GEEP (Gauss) MM charges instead. The QM/MM periodic electrostatics use SPME with `ALPHA 0.35` and box-dependent FFT-friendly mesh sizes. Classical MM electrostatics remain in GROMACS, so the generated CP2K MM Poisson section disables its Ewald sum. CP2K must be built with tblite support for the GFN1-xTB and GFN2-xTB methods, and GEEP (Gauss) tight-binding coupling requires CP2K 2027.1 or higher to support that method. Other DFTB parameter sets can be used by providing a custom CP2K input file.

### Providing your own CP2K input file

In addition it is possible to use custom external CP2K input file with *qmmm-cp2k-qmmethod=INPUT* (page 99) and providing file with *gmx grompp* (page 230) with `-qmi` option. The external file will be incorporated into the *tpr* (page 586) file of the simulation and are subject to the following restrictions:

1. `RUN_TYPE` option in the CP2K input should be equal to `ENERGY_FORCE`.
2. `CHARGE` option should be present.
3. `MULTIPILICTY` option should be present.
4. `COORD_FILE_NAME` option should be present pointing towards *pdb* (page 581) file.
5. Both `CHARGE_EXTENDED TRUE` and `COORD_FILE_FORMAT PDB` options should be present.
6. Incremental includes (`@INCLUDE` directive) are not allowed in the CP2K input file .

### Changing names of CP2K files

During *gmx mdrun* (page 258) simulation additional files will be produced with `.inp`, `.out` and `.pdb`. They contain CP2K input, CP2K output and *pdb* (page 581) file with point charges of MM atoms in the extended beta field. By default all CP2K related files names will be deduced from *tpr* (page 586) simulation file name by adding `_cp2k` suffix. In order to change it manually *qmmm-cp2k-qmfilenames* (page 99) option should be used.

### Output

The energy output file will contain an additional "Quantum En." term. This is the energy that is added to the system from the QM/MM interactions. In addition, a file containing CP2K output will appear in the simulation directory with the `.out` extension.

### Future developments

Support of the multiple time-stepping approach to speed-up simulation will be added. Excited state simulations will be implemented with TD-DFT description of the wavefunction.

### 5.8.15 MiMiC Hybrid Quantum Mechanical/Molecular Mechanical simulations

This section describes the coupling to a novel QM/MM interface. The Multiscale Modeling in Computational Chemistry (MiMiC) interface combines GROMACS with the CPMD QM code. To find information about other QM/MM implementations in GROMACS please refer to the section *Hybrid Quantum-Classical simulations (QM/MM) with CP2K interface* (page 631). Within a QM/MM approach, typically a small part of the system (e.g. active site of an enzyme where a chemical reaction can take place) is treated at the QM level of theory (as we cannot neglect electronic degrees of freedom while describing some processes e.g. chemical reactions), while the rest of the system (remainder of the protein, solvent, etc.) is described by the classical forcefield (MM).

#### Overview

MiMiC implements the QM/MM coupling scheme developed by the group of Prof. U. Roethlisberger described in *180* (page 692). This additive scheme uses electrostatic embedding of the classical system within the quantum Hamiltonian. The total QM/MM energy is calculated as a sum of subsystem contributions:

$$E_{tot} = E_{QM} + E_{MM} + E_{QM/MM}$$

The QM contribution is computed by CPMD, while the MM part is processed by GROMACS and the cross terms are treated by the MiMiC interface. Cross terms, i.e. the terms involving simultaneously atoms from the QM region and atoms from the MM region consist of both bonded and non-bonded interactions.

The bonded interactions are taken from the forcefield used to describe the MM part. Whenever there is a chemical bond crossing the QM/MM boundary additional care has to be taken to handle this situation correctly. Otherwise the QM atom involved in the cut bond is left with an unsaturated electronic orbital leading to unphysical system behaviour. Therefore, the dangling bond has to be capped with another QM atom. There are two different options available in CPMD for bond capping:

1. Hydrogen capping - the simplest approach is to cap the bond with a hydrogen atom, constraining its relative position
2. Link atom pseudo-potential - this strategy uses an ad-hoc pseudo-potential developed to cap the bond. This pseudo-potential would represent the real atom and, thus, will not require the bond constraint.

As in standard forcefields, the non-bonded contributions to $E_{QM/MM}$ can be separated into van der Waals and electrostatic contributions. The first contribution is again taken from the MM forcefield. The second part of non-bonded interactions is handled by MiMiC within the electrostatic embedding approach. This adds additional terms to the Hamiltonian of the system:

$$E^{es}_{QM/MM} = -\sum_{a}^{N_{mm}} Q_a \int \rho(\mathbf{r}) \frac{r_{c,a}^4 - |\mathbf{R_a} - \mathbf{r}|^4}{r_{c,a}^5 - |\mathbf{R_a} - \mathbf{r}|^5} d\mathbf{r} + \sum_{a}^{N_{mm}} \sum_{n}^{N_{qm}} Q_a Z_n \frac{r_{c,a}^4 - |\mathbf{R_a} - \mathbf{R_n}|^4}{r_{c,a}^5 - |\mathbf{R_a} - \mathbf{R_n}|^5}$$

where $N_{mm}$ is a number of MM atoms $N_{qm}$, is the number of QM atoms and $r_{c,a}$ is the covalent radius of the MM atoms. The first term above corresponds to the damped Coulomb interaction between the eletronic density $\rho(\mathbf{r})$ of the QM region and the MM atoms. The damping is needed due to the fact

that CPMD uses a plane-wave basis set to expand the electronic wavefunction. Unlike localized basis sets, plane waves are delocalized and this may give a rise to the so-called electron spill-out problem: positively charged MM atoms may artificially overpolarize the electronic cloud due to the absence of quantum mechanical effects (e.g. Pauli repusion) that would normally prevent it (in a fully quantum system). This functional form of the damped Coulomb potential from the equation above was introduced in *180* (page 692).

Since computing the integrals in the first term above can be computational extremely expensive, MiMiC also implements hierarchical electrostatic embedding scheme in order to mitigate the enormous computational effort needed to compute $N_{mm}$ integrals over the electronic grid. Within this scheme the MM atoms are grouped into two shells according to the distance from the QM region: the short-ranged and long-ranged one. For the MM atoms in the short-ranged shell the QM/MM interactions are calculated using the equation above. In contrast to that, the interactions involving MM atoms from the long-ranged shell are computed using the multipolar expansion of the QM electrostatic potential. More details about it can be found in *180* (page 692).

### Application coupling model

Unlike the majority of QM/MM interfaces, MiMiC uses a loose coupling between partner codes. This means that instead of compiling both codes into a single binary MiMiC builds separate executables for CPMD and GROMACS. The user will then prepare the input for both codes and run them simultaneously. Each of the codes is running using a separate pool of MPI processes and communicate the necessary data (e.g. coordinates, energies and forces) through MPI client-server mechanism. Within MiMiC framework CPMD acts as a server and GROMACS becomes the client.

### Software prerequisites

1. GROMACS version 2019+. Newer major releases may support multiple versions of MiMiC.
2. CPMD version 4.1+.

### Usage

After *Building with MiMiC QM/MM support* (page 26), to run a MiMiC QM/MM simulation one needs to:

1. Get and compile CPMD with MiMiC support.
2. Do a normal classical equilibration with GROMACS.
3. Create an index group representing QM atoms within GROMACS. Keep in mind that this group should also include link atoms bound to atoms in the QM region, as they have to be treated at quantum level.
4. Prepare input for CPMD and GROMACS according to the recommendations below.
5. Run both CPMD and GROMACS as two independent instances within a single batch job.

### Preparing the input file for GROMACS

In order to setup the *mdp* (page 579) file for a MiMiC simulation one needs to add two options:

1. *integrator=mimic* (page 53) to enable MiMiC workflow within GROMACS.
2. QMMM-grps=<name_of_qm_index_group> to indicate all the atoms that are going to be handled by CPMD.

Since CPMD is going to perform the MD integration, only *mdp* (page 579) options relating to force calculation and output are active.

After setting up the *mdp* (page 579) file one can run *grompp* (page 230) as usual. *grompp* (page 230) will set the charges of all the QM atoms to zero to avoid double-counting of Coulomb interactions. Moreover, it will update non-bonded exclusion lists to exclude LJ interactions between QM atoms (since they will be described by CPMD). Finally, it will remove bonds between QM atoms (if present). We recommend to output the preprocessed topology file using `gmx grompp -pp <preprocessed_topology_file>` as it will help to prepare the input for CPMD in an automated way.

### Preparing the input file for CPMD

This section will only describe the MiMiC-related input in CPMD - for the configuration of a DFT-related options - please refer to the CPMD manual. After preparing the input for GROMACS and having obtained the preprocessed topology file, simply run the Python preprocessor script provided within the MiMiC distribution to obtain MiMiC-related part of the CPMD input file. The usage of the script is simple:

```
prepare-qmmm.py <index_file> <gro_file> <preprocessed_topology_file>
 ↪ <qm_group_name>
```

Be advised that for MiMiC it is crucial that the forcefield contains the data about the element number of each atom type! If it does not provide it, the preprocessor will fail with the error:

```
It looks like the forcefield that you are using has no information␣
 ↪about the element number.
The element number is needed to run QM/MM simulations.
```

Given all the relevant information the script will print the part of the CPMD input that is related to MiMiC. Here is the sample output with the short descriptions of keywords that can be found in this part of CPMD input:

```
&MIMIC
PATHS
1
<some_absolute_path>
BOX
35.77988547402689 35.77988547402689 35.77988547402689
OVERLAPS
3
2 13 1 1
2 14 1 2
2 15 1 3
&END
```

(continues on next page)

(continued from previous page)

```
&ATOMS
O
1
17.23430225802002 17.76342557295923 18.576007806615877
H
2
18.557110545368047 19.086233860307257 18.727185896598506
17.57445296048094 16.705178943080806 17.06422690678956
&END
Suggested QM box size [12.661165036045407, 13.71941166592383, 13.
↪00131573850633]
```

`&MIMIC` section contains MiMiC settings:

> `PATHS` indicates number of MM client codes involved in the simulation and the absolute path to each of their respective folder. Keep in mind that this path has to point to the folder, where GROMACS is going to be run – otherwise it will cause a deadlock in CPMD! The next line contains the number of MM codes (1 in this case) and next $N$ lines contain paths to their respective working directories
>
> `BOX` indicates the size of the whole simulation box in Bohr in an `X Y Z` format
>
> `OVERLAPS` - sets the number and IDs of atoms within GROMACS that are going to be treated by CPMD. The format is the following:
>
> ```
> <code_id> <atom_id_in_code> <host_code_id> <atom_id_in_that_
> ↪code>
> ```
>
> CPMD host code id is always ID 1. Therefore, in a QM/MM simulation GROMACS will have code ID 2.
>
> (OPTIONAL) `LONG-RANGE COUPLING` - enables the faster multipole coupling for atoms located at a certain distance from the QM box
>
> (OPTIONAL) `CUTOFF DISTANCE` - the next line contains the cutoff for explicit Coulomb coupling (20 Bohr by default if `LONG-RANGE COUPLING` is present)
>
> (OPTIONAL) `MULTIPOLE ORDER` - The next line will contain the order at which the multipolar expansion will be truncated (default 2, maximum 20).

The `&ATOMS` section of CPMD input file contains all the QM atoms within the system and has a default CPMD formatting. Please refer to the CPMD manual to adjust it to your needs(one will need to set the correct pseudo-potential for each atom species).

Finally, the preprocessor suggests the size of the QM box where the electronic density is going to be contained. The suggested value is not final - further adjustment by user may be required.

### Running a MiMiC QM/MM simulation

In order to run the simulation, one will need to run both GROMACS and CPMD within one job. This is easily done within the vast majority of queueing systems. For example in case of SLURM queue system one can use two job steps within one job. Here is the example job script running a 242-node slurm job, allocating 2 nodes to GROMACS and 240 nodes to CPMD (both codes are launched in the same folder):

```
#!/bin/bash -x
#SBATCH --nodes=242
#SBATCH --output=mpi-out.%j
#SBATCH --error=mpi-err.%j
#SBATCH --time=00:25:00
#SBATCH --partition=batch

# *** start of job script ***

srun -N2 --ntasks-per-node=6 --cpus-per-task=4 -r0 gmx_mpi_d mdrun -
↪deffnm mimic -ntomp 4 &
srun -N240 --ntasks-per-node=6 --cpus-per-task=4 -r2 cpmd.x␣
↪benchmark.inp <path_to_pp_folder> > benchmark-240-4.out &
wait
```

### Known Issues

OpenMPI prior to version 3.x.x has a bug preventing the usage of MiMiC completely - please use newer versions or other MPI distributions.

With IntelMPI communication between CPMD and GROMACS may result in a deadlock in some situations. If it happens, setting an IntelMPI-related environment variable may help:

```
export FI_OFI_RXM_USE_SRX=1
```

## 5.8.16 Using VMD plug-ins for trajectory file I/O

GROMACS tools are able to use the plug-ins found in an existing installation of VMD in order to read and write trajectory files in formats that are not native to GROMACS. You will be able to supply an AMBER DCD-format trajectory filename directly to GROMACS tools, for example.

This requires a VMD installation not older than version 1.8, that your system provides the dlopen function so that programs can determine at run time what plug-ins exist, and that you build shared libraries when building GROMACS. CMake will find the vmd executable in your path, and from it, or the environment variable `VMDDIR` at configuration or run time, locate the plug-ins. Alternatively, the `VMD_PLUGIN_PATH` can be used at run time to specify a path where these plug-ins can be found. Note that these plug-ins are in a binary format, and that format must match the architecture of the machine attempting to use them.

### 5.8.17 Interactive Molecular Dynamics

GROMACS supports the interactive molecular dynamics (IMD) protocol as implemented by VMD to control a running simulation in NAMD. IMD allows to monitor a running GROMACS simulation from a VMD client. In addition, the user can interact with the simulation by pulling on atoms, residues or fragments with a mouse or a force-feedback device. Additional information about the GROMACS implementation and an exemplary GROMACS IMD system can be found on this homepage.

#### Simulation input preparation

The GROMACS implementation allows transmission and interaction with a part of the running simulation only, e.g. in cases where no water molecules should be transmitted or pulled. The group is specified via the *mdp* (page 579) option `IMD-group`. When `IMD-group` is empty, the IMD protocol is disabled and cannot be enabled via the switches in *mdrun* (page 258). To interact with the entire system, `IMD-group` can be set to `System`. When using *grompp* (page 230), a *gro* (page 577) file to be used as VMD input is written out (`-imd` switch of *grompp* (page 230)).

#### Starting the simulation

Communication between VMD and GROMACS is achieved via TCP sockets and thus enables controlling an *mdrun* (page 258) running locally or on a remote cluster. The port for the connection can be specified with the `-imdport` switch of *mdrun* (page 258), 8888 is the default. If a port number of 0 or smaller is provided, GROMACS automatically assigns a free port to use with IMD.

Every $N$ steps, the *mdrun* (page 258) client receives the applied forces from VMD and sends the new positions to the client. VMD permits increasing or decreasing the communication frequency interactively. By default, the simulation starts and runs even if no IMD client is connected. This behavior is changed by the `-imdwait` switch of *mdrun* (page 258). After startup and whenever the client has disconnected, the integration stops until reconnection of the client. When the `-imdterm` switch is used, the simulation can be terminated by pressing the stop button in VMD. This is disabled by default. Finally, to allow interacting with the simulation (i.e. pulling from VMD) the `-imdpull` switch has to be used. Therefore, a simulation can only be monitored but not influenced from the VMD client when none of `-imdwait`, `-imdterm` or `-imdpull` are set. However, since the IMD protocol requires no authentication, it is not advisable to run simulations on a host directly reachable from an insecure environment. Secure shell forwarding of TCP can be used to connect to running simulations not directly reachable from the interacting host. Note that the IMD command line switches of *mdrun* (page 258) are hidden by default and show up in the help text only with *gmx mdrun* (page 258) `-h -hidden`.

#### Connecting from VMD

In VMD, first the structure corresponding to the IMD group has to be loaded (*File* → *New Molecule*). Then the IMD connection window has to be used (*Extensions* → *Simulation* → *IMD Connect (NAMD)*). In the IMD connection window, hostname and port have to be specified and followed by pressing *Connect*. *Detach Sim* allows disconnecting without terminating the simulation, while *Stop Sim* ends the simulation on the next neighbor searching step (if allowed by `-imdterm`).

The timestep transfer rate allows adjusting the communication frequency between simulation and IMD client. Setting the keep rate loads every $N^{\text{th}}$ frame into VMD instead of discarding them when a new one is received. The displayed energies are in SI units in contrast to energies displayed from NAMD simulations.

### 5.8.18 Embedding proteins into the membranes

GROMACS is capable of inserting the protein into pre-equilibrated lipid bilayers with minimal perturbation of the lipids using the method, which was initially described as a ProtSqueeze technique, *157* (page 691) and later implemented as g_membed tool *158* (page 691). Currently the functionality of g_membed is available in mdrun as described in the user guide.

This method works by first artificially shrinking the protein in the $xy$-plane, then it removes lipids that overlap with that much smaller core. Then the protein atoms are gradually resized back to their initial configuration, using normal dynamics for the rest of the system, so the lipids adapt to the protein. Further lipids are removed as required.

### 5.8.19 Applying forces from three-dimensional densities

In density-guided simulations, additional forces are applied to atoms that depend on the gradient of similarity between a simulated density and a reference density.

By applying these forces protein structures can be made to "fit" densities from, e.g., cryo electron-microscopy. The implemented approach extends the ones described in *192* (page 693), and *193* (page 693).

#### Overview

The forces that are applied depend on:

- The forward model that describes how atom positions are translated into a simulated density, $\rho^{\text{sim}}(\mathbf{r})$.
- The similarity measure that describes how close the simulated density is to the reference density, $\rho^{\text{ref}}$, $S[\rho^{\text{ref}}, \rho^{\text{sim}}(\mathbf{r})]$.
- The scaling of these forces by a force constant, $k$.

The resulting effective potential energy is

$$U = U_{\text{forcefield}}(\mathbf{r}) - kS[\rho^{\text{ref}}, \rho^{\text{sim}}(\mathbf{r})]\,. \tag{5.392}$$

The corresponding density based forces that are added during the simulation are

$$\mathbf{F}_{\text{density}} = k\nabla_{\mathbf{r}} S[\rho^{\text{ref}}, \rho^{\text{sim}}(\mathbf{r})]\,. \tag{5.393}$$

This derivative decomposes into a similarity measure derivative and a simulated density model derivative, summed over all density voxels $\mathbf{v}$

$$\mathbf{F}_{\text{density}} = k\sum_{\mathbf{v}} \partial_{\rho^{\text{sim}}_{\mathbf{v}}} S[\rho^{\text{ref}}, \rho^{\text{sim}}] \cdot \nabla_{\mathbf{r}} \rho^{\text{sim}}_{\mathbf{v}}(\mathbf{r})\,. \tag{5.394}$$

Thus density-guided simulation force calculations are based on computing a simulated density and its derivative with respect to the atom positions, as well as a density-density derivative between the simulated and the reference density.

#### Usage

Density-guided simulations are controlled by setting `.mdp` options and providing a reference density map as a file additional to the `.tpr`.

All options that are related to density-guided simulations are prefixed with `density-guided-simulation`.

Setting `density-guided-simulation-active = yes` will trigger density-guided simulations with default parameters that will cause atoms to move into the reference density.

### The simulated density and its force contribution

Atoms are spread onto the regular three-dimensional lattice of the reference density. For spreading the atoms onto the grid, the discrete Gauss transform is used. The simulated density from atoms at positions $\mathbf{r_i}$ at a voxel with coordinates $\mathbf{v}$ is

$$\rho_{\mathbf{v}} = \sum_i A_i \frac{1}{\sqrt{2\pi}^3 \sigma^3} \exp[-\frac{(\mathbf{r_i} - \mathbf{v})^2}{2\sigma^2}] \,. \tag{5.395}$$

Where $A_i$ is an amplitude that is determined per atom type and may be the atom mass, partial charge, or unity for all atoms.

The width of the Gaussian spreading function is determined by $\sigma$. It is not recommended to use a spreading width that is smaller than the grid spacing of the reference density.

The factor for the density force is then

$$\nabla_r \rho^{\mathrm{sim}}_{\mathbf{v}}(\mathbf{r}) = \sum_i -A_i \frac{(\mathbf{r_i} - \mathbf{v})}{\sigma} \frac{1}{\sqrt{2\pi}^3 \sigma^3} \exp[-\frac{(\mathbf{r_i} - \mathbf{v})^2}{2\sigma^2}] \,. \tag{5.396}$$

### The density similarity measure and its force contribution

There are multiple valid similarity measures between the reference density and the simulated density, each motivated by the experimental source of the reference density data. For the density-guided simulations in GROMACS, the following measures are provided:

The inner product of the simulated density,

$$S_{\mathrm{inner-product}}[\rho^{\mathrm{ref}}, \rho^{\mathrm{sim}}] = \frac{1}{N_{\mathrm{voxel}}} \sum_{v=1}^{N_{\mathrm{voxel}}} \rho_v^{\mathrm{ref}} \rho_v^{\mathrm{sim}} \,. \tag{5.397}$$

The negative relative entropy between two densities,

$$S_{\mathrm{relative-entropy}}[\rho^{\mathrm{ref}}, \rho^{\mathrm{sim}}] = \sum_{v=1, \rho^{\mathrm{ref}}>0, \rho^{\mathrm{sim}}>0}^{N_{\mathrm{voxel}}} \rho^{\mathrm{ref}} [\log(\rho_v^{\mathrm{sim}}) - \log(\rho_v^{\mathrm{ref}})] \,. \tag{5.398}$$

The cross correlation between two densities,

$$S_{\mathrm{cross-correlation}}[\rho^{\mathrm{ref}}, \rho^{\mathrm{sim}}] = \frac{\sum_v \left( (\rho_v^{\mathrm{ref}} - \bar{\rho}^{\mathrm{ref}})(\rho_v^{\mathrm{sim}} - \bar{\rho}^{\mathrm{sim}}) \right)}{\sqrt{\sum_v (\rho_v^{\mathrm{ref}} - \bar{\rho}^{\mathrm{ref}})^2 \sum_v (\rho_v^{\mathrm{sim}} - \bar{\rho}^{\mathrm{sim}})^2}} . \tag{5.399}$$

### Declaring regions to fit

A subset of atoms may be chosen when pre-processing the simulation to which the density-guided simulation forces are applied. Only these atoms generate the simulated density that is compared to the reference density.

### Performance

The following factors affect the performance of density-guided simulations

- Number of atoms in the density-guided simulation group, $N_{\mathrm{atoms}}$.
- Spreading range in multiples of Gaussian width, $N_\sigma$.
- The ratio of spreading width to the input density grid spacing, $r_\sigma$.
- The number of voxels of the input density, $N_{\mathrm{voxel}}$.
- Frequency of force calculations, $N_{\mathrm{force}}$.
- The communication cost when using multiple ranks, that is reflected in a constant $c_{\mathrm{comm}}$.

The overall cost of the density-guided simulation is approximately proportional to

$$\frac{1}{N_{\mathrm{force}}}\left[N_{\mathrm{atoms}}\left(N_\sigma r_\sigma\right)^3 + c_{\mathrm{comm}} N_{\mathrm{voxel}}\right] . \tag{5.400}$$

### Applying force every N-th step

The cost of applying forces every integration step is reduced when applying the density-guided simulation forces only every $N$ steps. The applied force is scaled by $N$ to approximate the same effective Hamiltonian as when applying the forces every step, while maintaining time-reversibility and energy conservation. Note that for this setting, the energy output frequency must be a multiple of $N$.

The maximal time-step should not exceed the fastest oscillation period of any atom within the map potential divided by $\pi$. This oscillation period depends on the choice of reference density, the similarity measure and the force constant and is thus hard to estimate directly. It has been observed to be in the order of picoseconds for typical cryo electron-microscopy data, resulting in a `density-guided-simulation-nst` (page 97) setting in the order of 100.

### Combining density-guided simulations with pressure coupling

Note that the contribution of forces from density-guided simulations to the system virial are not accounted for. The size of the effect on the pressure-coupling algorithm grows with the total summed density-guided simulation force, as well as the angular momentum introduced by forces from density-guided simulations. To minimize this effect, align your structure to the density before running a pressure-coupled simulation.

Additionally, applying force every N-th steps does not work with the current implementation of infrequent evaluation of pressure coupling and the constraint virial.

### Periodic boundary condition treatment

Of all periodic images only the one closest to the center of the density map is considered.

### The reference density map format

Reference input for the densities are given in mrc format according to the "EMDB Map Distribution Format Description Version 1.01 (c) emdatabank.org 2014". Closely related formats like `ccp4` and `map` might work.

Be aware that different visualization software handles map formats differently. During simulations, reference densities are interpreted as visualised by `VMD`.

### Output

The energy output file will contain an additional "Density-fitting" term. This is the energy that is added to the system from the density-guided simulations. The lower the energy, the higher the similarity between simulated and reference density.

### Adaptive force constant scaling

To enable a steady increase in similarity between reference and simulated density while using as little force as possible, adaptive force scaling decreases the force constant when similarity increases and vice versa. To avoid large fluctuations in the force constant, change in similarity is measured with an exponential moving average that smoothens the time series of similarity measures with a time constant $tau$ that is given in ps. If the exponential moving average similarity increases, the force constant is scaled down by dividing by $1 + \delta t_{\text{density}}/tau$, where $\delta t_{\text{density}}$ is the time between density guided simulation steps. Conversely, if similarity between reference and simulated density is decreasing, the force constant is increased by multiplying by $1 + 2\delta t_{\text{density}}/tau$. Note that adaptive force scaling does not conserve energy and will ultimately lead to very high forces when similarity cannot be increased further.

### Mapping input structure to density data with affine transformations

To align input structure and density data, a transformation matrix $\mathbf{A}$ and shift vector $\mathbf{v}_{\text{shift}}$ may be defined that transform the input structure atom coordinates before evaluating density-guided-simulation energies and forces, so that

$$U = U_{\text{forcefield}}(\mathbf{r}) - kS[\rho^{\text{ref}}, \rho^{\text{sim}}(\mathbf{Ar} + \mathbf{v}_{\text{shift}})]\,. \tag{5.401}$$

$$\mathbf{F}_{\text{density}} = k\nabla_{\mathbf{r}} S[\rho^{\text{ref}}, \rho^{\text{sim}}(\mathbf{Ar} + \mathbf{v}_{\text{shift}})]\,. \tag{5.402}$$

Affine transformations may be used, amongst other things, to perform

- rotations, e.g., around the z-axis by an angle $\theta$ by using $A = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$.
- projection, e.g., onto the z-axis by using $A = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. This allows density-guided simulations to be steered by a density-profile along this axis.
- scaling the structure against the density by a factor $s$ by using $A = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{pmatrix}$. This proves useful when, e.g., voxel-sizes in cryo-EM densities have to be adjusted.

- and arbitrary combinations of these by matrix multiplications (note that matrix multiplications are not commutative).

#### Future developments

Further similarity measures might be added in the future, along with different methods to determine atom amplitudes. More automation in choosing a force constant as well as alignment of the input density map to the structure might be provided.

### 5.8.20 Collective Variable simulations with the Colvars module

The Colvars module enables on-the-fly computation of low-dimension quantities (collective variables or colvars) in simulations, the application of external forces (biases) to these colvars for restraining or enhanced sampling purposes, and the computation of free energy profiles and other properties. The Colvars library and module are described in ref. *195* (page 693) as well as in other references that are reported in the log file when the corresponding features are used.

#### Using Colvars

Colvars simulations are enabled by the following *mdp* (page 579) file options: `colvars-active`, `colvars-configfile`, and `colvars-seed`.

Setting `colvars-active = true` enables Colvars, using a configuration that can be defined by specifying a Colvars configuration file using `colvars-configfile`.

See *this section of the documentation* (page 100) for detailed usage of these options.

#### Configuration files for input

Colvars configuration files are plain text files describing specific collective variables and biasing and analysis algorithms to be applied onto them. Full documentation is available here.

Additionally, the Colvars Dashboard extension within VMD can be used to prepare a Colvars configuration file, leveraging input templates for many features; VMD version 1.9.4 or later is strongly recommended.

#### Colvars output files

When Colvars is active, additional output files are written during a GROMACS simulation. Their file names share the same prefix as the *edr* (page 576) file, which is `ener` by default, followed by a suffix specific to their content (e.g. `ener.colvars.traj` for the trajectory of the collective variables). These files are useful for analysis purposes, but they are not required for continuing a simulation, because the relevant data from Colvars is included in the binary checkpoint file.

#### Colvars checkpointing

The state of the Colvars library is written to the checkpoint file and read automatically upon restarting.

### 5.8.21 Using PLUMED

PLUMED functionality is enabled by using `gmx mdrun -plumed plumed.dat`.

The interface will look for an environment variable `PLUMED_KERNEL` that should contain the path and the name of a shared object that contains the PLUMED kernel, and usually is called `libPlumedKernel.so`.

If the library is not present an error message will inform the user to export the `PLUMED_KERNEL` variable.

Usually the PLUMED kernel is stored in `$plumed_prefix/lib/libPlumedKernel.so`, so it should be enough to `export PLUMED_KERNEL=$plumed_prefix/lib/libPlumedKernel.so`, where `$plumed_prefix` is the PLUMED installation prefix.

#### Configuration files for input

PLUMED input files are plain text files describing the analysis process and how PLUMED should influence the GROMACS simulation. The full documentation is available here

#### Limitations

The current implementation of the PLUMED interface does not support the following features:

- interaction with GROMACS energy
- PLUMED is incompatible with thread-MPI (PLUMED exits with an error when more than 1 thread-MPI rank is used)
- The various replica exchange flavors are not yet implemented

### 5.8.22 Neural Network Potentials

The NNPot module allows the use of potentials based on neural networks or machine learning in general. These models can be trained to reproduce forces and energies at *ab initio* levels of accuracy, based on training data from electronic structure calculations with e.g. DFT or CCSD(T). They take atomic descriptors as their input, most commonly the atom positions and atomic numbers, and output a value for the potential energy of the given conformation. The forces can then be calculated by backpropagating the gradients through the machine learning model (e.g. PyTorchs AutoGrad engine), w.r.t. the inputs. Currently, only models trained in PyTorch, exported using its TorchScript functionality, are supported.

### Hybrid NNP/MM Simulations

The NNPot interface also supports hybrid NNP/MM simulations, where only a small subsystem is modeled with the neural network potential, and the rest of the system is modeled using regular force fields. The embedding scheme used is analoguous to the additive mechanical embedding scheme in QM/MM simulations, where the total energy of the system is expressed as $E_{tot} = E_{NNP} + E_{MM} + E_{NNP-MM}$, where $E_{NNP}$ describes the energy predicted by the model for the NNP subsystem, $E_{MM}$ describes all other interactions calculated using the classical MM force field. The coupling term $E_{NNP-MM}$ term includes non-bonded electrostatic and LJ interactions between the atoms in the NNP and MM regions, calculated as usual in GROMACS. An exception to this is when the NNP model itself calculates the electrostatic interactions, in which case the coupling term only includes the LJ interactions. Bonded interactions are also described on the MM level, removing terms consisting of bonds containing 2 NNP atoms, angles and settles containing 2 or 3 NNP atoms, and dihedrals containing 3 or 4 NNP atoms. Broken chemical bonds between NNP and MM atoms are capped with a link atom, as is usual in QM/MM simulations. The link atom is placed at a fixed distance from the NNP atom along the vector pointing towards the MM atom, and the force acting on it is redistributed to the NNP and MM atoms, as if it was a 2FD-type virtual site. The type of link atom and the distance from the NNP atom can be specified via *mdp* (page 579) options, see below. All necessary modifications to the system topology are performed automatically during *gmx grompp* (page 230) preprocessing.

### Software Prerequisites

To perform simulations with the NNPot module, GROMACS needs to be linked with a LibTorch installation (version 2.0 or higher). For specific installations instructions please see *this section* (page 28) of the install guide.

### Usage

Simulations using the NNPot interface are controlled by setting *mdp* (page 579) file options. For a detailed description of all available options, please see the *mdp options documentation* (page 100). The inputs are passed to the model in order of their occurence in the mdp file. Note that there are no default values for the model input, so not specifying the model input will lead to errors. The model is expected to return a tensor containing the energy of the system and, optionally, a tensor containing the forces on the input atoms. This option can be useful for cases in which the forces can be computed by the model by some technique that is more efficient than via backpropagation. If the model does not provide its own forces, they are calculated by the interface as gradients w.r.t. the *first* input tensor. In the case of `electrostatic-model` embedding, forces on the NNP as well as MM atoms should be returned. By default, GROMACS will run inference on the GPU, if available. You can also explicitly specify the device on which you wish to run model inference using the environment variable `GMX_NN_DEVICE`. The options `cpu` and `gpu` are supported. Option `cuda` is deprecated, and is treated as `gpu`. If multiple GPUs are available, the active device will be inherited from mdrun. For `gpu`, a CUDA/HIP-aware LibTorch installation should be installed, and the corresponding CUDA/HIP installation should be available in your `PATH`.

Find below an example Python code snippet to export a pretrained model in PyTorch using TorchScript. As an example, we use the popular ANI models available from TorchANI.

```
import torch
from torch import nn
from torchani.models import ANI2x
```

(continues on next page)

(continued from previous page)

```
from typing import Optional

class GmxNNPotModelWrapper(nn.Module):
    def __init__(self):
        super().__init__()

        # Load a pre-trained ANI-2x model
        self.model = ANI2x(periodic_table_index=True)

        # GROMACS and TorchANI use different unit conventions
        self.length_conversion = 10.0   # nm --> Å
        self.energy_conversion = 2625.5 # Hartree --> kJ/mol

    def forward(self, positions, atomic_numbers,
                box: Optional[torch.Tensor]=None, pbc:␣
↪Optional[torch.Tensor]=None):

        # Prepare the inputs for the model
        atomic_numbers = atomic_numbers.unsqueeze(0)
        positions = positions.unsqueeze(0) * self.length_conversion
        if box is not None:
            box *= self.length_conversion

        # Forward pass
        result = self.model((atomic_numbers, positions), box, pbc)

        energy = result.energies[0] * self.energy_conversion

        return energy

model = GmxNNPotModelWrapper()

save_path = 'ani2x.pt'
torch.jit.script(model).save(save_path)
```

The model can then be used in GROMACS by specifying the path to the saved model. Take care that the LibTorch version linked to GROMACS matches the one that was used to train/export the model. In the *mdp* (page 579) file, the relevant options could look like:

```
nnpot-active          = true
nnpot-modelfile       = ani2x.pt
nnpot-input-group     = System
nnpot-model-input1    = atom-positions
nnpot-model-input2    = atom-numbers
nnpot-model-input3    = box
nnpot-model-input4    = pbc
```

### 5.8.23 Fast Multipole Method (FMM)

The **Fast Multipole Method (FMM)** is a numerical algorithm designed to accelerate the computation of long-range electrostatic interactions in particle systems. It achieves this by grouping distant particles into cells and approximating their collective influence through multipole and local expansions. This reduces the computational complexity from $O(N^2)$ to approximately $O(N)$.

Within the hierarchical subdivision of the FMM, interactions are divided into long-range and short-range components. Cells are considered well separated when they do not share a boundary point, meaning that only their collective multipole contributions are used for interaction. These multipole-based computations are referred to as far-field interactions, evaluated using multipole kernels (also known as far-field kernels). Conversely, cells that do share a boundary are not well separated, and their particle-particle interactions must be computed explicitly. These explicit computations between particles in neighboring cells are known as **direct interactions**, and they are evaluated using direct kernels (also called near-field kernels) without any multipole approximation.

In practice, the number of neighboring cells included in the direct calculation depends on the specific implementation. In FMM implementations, neighborhood may include one or two layers of adjacent cells, depending on the chosen expansion order, desired accuracy, and the relative performance of direct and multipole kernels.

A detailed description of the method can be found in Ref. *196* (page 693).

#### Integration with GROMACS

GROMACS provides an extensible **FMM module interface** that allows linking external FMM libraries. The interface supports flexible configurations, enabling forward and backward compatibility between GROMACS and external FMM libraries.

Specifically:

- Short-range electrostatics can be computed either by GROMACS kernels or by **FMM-provided kernels**.
- Long-range electrostatics can be offloaded entirely to an external FMM library.

Note that, while both configurations are supported by design, the current **|Gromacs| kernels** (PlainCPU, SIMD, and CUDA) compute only Lennard-Jones (LJ-only) interactions and LJ interactions are handled solely by GROMACS, not the FMM library. Support for electrostatic Coulomb interactions in GROMACS will be added in future versions.

In the current setup, FMM shares the same MPI ranks as GROMACS, meaning there are no dedicated FMM ranks. Although both GROMACS and the FMM library may use GPU-GPU communication internally, the current interface requires particle coordinates to be transferred from the GPU to the CPU before being passed to the FMM library.

### Supported Libraries

The following FMM implementations ("backends") are currently supported in the main GROMACS distribution:

- **ExaFMM** (https://github.com/exafmm/)
- **FMSolvr** (https://www.fz-juelich.de/en/jsc/projects/fmhub)

### Usage

To use an external FMM library with GROMACS, set the mdp parameter `fmm-backend` to either `exafmm` or `fmsolvr`. The behaviour of the selected backend can be controlled by mdp options.

See *this section of the documentation* (page 101) for detailed usage of these options.

### Linking an FMM Library

External FMM source code can be placed under `src/external/` in the GROMACS source tree. To integrate it, extend the `if(GMX_USE_EXT_FMM)` block in `src/gromacs/fmm/CMakeLists.txt`, define the correct source path, and link the built library target (for example, `exafmm-lib` or `fmsolvr-lib`) to the `fmm` target.

CUDA specific sources can be conditionally added within the same block using `GMX_USE_CUDA`.

Multiple FMM backends (for example, ExaFMM or FMSolvr) can coexist under `src/external/` or in its subdirectories. Each backend provides its own implementation of `FmmForceProvider` (for example, `FmmForceProvider::ExafmmImpl` or `FmmForceProvider::FmSolvrImpl`). While multiple backends can be compiled simultaneously, only one backend is active at runtime.

An example CMake snippet is provided below:

```
if(GMX_USE_EXT_FMM)
  # Specify the external FMM source path
  set(FMM_DIR ${CMAKE_SOURCE_DIR}/src/external/<fmm_backend>)

  # Collect all source files
  file(GLOB_RECURSE FMM_SOURCES ${FMM_DIR}/src/*.cpp)
  list(APPEND FMM_INCLUDE_DIRS ${FMM_DIR}/include)

  # Optionally include GPU sources
  if(GMX_GPU_CUDA)
    file(GLOB_RECURSE FMM_CUDA_SOURCES ${FMM_DIR}/src/*.cu)
    list(APPEND FMM_SOURCES ${FMM_CUDA_SOURCES})
    list(APPEND FMM_INCLUDE_DIRS ${FMM_DIR}/include/cuda)
  endif()

  # Define the external FMM library target
  add_library(fmm-lib STATIC ${FMM_SOURCES})
  target_include_directories(fmm-lib PUBLIC ${FMM_INCLUDE_DIRS})

  # Link it with the GROMACS FMM target
```

(continues on next page)

(continued from previous page)

```
  target_link_libraries(fmm PRIVATE fmm-lib)
endif()
```

## 5.9 Run parameters and Programs

### 5.9.1 Online documentation

We install standard UNIX man pages for all the programs. If you have sourced the `GMXRC` script in the GROMACS binary directory for your host they should already be present in your `MANPATH` environment variable, and you should be able to type *e.g.* `man gmx-grompp`. You can also use the `-h` flag on the command line (e.g. *gmx grompp* (page 230) `-h`) to see the same information, as well as `gmx help grompp`. The list of all programs are available from *gmx help* (page 247).

### 5.9.2 File types

Information about different file types can be found in *File formats* (page 573).

GROMACS files written in XDR format can be read on any architecture with GROMACS version 1.6 or later if the configuration script found the XDR libraries on your system. They should always be present on UNIX since they are necessary for NFS support.

### 5.9.3 Run Parameters

The descriptions of *mdp* (page 579) parameters can be found at under the link above both in your local GROMACS installation, or *here* (page 51).

## 5.10 Analysis

In this chapter different ways of analyzing your trajectory are described. The names of the corresponding analysis programs are given. Specific information on the in- and output of these programs can be found in the tool documentation *here* (page 142). The output files are often produced as finished Grace/Xmgr graphs.

First, in sec. *Using Groups* (page 653), the group concept in analysis is explained. *Selections* (page 655) explains a newer concept of dynamic selections, which is currently supported by a few tools. Then, the different analysis tools are presented.

### 5.10.1 Using Groups

In chapter *Algorithms* (page 429), it was explained how *groups of atoms* can be used in *mdrun* (page 258) (see sec. *The group concept* (page 432)). In most analysis programs, groups of atoms must also be chosen. Most programs can generate several default index groups, but groups can always be read from an index file. Let's consider the example of a simulation of a binary mixture of components A and B. When we want to calculate the radial distribution function (RDF) $g_{AB}(r)$ of A with respect to B, we have to calculate:

$$4\pi r^2 g_{AB}(r) \;=\; V \sum_{i \in A}^{N_A} \sum_{j \in B}^{N_B} P(r) \tag{5.403}$$

where $V$ is the volume and $P(r)$ is the probability of finding a B atom at distance $r$ from an A atom.

By having the user define the *atom numbers* for groups A and B in a simple file, we can calculate this $g_{AB}$ in the most general way, without having to make any assumptions in the RDF program about the type of particles.

Groups can therefore consist of a series of *atom numbers*, but in some cases also of *molecule numbers*. It is also possible to specify a series of angles by *triples* of *atom numbers*, dihedrals by *quadruples* of *atom numbers* and bonds or vectors (in a molecule) by *pairs* of *atom numbers*. When appropriate the type of index file will be specified for the following analysis programs. To help creating such *index file* (page 580) `index.ndx`), there are a couple of programs to generate them, using either your input configuration or the topology. To generate an index file consisting of a series of *atom numbers* (as in the example of $g_{AB}$), use *gmx make_ndx* (page 255) or *gmx select* (page 312). To generate an index file with angles or dihedrals, use *gmx mk_angndx* (page 266). Of course you can also make them by hand. The general format is presented here:

```
[ Oxygen ]
   1        4        7

[ Hydrogen ]
   2        3        5        6
   8        9
```

First, the group name is written between square brackets. The following atom numbers may be spread out over as many lines as you like. The atom numbering starts at 1.

Each tool that can use groups will offer the available alternatives for the user to choose. That choice can be made with the number of the group, or its name. In fact, the first few letters of the group name will suffice if that will distinguish the group from all others. There are ways to use Unix shell features to choose group names on the command line, rather than interactively. Consult our webpage for suggestions.

### Default Groups

When no index file is supplied to analysis tools or *grompp* (page 230), a number of default groups are generated to choose from:

**System**

all atoms in the system

**Protein**

all protein atoms

**Protein-H**

protein atoms excluding hydrogens

**C-alpha**

$C_\alpha$ atoms

**Backbone**

protein backbone atoms; N, $C_\alpha$ and C

**MainChain**

protein main chain atoms: N, $C_\alpha$, C and O, including oxygens in C-terminus

**MainChain+Cb**

protein main chain atoms including $C_\beta$

**MainChain+H**

protein main chain atoms including backbone amide hydrogens and hydrogens on the N-terminus

**SideChain**

protein side chain atoms; that is all atoms except N, $C_\alpha$, C, O, backbone amide hydrogens, oxygens in C-terminus and hydrogens on the N-terminus

**SideChain-H**

protein side chain atoms excluding all hydrogens

**Prot-Masses**

protein atoms excluding dummy masses (as used in virtual site constructions of $NH_3$ groups and tryptophan side-chains), see also sec. *Virtual sites* (page 538); this group is only included when it differs from the `Protein` group

**Non-Protein**

all non-protein atoms

**DNA**

all DNA atoms

**RNA**

all RNA atoms

**Water**

water molecules (names like `SOL`, `WAT`, `HOH`, etc.) See `residuetypes.dat` for a full listing

**non-Water**

anything not covered by the `Water` group

**Ion**

any name matching an Ion entry in `residuetypes.dat`

**Water_and_Ions**

combination of the `Water` and `Ions` groups

**molecule_name**

for all residues/molecules which are not recognized as protein, DNA, or RNA; one group per residue/molecule name is generated

**Other**

all atoms which are neither protein, DNA, nor RNA.

Empty groups will not be generated. Most of the groups only contain protein atoms. An atom is considered a protein atom if its residue name is listed in the `residuetypes.dat` file and is listed as a "Protein" entry. The process for determinding DNA, RNA, etc. is analogous. If you need to modify these classifications, then you can copy the file from the library directory into your working directory and edit the local copy.

### Selections

*gmx select* (page 312)

Currently, a few analysis tools support an extended concept of *(dynamic) selections*. There are three main differences to traditional index groups:

- The selections are specified as text instead of reading fixed atom indices from a file, using a syntax similar to VMD. The text can be entered interactively, provided on the command line, or from a file.
- The selections are not restricted to atoms, but can also specify that the analysis is to be performed on, e.g., center-of-mass positions of a group of atoms. Some tools may not support selections that do not evaluate to single atoms, e.g., if they require information that is available only for single atoms, like atom names or types.
- The selections can be dynamic, i.e., evaluate to different atoms for different trajectory frames. This allows analyzing only a subset of the system that satisfies some geometric criteria.

As an example of a simple selection, `resname ABC` and `within 2 of resname DEF` selects all atoms in residues named ABC that are within 2nm of any atom in a residue named DEF.

Tools that accept selections can also use traditional index files similarly to older tools: it is possible to give an *ndx* (page 580) file to the tool, and directly select a group from the index file as a selection, either by group number or by group name. The index groups can also be used as a part of a more complicated selection.

To get started, you can run *gmx select* (page 312) with a single structure, and use the interactive prompt to try out different selections. The tool provides, among others, output options `-on` and `-ofpdb` to write out the selected atoms to an index file and to a *pdb* (page 581) file, respectively. This does not allow testing selections that evaluate to center-of-mass positions, but other selections can be tested and the result examined.

The detailed syntax and the individual keywords that can be used in selections can be accessed by typing `help` in the interactive prompt of any selection-enabled tool, as well as with *gmx help* (page 247) selections. The help is divided into subtopics that can be accessed with, e.g., `help syntax`/ *gmx help* (page 247) `selections syntax`. Some individual selection keywords have extended help as well, which can be accessed with, e.g., `help keywords` within.

The interactive prompt does not currently provide much editing capabilities. If you need them, you can run the program under `rlwrap`.

For tools that do not yet support the selection syntax, you can use *gmx select* (page 312) -on to generate static index groups to pass to the tool. However, this only allows for a small subset (only the first bullet from the above list) of the flexibility that fully selection-aware tools offer.

It is also possible to write your own analysis tools to take advantage of the flexibility of these selections: see the `template.cpp` file in the `share/gromacs/template` directory of your installation for an example and https://manual.gromacs.org/current/doxygen/html-full/page_analysistemplate.xhtml for documentation.

### 5.10.2 Looking at your trajectory

Before analyzing your trajectory it is often informative to look at your trajectory first. There are several programs that can read the GROMACS trajectory formats – have a look at our webpage for up-to-date links.

### 5.10.3 General properties

*gmx energy* (page 214), *gmx traj* (page 328)
To analyze some or all *energies* and other properties, such as *total pressure*, *pressure tensor*, *density*, *box-volume* and *box-sizes*, use the program *gmx energy* (page 214). A choice can be made from a list a set of energies, like potential, kinetic or total energy, or individual contributions, like Lennard-Jones or dihedral energies.

The *center-of-mass velocity*, defined as

$$\mathbf{v}_{com} = \frac{1}{M}\sum_{i=1}^{N} m_i \mathbf{v}_i \tag{5.404}$$

with $M = \sum_{i=1}^{N} m_i$ the total mass of the system, can be monitored in time by the program *gmx traj* (page 328) `-com -ov`. It is however recommended to remove the center-of-mass velocity every step (see chapter *Algorithms* (page 429))!

### 5.10.4 Radial distribution functions

*gmx rdf* (page 291)
The *radial distribution function* (RDF) or pair correlation function $g_{AB}(r)$ between particles of type $A$ and $B$ is defined in the following way:

$$\begin{aligned} g_{AB}(r) &= \frac{\langle \rho_B(r) \rangle}{\langle \rho_B \rangle_{local}} \\ &= \frac{1}{\langle \rho_B \rangle_{local}} \frac{1}{N_A} \sum_{i \in A}^{N_A} \sum_{j \in B}^{N_B} \frac{\delta(r_{ij} - r)}{4\pi r^2} \end{aligned} \tag{5.405}$$

with $\langle \rho_B(r) \rangle$ the particle density of type $B$ at a distance $r$ around particles $A$, and $\langle \rho_B \rangle_{local}$ the particle density of type $B$ averaged over all spheres around particles $A$ with radius $r_{max}$ (see Fig. 5.52 C).

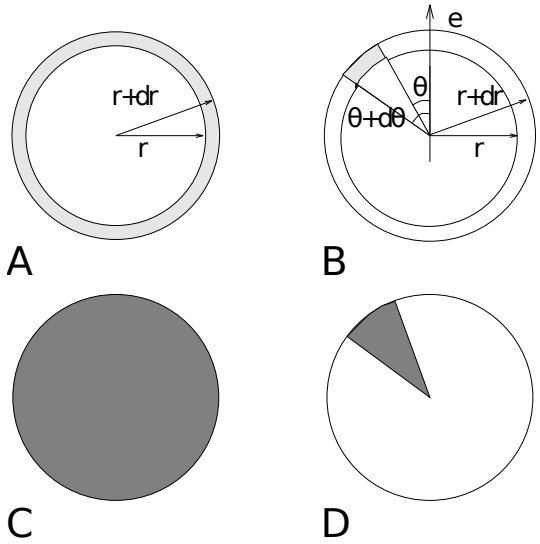



Fig. 5.52: Definition of slices in *gmx rdf* (page 291): A. $g_{AB}(r)$. B. $g_{AB}(r, \theta)$. The slices are colored gray. C. Normalization $\langle \rho_B \rangle_{local}$. D. Normalization $\langle \rho_B \rangle_{local,\, \theta}$. Normalization volumes are colored gray.

Usually the value of $r_{max}$ is half of the box length. The averaging is also performed in time. In practice the analysis program *gmx rdf* (page 291) divides the system into spherical slices (from $r$ to $r + dr$, see Fig. 5.52 A) and makes a histogram in stead of the $\delta$-function. An example of the RDF of oxygen-oxygen in SPC water *80* (page 687) is given in Fig. 5.53

With *gmx rdf* (page 291) it is also possible to calculate an angle dependent rdf $g_{AB}(r, \theta)$, where the angle $\theta$ is defined with respect to a certain laboratory axis $\mathbf{e}$, see Fig. 5.52 B.

$$g_{AB}(r, \theta) = \frac{1}{\langle \rho_B \rangle_{local,\, \theta}} \frac{1}{N_A} \sum_{i \in A}^{N_A} \sum_{j \in B}^{N_B} \frac{\delta(r_{ij} - r)\delta(\theta_{ij} - \theta)}{2\pi r^2 sin(\theta)} \tag{5.406}$$

$$cos(\theta_{ij}) = \frac{\mathbf{r}_{ij} \cdot \mathbf{e}}{\|r_{ij}\| \, \|e\|} \tag{5.407}$$

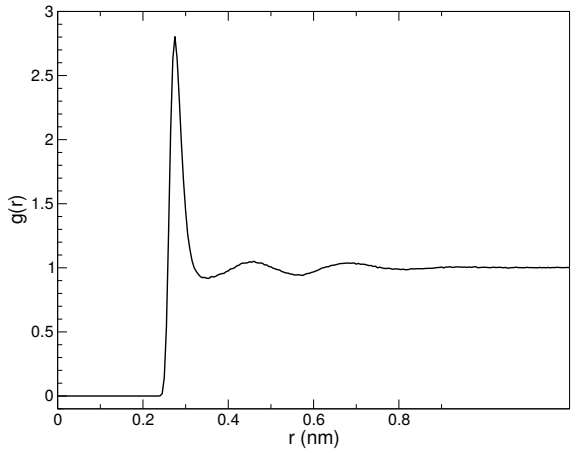



Fig. 5.53: $g_{OO}(r)$ for Oxygen-Oxygen of SPC-water.

This $g_{AB}(r,\theta)$ is useful for analyzing anisotropic systems. **Note** that in this case the normalization $\langle\rho_B\rangle_{local,\,\theta}$ is the average density in all angle slices from $\theta$ to $\theta + d\theta$ up to $r_{max}$, so angle dependent, see Fig. 5.52 D.

### 5.10.5 Correlation functions

#### Theory of correlation functions

The theory of correlation functions is well established *108* (page 688). We describe here the implementation of the various correlation function flavors in the GROMACS code. The definition of the autocorrelation function (ACF) $C_f(t)$ for a property $f(t)$ is:

$$C_f(t) \;=\; \langle f(\xi) f(\xi + t)\rangle_\xi \tag{5.408}$$

where the notation on the right hand side indicates averaging over $\xi$, *i.e.* over time origins. It is also possible to compute cross-correlation function from two properties $f(t)$ and $g(t)$:

$$C_{fg}(t) \;=\; \langle f(\xi) g(\xi + t)\rangle_\xi \tag{5.409}$$

however, in GROMACS there is no standard mechanism to do this (**note:** you can use the `xmgr` program to compute cross correlations). The integral of the correlation function over time is the correlation time $\tau_f$:

$$\tau_f \;=\; \int_0^\infty C_f(t)\mathrm{d}t \tag{5.410}$$

In practice, correlation functions are calculated based on data points with discrete time intervals $\Delta$t, so that the ACF from an MD simulation is:

$$C_f(j\Delta t) \;=\; \frac{1}{N-j}\sum_{i=0}^{N-1-j} f(i\Delta t) f((i+j)\Delta t) \tag{5.411}$$

where $N$ is the number of available time frames for the calculation. The resulting ACF is obviously only available at time points with the same interval $\Delta$t. Since, for many applications, it is necessary to know the short time behavior of the ACF (*e.g.* the first 10 ps) this often means that we have to save the data with intervals much shorter than the time scale of interest. Another implication of (5.411) is that in

principle we can not compute all points of the ACF with the same accuracy, since we have $N-1$ data points for $C_f(\Delta t)$ but only 1 for $C_f((N-1)\Delta t)$. However, if we decide to compute only an ACF of length $M\Delta t$, where $M \leq N/2$ we can compute all points with the same statistical accuracy:

$$C_f(j\Delta t) \;=\; \frac{1}{M}\sum_{i=0}^{N-1-M} f(i\Delta t)f((i+j)\Delta t) \tag{5.412}$$

Here of course $j < M$. $M$ is sometimes referred to as the time lag of the correlation function. When we decide to do this, we intentionally do not use all the available points for very short time intervals ($j << M$), but it makes it easier to interpret the results. Another aspect that may not be neglected when computing ACFs from simulation is that usually the time origins $\xi$ ((5.408)) are not statistically independent, which may introduce a bias in the results. This can be tested using a block-averaging procedure, where only time origins with a spacing at least the length of the time lag are included, *e.g.* using $k$ time origins with spacing of $M\Delta t$ (where $kM \leq N$):

$$C_f(j\Delta t) \;=\; \frac{1}{k}\sum_{i=0}^{k-1} f(iM\Delta t)f((iM+j)\Delta t) \tag{5.413}$$

However, one needs very long simulations to get good accuracy this way, because there are many fewer points that contribute to the ACF.

### Using FFT for computation of the ACF

The computational cost for calculating an ACF according to (5.411) is proportional to $N^2$, which is considerable. However, this can be improved by using fast Fourier transforms to do the convolution *108* (page 688).

### Special forms of the ACF

There are some important varieties on the ACF, *e.g.* the ACF of a vector $\mathbf{p}$:

$$C_{\mathbf{p}}(t) \;=\; \int_0^\infty P_n(\cos\angle\left(\mathbf{p}(\xi),\mathbf{p}(\xi+t)\right)\mathrm{d}\xi \tag{5.414}$$

where $P_n(x)$ is the $n^{th}$ order Legendre polynomial.[1] Such correlation times can actually be obtained experimentally using *e.g.* NMR or other relaxation experiments. GROMACS can compute correlations using the $1^{st}$ and $2^{nd}$ order Legendre polynomial ((5.414)). This can also be used for rotational autocorrelation (*gmx rotacf* (page 301)) and dipole autocorrelation (*gmx dipoles* (page 193)).

In order to study torsion angle dynamics, we define a dihedral autocorrelation function as *159* (page 691):

$$C(t) \;=\; \left\langle\cos(\theta(\tau)-\theta(\tau+t))\right\rangle_\tau \tag{5.415}$$

**Note** that this is not a product of two functions as is generally used for correlation functions, but it may be rewritten as the sum of two products:

$$C(t) \;=\; \left\langle\cos(\theta(\tau))\cos(\theta(\tau+t)) \;+\; \sin(\theta(\tau))\sin(\theta(\tau+t))\right\rangle_\tau \tag{5.416}$$

[1] $P_0(x)=1, P_1(x)=x, P_2(x)=(3x^2-1)/2$

### Some Applications

The program *gmx velacc* (page 348) calculates the *velocity autocorrelation function*.

$$C_{\mathbf{v}}(\tau) \;=\; \langle \mathbf{v}_i(\tau) \cdot \mathbf{v}_i(0) \rangle_{i \in A} \tag{5.417}$$

The self diffusion coefficient can be calculated using the Green-Kubo relation *108* (page 688):

$$D_A \;=\; \frac{1}{3} \int_0^\infty \langle \mathbf{v}_i(t) \cdot \mathbf{v}_i(0) \rangle_{i \in A} \, dt \tag{5.418}$$

which is just the integral of the velocity autocorrelation function. There is a widely-held belief that the velocity ACF converges faster than the mean square displacement (sec. *Mean Square Displacement* (page 662)), which can also be used for the computation of diffusion constants. However, Allen & Tildesley *108* (page 688) warn us that the long-time contribution to the velocity ACF can not be ignored, so care must be taken.

Another important quantity is the dipole correlation time. The *dipole correlation function* for particles of type $A$ is calculated as follows by *gmx dipoles* (page 193):

$$C_{\mu}(\tau) \;=\; \langle \mu_i(\tau) \cdot \mu_i(0) \rangle_{i \in A} \tag{5.419}$$

with $\mu_i = \sum_{j \in i} \mathbf{r}_j q_j$. The dipole correlation time can be computed using (5.410). For some applications see (**???**).

The viscosity of a liquid can be related to the correlation time of the Pressure tensor $\mathbf{P}$ *160* (page 691), *161* (page 691). *gmx energy* (page 214) can compute the viscosity, but this is not very accurate *149* (page 690), and actually the values do not converge.

## 5.10.6 Curve fitting in GROMACS

### Sum of exponential functions

Sometimes it is useful to fit a curve to an analytical function, for example in the case of autocorrelation functions with noisy tails. GROMACS is not a general purpose curve-fitting tool however and therefore GROMACS only supports a limited number of functions. Table 5.18 lists the available options with the corresponding command-line options. The underlying routines for fitting use the Levenberg-Marquardt algorithm as implemented in the lmfit package *162* (page 691) (a bare-bones version of which is included in GROMACS in which an option for error-weighted fitting was implemented).

Table 5.18: Overview of fitting functions supported in (most) analysis tools that compute autocorrelation functions. The **Note** column describes properties of the output parameters.

| Command line option | Functional form $f(t)$ | Note |
|---|---|---|
| exp | $e^{-t/a_0}$ | |
| aexp | $a_1 e^{-t/a_0}$ | |
| exp_exp | $a_1 e^{-t/a_0} + (1 - a_1) e^{-t/a_2}$ | $a_2 \geq a_0 \geq 0$ |
| exp5 | $a_1 e^{-t/a_0} + a_3 e^{-t/a_2} + a_4$ | $a_2 \geq a_0 \geq 0$ |
| exp7 | $a_1 e^{-t/a_0} + a_3 e^{-t/a_2} + a_5 e^{-t/a_4} + a_6$ | $a_4 \geq a_2 \geq a_0 \geq 0$ |
| exp9 | $a_1 e^{-t/a_0} + a_3 e^{-t/a_2} + a_5 e^{-t/a_4} + a_7 e^{-t/a_6} + a_8$ | $a_6 \geq a_4 \geq a_2 \geq a_0 \geq 0$ |

### Error estimation

Under the hood GROMACS implements some more fitting functions, namely a function to estimate the error in time-correlated data due to Hess *149* (page 690):

$$\varepsilon^2(t) = 2\alpha\tau_1\left(1+\frac{\tau_1}{t}\left(e^{-t/\tau_1}-1\right)\right)+2(1-\alpha)\tau_2\left(1+\frac{\tau_2}{t}\left(e^{-t/\tau_2}-1\right)\right) \tag{5.420}$$

where $\tau_1$ and $\tau_2$ are time constants (with $\tau_2 \geq \tau_1$) and $\alpha$ usually is close to 1 (in the fitting procedure it is enforced that $0 \leq \alpha \leq 1$). This is used in *gmx analyze* (page 154) for error estimation using

$$\lim_{t\to\infty}\varepsilon(t) = \sigma\sqrt{\frac{2(\alpha\tau_1+(1-\alpha)\tau_2)}{T}} \tag{5.421}$$

where $\sigma$ is the standard deviation of the data set and $T$ is the total simulation time *149* (page 690).

### Interphase boundary demarcation

In order to determine the position and width of an interface, Steen-Sæthre *et al.* fitted a density profile to the following function

$$f(x) \;=\; \frac{a_0+a_1}{2}-\frac{a_0-a_1}{2}\mathrm{erf}\left(\frac{x-a_2}{a_3^2}\right) \tag{5.422}$$

where $a_0$ and $a_1$ are densities of different phases, $x$ is the coordinate normal to the interface, $a_2$ is the position of the interface and $a_3$ is the width of the interface *163* (page 691). This is implemented in *gmx densorder* (page 189).

### Transverse current autocorrelation function

In order to establish the transverse current autocorrelation function (useful for computing viscosity *164* (page 691)) the following function is fitted:

$$f(x) \;=\; e^{-\nu}\left(\cosh(\omega\nu)+\frac{\sinh(\omega\nu)}{\omega}\right) \tag{5.423}$$

with $\nu = x/(2a_0)$ and $\omega = \sqrt{1-a_1}$. This is implemented in *gmx tcaf* (page 326).

### Viscosity estimation from pressure autocorrelation function

The viscosity is a notoriously difficult property to extract from simulations *149* (page 690), *165* (page 691). It is *in principle* possible to determine it by integrating the pressure autocorrelation function *160* (page 691), however this is often hampered by the noisy tail of the ACF. A workaround to this is fitting the ACF to the following function *166* (page 691):

$$f(t)/f(0) = (1-C)\cos(\omega t)e^{-(t/\tau_f)^{\beta_f}}+Ce^{-(t/\tau_s)^{\beta_s}} \tag{5.424}$$

where $\omega$ is the frequency of rapid pressure oscillations (mainly due to bonded forces in molecular simulations), $\tau_f$ and $\beta_f$ are the time constant and exponent of fast relaxation in a stretched-exponential approximation, $\tau_s$ and $\beta_s$ are constants for slow relaxation and $C$ is the pre-factor that determines the weight between fast and slow relaxation. After a fit, the integral of the function $f(t)$ is used to compute the viscosity:

$$\eta = \frac{V}{k_BT}\int_0^\infty f(t)dt \tag{5.425}$$

This equation has been applied to computing the bulk and shear viscosity using different elements from the pressure tensor *167* (page 691).

### 5.10.7 Mean Square Displacement

*gmx msd* (page 266)

To determine the self diffusion coefficient $D_A$ of particles of type $A$, one can use the Einstein relation *108* (page 688):

$$\lim_{t\to\infty} \langle \|\mathbf{r}_i(t) - \mathbf{r}_i(0)\|^2 \rangle_{i\in A} \;=\; 6D_A t \tag{5.426}$$

This *mean square displacement* and $D_A$ are calculated by the program *gmx msd* (page 266). Normally an index file containing atom numbers is used and the MSD is averaged over these atoms. For molecules consisting of more than one atom, $\mathbf{r}_i$ can be taken as the center of mass positions of the molecules. In that case, you should use an index file with molecule numbers. The results will be nearly identical to averaging over atoms, however. The *gmx msd* (page 266) program can also be used for calculating diffusion in one or two dimensions. This is useful for studying lateral diffusion on interfaces.

An example of the mean square displacement of SPC water is given in Fig. 5.54.

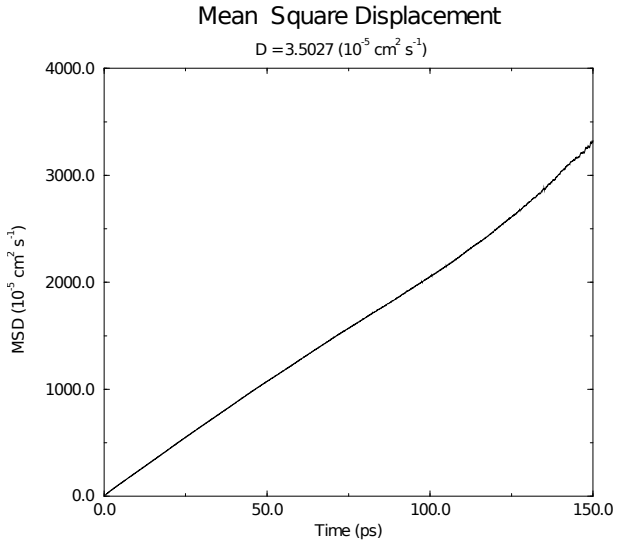



Fig. 5.54: Mean Square Displacement of SPC-water.

### 5.10.8 Bonds/distances, angles and dihedrals

*gmx distance* (page 198), *gmx angle* (page 158), *gmx gangle* (page 223)

To monitor specific *bonds* in your modules, or more generally distances between points, the program *gmx distance* (page 198) can calculate distances as a function of time, as well as the distribution of the distance. With a traditional index file, the groups should consist of pairs of atom numbers, for example:

```
[ bonds_1 ]
 1      2
 3      4
 9     10

[ bonds_2 ]
12     13
```

Selections are also supported, with first two positions defining the first distance, second pair of positions defining the second distance and so on. You can calculate the distances between CA and CB atoms in all your residues (assuming that every residue either has both atoms, or neither) using a selection such as:

```
name CA CB
```

The selections also allow more generic distances to be computed. For example, to compute the distances between centers of mass of two residues, you can use:

```
com of resname AAA plus com of resname BBB
```

The program *gmx angle* (page 158) calculates the distribution of *angles* and *dihedrals* in time. It also gives the average angle or dihedral. The index file consists of triplets or quadruples of atom numbers:

```
[ angles ]
 1      2      3
 2      3      4
 3      4      5

[ dihedrals ]
 1      2      3      4
 2      3      5      5
```

For the dihedral angles you can use either the "biochemical convention" ($\phi = 0 \equiv cis$) or "polymer convention" ($\phi = 0 \equiv trans$), see Fig. 5.55.

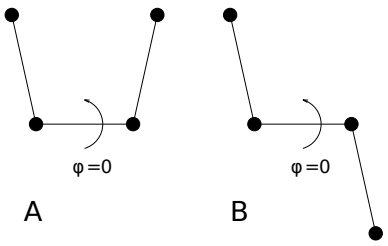



Fig. 5.55: Dihedral conventions: A. "Biochemical convention". B. "Polymer convention".

The program *gmx gangle* (page 223) provides a selection-enabled version to compute angles. This tool can also compute angles and dihedrals, but does not support all the options of *gmx angle* (page 158), such as autocorrelation or other time series analyses. In addition, it supports angles between two vectors, a vector and a plane, two planes (defined by 2 or 3 points, respectively), a vector/plane and the $z$ axis, or a vector/plane and the normal of a sphere (determined by a single position). Also the angle between a vector/plane compared to its position in the first frame is supported. For planes, *gmx gangle* (page 223) uses the normal vector perpendicular to the plane. See Fig. 5.56 A, B, C) for the definitions.

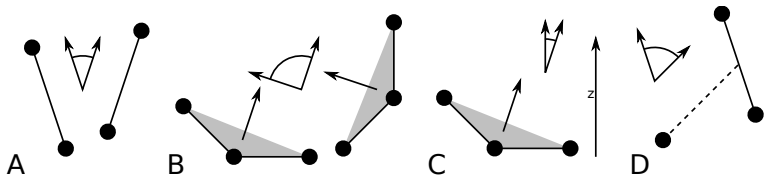



Fig. 5.56: Angle options of *gmx gangle* (page 223): A. Angle between two vectors. B. Angle between two planes. C. Angle between a vector and the $z$ axis. D. Angle between a vector and the normal of a sphere. Also other combinations are supported: planes and vectors can be used interchangeably.

### 5.10.9 Radius of gyration and distances

*gmx gyrate* (page 233), *gmx distance* (page 198), *gmx mindist* (page 264), *gmx mdmat* (page 256), *gmx pairdist* (page 279), *gmx xpm2ps* (page 357)

To have a rough measure for the compactness of a structure, you can calculate the *radius of gyration* with the program *gmx gyrate* (page 233) as follows:

$$R_g = \left( \frac{\sum_i \|\mathbf{r}_i\|^2 m_i}{\sum_i m_i} \right)^{\frac{1}{2}} \tag{5.427}$$

where $m_i$ is the mass of atom $i$ and $\mathbf{r}_i$ the position of atom $i$ with respect to the center of mass of the molecule. It is especially useful to characterize polymer solutions and proteins. The program will also provide the radius of gyration around the coordinate axis (or, optionally, principal axes) by only summing the radii components orthogonal to each axis, for instance

$$R_{g,x} = \left( \frac{\sum_i \left( r_{i,y}^2 + r_{i,z}^2 \right) m_i}{\sum_i m_i} \right)^{\frac{1}{2}} \tag{5.428}$$

Sometimes it is interesting to plot the *distance* between two atoms, or the *minimum* distance between two groups of atoms (*e.g.*: protein side-chains in a salt bridge). To calculate these distances between certain groups there are several possibilities:

- The *distance between the geometrical centers* of two groups can be calculated with the program *gmx distance* (page 198), as explained in sec. *Bonds/distances, angles and dihedrals* (page 662).
- The *minimum distance* between two groups of atoms during time can be calculated with the program *gmx mindist* (page 264). It also calculates the *number of contacts* between these groups within a certain radius $r_{max}$.
- *gmx pairdist* (page 279) is a selection-enabled version of *gmx mindist* (page 264).
- To monitor the *minimum distances between amino acid residues* within a (protein) molecule, you can use the program *gmx mdmat* (page 256). This minimum distance between two residues $\mathrm{A}_i$ and $\mathrm{A}_j$ is defined as the smallest distance between any pair of atoms ($\mathrm{i} \in \mathrm{A}_i$, $\mathrm{j} \in \mathrm{A}_j$). The output is a symmetrical matrix of smallest distances between all residues. To visualize this matrix, you can use a program such as `xv`. If you want to view the axes and legend or if you want to print the matrix, you can convert it with *xpm2ps* (page 357) into a Postscript Fig. 5.57.

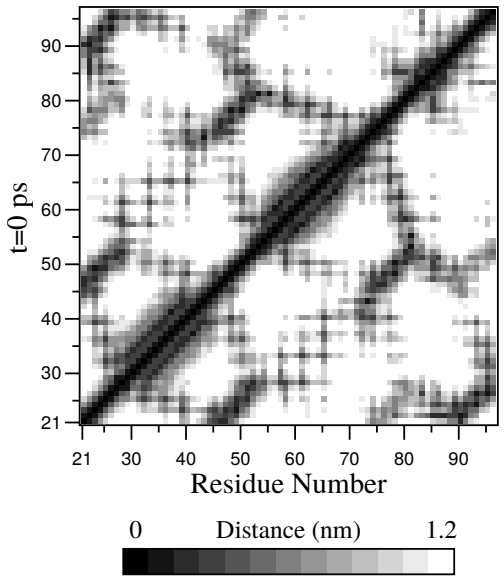



Fig. 5.57: A minimum distance matrix for a peptide *168* (page 691).

- Plotting these matrices for different time-frames, one can analyze changes in the structure, and *e.g.* forming of salt bridges.

### 5.10.10 Root mean square deviations in structure

*gmx rms* (page 294), *gmx rmsdist* (page 297)

The *root mean square deviation* ($RMSD$) of certain atoms in a molecule with respect to a reference structure can be calculated with the program *gmx rms* (page 294) by least-square fitting the structure to the reference structure ($t_2 = 0$) and subsequently calculating the $RMSD$ ((5.429)).

$$RMSD(t_1, t_2) = \left[ \frac{1}{M} \sum_{i=1}^{N} m_i \|\mathbf{r}_i(t_1) - \mathbf{r}_i(t_2)\|^2 \right]^{\frac{1}{2}} \tag{5.429}$$

where $M = \sum_{i=1}^{N} m_i$ and $\mathbf{r}_i(t)$ is the position of atom $i$ at time $t$. **Note** that fitting does not have to use the same atoms as the calculation of the $RMSD$; *e.g.* a protein is usually fitted on the backbone atoms (N, $C_\alpha$, C), but the $RMSD$ can be computed of the backbone or of the whole protein.

Instead of comparing the structures to the initial structure at time $t = 0$ (so for example a crystal structure), one can also calculate (5.429) with a structure at time $t_2 = t_1 - \tau$. This gives some insight in the mobility as a function of $\tau$. A matrix can also be made with the $RMSD$ as a function of $t_1$ and $t_2$, which gives a nice graphical interpretation of a trajectory. If there are transitions in a trajectory, they will clearly show up in such a matrix.

Alternatively the $RMSD$ can be computed using a fit-free method with the program *gmx rmsdist*

(page 297):

$$RMSD(t) \;=\; \left[\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\|\mathbf{r}_{ij}(t)-\mathbf{r}_{ij}(0)\|^2\right]^{\frac{1}{2}} \tag{5.430}$$

where the *distance* $\mathbf{r}_{ij}$ between atoms at time $t$ is compared with the distance between the same atoms at time 0.

### 5.10.11 Covariance analysis

Covariance analysis, also called principal component analysis or essential dynamics *169* (page 691), can find correlated motions. It uses the covariance matrix $C$ of the atomic coordinates:

$$C_{ij} = \left\langle M_{ii}^{\frac{1}{2}}(x_i - \langle x_i\rangle)M_{jj}^{\frac{1}{2}}(x_j - \langle x_j\rangle)\right\rangle \tag{5.431}$$

where $M$ is a diagonal matrix containing the masses of the atoms (mass-weighted analysis) or the unit matrix (non-mass weighted analysis). $C$ is a symmetric $3N \times 3N$ matrix, which can be diagonalized with an orthonormal transformation matrix $R$:

$$R^T C R = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{3N}) \quad \text{where} \;\; \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{3N} \tag{5.432}$$

The columns of $R$ are the eigenvectors, also called principal or essential modes. $R$ defines a transformation to a new coordinate system. The trajectory can be projected on the principal modes to give the principal components $p_i(t)$:

$$\mathbf{p}(t) = R^T M^{\frac{1}{2}}(\mathbf{x}(t) - \langle\mathbf{x}\rangle) \tag{5.433}$$

The eigenvalue $\lambda_i$ is the mean square fluctuation of principal component $i$. The first few principal modes often describe collective, global motions in the system. The trajectory can be filtered along one (or more) principal modes. For one principal mode $i$ this goes as follows:

$$\mathbf{x}^f(t) = \langle\mathbf{x}\rangle + M^{-\frac{1}{2}} R_{*i}\, p_i(t) \tag{5.434}$$

When the analysis is performed on a macromolecule, one often wants to remove the overall rotation and translation to look at the internal motion only. This can be achieved by least square fitting to a reference structure. Care has to be taken that the reference structure is representative for the ensemble, since the choice of reference structure influences the covariance matrix.

One should always check if the principal modes are well defined. If the first principal component resembles a half cosine and the second resembles a full cosine, you might be filtering noise (see below). A good way to check the relevance of the first few principal modes is to calculate the overlap of the sampling between the first and second half of the simulation. **Note** that this can only be done when the same reference structure is used for the two halves.

A good measure for the overlap has been defined in *170* (page 691). The elements of the covariance matrix are proportional to the square of the displacement, so we need to take the square root of the matrix to examine the extent of sampling. The square root can be calculated from the eigenvalues $\lambda_i$ and the eigenvectors, which are the columns of the rotation matrix $R$. For a symmetric and diagonally-dominant matrix $A$ of size $3N \times 3N$ the square root can be calculated as:

$$A^{\frac{1}{2}} = R\,\text{diag}(\lambda_1^{\frac{1}{2}}, \lambda_2^{\frac{1}{2}}, \ldots, \lambda_{3N}^{\frac{1}{2}})\,R^T \tag{5.435}$$

It can be verified easily that the product of this matrix with itself gives $A$. Now we can define a difference $d$ between covariance matrices $A$ and $B$ as follows:

$$\begin{aligned} d(A,B) &= \sqrt{\mathrm{tr}\left(\left(A^{\frac{1}{2}} - B^{\frac{1}{2}}\right)^2\right)} \\ &= \sqrt{\mathrm{tr}\left(A + B - 2A^{\frac{1}{2}}B^{\frac{1}{2}}\right)} \\ &= \left(\sum_{i=1}^{N}\left(\lambda_i^A + \lambda_i^B\right) - 2\sum_{i=1}^{N}\sum_{j=1}^{N}\sqrt{\lambda_i^A\lambda_j^B}\left(R_i^A \cdot R_j^B\right)^2\right)^{\frac{1}{2}} \end{aligned} \tag{5.436}$$

where tr is the trace of a matrix. We can now define the overlap $s$ as:

$$s(A,B) = 1 - \frac{d(A,B)}{\sqrt{\mathrm{tr}A + \mathrm{tr}B}} \tag{5.437}$$

The overlap is 1 if and only if matrices $A$ and $B$ are identical. It is 0 when the sampled subspaces are completely orthogonal.

A commonly-used measure is the subspace overlap of the first few eigenvectors of covariance matrices. The overlap of the subspace spanned by $m$ orthonormal vectors $\mathbf{w}_1, \ldots, \mathbf{w}_m$ with a reference subspace spanned by $n$ orthonormal vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ can be quantified as follows:

$$\mathrm{overlap}(\mathbf{v},\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}(\mathbf{v}_i \cdot \mathbf{w}_j)^2 \tag{5.438}$$

The overlap will increase with increasing $m$ and will be 1 when set $\mathbf{v}$ is a subspace of set $\mathbf{w}$. The disadvantage of this method is that it does not take the eigenvalues into account. All eigenvectors are weighted equally, and when degenerate subspaces are present (equal eigenvalues), the calculated overlap will be too low.

Another useful check is the cosine content. It has been proven that the the principal components of random diffusion are cosines with the number of periods equal to half the principal component index *170* (page 691), *171* (page 691). The eigenvalues are proportional to the index to the power $-2$. The cosine content is defined as:

$$\frac{2}{T}\left(\int_0^T \cos\left(\frac{i\pi t}{T}\right) p_i(t)\mathrm{d}t\right)^2 \left(\int_0^T p_i^2(t)\mathrm{d}t\right)^{-1} \tag{5.439}$$

When the cosine content of the first few principal components is close to 1, the largest fluctuations are not connected with the potential, but with random diffusion.

The covariance matrix is built and diagonalized by *gmx covar* (page 181). The principal components and overlap (and many more things) can be plotted and analyzed with *gmx anaeig* (page 151). The cosine content can be calculated with *gmx analyze* (page 154).

### 5.10.12 Dihedral principal component analysis

*gmx angle* (page 158), *gmx covar* (page 181), *gmx anaeig* (page 151)

Principal component analysis can be performed in dihedral space *172* (page 692) using GROMACS. You start by defining the dihedral angles of interest in an index file, either using *gmx mk_angndx* (page 266) or otherwise. Then you use the *gmx angle* (page 158) program with the `-or` flag to produce a new *trr* (page 586) file containing the cosine and sine of each dihedral angle in two coordinates,

respectively. That is, in the *trr* (page 586) file you will have a series of numbers corresponding to: $\cos(\phi_1)$, $\sin(\phi_1)$, $\cos(\phi_2)$, $\sin(\phi_2)$, $\ldots$, $\cos(\phi_n)$, $\sin(\phi_n)$, and the array is padded with zeros, if necessary. Then you can use this *trr* (page 586) file as input for the *gmx covar* (page 181) program and perform principal component analysis as usual. For this to work you will need to generate a reference file (*tpr* (page 586), *gro* (page 577), *pdb* (page 581) etc.) containing the same number of "atoms" as the new *trr* (page 586) file, that is for $n$ dihedrals you need $2n/3$ atoms (rounded up if not an integer number). You should use the `-nofit` option for *gmx covar* (page 181) since the coordinates in the dummy reference file do not correspond in any way to the information in the *trr* (page 586) file. Analysis of the results is done using *gmx anaeig* (page 151).

### 5.10.13 Hydrogen bonds

*gmx hbond* (page 237)

The program *gmx hbond* (page 237) analyzes the *hydrogen bonds* (H-bonds) between all possible donors D and acceptors A. To determine if an H-bond exists, a geometrical criterion is used, see also Fig. 5.58:

$$\begin{array}{rcrcl} r & \leq & r_{HB} & = & 0.35 \text{ nm} \\ \alpha & \leq & \alpha_{HB} & = & 30^o \end{array} \tag{5.440}$$

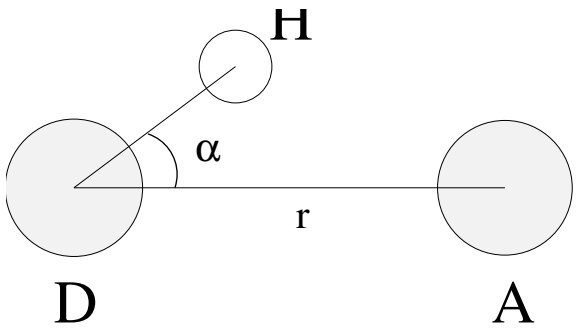



Fig. 5.58: Geometrical Hydrogen bond criterion.

The value of $r_{HB} = 0.35\text{nm}$ corresponds to the first minimum of the RDF of SPC water (see also Fig. 5.59).

The program *gmx hbond* (page 237) analyzes all hydrogen bonds existing between two groups of atoms (which must be either identical or non-overlapping) or in specified donor-hydrogen-acceptor triplets, in the following ways:

- Donor-Acceptor distance ($r$) distribution of all H-bonds
- Hydrogen-Donor-Acceptor angle ($\alpha$) distribution of all H-bonds
- The total number of H-bonds in each time frame
- The number of H-bonds in time between residues, divided into groups $n$-$n$+$i$ where $n$ and $n$+$i$ stand for residue numbers and $i$ goes from 0 to 6. The group for $i = 6$ also includes all H-bonds for $i > 6$. These groups include the $n$-$n$+3, $n$-$n$+4 and $n$-$n$+5 H-bonds, which provide a measure for the formation of $\alpha$-helices or $\beta$-turns or strands.

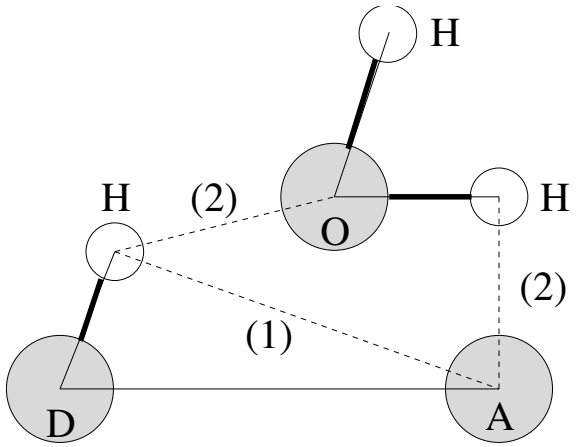



Fig. 5.59: Insertion of water into an H-bond. (1) Normal H-bond between two residues. (2) H-bonding bridge via a water molecule.

- The lifetime of the H-bonds is calculated from the average over all autocorrelation functions of the existence functions (either 0 or 1) of all H-bonds:

$$C(\tau) \;=\; \langle s_i(t)\; s_i(t+\tau)\rangle \tag{5.441}$$

- with $s_i(t) = \{0,1\}$ for H-bond $i$ at time $t$. The integral of $C(\tau)$ gives a rough estimate of the average H-bond lifetime $\tau_{HB}$:

$$\tau_{HB} \;=\; \int_0^\infty C(\tau)d\tau \tag{5.442}$$

- Both the integral and the complete autocorrelation function $C(\tau)$ will be output, so that more sophisticated analysis (*e.g.* using multi-exponential fits) can be used to get better estimates for $\tau_{HB}$. A more complete analysis is given in ref. *173* (page 692); one of the more fancy option is the Luzar and Chandler analysis of hydrogen bond kinetics *174* (page 692), *175* (page 692).
- An H-bond existence map can be generated of dimensions *# H-bonds*×*# frames*. The ordering is identical to the index file (see below), but reversed, meaning that the last triplet in the index file corresponds to the first row of the existence map.
- Index groups are output containing the analyzed groups, all donor-hydrogen atom pairs and acceptor atoms in these groups, donor-hydrogen-acceptor triplets involved in hydrogen bonds between the analyzed groups and all solvent atoms involved in insertion.

### 5.10.14 Protein-related items

*gmx dssp* (page 202), *gmx rama* (page 290), *gmx wheel* (page 354)
To analyze structural changes of a protein, you can calculate the radius of gyration or the minimum residue distances over time (see sec. *Radius of gyration and distances* (page 664)), or calculate the RMSD (sec. *Root mean square deviations in structure* (page 665)).

To analyze the secondary structure of a protein (not only for static structures, but also for trajectories), you can use the program *gmx dssp* (page 202), which is a native implementation of DSSP algorithm *176* (page 692), but also is based on the (DSSP V.4 algorithm) with some additional features. For example,

you can take into account native hydrogens from the structure (`-hmode gromacs`, set by default), while in the original algorithm, hydrogen atoms are set as pseudo-atoms with coordinates based on the coordinates of the MainChain atoms (`-hmode dssp`). Also, it is possible to conduct a fast search for neighboring residues using Neighbor Search (`-nb`, default), instead of the slow enumeration of protein residues among themselves according to the "each with each" principle implemented in the original algorithm (`-nonb`).

One other important analysis of proteins is the so-called *Ramachandran plot*. This is the projection of the structure on the two dihedral angles $\phi$ and $\psi$ of the protein backbone, see Fig. 5.60:

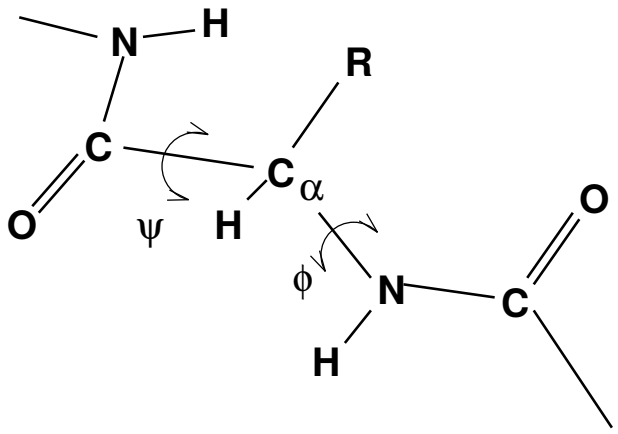



Fig. 5.60: Definition of the dihedral angles $\phi$ and $\psi$ of the protein backbone.

To evaluate this Ramachandran plot you can use the program *gmx rama* (page 290). A typical output is given in Fig. 5.61.

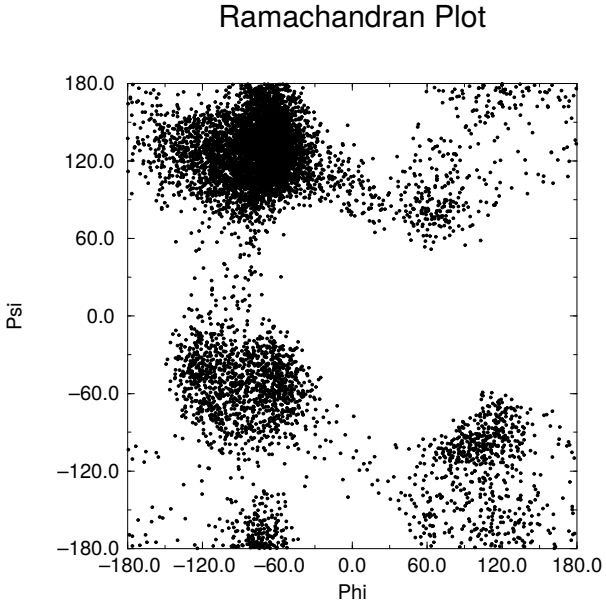



Fig. 5.61: Ramachandran plot of a small protein.

When studying $\alpha$-helices it is useful to have a *helical wheel* projection of your peptide, to see whether a peptide is amphipathic. This can be done using the *gmx wheel* (page 354) program. Two examples are plotted in Fig. 5.62.

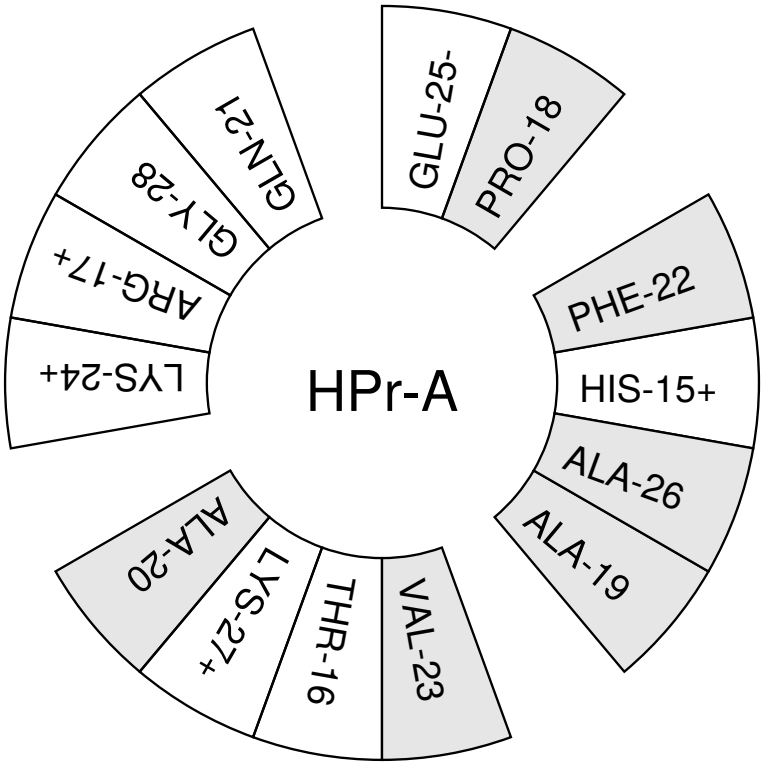



Fig. 5.62: Helical wheel projection of the N-terminal helix of HPr.

### 5.10.15 Interface-related items

*gmx order* (page 277), *gmx density* (page 185), *gmx potential* (page 287), *gmx traj* (page 328)
When simulating molecules with long carbon tails, it can be interesting to calculate their average orientation. There are several flavors of order parameters, most of which are related. The program *gmx order* (page 277) can calculate order parameters using the equation:

$$S_z = \frac{3}{2}\langle \cos^2 \theta_z \rangle - \frac{1}{2} \tag{5.443}$$

where $\theta_z$ is the angle between the $z$-axis of the simulation box and the molecular axis under consideration. The latter is defined as the vector from $C_{n-1}$ to $C_{n+1}$. The parameters $S_x$ and $S_y$ are defined in the same way. The brackets imply averaging over time and molecules. Order parameters can vary between 1 (full order along the interface normal) and $-1/2$ (full order perpendicular to the normal), with a value of zero in the case of isotropic orientation.

The program can do two things for you. It can calculate the order parameter for each $CH_2$ segment separately, for any of three axes, or it can divide the box in slices and calculate the average value of the order parameter per segment in one slice. The first method gives an idea of the ordering of a molecule from head to tail, the second method gives an idea of the ordering as function of the box length.

The electrostatic potential ($\psi$) across the interface can be computed from a trajectory by evaluating the

double integral of the charge density ($\rho(z)$):

$$\psi(z) - \psi(-\infty) = -\int_{-\infty}^{z} dz' \int_{-\infty}^{z'} \rho(z'')dz''/\epsilon_0 \tag{5.444}$$

where the position $z = -\infty$ is far enough in the bulk phase such that the field is zero. With this method, it is possible to "split" the total potential into separate contributions from lipid and water molecules. The program *gmx potential* (page 287) divides the box in slices and sums all charges of the atoms in each slice. It then integrates this charge density to give the electric field, which is in turn integrated to give the potential. Charge density, electric field, and potential are written to xvgr input files.

The program *gmx traj* (page 328) is a very simple analysis program. All it does is print the coordinates, velocities, or forces of selected atoms. It can also calculate the center of mass of one or more molecules and print the coordinates of the center of mass to three files. By itself, this is probably not a very useful analysis, but having the coordinates of selected molecules or atoms can be very handy for further analysis, not only in interfacial systems.

The program *gmx density* (page 185) calculates the mass density of groups and gives a plot of the density against a box axis. This is useful for looking at the distribution of groups or atoms across the interface.

## 5.11 Some implementation details

In this chapter we will present some implementation details. This is far from complete, but we deemed it necessary to clarify some things that would otherwise be hard to understand.

### 5.11.1 Single Sum Virial in GROMACS

The virial $\Xi$ can be written in full tensor form as:

$$\Xi = -\frac{1}{2}\sum_{i<j}^{N} \mathbf{r}_{ij} \otimes \mathbf{F}_{ij} \tag{5.445}$$

where $\otimes$ denotes the *direct product* of two vectors.[1] When this is computed in the inner loop of an MD program 9 multiplications and 9 additions are needed.[2]

Here it is shown how it is possible to extract the virial calculation from the inner loop *177* (page 692).

#### Virial

In a system with periodic boundary conditions, the periodicity must be taken into account for the virial:

$$\Xi = -\frac{1}{2}\sum_{i<j}^{N} \mathbf{r}_{ij}^{n} \otimes \mathbf{F}_{ij} \tag{5.446}$$

where $\mathbf{r}_{ij}^{n}$ denotes the distance vector of the *nearest image* of atom $i$ from atom $j$. In this definition we add a *shift vector* $\delta_i$ to the position vector $\mathbf{r}_i$ of atom $i$. The difference vector $\mathbf{r}_{ij}^{n}$ is thus equal to:

$$\mathbf{r}_{ij}^{n} = \mathbf{r}_i + \delta_i - \mathbf{r}_j \tag{5.447}$$

or in shorthand:

$$\mathbf{r}_{ij}^{n} = \mathbf{r}_i^{n} - \mathbf{r}_j \tag{5.448}$$

In a triclinic system, there are 27 possible images of $i$; when a truncated octahedron is used, there are 15 possible images.

#### Virial from non-bonded forces

Here the derivation for the single sum virial in the *non-bonded force* routine is given. There are a couple of considerations that are special to GROMACS that we take into account:

- When calculating short-range interactions, we apply the *minimum image convention* and only consider the closest image of each neighbor - and in particular we never allow interactions between a particle and any of its periodic images. For all the equations below, this means $i \neq j$.
- In general, either the $i$ or $j$ particle might be shifted to a neighbor cell to get the closest interaction (shift $\delta_{ij}$). However, with minimum image convention there can be at most 27 different shifts for particles in the central cell, and for typical (very short-ranged) biomolecular interactions there are typically only a few different shifts involved for each particle, not to mention that each interaction can only be present for one shift.

[1] Note that some derivations, an alternative notation $\xi_{\text{alt}} = v_\xi = p_\xi/Q$ is used.
[2] The calculation of Lennard-Jones and Coulomb forces is about 50 floating point operations.

- For the GROMACS nonbonded interactions we use this to split the neighborlist of each $i$ particle into multiple separate lists, where each list has a constant shift $\delta_i$ for the $i$ partlcle. We can represent this as a sum over shifts (for which we use index $s$), with the constraint that each particle interaction can only contribute to one of the terms in this sum, and the shift is no longer dependent on the $j$ particles. For any sum that does not contain complex dependence on $s$, this means the sum trivially reduces to just the sum over $i$ and/or $j$.
- To simplify some of the sums, we replace sums over $j < i$ with double sums over all particles (remember, $i \neq j$) and divide by 2.

Starting from the above definition of the virial, we then get

$$
\begin{aligned}
\Xi &= -\frac{1}{2}\sum_{i<j}^{N} \mathbf{r}_{ij}^{n} \otimes \mathbf{F}_{ij} \\
&= -\frac{1}{2}\sum_{i<j}^{N} (\mathbf{r}_i + \delta_{ij} - \mathbf{r}_j) \otimes \mathbf{F}_{ij} \\
&= -\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} (\mathbf{r}_i + \delta_{ij} - \mathbf{r}_j) \otimes \mathbf{F}_{ij} \\
&= -\frac{1}{4}\sum_{i=1}^{N}\sum_{s}\sum_{j=1}^{N} (\mathbf{r}_i + \delta_{i,s} - \mathbf{r}_j) \otimes \mathbf{F}_{ij,s} \\
&= -\frac{1}{4}\sum_{i=}^{N}\sum_{s}\sum_{j=1}^{N} ((\mathbf{r}_i + \delta_{i,s}) \otimes \mathbf{F}_{ij,s} - \mathbf{r}_j \otimes \mathbf{F}_{ij,s}) \\
&= -\frac{1}{4}\sum_{i=1}^{N}\sum_{s}\sum_{j=1}^{N} (\mathbf{r}_i + \delta_{i,s}) \otimes \mathbf{F}_{ij,s} + \frac{1}{4}\sum_{i=1}^{N}\sum_{s}\sum_{j=1}^{N} \mathbf{r}_j \otimes \mathbf{F}_{ij,s} \\
&= -\frac{1}{4}\sum_{i=1}^{N}\sum_{s}\sum_{j=1}^{N} (\mathbf{r}_i + \delta_{i,s}) \otimes \mathbf{F}_{ij,s} + \frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} \mathbf{r}_j \otimes \mathbf{F}_{ij} \\
&= -\frac{1}{4}\sum_{s}\sum_{i=1}^{N} (\mathbf{r}_i + \delta_{i,s}) \otimes \sum_{j=1}^{N} \mathbf{F}_{ij,s} + \frac{1}{4}\sum_{j=1}^{N} \mathbf{r}_j \otimes \sum_{i=1}^{N} \mathbf{F}_{ij} \\
&= -\frac{1}{4}\sum_{s}\sum_{i=1}^{N} (\mathbf{r}_i + \delta_{i,s}) \otimes \sum_{j=1}^{N} \mathbf{F}_{ij,s} - \frac{1}{4}\sum_{j=1}^{N} \mathbf{r}_j \otimes \sum_{i=1}^{N} \mathbf{F}_{ji} \\
&= -\frac{1}{4}\sum_{s}\sum_{i=1}^{N} (\mathbf{r}_i + \delta_{i,s}) \otimes \mathbf{F}_{i,s} - \frac{1}{4}\sum_{j=1}^{N} \mathbf{r}_j \otimes \mathbf{F}_j \\
&= -\frac{1}{4}\left(\sum_{i=1}^{N} \mathbf{r}_i \otimes \mathbf{F}_i + \sum_{j=1}^{N} \mathbf{r}_j \otimes \mathbf{F}_j\right) - \frac{1}{4}\sum_{s}\sum_{i=1}^{N} \delta_{i,s} \otimes \mathbf{F}_{i,s} \\
&= -\frac{1}{2}\sum_{i=1}^{N} \mathbf{r}_i \otimes \mathbf{F}_i - \frac{1}{4}\sum_{s}\sum_{i=1}^{N} \delta_{i,s} \otimes \mathbf{F}_{i,s} \\
&= -\frac{1}{2}\sum_{i=1}^{N} \mathbf{r}_i \otimes \mathbf{F}_i - \frac{1}{4}\sum_{s} \delta_s \otimes \mathbf{F}_s \\
&= \Xi_0 + \Xi_1
\end{aligned}
$$

In the second-last stage, we have used the property that each shift vector itself does not depend on the

coordinates of particle $i$, so it is possible to sum up all forces corresponding to each shift vector (in the nonbonded kernels), and then just use a sum over the different shift vectors outside the kernels. We have also used

$$\begin{aligned} \mathbf{F}_i &= \sum_{j=1}^{N} \mathbf{F}_{ij} \\ \mathbf{F}_j &= \sum_{i=1}^{N} \mathbf{F}_{ji} \end{aligned} \tag{5.449}$$

which is the total force on $i$ with respect to $j$. Because we use Newton's Third Law:

$$\mathbf{F}_{ij} = -\mathbf{F}_{ji} \tag{5.450}$$

we must, in the implementation, double the term containing the shift $\delta_i$. Similarly, in a few places we have summed the shift-dependent force over all shifts to come up with the total force per interaction or particle.

This separates the total virial $\Xi$ into a component $\Xi_0$ that is a single sum over particles, and a second component $\Xi_1$ that describes the influence of the particle shifts, and that is only a sum over the different shift vectors.

### The intra-molecular shift (mol-shift)

For the bonded forces and SHAKE it is possible to make a *mol-shift* list, in which the periodicity is stored. We simple have an array mshift in which for each atom an index in the shiftvec array is stored.

The algorithm to generate such a list can be derived from graph theory, considering each particle in a molecule as a bead in a graph, the bonds as edges.

1. Represent the bonds and atoms as bidirectional graph
2. Make all atoms white
3. Make one of the white atoms black (atom $i$) and put it in the central box
4. Make all of the neighbors of $i$ that are currently white, gray
5. Pick one of the gray atoms (atom $j$), give it the correct periodicity with respect to any of its black neighbors and make it black
6. Make all of the neighbors of $j$ that are currently white, gray
7. If any gray atom remains, go to [5]
8. If any white atom remains, go to [3]

Using this algorithm we can

- optimize the bonded force calculation as well as SHAKE
- calculate the virial from the bonded forces in the single sum method again

Find a representation of the bonds as a bidirectional graph.

### Virial from Covalent Bonds

Since the covalent bond force gives a contribution to the virial, we have:

$$\begin{aligned} b &= \|\mathbf{r}_{ij}^n\| \\ V_b &= \frac{1}{2}k_b(b-b_0)^2 \\ \mathbf{F}_i &= -\nabla V_b \\ &= k_b(b-b_0)\frac{\mathbf{r}_{ij}^n}{b} \\ \mathbf{F}_j &= -\mathbf{F}_i \end{aligned} \tag{5.451}$$

The virial contribution from the bonds then is:

$$\begin{aligned} \Xi_b &= -\frac{1}{2}(\mathbf{r}_i^n \otimes \mathbf{F}_i + \mathbf{r}_j \otimes \mathbf{F}_j) \\ &= -\frac{1}{2}\mathbf{r}_{ij}^n \otimes \mathbf{F}_i \end{aligned} \tag{5.452}$$

### Virial from SHAKE

An important contribution to the virial comes from shake. Satisfying the constraints a force $\mathbf{G}$ that is exerted on the particles "shaken." If this force does not come out of the algorithm (as in standard SHAKE) it can be calculated afterward (when using *leap-frog*) by:

$$\begin{aligned} \Delta\mathbf{r}_i &= \mathbf{r}_i(t+\Delta t) - [\mathbf{r}_i(t) + \mathbf{v}_i(t-\frac{\Delta t}{2})\Delta t + \frac{\mathbf{F}_i}{m_i}\Delta t^2] \\ \mathbf{G}_i &= \frac{m_i\Delta\mathbf{r}_i}{\Delta t^2 i} \end{aligned} \tag{5.453}$$

This does not help us in the general case. Only when no periodicity is needed (like in rigid water) this can be used, otherwise we must add the virial calculation in the inner loop of SHAKE.

When it *is* applicable the virial can be calculated in the single sum way:

$$\Xi = -\frac{1}{2}\sum_i^{N_c} \mathbf{r}_i \otimes \mathbf{F}_i \tag{5.454}$$

where $N_c$ is the number of constrained atoms.

## 5.11.2 Optimizations

Here we describe some of the algorithmic optimizations used in GROMACS, apart from parallelism.

### Inner Loops for Water

GROMACS uses special inner loops to calculate non-bonded interactions for water molecules with other atoms, and yet another set of loops for interactions between pairs of water molecules. There highly optimized loops for two types of water models. For three site models similar to SPC *80* (page 687), *i.e.*:

1. There are three atoms in the molecule.
2. The whole molecule is a single charge group.

3. The first atom has Lennard-Jones (sec. *The Lennard-Jones interaction* (page 481)) and Coulomb (sec. *Coulomb interaction* (page 483)) interactions.
4. Atoms two and three have only Coulomb interactions, and equal charges.

These loops also works for the SPC/E *178* (page 692) and TIP3P *128* (page 689) water models. And for four site water models similar to TIP4P *128* (page 689):

1. There are four atoms in the molecule.
2. The whole molecule is a single charge group.
3. The first atom has only Lennard-Jones (sec. *The Lennard-Jones interaction* (page 481)) interactions.
4. Atoms two and three have only Coulomb (sec. *Coulomb interaction* (page 483)) interactions, and equal charges.
5. Atom four has only Coulomb interactions.

The benefit of these implementations is that there are more floating-point operations in a single loop, which implies that some compilers can schedule the code better. However, it turns out that even some of the most advanced compilers have problems with scheduling, implying that manual tweaking is necessary to get optimum performance. This may include common-sub-expression elimination, or moving code around.

## 5.12 Averages and fluctuations

### 5.12.1 Formulae for averaging

**Note:** this section was taken from ref *179* (page 692).

When analyzing a MD trajectory averages $\langle x \rangle$ and fluctuations

$$\left\langle (\Delta x)^2 \right\rangle^{\frac{1}{2}} \; = \; \left\langle [x - \langle x \rangle]^2 \right\rangle^{\frac{1}{2}} \tag{5.455}$$

of a quantity $x$ are to be computed. The variance $\sigma_x$ of a series of $N_x$ values, $\{x_i\}$, can be computed from

$$\sigma_x \; = \; \sum_{i=1}^{N_x} x_i^2 \; - \; \frac{1}{N_x} \left( \sum_{i=1}^{N_x} x_i \right)^2 \tag{5.456}$$

Unfortunately this formula is numerically not very accurate, especially when $\sigma_x^{\frac{1}{2}}$ is small compared to the values of $x_i$. The following (equivalent) expression is numerically more accurate

$$\sigma_x \; = \; \sum_{i=1}^{N_x} [x_i - \langle x \rangle]^2 \tag{5.457}$$

with

$$\langle x \rangle \; = \; \frac{1}{N_x} \sum_{i=1}^{N_x} x_i \tag{5.458}$$

Using (5.456) and (5.458) one has to go through the series of $x_i$ values twice, once to determine $\langle x \rangle$ and again to compute $\sigma_x$, whereas (5.455) requires only one sequential scan of the series $\{x_i\}$. However, one may cast (5.456) in another form, containing partial sums, which allows for a sequential update algorithm. Define the partial sum

$$X_{n,m} \; = \; \sum_{i=n}^{m} x_i \tag{5.459}$$

and the partial variance

$$\sigma_{n,m} \; = \; \sum_{i=n}^{m} \left[ x_i - \frac{X_{n,m}}{m-n+1} \right]^2 \tag{5.460}$$

It can be shown that

$$X_{n,m+k} \; = \; X_{n,m} + X_{m+1,m+k} \tag{5.461}$$

and

$$\sigma_{n,m+k} = \; \sigma_{n,m} + \sigma_{m+1,m+k} + \left[ \frac{X_{n,m}}{m-n+1} - \frac{X_{n,m+k}}{m+k-n+1} \right]^2 * \frac{(m-n+1)(m+k-n+1)}{k}$$

For $n = 1$ one finds

$$\sigma_{1,m+k} \; = \; \sigma_{1,m} + \sigma_{m+1,m+k} \; + \; \left[ \frac{X_{1,m}}{m} - \frac{X_{1,m+k}}{m+k} \right]^2 \; \frac{m(m+k)}{k} \tag{5.462}$$

and for $n = 1$ and $k = 1$ (5.462) becomes

$$\begin{aligned} \sigma_{1,m+1} &= \sigma_{1,m} + \left[\frac{X_{1,m}}{m} - \frac{X_{1,m+1}}{m+1}\right]^2 m(m+1) \\ &= \sigma_{1,m} + \frac{[\, X_{1,m} - m x_{m+1} \,]^2}{m(m+1)} \end{aligned} \tag{5.463}$$

where we have used the relation

$$X_{1,m+1} = X_{1,m} + x_{m+1} \tag{5.464}$$

Using formulae (5.463) and (5.464) the average

$$\langle x \rangle = \frac{X_{1,N_x}}{N_x} \tag{5.465}$$

and the fluctuation

$$\left\langle (\Delta x)^2 \right\rangle^{\frac{1}{2}} = \left[\frac{\sigma_{1,N_x}}{N_x}\right]^{\frac{1}{2}} \tag{5.466}$$

can be obtained by one sweep through the data.

### 5.12.2 Implementation

In GROMACS the instantaneous energies $E(m)$ are stored in the *energy file* (page 576), along with the values of $\sigma_{1,m}$ and $X_{1,m}$. Although the steps are counted from 0, for the energy and fluctuations steps are counted from 1. This means that the equations presented here are the ones that are implemented. We give somewhat lengthy derivations in this section to simplify checking of code and equations later on.

#### Part of a Simulation

It is not uncommon to perform a simulation where the first part, *e.g.* 100 ps, is taken as equilibration. However, the averages and fluctuations as printed in the *log file* (page 578) are computed over the whole simulation. The equilibration time, which is now part of the simulation, may in such a case invalidate the averages and fluctuations, because these numbers are now dominated by the initial drift towards equilibrium.

Using (5.461) and (5.462) the average and standard deviation over part of the trajectory can be computed as:

$$\begin{aligned} X_{m+1,m+k} &= X_{1,m+k} - X_{1,m} \\ \sigma_{m+1,m+k} &= \sigma_{1,m+k} - \sigma_{1,m} - \left[\frac{X_{1,m}}{m} - \frac{X_{1,m+k}}{m+k}\right]^2 \frac{m(m+k)}{k} \end{aligned} \tag{5.467}$$

or, more generally (with $p \geq 1$ and $q \geq p$):

$$\begin{aligned} X_{p,q} &= X_{1,q} - X_{1,p-1} \\ \sigma_{p,q} &= \sigma_{1,q} - \sigma_{1,p-1} - \left[\frac{X_{1,p-1}}{p-1} - \frac{X_{1,q}}{q}\right]^2 \frac{(p-1)q}{q-p+1} \end{aligned} \tag{5.468}$$

**Note** that implementation of this is not entirely trivial, since energies are not stored every time step of the simulation. We therefore have to construct $X_{1,p-1}$ and $\sigma_{1,p-1}$ from the information at time $p$ using (5.463) and (5.464):

$$\begin{aligned} X_{1,p-1} &= X_{1,p} - x_p \\ \sigma_{1,p-1} &= \sigma_{1,p} - \frac{[\, X_{1,p-1} - (p-1)x_p \,]^2}{(p-1)p} \end{aligned} \tag{5.469}$$

### Combining two simulations

Another frequently occurring problem is, that the fluctuations of two simulations must be combined. Consider the following example: we have two simulations (A) of $n$ and (B) of $m$ steps, in which the second simulation is a continuation of the first. However, the second simulation starts numbering from 1 instead of from $n+1$. For the partial sum this is no problem, we have to add $X^A_{1,n}$ from run A:

$$X^{AB}_{1,n+m} \;=\; X^A_{1,n} + X^B_{1,m} \tag{5.470}$$

When we want to compute the partial variance from the two components we have to make a correction $\Delta\sigma$:

$$\sigma^{AB}_{1,n+m} \;=\; \sigma^A_{1,n} + \sigma^B_{1,m} + \Delta\sigma \tag{5.471}$$

if we define $x^{AB}_i$ as the combined and renumbered set of data points we can write:

$$\sigma^{AB}_{1,n+m} \;=\; \sum_{i=1}^{n+m}\left[x^{AB}_i - \frac{X^{AB}_{1,n+m}}{n+m}\right]^2 \tag{5.472}$$

and thus

$$\sum_{i=1}^{n+m}\left[x^{AB}_i - \frac{X^{AB}_{1,n+m}}{n+m}\right]^2 = \sum_{i=1}^{n}\left[x^{A}_i - \frac{X^{A}_{1,n}}{n}\right]^2 + \sum_{i=1}^{m}\left[x^{B}_i - \frac{X^{B}_{1,m}}{m}\right]^2 + \Delta\sigma \tag{5.473}$$

or

$$\begin{aligned}
&\sum_{i=1}^{n+m}\left[(x^{AB}_i)^2 - 2x^{AB}_i\frac{X^{AB}_{1,n+m}}{n+m} + \left(\frac{X^{AB}_{1,n+m}}{n+m}\right)^2\right] - \\
&\qquad\sum_{i=1}^{n}\left[(x^{A}_i)^2 - 2x^{A}_i\frac{X^{A}_{1,n}}{n} + \left(\frac{X^{A}_{1,n}}{n}\right)^2\right] - \\
&\qquad\sum_{i=1}^{m}\left[(x^{B}_i)^2 - 2x^{B}_i\frac{X^{B}_{1,m}}{m} + \left(\frac{X^{B}_{1,m}}{m}\right)^2\right] = \quad \Delta\sigma
\end{aligned}$$

all the $x^2_i$ terms drop out, and the terms independent of the summation counter $i$ can be simplified:

$$\begin{aligned}
&\frac{\left(X^{AB}_{1,n+m}\right)^2}{n+m} - \frac{\left(X^{A}_{1,n}\right)^2}{n} - \frac{\left(X^{B}_{1,m}\right)^2}{m} - \\
&2\,\frac{X^{AB}_{1,n+m}}{n+m}\sum_{i=1}^{n+m}x^{AB}_i + 2\,\frac{X^{A}_{1,n}}{n}\sum_{i=1}^{n}x^{A}_i + 2\,\frac{X^{B}_{1,m}}{m}\sum_{i=1}^{m}x^{B}_i = \quad \Delta\sigma
\end{aligned}$$

we recognize the three partial sums on the second line and use (5.470) to obtain:

$$\Delta\sigma \;=\; \frac{\left(mX^A_{1,n} - nX^B_{1,m}\right)^2}{nm(n+m)} \tag{5.474}$$

if we check this by inserting $m=1$ we get back (5.463)

### Summing energy terms

The *gmx energy* (page 214) program can also sum energy terms into one, *e.g.* potential + kinetic = total. For the partial averages this is again easy if we have $S$ energy components $s$:

$$X^S_{m,n} = \sum_{i=m}^{n}\sum_{s=1}^{S} x_i^s = \sum_{s=1}^{S}\sum_{i=m}^{n} x_i^s = \sum_{s=1}^{S} X^s_{m,n} \tag{5.475}$$

For the fluctuations it is less trivial again, considering for example that the fluctuation in potential and kinetic energy should cancel. Nevertheless we can try the same approach as before by writing:

$$\sigma^S_{m,n} = \sum_{s=1}^{S} \sigma^s_{m,n} + \Delta\sigma \tag{5.476}$$

if we fill in (5.460):

$$\sum_{i=m}^{n}\left[\left(\sum_{s=1}^{S} x_i^s\right) - \frac{X^S_{m,n}}{m-n+1}\right]^2 = \sum_{s=1}^{S}\sum_{i=m}^{n}\left[(x_i^s) - \frac{X^s_{m,n}}{m-n+1}\right]^2 + \Delta\sigma \tag{5.477}$$

which we can expand to:

$$\sum_{i=m}^{n}\left[\sum_{s=1}^{S}(x_i^s)^2 + \left(\frac{X^S_{m,n}}{m-n+1}\right)^2 - 2\left(\frac{X^S_{m,n}}{m-n+1}\sum_{s=1}^{S} x_i^s + \sum_{s=1}^{S}\sum_{s'=s+1}^{S} x_i^s x_i^{s'}\right)\right]$$
$$- \sum_{s=1}^{S}\sum_{i=m}^{n}\left[(x_i^s)^2 - 2\frac{X^s_{m,n}}{m-n+1}x_i^s + \left(\frac{X^s_{m,n}}{m-n+1}\right)^2\right] = \Delta\sigma$$

the terms with $(x_i^s)^2$ cancel, so that we can simplify to:

$$\frac{\left(X^S_{m,n}\right)^2}{m-n+1} - 2\frac{X^S_{m,n}}{m-n+1}\sum_{i=m}^{n}\sum_{s=1}^{S} x_i^s - 2\sum_{i=m}^{n}\sum_{s=1}^{S}\sum_{s'=s+1}^{S} x_i^s x_i^{s'} -$$
$$\sum_{s=1}^{S}\sum_{i=m}^{n}\left[-2\frac{X^s_{m,n}}{m-n+1}x_i^s + \left(\frac{X^s_{m,n}}{m-n+1}\right)^2\right] = \Delta\sigma$$

or

$$-\frac{\left(X^S_{m,n}\right)^2}{m-n+1} - 2\sum_{i=m}^{n}\sum_{s=1}^{S}\sum_{s'=s+1}^{S} x_i^s x_i^{s'} + \sum_{s=1}^{S}\frac{\left(X^s_{m,n}\right)^2}{m-n+1} = \Delta\sigma \tag{5.478}$$

If we now expand the first term using (5.475) we obtain:

$$-\frac{\left(\sum_{s=1}^{S} X^s_{m,n}\right)^2}{m-n+1} - 2\sum_{i=m}^{n}\sum_{s=1}^{S}\sum_{s'=s+1}^{S} x_i^s x_i^{s'} + \sum_{s=1}^{S}\frac{\left(X^s_{m,n}\right)^2}{m-n+1} = \Delta\sigma \tag{5.479}$$

which we can reformulate to:

$$-2\left[\sum_{s=1}^{S}\sum_{s'=s+1}^{S} X^s_{m,n}X^{s'}_{m,n} + \sum_{i=m}^{n}\sum_{s=1}^{S}\sum_{s'=s+1}^{S} x_i^s x_i^{s'}\right] = \Delta\sigma \tag{5.480}$$

or

$$-2\left[\sum_{s=1}^{S} X^s_{m,n}\sum_{s'=s+1}^{S} X^{s'}_{m,n} + \sum_{s=1}^{S}\sum_{i=m}^{n} x_i^s \sum_{s'=s+1}^{S} x_i^{s'}\right] = \Delta\sigma \tag{5.481}$$

which gives

$$-2\sum_{s=1}^{S}\left[X_{m,n}^{s}\sum_{s'=s+1}^{S}\sum_{i=m}^{n}x_i^{s'}+\sum_{i=m}^{n}x_i^{s}\sum_{s'=s+1}^{S}x_i^{s'}\right]=\Delta\sigma \tag{5.482}$$

Since we need all data points $i$ to evaluate this, in general this is not possible. We can then make an estimate of $\sigma_{m,n}^{S}$ using only the data points that are available using the left hand side of (5.477). While the average can be computed using all time steps in the simulation, the accuracy of the fluctuations is thus limited by the frequency with which energies are saved. Since this can be easily done with a program such as `xmgr` this is not built-in in GROMACS.

## 5.13 Bibliography

[1] H. Bekker, H.J.C. Berendsen, E.J. Dijkstra, S. Achterop, R. van Drunen, D. van der Spoel, A. Sijbers, and H. Keegstra *et al.*, "Gromacs: A parallel computer for molecular dynamics simulations"; pp. 252–256 in *Physics computing 92*. Edited by R.A. de Groot and J. Nadrchal. World Scientific, Singapore, 1993.

[2] H.J.C. Berendsen, D. van der Spoel, and R. van Drunen, "GROMACS: A message-passing parallel molecular dynamics implementation," *Comp. Phys. Comm.*, **91** 43–56 (1995).

[3] E. Lindahl, B. Hess, and D. van der Spoel, "GROMACS 3.0: A package for molecular simulation and trajectory analysis," *J. Mol. Mod.*, **7** 306–317 (2001).

[4] D. van der Spoel, E. Lindahl, B. Hess, G. Groenhof, A.E. Mark, and H.J.C. Berendsen, "GROMACS: Fast, Flexible and Free," *J. Comp. Chem.*, **26** 1701–1718 (2005).

[5] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, "GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation," *J. Chem. Theory Comput.*, **4** [3] 435–447 (2008).

[6] S. Pronk, S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M.R. Shirts, and J.C. Smith *et al.*, "GROMACS 4.5: A high-throughput and highly parallel open source molecular simulation toolkit," *Bioinformatics*, **29** [7] 845–854 (2013).

[7] S. Páll, M.J. Abraham, C. Kutzner, B. Hess, and E. Lindahl, "Tackling exascale software challenges in molecular dynamics simulations with GROMACS"; pp. 3–27 in *Solving software challenges for exascale*. Edited by S. Markidis and E. Laure. Springer International Publishing Switzerland, London, 2015.

[8] M.J. Abraham, T. Murtola, R. Schulz, S. Páll, J.C. Smith, B. Hess, and E. Lindahl, "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers," *SoftwareX*, **1–2** 19–25 (2015).

[9] W.F. van Gunsteren and H.J.C. Berendsen, "Computer simulation of molecular dynamics: Methodology, applications, and perspectives in chemistry," *Angew. Chem. Int. Ed. Engl.*, **29** 992–1023 (1990).

[10] J.G.E.M. Fraaije, "Dynamic density functional theory for microphase separation kinetics of block copolymer melts," *J. Chem. Phys.*, **99** 9202–9212 (1993).

[11] D.A. McQuarrie, *Statistical mechanics*. Harper & Row, New York, 1976.

[12] W.F. van Gunsteren and H.J.C. Berendsen, "Algorithms for macromolecular dynamics and constraint dynamics," *Mol. Phys.*, **34** 1311–1327 (1977).

[13] W.F. van Gunsteren and M. Karplus, "Effect of constraints on the dynamics of macromolecules," *Macromolecules*, **15** 1528–1544 (1982).

[14] T. Darden, D. York, and L. Pedersen, "Particle mesh Ewald: An N•log(N) method for Ewald sums in large systems," *J. Chem. Phys.*, **98** 10089–10092 (1993).

[15] U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee, and L.G. Pedersen, "A smooth particle mesh ewald potential," *J. Chem. Phys.*, **103** 8577–8592 (1995).

[16] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images," *IEEE Trans. Patt. Anal. Mach. Int.*, **6** 721 (1984).

[17] M. Nilges, G.M. Clore, and A.M. Gronenborn, "Determination of three-dimensional structures of proteins from interproton distance data by dynamical simulated annealing from a random array of atoms," *FEBS Lett.*, **239** 129–136 (1988).

[18] R.C. van Schaik, H.J.C. Berendsen, A.E. Torda, and W.F. van Gunsteren, "A structure refinement method based on molecular dynamics in 4 spatial dimensions," *J. Mol. Biol.*, **234** 751–762 (1993).

[19] K. Zimmerman, "All purpose molecular mechanics simulator and energy minimizer," *J. Comp. Chem.*, **12** 310–319 (1991).

[20] D.J. Adams, E.M. Adams, and G.J. Hills, "The computer simulation of polar liquids," *Mol. Phys.*, **38** 387–400 (1979).

[21] H. Bekker, E.J. Dijkstra, M.K.R. Renardus, and H.J.C. Berendsen, "An efficient, box shape independent non-bonded force and virial algorithm for molecular dynamics," *Mol. Sim.*, **14** 137–152 (1995).

[22] R.W. Hockney, S.P. Goel, and J. Eastwood, "Quiet High Resolution Computer Models of a Plasma," *J. Comp. Phys.*, **14** 148–158 (1974).

[23] L. Verlet., "Computer experiments on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules," *Phys. Rev.*, **159** 98–103 (1967).

[24] H.J.C. Berendsen and W.F. van Gunsteren, "Practical algorithms for dynamics simulations"; in 1986.

[25] W.C. Swope, H.C. Andersen, P.H. Berens, and K.R. Wilson, "A computer-simulation method for the calculation of equilibrium-constants for the formation of physical clusters of molecules: Application to small water clusters," *J. Chem. Phys.*, **76** 637–649 (1982).

[26] H.J.C. Berendsen, J.P.M. Postma, A. DiNola, and J.R. Haak, "Molecular dynamics with coupling to an external bath," *J. Chem. Phys.*, **81** 3684–3690 (1984).

[27] H.C. Andersen, "Molecular dynamics simulations at constant pressure and/or temperature," *J. Chem. Phys.*, **72** 2384 (1980).

[28] S. Nosé, "A molecular dynamics method for simulations in the canonical ensemble," *Mol. Phys.*, **52** 255–268 (1984).

[29] W.G. Hoover, "Canonical dynamics: Equilibrium phase-space distributions," *Phys. Rev. **A**, **31** 1695–1697 (1985).

[30] G. Bussi, D. Donadio, and M. Parrinello, "Canonical sampling through velocity rescaling," *J. Chem. Phys.*, **126** 014101 (2007).

[31] H.J.C. Berendsen, "Transport properties computed by linear response through weak coupling to a bath"; pp. 139–155 in *Computer simulations in material science*. Edited by M. Meyer and V. Pontikis. Kluwer, 1991.

[32] J.E. Basconi and M.R. Shirts, "Effects of temperature control algorithms on transport properties and kinetics in molecular dynamics simulations," *J. Chem. Theory Comput.*, **9** [7] 2887–2899 (2013).

[33] B. Cooke and S.J. Schmidler, "Preserving the Boltzmann ensemble in replica-exchange molecular dynamics," *J. Chem. Phys.*, **129** 164112 (2008).

[34] G.J. Martyna, M.L. Klein, and M.E. Tuckerman, "Nosé-Hoover chains: The canonical ensemble via continuous dynamics," *J. Chem. Phys.*, **97** 2635–2643 (1992).

[35] G.J. Martyna, M.E. Tuckerman, D.J. Tobias, and M.L. Klein, "Explicit reversible integrators for extended systems dynamics," *Mol. Phys.*, **87** 1117–1157 (1996).

[36] B.L. Holian, A.F. Voter, and R. Ravelo, "Thermostatted molecular dynamics: How to avoid the Toda demon hidden in Nosé-Hoover dynamics," *Phys. Rev. E*, **52** [3] 2338–2347 (1995).

[37] M.P. Eastwood, K.A. Stafford, R.A. Lippert, M.Ø. Jensen, P. Maragakis, C. Predescu, R.O. Dror, and D.E. Shaw, "Equipartition and the calculation of temperature in biomolecular simulations," *J. Chem. Theory Comput.*, **ASAP** DOI: 10.1021/ct9002916 (2010).

[38] M. Parrinello and A. Rahman, "Polymorphic transitions in single crystals: A new molecular dynamics method," *J. Appl. Phys.*, **52** 7182–7190 (1981).

[39] S. Nosé and M.L. Klein, "Constant pressure molecular dynamics for molecular systems," *Mol. Phys.*, **50** 1055–1076 (1983).

[40] G. Liu, "Dynamical equations for the period vectors in a periodic system under constant external stress," *Can. J. Phys.*, **93** 974–978 (2015).

[41] M.E. Tuckerman, J. Alejandre, R. López-Rendón, A.L. Jochim, and G.J. Martyna, "A Liouville-operator derived measure-preserving integrator for molecular dynamics simulations in the isothermal-isobaric ensemble," *J. Phys. A.*, **59** 5629–5651 (2006).

[42] T.-Q. Yu, J. Alejandre, R. Lopez-Rendon, G.J. Martyna, and M.E. Tuckerman, "Measure-preserving integrators for molecular dynamics in the isothermal-isobaric ensemble derived from the liouville operator," *Chem. Phys.*, **370** 294–305 (2010).

[43] B.G. Dick and A.W. Overhauser, "Theory of the dielectric constants of alkali halide crystals," *Phys. Rev.*, **112** 90–103 (1958).

[44] P.C. Jordan, P.J. van Maaren, J. Mavri, D. van der Spoel, and H.J.C. Berendsen, "Towards phase transferable potential functions: Methodology and application to nitrogen," *J. Chem. Phys.*, **103** 2272–2285 (1995).

[45] P.J. van Maaren and D. van der Spoel, "Molecular dynamics simulations of a water with a novel shell-model potential," *J. Phys. Chem. B.*, **105** 2618–2626 (2001).

[46] J.P. Ryckaert, G. Ciccotti, and H.J.C. Berendsen, "Numerical integration of the cartesian equations of motion of a system with constraints; molecular dynamics of n-alkanes," *J. Comp. Phys.*, **23** 327–341 (1977).

[47] S. Miyamoto and P.A. Kollman, "SETTLE: An analytical version of the SHAKE and RATTLE algorithms for rigid water models," *J. Comp. Chem.*, **13** 952–962 (1992).

[48] H.C. Andersen, "RATTLE: A 'Velocity' version of the SHAKE algorithm for molecular dynamics calculations," *J. Comp. Phys.*, **52** 24–34 (1983).

[49] B. Hess, H. Bekker, H.J.C. Berendsen, and J.G.E.M. Fraaije, "LINCS: A linear constraint solver for molecular simulations," *J. Comp. Chem.*, **18** 1463–1472 (1997).

[50] B. Hess, "P-LINCS: A parallel linear constraint solver for molecular simulation," *J. Chem. Theory Comput.*, **4** 116–122 (2007).

[51] N. Goga, A.J. Rzepiela, A.H. de Vries, S.J. Marrink, and H.J.C. Berendsen, "Efficient algorithms for Langevin and DPD dynamics," *J. Chem. Theory Comput.*, **8** 3637–3649 (2012).

[52] R.H. Byrd, P. Lu, and J. Nocedal, "A limited memory algorithm for bound constrained optimization," *SIAM J. Scientif. Statistic. Comput.*, **16** 1190–1208 (1995).

[53] C. Zhu, R.H. Byrd, and J. Nocedal, "L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization," *ACM Trans. Math. Softw.*, **23** 550–560 (1997).

[54] M. Levitt, C. Sander, and P.S. Stern, "The normal modes of a protein: Native bovine pancreatic trypsin inhibitor," *Int. J. Quant. Chem: Quant. Biol. Symp.*, **10** 181–199 (1983).

[55] N. Gō, T. Noguti, and T. Nishikawa, "Dynamics of a small globular protein in terms of low-frequency vibrational modes," *Proc. Natl. Acad. Sci. USA*, **80** 3696–3700 (1983).

[56] B. Brooks and M. Karplus, "Harmonic dynamics of proteins: Normal modes and fluctuations in bovine pancreatic trypsin inhibitor," *Proc. Natl. Acad. Sci. USA*, **80** 6571–6575 (1983).

[57] S. Hayward and N. Gō, "Collective variable description of native protein dynamics," *Annu. Rev. Phys. Chem.*, **46** 223–250 (1995).

[58] C.H. Bennett, "Efficient Estimation of Free Energy Differences from Monte Carlo Data," *J. Comp. Phys.*, **22** 245–268 (1976).

[59] M.R. Shirts and J.D. Chodera, "Statistically optimal analysis of multiple equilibrium simulations," *J. Chem. Phys.*, **129** 124105 (2008).

[60] K. Hukushima and K. Nemoto, "Exchange Monte Carlo Method and Application to Spin Glass Simulations," *J. Phys. Soc. Jpn.*, **65** 1604–1608 (1996).

[61] Y. Sugita and Y. Okamoto, "Replica-exchange molecular dynamics method for protein folding," *Chem. Phys. Lett.*, **314** 141–151 (1999).

[62] M. Seibert, A. Patriksson, B. Hess, and D. van der Spoel, "Reproducible polypeptide folding and structure prediction using molecular dynamics simulations," *J. Mol. Biol.*, **354** 173–183 (2005).

[63] T. Okabe, M. Kawata, Y. Okamoto, and M. Mikami, "Replica-exchange Monte Carlo method for the isobaric-isothermal ensemble," *Chem. Phys. Lett.*, **335** 435–439 (2001).

[64] J.D. Chodera and M.R. Shirts, "Replica exchange and expanded ensemble simulations as gibbs sampling: Simple improvements for enhanced mixing," *J. Chem. Phys.*, **135** 194110 (2011).

[65] B.L. de Groot, A. Amadei, D.M.F. van Aalten, and H.J.C. Berendsen, "Towards an exhaustive sampling of the configurational spaces of the two forms of the peptide hormone guanylin," *J. Biomol. Str. Dyn.*, **13** [5] 741–751 (1996).

[66] B.L. de Groot, A. Amadei, R.M. Scheek, N.A.J. van Nuland, and H.J.C. Berendsen, "An extended sampling of the configurational space of HPr from *E. coli*," *PROTEINS: Struct. Funct. Gen.*, **26** 314–322 (1996).

[67] O.E. Lange, L.V. Schafer, and H. Grubmuller, "Flooding in GROMACS: Accelerated barrier crossings in molecular dynamics," *J. Comp. Chem.*, **27** 1693–1702 (2006).

[68] A.P. Lyubartsev, A.A. Martsinovski, S.V. Shevkunov, and P.N. Vorontsov-Velyaminov, "New approach to Monte Carlo calculation of the free energy: Method of expanded ensembles," *J. Chem. Phys.*, **96** 1776–1783 (1992).

[69] S.Y. Liem, D. Brown, and J.H.R. Clarke, "Molecular dynamics simulations on distributed memory machines," *Comput. Phys. Commun.*, **67** [2] 261–267 (1991).

[70] K.J. Bowers, R.O. Dror, and D.E. Shaw, "The midpoint method for parallelization of particle simulations," *J. Chem. Phys.*, **124** [18] 184109–184109 (2006).

[72] D. van der Spoel and P.J. van Maaren, "The origin of layer structure artifacts in simulations of liquid water," *J. Chem. Theory Comput.*, **2** 1–11 (2006).

[73] I. Ohmine, H. Tanaka, and P.G. Wolynes, "Large local energy fluctuations in water. II. Cooperative motions and fluctuations," *J. Chem. Phys.*, **89** 5852–5860 (1988).

[74] D.B. Kitchen, F. Hirata, J.D. Westbrook, R. Levy, D. Kofke, and M. Yarmush, "Conserving energy during molecular dynamics simulations of water, proteins, and proteins in water," *J. Comp. Chem.*, **11** 1169–1180 (1990).

[75] J. Guenot and P.A. Kollman, "Conformational and energetic effects of truncating nonbonded interactions in an aqueous protein dynamics simulation," *J. Comp. Chem.*, **14** 295–311 (1993).

[76] P.J. Steinbach and B.R. Brooks, "New spherical-cutoff methods for long-range forces in macromolecular simulation," *J. Comp. Chem.*, **15** 667–683 (1994).

[77] W.F. van Gunsteren, S.R. Billeter, A.A. Eising, P.H. Hünenberger, P. Krüger, A.E. Mark, W.R.P. Scott, and I.G. Tironi, *Biomolecular simulation: The GROMOS96 manual and user guide*. Hochschulverlag AG an der ETH Zürich, Zürich, Switzerland, 1996.

[78] W.F. van Gunsteren and H.J.C. Berendsen, *Gromos-87 manual*. Biomos BV, Nijenborgh 4, 9747 AG Groningen, The Netherlands, 1987.

[79] P.M. Morse, "Diatomic molecules according to the wave mechanics. II. vibrational levels." *Phys. Rev.*, **34** 57–64 (1929).

[80] H.J.C. Berendsen, J.P.M. Postma, W.F. van Gunsteren, and J. Hermans, "Interaction models for water in relation to protein hydration"; pp. 331–342 in *Intermolecular forces*. Edited by B. Pullman. D. Reidel Publishing Company, Dordrecht, 1981.

[81] D.M. Ferguson, "Parametrization and evaluation of a flexible water model," *J. Comp. Chem.*, **16** 501–511 (1995).

[82] H.R. Warner Jr., "Kinetic theory and rheology of dilute suspensions of finitely extendible dumbbells," *Ind. Eng. Chem. Fundam.*, **11** [3] 379–387 (1972).

[83] M. Bulacu, N. Goga, W. Zhao, G. Rossi, L. Monticelli, X. Periole, D. Tieleman, and S. Marrink, "Improved angle potentials for coarse-grained molecular dynamics simulations," *J. Chem. Theory Comput.*, **9** [8] 3282-3292 (2013).

[84] B.R. Brooks, R.E. Bruccoleri, B.D. Olafson, D.J. States, S. Swaminathan, and M. Karplus, "CHARMM: A program for macromolecular energy, minimization, and dynamics calculation," *J. Comp. Chem.*, **4** 187–217 (1983).

[85] C.P. Lawrence and J.L. Skinner, "Flexible TIP4P model for molecular dynamics simulation of liquid water," *Chem. Phys. Lett.*, **372** 842–847 (2003).

[86] W.L. Jorgensen, D.S. Maxwell, and J. Tirado-Rives, "Development and testing of the oPLS all-atom force field on conformational energetics and properties of organic liquids," *J. Am. Chem. Soc.*, **118** 11225–11236 (1996).

[87] M.J. Robertson, J. Tirado-Rives, and W.L. Jorgensen, "Improved peptide and protein torsional energetics with the oPLS-aA force field," *J. Chem. Theory Comput.*, **11** 3499–3509 (2015).

[88] M. Bulacu and E. van der Giessen, "Effect of bending and torsion rigidity on self-diffusion in polymer melts: A molecular-dynamics study," *J. Chem. Phys.*, **123** [11] 114901 (2005).

[89] R.A. Scott and H. Scheraga, "Conformational analysis of macromolecules," *J. Chem. Phys.*, **44** 3054–3069 (1966).

[90] L. Pauling, *The nature of chemical bond*. Cornell University Press, Ithaca; New York, 1960.

[91] A.E. Torda, R.M. Scheek, and W.F. van Gunsteren, "Time-dependent distance restraints in molecular dynamics simulations," *Chem. Phys. Lett.*, **157** 289–294 (1989).

[92] B. Hess and R.M. Scheek, "Orientation restraints in molecular dynamics simulations using time and ensemble averaging," *J. Magn. Reson.*, **164** 19–27 (2003).

[93] P.E.M. Lopes, J. Huang, J. Shim, Y. Luo, H. Li, B. Roux, and J. MacKerell Alexander D., "Polarizable force field for peptides and proteins based on the classical drude oscillator," *J. Chem. Theory Comput*, **9** 5430–5449 (2013).

[94] H. Yu, T.W. Whitfield, E. Harder, G. Lamoureux, I. Vorobyov, V.M. Anisimov, A.D. MacKerell, Jr., and B. Roux, "Simulating Monovalent and Divalent Ions in Aqueous Solution Using a Drude Polarizable Force Field," *J. Chem. Theory Comput.*, **6** 774–786 (2010).

[95] B.T. Thole, "Molecular polarizabilities with a modified dipole interaction," *Chem. Phys.*, **59** 341–345 (1981).

[96] G. Lamoureux and B. Roux, "Modeling induced polarization with classical drude oscillators: Theory and molecular dynamics simulation algorithm," *J. Chem. Phys.*, **119** 3025–3039 (2003).

[97] G. Lamoureux, A.D. MacKerell, and B. Roux, "A simple polarizable model of water based on classical drude oscillators," *J. Chem. Phys.*, **119** 5185–5197 (2003).

[98] S.Y. Noskov, G. Lamoureux, and B. Roux, "Molecular dynamics study of hydration in ethanol-water mixtures using a polarizable force field," *J. Phys. Chem. B.*, **109** 6705–6713 (2005).

[99] W.F. van Gunsteren and A.E. Mark, "Validation of molecular dynamics simulations," *J. Chem. Phys.*, **108** 6109–6116 (1998).

[100] T.C. Beutler, A.E. Mark, R.C. van Schaik, P.R. Greber, and W.F. van Gunsteren, "Avoiding singularities and numerical instabilities in free energy calculations based on molecular simulations," *Chem. Phys. Lett.*, **222** 529–539 (1994).

[103] W.L. Jorgensen and J. Tirado-Rives, "The OPLS potential functions for proteins. energy minimizations for crystals of cyclic peptides and crambin," *J. Am. Chem. Soc.*, **110** 1657–1666 (1988).

[104] H.J.C. Berendsen and W.F. van Gunsteren, "Molecular dynamics simulations: Techniques and approaches"; pp. 475–500 in *Molecular liquids-dynamics and interactions*. Edited by A.J.B. et al. Reidel, Dordrecht, The Netherlands, 1984.

[105] P.P. Ewald, "Die Berechnung optischer und elektrostatischer Gitterpotentiale," *Ann. Phys.*, **64** 253–287 (1921).

[106] R.W. Hockney and J.W. Eastwood, *Computer simulation using particles*. McGraw-Hill, New York, 1981.

[107] V. Ballenegger, J.J. Cerdà, and C. Holm, "How to convert SPME to P3M: Influence functions and error estimates," *J. Chem. Theory Comput.*, **8** [3] 936–947 (2012).

[108] M.P. Allen and D.J. Tildesley, *Computer simulations of liquids*. Oxford Science Publications, Oxford, 1987.

[109] C.L. Wennberg, T. Murtola, B. Hess, and E. Lindahl, "Lennard-Jones Lattice Summation in Bilayer Simulations Has Critical Effects on Surface Tension and Lipid Properties," *J. Chem. Theory Comput.*, **9** 3527–3537 (2013).

[110] C. Oostenbrink, A. Villa, A.E. Mark, and W.F. Van Gunsteren, "A biomolecular force field based on the free enthalpy of hydration and solvation: The GROMOS force-field parameter sets 53A5 and 53A6," *Journal of Computational Chemistry*, **25** [13] 1656–1676 (2004).

[111] W.D. Cornell, P. Cieplak, C.I. Bayly, I.R. Gould, K.R. Merz Jr., D.M. Ferguson, D.C. Spellmeyer, and T. Fox *et al.*, "A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules," *J. Am. Chem. Soc.*, **117** [19] 5179–5197 (1995).

[112] P.A. Kollman, "Advances and Continuing Challenges in Achieving Realistic and Predictive Simulations of the Properties of Organic and Biological Molecules," *Acc. Chem. Res.*, **29** [10] 461–469 (1996).

[113] J. Wang, P. Cieplak, and P.A. Kollman, "How Well Does a Restrained Electrostatic Potential (RESP) Model Perform in Calculating Conformational Energies of Organic and Biological Molecules?" *J. Comp. Chem.*, **21** [12] 1049–1074 (2000).

[114] V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg, and C. Simmerling, "Comparison of Multiple Amber Force Fields and Development of Improved Protein Backbone Parameters," *PROTEINS:*

*Struct. Funct. Gen.*, **65** 712–725 (2006).

[115] K. Lindorff-Larsen, S. Piana, K. Palmo, P. Maragakis, J.L. Klepeis, R.O. Dorr, and D.E. Shaw, "Improved side-chain torsion potentials for the AMBER ff99SB protein force field," *PROTEINS: Struct. Funct. Gen.*, **78** 1950–1958 (2010).

[116] Y. Duan, C. Wu, S. Chowdhury, M.C. Lee, G. Xiong, W. Zhang, R. Yang, and P. Cieplak *et al.*, "A Point-Charge Force Field for Molecular Mechanics Simulations of Proteins Based on Condensed-Phase Quantum Mechanical Calculations," *J. Comp. Chem.*, **24** [16] 1999–2012 (2003).

[117] A.E. García and K.Y. Sanbonmatsu, "$\alpha$-Helical stabilization by side chain shielding of backbone hydrogen bonds," *Proc. Natl. Acad. Sci. USA*, **99** [5] 2782–2787 (2002).

[118] J. MacKerell A. D., M. Feig, and C.L. Brooks III, "Extending the treatment of backbone energetics in protein force fields: Limitations of gas-phase quantum mechanics in reproducing protein conformational distributions in molecular dynamics simulations," *J. Comp. Chem.*, **25** [11] 1400–15 (2004).

[119] A.D. MacKerell, D. Bashford, Bellott, R.L. Dunbrack, J.D. Evanseck, M.J. Field, S. Fischer, and J. Gao *et al.*, "All-atom empirical potential for molecular modeling and dynamics studies of proteins," *J. Phys. Chem. B.*, **102** [18] 3586–3616 (1998).

[120] S.E. Feller and A.D. MacKerell, "An improved empirical potential energy function for molecular simulations of phospholipids," *J. Phys. Chem. B.*, **104** [31] 7510–7515 (2000).

[121] N. Foloppe and A.D. MacKerell, "All-atom empirical force field for nucleic acids: I. Parameter optimization based on small molecule and condensed phase macromolecular target data," *J. Comp. Chem.*, **21** [2] 86–104 (2000).

[122] A.D. MacKerell and N.K. Banavali, "All-atom empirical force field for nucleic acids: II. application to molecular dynamics simulations of DNA and RNA in solution," *J. Comp. Chem.*, **21** [2] 105–120 (2000).

[123] P. Larsson and E. Lindahl, "A High-Performance Parallel-Generalized Born Implementation Enabled by Tabulated Interaction Rescaling," *J. Comp. Chem.*, **31** [14] 2593–2600 (2010).

[124] P. Bjelkmar, P. Larsson, M.A. Cuendet, B. Hess, and E. Lindahl, "Implementation of the CHARMM force field in GROMACS: Analysis of protein stability effects from correction maps, virtual interaction sites, and water models," *J. Chem. Theory Comput.*, **6** 459–466 (2010).

[125] A. Kohlmeyer and J. Vermaas, *TopoTools: Release 1.6 with CHARMM export in topogromacs*, (2016).

[126] T. Bereau, Z.-J. Wang, and M. Deserno, *Solvent-free coarse-grained model for unbiased high-resolution protein-lipid interactions*, (n.d.).

[127] Z.-J. Wang and M. Deserno, "A systematically coarse-grained solvent-free model for quantitative phospholipid bilayer simulations," *J. Phys. Chem. B.*, **114** [34] 11207–11220 (2010).

[128] W.L. Jorgensen, J. Chandrasekhar, J.D. Madura, R.W. Impey, and M.L. Klein, "Comparison of simple potential functions for simulating liquid water," *J. Chem. Phys.*, **79** 926–935 (1983).

[129] IUPAC-IUB Commission on Biochemical Nomenclature, "Abbreviations and Symbols for the Description of the Conformation of Polypeptide Chains. Tentative Rules (1969)," *Biochemistry*, **9** 3471–3478 (1970).

[130] M.W. Mahoney and W.L. Jorgensen, "A five-site model for liquid water and the reproduction of the density anomaly by rigid, nonpolarizable potential functions," *J. Chem. Phys.*, **112** 8910–8922 (2000).

[131] J.P. Ryckaert and A. Bellemans, "Molecular dynamics of liquid alkanes," *Far. Disc. Chem. Soc.*, **66** 95–106 (1978).

[132] H. de Loof, L. Nilsson, and R. Rigler, "Molecular dynamics simulations of galanin in aqueous and nonaqueous solution," *J. Am. Chem. Soc.*, **114** 4028–4035 (1992).

[133] A.R. van Buuren and H.J.C. Berendsen, "Molecular Dynamics simulation of the stability of a 22 residue alpha-helix in water and 30% trifluoroethanol," *Biopolymers*, **33** 1159–1166 (1993).

[134] R.M. Neumann, "Entropic approach to Brownian Movement," *Am. J. Phys.*, **48** 354–357 (1980).

[135] C. Jarzynski, "Nonequilibrium equality for free energy differences," *Phys. Rev. Lett.*, **78** [14] 2690–2693 ().

[136] M.S. O. Engin A. Villa and B. Hess, "Driving forces for adsorption of amphiphilic peptides to air-water interface," *J. Phys. Chem. B.*, (2010).

[137] V. Lindahl, J. Lidmar, and B. Hess, "Accelerated weight histogram method for exploring free energy landscapes," *The Journal of chemical physics*, **141** [4] 044110 (2014).

[138] F. Wang and D. Landau, "Efficient, multiple-range random walk algorithm to calculate the density of states," *Physical review letters*, **86** [10] 2050 (2001).

[139] T. Huber, A.E. Torda, and W.F. van Gunsteren, "Local elevation: A method for improving the searching properties of molecular dynamics simulation," *Journal of computer-aided molecular design*, **8** [6] 695–708 (1994).

[140] A. Laio and M. Parrinello, "Escaping free-energy minima," *Proceedings of the National Academy of Sciences*, **99** [20] 12562–12566 (2002).

[141] R. Belardinelli and V. Pereyra, "Fast algorithm to calculate density of states," *Physical Review E*, **75** [4] 046701 (2007).

[142] A. Barducci, G. Bussi, and M. Parrinello, "Well-tempered metadynamics: A smoothly converging and tunable free-energy method," *Physical review letters*, **100** [2] 020603 (2008).

[143] V. Lindahl, A. Villa, and B. Hess, "Sequence dependency of canonical base pair opening in the dNA double helix," *PLoS computational biology*, **13** [4] e1005463 (2017).

[144] D.A. Sivak and G.E. Crooks, "Thermodynamic metrics and optimal paths," *Physical review letters*, **108** [19] 190602 (2012).

[145] C. Kutzner, J. Czub, and H. Grubmüller, "Keep it flexible: Driving macromolecular rotary motions in atomistic simulations with GROMACS," *J. Chem. Theory Comput.*, **7** 1381–1393 (2011).

[146] C. Caleman and D. van der Spoel, "Picosecond Melting of Ice by an Infrared Laser Pulse - A simulation study," *Angew. Chem., Int. Ed. Engl.*, **47** 1417–1420 (2008).

[147] C. Kutzner, H. Grubmüller, B.L. de Groot, and U. Zachariae, "Computational electrophysiology: The molecular dynamics of ion channel permeation and selectivity in atomistic detail," *Biophys. J.*, **101** 809–817 (2011).

[148] K.A. Feenstra, B. Hess, and H.J.C. Berendsen, "Improving efficiency of large time-scale molecular dynamics simulations of hydrogen-rich systems," *J. Comp. Chem.*, **20** 786–798 (1999).

[149] B. Hess, "Determining the shear viscosity of model liquids from molecular dynamics," *J. Chem. Phys.*, **116** 209–217 (2002).

[150] M.J.S. Dewar, "Development and status of MINDO/3 and MNDO," *J. Mol. Struct.*, **100** 41 (1983).

[151] M.F. Guest, R.J. Harrison, J.H. van Lenthe, and L.C.H. van Corler, "Computational chemistry on the FPS-X64 scientific computers - Experience on single- and multi-processor systems," *Theor. Chim. Act.*, **71** 117 (1987).

[152] M.J. Frisch, G.W. Trucks, H.B. Schlegel, G.E. Scuseria, M.A. Robb, J.R. Cheeseman, J.A. Montgomery Jr., and T. Vreven *et al.*, *Gaussian 03, Revision C.02*, (n.d.).

[153] R. Car and M. Parrinello, "Unified approach for molecular dynamics and density-functional theory," *Phys. Rev. Lett.*, **55** 2471–2474 (1985).

[154] M. Field, P.A. Bash, and M. Karplus, "A combined quantum mechanical and molecular mechanical potential for molecular dynamics simulation," *J. Comp. Chem.*, **11** 700 (1990).

[155] F. Maseras and K. Morokuma, "IMOMM: A New Ab Initio + Molecular Mechanics Geometry Optimization Scheme of Equilibrium Structures and Transition States," *J. Comp. Chem.*, **16** 1170–1179 (1995).

[156] M. Svensson, S. Humbel, R.D.J. Froes, T. Matsubara, S. Sieber, and K. Morokuma, "ONIOM a multilayered integrated MO + MM method for geometry optimizations and single point energy predictions. a test for Diels-Alder reactions and Pt(P(t-Bu)3)2 + H2 oxidative addition," *J. Phys. Chem.*, **100** 19357 (1996).

[157] S. Yesylevskyy, "ProtSqueeze: Simple and effective automated tool for setting up membrane protein simulations," *J. Chem. Inf. Model.*, **47** 1986–1994 (2007).

[158] M. Wolf, M. Hoefling, C. Aponte-Santamaría, H. Grubmüller, and G. Groenhof, "g_membed: Efficient insertion of a membrane protein into an equilibrated lipid bilayer with minimal perturbation," *J. Comp. Chem.*, **31** 2169–2174 (2010).

[159] D. van der Spoel and H.J.C. Berendsen, "Molecular dynamics simulations of Leu-enkephalin in water and DMSO," *Biophys. J.*, **72** 2032–2041 (1997).

[160] P.E. Smith and W.F. van Gunsteren, "The Viscosity of SPC and SPC/E Water," *Chem. Phys. Lett.*, **215** 315–318 (1993).

[161] S. Balasubramanian, C.J. Mundy, and M.L. Klein, "Shear viscosity of polar fluids: Molecular dynamics calculations of water," *J. Chem. Phys.*, **105** 11190–11195 (1996).

[162] J. Wuttke, *Lmfit*, (2013).

[163] B. Steen-Sæthre, A.C. Hoffmann, and D. van der Spoel, "Order parameters and algorithmic approaches for detection and demarcation of interfaces in hydrate-fluid and ice-fluid systems," *J. Chem. Theor. Comput.*, **10** 5606–5615 (2014).

[164] B.J. Palmer, "Transverse-current autocorrelation-function calculations of the shear viscosity for molecular liquids." *Phys. Rev. E*, **49** 359–366 (1994).

[165] E.J.W. Wensink, A.C. Hoffmann, P.J. van Maaren, and D. van der Spoel, "Dynamic properties of water/alcohol mixtures studied by computer simulation," *J. Chem. Phys.*, **119** 7308–7317 (2003).

[166] G.-J. Guo, Y.-G. Zhang, K. Refson, and Y.-J. Zhao, "Viscosity and stress autocorrelation function in supercooled water: A molecular dynamics study," *Mol. Phys.*, **100** 2617–2627 (2002).

[167] G.S. Fanourgakis, J.S. Medina, and R. Prosmiti, "Determining the bulk viscosity of rigid water models," *J. Phys. Chem. A*, **116** 2564–2570 (2012).

[168] D. van der Spoel, H.J. Vogel, and H.J.C. Berendsen, "Molecular dynamics simulations of N-terminal peptides from a nucleotide binding protein," *PROTEINS: Struct. Funct. Gen.*, **24** 450–466 (1996).

[169] A. Amadei, A.B.M. Linssen, and H.J.C. Berendsen, "Essential dynamics of proteins," *PROTEINS: Struct. Funct. Gen.*, **17** 412–425 (1993).

[170] B. Hess, "Convergence of sampling in protein simulations," *Phys. Rev. **E**, **65** 031910 (2002).

[171] B. Hess, "Similarities between principal components of protein dynamics and random diffusion," *Phys. Rev. **E**", **62** 8438–8448 (2000).

[172] Y. Mu, P.H. Nguyen, and G. Stock, "Energy landscape of a small peptide revealed by dihedral angle principal component analysis," *PROTEINS: Struct. Funct. Gen.*, **58** 45–52 (2005).

[173] D. van der Spoel, P.J. van Maaren, P. Larsson, and N. Timneanu, "Thermodynamics of hydrogen bonding in hydrophilic and hydrophobic media," *J. Phys. Chem. B.*, **110** 4393–4398 (2006).

[174] A. Luzar and D. Chandler, "Hydrogen-bond kinetics in liquid water," *Nature*, **379** 55–57 (1996).

[175] A. Luzar, "Resolving the hydrogen bond dynamics conundrum," *J. Chem. Phys.*, **113** 10663–10675 (2000).

[176] W. Kabsch and C. Sander, "Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features," *Biopolymers*, **22** 2577–2637 (1983).

[177] H. Bekker, H.J.C. Berendsen, E.J. Dijkstra, S. Achterop, R. v. Drunen, D. v. d. Spoel, A. Sijbers, and H. Keegstra *et al.*, "Gromacs Method of Virial Calculation Using a Single Sum"; pp. 257–261 in *Physics computing 92*. Edited by R.A. de Groot and J. Nadrchal. World Scientific, Singapore, 1993.

[178] H.J.C. Berendsen, J.R. Grigera, and T.P. Straatsma, "The missing term in effective pair potentials," *J. Phys. Chem.*, **91** 6269–6271 (1987).

[179] W.F. van Gunsteren and H.J.C. Berendsen, *Molecular dynamics of simple systems*, (1994).

[180] A. Laio, J. VandeVondele, U. Rothlisberger, *A Hamiltonian electrostatic coupling scheme for hybrid Car-Parrinello molecular dynamics simulations*, (2002).

[181] Hub, J. S., de Groot, B. L., Grubmüller, H., Groenhof, G., "Quantifying artifacts in Ewald simulations of inhomogeneous systems with a net charge," *J. Chem. Theory Comput.*, **10**, 381–390 (2014).

[182] Páll, S., Hess, B., "A flexible algorithm for calculating pair interactions on SIMD architectures," *Comput. Phys. Commun.*, **183**, 2641–2650 (2013).

[183] V. Gapsys, D. Seeliger, and B.L. de Groot, "New Soft-Core Potential Function for Molecular Dynamics Based Alchemical Free Energy Calculations", *J. Chem. Theor. Comput.*, **8** 2373-2382 (2012).

[184] Bernetti, M. and Bussi G., "Pressure control using stochastic cell rescaling", *J. Chem. Phys.*, **153**, 114107 (2020).

[185] Lidmar J., "Improving the efficiency of extended ensemble simulations: The accelerated weight histogram method", *Phys. Rev. E*, **85**, 0256708 (2012).

[186] Lindahl V., Lidmar J. and Hess B., "Riemann metric approach to optimal sampling of multidimensional free-energy landscapes", *Phys. Rev. E*, **98**, 023312 (2018).

[187] Lundborg M., Lidmar J. and Hess B., "The accelerated weight histogram method for alchemical free energy calculations", *J. Chem. Phys.*, **154**, 204103 (2021).

[188] Kühne T., Iannuzzi M., Del Ben M. and Hutter J. *et al.*, "CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations", *J. Chem. Phys.*, **152**, 194103 (2020).

[189] Laino T., Mohamed F., Laio A. and Parrinello M., "An Efficient Real Space Multigrid QM/MM Electrostatic Coupling", *J. Chem. Theory Comput.*, **1**, 1176 (2005).

[190] D. van der Spoel, H. Henschel, P. J. van Maaren, M. M. Ghahremanpour , and L. T. Costa, "A potential for molecular simulation of compounds with linear moieties", *J. Chem. Phys.*, **153** 084503 (2020).

[191] M. Tuckerman, B. J. Berne, and G. J. Martyna, "Reversible multiple time scale molecular dynamics", *J. Chem. Phys.*, **97** 1990 (1992).

[192] Orzechowski M, Tama F., "Flexible fitting of high-resolution x-ray structures into cryoelectron microscopy maps using biased molecular dynamics simulations", *Biophysical journal*, *95*, 5692–705, (2008).

[193] Igaev, M., Kutzner, C., Bock, L. V., Vaiana, A. C., & Grubmüller, H., "Automated cryo-EM structure refinement using correlation-driven molecular dynamics", *eLife*, **8**, e43542 (2019).

[194] Lundborg M., Lidmar J. and Hess B., "On the Path to Optimal Alchemistry", *Protein J.*, **42**, 477–489 (2023).

[195] G. Fiorin, M. L. Klein, and J. Hénin, "Using collective variables to drive molecular dynamics simulations", *Mol. Phys.* **111** 3345-3362 (2013).

[196] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations", *J. Comput. Phys.* **73**, 325–348 (1987).

[197] Maier, J. A., Martinez, C., Kasavajhala, K., Wickstrom, L., Hauser, K. E., and Simmerling, C. , and Simmerling, C., "ff14SB: Improving the Accuracy of Protein Side Chain and Backbone Parameters from ff99SB", *J. Chem. Theory Comput.* **11**, 3696–3713 (2015).

[198] Tian, C., Kasavajhala, K., Belfon, K. A. A., Raguette, L., Huang, H., Migues, A. N., Bickel, J., Wang, Y., Pincay, J., Wu, Q., and Simmerling, C., "ff19SB: Amino-Acid-Specific Protein Backbone Parameters Trained against Quantum Mechanics Energy Surfaces in Solution", *J. Chem. Theory Comput.* **16**, 528–552 (2020).

[199] Miletić, V., Wieczór, M., Simmerling, C., Rampp, M., De Groot, B. L., Kutzner, C. & Gapsys, V. Force Fields and Simulation Engines: Lessons Learned from Porting Amber ff14SB and ff19SB to GROMACS. Preprint at doi:10.26434/chemrxiv.15006112/v1 (2026).

[200] In-Chul, Y., and , Berkowitz, M. L. "Ewald summation for systems with slab geometry", *J. Chem. Phys.* **111**, 3155-3162 (1999).

[201] D. Morozov, C. Blau, O. Schütt, V. Mironov, A. Proeme, H. Judge, T. D. Kühne, B. Hess, and G. Groenhof, "Seamless QM/MM Simulations via a GROMACS-CP2K Interface", *Journal of Chemical Information and Modeling* (2026). doi:10.1021/acs.jcim.6c02073.

CHAPTER

SIX

# GMXAPI PYTHON PACKAGE

Version 0.5.0a1.

*gmxapi* (page 718) allows molecular simulation and analysis work to be staged and run from Python.

After installing GROMACS and the gmxapi Python package, you can use **pydoc gmxapi** from the command line or `import gmxapi; help(gmxapi)` within a Python interpreter for concise usage help. Complete documentation is collected in the following sections.

## 6.1 Full installation instructions

Installation instructions for the *gmxapi* (page 718) Python package, built on GROMACS.

Command line examples assume the bash shell.

**Regarding multiple GROMACS installations**

Many GROMACS users switch between multiple GROMACS installations on the same computer using an HPC module system and/or a *GMXRC* (page 30) configuration script. For the equivalent sort of environment switching with the *gmxapi* (page 718) Python package, we recommend installing it in a different Python virtual environment for each GROMACS installation. Once built, a particular copy of the *gmxapi* (page 718) Python package always refers to the same GROMACS installation.

**Unprivileged `pip install`**

The following documentation contains frequent references to the pip tool for installing Python packages. In some cases, an unprivileged user should use the `--user` command line flag to tell pip to install packages into the user site-packages directory rather than the default site-packages directory for the Python installation. This flag is not appropriate when running **pip** in a virtual environment (as recommended) and is omitted in this documentation. If you need the `--user` flag, you should modify the example commands to look something like **pip install --upgrade somepackage --user**

**Python 3 executable names**

These instructions use the executable names **python** and **pip** instead of **python3** or **pip3**. Some Python installations require the `3` suffix, but it is usually not necessary if you have already activated

a Python virtual environment (recommended).

### 6.1.1 Overview

Typically, setting up the *gmxapi* Python package follows these three steps. If this overview is sufficient for your computing environment, you may disregard the rest of this document.

#### Install GROMACS

Locate your GROMACS installation, or build and install. GROMACS 2022 or higher is recommended.

**See also**

GROMACS installation

The following assumes GROMACS is installed to `/path/to/gromacs`

#### Set up a Python virtual environment

**See also**

*Set up a Python virtual environment* (page 700)

**Note**

`mpi4py` may require additional arguments (compiler hints). See *MPI requirements* (page 698)

```
python3 -m venv $HOME/myvenv
. $HOME/myvenv/bin/activate
python -m ensurepip --default-pip
pip install --upgrade pip setuptools wheel
pip install mpi4py
```

#### Install the gmxapi Python package

Pull the `gmxapi` package from PyPI, build it for the GROMACS installation at `/path/to/gromacs`, and install it to the Python prefix for the current environment.

```
. /path/to/gromacs/bin/GMXRC
pip install --no-cache-dir gmxapi
```

**See also**

*Installing the Python package* (page 699)

### 6.1.2 Background

*gmxapi* comes in three parts:

- GROMACS gmxapi library for C++.
- This Python package, supporting Python 3.9 and higher
- MD restraint plugins and sample gmxapi client code

#### GROMACS requirements

The Python package requires a GROMACS installation. Locate an existing GROMACS installation, or build and install GROMACS before proceeding.

**Note**

Note that gmxapi requires that GROMACS is configured with `GMXAPI=ON` and `BUILD_SHARED_LIBS=ON`. These are enabled by default in most cases. If these options were overridden for your GROMACS installation, you will see CMake errors when trying to build and install the gmxapi Python package or other client software.

If your installation has a `GMXRC` file, "source" the file *as you normally would* (page 30) before using GROMACS. Otherwise, note the installation location so that you can provide it when building the gmxapi package.

#### Build system requirements

gmxapi can be built for Python 3.9 and higher.

You will need a C++ 17 compatible compiler and a reasonably up-to-date version of CMake. Full gmxapi functionality may also require an MPI compiler (e.g. **`mpicc`**).

Important: To build a module that can be imported by Python, you need a Python installation that includes the Python headers. Unfortunately, it is not always obvious whether these headers are present or where to find them. The simplest answer is to just try to build the Python package using these instructions, and if gmxapi is unable to find the Python tools it needs, try a different Python installation or install the additional development packages.

On a Linux system, this may require installing packages such as `python-dev` and/or `python3-dev`. If you are building Python, either from scratch or with a tool like **`pyenv install`** (see wiki entry ), be sure to enable installation of the Python C library with the `--enable-shared` flag. Alternatively, various Python distributions provide a sufficient build environment while only requiring installation into a user home directory. (Some examples below.)

If you are using an HPC system with software available through modules you may be able to just **`module load`** a different Python installation and find one that works.

### Python environment requirements

gmxapi requires Python 3.9 or higher. Check your version with **`python3 --version`** or **`python --version`**.

> **Note**
>
> The following documentation assumes you do not need to use a trailing '3' to access a Python 3 interpreter on your system. The default Python interpreter on your system may use **`python3`** and **`pip3`** instead of **`python`** and **`pip`**. You can check the version with **`python3 --version`** or **`python --version`** and **`pip --version`**.

To build and install, you need the Python packages for cmake, networkx, and setuptools (all available from PyPI with pip).

For full functionality, you should also have mpi4py and numpy. These requirements and version numbers are listed in `requirements.txt`.

The easiest way to make sure you have the requirements installed, first update pip, then use the `requirements.txt` file provided with the repository. File paths in this section are relative to the root directory of your local copy of the GROMACS source.

Confirm that pip is available, install pip if it is missing, or get instructions on how to install pip:

```
python -m ensurepip --default-pip
```

Install or upgrade required components:

```
python -m pip install --upgrade pip
pip install --upgrade setuptools wheel
```

### "requirements" files in GROMACS source tree

If you are building from source code in a local copy of the GROMACS source repository, a `requirements.txt` allows you to preinstall the Python requirements before installing the *gmxapi* (page 718) package.

```
pip install -r python_packaging/gmxapi/requirements.txt
```

### Documentation build requirements

See *Accessing gmxapi documentation* (page 705)

### Testing requirements

Note that the test suite is only available in the GROMACS source tree. (It is not part of the installed package.) Acquire the GROMACS sources with **`git`** or by downloading an archive, as documented elsewhere.

Testing is performed with pytest.

`python_packaging/gmxapi/requirements.txt` lists additional requirements for testing. With pip:

```
pip install -r python_packaging/gmxapi/requirements.txt
```

To test the full functionality also requires an MPI parallel environment. You will need the mpi4py Python package and an MPI launcher (such as **`mpiexec`**, **`mpirun`**, a launcher provided by your HPC queuing system, or whatever is provided by your favorite MPI package for your operating system).

### MPI requirements

For the ensemble simulations features, you will need an MPI installation.

On an HPC system, this means you will probably have to use **`module load`** to load a compatible set of MPI tools and compilers. Check your HPC documentation or try **`module avail`** to look for an `openmpi`, `mpich`, or `mvapich` module and matching compiler module. This may be as simple as:

```
module load gcc
module load mpicc
```

If you are using a GROMACS installation that is already available through `module load`, try to find a Python installation with the `mpi4py` package that is also available through `module load`. The *module* system will generally enforce toolchain compatibility between the loaded modules. If you `module load` mpi4py or a Python installation with mpi4py, you will probably want to use this version of the package in your venv. (See *Set up a Python virtual environment* (page 700)) If you `module load` an MPI-enabled GROMACS installation, `gmxapi` will try to check `mpi4py` for compatibility.

Note that the compilers loaded might not be the first compilers discovered automatically by the build tools we will use below, so you may have to specify compilers on the command line for consistency. It may be necessary to require that GROMACS, gmxapi, and the sample code are built with the same compiler(s).

Note that strange errors have been known to occur when mpi4py is built with a different tool set than has been used to build Python and gmxapi. If the default compilers on your system are not sufficient for GROMACS or gmxapi, you may need to build, e.g., OpenMPI or MPICH, and/or build mpi4py with a specific MPI compiler wrapper. This can complicate building in environments such as Conda. You should be able to confirm that your MPI compiler wrapper is consistent with your GROMACS tool chain by comparing the output of **`mpicc --version`** with the compiler information reported by **`gmx --version`**.

Set the `MPICC` environment variable to the MPI compiler wrapper and forcibly reinstall mpi4py:

```
export MPICC=`which mpicc`
pip install --no-cache-dir --upgrade --no-binary ":all:" --force-
↪reinstall mpi4py
```

If you have a different MPI C compiler wrapper, substitute it for **`mpicc`** above.

While `gmxapi` is configuring its build system during installation, it will try to confirm the compatibility of the `mpi4py` toolchain with that of the GROMACS installation. If they appear incompatible, you should see a `CMake` message that includes a guess at what you might try using for `MPICC`. (If using `pip`, consider using the `--verbose` option for more build output.)

### 6.1.3 Installing the Python package

We recommend using Python's pip package installer to automatically download, build, and install the latest version of the gmxapi package into a Python virtual environment, though it is also possible to install without a virtual environment. If installing without a virtual environment as an un-privileged user, you may need to use the `--user` option with **`pip install`**.

#### Recommended installation

The instructions in this section assume that *pip* is able to download files from the internet. Alternatively, refer to *Offline install* (page 703).

#### Locate or install GROMACS

You need a GROMACS installation that includes the gmxapi headers and library.

> **Warning**
>
> gmxapi does not recognize multiple GROMACS installations to the same `CMAKE_INSTALL_PREFIX`.
>
> The Python package uses files installed to `.../share/cmake/gmxapi/` to configure its C++ component. These configuration files are overwritten when installing GROMACS to the same CMAKE_INSTALL_PREFIX. Overlapping GROMACS installations may occur when GROMACS is installed for multiple configurations of MPI support and floating point precision. (See Issue 4334 and related issues.)

If GROMACS 2020 or higher is already installed, *and* was configured with `GMXAPI=ON` at build time (the default), you may be able to just source the *GMXRC* (page 30) (so that the Python package knows where to find GROMACS) and skip to the next section. Note that some GROMACS installations, such as in high-performance computing environments, may not install a `GMXRC`, and may instead provide access to the GROMACS installation through a **`module load gromacs`** or similar command.

If necessary, install a supported version of GROMACS. When building GROMACS from source, be sure to configure cmake with the flag `-DGMXAPI=ON` (default).

Set the environment variables for the GROMACS installation so that the gmxapi headers and library can be found when building the Python package. If you installed to a `gromacs-gmxapi` directory in your home directory as above and you use the **`bash`** shell, do:

```
source $HOME/gromacs-gmxapi/bin/GMXRC
```

If you are using a GROMACS installation that does not provide `GMXRC`, see *gmxapi cmake hints* (page 702) and additional CMake hints below.

### Set up a Python virtual environment

We recommend installing the Python package in a virtual environment. If not installing in a virtual environment, you may not be able to install necessary prerequisites (e.g. if you are not an administrator of the system you are on).

The following instructions use the `venv` module. Alternative virtual environments, such as Conda, should work fine, but are beyond the scope of this document. (We welcome contributed recipes!)

Depending on your computing environment, the Python 3 interpreter may be accessed with the command **`python`** or **`python3`**. Use **`python --version`** and **`python3 --version`** to figure out which you need to use. The following assumes the Python 3 interpreter is accessed with **`python3`**.

**–system-site-packages**

It can be tricky to properly or optimally build MPI enabled software in computing clusters, and administrators often provide prebuilt packages like `mpi4py`. If your computing environment has multiple Python installations, try to choose one that already includes `mpi4py`. When you are using a Python installation that provides `mpi4py`, generally, you should be sure to use the existing `mpi4py` installation in your new virtual environment by creating the `venv` with the `--system-site-packages` option.

In personal computing environments (laptops and workstations), it is common to have multiple Python installations, and it can be hard to keep packages in the different installations from conflicting with each other. Unless you know that you want to inherit the `mpi4py` package from the system installation, it is generally cleaner *not* to inherit the system site-packages.

Create a Python 3 virtual environment:

```
python3 -m venv $HOME/myvenv
```

*or* (see note):

```
python3 -m venv --system-site-packages $HOME/myvenv
```

Activate the virtual environment. Your shell prompt will probably be updated with the name of the environment you created to make it more obvious.

```
$ source $HOME/myvenv/bin/activate
(myvenv) $
```

**Note**

After activating the *venv*, **`python`** and **`pip`** are sufficient. (The '3' suffix will no longer be necessary and will be omitted in the rest of this document.)

Activating the virtual environment may change your shell prompt to indicate the environment is active. The prompt is omitted from the remaining examples, but the remaining examples assume the virtual environment is still active. (Don't do it now, but you can deactivate the environment by running **`deactivate`**.)

### Install dependencies

It is always a good idea to update pip, setuptools, and wheel before installing new Python packages:

```
pip install --upgrade pip setuptools wheel
```

The gmxapi installer requires a few additional packages. It is best to make sure they are installed and up to date before proceeding.

```
pip install --upgrade cmake pybind11
```

We use mpi4py for some features and to ensure compatible MPI bindings throughout your Python environment. **If you did not inherit mpi4py from system site-packages** (see *above* (page 700)), make sure to install it using the same MPI installation that we are building GROMACS against, and build with compatible compilers.

```
MPICC=`which mpicc` pip install --no-cache-dir --upgrade mpi4py
```

**See also**

*MPI requirements* (page 698)

### Install the latest version of gmxapi

Fetch and install the latest official version of gmxapi from the Python Packaging Index. Avoid locally cached previously-built packages that may be incompatible with your current environment or GROMACS installation:

```
# Get the latest official release.
pip install --no-cache-dir gmxapi
```

or:

```
pip download gmxapi
pip install gmxapi-<version>.tar.gz
```

substituting the name of the downloaded source distribution archive.

**Avoid cached "wheel" packages.**

`pip` downloads a source distribution archive for gmxapi, then builds a "wheel" package for your GROMACS installation. This "wheel" normally gets cached, and will be used by any later attempt to `pip install gmxapi` instead of rebuilding. This is not what you want, if you upgrade GROMACS or if you want to install the Python package for a different GROMACS configuration (e.g. double-precision or different MPI option.)

You can use `--no-cache-dir` to force rebuild of the package and its build dependencies. This may be slow, however, and you may want to use cached dependencies. You can avoid wheel cache for just one target package by installing from the filesystem instead of directly from PyPI.

See also Issue 4335

The PyPI repository may include pre-release versions, but **pip** will ignore them unless you use the `--pre` flag:

```
# Get the latest version, including pre-release versions.
pip install --no-cache-dir --pre gmxapi
```

If **pip** does not find your GROMACS installation, use one of the following environment variables to provide a hint.

### CMake hints

The *gmxapi* (page 718) package is distributed with C++ source code that needs to be compiled against GROMACS libraries. The build system is configured using CMake, mediated by scikit-build-core. Refer to scikit-build-core documentation for the best information on passing build options through the Python package installer (*e.g.* pip). (Be sure to look at the "config-settings" and "Environment" tabs. The "pyproject.toml" tabs is for package maintainers.)

### gmxapi_ROOT

If you have a single GROMACS installation at `/path/to/gromacs`, it is usually sufficient to provide this location to **pip** through the `gmxapi_ROOT` environment variable, or as a CMake variable definition.

```
gmxapi_ROOT=/path/to/gromacs pip install --no-cache-dir gmxapi
```

```
pip install --no-cache-dir gmxapi --config-setting=cmake.define.
↪gmxapi_ROOT=/path/to/gromacs
```

### GROMACS CMake hints

It can be important to use the same compiler tool chain for both GROMACS and for client software (the Python package C++ extension).

You can check `gmx --version` to see what compilers your installation used, and make sure that you don't have incompatible compilers declared by environment variables such as `$CXX`.

You can also use the GROMACS provided CMake cache file to provide extra hints to the Python extension build system about the software tools that were used to build GROMACS. (For more information, read about the `-C` command line option for CMake.)

In the following example,

- `${UNIQUE_PREFIX}` is the path to the directory that holds the GROMACS `bin`, `lib`, `share` directories, *etc*. It is *unique* because GROMACS provides CMake support for only one build configuration at a time through `.../share/cmake/gmxapi/`, even if there are multiple library configurations installed to the same location. See Issue 4334.

- `${SUFFIX}` is the suffix (e.g. *_d*, *_mpi*, etcetera) that distinguishes the particular build of GROMACS you want to target (refer to GROMACS installation instructions for more information.) `${SUFFIX}` may simply be empty, or `''`.

```
pip install gmxapi \
  --config-settings=cmake.define.gmxapi_ROOT=${UNIQUE_PREFIX} \
  --config-settings=cmake.args=-C${UNIQUE_PREFIX}/share/cmake/
↪gromacs${SUFFIX}/gromacs-hints${SUFFIX}.cmake
```

In sufficiently new `pip` versions, `-C` is a shorter alternative to `--config-settings=`. Do not confuse the `-C` option to `pip` with the `-C` option to `cmake`.

**See also**

scikit-build-core config-settings

### Install from source

You can also install the *gmxapi* (page 718) Python package from within a local copy of the GROMACS source repository. Assuming you have already obtained the GROMACS source code and you are in the root directory of the source tree, you will find the *gmxapi* (page 718) Python package sources in the `python_packaging/gmxapi` directory.

```
cd python_packaging/gmxapi
pip install -r requirements.txt
pip install .
```

### Offline install

**Recommended, first**

**`pip install --upgrade build pip setuptools wheel`**

You can use **`python -m build --skip-dependency-check`** to build a binary distribution archive (from the source distribution) for just the *gmxapi* package, but then you will have to manually satisfy (separate) dependencies in both the build and installation environments.

While you have internet access, you need to get access to the *gmxapi* source distribution and the package dependencies. You will also want the `wheel` and `build` packages in environments where the package(s) will be built. Only `pip` is necessary once a gmxapi `wheel` is built.

The following instructions are paraphrased from https://pip.pypa.io/en/stable/user_guide/#installing-from-local-packages

To build with internet access and then install without:

```
# Remove any locally cached (previously built) wheels.
pip cache remove gmxapi
```

(continues on next page)

(continued from previous page)

```
# Download gmxapi and dependencies from pypi.
pip wheel --wheel-dir DIR gmxapi
# or, using package source from the GROMACS repository
cd python_packaging/gmxapi
pip wheel --wheel-dir DIR .

# Later, install.
pip install --no-index --find-links=DIR DIR/gmxapi*whl
```

To download packages and dependencies for later build and installation:

```
# if in the GROMACS source repository
cd python_packaging/gmxapi
# or download and expand the archive
pip download --destination-directory DIR gmxapi
tar xf DIR/gmxapi*
cd gmxapi*

# Pre-fetch dependencies to DIR
pip download --destination-directory DIR .

# Build and install from the source directory.
pip install --no-index --find-links=DIR .
```

### Building a source archive

A source archive for the gmxapi python package can be built from the GROMACS source repository using the Python build module.

Example:

```
pip install --upgrade setuptools build
cd python_packaging/gmxapi
python -m build --sdist
```

This command will create a `dist` directory containing a source distribution archive file. The file name has the form `gmxapi-`*`version`*`.`*`suffix`*, where *version* is the version from the package metadata, and *suffix* is an archive file extension determined by the local environment and the current packaging specifications.

The version information is derived from `gmxapi.__version__` defined by the *`gmxapi.version`* (page 728) module. Pending refinement under Issue 3851, the gmxapi version information is hard coded in the `version.py`. Make sure you have an up-to-date version of the sources and that the version information is appropriate before distributing a new release.

**See also**

Python documentation for creating a source distribution

Package maintainers may update the online repository by uploading a freshly built `sdist` with

`python -m twine upload dist/gmxapi-{version}.{suffix}`. To update the repository at the PyPI test server, use `python -m twine upload --repository testpypi dist/gmxapi-{version}.{suffix}`.

### 6.1.4 Accessing gmxapi documentation

Documentation for the Python classes and functions in the gmx module can be accessed in the usual ways, using `pydoc` from the command line or `help()` in an interactive Python session.

The complete documentation (which you are currently reading) can be browsed online or built from a copy of the GROMACS source repository.

Documentation is built from a combination of Python module documentation and static content, and requires a local copy of the GROMACS source repository.

#### Build with GROMACS

To build the full gmxapi documentation with GROMACS, configure GROMACS with `-DGMX_PYTHON_PACKAGE=ON` and build the GROMACS documentation normally. This will first build the *gmxapi* Python package and install it to a temporary location in the build tree. Sphinx can then import the package to automatically extract Python docstrings.

Note that this is an entirely CMake-driven installation and Python dependencies will not be installed automatically. You can update your Python environment (before configuring with CMake) using the `requirements.txt` files provided in the `python_packaging/` directory of the repository. Example:

```
pip install -r python_packaging/gmxapi/requirements.txt
```

Sometimes the build environment can choose a different Python interpreter than the one you intended. You can set the `Python3_ROOT_DIR` or `CMAKE_PREFIX_PATH` CMake variable to explicitly choose the Python installation or *venv* directory. See also CMake FindPython3.

If you use pyenv or pyenv-virtualenv to dynamically manage your Python version, you can help identify a particular version with `pyenv version-name` and the directory with `pyenv prefix {version}`. For example:

```
-DPython3_ROOT_DIR=$(pyenv prefix $(pyenv version-name))
```

### 6.1.5 Testing

Note *testing requirements* (page 698) above.

After installing the *gmxapi* (page 718) Python package, you can run the Python test suite from the GROMACS source tree. Example:

```
# Assuming you are in the root directory of the repository:
pytest python_packaging/gmxapi/test/
```

Refer to `python_packaging/README.md` for more detailed information.

### 6.1.6 Troubleshooting

#### ImportError at run time with dynamic linking error

Symptom: Python fails with a weird `ImportError` citing something like `dlopen`:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: dlopen(/.../gmxapi/_gmxapi.so, 0x0002): Symbol not␣
↪found:
__ZN12gmxapicompat11readTprFileERKNSt7__cxx1112basic_
↪stringIcSt11char_traitsIcESaIcEEE
  Referenced from: /.../gmxapi/_gmxapi.so
  Expected in: /path/to/gromacs/lib/libgmxapi_mpi_d.0.3.1.dylib
```

Inconsistencies in the build and run time environments can cause dynamic linking problems at run time. This could occur if you reinstall GROMACS built with a different compiler, or if `pip` or `CMake` somehow get tricked into using the wrong compiler tool chain.

Refer to the *gmxapi cmake hints* (page 702) for notes about compiler toolchains. Rebuild and reinstall the gmxapi Python package with `--no-cache-dir` and provide the `gromacs-hints.cmake` file for the GROMACS installation you intend to use.

#### AttributeError: module 'enum' has no attribute 'IntFlag'

If you had older versions of some of the dependencies installed, you might have picked up a transitive dependency on the `enum34` package. Try:

```
pip uninstall -y enum34
```

and see if that fixes the problem. If not, try a fresh virtual environment (see above) to help narrow down the problem before you open an issue.

#### Errors regarding pybind11

An error may occur in `setup.py` with output that contains something like the following:

```
  ModuleNotFoundError: No module named 'pybind11'
  Building wheel for gmxapi (pyproject.toml): finished with status
↪'error'
  ERROR: Failed building wheel for gmxapi
Failed to build gmxapi
ERROR: Could not build wheels for gmxapi, which is required to␣
↪install pyproject.toml-based projects
```

The important information here is that `pybind11` was not found.

Build dependencies aren't always automatically installed. Even if you are using `pip`, you may have disabled automatic dependency fulfillment with an option like `--no-build-isolation` or `--no-deps`.

In any case, the problem should be resolved by explicitly installing the `pybind11` Python package before attempting to build `gmxapi`:

```
pip install --upgrade pybind11
```

### Couldn't find the `gmxapi` support library?

If you don't want to "source" your *GMXRC* (page 30) file, you can tell the package where to find a gmxapi compatible GROMACS installation with `gmxapi_ROOT`. E.g. `gmxapi_ROOT=/path/to/gromacs pip install .`

Before updating the `gmxapi` package it is generally a good idea to remove the previous installation and to start with a fresh build directory. You should be able to just `pip uninstall gmxapi`.

Do you see something like the following?

```
CMake Error at gmx/core/CMakeLists.txt:45 (find_package):
  Could not find a package configuration file provided by "gmxapi"␣
↪with any
  of the following names:

    gmxapiConfig.cmake
    gmxapi-config.cmake

  Add the installation prefix of "gmxapi" to CMAKE_PREFIX_PATH or␣
↪set
  "gmxapi_ROOT" to a directory containing one of the above files. ␣
↪If "gmxapi"
  provides a separate development package or SDK, be sure it has␣
↪been
  installed.
```

This could be because

- GROMACS is not already installed
- GROMACS was built without the CMake variable `GMXAPI=ON`
- or if `gmxapi_ROOT` (or `GROMACS_DIR`) is not a path containing directories like `bin` and `share`.

If you are not a system administrator you are encouraged to install in a Python virtual environment, created with virtualenv or Conda. Otherwise, you will need to specify the `--user` flag to `pip`.

Two of the easiest problems to run into are incompatible compilers and incompatible Python. Try to make sure that you use the same C and C++ compilers for GROMACS, for the Python package, and for the sample plugin. These compilers should also correspond to the **`mpicc`** compiler wrapper used to compile mpi4py. In order to build the Python package, you will need the Python headers or development installation, which might not already be installed on the machine you are using. (If not, then you will get an error about missing `Python.h` at some point.) If you have multiple Python installations (or modules available on an HPC system), you could try one of the other Python installations, or you or a system administrator could install an appropriate Python dev package. Alternatively, you might try installing your own Anaconda or MiniConda in your home directory.

If an attempted installation fails with CMake errors about missing "gmxapi", make sure that GROMACS is installed and can be found during installation. For instance,

```
gmxapi_ROOT=/Users/eric/gromacs pip install --verbose gmxapi
```

Pip and related Python package management tools can be a little too flexible and ambiguous sometimes. If things get really messed up, try explicitly uninstalling the *gmxapi* (page 718) module and its dependencies, then do it again and repeat until **pip** can no longer find any version of any of the packages.

```
pip uninstall gmxapi
pip uninstall cmake
# ...
```

Successfully running the test suite is not essential to having a working *gmxapi* (page 718) package. We are working to make the testing more robust, but right now the test suite is a bit delicate and may not work right, even though you have a successfully built the *gmxapi* (page 718) package. If you want to troubleshoot, though, the main problems seem to be that automatic installation of required python packages may not work (requiring manual installations, such as with **pip install somepackage**) and ambiguities between python versions.

If you are working in a development branch of the repository, note that the upstream branch may be reset to `main` after a new release is tagged. In general, but particularly on the `devel` branch, when you do a **git pull**, you should use the `--rebase` flag.

If you fetch this repository and then see a git status like this:

```
$ git status
On branch devel
Your branch and 'origin/devel' have diverged,
and have 31 and 29 different commits each, respectively.
```

then *gmxapi* (page 718) has probably entered a new development cycle. You can do **git pull --rebase** to update to the latest development branch.

If you do a **git pull** while in `devel` and get a bunch of unexpected merge conflicts, do **git merge --abort; git pull --rebase** and you should be back on track.

If you are developing code for gmxapi, this should be an indication to rebase your feature branches for the new development cycle.

## 6.2 Using the Python package

After installing GROMACS, sourcing the "GMXRC" (see GROMACS docs), and installing the gmxapi Python package (see *Full installation instructions* (page 694)), import the package in a Python script or interactive interpreter. This documentation assumes a convenient alias of `gmx` to refer to the `gmxapi` Python package.

```
import gmxapi as gmx
```

For full documentation of the Python-level interface and API, use the `pydoc` command line tool or the `help()` interactive Python function, or refer to the *gmxapi Python module reference* (page 716).

Any Python *exception* raised by gmxapi should be descended from (and catchable as) *gmxapi.exceptions.Error* (page 727). Additional status messages can be acquired through the *Logging*

(page 715) facility. Unfortunately, some errors occurring in the GROMACS library are not yet recoverable at the Python level, and much of the standard GROMACS terminal output is not yet accessible through Python. If you find a particularly problematic scenario, please file a GROMACS bug report.

During installation, the *gmxapi* Python package becomes tied to a specific GROMACS installation. If you would like to access multiple GROMACS installations from Python, build and install *gmxapi* in separate *virtual environments* (page 700).

### 6.2.1 Notes on parallelism and MPI

The GROMACS library can be built for parallel computation using various strategies. If GROMACS was configured with `-DGMX_MPI=ON`, the same MPI library and compiler tool chain must be used for *gmxapi* and `mpi4py`. In any case, the Python package must be built with `mpi4py` installed. See *MPI requirements* (page 698).

> **Note**
>
> This section uses "mpiexec" generically to refer to the MPI program launcher. Depending on your MPI implementation and system details, your environment may use "mpirun", or some other command instead.

*gmxapi* scripts manage batches of simulations (as "ensembles") using MPI and `mpi4py`. To check whether your installed *gmxapi* package was built with MPI bindings, you can check for the `mpi_bindings` feature using *`gmxapi.version.has_feature()`* (page 729). The following command will produce an error if the feature is not available.

```
python -c 'import gmxapi; assert gmxapi.version.has_feature("mpi_
↪bindings")'
```

Assuming you use **`mpiexec`** to launch MPI jobs in your environment, run a *gmxapi* script on two ranks with something like the following. Note that it can be helpful to provide **`mpiexec`** with the full path to the intended Python interpreter since new process environments are being created.

```
mpiexec -n 2 `which python` -m mpi4py myscript.py
```

The `-m mpi4py` ensures that the `mpi4py` package is available and allows for proper clean-up of resources. (See `mpi4py.run` for details.)

#### Mapping ranks to ensemble members

*gmxapi* divides the root communicator into separate sub-communicators for each simulator in an ensemble simulation task. Consider a root communicator of size *S* being allocated to *N* simulators. Each rank *R* in the root communicator is assigned to ensemble member *M(R)* as follows.

When GROMACS is built with MPI library support, *gmxapi* allocates available MPI ranks to simulators in (approximately) equal size consecutive chunks.

$$M(R) = \text{trunc}(R * N/S)$$

For thread-MPI (or no-MPI) GROMACS builds, each simulator is assigned one process (with an attempt at even distribution). Based on the preceding formula, thread-MPI ensemble member assignment looks

like the following.

$$M_T(R) = \begin{cases} M(R) & , \ M(R) \neq M(R-1) \\ \text{null} & , \ \text{otherwise} \end{cases}$$

In other words, without an MPI library, only the root rank from *M(R)* is assigned.

Changed in version 0.4.0: In earlier releases, ranks were assigned to thread-MPI simulators contiguously, such that high-numbered ranks *R>N* were unused. MPI simulators were not supported for ensemble simulation tasks.

#### Caveats for MPI jobs

Changed in version 0.3.0: By default, most commands outside *gmxapi.simulation* (page 722) launch only on the root rank. (Results are synchronized to all ranks.) *gmxapi.function_wrapper* (page 718) allows you to set *allow_duplicate=True*, if your script logic or data transfer overhead require tasks to be executed on all ranks (computation is duplicated).

If *gmxapi.commandline_operation* (page 718) is used to wrap an MPI-enabled executable, the executable could behave unpredictably when the script is run in an MPI context. By default, *commandline_operation* subprocesses get a copy of the environment from the Python interpreter from which they are launched, and an executable may think it was launched directly by **mpiexec**, causing MPI errors when it tries to assert ownership of the MPI resources.

When a gmxapi script is launched in an MPI context, it may be necessary to hide the MPI context from MPI-aware commands run in subprocesses, since *gmxapi.commandline_operation* (page 718) executables are generally only launched on a single process. *gmxapi.runtime.filtered_mpi_environ()* (page 726) is available to provide a copy of the `os.environ` dictionary with known MPI-related environment variables filtered out.

Changed in version 0.3.1: You can use the *env* key word argument to *gmxapi.commandline_operation* (page 718) to replace the default map of environment variables. By pruning out the environment variables set by the MPI launcher, you can prevent the executable from automatically detecting an MPI context that it shouldn't use. See also Issue 4421

Changed in version 0.4.1: Added *gmxapi.runtime.filtered_mpi_environ()* (page 726).

gmxapi does not currently have an abstraction for subprocess launch methods. While such a feature is under investigation, *allow_duplicate* (*function_wrapper()* (page 718)) and *env* (*commandline_operation()* (page 718)) should allow users to wrap tools in custom launchers. Discussion welcome on the forum!

### 6.2.2 Running simple simulations

Once the `gmxapi` package is installed, running simulations is easy with *gmxapi.read_tpr()* (page 722).

```
import gmxapi as gmx
simulation_input = gmx.read_tpr(tpr_filename)
md = gmx.mdrun(simulation_input)
```

Note that this sets up the work you want to perform, but does not immediately trigger execution. You can explicitly trigger execution with:

```
md.run()
```

or you can let gmxapi automatically launch work in response to the data you request (by calling *result()* (page 717) on a named *output* member).

The *gmxapi.mdrun()* (page 723) operation produces a simulation trajectory output. You can use `md.output.trajectory` as input to other operations, or you can get the output directly by calling `md.output.trajectory.result()`. If the simulation has not been run yet when `result()` is called, the simulation will be run before the function returns.

### 6.2.3 Running ensemble simulations

To run a batch of simulations, just pass an array of inputs.:

```
md = gmx.read_tpr([tpr_filename1, tpr_filename2, ...])
md.run()
```

Make sure to launch the script in an MPI environment with a sufficient number of ranks to allow one rank per simulation.

**See also**

*Notes on parallelism and MPI* (page 709)

### 6.2.4 Input arguments and "ensemble" syntax

When a `list` of input is provided to a command argument that expects some other type, *gmxapi* generates an *ensemble* operation. The command is applied to each element of input, and the `result()` will be a list. When an *output* member of an ensemble operation is provided as input to another command, the consuming command will also be an ensemble operation.

*gmxapi* uses MPI to manage ensemble members across available resources. It is important that the same *gmxapi* commands are called on all processes so that underlying collective MPI calls are made as expected. In other words, if you are using `mpi4py` in your script, be careful with conditional execution like the following.

```
if mpi4py.MPI.COMM_WORLD.Get_rank() == 0:
    # don't put any gmxapi commands here, including method calls
    # like `obj.result()`, unless you have an `else`
    # to make sure the same gmxapi command runs on every rank.
    ...
```

For commands that already integrate well with *gmxapi's* MPI-based ensemble management (like *mdrun()* (page 723)), available resources can be split up automatically, and applied to run the ensemble members concurrently. Other operations may require further development of Resource Management API features for the *gmxapi* framework to most effectively apply multi-core computing resources. See Issue 3718 and the wiki for more information.

See also *Notes on parallelism and MPI* (page 709).

### 6.2.5 Accessing command line tools

In *gmxapi* 0.1, most GROMACS tools are not yet exposed as *gmxapi* Python operations. *gmxapi.commandline_operation* (page 718) provides a way to convert a **gmx** (or other) command line tool into an operation that can be used in a *gmxapi* script.

In order to establish data dependencies, input and output files need to be indicated with the `input_files` and `output_files` parameters. `input_files` and `output_files` key word arguments are dictionaries consisting of files keyed by command line flags.

For example, you might create a **gmx solvate** operation as:

```
solvate = gmx.commandline_operation('gmx',
                                    arguments=['solvate', '-box', '5
↪', '5', '5'],
                                    input_files={'-cs':␣
↪structurefile},
                                    output_files={'-p': topfile,
                                                  '-o':␣
↪structurefile,
                                                  }
```

To check the status or error output of a command line operation, refer to the `returncode` and `stderr` outputs. To access the results from the output file arguments, use the command line flags as keys in the `file` dictionary output.

Example:

```
structurefile = solvate.output.file['-o'].result()
if solvate.output.returncode.result() != 0:
    print(solvate.output.erroroutput.result())
```

### 6.2.6 Preparing simulations

Continuing the previous example, the output of `solvate` may be used as the input for `grompp`:

```
grompp = gmx.commandline_operation('gmx', 'grompp',
                                   input_files={
                                       '-f': mdpfile,
                                       '-p': solvate.output.file['-p
↪'],
                                       '-c': solvate.output.file['-o
↪'],
                                       '-po': mdout_mdp,
                                   },
                                   output_files={'-o': tprfile})
```

Then, `grompp.output.file['-o']` can be used as the input for *gmxapi.read_tpr()* (page 722).

Simulation input can be modified with the *gmxapi.modify_input()* (page 722) operation before being passed to *gmxapi.mdrun()* (page 723). For *gmxapi* 0.1, a subset of MDP parameters may be overridden using the dictionary passed with the `parameters` key word argument.

Example:

```
simulation_input = gmx.read_tpr(grompp.output.file['-o'])
modified_input = gmx.modify_input(input=simulation_input,␣
↪parameters={'nsteps': 1000})
md = gmx.mdrun(input=modified_input)
md.run()
```

### 6.2.7 Using arbitrary Python functions

Generally, a function in the *gmxapi* package returns an object that references a node in a work graph, representing an operation that will be run when the graph executes. The object has an `output` attribute providing access to data Futures that can be provided as inputs to other operations before computation has actually been performed.

You can also provide native Python data as input to operations, or you can operate on native results retrieved from a Future's `result()` method. However, it is trivial to convert most Python functions into *gmxapi* compatible operations with *gmxapi.function_wrapper()* (page 718). All function inputs and outputs must have a name and type. Additionally, functions should be stateless and importable (e.g. via Python `from some.module import myfunction`) for future compatibility.

Simple functions can just use `return()` to publish their output, as long as they are defined with a return value type annotation. Functions with multiple outputs can accept an `output` key word argument and assign values to named attributes on the received argument.

Examples:

```
from gmxapi import function_wrapper

@function_wrapper(output={'data': float})
def add_float(a: float, b: float) -> float:
    return a + b


@function_wrapper(output={'data': bool})
def less_than(lhs: float, rhs: float, output=None):
    output.data = lhs < rhs
```

**See also**

For more on Python type hinting with function annotations, check out **PEP 3107**.

### 6.2.8 Subgraphs

Basic *gmxapi* work consists of a flow of data from operation outputs to operation inputs, forming a directed acyclic graph (DAG). In many cases, it can be useful to repeat execution of a subgraph with updated inputs. You may want a data reference that is not tied to the immutable result of a single node in the work graph, but which instead refers to the most recent result of a repeated operation.

One or more operations can be staged in a `gmxapi.operation.Subgraph`, a sort of meta-operation factory that can store input binding behavior so that instances can be created without providing input arguments.

The subgraph *variables* serve as input, output, and mutable internal data references which can be updated by operations in the subgraph. Variables also allow state to be propagated between iterations when a subgraph is used in a *while* loop.

Use *gmxapi.subgraph()* (page 720) to create a new empty subgraph. The `variables` argument declares data handles that define the state of the subgraph when it is run. To initialize input to the subgraph, give each variable a name and a value.

To populate a subgraph, enter a SubgraphContext by using a `with()` statement. Operations created in the *with* block will be captued by the SubgraphContext. Define the subgraph outputs by assigning operation outputs to subgraph variables within the *with* block.

After exiting the *with* block, the subgraph may be used to create operation instances or may be executed repeatedly in a *while* loop.

> **Note**
>
> The object returned by *gmxapi.subgraph()* (page 720) is atypical of *gmxapi* operations, and has some special behaviors. When used as a Python context manager, it enters a "builder" state that changes the behavior of its attribute variables and of operaton instantiation. After exiting the `with()` block, the subgraph variables are no longer assignable, and operation references obtained within the block are no longer valid.

### 6.2.9 Looping

An operation can be executed an arbitrary number of times with a *gmxapi.while_loop()* (page 720) by providing a factory function as the *operation* argument. When the loop operation is run, the *operation* is instantiated and run repeatedly until *condition* evaluates `True`.

*gmxapi.while_loop()* (page 720) does not provide a direct way to provide *operation* arguments. Use a *subgraph* to define the data flow for iterative operations.

When a *condition* is a subgraph variable, the variable is evaluated in the running subgraph instance at the beginning of an iteration.

Example:

```
subgraph = gmx.subgraph(variables={'float_with_default': 1.0, 'bool_
↪data': True})
with subgraph:
    # Define the update for float_with_default to come from an add_
↪float operation.
```

(continues on next page)

(continued from previous page)

```
    subgraph.float_with_default = add_float(subgraph.float_with_
↪default, 1.).output.data
    subgraph.bool_data = less_than(lhs=subgraph.float_with_default,␣
↪rhs=6.).output.data
operation_instance = subgraph()
operation_instance.run()
assert operation_instance.values['float_with_default'] == 2.

loop = gmx.while_loop(operation=subgraph, condition=subgraph.bool_
↪data)
handle = loop()
assert handle.output.float_with_default.result() == 6
```

### 6.2.10 Logging

*gmxapi* uses the Python `logging` module to provide hierarchical logging, organized by submodule. You can access the logger at `gmxapi.logger` or through the Python logging framework:

```
import gmxapi as gmx
import logging

# Get the root gmxapi logger.
gmx_logger = logging.getLogger('gmxapi')
# Set a low default logging level
gmx_logger.setLevel(logging.WARNING)
# Make some tools very verbose
#  by descending the hierarchy
gmx_logger.getChild('commandline').setLevel(logging.DEBUG)
#  or by direct reference
logging.getLogger('gmxapi.mdrun').setLevel(logging.DEBUG)
```

You may prefer to adjust the log format or manipulate the log handlers. For example, tag the log output with MPI rank:

```
try:
    from mpi4py import MPI
    rank_number = MPI.COMM_WORLD.Get_rank()
except ImportError:
    rank_number = 0
    rank_tag = ''
    MPI = None
else:
    rank_tag = 'rank{}:'.format(rank_number)

formatter = logging.Formatter(rank_tag + '%(name)s:%(levelname)s:
↪%(message)s')

# For additional console logging, create and attach a stream␣
↪handler.
```

(continues on next page)

(continued from previous page)

```
ch = logging.StreamHandler()
ch.setFormatter(formatter)
logging.getLogger().addHandler(ch)
```

For more information, refer to the Python logging documentation.

### 6.2.11 More

Refer to the *gmxapi Python module reference* (page 716) for complete and granular documentation.

For more information on writing or using pluggable simulation extension code, refer to https://gitlab.com/gromacs/gromacs/-/issues/3133. (For gmxapi 0.0.7 and GROMACS 2019, see https://github.com/kassonlab/sample_restraint)

## 6.3 gmxapi Python module reference

Version 0.5.0a1.

The GROMACS Python package includes a high-level scripting interface implemented in pure Python and a lower-level API implemented as a C++ extension module. The pure Python implementation provides the basic `gmxapi` module and classes with a very stable syntax that can be maintained with maximal compatibility while mapping to lower level interfaces that may take a while to sort out. The separation also serves as a reminder that different execution contexts may be implemented quite differently, though Python scripts using only the high-level interface should execute on all.

Package documentation is extracted from the `gmxapi` Python module and is also available directly, using either `pydoc` from the command line or `help()` from within Python, such as during an interactive session.

Refer to the Python source code itself for additional clarification.

**See also**

*Accessing gmxapi documentation* (page 705)

### 6.3.1 Interface concepts

*gmxapi* commands return *references* to operations. Generally, the operations are collected into a graph of data flow dependencies, and only executed when the results are requested.

**class** gmxapi.abc.**OperationReference**

Client interface to an element of computational work already configured.

An "instance" of an operation is assumed to be a node in a computational work graph, owned and managed by a Context. This class describes the interface of the reference held by a client once the node exists.

The convergence of OperationReferences with Nodes as the results of the action of a Director implies that a Python user should also be able to "subscribe" to an operation handle (or its member resources). This could be a handy feature with which a user could register a call-back. Note that

we will want to provide an optional way for the call-back (as with any subscriber) to assert a chain of prioritized Contexts to find the optimal path of subscription.

**abstract property output: OutputDataProxy**

Get a proxy collection to the output of the operation.

Developer note: The 'output' property exists to isolate the namespace of output data from other operation handle attributes and we should consider whether it is actually necessary or helpful. To facilitate its possible future removal, do not enrich its interface beyond that of a collection of OutputDescriptor attributes. The OutputDataProxy also serves as a Mapping, with keys matching the attributes. We may choose to keep only this aspect of the interface instead of trying to keep track of the set of attributes.

**abstract run**()

Assert execution of an operation.

After calling run(), the operation results are guaranteed to be available in the local context.

*gmxapi* uses a *Future* (page 717) to reference an operation output or data that may not yet be available.

**class** gmxapi.abc.**Future**

Data source that may represent Operation output that does not yet exist.

Futures represent "immutable resources," or fixed points in the data flow.

**abstract property dtype: type**

Data type for the promised result.

**abstract result**() → Any

Fetch data to the caller's Context.

Returns the actual result of the operation supplying this Future.

Ensemble data are returned as a list. Scalar results or results from single member ensembles are returned as scalars. If this behavior is confusing or problematic for you, please reopen https://gitlab.com/gromacs/gromacs/-/issues/3179 or a new issue and join the discussion.

An *OperationReference* (page 716) may provide several named Futures on its *output* attribute.

A *Future* (page 717) may be provided directly as inputs to other *gmxapi* commands. *gmxapi* will execute the required operation to get the data when it is needed.

To get an actual result in your Python script, you can call *result()* (page 717) on any *gmxapi* data reference. If the operation has not yet been executed, the operation (and any operation dependencies) will be executed immediately.

You can also force an operation to run by calling its *run()* (page 717) method. But this is not generally necessary unless your only goal is to produce output files on disk that are not consumed in the same script.

In some cases, a *Future* (page 717) can be subscripted to get a new Future representing a slice of the original. For instance, *commandline_operation* (page 718) objects have a *file* output that produces a mapping of command line flags to output files (per the *output_files* parameter). This *file* output can be subscripted with a single command line option to get a *Future* (page 717) for just one output file type. See *Preparing simulations* (page 712) for an illustrative example.

#### Ensemble data flow

*gmxapi* automatically generates arrays of operations and parallel data flow, when parallel inputs are provided to *gmxapi* command parameters.

When a *Future* (page 717) represents the output of an ensemble operation, *result()* (page 717) returns a list with elements corresponding to the ensemble members.

It is not currently possible to get a *Future* (page 717) for a specific ensemble member.

See *Input arguments and "ensemble" syntax* (page 711) for more information.

### 6.3.2 gmxapi basic package

```
import gmxapi as gmx
```

gmxapi Python package for GROMACS.

This package provides Python access to GROMACS molecular simulation tools. Operations can be connected flexibly to allow high performance simulation and analysis with complex control and data flows. Users can define new operations in C++ or Python with the same tool kit used to implement this package.

`@gmxapi.function_wrapper`(*output: dict | None = None*, *allow_duplicate=False*)

Generate a decorator for wrapped functions with signature manipulation.

New function accepts the same arguments, with additional arguments required by the API.

The new function returns an object with an `output` attribute containing the named outputs.

#### Example

```
>>> @function_wrapper(output={'spam': str, 'foo': str})
... def myfunc(parameter: str = None, output=None):
...     output.spam = parameter
...     output.foo = parameter + ' ' + parameter
...
>>> operation1 = myfunc(parameter='spam spam')
>>> assert operation1.output.spam.result() == 'spam spam'
>>> assert operation1.output.foo.result() == 'spam spam spam␣
↪spam'
```

**Parameters**

**output** (*dict*) – output names and types

If `output` is provided to the wrapper, a data structure will be passed to the wrapped functions with the named attributes so that the function can easily publish multiple named results. Otherwise, the `output` of the generated operation will just capture the return value of the wrapped function.

gmxapi.**commandline_operation**(*executable=None*, *arguments=()*, *input_files: dict | Iterable[dict] | None = None*, *output_files: dict | Iterable[dict] | None = None*, *stdin: str | Iterable[str] | None = None*, *env: dict | Iterable[dict] | None = None*, ***kwargs*)

Helper function to define a new operation that executes a subprocess in gmxapi data flow.

Define a new Operation for a particular executable and input/output parameter set. Generate a chain of operations to process the named key word arguments and handle input/output data dependencies.

Note that the operation will be executed in a subprocess in an automatically generated subdirectory. See below for more information.

**Parameters**

- **arguments** – list of positional arguments to insert at `argv[1]`
- **env** – Optional replacement for the environment variables seen by the subprocess.
- **executable** – name of an executable on the path
- **input_files** – mapping of command-line flags to input file paths
- **output_files** – mapping of command-line flags to output file names
- **stdin** (*str*) – String input to send to STDIN (terminal input) of the executable (optional).

Changed in version 0.5.0: Relative paths in *input_files* are considered relative to the current working directory, and are immediately converted to absolute paths.

Multi-line text sent to *stdin* should be joined into a single string. E.g.:

```
commandline_operation(..., stdin='\n'.join(list_of_strings) + '\
↪n')
```

If multiple strings are provided to *stdin*, gmxapi will assume an ensemble, and will run one operation for each provided string.

Only string input (`str()`) to *stdin* is currently supported. If you have a use case that requires streaming input or binary input, please open an issue or contact the author(s).

Changed in version 0.3.0: output_files paths are converted to absolute paths at run time.

If non-absolute paths are provided to *output_files*, paths are resolved relative to the working directory of the command instance (not relative to the working directory of the workflow script).

By default, *executable* runs in a subprocess that inherits its environment and resources from the Python interpreter. (See https://docs.python.org/3/library/subprocess.html#subprocess.run)

Added in version 0.3.1: If specified, *env* replaces the default environment variables map seen by *executable* in the subprocess.

Changed in version 0.4.1: If unspecified, *env* defaults to a **filtered** copy of the current environment (with MPI-related environment variables removed). See *gmxapi.runtime.filtered_mpi_environ()* (page 726).

In addition to controlling environment variables used for user-input, it may be necessary to adjust the environment to prevent the subprocess from inheriting variables that it should not. This is particularly relevant if the Python script is launched with `mpiexec` and then `commandline_wrapper` is used to launch an MPI-aware executable that may try to manage the MPI context. (Reference Issue 4421)

When overriding the environment variables, don't forget to include basic variables like PATH that are necessary for the executable to run. `os.getenv` can help. E.g. `commandline_operation(..., env={'PATH': os.getenv('PATH'), ...})`

**See also**

*gmxapi.runtime.filtered_mpi_environ()* (page 726).

**Output:**

The output node of the resulting operation handle contains

- `directory`: filesystem path that was used as the working directory for the subprocess
- `file`: the mapping of CLI flags to filename strings resulting from the `output_files` kwarg
- `returncode`: return code of the subprocess.
- **`stderr`: A string mapping from process STDERR; it will be the** error output (if any) if the process failed.
- `stdout`: A string mapping from process STDOUT.

Changed in version 0.3: Subprocesses run in directories managed by gmxapi.

Added in version 0.3: The *directory* output.

Working directory names are details of the gmxapi implementation; the naming scheme is not yet specified by the API, but is intended to be related to the operation ID.

Note that re-executing a gmxapi script in the same directory will cause commands to be executed again in the same directories. If this presents a risk of data corruption (or just wasted compute cycles), you may include the key word argument `_exist_ok=False` to force an error. Please consider contacting the developers through any of the various GROMACS community channels to further discuss your use case.

`gmxapi.`**`subgraph`**(*variables: Mapping | None = None*)

Allow operations to be configured in a sub-context.

The object returned functions as a Python context manager. When entering the context manager (the beginning of the `with` block), the object has an attribute for each of the named `variables`. Reading from these variables gets a proxy for the initial value or its update from a previous loop iteration. At the end of the `with` block, any values or data flows assigned to these attributes become the output for an iteration.

After leaving the `with` block, the variables are no longer assignable, but can be called as bound methods to get the current value of a variable.

When the object is run, operations bound to the variables are `reset` and run to update the variables.

gmxapi.**while_loop**(*, *operation*, *condition*, *max_iteration=10*)

Generate and run a chain of operations such that condition evaluates True.

Returns and operation instance that acts like a single node in the current work graph, but which is a proxy to the operation at the end of a dynamically generated chain of operations. At run time, condition is evaluated for the last element in the current chain. If condition evaluates False, the chain is extended and the next element is executed. When condition evaluates True, the object returned by `while_loop` becomes a proxy for the last element in the chain.

Equivalent to calling operation.while(condition), where available.

**Parameters**

- **operation** – a callable that produces an instance of an operation when called with no arguments.
- **condition** – a callable that accepts an object (returned by `operation`) that returns a boolean.
- **max_iteration** – execute the loop no more than this many times (default 10)

> **Warning**
>
> *max_iteration* is provided in part to minimize the cost of bugs in early versions of this software. The default value may be changed or removed on short notice.

> **Warning**
>
> The protocol by which `while_loop` interacts with `operation` and `condition` is very unstable right now. Please refer to this documentation when installing new versions of the package.

**Protocol:**

**Warning:**
This protocol will be changed before the API is finalized.

When called, `while_loop` calls `operation` without arguments and captures the return value for inspection of outputs. The object produced by `operation()` must have a `reset`, a `run` method, and an `output` attribute. From gmxapi 0.1, an additional `values` attribute is examined to advertise the output members that will appear on the `while_loop` output.

This is inspected to determine the output data proxy for the operation produced by the call to `while_loop`. When that operation is called, it does the equivalent of

**while(condition(self._operation)):**
self._operation.reset() self._operation.run()

Then, the output data proxy of `self` is updated with the results from self._operation.output.

### 6.3.3 Simulation module

GROMACS simulation subpackage for gmxapi.

Provides operations for configuring and running molecular simulations.

The initial version of this module is a port of the gmxapi 0.0.7 facilities from https://github.com/kassonlab/gmxapi and is not completely integrated with the gmxapi 0.1 specification. Operation execution is dispatched to the old execution manager for effective ensemble handling and C++ MD module binding. This should be an implementation detail that is not apparent to the typical user, but it is worth noting that chains of gmxapi.simulation module operations will be automatically bundled for execution as gmxapi 0.0.7 style API sessions. Run time options and file handling will necessarily change as gmxapi data flow handling evolves.

In other words, if you rely on behavior not specified explicitly in the user documentation, please keep an eye on the module documentation when updating gmxapi and please participate in the ongoing discussions for design and implementation.

#### Preparing simulations

gmxapi.**read_tpr**(*filename*, *label: str | None = None*, *context=None*)

Get simulation input from a TPR file.

**Parameters**

- **filename** – input file name
- **label** – optional human-readable label with which to tag the new node
- **context** – Context in which to return a handle to the new node. Use default (None) for Python scripting interface

**Returns**

Reference (handle) to the new operation instance (node).

See *OutputDataProxy* (page 722) for members of the *output* attribute.

**class** gmxapi.simulation.read_tpr.**OutputDataProxy**(**args*, ***kwargs*)

Implement the 'output' attribute of *read_tpr* (page 722) operations.

**parameters**

Dictionary of simulation parameters.

Additionally (through an unspecified interface), the object serves as a complete simulation input to other gmxapi operations.

gmxapi.**modify_input**(*input*, *parameters: dict*, *label: str | None = None*, *context=None*)

Modify simulation input with data flow operations.

Given simulation input *input*, override components of simulation input with additional arguments, such as *parameters*.

See *OutputDataProxy* (page 722) for *output* attribute members.

**class** gmxapi.simulation.modify_input.**OutputDataProxy**(**args*, ***kwargs*)

Implement the 'output' attribute of *modify_input* (page 722) operations.

**parameters**

Aggregated dictionary simulation parameters for the resulting simulation input.

Additionally (through an unspecified interface), the object serves as a complete simulation input to other gmxapi operations.

## Running simulations

gmxapi.**mdrun**(*input*, *runtime_args: dict | Sequence[dict] | None = None*, *label: str | None = None*, *context=None*)

MD simulation operation.

**Parameters**

- **input** – valid simulation input
- **runtime_args** (*dict*) – command line flags and arguments to be passed to mdrun (optional)

**Returns**

runnable operation to perform the specified simulation

See *OutputDataProxy* (page 723) for members of the *output* attribute.

*input* may be a TPR file name or an object providing the SimulationInput interface.

*runtime_args* allows an optional dictionary of mdrun options, using the option flag (including the leading hyphen −) as the dictionary key. For mdrun command line options that do not take a value (e.g. `-noappend`), use `None` as the dictionary value.

**Warning**

Run time argument processing does not have Python bindings at this time. key-value pairs are passed as plain text to the underlying library. Usage errors can be hard to discover. Refer to the MD log file in the output directory for messages regarding argument processing.

Note, in particular, that the available `mdrun` arguments can depend on the GROMACS build configuration, such as whether an MPI library or thread-MPI is enabled.

**See also**

The *gmx mdrun* (page 258) command line tool.

**class** gmxapi.simulation.mdrun.**OutputDataProxy**(*instance: SourceResource*, *client_id: int | None = None*)

Implement the 'output' attribute of *mdrun* (page 723) operations.

**checkpoint**

Full path to `cpt` file.

**Type**

str

**directory**

Full path to the working directory in which the simulation ran.

**Type**
str

**parameters**

Dictionary of parameters with which the simulation was run.

**Type**
dict

**stderr**

Full path to the text file that captured stderr during the simulation.

**Type**
str

**stdout**

Full path to the text file that captured stdout during the simulation.

**Type**
str

**trajectory**

Full path to trajectory output (corresponding to the `-o` flag, if provided).

**Type**
str

#### Notes

For multi-rank MPI-enabled simulators, *stderr* and *stdout* are reported for the root rank only, in line with how GROMACS behaves.

Changed in version 0.4: Added *directory* output, replacing an earlier "hidden" *_work_dir* output.

Added in version 0.4: *stderr* and *stdout* provide paths to the captured standard I/O. Previously, a lot of output from the underlying library bypassed Python and went straight to the standard output and standard error of the calling process.

### 6.3.4 Utilities

Provide some additional utilities.

`gmxapi.utility.config()`

Get the GROMACS configuration detected during installation.

Returns read-only dictionary proxy to file written during installation. The `Mapping` contains information about the supporting GROMACS installation that was used to configure the Python package installation. The exact keys in the Mapping is not formally specified, and mostly for internal use.

Added in version 0.4.

gmxapi.utility.**join_path**(*first: str*, *second: str*, *output=None*)

Get a Future path for use in data flow.

This is useful when a base path or filename is not known until runtime.

output.**path**

*first* and *second*, joined by the native filesystem path separator.

**Type**

str

gmxapi.**concatenate_lists**(*sublists: Sequence[Sequence[Scalar]] = ()*) → ArrayFuture[Scalar]

Combine data sources into a single list.

A trivial data flow restructuring helper.

gmxapi.**join_arrays**(*, *front: NDArray = ()*, *back: NDArray = ()*) → Future[NDArray]

Consumes two sequences and produces a concatenated single sequence.

Note that the exact signature of the operation is not determined until this helper is called. Helper functions may dispatch to factories for different operations based on the inputs. In this case, the dtype and shape of the inputs determines dtype and shape of the output. An operation instance must have strongly typed output, but the input must be strongly typed on an object definition so that a Context can make runtime decisions about dispatching work and data before instantiating.

gmxapi.**logical_not**(*value*)

Boolean negation.

If the argument is a gmxapi compatible Data or Future object, a new View or Future is created that proxies the boolean opposite of the input.

If the argument is a callable, logical_not returns a wrapper function that produces the logical opposite of the result of the callable. If the callable produces a (non-string) sequence, the wrapper returns a list of the negated results of the callable.

gmxapi.**make_constant**(*value: Scalar*) → Future[Scalar]

Provide a predetermined value at run time.

This is a trivial operation that provides a (typed) value, primarily for internally use to manage gmxapi data flow.

Accepts a value of any type. The object returned has a definite type and provides same interface as other gmxapi outputs. Additional constraints or guarantees on data type may appear in future versions.

### Run time details

**Note**

The *gmxapi.runtime* Python module is evolving. Some details are not yet well specified.

Manage computing resources and runtime context.

Utilities, context managers, and singletons for handling resource allocation and lifetime management.

For the purposes of this module, the term "assignment" refers to resources that have been reserved (usually for exclusive use) within a well-defined scope (such as a specific function call or phase of program execution).

"Allocation" can be taken to mean "resources that are available to be assigned," and may or may not be tightly scoped.

Resource assignment may be nested, in which case an Assignment with a broader scope in the program is used as the Allocation from which a more local Assignment is made.

Assignment and Allocation are *roles* that can be served by Context objects. Abstractly, a Context is a notion of some aspect of program or resource *state*. A Context may be represented by a concrete class when an object manages the details needed to participate in a stateful protocol.

Note that resource assignment is usually a collective operation within the scope of an allocation. For instance, an `MPI_Comm_split` call must be made (synchronously) on all members of the parent MPI communicator. Similar care must be taken with the Allocation and Assignment protocols in this module.

Added in version 0.4.0: This module generalizes some of the resource management previously in `gmxapi.simulation.mdrun`, decoupling MPI communicator handling from `gmxapi.simulation.context`. See also https://gitlab.com/gromacs/gromacs/-/issues/3718

`gmxapi.runtime.`**`filtered_mpi_environ`**`()` → dict

Return a filtered environment variables map with MPI-related entries removed.

> **See also**
>
> *`filtered_prefixes`* (page 726)

`gmxapi.runtime.`**`filtered_prefixes = ('DCMF_', 'MPICH_', 'MPIEXEC_', 'MPIO_', 'MV2_', 'MVAPICH_', 'HYDRA_', 'OMPI_', 'PMI_', 'PMIX_', 'I_MPI_')`**

MPI-related environment variable prefixes.

Environment variable prefixes known to be associated with MPI implementations, which may affect MPI context detection, and which should not matter outside of MPI contexts.

#### References

- Issue 4423
- Issue 4736

### 6.3.5 Status messages and Logging

Python logging facilities use the built-in logging module.

Upon import, the gmxapi package sets a placeholder "NullHandler" to block propagation of log messages to the root logger (and sys.stderr, if not handled).

If you want to see gmxapi logging output on `sys.stderr`, import `logging` in your script or module and configure it. For the simplest case, consider `logging.basicConfig`:

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG)
```

For more advanced usage, consider attaching a `logging.StreamHandler` to the `gmxapi` logger.

The gmxapi logging module adds an additional `rank_tag` log formatter field that can be particularly helpful in ensemble MPI workflows.

Example:

```
ch = logging.StreamHandler()
# Optional: Set log level.
ch.setLevel(logging.DEBUG)
# Optional: create formatter and add to character stream handler
formatter = logging.Formatter('%(levelname)s %(asctime)s:%(name)s
↪%(rank_tag)s%(message)s')
ch.setFormatter(formatter)
# add handler to logger
logging.getLogger('gmxapi').addHandler(ch)
```

To handle log messages that are issued while importing *gmxapi* (page 718) and its submodules, attach the handler before importing *gmxapi* (page 718)

Each module in the gmxapi package uses its own hierarchical logger to allow granular control of log handling (e.g. `logging.getLogger('gmxapi.operation')`). Refer to the Python `logging` module for information on connecting to and handling logger output.

### 6.3.6 Exceptions module

Exceptions and Warnings raised by gmxapi module operations.

Errors, warnings, and other exceptions used in the GROMACS Python package are defined in the *exceptions* (page 727) submodule.

The gmxapi Python package defines a root exception, exceptions.Error, from which all Exceptions thrown from within the module should derive. If a published component of the gmxapi package throws an exception that cannot be caught as a gmxapi.exceptions.Error, please report the bug.

**exception** gmxapi.exceptions.**ApiError**

An API operation was attempted with an incompatible object.

**exception** gmxapi.exceptions.**DataShapeError**

An object has an incompatible shape.

This exception does not imply that the Type or any other aspect of the data has been checked.

**exception** gmxapi.exceptions.**Error**

Base exception for gmx.exceptions classes.

**exception** gmxapi.exceptions.**FeatureNotAvailableError**

Requested feature not available in the current environment.

This exception will usually indicate an issue with the user's environment or run time details. There may be a missing optional dependency, which should be specified in the exception message.

**exception** gmxapi.exceptions.**MissingImplementationError**

Specified feature is not implemented in the current code.

This exception indicates that the implemented code does not support the API as specified. The calling code has used valid syntax, as documented for the API, but has reached incompletely implemented code, which should be considered a bug.

Changed in version 0.3: Named changed to avoid conflict with built-in `NotImplementedError` exception

**exception** gmxapi.exceptions.**ProtocolError**

Unexpected API behavior or protocol violation.

This exception generally indicates a gmxapi bug, since it should only occur through incorrect assumptions or misuse of API implementation internals.

**exception** gmxapi.exceptions.**TypeError**

Incompatible type for gmxapi data.

Reference datamodel.rst for more on gmxapi data typing.

**exception** gmxapi.exceptions.**UsageError**

Unsupported syntax or call signatures.

Generic usage error for gmxapi module.

**exception** gmxapi.exceptions.**ValueError**

A user-provided value cannot be interpreted or doesn't make sense.

**exception** gmxapi.exceptions.**Warning**

Base warning class for gmx.exceptions.

### 6.3.7 gmx.version module

gmxapi version and release information.

The `gmxapi.__version__` attribute contains a **version string**. The more general way to access the package version is with the `pkg_resources` module:

```
pkg_resources.get_distribution('gmxapi').version
```

*gmxapi.version* (page 728) module functions *api_is_at_least()* (page 729) and *has_feature()* (page 729) support additional convenience and introspection.

Changed in version 0.2: This module no longer provides public data attributes. Instead, use the module functions or `packaging.version`.

**See also**

Consider https://packaging.pypa.io/en/latest/version/ for programmatic handling of the version string. For example:

```
import pkg_resources
from packaging.version import parse
gmxapi_version = pkg_resources.get_distribution('gmxapi').version
if parse(gmxapi_version).is_prerelease:
    print('The early bird gets the worm.')
```

gmxapi.version.**api_is_at_least**(*major_version*, *minor_version=0*, *patch_version=0*)

Allow client to check whether installed module supports the requested API level.

**Parameters**

- **major_version** (*int*) – gmxapi major version number.
- **minor_version** (*int*) – optional gmxapi minor version number (default: 0).
- **patch_version** (*int*) – optional gmxapi patch level number (default: 0).

**Returns**

True if installed gmx package is greater than or equal to the input level

Note that if gmxapi.version.release is False, the package is not guaranteed to correctly or fully support the reported API level.

gmxapi.version.**has_feature**(*name: str*, *enable_exception=False*) → bool

Query whether a named feature is available in the installed package.

Between updates to the API specification, new features or experimental aspects may be introduced into the package and need to be detectable. This function is intended to facilitate code testing and resolving differences between development branches. Users should refer to the documentation for the package modules and API level.

The primary use case is, in conjunction with *api_is_at_least()* (page 729), to allow client code to robustly identify expected behavior and API support through conditional execution and branching. Note that behavior is strongly specified by the API major version number. Features that have become part of the specification and bug-fixes referring to previous major versions should not be checked with *has_feature()*. Using *has_feature()* with old feature names will produce a DeprecationWarning for at least one major version, and client code should be updated to avoid logic errors in future versions.

For convenience, setting `enable_exception = True` causes the function to instead raise a gmxapi.exceptions.FeatureNotAvailableError for unrecognized feature names. This allows extension code to cleanly produce a gmxapi exception instead of first performing a boolean check. Also, some code may be unexecutable for more than one reason, and sometimes it is cleaner to catch all *gmxapi.exceptions.Error* (page 727) exceptions for a code block, rather than to construct complex conditionals.

**Returns**

True if named feature is recognized by the installed package, else False.

**Raises**

***gmxapi.exceptions.FeatureNotAvailableError*** (page 727) – If `enable_exception == True` and feature is not found.

### 6.3.8 Core API

#### gmxapi core module

gmxapi._gmxapi provides Python access to the GROMACS C++ API so that client code can be implemented in Python, C++, or a mixture. The classes provided are mirrored on the C++ side in the gmxapi namespace as best as possible.

This documentation is generated from C++ extension code. Refer to C++ source code and developer documentation for more details.

#### Exceptions

#### Module Exceptions

**exception** gmxapi._gmxapi.**Exception**

Root exception for the C++ extension module. Derives from *gmxapi.exceptions.Error* (page 727).

**exception** gmxapi._gmxapi.**FeatureNotAvailable**

An API feature is not available in the current installation.

This may occur when a new gmxapi Python package is installed with an older GROMACS installation that does not have the library support for a newer feature.

#### Wrapped C++ exceptions emitted through the supporting GROMACS library

**exception** gmxapi._gmxapi.**MissingImplementationError**

Expected feature is not implemented.

Changed in version 0.3: Renamed from NotImplementedError.

**exception** gmxapi._gmxapi.**ProtocolError**

Behavioral protocol violated.

**exception** gmxapi._gmxapi.**UsageError**

Unacceptable API usage.

#### Other

No other C++ exceptions are expected, but will be wrapped in a *Exception* (page 730) to help tracing and reporting bugs.

**exception** gmxapi._gmxapi.**UnknownException**

Catch-all exception wrapper.

GROMACS library produced an exception that is not mapped in gmxapi or which should have been caught at a lower level. I.e. a bug. (Please report.)

### Functions

This documentation is provided for completeness and as an aid to developers. Users of the *gmxapi* (page 718) package, generally, should not need to use the following tools directly.

### Tools for launching simulations

gmxapi._gmxapi.**from_tpr**(*arg0: str*) → *gmxapi._gmxapi.MDSystem* (page 732)

Return a system container initialized from the given input record.

gmxapi._gmxapi.**create_context**(**args*, ***kwargs*)

Overloaded function.

1. create_context() -> gmxapi._gmxapi.Context

Initialize a new API Context to manage resources and software environment.

2. create_context(arg0: object) -> gmxapi._gmxapi.Context

Initialize a new API Context to manage resources and software environment.

### Tools to manipulate TPR input files

gmxapi._gmxapi.**copy_tprfile**(*source:* gmxapi._gmxapi.TprFile *(page 732)*, *destination: str*) → bool

Copy a TPR file from `source` to `destination`.

gmxapi._gmxapi.**read_tprfile**(*filename: str*) → *gmxapi._gmxapi.TprFile* (page 732)

Get a handle to a TPR file resource for a given file name.

gmxapi._gmxapi.**write_tprfile**(*filename: str*, *parameters:* gmxapi._gmxapi.SimulationParameters *(page 732)*) → None

Write a new TPR file with the provided data.

gmxapi._gmxapi.**rewrite_tprfile**(*source: str*, *destination: str*, *end_time: float*) → bool

Copy a TPR file from `source` to `destination`, replacing *nsteps* (page 53) with `end_time`.

### Utilities

gmxapi._gmxapi.**has_feature**(*arg0: str*) → bool

Check feature *name* first with the bindings package, then the supporting library.

Available features may depend on the package version, the details of the supporting GROMACS installation, the software environment detected when the package was built, or possibly on detected runtime details. These feature checks are largely for internal use. The *gmxapi* (page 718) commands may adjust their behavior slightly depending on feature checks, and (at worst) should produce meaningful error messages or exceptions.

Named features:

- *create_context*: `create_context` (page 731) can be used to initialize a `Context` (page 732) with assigned resources.
- *mpi_bindings*: C++ extension module was built with `mpi4py` compatibility.

### Classes

**class** gmxapi._gmxapi.**Context**

**add_mdmodule** (*self:* gmxapi._gmxapi.Context *(page 732)*, *arg0:* *object*) → None

Add an MD plugin for the simulation.

**setMDArgs** (*self:* gmxapi._gmxapi.Context *(page 732)*, *arg0:* gmxapi._gmxapi.MDArgs *(page 732)*) → None

Set MD runtime parameters.

**class** gmxapi._gmxapi.**MDArgs**

**get_args** (*self:* gmxapi._gmxapi.MDArgs *(page 732)*) → list

Get an iterator of command line argument tokens, if possible and relevant.

**set** (*self:* gmxapi._gmxapi.MDArgs *(page 732)*, *arg0:* *dict*) → None

Assign parameters in MDArgs from Python dict.

**class** gmxapi._gmxapi.**MDSession**

**close** (*self:* gmxapi._gmxapi.MDSession *(page 732)*) → gmxapi._gmxapi.Status

Shut down the execution environment and close the session.

**run** (*self:* gmxapi._gmxapi.MDSession *(page 732)*) → gmxapi._gmxapi.Status

Run the simulation workflow

**class** gmxapi._gmxapi.**MDSystem**

**launch** (*self:* gmxapi._gmxapi.MDSystem *(page 732)*, *arg0:* gmxapi._gmxapi.Context *(page 732)*) → *gmxapi._gmxapi.MDSession* (page 732)

Launch the configured workflow in the provided context.

**class** gmxapi._gmxapi.**SimulationParameters**

**extract** (*self:* gmxapi._gmxapi.SimulationParameters *(page 732)*) → dict

Get a dictionary of the parameters.

**set** (**args*, ***kwargs*)

Overloaded function.

1. set(self: gmxapi._gmxapi.SimulationParameters, key: str, value: int) -> None

Use a dictionary to update simulation parameters.

2. set(self: gmxapi._gmxapi.SimulationParameters, key: str, value: float) -> None

Use a dictionary to update simulation parameters.

3. set(self: gmxapi._gmxapi.SimulationParameters, key: str, value: None) -> None

Use a dictionary to update simulation parameters.

**class** gmxapi._gmxapi.**TprFile**

**params**(*self:* gmxapi._gmxapi.TprFile *(page 732)*) → *gmxapi._gmxapi.SimulationParameters* (page 732)

To discuss *gmxapi*, use the `#gmxapi` tag on the GROMACS user discussion forum. Report bugs through the GROMACS issue tracker.

Complete documentation for the *gmxapi* Python package is part of the GROMACS manual since *gmxapi* version 0.1. Older releases can also be found at GitHub. Pre-release documentation may be found at https://manual.gromacs.org/nightly/gmxapi/.

**See also**

gmxapi publications

Irrgang, M. E., Davis, C., & Kasson, P. M. gmxapi: A GROMACS-native Python interface for molecular dynamics with ensemble and plugin support. *PLOS Comput Biol* 2022. DOI: 10.1371/journal.pcbi.1009835

Irrgang, M. E., Hays, J. M., & Kasson, P. M. gmxapi: a high-level interface for advanced control and extension of molecular dynamics simulations. *Bioinformatics* 2018. DOI: 10.1093/bioinformatics/bty484

CHAPTER

SEVEN

# (NON-)BONDED LIBRARY (NB-LIB) API

This documentation is part of the GROMACS manual and describes the (Non-)Bonded LIBrary (NB-LIB) API.

## 7.1 Guide to Writing MD Programs

The goal of NB-LIB's is to enable researchers to programmatically define molecular simulations. Traditionally these have been performed using a collection of executables and a manual workflow followed by a "black-box" simulation engine. NB-LIB allows users to script a variety of novel simulation and analysis workflows at a more granular level.

Many possible use cases are facilitated by the flexibility that NB-LIB allows. These include customized update rules, defining custom forces, or orchestrating swarms of simulations. NB-LIB also allows for writing conventional MD simulations and analysis.

This document goes over the steps to write MD programs using the API in NB-LIB that exposes features that are a part of the GROMACS package.

### 7.1.1 Global Definitions

NB-LIB programs are written in C++ so its headers for I/O or advanced tasks must be included. In addition, one must include the headers for various capabilities and abstractions NB-LIB exposes as well. This can be directly copied from here. Finally, we use the namespace `nblib` for the data structures defined in the library. The last line in the block allows one to skip this specifier each time a function or a data structure is used.

```
#include <cstdio>

#include "nblib/box.h"
#include "nblib/forcecalculator.h"
#include "nblib/integrator.h"
#include "nblib/molecules.h"
#include "nblib/nbkerneloptions.h"
#include "nblib/particletype.h"
#include "nblib/simulationstate.h"
#include "nblib/topology.h"

using namespace nblib;
```

### 7.1.2 Define Particle Data

```
// Parameters from a GROMOS compatible force-field 2016H66

struct OWaterAtom
{
    ParticleName          name = "Ow";
    Mass                  mass = 15.999;
    C6                    c6   = 0.0026173456;
    C12                   c12  = 2.634129e-06;
};

struct HwAtom
{
    ParticleName          name = "Hw";
    Mass                  mass = 1.00784;
    C6                    c6   = 0.0;
    C12                   c12  = 0.0;
};

struct CMethAtom
{
    ParticleName          name = "Cm";
    Mass                  mass = 12.0107;
    C6                    c6   = 0.01317904;
    C12                   c12  = 34.363044e-06;
};

struct HcAtom
{
    ParticleName          name = "Hc";
    Mass                  mass = 1.00784;
    C6                    c6   = 8.464e-05;
    C12                   c12  = 15.129e-09;
};
```

There can be as many structs of this kind as there are particle types in the system. Organizing the data like this is not strictly necessary, but is shown for the purpose of clarity. As shown here, there can be multiple particles that correspond to a single element as atomic mass can vary by molecular context. For example, the carbon atom in a carboxyl group would have different parameters from one in the methyl group. We can obtain the parameter set from any standard force-field, or generate new parameters to study new compounds or force fields. This example comes from the 2016H66 Parameter Set.

### 7.1.3 Defining Coordinates, Velocities and Force Buffers

```
std::vector<gmx::RVec> coordinates = {
    { 0.794, 1.439, 0.610 }, { 1.397, 0.673, 1.916 }, { 0.659, 1.
↪080, 0.573 },
    { 1.105, 0.090, 3.431 }, { 1.741, 1.291, 3.432 }, { 1.936, 1.
↪441, 5.873 },
    { 0.960, 2.246, 1.659 }, { 0.382, 3.023, 2.793 }, { 0.053, 4.
↪857, 4.242 },
    { 2.655, 5.057, 2.211 }, { 4.114, 0.737, 0.614 }, { 5.977, 5.
↪104, 5.217 },
};

std::vector<gmx::RVec> velocities = {
    { 0.0055, -0.1400, 0.2127 }, { 0.0930, -0.0160, -0.0086 }, { 0.
↪1678, 0.2476, -0.0660 },
    { 0.1591, -0.0934, -0.0835 }, { -0.0317, 0.0573, 0.1453 }, { 0.
↪0597, 0.0013, -0.0462 },
    { 0.0484, -0.0357, 0.0168 }, { 0.0530, 0.0295, -0.2694 }, { -0.
↪0550, -0.0896, 0.0494 },
    { -0.0799, -0.2534, -0.0079 }, { 0.0436, -0.1557, 0.1849 }, { -
↪0.0214, 0.0446, 0.0758},
};

std::vector<gmx::RVec> forces = {
    { 0.0000, 0.0000, 0.0000 }, { 0.0000, 0.0000, 0.0000 }, { 0.
↪0000, 0.0000, 0.0000 },
    { 0.0000, 0.0000, 0.0000 }, { 0.0000, 0.0000, 0.0000 }, { 0.
↪0000, 0.0000, 0.0000 },
    { 0.0000, 0.0000, 0.0000 }, { 0.0000, 0.0000, 0.0000 }, { 0.
↪0000, 0.0000, 0.0000 },
    { 0.0000, 0.0000, 0.0000 }, { 0.0000, 0.0000, 0.0000 }, { 0.
↪0000, 0.0000, 0.0000 },
};
```

We can initialize coordinates for our particles using `std::vector` of `gmx::RVec` which is a specific data type for holding 3D vector quantities. Doxygen page on RVec here.

### 7.1.4 Writing the MD Program

As with as any basic C++ program, there needs to be a `main()` function.

#### Define ParticleTypes

```
int main()
{
    // Bring the parameter structs to scope
    OwAtom      owAtom;
    HwAtom      hwAtom;
    CMethAtom   cmethAtom;
    HcAtom      hcAtom;

    // Create the particles
    ParticleType Ow(owAtom.name, owAtom.mass);
    ParticleType Hw(hwAtom.name, hwAtom.mass);
    ParticleType Cm(cmethAtom.name, cmethAtom.mass);
    ParticleType Hc(hcAtom.name, hcAtom.mass);
```

As before, the helper struct to define `ParticleType` data is not strictly needed, but is shown for clarity. The line `ParticleType CMethAtom(ParticleName("Cm"), Mass(12.0107));` would be sufficient.

#### Define Non-Bonded Interactions

```
ParticleTypeInteractions interactions(CombinationRule::Geometric);

// add non-bonded interactions for the particle types
interactions.add(owAtom.name, owAtom.c6, owAtom.c12);
interactions.add(hwAtom.name, hwAtom.c6, hwAtom.c12);
interactions.add(cmethAtom.name, cmethAtom.c6, cmethAtom.c12);
interactions.add(hcAtom.name, hcAtom.c6, hcAtom.c12);
```

For the Lennard-Jones interactions, we define a `ParticleTypeInteractions` object. Each particle of the `ParticleType` interacts with each other based on the `C6` and `C12` parameters. These parameters of the two different particles are averaged using `Geometric` or `LorentzBerthelot` `CombinationRule`. More details here. By default `CombinationRule::Geometric` is selected.

We add the interaction parameters of each of the particle types into the `ParticleTypeInteractions` object. The result is a table that has interactions specified for all `ParticleType` pairs. The following matrix describes the pair-wise C6 parameter created using `CombinationRule::Geometric`.

| # | Ow | Hw | Cm | Hc |
|---|---|---|---|---|
| Ow | 0.0026 | 0.0 | 0.42 | 4.7e-4 |
| Hw | 0.0 | 0.0 | 0.0 | 0.0 |
| Cm | 0.42 | 0.0 | 0.013 | 1.05e-3 |
| Hc | 4.7e-4 | 0.0 | 1.05e-3 | 8.5e-5 |

For a particular interaction pair, the user can also override the specified `CombinationRule` with custom parameters. The following overload would replace the parameters computed from a `CombinationRule` between `Ow` and `Cm` particle types.

```
interactions.add("Ow", "Cm", 0.42, 42e-6);
```

To facilitate modular, reusable code, it is possible to combine multiple `ParticleTypeInteractions` objects. Assuming `otherInteractions` is defined, this can be done with `interactions.merge(otherInteractions)`

### Define Molecules

```
Molecule water("Water");
Molecule methane("Methane");

water.addParticle(ParticleName("O"), Ow);
water.addParticle(ParticleName("H1"), Hw);
water.addParticle(ParticleName("H2"), Hw);

water.addExclusion("H1", "O");
water.addExclusion("H2", "O");

methane.addParticle(ParticleName("C"), Cm);
methane.addParticle(ParticleName("H1"), Hc);
methane.addParticle(ParticleName("H2"), Hc);
methane.addParticle(ParticleName("H3"), Hc);
methane.addParticle(ParticleName("H4"), Hc);

methane.addExclusion("H1", "C");
methane.addExclusion("H2", "C");
methane.addExclusion("H3", "C");
methane.addExclusion("H4", "C");
```

We begin declaring molecules with their constituent particles. A string identifier must uniquely identify a specific particle within the molecule. It is also possible to define partial charges on each particle for the computation of Coulomb interactions. `water.addParticle(ParticleName("O"), Charge(-0.04), Ow);`

Adding exclusions ensures that non-bonded interactions are only computed when necessary. For example, if two particles share a bond, the potential energy of the bond makes the non-bonded term negligible. Particle self-exclusions are enabled by default. We use the unique identifiers specified during `addParticle()` for this and the listed interactions later.

### Define Listed Interactions

Within a molecule, one can define interactions such as bonds, angles and dihedrals between the constituent particles. NB-LIB provides concrete implementations of several commonly used 2, 3 and 4 center interactions.

```
HarmonicBondType ohHarmonicBond(1, 1);
HarmonicBondType hcHarmonicBond(2, 1);

DefaultAngle hohAngle(Degrees(120), 1);
DefaultAngle hchAngle(Degrees(109.5), 1);

//add harmonic bonds for water
water.addInteraction("O", "H1", ohHarmonicBond);
water.addInteraction("O", "H2", ohHarmonicBond);

// add the angle for water
water.addInteraction("H1", "O", "H2", hohAngle);

// add harmonic bonds for methane
methane.addInteraction("H1", "C", hcHarmonicBond);
methane.addInteraction("H2", "C", hcHarmonicBond);
methane.addInteraction("H3", "C", hcHarmonicBond);
methane.addInteraction("H4", "C", hhcHarmonicBondc);

// add the angles for methane
methane.addInteraction("H1", "C", "H2", hchAngle);
methane.addInteraction("H1", "C", "H3", hchAngle);
methane.addInteraction("H1", "C", "H4", hchAngle);
methane.addInteraction("H2", "C", "H3", hchAngle);
methane.addInteraction("H2", "C", "H4", hchAngle);
methane.addInteraction("H3", "C", "H4", hchAngle);
```

### Define Options for the Simulation and Non-Bonded Calculations

```
// Define a box for the simulation
Box box(6.05449);

// Define options for the non-bonded kernels
NBKernelOptions options;
```

One can define the bounding box either with a single argument for a cube and 3 arguments to specify length, breadth and height separately.

`NBKernelOptions` contains a set of flags and configuration options for both hardware context and the relevant calculations for the simulation. The following table describes the possible options that can be set.

| Flag or Config Option | Type | Implications |
|---|---|---|
| `useGpu` | Boolean | Use GPU for non-bonded computations |
| `numThreads` | Integer | Number of CPU threads to use |
| `nbnxmSimd` | Enum | Kernel SIMD type (`SimdAuto`/`SimdNo`/`Simd4XM`/`Simd2XMM`) |
| `useHalfLJOptimization` | Boolean | Enable i-cluster half-LJ optimization |
| `pairlistCutoff` | Real | Specify pairlist and interaction cut-off |
| `computeVirialAndEnergy` | Boolean | Enable energy computations |
| `coulombType` | Enum | Coulomb interaction function (`Pme`/`Cutoff`/`ReactionField`) |
| `useTabulatedEwaldCorr` | Boolean | Use tabulated PME grid correction instead of analytical |
| `numIterations` | Integer | Specify number of iterations for each kernel |
| `cyclesPerPair` | Boolean | Enable printing cycles/pair instead of pairs/cycle |
| `timestep` | Real | Specify the time step |

### Define Topology and Simulation State

We build the system topology using the `TopologyBuilder` class. We add the `Molecule` objects that we defined previously along with the `ParticleTypesInteractions` using its public functions. We get the actual `Topology` object complete with all exclusions, interaction maps and listed interaction data constructed based on the defined entities using the `buildTopology()` function.

```
TopologyBuilder topologyBuilder;

// add molecules
topologyBuilder.addMolecule(water, 10);
topologyBuilder.addMolecule(methane, 10);

// add non-bonded interaction map
topologyBuilder.addParticleTypesInteractions(interactions);

Topology topology = topologyBuilder.buildTopology();
```

We now have all we need to fully describe our system using the `SimulationState` object. This is built using the topology, the box, and the particle coordinates and velocities. This object serves as a snapshot of the system that can be used for analysis or to start simulations from known states.

```
SimulationState simulationState(coordinates, velocities, forces,␣
↪box, topology);
```

### Writing the MD Loop

Now that we have fully described our system and the problem, we need two entities to write an MD loop. The first is the `ForceCalculator` and the second is an Integrator. NB-LIB comes with a `LeapFrog` integrator but it is also possible for users to write custom integrators.

```
// The force calculator contains all the data needed to compute␣
↪forces
ForceCalculator forceCalculator(simulationState, options);

// Integration requires masses, positions, and forces
LeapFrog integrator(simulationState);

// Allocate a force buffer
gmx::ArrayRef<gmx::RVec> userForces(topology.numParticles());

// MD Loop
int numSteps = 100;

for (i = 0; i < numSteps; i++)
{
  userForces = forceCalculator.compute();

  // The forces are not automatically updated in case the user␣
↪wants to add their own
  std::copy(userForces.begin(), userForces.end(),␣
↪begin(simulationState.forces()));

  // Integrate with a time step of 1 fs
  integrator.integrate(1.0);
}

return 0;
} // main
```

CHAPTER

EIGHT

# DEVELOPER GUIDE

This set of pages contains guidelines, instructions, and explanations related to GROMACS development. The actual code is documented in Doxygen documentation linked below.

The focus is (at least for now) on things that are tightly tied to the code itself, such as helper scripts that reside in the source repository and organization of the code itself, and may require the documentation to be updated in sync.

The guide is currently split into a few main parts:

- Overview of the GROMACS codebase.
- Collection of overview pages that describe some important implementation aspects.
- Generic guidelines to follow when developing GROMACS. For some of the guidelines, scripts exist (see below) to automatically reformat the code and/or enforce the guidelines for each commit.
- Instructions on what tools are used, and how to use them.

**The full code documentation generated from Doxygen can be found in the online documentation. It is not included here in order to save the trees.**

Some overview documentation that is closely related to the actual C/C++ code appears in the Doxygen documentation, while some other overview content is in the developer guide. The reasons are partially technical, but crosslinks between the developer guide and the Doxygen documentation are provided whenever related content appears split between the two sources.

The documentation does not yet cover all areas, but more content is being (slowly) added.

## 8.1 Contribute to GROMACS

GROMACS is a community-driven project, and we love getting contributions from people. Contributions are welcome in many forms, including improvements to documentation, patches to fix bugs, advice on the forums, bug reports that let us reproduce the issue, and new functionality.

If you are planning to contribute new functionality to GROMACS, we strongly encourage you to get in contact with us first at an early stage. New things can lead to exciting science, and we love that. However, the subsequent code maintenance is time-consuming and requires both "up front" and long-term commitment from you, and others who might not share your particular scientific enthusiasm. Please read the following pages first, and at least post on the developer discussion forum. Looking through the gmx-developers mailing list archive might also be helpful. Sometimes we will be able to save you a lot of time even at the planning stage!

Much of the documentation is found alongside the source code in the git repository. If you have changes to suggest there, those contributions can be done using the same mechanism as the source code contributions, and will be reviewed in similar ways.

### 8.1.1 Checklist

Before you send us your code for review and inclusion into GROMACS, please make sure that you have checked all the points on this list:

- *Usefulness*: Your code should have wide applicability within the scientific community. You are welcome to have smaller projects tracking our code, but we are not prepared to include and maintain code that will only have limited application. Evidence that people are already using your code or method is one good way to show that your code is useful. Scientific publications is another, but those publications should ideally come from several different research groups to show widespread adoption of the method.
- *Advance discussion*: Please communicate with the other developers, e.g. on the developer discussion forum, or issue tracker to let them know of the general nature of your plans. This will prevent duplicate or wasted effort. It is also a good idea to search those resources as well as the literature and WWW for other projects that may be relevant.
- *Verifiable*: If you propose a new method that passes the first check, please make sure that we can easily verify that it will be correct from a physics point of view. That must include documentation (both in the source code and as later additions to the user guide and/or reference manual) that a capable graduate student can read and understand well enough to use your method appropriately. The source code documentation will also help in maintenance and later development.

  This will be facilitated by the inclusions of unit tests for your code, as described in the section on how to write *new tests* (page 826).

  We also need some form of automated high-level test of your code, because people who do not understand its details need to be able to change the infrastructure that you depend on. GROMACS uses automated continuous-integration testing in *GitLab* (page 802), and we need quick feedback about whether your code would be affected by a proposed change. This means the users of your feature can continue to do good science based upon trustworthy results generated by new versions of GROMACS released after you have contributed your feature.
- *Structured change process*: Reviewing code for correctness, quality and performance is a very time consuming process, which we are committed to because it is necessary in order to deliver software that is of high enough quality for reliable scientific results. However, human beings are busy and have short attention spans, and a proposed change affecting 10,000 lines of code is likely to generate little enthusiasm from other developers to review it. Your local git commit history is likely full of changes that are no longer present in the version you would like to contribute, so we cannot reasonably review that, either. It might be reasonable to break the process into manageable pieces, such as
  - the functionality to read the *mdp settings* (page 51) you might require and write a *tpr* (page 586),
  - the functionality for *mdrun* (page 258) to execute the simplest form of your feature,
  - further extensions and/or optimizations for your feature, and
  - functionality for an analysis tool to do useful things with the simulation output.

  Do get in touch with us, e.g. on the developer discussion forum, to exchange ideas here.

- *Timeliness*: We make an annual release of GROMACS, with a feature freeze (and git branch fork) on a fixed date, which is agreed more than six months in advance. We still need a month or more to do quality testing on that branch, after the fork and before the release, so there is a period when we cannot accept certain kinds of potentially risky changes. (The main branch will remain open for all kinds of changes, but it is likely that the focus of many of the core developers will be on the release process.) If you have a large change to propose, you need to
  - make a group of smaller changes,
  - negotiate in advance who will do the code review, and
  - have them available for review and improvement months(!) before that date. Even smaller changes are unlikely to be prioritized by others for review in the last month or so!
- *Coding style*: Please make sure that your code follows all the *coding style* (page 771) and *code formatting* (page 771) guidelines. This will make the code review go more smoothly on both sides. There are a number of tools already included with GROMACS to facilitate this, please have a look at *the respective part of the documentation* (page 818).
- *Code documentation*: To ensure proper code documentation, please follow the instructions provided for the use of *doxygen* (page 786). In addition to this, the new functionality should be documented in the manual and possibly the user guide .
- In addition to coding style, please also follow the instructions given concerning the *commit style* (page 783). This will also facilitate the code review process.

### 8.1.2 Preparing code for submission

GROMACS uses `git` for *Change Management* (page 759). Instead of accepting "pull requests", GROMACS changes are submitted as individual commits on the tip of the `main` branch hosted at gitlab. Preparing, submitting, and managing patches for a change requires a little bit of set-up. Refer to *Change Management* (page 759) for information about

- accessing the GROMACS *git* repository
- structure of the repository
- source control without merge commits
- `git` usage that may be less common in other development work flows

### 8.1.3 Alternatives

GROMACS has a public mirror available on GitHub at https://github.com/gromacs/gromacs. You may wish to fork the project under your own GitHub account and make your feature available that way. This should help you to generate a following of users that would help make the case for contributing the feature to the core. This process would then still need to follow the remaining criteria outlined here. If you fork GROMACS, please set the CMake variable `GMX_VERSION_STRING_OF_FORK` to an appropriate descriptive string - see cmake/gmxVersionInfo.cmake for details.

There is a project underway to develop a stable API for GROMACS, which promises to be a great tool for permitting innovation while ensuring ongoing quality of the core functionality. You might prefer to plan to port your functionality to that API when it becomes available. Do keep in touch on the developer discussion forum, so you will be the first to know when such functionality is ready for people to explore!

### 8.1.4 Do you have more questions?

If you have questions regarding these points, or would like feedback on your ideas for contributing, please feel free to contact us through the developer discussion forum. If your code is of interest to the wider GROMACS community, we will be happy to assist you in the process of including it in the main source tree.

### 8.1.5 Removing functionality

This is occasionally necessary, and there is *policy for such occasions* (page 406). For users, there are also lists of anticipated changes, including deprecated functionality, in the "Major release" *notes* (page 838).

## 8.2 Codebase overview

The root directory of the GROMACS repository only contains `CMakeLists.txt` (the root file for the CMake build system), a few files supporting the build system, and a few standard informative files (`README` etc.). The `INSTALL` is generated for source packages from `docs/install-guide/index.rst`.

All other content is in the following top-level directories:

**`admin/`**
: Contains various scripts for developer use, as well as configuration files and scripts for some of the tools used.

**`api/`**
: Contains code for the installable *C++ API* (page 833).

**`cmake/`**
: Contains code fragments and find modules for CMake. Some content here is copied and/or adapted from newer versions of CMake than the minimum currently supported. Default suppression file for valgrind is also included here. See *Build system overview* (page 750) for details of the build system.

**`docs/`**
: Contains the build system logic and source code for all documentation, both user-facing and developer-facing. Some of the documentation is generated from the source code under `src/`; see *Documentation organization* (page 748). This directory also contains some developer scripts that use the Doxygen documentation for their operation.

**`scripts/`**
: Contains the templates for `GMXRC` script, some other installed scripts, as well as installation rules for all these scripts.

**`share/`**
: Contains data files that will be installed under `share/`. These include a template for writing C++ analysis tools, and data files used by GROMACS.

**`src/`**
: Contains all source code. See *Source code organization* (page 746).

**`tests/`**
: Contains build system logic for some high-level tests. Currently, only the regression test build system logic, while other tests are under `src/`.

### 8.2.1 Source code organization

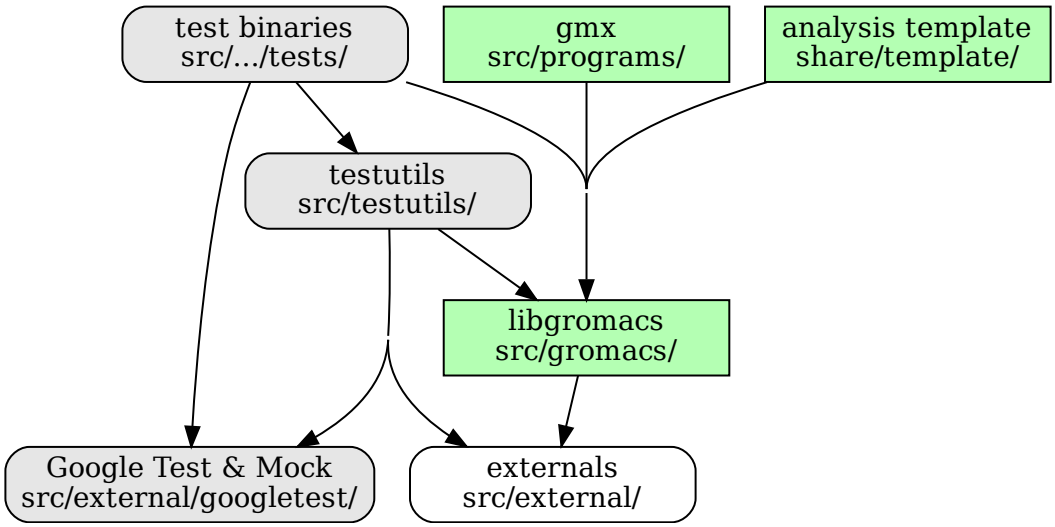
The sample code for the Trajectory Analysis Framework is in `share/template/`.

Code for the gmxapi Python package and the sample MD extension module is in `python_packaging/`.

`api/` holds code for the installable *C++ API* (page 833), including the legacy `gromacs` headers and full sources for `libgmxapi` and *(Non-)Bonded LIBrary (NB-LIB) API* (page 734).

The rest of the source code is under the `src/` directory.

The following figure shows a high-level view of components of what gets built from the source code under `src/` and how the code is organized. Arrows indicate the direction of dependencies. The build system is described in detail in *Build system overview* (page 750). With default options, the green and white components are built as part of the default target. The gray parts are for testing, and are by default only built as part of the `tests` target, but if `GMX_DEVELOPER_BUILD` is `ON`, then these are included in the default build target. See *Unit testing* (page 824) for details of the testing side.




Only a few files related to the build system are included at the root level. All actual code is in subdirectories:

**`src/gromacs/`**
: The code under this directory is built into a single library, `libgromacs`. Installed headers are also located in this hierarchy. This is the main part of the code, and is organized into further subdirectories as *modules*. See below for details.

**`src/programs/`**
: The GROMACS executable `gmx` is built from code under this directory. Also found here is some of the driver code for the `mdrun` module called by `gmx`, and numerous end-to-end tests of `gmx mdrun`.

**`src/.../tests/`**
: Various subdirectories under `src/` contain a subdirectory named `tests/`. The code from each such directory is built into a test binary. Some such directories also provide shared test code as

object libraries that is linked into multiple test binaries from different folders. See *Unit testing* (page 824) for details.

**src/testutils/**
Contains shared utility code for writing Google Test tests. See *Unit testing* (page 824) for details.

**src/external/**
Contains bundled source code for various libraries and components that GROMACS uses internally. All the code from these directories are built using our custom build rules into `libgromacs`, or in some cases into the test binaries. Some CMake options change which parts of this code are included in the build. See *Build system overview* (page 750) for some explanation about how the code in this directory is used.

**src/external/build-fftw/**
This folder contains the build system code for downloading and building FFTW to be included into `libgromacs`.

When compiling, the include search path is set to `src/` by the `legacy_modules` CMake target for many of the interfaces that do not have clearer module ownership.

Some directories from under `src/external/` may also be included, depending on the compilation options.

### Organization under `src/gromacs/`

The `libgromacs` library is built from code under `src/gromacs/`. Again, the top-level directory contains build and installation rules for the library.

The code is organized into subdirectories. These subdirectories are denoted as *modules* throughout this documentation. Each module consists of a set of routines that do some well-defined task or a collection of tasks. Many modules are represented by distinct CMake targets, and target_link_libraries() should be used to get access to the headers and linkable symbols. Other modules are only expressed by the filesystem hierarchy, and their source files are compiled directly into the monolithic `libgromacs` CMake target.

Modules under `src/gromacs/` are not part of the public installed interface. However, some of the headers that were traditionally installed have been moved `api/legacy/include` (not duplicated in `src/`) pending specification of an updated public API. These interfaces are grouped into the `legacy_api` CMake target (in the build tree), and are available through the IMPORTED `Gromacs::libgromacs` target for an installation configured with `GMX_INSTALL_LEGACY_API=ON`. They are installed into a corresponding hierarchy under `include/gromacs/` in the installation directory.

> Historically, comments at the top of the header files contain a note about their visibility: public (installed), intra-library (can be used from inside the library), or intra-module/intra-file. Many of these comments remain, but they are no longer maintained.

All headers should compile by themselves, with installed headers doing so without reference to variables defined in `config.h` or requiring other headers to be included before it. No installed headers are allowed to include `config.h`. Cyclic include dependencies prevent this, and must be avoided because of this. This is best guaranteed by including every header in some source file as the first header, even before `config.h`.

Code inside the library should not unnecessarily include headers. In particular, headers should not include other headers if a forward declaration of a type is enough for the header. Within the library source files, include only headers from other modules that are necessary for that file. Check the `CMakeLists.txt` for the module to see whether you need to target_link_libraries(). Many modules distinguish between a public interface and a private interface intended only for use inside the module implementation. In such cases, the public module headers (for use by other modules in the library) are in `src/gromacs/modulename/include/gromacs/modulename` subdirectories. Module private headers (located with the source files) may be leaked into the include path, such as through the `legacy_modules` target, but should not be used by other modules!

See *Naming conventions* (page 773) for some common naming patterns for files that can help locating declarations.

Tests, and data required for them, are in a `tests/` subdirectory under the module directory. See *Unit testing* (page 824) for more details.

### 8.2.2 Documentation organization

All documentation (including this developer guide) is produced from source files under `docs/`, except for some command-line help that is generated from the source code (by executing the compiled `gmx` binary). The build system provides various custom targets that build the documentation; see *Build system overview* (page 750) for details.

**`docs/fragments/`**
: Contains reStructuredText fragments used through `.. include::` mechanism from various places in the documentation.

#### User documentation

**`docs/install-guide/`**
: Contains reStructuredText source files for building the install guide section of the user documentation, as well as the `INSTALL` file for the source package. The build rules are in `docs/CMakeLists.txt`.

**`docs/reference-manual/`**
: Contains reStructuredText source files to generate the reference manual for html and LaTeX.

**`docs/manual/`**
: Contains LaTeX helper files to build the reference (PDF) manual.

**`docs/user-guide/`**
: Contains reStructuredText source files used to build the user guide section of the user documentation. The build rules are in `docs/CMakeLists.txt`.

**`docs/how-to/`**
: Contains reStructuredText source files building the how-to section of the user focused documentation.

### Unix man pages

Man pages for programs are generated by running the `gmx` executable after compiling it, and then using Sphinx on the reStructuredText files that `gmx` writes out.

The build rules for the man pages are in `docs/CMakeLists.txt`.

### Developer guide

**`docs/dev-manual/`**
: Contains reStructuredText source files used to build the developer guide. The build rules are in `docs/CMakeLists.txt`.

The organization of the developer guide is explained on the *front page of the guide* (page 742).

### Doxygen documentation

**`docs/doxygen/`**
: Contains the build rules and some overview content for the Doxygen documentation. See *Using Doxygen* (page 786) for details of how the Doxygen documentation is built and organized.

The Doxygen documentation is made of a few different parts. Use the list below as a guideline on where to look for a particular kind of content. Since the documentation has been written over a long period of time and the approach has evolved, not all the documentation yet follows these guidelines, but this is what we are aiming at.

**documentation pages**
: These contain mainly overview content, from general-level introduction down into explanation of some particular areas of individual modules. These are generally the place to start familiarizing with the code or a new area of the code. They can be reached by links from the main page, and also through cross-links from places in the documentation where that information is relevant to understand the context.

**module documentation**
: These contain mainly technical content, explaining the general implementation of a particular module and listing the classes, functions etc. in the module. They complement pages that describe the concepts. They can be reached from the Modules tab, and also from all individual classes, functions etc. that make up the module.

**class documentation**
: These document the usage of an individual class, and in some cases that of closely related classes. Where necessary (and time allowing), a broader overview is given on a separate page and/or in the module documentation.

**method documentation**
: These document the individual method. Typically, the class documentation or other overview content is the place to look for how different methods interact.

**file and namespace documentation**
: These are generally only placeholders for links, and do not contain much else. The main content is the list of classes and other entities declared in that file.

## 8.3 Build system overview

The GROMACS build system uses CMake (version 3.28 or newer is required) to generate the actual build system for the build tool chosen by the user. See CMake documentation for general introduction to CMake and how to use it. This documentation focuses on how the GROMACS build system is organized and implemented, and what features it provides to developers (some of which may be of interest to advanced users).

Most developers use `make` or `ninja` as the underlying build system, so there can be parts of the build system that are specifically designed for command-line consumption with these tools, and may not work ideally with other environments, but basic building should be possible with all the environments supported by CMake.

Also, the build system and version control is designed for out-of-source builds. In-source builds mostly work (there are a few custom targets that do not), but no particular effort has been put to, e.g., having `.gitignore` files that exclude all the build outputs, or to have the `clean` target remove all possible build outputs.

### 8.3.1 Build types

Build types is a CMake concept that provides overall control of how the build tools are used on the given platform to produce executable code. These can be set in CMake in various ways, including on a command line such as `cmake -DCMAKE_BUILD_TYPE=Debug`. GROMACS supports the following standard CMake build types:

**Release**
: Fully optimized code intended for use in production simulation. This is the default.

**Debug**
: Compiled code intended for use with debugging tools, with low optimization levels and debug information for symbols.

**RelWithDebInfo**
: As Release, but with debug information for symbol names, which can help debugging issues that only emerge in optimized code.

**MinSizeRel**
: As Release, but optimized to minimize the size of the resulting executable. This is never a concern for GROMACS installations, so should not be used, but probably works.

Additionally, GROMACS provides the following build types for development and testing. Their implementations can be found in `cmake/gmxBuildTypeXXX.cmake`.

**Reference**
: This build type compiles a version of GROMACS aimed solely at correctness. All parallelization and optimization possibilities are disabled. This build type is compiled with GCC 11 to generate the regression test reference values, against which all other GROMACS builds are tested.

**RelWithAssert**
: As Release, but removes `-DNDEBUG` from compiler command lines, which makes all assertion statements active (and can have other safety-related side effects in GROMACS and code upon which it depends).

**Profile**
: As Release, but adds `-pg` for use with profiling tools. This is not likely to be effective for profiling

the performance of *gmx mdrun* (page 258), but can be useful for the tools.

**TSAN**
: Builds GROMACS for use with ThreadSanitizer in GCC and Clang to detect data races. This disables the use of atomics in ThreadMPI, preferring the mutex-based implementation.

**ASAN**
: Builds GROMACS for use with AddressSanitizer in GCC and Clang to detect many kinds of memory mis-use. By default, AddressSanitizer includes LeakSanitizer (LSAN) but in many cases GROMACS suppresses leak detection either from particular functions known to leak, or in bulk.

**MSAN**
: Builds GROMACS for use with MemorySanitizer in Clang to detect reads of uninitialized memory. This functionality requires that dependencies of the GROMACS build have been built in a compatible way (roughly, static libraries with `-g -fsanitize=memory -fno-omit-frame-pointer`), which generally requires at least the C++ standard library to have been built specially. The path where the includes and libraries for dependencies should be found for this build type is set in the CMake cache variable `GMX_MSAN_PATH`. Only internal XDR and internal fftpack are supported at this time.

**UBSAN**
: Builds GROMACS for use with UndefinedBehaviorSanitizer in GCC and Clang to detect undefined behavior during execution. The checks performed during execution have a small cost and do not impact address space layout and application binary interface.

For all of the sanitizer builds, to get readable stack traces, you may need to ensure that the `ASAN_SYMBOLIZER_PATH` environment variable (or your `PATH`) includes that of the `llvm-symbolizer` binary.

With some generators, CMake generates the build system for more than a single `CMAKE_BUILD_TYPE` from one pass over the `CMakeLists.txt` files, so any code that uses `CMAKE_BUILD_TYPE` in `CMakeLists.txt` directly will break. GROMACS does use such CMake code, so we do not fully support all these build types in such generators (which includes Visual Studio).

### 8.3.2 CMake cache variables

This section provides a (currently incomplete) list of cache variables that developers or advanced users can set to affect what CMake generates and/or what will get built.

#### Compiler flags

Standard CMake mechanism for specifying the compiler flags is to use `CMAKE_C_FLAGS`/`CMAKE_CXX_FLAGS` for flags that affect all build types, and `CMAKE_C_FLAGS_`*`buildtype`*/`CMAKE_CXX_FLAGS_`*`buildtype`* for flags that only affect a specific build type. CMake provides some default flags.

GROMACS determines its own set of default flags, grouped into two categories:

- Generic flags that are appended to the above default CMake flag variables (possibly for multiple build types), generally specifying optimization flags to use and controlling compiler warnings.
- Specific flags for certain features that the build system determines to be necessary for successful compilation. One example is flags that determine what SIMD instruction set the compiler is allowed to use/needs to support.

All of the above flags are only added after testing that they work with the provided compiler.

There is one cache variable to control the behavior of automatic compiler flags:

**GMX_SKIP_DEFAULT_CFLAGS**

If set ON, the build system will not add any compiler flags automatically (neither generic nor specific as defined above), and will skip most linker flags as well. The default flags that would have been added are instead printed out when **cmake** is run, and the user can set the flags themselves using the CMake variables. If OFF (the default), the flags are added as described above.

The code the determine the default generic flags is in cmake/gmxCFlags.cmake. Code that sets the specific flags (e.g., SIMD flags) is in the main CMakeLists.txt; search for *GMX_SKIP_DEFAULT_CFLAGS* (page 752). The variables used there can be traced back to the locations where the actual flags to use are determined.

### Variables affecting compilation/linking

**GMX_BROKEN_CALLOC**

Enable emulation of calloc via malloc/memset. Only needed on machines with a broken calloc(3), e.g. in -lgmalloc on Cray XT3. Defaults to OFF, and there should not be any need to change this unless you are sure it is required.

**GMX_BUILD_FOR_COVERAGE**

Special variable set ON by CI when doing a build for the coverage job. Allows the build system to set options to produce as useful coverage metrics as possible. Currently, it disables all asserts to avoid them showing up as poor conditional coverage. Defaults to OFF, and there should not be any need to change this in a manual build.

**GMX_BUILD_OWN_FFTW**

If set ON, GROMACS build system will download and build FFTW from source automatically. Not supported on Windows or with ninja build system. In complicated scenarios (e.g., when cross-compiling or using a toolchain file), we recommend not relying on this feature and building FFTW manually.

**GMX_BUILD_SHARED_EXE**

Build executables as shared binaries. If not set, this disables -rpath and dynamic linker flags in an attempt to build a static binary, but this may require setting up the toolchain properly and making appropriate libraries available. Defaults to ON.

**GMX_COMPILER_WARNINGS**

If set ON, various compiler warnings are enabled for compilers that CI uses for verification. Defaults to OFF when building from a source tarball so that users compiling with versions not tested in CI are not exposed to our rather aggressive warning flags that can trigger a lot of warnings with, e.g., new versions of the compilers we use. When building from a git repository, defaults to ON.

**GMX_CYCLE_SUBCOUNTERS**

If set to ON, enables performance subcounters that offer more fine-grained mdrun performance measurement and evaluation than the default counters. See *Getting good performance from mdrun* (page 107) for the description of subcounters which are available. Defaults to OFF.

**GMX_ENABLE_CCACHE**

If set to ON, attempts to set up the ccache caching compiler wrapper to speed up repeated builds. The ccache executable is searched for with find_package() if CMake is being run with a compatible build type. If the executable is found and a compatible compiler is configured, CMake

launch wrapper scripts are set. If enabled, the `ccache` executable location discovered by CMake must be accessible during build, as well. Defaults to `OFF` to minimize build system complexity.

**GMX_INSTALL_DATASUBDIR**

Sets the subdirectory under CMAKE_INSTALL_DATADIR where GROMACS-specific read-only architecture-independent data files are installed. The default is `gromacs`, which means the files will go under `share/gromacs`. To alter the `share` part, change CMAKE_INSTALL_-DATADIR. See *Relocatable binaries* (page 766) for how this influences the build.

**GMX_DOUBLE**

Many part of GROMACS are implemented in terms of "real" precision, which is actually either a single- or double-precision type, according to the value of this flag. Some parts of the code deliberately use single- or double-precision types, and these are unaffected by this setting. See *Mixed or Double precision* (page 425) for further information.

**GMX_EXTERNAL_BLAS**

If not set (the default), CMake will first try to use an external BLAS library, and, if unsuccessful, fall back to using the one bundled with GROMACS. If set to `OFF`, CMake will use the bundled one immediately. If set to `ON`, CMake will use the external one, and raise an error if it is not found.

**GMX_EXTERNAL_LAPACK**

See `GMX_EXTERNAL_BLAS`.

**GMX_EXTERNAL_TNG**

Use external TNG library for trajectory-file handling. Default: `OFF`.

**GMX_FFT_LIBRARY**

Choose the CPU FFT library to use. Possible values: `fftw`, `mkl`, `fftpack`. The default selection depends on the compiler and build type.

**GMX_GIT_VERSION_INFO**

Whether to generate version information dynamically from git for each build (e.g., HEAD commit hash). Defaults to `ON` if the build is from a git repository and **`git`** is found, otherwise `OFF`. If `OFF`, static version information from `cmake/gmxVersionInfo.cmake` is used.

**GMX_GPU**

Choose the backend for GPU offload. Possible values: `CUDA`, `OpenCL`, `SYCL`. Please see the *Installation guide* (page 8) for more information.

**GMX_CLANG_CUDA**

Use clang for compiling CUDA GPU code, both host and device. Please see the *Installation guide* (page 8) for more information.

**GMX_CUDA_CLANG_FLAGS**

Pass additional CUDA-only compiler flags to clang using this variable.

**CMAKE_INSTALL_LIBDIR**

Sets the installation directory for libraries (default is determined by standard CMake package `GNUInstallDirs`). See *Relocatable binaries* (page 766) for how this influences the build.

**GMX_USE_PLUGINS**

Enable support for dynamic plugins (e.g. VMD-supported file formats). Default: `OFF`.

**GMX_MPI**

Enable MPI (not thread-MPI) support for inter-node parallelism. Defaults to OFF. Please see the *Installation guide* (page 10) for more information.

**GMX_OPENMP**

Enable OpenMP support. Default is ON.

**GMX_PREFER_STATIC_LIBS**

Prefer statically linking to external libraries. Defaults to OFF, unless GMX_BUILD_SHARED_EXE is disabled.

**GMX_SIMD**

Choose SIMD instruction set to use. Default is: Auto (best one for the current CPU). Please see the *Installation guide* (page 18) for more information.

**GMX_THREAD_MPI**

Enable thread-MPI support for intra-node parallelism. Defaults to ON.

**GMX_USE_RDTSCP**

Use low-latency RDTSCP instruction for x86 CPU-based timers for mdrun execution. Ignored on non-x86 machines. Might need to be set to OFF when compiling for for heterogeneous environments or a very old x86 CPU.

**GMX_USE_TNG**

Use the TNG library for trajectory I/O. Defaults to ON.

**GMX_USE_ITT**

Use the Intel ITT library for annotating GROMACS tasks in the Intel tracing tools. Defaults to OFF. Relies on the VTUNE_PROFILER_DIR environment variable set when loading the oneAPI toolkit to find the library.

**GMX_USE_NVTX**

Use the NVTX library for annotating GROMACS tasks in the NVIDIA tracing tools. Defaults to OFF. Relies on the CUDA_HOME environment variable to find the library.

**GMX_USE_ROCTX**

Use the ROC-TX library for annotating GROMACS tasks in the AMD ROCm tracing tools. Defaults to OFF. Relies on the ROCM_HOME environment variable to find the library.

**GMX_VMD_PLUGIN_PATH**

Path to VMD plugins for molfile I/O. Only used when GMX_USE_PLUGINS is enabled.

### Variables affecting the all target

**BUILD_TESTING**

Standard variable created by CTest that enables/disables all tests. Defaults to ON.

**GMX_BUILD_HELP**

Controls handling of man pages and shell completions. Possible values:

**OFF (default for builds from release source distribution)**
Man pages and shell completions are not generated as part of the all target, and only installed if compiling from a source package.

**AUTO (default for builds from development version)**
: Shell completions are generated by executing the `gmx` binary as part of the `all` target. If it fails, a message is printed, but the build succeeds. Man pages need to be generated manually by invoking the `man` target. Man pages and shell completions are installed if they have been successfully generated.

**ON**
: Works the same as `AUTO`, except that if invoking the `gmx` binary fails, the build fails as well.

**GMX_DEVELOPER_BUILD**

If set `ON`, the `all` target will include also the test binaries using Google Test (if *GMX_BUILD_UNITTESTS* (page 756) is `ON`), while `webpage` target will also include Reference manual in PDF format. Also, *GMX_COMPILER_WARNINGS* (page 752) and CMAKE_EXPORT_COMPILE_COMMANDS are always enabled. In the future, other developer convenience features (as well as features inconvenient for a general user) can be added to the set controlled by this variable.

**GMX_CLANG_TIDY**

clang-tidy is used for static code analysis and (some) automated fixing of issues detected. clang-tidy is easy to install. It is contained in the llvm binary package. Only version 18.0.* is supported. Others might miss tests or give false positives. It is run automatically in GitLab CI for each commit. Many checks have fixes which can automatically be applied. To run it, the build has to be configured with `cmake -DGMX_CLANG_TIDY=ON -DCMAKE_BUILD_TYPE=Debug`. Any `CMAKE_BUILD_TYPE` which enables asserts (e.g. ASAN) works. Such a configured build will run both the compiler as well as clang-tidy when building. The name of the clang-tidy executable is set with `-DCLANG_TIDY=...`, and the full path to it can be set with `-DCLANG_TIDY_EXE=...`. To apply the automatic fixes to the issues identified, clang-tidy should be run separately (running clang-tidy with `-fix-errors` as part of the build can corrupt header files). To fix a specific file run `clang-tidy -fix-errors -header-filter '.*' {file}`, to fix all files in parallel `run-clang-tidy.py -fix -header-filter '.*' '(?<!/selection/parser\.cpp|selection/scanner\.cpp)$'`, and to fix all modified files `run-clang-tidy.py -fix -header-filter '.*' $(git diff HEAD --name-only)`. The `run-clang-tidy.py` script is in the `share/clang/` subfolder of the llvm distribution. `clang-tidy` has to be able to find the `compile_commands.json` file. Either run from the build folder or add a symlink to the source folder. *GMX_ENABLE_CCACHE* (page 752) does not work with clang-tidy.

### Variables affecting special targets

**GMX_INSTALL_LEGACY_API**

Default `OFF`. If set to `ON`, headers will be installed to `gromacs/` in the CMake header destination folder to allow use of the `::gmx` C++ namespace, supported by the `libgromacs` library. See Legacy API.

**GMX_INSTALL_NBLIB_API**

If set to `ON` (default, when `BUILD_SHARED_LIBS` on non-Windows platforms), build and install the `libnb_gmx` and `nblib/` headers. See *(Non-)Bonded LIBrary (NB-LIB) API* (page 734).

**GMXAPI**

If set `ON` (default, when `BUILD_SHARED_LIBS` on non-Windows platforms), the additional `gmxapi` C++ library is configured and the `gmxapi` headers will be installed. Provides the additional build tree targets `gmxapi-cppdocs` and `gmxapi-cppdocs-dev` when Doxygen is available. Also exports CMake configuration files for `gmxapi` that allow `find_-`

`package(gmxapi)` to import the `Gromacs::gmxapi` CMake target in client projects that search the GROMACS installation root.

**GMX_BUILD_MANUAL**

If set `ON`, CMake detection for LaTeX and other prerequisites for the reference PDF manual is done, and the `manual` target for building the manual is generated. If `OFF` (the default), all detection is skipped and the manual cannot be built.

**GMX_BUILD_TARBALL**

If set `ON`, `-dev` suffix is stripped off from version strings and some other version info logic is adjusted such that the man pages and other documentation generated from this build is suitable for releasing (on the web page and/or in the source distribution package). Defaults to `OFF`.

**GMX_BUILD_UNITTESTS**

If `ON`, test binaries using Google Test are built (either as the separate `tests` target, or also as part of the `all` target, depending on *GMX_DEVELOPER_BUILD* (page 755)). All dependencies required for building the tests (Google Test and Google Mock frameworks, and tinyxml2) are included in `src/external/`. Defaults to `ON` if *BUILD_TESTING* (page 754) is `ON`.

**GMX_COMPACT_DOXYGEN**

If set `ON`, Doxygen configuration is changed to avoid generating large dependency graphs, which makes it significantly faster to run Doxygen and reduces disk usage. This is typically useful when developing the documentation to reduce the build times. Defaults to `OFF`.

**REGRESSIONTEST_DOWNLOAD**

If set `ON`, CMake will download the regression tests and extract them to a local directory. *REGRESSIONTEST_PATH* (page 756) is set to the extracted tests. Note that this happens during the configure phase, not during the build. After the download is done, the variable is automatically reset to `OFF` again to avoid repeated downloads. Can be set to `ON` to download again. Defaults to `OFF`.

**REGRESSIONTEST_PATH**

Path to extracted regression test suite matching the source tree (the directory containing `gmxtest.pl`) If set, CTest tests are generated to run the regression tests. Defaults to empty.

**SOURCE_MD5SUM**

Sets the MD5 sum of the release tarball when generating the HTML documentation. It gets inserted into the download section of the HTML pages.

### 8.3.3 External libraries

### 8.3.4 Special targets

In addition to the default `all` target, the generated build system has several custom targets that are intended to be explicitly built to perform various tasks (some of these may also run automatically). There are various other targets as well used internally by these, but those are typically not intended to be invoked directly.

**check**

Builds all the binaries needed by the tests and runs the tests. If some types of tests are not available, shows a note to the user. This is the main target intended for normal users to run the tests. See *Unit testing* (page 824).

**check-source**
Runs a custom Python checker script to check for various source-level issues. Uses Doxygen XML documentation as well as rudimentary parsing of some parts of the source files. This target is used as part of the CI. All CMake code is currently in `docs/doxygen/`. See *Source tree checker scripts* (page 815).

**completion**
Runs the compiled `gmx` executable to generate shell command-line completion definitions. This target is only added if *GMX_BUILD_HELP* (page 754) is not `OFF`, and it is run automatically as part of the default `all` target. See *GMX_BUILD_HELP* (page 754). All CMake code is in `src/programs/`.

**dep-graphs***
Builds include dependency graphs for the source files using **`dot`** from graphviz. All CMake code is in `docs/doxygen/`. See *Source tree checker scripts* (page 815).

**doxygen-***
Targets that run Doxygen to generate the documentation. The `doxygen-all` target runs as part of the `webpage` target, which in turn runs as part of the CI. All CMake code is in `docs/doxygen/`. See *Using Doxygen* (page 786).

**gmxapi-cppdocs**
Builds API documentation for gmxapi. Useful to authors of client software. Documentation is generated in `docs/api-user` in the build directory.

**gmxapi-cppdocs-dev**
Extract documentation for gmxapi and GROMACS developers to `docs/api-dev`.

**install-guide**
Runs Sphinx to generate a plain-text INSTALL file for the source package. The files is generated at `docs/install-guide/text/`, from where it gets put at the root of the source package by CPack. All CMake code is in `docs/`.

**man**
Runs Sphinx to generate man pages for the programs. Internally, also runs the compiled `gmx` executable to generate the input files for Sphinx. All CMake code is in `docs/`. See *GMX_BUILD_HELP* (page 754) for information on when the man pages are installed.

**manual**
Runs LaTeX to generate the reference PDF manual. All CMake code is in `docs/manual/`. See *GMX_BUILD_MANUAL* (page 756).

**package_source**
Standard target created by CPack that builds a source package. This target is used to generate the released source packages.

**test**
Standard target created by CTest that runs all the registered tests. Note that this does not build the test binaries, only runs them, so you need to first ensure that they are up-to-date. See *Unit testing* (page 824).

**tests**
Builds all the binaries needed by the tests (but not `gmx`). See *Unit testing* (page 824).

**webpage**
Collection target that runs the other documentation targets to generate the full set of HTML (and linked) documentation. This target is used as part of the CI. All CMake code is in `docs/`.

**webpage-sphinx**
Runs Sphinx to generate most content for the HTML documentation (the set of web pages this developer guide is also part of). Internally, also runs the compiled `gmx` executable to generate some input files for Sphinx. All CMake code is in `docs/`.

### 8.3.5 Passing information to source code

The build system uses a few different mechanisms to influence the compilation:

- On the highest level, some CMake options select what files will be compiled.
- Some options are passed on the compiler command line using `-D` or equivalent, such that they are available in every compilation unit. This should be used with care to keep the compiler command lines manageable. You can find the current set of such defines with

```
git grep add_definitions
```

- A few header files are generated using CMake `configure_file()` and included in the desired source files. These files must exist for the compilation to pass. Only a few files use an `#ifdef HAVE_CONFIG_H` to protect against inclusion in case the define is not set; this is used in files that may get compiled outside the main build system.

**`buildinfo.h`**
Contains various strings about the build environment, used mainly for outputting version information to log files and when requested.

**`config.h`**
Contains defines for conditional compilation within source files.

**`gmxpre-config.h`**
Included by `gmxpre.h` as the first thing in every source file. Should only contain defines that are required before any other header for correct operation. For example, defines that affect the behavior of system headers fall in this category. See Doxygen documentation for `gmxpre.h`.

The above files are available through the INTERFACE_INCLUDE_DIR of the `common` CMake target. I.e. to `#include "config.h"`, be sure to `target_link_libraries(mymodule PRIVATE common)`

Additionally, the following file is generated by the build system:

**`baseversion-gen.cpp`**
Provides definitions for declarations in `baseversion_gen.h` for version info output. The contents are generated either from Git version info, or from static version info if not building from a git repository.

## 8.4 Change Management

This documentation assumes the reader is already familiar with using `git` for managing file revisions.

### 8.4.1 Getting started

GROMACS development happens on gitlab at https://gitlab.com/gromacs/gromacs. Create a user account at https://gitlab.com/users/sign_in#register-pane or use an existing account at gitlab.com. For more information on how to use gitlab have a look at their extensive user documentation at https://docs.gitlab.com/ee/user/index.html.

If you do not already have a GROMACS repository set up, use `git clone git@gitlab.com:gromacs/gromacs.git` to obtain the current GROMACS repository from gitlab. Otherwise use `git remote add gitlab git@gitlab.com:gromacs/gromacs.git`.

Using gitlab, new code enters GROMACS by merging git development branches into the main branch.

To automatically detect issues in new code, it is tested within continuous integration (CI) with a large combination of settings. See *Automatic source code formatting* (page 818) for help meeting and testing the style guidelines.

More information about change management is available on the gitlab wiki.

#### Setting up login credentials with gitlab

You will need a public ssh key:

```
ssh-keygen -t rsa -C "your.email@address.com"
cat ~/.ssh/id_rsa.pub
```

Copy the output of the last command, go to gitlab.com, click your user symbol in the right top corner of the panel to the left, and select Preferences.

Choose SSH keys in the menu on the left and paste your key in the text field.

#### Creating issues

The meta-level code design and discussions is organised in issues and visible at https://gitlab.com/gromacs/gromacs/-/issues. Please check if your issue, or a similar issue, already exists before creating a new one. See *Guidelines for creating meaningful issue reports* (page 781) for more information.

Note that all Redmine issues have been transferred to gitlab retaining the original issue number from Redmine. However, comments and discussion are now represented by gitlab user @acmnpv - the original authors are found inline at the bottom of the comments.

### Uploading code for review - creating a merge request

Issues are addressed with new code via "merge requests" (MR). Find the current MRs at https://gitlab.com/gromacs/gromacs/-/merge_requests. There are two ways of creating a merge request - either via the gitlab graphical user interface or via the command line.

To use the GUI, find the relevant issue or open a new one, then find the "create merge request" button to create a merge request related to that issue in gitlab. We recommend using the "Mark as draft" option until you are completely satisfied with the code yourself and all tests are passed.

Select milestone and assignees to make tracking of the progress easier. Keep the requirements for merging as they are set by default.

You can also use `git push` on the command line directly and create a merge request following the link that is output on the command line.

Your repository should be in sync with the GROMACS repository. To ensure this, use `git fetch` to obtain the newest branches, then merge the main branch into your branch with `git merge main` while on your branch.

You can create MRs without having a `Developer` role account, but in order to trigger MR test pipelines you must be a project member with (at least) the `Developer` role. Project membership can be requested via the Gitlab web interface. The `Developer` role is granted based on acquaintance and/or knowledge about intentions to contribute to the project.

As mentioned in *Code Review* (page 761), the `Developer` role is different to membership in the `GMX Developers` approval group. Approving and merging changes require higher privileges than what is required for counting as a `Developer`.

### Naming branches

Good names: documentation_UpdateDevelopersDocsTOGitLab, nbnxm_MakeNbnxmGPUIntoClass, pme_FEPPMEGPU, 1234-ml-fix-issue.

Bad names: branch1234, mybranch, test, etc

N.b., prefixing the branch with an issue number (such as 1234-ml-fix-issue above), followed by a hyphen, automatically links the branch to the issue. Milestones and labels are copied and the issue will be closed when the MR is merged, see the Gitlab documentation.

### Documentation

Contributors and reviewers frequently overlook the effects of changes on the built documentation. Contributors and reviewers should note that the build artifacts from the automated test jobs are available for download through the GitLab CI web interface (`webpage:build` job artifacts). For earlier review or alternative preferences, consider building and sharing a Docker image containing the built documentation. See docs/docs.dockerfile in the source tree.

### 8.4.2 Labels

Labels help developers by allowing additional filtering of issues and merge requests.

The GROMACS project defines many labels.

To minimize duplicated documentation, refer to the GitLab project Labels web interface for label descriptions.

When creating a new label, please provide a short description so that people can understand what the label is intended to convey, and when they should apply it to their own issues or merge requests.

In general:

- Ongoing categorizations to help specify the GROMACS component or development area use the `#7F8C8D` color.
- Specific features or subproject areas targeting an upcoming release use the `#8E44AD` background color.
- Status labels use `#428BCA`. Note that Status labels are also used for Issues, and are used according to *status label guidelines* (page 762)

### 8.4.3 Code Review

#### Reviewing someone else's uploaded code

The reviewing workflow is the following:

1. https://gitlab.com/gromacs/gromacs/-/merge_requests shows all open changes
2. A change needs two approvals to go in, of which one approval has to come from a member of either `GMX Core` or `GMX Developers` approval groups.
3. Usually a patch goes through several cycles of voting, commenting and updating before it becomes merged, with votes from the developers indicating if they think that the change has progressed enough to be included.
4. A change is submitted for merging and post-submit testing by clicking "Merge".

Do not review your own code. The point of the policy is that at least two non-authors have approved, and that the issues are resolved in the opinion of the person who applies an approval before a merge. If you have uploaded a minor fix to someone else's patch, use your judgement in whether to approve yourself.

Membership in the `GMX Developers` and `GMX Core` approval groups are based on long-term engagement in the project and does not correlate directly with a `Developer` role account.

### Guide for reviewing

- First and foremost, check correctness to the extent possible;
- As portability and performance are the next most important things, do check for potential issues;
- Check adherence to the *coding standards* (page 771);
- We should try to ensure that commits that implement bugfixes (as well as important features and tasks) get an issue tracker entry created and linked. The linking is done **automatically** through special syntax
- If the commit is a **bugfix**:
  - if present in the issue tracker, it has to contain a valid reference to the issue;
  - if it is a **major bug**, there has to be a bug report filed in the issue tracker (with urgent or immediate priority) and referenced appropriately.
- If the commit is a **feature/task** implementation:
  - if it is present in the issue tracker it has to contain a valid reference to the issue;
  - If no current issue is currently present and the change would benefit of one for future explanation on why it was added, a new issue should be created.

### Update the Status label

- Please update the Status label *for the issue* (page 782) when a merge request is under review.
- Please update the Status label *for the merge request* (page 762) when it is closed.

### Closing Merge Requests

A merge request that has had no updates for six months or more can acquire the status label "Status::Stale" If the proposed change still seems important and the next steps are unclear, contributors with stale issues *are encouraged...*

- to contact existing reviewers (or potential reviewers),
- to participate in the developer discussion forum, and
- to attend the biweekly teleconference to coordinate.

If the future of the merge request has not become clear within a month (especially if it has become stale multiple times), developers may close the merge request with a label indicating why it has entered a "closed" state. "Status::MR::..." labels do not indicate that the merge request has been reviewed unless it is explicitly rejected.

See Issue 4126 for background discussion.

- Status::MR::Inactive: No response from contributor or no reviewers available for over six months.
- Status::MR::Superseded: This merge request is no longer necessary.
- Status::MR::Rejected: The solution (or its associated issue) will not be accepted.
- Status::MR::Needs discussion: More discussion must take place at the tracked issue before a MR is opened.

- Status::Stale: No activity for over six months.

**See also**

*General issue workflow* (page 782) for use of Status labels in Issue management.

### 8.4.4 More git tips

#### Q: Are there some other useful git configuration settings?

A: If you need to work with branches that have large differences (in particular, if a lot of files have moved), it can be helpful to set

```
git config diff.renamelimit 5000
```

to increase the limit of inexact renames that Git considers. The default value is not sufficient, for example, if you need to do a merge or a cherry-pick from a release branch to main.

#### Q: How do I use git rebase (also `git pull --rebase`)?

A: Assume you have a local feature branch checked out, that it is based on main, and main has gotten new commits. You can then do

```
git rebase main
```

to move your commits on top of the newest commit in main. This will save each commit you did, and replay them on top of main. If any commit results in conflicts, you need to resolve them as usual (including marking them as resolved using git add), and then use

```
git rebase --continue
```

Note that unless you are sure about what you are doing, you should not use any commands that create or delete commits (git commit, or git checkout or git reset without paths). `git rebase --continue` will create the commit after conflicts have been resolved, with the original commit message (you will get a chance to edit it).

If you realize that the conflicts are too messy to resolve (or that you made a mistake that resulted in messy conflicts), you can use

```
git rebase --abort
```

to get back into the state you started from (before the original git rebase main invocation). If the rebase is already finished, and you realize you made a mistake, you can get back where you started with (use git log <my-branch>@{1} and/or git reflog <my-branch> to check that this is where you want to go)

```
git reset --hard <my-branch>@{1}
```

### Q: How do I prepare several commits at once?

A: Assume I have multiple independent changes in my working tree. Use

```
git add [-p] [file]
```

to add one independent change at a time to the index. Use

```
git diff --cached
```

to check that the index contains the changes you want. You can then commit this one change:

```
git commit
```

If you want to test that the change works, use to temporarily store away other changes, and do your testing.

```
git stash
```

If the testing fails, you can amend your existing commit with `git commit --amend`. After you are satisfied, you can push the commit for review. If you stashed away your changes and you want the next change to be reviewed independently, do

```
git reset --hard HEAD^
git stash pop
```

(only do this if you pushed the previous change upstream, otherwise it is difficult to get the old changes back!) and repeat until each independent change is in its own commit. If you skip the `git reset --hard` step, you can also prepare a local feature branch from your changes.

### Q: How do I edit an earlier commit?

A: If you want to edit the latest commit, you can simply do the changes and use

```
git commit --amend
```

If you want to edit some other commit, and commits after that have not changed the same lines, you can do the changes as usual and use

```
git commit --fixup <commit>
```

or

```
git commit --squash <commit>
```

where <commit> is the commit you want to change (the difference is that `--fixup` keeps the original commit message, while `--squash` allows you to input additional notes and then edit the original commit message during `git rebase -i`). You can do multiple commits in this way. You can also mix `--fixup`/`--squash` commits with normal commits. When you are done, use

```
git rebase -i --autosquash <base-branch>
```

to merge the `--fixup/--squash` commits to the commits they amend. See separate question on `git rebase -i` on how to choose <base-branch>.

In this kind of workflow, you should try to avoid to change the same lines in multiple commits (except in `--fixup/--squash` commits), but if you have already changed some lines and want to edit an earlier commit, you can use

```
git rebase -i <base-branch>
```

but you likely need to resolve some conflicts later. See `git rebase -i` question later.

### Q: How do I split a commit?

A: The instructions below apply to splitting the HEAD commit; see above how to use `git rebase -i` to get an earlier commit as HEAD to split it.

The simplest case is if you want to split a commit A into a chain A'-B-C, where A' is the first new commit, and contains most of the original commit, including the commit message. Then you can do

```
git reset -p HEAD^ [-- <paths>]
git commit --amend
```

to selectively remove parts from commit A, but leave them in your working tree. Then you can create one or more commits of the remaining changes as described in other tips.

If you want to split a commit A into a chain where the original commit message is reused for something else than the first commit (e.g., B-A'-C), then you can do

```
git reset HEAD^
```

to remove the HEAD commit, but leave everything in your working tree. Then you can create your commits as described in other tips. When you come to a point where you want to reuse the original commit message, you can use

```
git reflog
```

to find how to refer to your original commit as `HEAD@{n}`, and then do

```
git commit -c HEAD@{n}
```

### Q: How do I use git rebase -i to only edit local commits?

A: Assume that you have a local feature branch checked out, this branch has three commits, and that it is based on main. Further, assume that main has gotten a few more commits after you branched off. If you want to use `git rebase -i` to edit your feature branch (see above), you probably want to do

```
git rebase -i HEAD~3
```

followed by a separate

```
git rebase main
```

The first command allows you to edit your local branch without getting conflicts from changes in main. The latter allows you to resolve those conflicts in a separate rebase run. If you feel brave enough, you can also do both at the same time using

```
git rebase -i main
```

## 8.5 Relocatable binaries

GROMACS (mostly) implements the concept of relocatable binaries, i.e., that after initial installation to `CMAKE_INSTALL_PREFIX` (or binary packaging with CPack), the whole installation tree can be moved to a different folder and GROMACS continues to work without further changes to the installation tree. This page explains how this is implemented, and the known limitations in the implementation. This information is mainly of interest to developers who need to understand this or change the code, but it can also be useful for people installing or packaging GROMACS.

A related feature that needs to be considered in all the code related to this is that the executables should work directly when executed from the build tree, before installation. In such a case, the data files should also be looked up from the source tree to make development easy.

### 8.5.1 Finding shared libraries

If GROMACS is built with dynamic linking, the first part of making the binaries relocatable is to make it possible for the executable to find `libgromacs`, no matter how it is executed. On platforms that support a relative RPATH, this is used to make the GROMACS executables find the `libgromacs` from the same installation prefix. This makes the executables fully relocatable when it comes to linking, as long as the relative folder structure between the executables and the library is kept the same.

If the RPATH mechanism does not work, `GMXRC` also adds the absolute path to the `libgromacs` installed with it to `LD_LIBRARY_PATH`. On platforms that support this, this makes the linker search for the library here, but it is less robust, e.g., when mixing calls to different versions of GROMACS. Note that `GMXRC` is currently not relocatable, but hardcodes the absolute path.

On native Windows, DLLs are not fully supported; it is currently only possible to compile a DLL with MinGW, not with Visual Studio or with Intel compilers. In this case, the DLLs are placed in the `bin/` directory instead of `lib/` (automatically by CMake, based on the generic binary type assignment in `CMakeLists.txt`). Windows automatically searches DLLs from the executable directory, so the correct DLL should always be found.

For external libraries, standard CMake linking mechanisms are used and RPATH for the external dependencies is included in the executable; on Windows, dynamic linking may require extra effort to make the loader locate the correct external libraries.

To support executing the built binaries from the build tree without installation (critical for executing tests during development), standard CMake mechanism is used: when the binaries are built, the RPATH is set to the build tree, and during installation, the RPATH in the binaries is rewritten by CMake to the final (relative) value. As an extra optimization, if the installation tree has the same relative folder structure as the build tree, the final relative RPATH is used already during the initial build.

The RPATH settings are in the root `CMakeLists.txt`. It is possible to disable the use of RPATH during installation with standard CMake variables, such as setting `CMAKE_SKIP_INSTALL_RPATH=ON`.

### 8.5.2 Finding data files

The other, GROMACS-specific part, of making the binaries relocatable is to make them able to find data files from the installation tree. Such data files are used for multiple purposes, including showing the quotes at the end of program execution. If the quote database is not found, the quotes are simply not printed, but other files (mostly used by system preparation tools like *gmx pdb2gmx* (page 281) and *gmx grompp* (page 230), and by various analysis tools for static data) will cause fatal errors if not found.

There are several considerations here:

- For relocation to work, finding the data files cannot rely on any hard-coded absolute path, but it must find out the location of the executing code by inspecting the system. As a fallback, environment variables or such set by `GMXRC` or similar could be used (but currently are not).
- When running executables from the build tree, it is desirable that they will automatically use the data files from the matching source tree to facilitate easy testing. The data files are not copied into the build tree, and the user is free to choose any relative locations for the source and build trees. Also, the data files are not in the same relative path in the source tree and in the installation tree (the source tree has `share/top/`, the installation tree `share/gromacs/top/`; the latter is customizable during CMake configuration).
- In addition to GROMACS executables, programs that link against `libgromacs` need to be able to find the data files if they call certain functions in the library. In this case, the executable may not be in the same directory where GROMACS is. In case of static linking, no part of the code is actually loaded from the GROMACS installation prefix, which makes it impossible to find the data files without external information.
- The user can always use the `GMXLIB` environment variable to provide alternative locations for the data files, but ideally this should never be necessary for using the data files from the installation.

Not all the above considerations are fully addressed by the current implementation, which works like this:

1. It finds the path to the current executable based on `argv[0]`. If the value contains a directory, this is interpreted as absolute or as relative to the current working directory. If there is no directory, then a file by that name is searched from the directories listed in `PATH`. On Windows, the current directory is also searched before `PATH`. If a file with a matching name is found, this is used without further checking.
2. If the executable is found and is a symbolic link, the symbolic links are traversed until a real file is found. Note that links in the directory name are not resolved, and if some of the links contain relative paths, the end result may contain `..` components and such.
3. If an absolute path to the executable was found, the code checks whether the executable is located in the build output directory (using `stat()` or similar to account for possible symbolic links in the directory components). If it is, then the hard-coded source tree location is returned.
4. If an absolute path to the executable was found and it was not in the build tree, then all parent directories are checked. If a parent directory contains `share/`*`gromacs`*`/top/gurgle.dat`, this directory is returned as the installation prefix. The file name `gurgle.dat` and the location are considered unique enough to ensure that the correct directory has been found. The installation directory for read-only architecture-independent data files can be customized during CMake configuration by setting `CMAKE_INSTALL_DATADIR`, and the subdirectory under this that hosts the GROMACS-specific data is set by `GMX_INSTALL_DATASUBDIR`.

   Note that this search does not resolve symbolic links or normalize the input path beforehand: if there are `..` components *and* symbolic links in the path, the search may proceed to unexpected

directories, but this should not be an issue as the correct installation prefix should be found before encountering such symbolic links (as long as the `bin/` directory is not a symbolic link).

5. If the data files have not been found yet, try a few hard-coded guesses (like the original installation `CMAKE_INSTALL_PREFIX` and `/usr/local/`). The first guess that contains suitable files (`gurgle.dat`) is returned.
6. If still nothing is found, return `CMAKE_INSTALL_PREFIX` and let the subsequent data file opening fail.

The above logic to find the installation prefix is in `src/gromacs/commandline/cmdlineprogramcontext.cpp`. Note that code that links to `libgromacs` can provide an alternative implementation for `gmx::IProgramContext` for locating the data files, and is then fully responsible of the above considerations.

Information about the used data directories is printed into the console output (unless run with `-quiet`), as well as to (some) error messages when locating data files, to help diagnosing issues.

There is no mechanism to disable this probing search or affect the process during compilation time, except for the CMake variables mentioned above.

### 8.5.3 Known issues

- `GMXRC` is not relocatable: it hardcodes the absolute installation path in one assignment within the script, which no longer works after relocation. Contributions to get rid of this on all the shells the `GMXRC` currently supports are welcome.
- There is no version checking in the search for the data files; in case of issues with the search, it may happen that the installation prefix from some other installation of GROMACS is returned instead, and only cryptic errors about missing or invalid files may reveal this.
- If the searching for the installation prefix is not successful, hard-coded absolute guesses are used, and one of those returned. These guesses include the absolute path in `CMAKE_INSTALL_PREFIX` used during compilation of `libgromacs`, which will be incorrect after relocation.
- The search for the installation prefix is based on the locating the executable. This does not work for programs that link against `libgromacs`, but are not installed in the same prefix. For such cases, the hard-coded guesses will be used, so the search will not find the correct data files after relocation. The calling code can, however, programmatically provide the GROMACS installation prefix, but ideally this would work without offloading work to the calling code.
- One option to (partially) solve the two above issues would be to use the `GMXDATA` environment variable set by `GMXRC` as the fallback (currently this environment variable is set, but very rarely used).
- Installed `pkg-config` files are not relocatable: they hardcode the absolute installation path.

## 8.6 Documentation generation

### 8.6.1 Building the GROMACS documentation

For now, there are multiple components, formats and tools for the GROMACS documentation, which is aimed primarily at version-specific deployment of the complete documentation on the website and in the release tarball.

This is quite complex, because the dependencies for building the documentation must not get in the way of building the code (particularly when cross-compiling), and yet the code must build and run in order for some documentation to be generated. Also, man page documentation (and command-line completions) must be built from the wrapper binary, in order to be bundled into the tarball. This helps ensure that the functionality and the documentation remain in sync.

The outputs of interest to most developers are generally produced in the `docs/html/` subdirectory of the build tree.

You need to enable at least some of the following CMake options:

**GMX_BUILD_MANUAL**
: Option needed for trying to build the PDF reference manual (requires LaTeX and ImageMagick). See *GMX_BUILD_MANUAL* (page 756).

**GMX_BUILD_HELP**
: Option that controls 1) whether shell completions are built automatically, and 2) whether built man pages are installed if available (the user still needs to build the `man` target manually before installing). See *GMX_BUILD_HELP* (page 754).

To include the full Python package documentation with the `webpage` CMake target (see below), also configure CMake with `GMX_PYTHON_PACKAGE=ON`, and install Python package dependencies from `python_packaging/gmxapi/requirements.txt`:

```
pip install -r python_packaging/gmxapi/requirements.txt
```

Some documentation cannot be built when cross-compiling, as it requires executing the `gmx` binary.

The following make targets are the most useful:

**manual**
: Builds the PDF reference manual.

**man**
: Makes man pages from the wrapper binary with Sphinx.

**doxygen-all**
: Makes the code documentation with Doxygen.

**install-guide**
: Makes the INSTALL file for the tarball with Sphinx.

**webpage-sphinx**
: Makes all the components of the GROMACS webpage that require Sphinx, including install guide and user guide.

**webpage**
: Makes the complete GROMACS webpage, requires everything. When complete, you can browse `docs/html/index.html` to find everything.

If built from a release tarball, the `SOURCE_MD5SUM`, `SOURCE_TARBALL`, `REGRESSIONTESTS_MD5SUM`, and `REGRESSIONTESTS_TARBALL` CMake variables can be set to pass in the md5sum values and names of those tarballs, for embedding into the final deployment to the GROMACS website.

### 8.6.2 Needed build tools

The following tools are used in building parts of the documentation. Make sure they are installed *before* configuring the build system with CMake.

**Doxygen**

Doxygen is used to extract documentation from source code comments. Also some other overview content is laid out by Doxygen from Markdown source files. Currently, version 1.8.5 is required for a warning-free build. Thorough explanation of the Doxygen setup and instructions for documenting the source code can be found on a separate page: *Using Doxygen* (page 786).

**graphviz (dot)**

The Doxygen documentation uses `dot` from graphviz for building some graphs. The tool is not mandatory, but the Doxygen build will produce warnings if it is not available, and the graphs are omitted from the documentation.

**mscgen**

The Doxygen documentation uses mscgen for building some graphs. As with `dot`, the tool is not mandatory, but not having it available will result in warnings and missing graphs.

**Doxygen issue checker**

Doxygen produces warnings about some incorrect uses and wrong documentation, but there are many common mistakes that it does not detect. GROMACS uses an additional, custom Python script to check for such issues. This is most easily invoked through a `check-source` target in the build system. The script also checks that documentation for a header matches its use in the source code (e.g., that a header documented as internal to a module is not actually used from outside the module). These checks are run in CI. Details for the custom checker are on a separate page (common for several checkers): *Source tree checker scripts* (page 815).

**module dependency graphs**

GROMACS uses a custom Python script to generate an annotated dependency graph for the code, showing #include dependencies between modules. The generated graph is embedded into the Doxygen documentation: Module dependency graph This script shares most of its implementation with the custom checkers, and is documented on the same page: *Source tree checker scripts* (page 815).

**Sphinx**

Sphinx; at least version 4.0.0 is used for building some parts of the documentation from reStructuredText source files. To install an appropriate version of `sphinx-build` and other required Python packages, you can use the `requirements.txt` file in the `docs` repository directory. E.g. **`pip install -r docs/requirements.txt`**

**LaTeX**

Also requires ImageMagick for converting graphics file formats.

**linkchecker**

linkchecker is used together with the `docs/linkcheckerrc` file to ensure that all the links in the documentation can be resolved correctly.

**documentation exported from source files**
: For man pages, HTML documentation of command-line options for executables, and for shell completions, the `gmx` binary has explicit C++ code to export the information required. The build system provides targets that then invoke the built `gmx` binary to produce these documentation items. The generated items are packaged into source tarballs so that this is not necessary when building from a source distribution (since in general, it will not work in cross-compilation scenarios). To build and install these from a git distribution, explicit action is required. See Doxygen documentation on the wrapper binary for some additional details.

## 8.7 Style guidelines

Different style guidelines are available under the respective sections of this page.

### 8.7.1 Guidelines for code formatting

Python code complies with PEP 8, but using the stricter Black code style.

The following list provides the general formatting/indentation rules for C++ GROMACS code:

- Basic indentation is four spaces.
- Keep lines at a reasonable length. Keep every line at least below 120 characters. If you end up indenting very deeply, consider splitting the code into functions.
- Do not use tabs, only spaces. Most editors can be configured to generate spaces even when pressing tab. Tabs (in particular when mixed with spaces) easily break indentation in contexts where settings are not exactly equal (e.g., in `git diff` output).
- No trailing whitespace.
- Use braces always for delimiting blocks, even when there is only a single statement in an `if` block or similar.
- Put braces on their own lines. The only exception is short one-line inline functions in C++ classes, which can be put on a single line.
- Use spaces liberally.
- `extern "C"` and `namespace` blocks are not indented, but all others (including `class` and `switch` bodies) are. Namespace blocks have to have a closing comment with the name of it.

Additionally:

- All source files and other non-trivial scripts should contain a copyright header with a predetermined format and license information (check existing files). Copyright holder should be "the GROMACS development team" for the years where the code has been in the GROMACS source repository, but earlier years can hold other copyrights.
- Whenever you update a file, you should check that the current year is listed as a copyright year.

Most of the above guidelines are enforced using clang-format, an automatic source code formatting tool. The copyright guidelines are enforced by a separate Python script. See *Automatic source code formatting* (page 818) for details. Note that due to the nature of those scripts (they only do all-or-nothing formatting), all the noted formatting rules are enforced at the same time.

Enforcing a consistent formatting has a few advantages:

- No one needs to manually review code for most of these formatting issues, and people can focus on content.
- A separate automatic script (see below) can be applied to re-establish the formatting after refactoring like renaming symbols or changing some parameters, without needing to manually do it all.

Many IDEs will detect `.clang-format` configuration files and be able to format the code automatically. However, clang-format behavior is very version-dependent, so there still might be some minor differences from what is enforced by our scripts and automated testing system.

### 8.7.2 Guidelines for #include directives

The following include order is used in GROMACS and enforced by `clang-format`. An empty line should appear between each group, and headers within each group sorted alphabetically.

1. Each *source* file should include `gmxpre.h` first.
2. If a *source* file has a corresponding header, it should be included next. If the header is in the same directory as the source, then it is included without any path (i.e., relative to the source). Otherwise, the canonical include path of `libraryname/modulename/header.h` is used.
3. If the file depends on defines from `config.h`, that comes next.
4. This is followed by standard C/C++ headers, grouped as follows:
    1. Standard C headers (e.g., `<stdio.h>`)
    2. C++ versions of the above (e.g., `<cstdio>`)
    3. Standard C++ headers (e.g., `<vector>`)

    Preferably, only one of the first two groups is present, but this is not enforced.
5. This is followed by other system headers: platform-specific headers such as `<unistd.h>`, as well as external libraries such as `<gtest/gtest.h>`.
6. GROMACS-specific libraries from `src/external/`, such as `"thread_mpi/threads.h"`.
7. GROMACS headers that are not part of the including module.
8. Public GROMACS headers that are part of the including module.
9. Finally, GROMACS headers that are internal to the including module, executable, or test target (typically at the same path as the source file).

All GROMACS headers are included with quotes (`"gromacs/utility/path.h"`), other headers with angle brackets (`<stdio.h>`). Headers under `src/external/` are generally included with quotes (whenever the include path is relative to `src/`, as well as for thread-MPI and TNG), but larger third-party entities are included as if they were provided by the system. The latter group currently includes gtest/gmock.

In some cases, the include paths available to build targets may leak visibility of headers inappropriately. This is usually encountered as a header that can be used by an `#include` with an unusual or long path. If a header cannot be included as described above, check that the appropriate CMake target is referenced by a target_link_libraries() command. Many modules provide their own CMake target. Additionally, note

- The `common` CMake target provides access to `gmxpre.h`, `config.h`, `gmxpre-config.h`, `buildinfo.h`, and `contributors.h`

- `legacy_api` provides access to those of the old `gromacs/modulename` headers that are in `api/legacy/include`
- `legacy_modules` adds `src/` to the include path, exposing all headers in `gromacs/` and `gromacs/*/` for `#include` lines that would appear to comply with the guidelines above, but which may not be intended for "public" use. (This target was intended as a temporary measure while working towards Issue 3288.)

If there are any conditionally included headers (typically, only when some #defines from `config.h` are set), these should be included at the end of their respective group. Note that the automatic checker/sorter script does not act on such headers, nor on comments that are between #include statements; it is up to the author of the code to put the headers in proper order in such cases. Trailing comments on the same line as #include statements are preserved and do not affect the checker/sorter.

As part of the effort to build a proper API, a new scheme of separating between public, library and module functionality in header files is planned. See also *Source tree checker scripts* (page 815) and API restructuring issues for details.

Enforcing a consistent order and style has a few advantages:

- It makes it easy at a quick glance to find the dependencies of a file, without scanning through a long list of unorganized #includes.
- Including the header corresponding to the source file first makes most headers included first in some source file, revealing potential problems where headers would not compile unless some other header would be included first. With this order, the person working on the header is most likely to see these problems instead of someone else seeing them later when refactoring unrelated code.
- Consistent usage of paths in `#include` directives makes it easy to use `grep` to find all uses of a header, as well as all include dependencies between two modules.
- An automatic script can be used to re-establish clean code after semi-automatic refactoring like renaming an include file with `sed`, without causing other unnecessary changes.

### 8.7.3 Naming conventions

The conventions here should be applied to all new code, and with common sense when modifying existing code. For example, renaming a widely used, existing function to follow these conventions may not be justified unless the whole code is getting a rework.

Currently, this only documents the present state of the code: no particular attempt has been made to consolidate the naming.

#### Files

- C++ source files have a `.cpp` extension, C source files `.c`, and headers for both use `.h`.
- For source file *`file`*`.c`/*`file`*`.cpp`, declarations that are visible outside the source file should go into a correspondingly named header: *`file`*`.h`. Some code may deviate from this rule to improve readability and/or usability of the API, but this should then be clearly documented.

  There can also be a *`file`*`_impl.h` file that declares classes or functions that are not accessible outside the module. If the whole file only declares symbols internal to the module, then the `_impl.h` suffix is omitted.

In most cases, declarations that are not used outside a single source file are in the source file.

- Use suffix `-doc.h` for files that contain only Doxygen documentation for some module or such, for cases where there is no natural single header for putting the documentation.
- For C++ files, prefer naming the file the same as the (main) class it contains. Currently all file names are all-lowercase, even though class names contain capital letters. It is OK to use commonly known abbreviations, and/or omit the name of the containing directory if that would cause unnecessary repetition (e.g., as a common prefix to every file name in the directory) and the remaining part of the name is unique enough.
- Avoid having multiple files with the same name in different places within the same library. In addition to making things harder to find, C++ source files with the same name can cause obscure problems with some compilers. Currently, unit tests are an exception to the rule (there is only one particular compiler that had problems with this, and a workaround is possible if/when that starts to affect more than a few of the test files).

### Common guidelines for C and C++ code

- Preprocessor macros should be all upper-case. Do not use leading underscores, as all such names are reserved according to the C/C++ standard.
- Name include guards like `GMX_DIRNAME_HEADERNAME_H`.
- Avoid abbreviations that are not obvious to a general reader.
- If you use acronyms (e.g., PME, DD) in names, follow the Microsoft policy on casing: two letters is uppercase (DD), three or more is lowercase (Pme). If the first letter would be lowercase in the context where it is used (e.g., at the beginning of a function name, or anywhere in a C function name), it is clearest to use all-lowercase acronym.

### C code

- All function and variable names are lowercase, with underscores as word separators where needed for clarity.
- All functions that are part of the public API should start with `gmx_`. Preferably, other functions should as well. Some parts of the code use a `_gmx_` prefix for internal functions, but strictly speaking, these are reserved names, so, e.g., a trailing underscore would be better.
- Old C code and changes to it can still use the hungarian notation for booleans and enumerated variable names, as well as enum values, where they are prefixed with `b` and `e` respectively, or you can gradually move to the C++ practice below. Whatever you choose, avoid complex abbreviations.

### C++ code

- Use CamelCase for all names. Start types (such as classes, structs, typedefs and enum values) with a capital letter, other names (functions, variables) with a lowercase letter. You may use an all-lowercase name with underscores if your class closely resembles an external construct (e.g., a standard library construct) named that way.
- C++ interfaces are named with an `I` prefix, such as in ICommandLineModule. This keeps interfaces identifiable, without introducing too much clutter (as the interface is typically used quite widely, spelling out `Interface` would make many of the names unnecessarily long).

- Abstract base classes are typically named with an `Abstract` prefix.
- Member variables are named with a trailing underscore.
- Accessors for a variable `foo_` are named `foo()` and `setFoo()`.
- Global variables are named with a `g_` prefix.
- Global and file-static variables are named with a `g_` prefix.
- Static class and function variables are named with an `s_` prefix.
- Static `constexpr` file, class, or function members are named with a `sc_` prefix.
- Global constants are often named with a `c_` prefix.
- If the main responsibility of a file is to implement a particular class, then the name of the file should match that class, except for possible abbreviations to avoid repetition in file names (e.g., if all classes within a module start with the module name, omitting or abbreviating the module name is OK). Currently, all source file names are lowercase, but this casing difference should be the only difference.
- For new C++ code, avoid using the hungarian notation that is a descendant from the C code (i.e., the practice of using a `b` prefix for boolean variables and an `e` prefix for enumerated variables and/or values). Instead, make the names long with a good description of what they control, typically including a verb for boolean variables, like `foundAtom`.
- Prefer class enums over regular ones, so that unexpected conversions to int do not happen.
- Name functions to convert class enum values to strings as `enumValueToString`.
- When using a non-class enum, prefer to include the name of the enumeration type as a base in the name of enum values, e.g., `HelpOutputFormat_Console`, in particular for settings exposed to other modules.
- Prefer to use enumerated types and values instead of booleans as control parameters to functions. It is reasonably easy to understand what the argument `HelpOutputFormat_Console` is controlling, while it is almost impossible to decipher `TRUE` in the same place without checking the documentation for the role of the parameter.

The rationale for the trailing underscore and the global/static prefixes is that it is immediately clear whether a variable referenced in a method is local to the function or has wider scope, improving the readability of the code.

### Code for GPUs

Rationale: on GPUs, using the right memory space is often performance critical.

- In CUDA device code `sm_`, `gm_`, and `cm_` prefixes are used for shared, global and constant memory. The absence of a prefix indicates register space. Same prefixes are used in OpenCL code, where `sm_` indicates local memory and no prefixes are added to variables in private address space.
- Data transferred to and from host has to live in both CPU and GPU memory spaces. Therefore it is typical to have a pointer or container (in CUDA), or memory buffer (in OpenCL) in host memory that has a device-based counterpart. To easily distinguish these, the variables names for such objects are prefixed `h_` and `d_` and have identical names otherwise. Example: `h_masses`, and `d_masses`.

- In all other cases, pointers to host memory are not required to have the prefix `h_` (even in parts of the host code, where both host and device pointers are present). The device pointers should always have the prefix `d_` or `gm_`.
- In case GPU kernel arguments are combined into a structure, it is preferred that all device memory pointers within the structure have the prefix `d_` (i.e. `kernelArgs.d_data` is preferred to `d_kernelArgs.data`, whereas both `d_kernelArgs.d_data` and `kernelArgs.data` are not acceptable).
- Note that the same pointer can have the prefix `d_` in the host code, and `gm_` in the device code. For example, if `d_data` is passed to the kernel as an argument, it should be aliased to `gm_data` in the kernel arguments list. In case a device pointer is a field of a passed structure, it can be used directly or aliased to a pointer with `gm_` prefix (i.e. `kernelArgs.d_data` can be used as is or aliased to `gm_data` inside the kernel).
- Avoid using uninformative names for CUDA warp, thread, block indexes and their OpenCL analogs (i.e. `threadIndex` is preferred to `i` or `atomIndex`).

#### Unit tests

- Test fixtures (the first parameter to `TEST`/`TEST_F`) are named with a `Test` suffix.
- Classes meant as base classes for test fixtures (or as names to be typedefed to be fixtures) are named with a `TestBase` or `Fixture` suffix.
- The CTest test is named with CamelCase, ending with `Tests` (e.g., `OptionsUnitTests`).
- The test binary is named with the name of the module and a `-test` suffix.

### 8.7.4 Allowed language features

Most of these are not strict rules, but you should have a very good reason for deviating from them.

#### Portability considerations

Most GROMACS files compile as C++17, but some files remain that compile as C99. C++ has a lot of features, but to keep the source code maintainable and easy to read, we will avoid using some of them in GROMACS code. The basic principle is to keep things as simple as possible.

- MSVC supports only a subset of C99 and work-arounds are required in those cases.
- We should be able to use virtually all C++17 features; see "GPU API considerations" below for exceptions.

### C++ Standard Library

GROMACS code must support the lowest common denominator of C++17 standard library features available on supported platforms. Some modern features are useful enough to warrant back-porting. Consistent and forward-compatible headers are provided in `src/gromacs/compat/` as described in the Library documentation

### General considerations

As a baseline, GROMACS follows the C++ Core Guidelines c++ guidelines, unless our own more specific guidelines below say otherwise. We tend to be more restrictive in some areas, both because we depend on the code compiling with a lot of different C++ compilers, and because we want to increase readability. However, GROMACS is an advanced projects in constant development, and as our needs evolve we will both relax and tighten many of these points. Some of these changes happen naturally as part of agreements in code review, while major parts where we don't agree should be pushed to a issue tracker thread. Large changes should be suggested early in the development cycle for each release so we avoid being hit by last-minute compiler bugs just before a release.

- Use namespaces.
  - Everything in the GROMACS, `gmxapi`, `nblib` libraries/layers should be in the respective `gmx`, `gmxapi`, `nblib` namespaces. See also here
  - Code used in testing should be in a nested `test` namespace so that it is clearly distinct from both the code being tested and the `testing` namespace used by GoogleTest.
  - Use anonymous namespaces in source files to describe symbols that should not have external linkage (see here).
  - Use the `internal` namespace in header files to denote implementation details that cannot be depended upon, because anonymous namespaces cannot be used (see here).
  - Otherwise, avoid nested namespaces unless needing to expose a group of related free functions in a module header.
  - Don't use `using` in headers except possibly for aliasing some commonly-used names, and avoid file-level blanket `using namespace gmx` and similar. If only a small number of `gmx` namespace symbols needed in a not-yet-updated file, consider importing just those symbols. See also here.
  - Generally avoid using the `gmx::` prefix when working inside that namespace. Do use it when clarity or correctness are an issue, e.g. to differentiate between the same name in different namespaces.
  - With the exception of basic types such as `int64_t` or `size_t`, generally use the `std::` prefix for symbols in STL.
- Use STL, but do not use iostreams outside of the unit tests. iostreams can have a negative impact on performance compared to other forms of string streams, depending on the use case. Also, they don't always play well with using C `stdio` routines at the same time, which are used extensively in the current code. However, since Google tests rely on iostreams, you should use it in the unit test code.
- Don't use non-const references as function parameters. They make it impossible to tell whether a variable passed as a parameter may change as a result of a function call without looking up the prototype.

- Use `not_null<T>` pointers wherever possible to convey the semantics that a pointer to a valid is required, and a reference is inappropriate. See also here and here.
- Use `string_view` in cases where you want to only use a read-only-sequence of characters instead of using `const std::string &`. See also here. Because null termination expected by some C APIs (e.g. fopen, fputs, fprintf) is not guaranteed, string_view should not be used in such cases.
- Use `ArrayRef<T>` or `ArrayRef<const T>` (and their padded forms) to express a view over a contiguous range of (respectively) modifiable, or non-modifiable values, particularly as a function parameter. This type is cheap to copy and more expressive than a pair of iterator parameters or pointer and size parameters (albeit that the `restrict` qualifier cannot be used directly with `ArrayRef<T>`). It also provides flexibility for the caller to easily provide values from a C-array, ``array, `vector` (of any allocator type), or something else, without the implementor having to provide a family of overloads (which scales badly with multiple view parameters to the same function). Use e.g. `const ArrayRef<T>` in the definition of the function to express that the function implementation will not modify the view, even though it can modifiy the values in the view. See also here and here and other parts of the C++ Core Guidelines. GROMACS will adopt the similar C++20 `span` for this purpose in due course.
- Use `optional<T>` types in situations where there is exactly one, reason (that is clear to all parties) for having no value of type T, and where the lack of value is as natural as having any regular value of T, see here. Good examples include the return type of a function that parses an integer value from a string, searching for a matching element in a range, or providing an optional name for a residue type. Do use optional for lazy loading of resources, e.g., objects that have no default constructor and are hard to construct. Prefer other constructs when the logic requires an explanation of the reason why no regular value for T exists, e.g., do not use `optional<T>` for error handling. `optional<T>` "models an object, not a pointer, even though operator*() and operator->() are defined" (cppreference). No dynamic memory allocation ever takes place and forward declaration of objects stored in `optional<T>` does not work. Thus refrain from optional when passing handles; in contrast to unique_ptr, optional has value semantics, not reference semantics.
- Don't use C-style casts; use `const_cast`, `static_cast` or `reinterpret_cast` as appropriate. See the point on RTTI for `dynamic_cast`. For emphasizing type (e.g. intentional integer division) use constructor syntax. For creating real constants use the user-defined literal `_real` (e.g. `2.5_real` instead of `static_cast<real>(2.5)`).
- Use signed integers for arithmetic (including loop indices). Use `ssize` (available as free function and member of `ArrayRef`) to avoid casting.
- Avoid overloading functions unless all variants really do the same thing, just with different types. Instead, consider making the function names more descriptive.
- Avoid using default function arguments. They can lead to the code being less readable than without (see here). If you think that your specific case improves readability (see here), you can justify their use.
- Don't overload operators before thorough consideration whether it really is the best thing to do. Never overload `&&`, `||`, or the comma operator, because it's impossible to keep their original behavior with respect to evaluation order.
- Try to avoid complex templates, complex template specialization or techniques like SFINAE as much as possible. If nothing else, they can make the code more difficult to understand.

- Don't use multiple inheritance. Inheriting from multiple pure interfaces is OK, as long as at most one base class (which should be the first base class) has any code. Please also refer to the explanation here and here.
- Don't write excessively deep inheritance graphs. Try to not inherit implementation just to save a bit of coding; follow the principle "inherit to be reused, not to reuse." Also, you should not mix implementation and interface inheritance. For explanation please see here.
- Don't include unnecessary headers. In header files, prefer to forward declare the names of types used only "in name" in the header file. This reduces compilation coupling and thus time. If a source file also only uses the type by name (e.g. passing a pointer received from the caller to a callee), then no include statements are needed!
- Make liberal use of assertions to help document your intentions (but prefer to write the code such that no assertion is necessary).
- Prefer `GMX_ASSERT()` and `GMX_RELEASE_ASSERT()` to naked `assert()` because the former permit you to add descriptive text.
- Use gmx::Mutex rather than pthreads, std or raw thread-MPI mutexes.
- Use proper enums for variable whose type can only contain one of a limited set of values. C++ is much better than C in catching errors in such code. Ideally, all enums should be typed enums, please see here.
- When writing a new class, think whether it will be necessary to make copies of that class. If not, declare the copy constructor and the assignment operator as private and don't define them, making any attempt to copy objects of that class fail. If you allow copies, either provide the copy constructor and the assignment operator, or write a clear comment that the compiler-generated ones will do (and make sure that they do what you want). `src/gromacs/utility/classhelpers.h` has some convenience macros for doing this well. You can also use deleted functions in this case.
- Declare all constructors with one parameter as explicit unless you really know what you are doing. Otherwise, they can be used for implicit type conversions, which can make the code difficult to understand, or even hide bugs that would be otherwise reported by the compiler. For the same reason, don't declare operators for converting your classes to other types without thorough consideration. For an explanation, please see here.
- Write const-correct code (no `const_cast` unless absolutely necessary).
- Avoid using RTTI (run-time type information, in practice `dynamic_cast` and `typeid`) unless you really need it. The cost of RTTI is very high, both in binary size (which you always pay if you compile with it) and in execution time (which you pay only if you use it). If your problem seems to require RTTI, think about whether there would be an alternative design that wouldn't. Such alternative designs are often better.
- Don't depend on compiler metadata propagation. struct elements and captured lambda parameters tend to have `restrict` and alignment qualifiers discarded by compilers, so when you later define an instance of that structure or allocate memory to hold it, the data member might not be aligned at all.
- Plan for code that runs in compute-sensitive kernels to have useful data layout for re-use, alignment for SIMD memory operations
- Recognize that some parts of the code have different requirements - compute kernels, mdrun setup code, high-level MD-loop code, simulation setup tools, and analysis tools have different needs, and the trade-off point between correctness vs reviewer time vs developer time vs compile time vs run time will differ.

- Be restrictive when using `auto` to define variables. It is fine to use `auto` if the variable type is immediately apparent, or completely unnecessary, to a future reader of the code. In some case it may be necessary to use `auto`, e.g., together with generic templates. It is recommended to use `auto` with lengthy types, such as iterators or lambdas, where specifying the type explicitly would reduce readability. If in doubt, avoid using `auto`.
- When writing a class, put common infrastructure like constructors, destructors, move and copy operations first, then public member functions, then private member functions, and finally private data. Newly written classes should not have public data.
- Prefer storing handles (ie. pointers, references) to common "constant" infrastructure objects like `MpiComm` and `gmx_wallcycle`, even if they are small and fast to copy. We want modules to have consistent usage patterns, be easy to debug, and (where possible) to depend only on type names.

### Implementing exceptions for error handling

See *Error handling* (page 784) for the approach to handling run-time errors, ie. use exceptions.

- Write exception-safe code. All new code has to offer at least the basic or nothrow guarantee to make this feasible.
- Use std (or custom) containers wherever possible.
- Use smart pointers for memory management. By default, use `std::unique_ptr` and `gmx::unique_cptr` in association with any necessary raw `new` or `snew` calls. `std::shared_ptr` can be used wherever responsibility for lifetime must be shared. Never use `malloc`.
- Use RAII for managing resources (memory, mutexes, file handles, ...).
- It is preferable to avoid calling a function which might throw an exception from a legacy function which is not exception safe. However, we make the practical exception to permit the use of features such as `std::vector` and `std::string` that could throw `std::bad_alloc` when out of memory. In particular, GROMACS has a lot of old C-style memory handling that checking tools continue to issue valid warnings about as the tools acquire more functionality, and fixing these with old constructs is an inefficient use of developer time.
- Functions / methods should be commented whether they are exception safe, whether they might throw an exception (even indirectly), and if so, which exception(s) they might throw.

### GPU API considerations

- Write OpenCL as C (specifically, C99) code. Using C++ in OpenCL kernels is not well supported.
- Use SYCL 2020 standard. The vendor-specific extensions and backend-specific code can be used when needed for performance, but a reasonable fallback must be provided for all other supported targets.
- Use USM and in-order queues in SYCL code instead of `sycl::buffer`. This makes the code more uniform across all GPU backends. Besides, buffers are more challenging for the compilers to optimize in kernels, leading to worse performance (as of 2022).

#### Preprocessor considerations

- Don't use preprocessor defines for things other than directly related to configuring the build. Use templates or inline functions to generate code, and enums or const variables for constants.
- Preprocessing variables used for configuring the build should be organized so that a valid value is always defined, i.e. we never test whether one of our preprocessor variables is defined, rather we test what value it has. This is much more robust under maintenance, because a compiler can tell you that the variable is undefined.
- Avoid code with lengthy segments whose compilation depends on #if (or worse, #ifdef of symbols provided from outside GROMACS).
- Prefer to organize the definition of a const variable at the top of the source code file, and use that in the code. This helps keep all compilation paths built in all configurations, which reduces the incidence of silent bugs.
- Indent nested preprocessor conditions if nesting is necessary and the result looks clearer than without indenting.
- Please strongly consider a comment repeating the preprocessor condition at the end of the region, if a lengthy region is necessary and benefits from that. For long regions this greatly helps in understanding and debugging the code.

### 8.7.5 Guidelines for creating meaningful issue reports

This section gives some started on how to generate useful issues on the GROMACS issue tracker. The information here comes to a large extent directly from there, to help you in preparing your reports.

#### What to report

Please only report issues you have confirmed to be caused by GROMACS behaving in an unintended way, and that you have investigated to the best of your ability. If you have large simulations fail at some point, try to also trigger the problem with smaller test cases that are more easily debuggable.

Bugs resulting from the use third-party software should be investigated first to make sure that the fault is in GROMACS and not in other parts of the toolchain.

Please do not submit generic issues resulting from system instabilities and systems *Blowing up* (page 390).

#### What should be included

The report should include a general description of the problem with GROMACS indicating both the expected behaviour and the actual outcome. If the issue causes program crashes, the report should indicate where the crash happens and if possible include the stack trace right up to the crash.

All bugs should include the necessary information for the developers to reproduce the errors, including if needed minimal input files (*tpr, *top, *mdp, etc), run commands or minimal version of run scripts, how you compiled GROMACS and if possible the system architecture.

The emphasis should be on having a *minimal* working example that is easy to follow for the developers, that does not result in any warnings or errors in itself. If your example generates errors, your issue will

likely not be considered as *real*, or at the minimum it will be much harder to analyse to find the actual issue.

If your inputs are sensitive, then it is possible to create private issues so that the developer team can have access to solve the problem, while preventing widespread visibility on the internet.

### Supporting the developers

In general you should be able to answer questions posed to you by the developers working on the program, if you want to help them in fixing the bug you found. This may include things such as explaining run scripts or simulation set-up, as well as confirming issues with different versions of the program and different combinations of supported libraries and compilers.

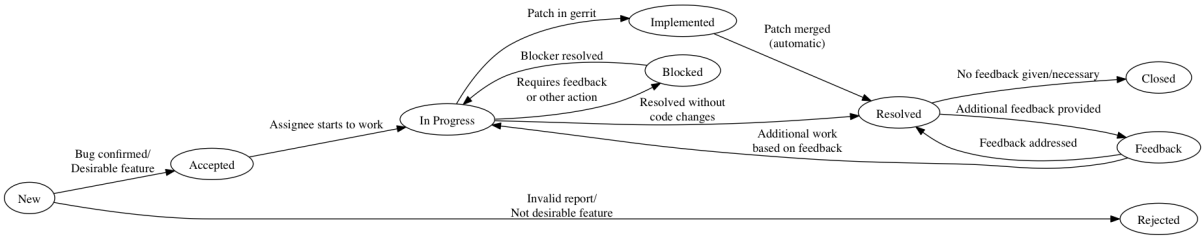
Please refrain from setting things such as target version or deciding on unreasonable priorities. If you decide to fix the issue on your own, please adhere to the other standards mentioned on the related pages *Guidelines for code formatting* (page 771) and *Guidelines for formatting of git commits* (page 783).

**See also**

*Contribute to GROMACS* (page 742)

### General issue workflow

The general issue workflow is shown in the figure below:




Project maintainers will apply Status labels as the issue is processed.

- Status::Accepted: Bug confirmed / Desirable feature.
- Status::In Progress: Assignee starts to work.
- Status::Blocked: Progress requires feedback or other action.
- Status::Rejected: Invalid report or not a desirable feature.
- Status::Fix uploaded: Merge request is available for review
- Status::Feedback-wanted: Resolution pending additional feedback or response
- Status::Resolved: The issue will be closed if there is no further discussion.

### 8.7.6 Guidelines for formatting of git commits

While there is no true correct way on how to submit new commits for code review for GROMACS, following these guidelines will help the review process go smoothly.

#### General rules for newly submitted code

New code should follow the other *style rules* (page 771) outlined above before submitting. This will make it less likely that your change will be rejected due to that. If your change modifies some existing code that does not yet conform to the style, then a preliminary patch that cleans up the surrounding area is a good idea. We like to slowly improve the quality while we add or change functionality.

#### Guidelines for git commit messages

Commit messages should contain a quick explanation in verb form on what has been changed or what has been the purpose of the change. If available, the final part of the message before the ChangeId should be a short section like **Fixes #issue-id** to link the change to a possibly previously posted issue, or **Refs #issue-id** if the present patch is somehow related to that work without necessarily fixing the whole issue.

#### Concerning inline code comments

New code should be sufficiently commented so that other people will be able to understand the purpose of the code, and less about the current operation. Preferably the variable naming and code structure clarify the mechanics, and comments should only refer to higher-level things, such as choice of algorithm, or the desire to be consistent with some other part of the code.

For example, the following comment would be insufficient to explain the (made up example) of iteration over a list of interactions:

```
/* Code takes each item and iterates over them in a loop
 * to store them.
 */
```

A much better example would be explaining why the iteration takes place:

```
/* We iterate over the items in the list to get
 * the specific interaction type for all of them
 * and store them in the new data type for future
 * use in function foo
 */
```

From the second example, someone debugging might be able to deduce better if an error observed in *foo* is actually caused by the previous assignment.

### 8.7.7 Error handling

To make GROMACS behave like a proper library, we need to handle errors in a consistent and predictable way. In this section, "user" refers to the end user of GROMACS whether via some command-line tool, or a workflow, or a call to a public API. There are different types of errors, and the handling reflects this. This section is a work in progress, particularly as the broader C++ community is a long way from consensus in these areas.

#### Brief summary on which method to use

More detailed rules and rationale are written below, but in short, when a reason exists that code is unable to do its job:

- If the reason can be checked at compile-time, then use `static_assert`.
- If the reason is normal in context, then express that in the types used (e.g. return `std::optional`) and document that this is normal.
- If the reason is that an internal invariant or pre-condition is violated (e.g. unexpected null pointer passed) on a hot code path, then use `GMX_ASSERT`.
- Otherwise, if the reason is that an internal invariant or pre-condition is violated then use `GMX_RELEASE_ASSERT`.
- Otherwise, (typically an error returned from system call or GPU SDK, bad user input), then use `GMX_THROW`.

#### Guiding principles

- GROMACS should adopt approaches that have achieved consensus elsewhere, e.g. in the C++ Core Guidelines. In particular, be guided by its section on error handling
- The library should not print out anything to stdio/stderr unless it is part of the API specification, and even then, there should be a way for the user to suppress or redirect the output.
- The library should normally not terminate the program without the user having control over this.
- Design interfaces of functions, classes, modules, and libraries so that values passed at run time are valid. Pass const references or `not_null` pointers rather than raw pointers. Return objects where possible. Use e.g. class enums for the type of passed values. Consider such enums as template parameters, rather than passing run-time values. Refactor existing interfaces to improve such aspects when starting new work in an area.
- Check user input at API boundaries and establish invariants as soon as possible, e.g. by expressing the user's choice in the type system. These form the pre-conditions that error handling will rely on.
- Use assertions to validate invariants and pre-conditions. There is value in using a different technique for checking such violations in order to make the reason for the check clear to the maintainer.

### Specific rules

- Use `static_assert` wherever possible to detect errors at compile time.
- Throw *exceptions* to indicate that a function cannot do its assigned task, per the C++ Core Guidelines E.2. In particular, constructors should throw when they cannot construct a valid object, per C++ Core Guidelines C.42. However, recognize that in some cases the underlying reason is that some other component has not set up the correct pre-condition, and such cases should be handled with assertions (see below).
- At API boundaries, the assigned task of some code will be to validate the input, and that code should express failure to validate by throwing.
- Many programming errors violate pre-conditions of other functions. Until there is language support for contracts, the best that can be done is to check these with *assertions*. Note that only one component should have the responsibility for validating any particular input from the user, and other components should rely upon that validation in their pre-conditions.
- When asserting, use `GMX_RELEASE_ASSERT` by default. This macro will run its check in all build configurations, including `Release`.
- When asserting in cases where the code is called in an inner loop of e.g. the MD step, `GMX_ASSERT` can be used. This macro will run its check only when `NDEBUG` is not defined, including the `RelWithAssert` build configuration (which is the default build type used in CI).
- It can be appropriate to provide both checked and unchecked interfaces, as `std::vector` does with `at()` and `operator[]`, respectively. Note that even the latter is checked if you build e.g. `libstdc++` in the right configuration!
- When calling low-level APIs (including C and C++ standard libraries, GPU SDKs) always check for success/failure. Generally the correct thing to do upon failure will be to throw, perhaps including a descriptive string obtained from an error code with another API call.
- Do catch exceptions from lower-level components memory or file system IO errors. As a general guideline, incorrect user input should not produce an untrapped exception resulting in execution termination telling the user an exception occured. Instead, you should catch exceptions in an earlier stack frame, make a suitable decision about diagnostic messages, and then decide whether execution should be terminated (if that is in the scope of the code making the decision) and, if so, how to terminate.
- There is a global list of possible exceptions in `api/legacy/include/gromacs/utility/exceptions.h`, and the library should throw one of these when it fails, possibly providing a more detailed description of the reason for the failure. The types of exceptions can be extended, and currently include:
  - Out of memory (e.g. `std::bad_alloc`)
  - File I/O error (e.g. not found)
  - Invalid user input (could not be understood)
  - Inconsistent user input (parsed correctly, but has internal conflicts)
  - Simulation instability
  - Invalid API call/value/internal error (an assertion might also be used in such cases)
  - In the internals of a module called from code that is not exception safe, you can use exceptions for error handling, but avoid propagating them to caller code.

- Avoid using exceptions to propagate errors across regions that start or join threads with OpenMP, since OpenMP cannot make guarantees about whether exceptions are caught or if the program will crash. Currently we catch all exceptions before we leave an OpenMP threaded region. If you throw an exception, make sure that it is caught and handled appropriately in the same thread/OpenMP section.
- Avoid using exceptions to propagate errors within regions where non-blocking API calls (e.g. to MPI or GPU SDKs) have been made, because the possible advantage of catching at a higher level and continuing execution is absent when the partner in the API call may be left blocked.
- There are also cases where a library routine wants to report a warning or a non-fatal error, but is still able to continue processing. In this case you should try to collect all issues and report and report them (similar to what grompp does with notes, warnings and errors) instead of just returning the first error. It is irritating to users if they fix the reported error, but then they keep getting a new error message every time the rerun the program.
- A function should not fail as part of its normal operation. However, doing nothing can be considered normal operation. A function accessing data should typically also be callable when no such data is available, but still return through normal means. If the failure is not normal, it is OK to rather throw an exception.
- Error handling with `gmx_fatal`, `gmx_warning`, `gmx_incons`, `gmx_comm` etc. is deprecated and should generally be refactored to throw or assert according to the above guidelines.
- There is currently no attempt made to check for error states on other MPI ranks during the simulation and provide a coordinated recovery. However setup code should do such checks routinely.
- We use `GMX_RELEASE_ASSERT` and `GMX_ASSERT` rather than `assert` to ensure that non-immediate strings can be used to describe the problem when the error is reported. This is particularly useful when troubleshooting issues where missing test coverage leads users to uncover such errors.

For coding guidelines to make this all work, see *Implementing exceptions for error handling* (page 780).

## 8.8 Development-time tools

### 8.8.1 Using Doxygen

This section documents how Doxygen is set up in the GROMACS source tree, as well as guidelines for adding new Doxygen comments. Examples are included, as well as tips and tricks for avoiding Doxygen warnings. The guidelines focus on C++ code and other new code that follows the new module layout. Parts of the guidelines are still applicable to documenting older code (e.g., within `gmxlib/` or `mdlib/` ), in particular the guidelines about formatting the Doxygen comments and the use of `\internal`. See *Documentation organization* (page 748) for the overall structure of the documentation.

To get started quickly, you only need to read the first two sections to understand the overall structure of the documentation, and take a look at the examples at the end. The remaining sections provide the details for understanding why the examples are the way they are, and for more complex situations. They are meant more as a reference to look up solutions for particular problems, rather than single-time reading. To understand or find individual Doxygen commands, you should first look at the Doxygen documentation (http://www.doxygen.nl/manual/).

### Documentation flavors

The GROMACS source tree is set up to produce several different levels of Doxygen documentation:

1. Public API documentation (suffix `-user`), which documents functions and classes exported from the library and intended for use outside the GROMACS library.
2. Library API documentation (suffix `-lib`), which additionally includes functions and classes that are designed to be used from other parts of GROMACS, as well as some guidelines that are mostly of interest to developers.
3. Full documentation (suffix `-full`), which includes (nearly) all (documented) functions and classes in the source tree.
4. Maximally verbose documentation (suffix `-dev`) with everything doxygen can extract as well as additional internal links.

Each subsequent level of documentation includes all the documentation from the levels above it. The suffixes above refer to the suffixes of Doxygen input and output files, as well as the name of the output directory. When all the flavors have been built, the front pages of the documentation contain links to the other flavors, and explain the differences in more detail.

As a general guideline, the public API documentation should be kept free of anything that a user linking against an unmodified GROMACS does not see. In other words, the public API documentation should mainly document the contents of installed headers, and provide the necessary overview of using those. Also, verbosity requirements for the public API documentation are higher: ideally, readers of the documentation could immediately start using the API based on the documentation, without any need to look at the implementation.

Similarly, the library API documentation should not contain things that other modules in GROMACS can or should never call. In particular, anything declared locally in source files should be only available in the full documentation. Also, if something is documented, and is not identified to be in the library API, then it should not be necessary to call that function from outside its module.

### Building the documentation

If you want to see up-to-date documentation, you can download artifacts from the `webpage` job of the latest scheduled pipeline for a corresponding branch (https://gitlab.com/gromacs/gromacs/-/pipelines?page=1&scope=all&source=schedule). CI also runs Doxygen for all changes pushed to GitLab for release and main branches, and the resulting documentation can be found in the artifacts of the corresponding `webpage` job. The Doxygen job will fail if it introduces any Doxygen warnings.

You may need to build the documentation locally if you want to check the results after adding/modifying a significant amount of comments. This is recommended in particular if you do not have much experience with Doxygen. It is a good idea to build with all the different settings to see that the result is what you want, and that you do not produce any warnings. For local work, it is generally a good idea to set `GMX_COMPACT_DOXYGEN=ON` CMake option, which removes some large generated graphs from the documentation and speeds up the process significantly. There are also "fast" versions of the `make` targets that skip the additional diagrams built for the `lib` level and lower.

All files related to Doxygen reside in the `docs/doxygen/` subdirectory in the source and build trees. In a freshly checked out source tree, this directory contains various `Doxyfile-*.cmakein` files. When you run CMake, corresponding files `Doxyfile-user`, `Doxyfile-lib`, `Doxyfile-full`, `Doxyfile-dev` are generated at the corresponding location in the build tree. There is also a `Doxyfile-common.cmakein`, which is used to produce `Doxyfile-common`. This file contains

settings that are shared between all the input files. `Doxyfile-compact` provides the extra settings for `GMX_COMPACT_DOXYGEN=ON`.

You can run Doxygen directly with one of the generated files (all output will be produced under the current working directory), or build one of the `doxygen-user`, `doxygen-lib`, `doxygen-full`, `doxygen-dev` targets. The targets run Doxygen in a quieter mode and only show the warnings if there were any, and put the output under `docs/html/doxygen/` in the build tree, so that the Doxygen build cooperates with the broader `webpage` target. The `doxygen-all` target builds all three targets with less typing.

The generated documentation is put under `html-user/`, `html-lib/`, `html-full/`, and/or `html-dev/`. Open `index.xhtml` file from one of these subdirectories to start browsing (for GROMACS developers, the `html-lib/` is a reasonable starting point). Log files with all Doxygen warnings are also produced as `docs/doxygen/doxygen-*.log`, so you can inspect them after the run.

You will need Doxygen 1.8.5 to build the current documentation. Other versions may work, but likely also produce warnings. Additionally, graphviz and mscgen are required for some graphs in the documentation, and `latex` for formulas. Working versions are likely available through most package managers. It is possible to build the documentation without these tools, but you will see some errors and the related figures will be missing from the documentation.

### General guidelines for Doxygen markup

Doxygen provides quite a few different alternative styles for documenting the source code. There are subtleties in how Doxygen treats the different types of comments, and this also depends somewhat on the Doxygen configuration. It is possible to change the meaning of a comment by just changing the style of comment it is enclosed in. To avoid such issues, and to avoid needing to manage all the alternatives, a single style throughout the source tree is preferable. When it comes to treatment of styles, GROMACS uses the default Doxygen configuration with one exception: `JAVADOC_AUTOBRIEF` is set `ON` to allow more convenient one-line brief descriptions in C code.

Majority of existing comments in GROMACS uses Qt-style comments (`/*!` and `//!` instead of `/**` and `///`, `\brief` instead of `@brief` etc.), so these should be used also for new documentation. There is a single exception for brief comments in C code; see below.

Similarly, existing comments use `/*!` for multiline comments in both C and C++ code, instead of using multiple `//!` lines for C++. The rationale is that since the code will be a mixture of both languages for a long time, it is more uniform to use similar style in both. Also, since files will likely transition from C to C++ gradually, rewriting the comments because of different style issues should not generally be necessary. Finally, multi-line `//!` comments can work differently depending on Doxygen configuration, so it is better to avoid that ambiguity.

When adding comments, ensure that a short brief description is always produced. This is used in various listings, and should briefly explain the purpose of the method without unnecessarily expanding those lists. The basic guideline is to start all comment blocks with `\brief` (possibly after some other Doxygen commands). If you want to avoid the `\brief` for one-liners, you can use `//!`, but the description must fit on a single line; otherwise, it is not interpreted as a brief comment. Note in particular that a simple `/*!` without a `\brief` does not produce a brief description. Also note that `\brief` marks the whole following paragraph as a brief description, so you should insert an empty line after the intended brief description.

In C code, `//` comments must be avoided because some compilers do not like them. If you want to avoid the `\brief` for one-liners in C code, use `/**` instead of `//!`. If you do this, the brief description

should not contain unescaped periods except at the end. Because of this, you should prefer `//!` in C++ code.

Put the documentation comments in the header file that contains the declaration, if such a header exists. Implementation-specific comments that do not influence how a method is used can go into the source file, just before the method definition, with an `\internal` tag in the beginning of the comment block. Doxygen-style comments within functions are not generally usable.

At times, you may need to exclude some part of a header or a source file such that Doxygen does not see it at all. In general, you should try to avoid this, but it may be necessary to remove some functions that you do not want to appear in the public API documentation, and which would generate warnings if left undocumented, or to avoid Doxygen warnings from code it does not understand. Prefer `\cond` and `\endcond` to do this. If `\cond` does not work for you, you can also use `#ifndef DOXYGEN`. If you exclude a class method in a header, you also need to exclude it in the source code to avoid warnings.

### GROMACS specifics

The general guidelines on the style of Doxygen comments were given above. This section introduces GROMACS specific constructs currently used in Doxygen documentation, as well as how GROMACS uses Doxygen groups to organize the documentation.

Some consistency checks are done automatically using custom scripts. See *Source tree checker scripts* (page 815) for details.

### Controlling documentation visibility

To control in which level of documentation a certain function appears, three different mechanisms are used:

- Global Doxygen configuration. This is mainly used to include declarations local to source files only in the full documentation. You can find the details from the `Doxyfile-*.cmakein` files, and some of them are also mentioned below on individual code constructs.
- The standard Doxygen command `\internal` marks the documentation to be only extracted into the full documentation (`INTERNAL_DOCS` is `ON` only for the full documentation). This should be used as a first command in a comment block to exclude all the documentation. It is possible to use `\internal` and `\endinternal` to exclude individual paragraphs, but `\if internal` is preferred (see below). In addition, GROMACS-specific custom Doxygen command `\libinternal` is provided, which should be used the same way to exclude the documentation from the public API documentation. This command expands to either `\internal` or to a no-op, depending on the documentation level.
- Doxygen commands `\if` and `\cond` can be used with section names `libapi` and `internal` to only include the documentation in library API and the full documentation, respectively. `libapi` is also defined in the full documentation. These are declared using `ENABLED_SECTIONS` in the Doxygen configuration files.

Examples of locations where it is necessary to use these explicit commands are given below in the sections on individual code constructs.

### Modules as Doxygen groups

As described in *Source code organization* (page 746), each subdirectory under `src/gromacs/` represents a *module*, i.e., a somewhat coherent collection of routines. Doxygen cannot automatically generate a list of routines in a module; it only extracts various alphabetical indexes that contain more or less all documented functions and classes. To help reading the documentation, the routines for a module should be visible in one place.

GROMACS uses Doxygen groups to achieve this: for each documented module, there is a `\defgroup` definition for the module, and all the relevant classes and functions need to be manually added to this group using `\ingroup` and `\addtogroup`. The group page also provides a natural place for overview documentation about the module, and can be navigated to directly from the "Modules" tab in the generated documentation.

Some notes about using `\addtogroup` are in order:

- `\addtogroup` only adds the elements that it directly contains into the group. If it contains a namespace declaration, only the namespace is added to the group, but none of the namespace contents are. For this reason, `\addtogroup` should go within the innermost scope, around the members that should actually be added.
- If the module should not appear in the public API documentation, its definition (`\defgroup`) should be prefixed with a `\libinternal`. In this case, also all `\addtogroup` commands for this module should be similarly prefixed. Otherwise, they create the group in the public API documentation, but without any of the content from the `\defgroup` definition. This may also cause the contents of the `\addtogroup` section to appear in the public API documentation, even if it otherwise would not.

### Public API and library API groups

In addition to the module groups, two fixed groups are provided: `group_publicapi` and `group_libraryapi`. Classes and files can be added to these groups using GROMACS specific custom `\inpublicapi` and `\inlibraryapi` commands. The generated group documentation pages are not very useful, but annotated classes and files show the API definition under the name, making this information more easily accessible. These commands in file-level comments are also used for some automatic intermodule dependency validation (see below).

Note that functions, enumerations, and other entities that do not have a separate page in the generated documentation can only belong to one group; in such a case, the module group is preferred over the API group.

### Documenting specific code constructs

This section describes the techical details and some tips and tricks for documenting specific code constructs such that useful documentation is produced. If you are wondering where to document a certain piece of information, see the documentation structure section in *Documentation organization* (page 748). The focus of the documentation should be on the overview content: Doxygen pages and the module documentation. An experienced developer can relatively easily read and understand individual functions, but the documentation should help in getting the big picture.

### Doxygen pages

The pages that are accessible through navigation from the front page are written using Markdown and are located under `docs/doxygen/`. Each page should be placed in the page hierarchy by making it a subpage of another page, i.e., it should be referenced once using `\subpage`. `mainpage.md` is the root of the hierarchy.

There are two subdirectories, `user/` and `lib/`, determining the highest documentation level where the page appears. If you add pages to `lib/`, ensure that there are no references to the page from public API documentation. `\if libapi` can be used to add references in content that is otherwise public. Generally, the pages should be on a high enough level and provide overview content that is useful enough such that it is not necessary to exclude them from the library API documentation.

### Modules

For each module, decide on a header file that is the most important one for that module (if there is no self-evident header, it may be better to designate, e.g., `module-doc.h` for this purpose, but this is currently not done for any module). This header should contain the `\defgroup` definition for the module. The name of the group should be `module_`*`name`*, where *name* is the name of the subdirectory that hosts the module.

The module should be added to an appropriate group (see `docs/doxygen/misc.cpp` for definitions) using `\ingroup` to organize the "Modules" tab in the generated documentation.

One or more contact persons who know about the contents of the module should be listed using `\author` commands. This provides a point of contact if one has questions. Authors should be listed in chronological order of contributions, where possible.

### Classes/structs

Classes and structs in header files appear always in Doxygen documentation, even if their enclosing file is not documented. So start the documentation blocks of classes that are not part of the public API with `\internal` or `\libinternal`. Classes declared locally in source files or in unnamed namespaces only appear in the full documentation.

If a whole class is not documented, this does not currently generate any warning. The class is simply exluded from the documentation. But if a member of a documented class is not documented, a warning is generated. Guidelines for documenting free functions apply to methods of a class as well.

For base classes, the API classification (`\inpublicapi` or `\inlibraryapi`) should be based on where the class is meant to be subclassed. The visibility (`\internal` or `\libinternal`), in contrast, should reflect the API classification of derived classes such that the base class documentation is always generated together with the derived classes.

For classes that are meant to be subclassed and have protected members, the protected members should only appear at the documentation level where the class is meant to be subclassed. For example, if a class is meant to be subclassed only within a module, the protected members should only appear in the full documentation. This can be accomplished using `\cond` (note that you will need to add the `\cond` command also to the source files to hide the same methods from Doxygen, otherwise you will get confusing warnings).

### Methods/functions/enums/macros

These items do not appear in the documentation unless their enclosing scope is documented. For class members, the scope is the class; otherwise, it is the namespace if one exists, or the file. An `\addtogroup` can also define a scope if the group has higher visibility than the scope outside it. So if a function is not within a namespace (mostly applicable to C code) and has the same visibility as its enclosing file, it is not necessary to add a `\internal` or `\libinternal`.

Static functions are currently extracted for all documentation flavors to allow headers to declare `static inline` functions (used in, for example, math code). Functions in anonymous namespaces are only extracted into the full documentation. Together with the above rules, this means that you should avoid putting a `static` function within a documented namespace, even within source files, or it may inadvertently appear in the public API documentation.

If you want to exclude an item from the documentation, you need to put in inside a `\cond` block such that Doxygen does not see it. Otherwise, a warning for an undocumented function is generated. You need to enclose both the declaration and the definition with `\cond`.

### Files

Each documented file should start with a documentation block (right after the copyright notice) that documents the file. See the examples section for exact formatting. Things to note:

- Please do not specify the file name explicitly after `\file`. By default, a file comment applies to the file it is contained in, and an explicit file name only adds one more thing that can get out of date.
- `\brief` cannot appear on the same line as the `\file`, but should be on the next line.
- `\internal` or `\libinternal` should indicate where the header is visible. As a general guideline, all installed headers should appear in the public API documentation, i.e., not contain these commands. If nothing else, then to document that it does not contain any public API functions. Headers that declare anything in the library API should be marked with `\libinternal`, and the rest with `\internal`.
- All source files, as well as most test files, should be documented with `\internal`, since they do not provide anything to public or library API, and this avoids unintentionally extracting things from the file into those documentations. Shared test files used in tests from other modules should be marked with `\libinternal`.
- `\inpublicapi` or `\inlibraryapi` should be used to indicate where the header is meant to be directly included.
- As with modules, one or more contact persons should be listed with `\author`. If you make significant modifications or additions to a file, consider adding an `\author` line for yourself.

### Directories

Directory documentation does not typically contain useful information beyond a possible brief description, since they correspond very closely to modules, and the modules themselves are documented. A brief description is still useful to provide a high-level overview of the source tree on the generated "Files" page. A reference to the module is typically sufficient as a brief description for a directory. All directories are currently documented in `docs/doxygen/directories.cpp`.

### Examples

### Basic C++

Here is an example of documenting a C++ class and its containing header file. Comments in the code and the actual documentation explain the used Doxygen constructs.

```
/*! \libinternal \file
 * \brief
 * Declares gmx::MyClass.
 *
 * More details.  The documentation is still extracted for the␣
↪class even if
 * this whole comment block is missing.
 *
 * \author Example Author <example@author.com>
 * \inlibraryapi
 * \ingroup module_mymodule
 */

namespace gmx
{

/*! \libinternal
 * \brief
 * Brief description for the class.
 *
 * More details.  The \libinternal tag is required for classes,␣
↪since they are
 * extracted into the documentation even in the absence of␣
↪documentation for
 * the enclosing scope.
 * The \libinternal tag is on a separate line because of a bug in␣
↪Doxygen
 * 1.8.5 (only affects \internal, but for clarity it is also worked␣
↪around
 * here).
 *
 * \inlibraryapi
 * \ingroup module_mymodule
 */
class MyClass
```

(continues on next page)

(continued from previous page)

```
{
    public:
        // Trivial constructors or destructors do not require␣
↪documentation.
        // But if a constructor takes parameters, it should be␣
↪documented like
        // methods below.
        MyClass();
        ~MyClass();

        /*! \brief
         * Brief description for the method.
         *
         * \param[in] param1  Description of the first parameter.
         * \param[in] param2  Description of the second parameter.
         * \returns   Description of the return value.
         * \throws    std::bad_alloc if out of memory.
         *
         * More details describing the method.  It is not an error␣
↪to put this
         * above the parameter block, but most existing code has it␣
↪here.
         */
        int myMethod(int param1, const char *param2) const;

        //! Brief description for the accessor.
        int simpleAccessor() const { return var_; }
        /*! \brief
         * Alternative, more verbose way of specifying a brief␣
↪description.
         */
        int anotherAccessor() const;
        /*! \brief
         * Brief description for another accessor that is so long␣
↪that it does
         * not conveniently fit on a single line cannot be␣
↪specified with //!.
         */
        int secondAccessor() const;

    private:
        // Private members (whether methods or variables) are␣
↪currently ignored
        // by Doxygen, so they don't need to be documented. ␣
↪Documentation
        // doesn't hurt, though.
        int var_;
};

} // namespace gmx
```

### Basic C

Here is another example of documenting a C header file (so avoiding all C++-style comments), and including free functions. It also demonstrates the use of `\addtogroup` to add multiple functions into a module group without repeated `\ingroup` tags.

```
/*! \file
 * \brief
 * Declares a collection of functions for performing a certain task.
 *
 * More details can go here.
 *
 * \author Example Author <example@author.com>
 * \inpublicapi
 * \ingroup module_mymodule
 */

/*! \addtogroup module_mymodule */
/*! \{ */

/*! \brief
 * Brief description for the data structure.
 *
 * More details.
 *
 * \inpublicapi
 */
typedef struct {
    /** Brief description for member. */
    int  member;
    int  second; /**< Brief description for the second member. */
    /*! \brief
     * Brief description for the third member.
     *
     * Details.
     */
    int  third;
} gmx_mystruct_t;

/*! \brief
 * Performs a simple operation.
 *
 * \param[in] value  Input value.
 * \returns   Computed value.
 *
 * Detailed description.
 * \inpublicapi cannot be used here, because Doxygen only allows a␣
↪single
 * group for functions, and module_mymodule is the preferred group.
 */
```

(continues on next page)

(continued from previous page)

```
int gmx_function(int value);

/* Any . in the brief description, except at the end, should be␣
 ↪escaped as \. */
/** Brief description for this function. */
int gmx_simple_function();

/*! \} */
```

### Scoping and visibility rules

The rules where Doxygen expects something to be documented, and when are commands like `\internal` needed, can be complex. The examples below describe some of the pitfalls.

```
/*! \libinternal \file
 * \brief
 * ...
 *
 * The examples below assume that the file is documented like this:
 * with an \libinternal definition at the beginning, with an intent␣
 ↪to not
 * expose anything from the file in the public API.  Things work␣
 ↪similarly for
 * the full documentation if you replace \libinternal with \internal
 * everywhere in the example.
 *
 * \ingroup module_example
 */


/*! \brief
 * Brief description for a free function.
 *
 * A free function is not extracted into the documentation unless␣
 ↪the enclosing
 * scope (in this case, the file) is.  So a \libinternal is not␣
 ↪necessary.
 */
void gmx_function();

// Assume that the module_example group is defined in the public␣
 ↪API.

//! \addtogroup module_example
//! \{

//! \cond libapi
/*! \brief
```

(continues on next page)

(continued from previous page)

```
 * Brief description for a free function within \addtogroup.
 *
 * In this case, the enclosing scope is actually the module_example␣
↪group,
 * which is documented, so the function needs to be explicitly␣
↪excluded.
 * \\libinternal does not work, since it would produce warnings␣
↪about an
 * undocumented function, so the whole declaration is hidden from␣
↪Doxygen.
 */
void gmx_function();
//! \endcond

//! \}

// For modules that are only declared in the library API, \
↪addtogroup
// cannot be used without an enclosing \cond.  Otherwise, it will␣
↪create
// a dummy module with the identifier as the name...

//! \cond libapi
//! \addtogroup module_libmodule
//! \{

/*! \brief
 * Brief description.
 *
 * No \libinternal is necessary here because of the enclosing \cond.
 */
void gmx_function();

//! \}
//! \endcond

// An alternative to the above is use this, if the enclosing scope␣
↪is only
// documented in the library API:

//! \libinternal \addtogroup module_libmodule
//! \{

//! Brief description.
void gmx_function()

//! \}

/*! \libinternal \brief
```

(continues on next page)

(continued from previous page)

```
 * Brief description for a struct.
 *
 * Documented structs and classes from headers are always extracted␣
↪into the
 * documentation, so \libinternal is necessary to exclude it.
 * Currently, undocumented structs/classes do not produce warnings,␣
↪so \cond
 * is not necessary.
 */
struct t_example
{
    int  member1; //!< Each non-private member should be documented.
    bool member2; //!< Otherwise, Doxygen will produce warnings.
};

// This namespace is documented in the public API.
namespace gmx
{

//! \cond libapi
/*! \brief
 * Brief description for a free function within a documented␣
↪namespace.
 *
 * In this case, the enclosing scope is the documented namespace,
 * so a \cond is necessary to avoid warnings.
 */
void gmx_function();
//! \endcond

/*! \brief
 * Class meant for subclassing only within the module, but the␣
↪subclasses will
 * be public.
 *
 * This base class still provides public methods that are visible␣
↪through the
 * subclasses, so it should appear in the public documentation.
 * But it is not marked with \inpublicapi.
 */
class BaseClass
{
    public:
        /*! \brief
         * A public method.
         *
         * This method also appears in the documentation of each␣
↪subclass in
         * the public and library API docs.
```

(continues on next page)

(continued from previous page)

```
         */
        void method();

    protected:
        // The \cond is necessary to exlude this documentation from␣
↪the public
        // API, since the public API does not support subclassing.
        //! \cond internal
        //! A method that only subclasses inside the module see.
        void methodForSubclassToCall();

        //! A method that needs to be implemented by subclasses.
        virtual void virtualMethodToImplement() = 0;
        //! \endcond
};

} // namespace gmx
```

### Module documentation

Documenting a new module should place a comment like this in a central header for the module, such that the "Modules" tab in the generated documentation can be used to navigate to the module.

```
/*! \defgroup module_example "Example module (example)"
 * \ingroup group_utilitymodules
 * \brief
 * Brief description for the module.
 *
 * Detailed description of the module.  Can link to a separate␣
↪Doxygen page for
 * overview, and/or describe the most important headers and/or␣
↪classes in the
 * module as part of this documentation.
 *
 * For modules not exposed publicly, \libinternal can be added at␣
↪the
 * beginning (before \defgroup).
 *
 * \author Author Name <author.name@email.com>
 */

// In other code, use \addtogroup module_example and \ingroup␣
↪module_example to
// add content (classes, functions, etc.) onto the module page.
```

### Common mistakes

The most common mistake, in particular in C code, is to forget to document the file. This causes Doxygen to ignore most comments in the file, so it does not validate the contents of the comments either, nor is it possible to actually check how the generated documentation looks like.

The following examples show some other common mistakes (and some less common) that do not produce correct documentation, as well as Doxygen "features"/bugs that can be confusing.

- The struct itself is not documented; other comments within the declaration are ignored.

```
struct t_struct {

    // The comment tries to document both members at once, but␣
↪it only
    // applies to the first.  The second produces warnings␣
↪about missing
    // documentation (if the enclosing struct was documented).

    //! Angle parameters.
    double alpha, beta;
};
```

- This does not produce any brief documentation. An explicit `\brief` is required, or `//!` (C++) or `/** */` (C) should be used.

```
/*! Brief comment. */
int gmx_function();
```

- This does not produce any documentation at all, since a `!` is missing at the beginning.

```
/* \brief
 * Brief description.
 *
 * More details.
 */
int gmx_function();
```

- This puts the whole paragraph into the brief description. A short description is preferable, separated by an empty line from the rest of the text.

```
/*! \brief
 * Brief description. The description continues with all kinds␣
↪of details about
 * what the function does and how it should be called.
 */
int gmx_function();
```

- This may be a Doxygen bug, but this does not produce any brief description.

```
/** \internal Brief description. */
int gmx_function();
```

- If the first declaration below appears in a header, and the second in a source file, then Doxygen does not associate them correctly and complains about missing documentation for the latter. The solution is to explicitly add a namespace prefix also in the source file, even though the compiler does not require it.

```
// Header file
//! Example function with a namespace-qualified parameter type.
int gmx_function(const gmx::SomeClass &param);

// Source file
using gmx::SomeClass;

int gmx_function(const SomeClass &param);
```

- This puts the namespace into the mentioned module, instead of the contents of the namespace. `\addtogroup` should go within the innermost scope.

```
//! \addtogroup module_example
//! \{

namespace gmx
{

//! Function intended to be part of module_example.
int gmx_function();

}
```

### Existing code

More examples you can find by looking at existing code in the source tree. In particular new C++ code such as that in the `src/gromacs/analysisdata/` and `src/gromacs/options/` subdirectories contains a large amount of code documented mostly along these guidelines. Some comments in `src/gromacs/selection/` (in particular, any C-like code) predate the introduction of these guidelines, so those are not the best examples.

### 8.8.2 Automation and Infrastructure

Starting from 2020 release, automated testing and documentation builds are performed by GitLab and GitLab Runner.

#### GitLab CI Pipeline Execution

The repository contains DockerFiles and GitLab Runner configuration files to support automated testing and documentation builds. General information on configuring GitLab CI pipelines can be found in the official Gitlab documentation.

The GitLab CI configuration entry point is the `.gitlab-ci.yml` file at the root of the source tree. Configuration templates are found in the files in the `admin/ci-templates/` directory.

Docker images used by GitLab Runner are available in our GitLab Container Registry. (See *Containers* (page 807).) Images are (re)built manually using details in `admin/containers`. (See *Tools* (page 807).)

> **Note**
>
> Full automated testing is only available for merge requests originating from branches of the main https://gitlab.com/gromacs/gromacs repository. GitLab CI pipelines created for forked repositories will include fewer jobs in the testing pipeline. Non-trivial merge requests may need to be issued from a branch in the `gromacs` project namespace in order to receive sufficient testing before acceptance.

#### Configuration files

At the root of the repository, `.gitlab-ci.yml` defines the stages and some default parameters, then includes files from `admin/gitlab-ci/` to define jobs to be executed in the pipelines.

Note that job names beginning with a period (`.`) are "hidden". Such jobs are not directly eligible to run, but may be used as templates via the *extends* job property.

#### Job parameters

Refer to https://docs.gitlab.com/ee/ci/yaml for complete documentation on GitLab CI job parameters, but note the following GROMACS-specific conventions.

**before_script**
: Used by several of our templates to prepend shell commands to a job *script* parameter. Avoid using *before-script* directly, and be cautious about nested *extends* overriding multiple *before_-script* definitions.

**job cache**
: There is no global default, but jobs that build software will likely set *cache*. To explicitly unset *cache* directives, specify a job parameter of `cache: {}`. Refer to GitLab docs for details. In particular, note the details of cache identity according to cache:key

**image**
: See *Containers* (page 807) for more about the Docker images used for the CI pipelines. If a job

depends on artifacts from previous jobs, be sure to use the same (or a compatible) image as the dependency!

**rules**
**only**
**except**
**when**

*Job* parameters for controlling the circumstances under which jobs run. (Some key words may have different meanings when occurring as elements of other parameters, such as *archive:when*, to which this note is not intended to apply.) Rules in GitLab are special, since the first matching rule will cause a job to trigger, and then all remaining rules are ignored. To create rules to skip jobs, write rules that use the execution time "never". Errors or unexpected behavior will occur if you specify more than one *.rules:...* template, or if you use these parameters in combination with a *.rules...* template - it is thus NOT possible to combine rules through inheritance with the `extends` tag. Instead, to combine sequences of rules we recommend using a plain rules tag where you reference rule entries with the !reference tag, e.g. `!reference [.rules:<something>, rules]`. Each such reference can be used as an individual rule in the list. To reduce errors and unexpected behavior, restrict usage of these controls to regular job definitions (do not use in "hidden" or parent jobs). Note that *rules* is not compatible with the older *only* and *except* parameters. We have standardized on the (newer) *rules* mechanism.

**tags**

We no longer use any special tags for general (meaning CPU-only) GROMACS CI jobs, to make sure at least the CPU jobs can still run even if somebody clones the repo. For testing you can still add a default tag at the start of the top-level `.gitlab-ci.yml`, but this should only be used to check that a specific runner works - for production we handle it in GitLab instead by selecting what runners accept untagged jobs. By default we currently run those on the infrastructure in Stockholm for the GROMACS project, but please design all CPU jobs so they will work on the shared runners too.

**variables**

Many job definitions will add or override keys in *variables*. Refer to GitLab for details of the merging behavior. Refer to *Updating regression tests* (page 804) for local usage.

### Schedules and triggers

Pipeline schedules are configured through the GitLab web interface. Scheduled pipelines may provide different variable definitions through the environment to jobs that run under the `schedules` condition.

Nightly scheduled pipelines run against `main` and *release* branches in the GROMACS repository.

Some of the rules defined in `rules.gitlab-ci.yml` restrict jobs to run *only* for scheduled pipelines, or only for *specific* schedules according to the variables defined for that schedule in the web interface. For example, the rule element `if-weekly-then-on-success` causes a job to run only if the schedule sets `GMX_PIPELINE_SCHEDULE=weekly`.

**Running post-merge-acceptance pipelines**

The Gitlab CI for GROMACS runs a set of jobs by default only after a MR has been accepted and the resulting commit is included in the target branch if it is `main` or one of the *release* branches. Those jobs can be triggered manually using the `POST_MERGE_ACCEPTANCE` input variable documented below when executing a new pipeline through the Gitlab web interface.

See also *trigger-post-merge.py* (page 808).

### Global templates

In addition to the templates in the main job definition files, common "mix-in" functionality and behavioral templates are defined in `admin/gitlab-ci/global.gitlab-ci.yml`. For readability, some parameters may be separated into their own files, named according to the parameter (e.g. `rules.gitlab-ci.yml`).

Jobs beginning with `.use-` provide mix-in behavior, such as boilerplate for jobs using a particular tool chain.

Jobs beginning with a parameter name allow parameters to be set in a single place for common job characteristics. If providing more than a default parameter value, the job name should be suffixed by a meaningful descriptor and documented within `admin/gitlab-ci/global.gitlab-ci.yml`

### Job names

Job names should

1. Indicate the purpose of the job.
2. Indicate relationships between multi-stage tasks.
3. Distinguish jobs in the same stage.
4. Distinguish job definitions throughout the configuration.

Jobs may be reassigned to different stages over time, so including the stage name in the job name is not helpful, generally. If tags like "pre" and "post," or "build" and "test" are necessary to distinguish phases of, say, "webpage," then such tags can be buried at the end of the job name.

Stylistically, it is helpful to use delimiters like `:` to distinguish the basic job name from qualifiers or details. Also consider grouping jobs

### Updating regression tests

Changes in GROMACS that require changes in regression-tests are notoriously hard, because a merge request that tests against the non-updated version of the regression tests will necessarily fail, while updating regression tests while the current change is not integrated into main, might cause other merge request pipelines to fail.

The solution is a new regression-test branch or commit, uploaded to gitlab. Then set that regression test branch with REGRESSIONTESTBRANCH or the specific commit with REGRESSIONTESTCOMMIT when running the specific pipeline that requires the regressiontest-update. See below on how to set variables for specific pipelines.

### Variables

The GitLab CI framework, GitLab Runner, plugins, and our own scripts set and use several variables.

Default values are available from the top level `variables` definition in `global.gitlab-ci.yml`. Many of the mix-in / template jobs provide additional or overriding definitions. Other variables may be set when making final job definitions.

Variables may control the behavior of GitLab-CI (those beginning with `CI_`), GitLab Runner and supporting infrastructure, or may be used by job definitions, or passed along to the environment of executed commands.

*variables* keys beginning with `KUBERNETES_` relate to the GitLab Runner Kubernets executor

Other important variable keys are as follows.

**BUILD_DIR**
: GROMACS specific directory to perform configuration, building and testing in. Usually job dependent, needs to be the same for all tasks of dependent jobs.

**CI_PROJECT_NAMESPACE**
: Distinguishes pipelines created for repositories in the `gromacs` GitLab project space. May be used to pre-screen jobs to determine whether GROMACS GitLab infrastructure is available to the pipeline before the job is created.

**COMPILER_MAJOR_VERSION**
: Integer version number provided by toolchain mix-in for convenience and internal use.

**CMAKE**
: `gromacs/ci-...` Docker images built after October 2020 have several versions of CMake installed. The most recent version of CMake in the container will be appear first in `PATH`. To allow individual jobs to use specific versions of CMake, please write the job *script* sections using `$CMAKE` instead of `cmake` and begin the *script* section with a line such as `-CMAKE=${CMAKE:-$(which cmake)}`. Specify a CMake version by setting the *CMAKE* variable to the full executable path for the CMake version you would like to use. See also *Containers* (page 807).

**CMAKE_COMPILER_SCRIPT**
: CMake command line options for a tool chain. A definition is provided by the mix-in toolchain definitions (e.g. `.use-gcc8`) to be appended to **`cmake`** calls in a job's *script*.

**CMAKE_MPI_OPTIONS**
: Provide CMake command line arguments to define GROMACS MPI build options.

**DRY_RUN**
: Read-only environment variable used to control behaviour of script uploading artifact files to the ftp and web servers. Set to false to actually upload files. This is usually done through the pipeline submission script, but can be done manual as well through the web interface.

**GROMACS_MAJOR_VERSION**
: Read-only environment variable for CI scripts to check the library API version to expect from the `build` job artifacts. Initially, this variable is only defined in `admin/gitlab-ci/api-client.matrix/gromacs-main.gitlab-ci.yml` but could be moved to `admin/gitlab-ci/global.gitlab-ci.yml` if found to be of general utility.

**GROMACS_RELEASE**
: Read-only environment variable that can be checked to see if a job is executing in a pipeline for

preparing a tagged release. Can be set when launching pipelines via the GitLab web interface. For example, see *rules* mix-ins in `admin/gitlab-ci/global.gitlab-ci.yml`.

**REGRESSIONTESTBRANCH**
Use this branch of the regressiontests rather than main to allow for merge requests that require updated regression tests with valid CI tests.

**REGRESSIONTESTCOMMIT**
Use this commit to the regressiontests rather than the head on main to allow for merge requests that require updated regression tests with valid CI tests.

**POST_MERGE_ACCEPTANCE**
Read-only environment variable that indicates that only jobs scheduled to run after a commit has been merged into its target branch should be executed. Can be set to run pipelines through the web interface or as schedules. For use please see the *rules* mix-ins in `admin/gitlab-ci/global.gitlab-ci.yml`.

**GMX_PIPELINE_SCHEDULE**
Read-only environment variable used exclusively by job rules. Rule elements of the form `if-<value>-then-on-success` check whether `GMX_PIPELINE_SCHEDULE==value`. Allowed values are determined by the rule elements available in `admin/gitlab-ci/rules.gitlab-ci.yml`, and include `nightly` and `weekly` to restrict jobs to only run in the corresponding schedules.

### Setting variables

Variables for individual piplelines are set in the gitlab interface under `CI/CD`; `Pipelines`. Then chose in the top right corner `Run Piplelines`. Under `Run for`, the desired branch may be selected, and variables may be set in the fields below.

### Using GPUs in Gitlab-runner

Previously, GROMACS used a hacked local version of Gitlab-runner where we had added support for Kubernetes extended resources. However, Gitlab has unfortunately not shown interest in merging these, and as the runner has evolved it is difficult to keep up. In the future it might be possible to select GPUs directly in the job configuration, but for now we use the ability to specify it in each Gitlab-runner configuration and thus have separate runners going for CPU-only as well as single or dual GPU devices from Nvidia, AMD, and Intel.

To enable both us and other users to also use the shared Gitlab runners, the top-level configuration `.gitlab-ci.yml` now contains a few variables where you can select what tags to use for Gitlab-runners to get single or dual devices from each vendor. There are also variables that allow you to set the largest number of devices you have (on single nodes) in these runners; if any tests cannot be run because you do not have the right hardware, we will simply skip those tests.

### Containers

GROMACS project infrastructure uses Docker containerization to isolate automated tasks. A number of images are maintained to provide a breadth of testing coverage.

Scripts and configuration files for building images are stored in the repository under `admin/containers/`. Images are (re)built manually by GROMACS project staff and pushed to DockerHub and GitLab. See https://hub.docker.com/u/gromacs and https://gitlab.com/gromacs/gromacs/container_registry

### GitLab Container Registry

CI Pipelines use a GitLab container registry instead of pulling from Docker Hub.

Project members with role `Developer` or higher privilege can push images to the container registry.

Steps:

1. Create a personal access token (docs) with `write_registry` and `read_registry` scopes. Save the hash!
2. Authenticate from the command line with `docker login registry.gitlab.com -u <user name> -p <hash>`
3. `docker push registry.gitlab.com/gromacs/gromacs/<imagename>`

Refer to `buildall.sh` in the `main` branch for the set of images currently built.

Within *pipeline jobs* (page 802), jobs specify a Docker image with the *image* property. For image naming convention, see *utility.image_name()* (page 814). Images from the GitLab registry are easily accessible with the same identifier as above. For portability, CI environment variables may be preferable for parts of the image identifier. Example:

```
some_job:
  image: ${CI_REGISTRY_IMAGE}/ci-<configuration>
  ...
```

For more granularity, consider equivalent expressions `${CI_REGISTRY}/${CI_PROJECT_PATH}` or `${CI_REGISTRY}/${CI_PROJECT_NAMESPACE}/${CI_PROJECT_NAME}` Ref: https://docs.gitlab.com/ee/ci/variables/predefined_variables.html

### Tools

(in `admin/`)

### make-release-build.py

Automatic release options.

```
usage: make-release-build.py [-h] (--local | --server) [--token␣
↪TOKEN]
                             [--ssh-key SSH_KEY] [--release | --no-
↪release]
                             [--dry-run | --no-dry-run] [--branch␣
↪BRANCH]
```

**-h, --help**
show this help message and exit

**--local**
Set when running in local (submit pipeline) mode.

**--server**
Set when running in server (upload artefacts) mode.

**--token** <token>
GitLab access token needed to launch pipelines

**--ssh-key** <ssh_key>
Path to SSH key needed to upload things to server. Pass in local mode to have it during the job

**--release**

**--no-release**

**--dry-run**

**--no-dry-run**

**--branch** <branch>
Branch to run pipeline for (default "main")

### trigger-post-merge.py

Options for manually submitting pipelines.

```
usage: trigger-post-merge.py [-h] [--type TYPE] --token TOKEN
                             [--branch BRANCH]
                             [--regtest-branch REGTEST_BRANCH]
                             [--regtest-commit REGTEST_COMMIT]
```

**-h, --help**
show this help message and exit

**--type** <type>
What kind of pipeline to run (default is "POST_MERGE_ACCEPTANCE")

**--token** <token>
: GitLab access token needed to launch pipelines

**--branch** <branch>
: Branch to run pipeline for (default "main")

**--regtest-branch** <regtest_branch>
: Regressiontest branch to use to for running regression tests (default none, which means fall back to main)

**--regtest-commit** <regtest_commit>
: Commit to use instead of the regtest-branch tip for running tests (default empty)

### admin/containers/buildall.sh

Uses NVidia's HPC Container Maker to generate DockerFiles using our *scripted_gmx_docker_builds* (page 811) module. Refer to the contents of `admin/buildall.sh` for the flags currently in use. Run the script to see the tagged images currently being produced.

### scripted_gmx_docker_builds.py

(in `admin/containers/`)

GROMACS CI image creation script

```
usage: scripted_gmx_docker_builds.py [-h] [--cmake [CMAKE ...]]
                                     [--gcc GCC | --llvm [LLVM] | --
↪oneapi
                                     [ONEAPI] | --intel-llvm [INTEL_
↪LLVM]]
                                     [--ubuntu [UBUNTU] | --centos␣
↪[CENTOS]]
                                     [--cuda [CUDA]] [--mpi [MPI]]
                                     [--tsan [TSAN]]
                                     [--adaptivecpp [ADAPTIVECPP]]
                                     [--rocm [ROCM]] [--intel-
↪compute-runtime]
                                     [--oneapi-plugin-nvidia]
                                     [--oneapi-plugin-amd] [--clfft␣
↪[CLFFT]]
                                     [--heffte [HEFFTE]]
                                     [--nvhpcsdk [NVHPCSDK]] [--
↪rocfftmp]
                                     [--rocshmem] [--doxygen␣
↪[DOXYGEN]]
                                     [--cp2k [CP2K]] [--hdf5] [--
↪plumed]
                                     [--cross [{aarch64,riscv64}]]
                                     [--libtorch [LIBTORCH]]
```

(continues on next page)

(continued from previous page)

```
                                        [--venvs [VENVS ...]]
                                        [--format {docker,singularity}]
```

### Named Arguments

| | |
|---|---|
| **--cmake** | Selection of CMake version to provide to base image. (default: ['3.28.0', '3.30.3', '4.0.5']) |
| **--gcc** | Select GNU compiler tool chain. (default: 11) Some checking is implemented to avoid incompatible combinations |
| **--llvm** | Select LLVM compiler tool chain. Some checking is implemented to avoid incompatible combinations |
| **--oneapi** | Select Intel oneAPI package version. |
| **--intel-llvm** | Select Intel LLVM release (GitHub tag). |
| **--ubuntu** | Select Ubuntu Linux base image. (default: "24.04") |
| **--centos** | Select Centos Linux base image. |
| **--cuda** | Select a CUDA version for a base Linux image from NVIDIA. |
| **--mpi** | Enable MPI (default disabled) and optionally select distribution (default: openmpi) |
| **--tsan** | Build special compiler versions with TSAN OpenMP support |
| **--adaptivecpp** | Select AdaptiveCpp repository tag/commit/branch. |
| **--rocm** | Select AMD compute engine version. |
| **--intel-compute-runtime** | Include Intel Compute Runtime. |
| **--oneapi-plugin-nvidia** | Install Codeplay oneAPI NVIDIA plugin. |
| **--oneapi-plugin-amd** | Install Codeplay oneAPI AMD plugin. |
| **--clfft** | Add external clFFT libraries to the build image |
| **--heffte** | Select heffte repository tag/commit/branch. |
| **--nvhpcsdk** | Select NVIDIA HPC SDK version. |
| **--rocfftmp** | Install rocfftMp for current ROCm version |
| **--rocshmem** | Install rocshmem for current ROCm version |
| **--doxygen** | Add doxygen environment for documentation builds. Also adds other requirements needed for final docs images. |
| **--cp2k** | Add build environment for CP2K QM/MM support |
| **--hdf5** | Install an HDF5 library. |
| **--plumed** | Install PLUMED library. |
| **--cross** | Possible choices: aarch64, riscv64 |

| | |
|---|---|
| | Add cross-compilation support for the specified architecture with QEMU emulation |
| **--libtorch** | Add build environment for LibTorch support |
| **--venvs** | List of Python versions ("major.minor.patch") for which to install venvs. (default: ['3.9.13', '3.12.5']) |
| **--format** | Possible choices: docker, singularity<br>Container specification format (default: "docker") |

### Supporting modules in `admin/containers`

#### `scripted_gmx_docker_builds.py`

Building block based Dockerfile generation for CI testing images.

Generates a set of docker images used for running GROMACS CI on Gitlab. The images are prepared according to a selection of build configuration targets that hope to cover a broad enough scope of different possible systems, allowing us to check compiler types and versions, as well as libraries used for accelerators and parallel communication systems. Each combinations is described as an entry in the build_configs dictionary, with the script analysing the logic and adding build stages as needed.

Based on the example script provided by the NVidia HPCCM repository.

**Reference:**
NVidia HPC Container Maker

**Authors:**

- Paul Bauer <paul.bauer.q@gmail.com>
- Eric Irrgang <ericirrgang@gmail.com>
- Joe Jordan <e.jjordan12@gmail.com>
- Mark Abraham <mark.j.abraham@gmail.com>
- Gaurav Garg <gaugarg@nvidia.com>

Usage:

```
$ python3 scripted_gmx_docker_builds.py --help
$ python3 scripted_gmx_docker_builds.py --format docker >␣
↪Dockerfile && docker build .
$ python3 scripted_gmx_docker_builds.py | docker build -
```

**See also**

`buildall.sh`

scripted_gmx_docker_builds.**add_base_stage**(*name: str*, *input_args*, *output_stages: MutableMapping[str, hpccm.Stage]*)

Establish dependencies that are shared by multiple parallel stages.

scripted_gmx_docker_builds.**add_cross_compilation_support**(*args*, *output_stages: MutableMapping[str, hpccm.Stage]*)

Configure cross-compilation toolchain and QEMU support.

scripted_gmx_docker_builds.**add_documentation_dependencies**(*input_args*, *output_stages: MutableMapping[str, hpccm.Stage]*)

Add appropriate layers according to doxygen input arguments.

scripted_gmx_docker_builds.**add_intel_llvm_compiler_build_stage**(*input_args*, *output_stages: Mapping[str, hpccm.Stage]*)

Isolate the Intel LLVM (open-source oneAPI) preparation stage.

This stage is isolated so that its installed components are minimized in the final image (chiefly /opt/intel) and its environment setup script can be sourced. This also helps with rebuild time and final image size.

scripted_gmx_docker_builds.**add_oneapi_compiler_build_stage**(*input_args*, *output_stages: Mapping[str, hpccm.Stage]*)

Isolate the oneAPI preparation stage.

This stage is isolated so that its installed components are minimized in the final image (chiefly /opt/intel) and its environment setup script can be sourced. This also helps with rebuild time and final image size.

scripted_gmx_docker_builds.**add_python_stages**(*input_args: Namespace*, *, *base: str*, *output_stages: MutableMapping[str, hpccm.Stage]*)

Add the stage(s) necessary for the requested venvs.

One intermediate build stage is created for each venv (see –venv option).

Each stage partially populates Python installations and venvs in the home directory. The home directory is collected by the 'pyenv' stage for use by the main build stage.

scripted_gmx_docker_builds.**add_tsan_compiler_build_stage**(*input_args*, *output_stages: Mapping[str, hpccm.Stage]*)

Isolate the expensive TSAN preparation stage.

This is a very expensive stage, but has few and disjoint dependencies, and its output is easily compartmentalized (/usr/local) so we can isolate this build stage to maximize build cache hits and reduce rebuild time, bookkeeping, and final image size.

scripted_gmx_docker_builds.**base_image_tag**(*args*) → str

Generate *image* for hpccm.baseimage().

scripted_gmx_docker_builds.**build_stages**(*args*) → Iterable[hpccm.Stage]

Define and sequence the stages for the recipe corresponding to *args*.

scripted_gmx_docker_builds.**get_cmake_stages**(*, *input_args: Namespace*, *base: str*)

Get the stage(s) necessary for the requested CMake versions.

One (intermediate) build stage is created for each CMake version, based on the *base* stage. See `--cmake` option.

**Each stage uses the version number to determine an installation location:**
/usr/local/cmake-{version}

The resulting path is easily copied into the main stage.

**Returns**
dict of isolated CMake installation stages with keys from `cmake-{version}`

scripted_gmx_docker_builds.**get_cross_compilation_packages**(*args*)

Get packages needed for cross-compilation.

scripted_gmx_docker_builds.**hpccm_distro_name**(*args*) → str

Generate *_distro* for hpccm.baseimage().

Convert the linux distribution variables into something that hpccm understands.

The same format is used by the lower level hpccm.config.set_linux_distro().

scripted_gmx_docker_builds.**prepare_venv**(*version: Version*) → Sequence[str]

Get shell commands to set up the venv for the requested Python version.

### utility.py

A utility module to help manage the matrix of configurations for CI testing and build containers.

When called as a stand alone script, prints a Docker image name based on the command line arguments. The Docker image name is of the form used in the GROMACS CI pipeline jobs.

Example:

```
$ python3 -m utility --llvm --doxygen
gromacs/ci-ubuntu-24.04-llvm-14-docs
```

**See also**

`buildall.sh`

As a module, provides importable argument parser and docker image name generator.

Note that the parser is created with `add_help=False` to make it friendly as a parent parser, but this means that you must derive a new parser from it if you want to see the full generated command line help.

Example:

```
import utility.parser
# utility.parser does not support `-h` or `--help`
parser = argparse.ArgumentParser(
    description='GROMACS CI image creation script',
    parents=[utility.parser])
# ArgumentParser(add_help=True) is default, so parser supports `-h`␣
↪and `--help`
```

**See also**

`scripted_gmx_docker_builds.py`

**Authors:**

- Paul Bauer <paul.bauer.q@gmail.com>
- Eric Irrgang <ericirrgang@gmail.com>
- Joe Jordan <e.jjordan12@gmail.com>
- Mark Abraham <mark.j.abraham@gmail.com>
- Gaurav Garg <gaugarg@nvidia.com>

`utility.`**`image_name`**(*configuration: Namespace*) → str

Generate docker image name.

Image names have the form `ci-<slug>`, where the configuration slug has the form:

```
<distro>-<version>-<compiler>-<major version>[-<gpusdk>-
↪<version>][-<use case>]
```

This function also applies an appropriate Docker image repository prefix.

**Parameters**

**`configuration`** – Docker image configuration as described by the parsed arguments.

`utility.`**`parser = ArgumentParser(prog='sphinx-build', usage=None, description='GROMACS CI image slug options.', formatter_class=<class 'argparse.HelpFormatter'>, conflict_handler='error', add_help=False)`**

A parent parser for tools referencing image parameters.

This argparse parser is defined for convenience and may be used to partially initialize parsers for tools.

**Warning**

Do not modify this parser.

Instead, inherit from it with the *parents* argument to `argparse.ArgumentParser`

### 8.8.3 Source tree checker scripts

> **Warning**
>
> This section is out of date. Several of the checks described are no longer performed or are deprecated per Issue 3288 and related issues.

There is a set of Python scripts, currently under `docs/doxygen/`, that check various aspects of the source tree for consistency. The script is based on producing an abstract representation of the source tree from various sources:

- List of files in the source tree (for overall layout of the source tree)
- List of installed headers (extracted from the generated build system)
- git attributes (to limit the scope of some checks)
- Doxygen XML documentation:
  - For tags about public/private nature of documented headers and other constructs
  - For actual documented constructs, to check them for consistency
- Hard-coded knowledge about the GROMACS source tree layout

This representation is then used for various purposes:

- Checking Doxygen documentation elements for common mistakes: missing brief descriptions, mismatches in file and class visibility, etc.
- Checking for consistent usage and documentation of headers: e.g., a header that is documented as internal to a module should not be used outside that module.
- Checking for module-level cyclic dependencies
- Generating dependency graphs between modules and for files within modules

The checks are run as part of a single `check-source` target, but are described in separate sections below. In addition to printing the issues to `stderr`, the script also writes them into `docs/doxygen/check-source.log` for later inspection. CI runs the checks as part of all pipelines and CI will fail if any issues are found.

For correct functionality, the scripts depend on correct usage of Doxygen annotations described in *Using Doxygen* (page 786), in particular the visibility and API definitions in file-level comments.

For some false positives from the script, the suppression mechanism described below is the easiest way to silence the script, but otherwise the goal would be to minimize the number of suppressions.

The scripts require Python 2.7 (other versions may work, but have not been tested).

To understand how the scripts work internally, see comments in the Python source files under `docs/doxygen/`.

### Checker details

The `check-source` target currently checks for a few different types of issues. These are listed in detail below, mainly related to documentation and include dependencies. Note in particular that the include dependency checks are much stricter for code in modules/directories that are documented with a `\defgroup`: all undocumented code is assumed to be internal to such modules. The rationale is that such code has gotten some more attention, and some effort should also have been put into defining what is the external interface of the module and documenting it.

- For all Doxygen documentation (currently does not apply for members that do not appear in the documentation):
  - If a member has documentation, it should have a brief description.
  - A note is issued for in-body documentation for functions, since this is ignored by our current settings.
  - If a class has documentation, it should have public documentation only if it appears in an installed header.
  - If a class and its containing file has documentation, the class documentation should not be visible if the file documentation is not.
- For all files:
  - Consistent usage of

    ```
    #include "..." // This should be used for internal (gromacs)␣
    ↪headers
    ```

    and

    ```
    #include <...> // This should be used for system and␣
    ↪external headers
    ```

  - When we again have installed headers, they must not include non-installed headers. Headers should be marked for install within `CMakeLists.txt` files of their respective modules.
  - All source files must include "gmxpre.h" as the first header.
  - A source/header file should include "config.h," "gromacs/simd/simd.h", or "gromacs/ewald/pme_simd.h" if and only if it uses a macro declared in such files.
- For documented files:
  - Installed headers should have public documentation, and other files should not.
  - The API level specified for a file should not be higher than where its documentation is visible. For example, only publicly documented headers should be specified as part of the public API.
  - If an `\ingroup module_foo` exists, it should match the subdirectory that the file is actually part of in the file system.
  - If a `\defgroup module_foo` exists for the subdirectory where the file is, the file should contain `\ingroup module_foo`.
  - Files should not include other files whose documentation visibility is lower (if the included file is not documented, the check is skipped).

- For files that are part of documented modules (`\defgroup module_foo` exists for the subdirectory), or are explicitly documented to be internal or in the library API:
  - Such files should not be included from outside their module if they are undocumented (for documented modules) or are not specified as part of library or public API.
- For all modules:
  - There should not be cyclic include dependencies between modules.

As a side effect, the XML extraction makes Doxygen parse all comments in the code, even if they do not appear in the documentation. This can reveal latent issues in the comments, like invalid Doxygen syntax. The messages from the XML parsing are stored in `docs/doxygen/doxygen-xml.log` in the build tree, similar to other Doxygen runs.

### Suppressing issues

The script is not currently perfect (either because of unfinished implementation, or because Doxygen bugs or incompleteness of the Doxygen XML output), and the current code also contains issues that the script detects, but the authors have not fixed. To allow the script to still be used, `doxygen/suppressions.txt` contains a list of issues that are filtered out from the report. The syntax is simple:

```
<file>: <text>
```

where `<file>` is a path to the file that reports the message, and `<text>` is the text reported. Both support `*` as a wildcard. If `<file>` is empty, the suppression matches only messages that do not have an associated file. `<file>` is matched against the trailing portion of the file name to make it work even though the script reports absolute paths. Empty lines and lines starting with # are ignored.

To add a suppression for an issue, the line that reports the issue can be copied into `suppressions.txt`, and the line number (if any) removed. If the issue does not have a file name (or a pseudo-file) associated, a leading `:` must be added. To cover many similar issues, parts of the line can then be replaced with wildcards.

### Include dependency graphs

The same set of Python scripts can also produce include dependency graphs with some additional annotations compared to what, e.g., Doxygen produces for a directory dependency graph. Currently, a module-level graph is automatically built when the Doxygen documentation is built and embedded in the documentation (not in the public API documentation). The graph, together with a legend, is on a separate page: Module dependency graph

The Python script produces the graphs in a format suitable for `dot` (from the `graphviz` package) to lay them out. The build system also provides a `dep-graphs` target that generates PNG files from the intermediate `dot` files. In addition to the module-level graph, a file-level graph is produced for each module, showing the include dependencies within that module. The file-level graphs can only be viewed as the PNG files, with some explanation of the notation below. Currently, these are mostly for eye candy, but they can also be used for analyzing problematic dependencies to clean up the architecture.

Both the intermediate `.dot` files and the final PNG files are put under `docs/doxygen/depgraphs/` in the build tree.

#### File graphs

The graphs are written to *module_name*`-deps.dot.png`.

Node colors:

**light blue**
: public API (installed) headers

**dark blue**
: library API headers

**gray**
: source files

**light green**
: test files

**white**
: other files

Each edge signifies an include dependency; there is no additional information currently included.

### 8.8.4 Automatic source code formatting

Python sources can be automatically formatted with Black from Python 3.9.3.

C++ source code can be automatically formatted using clang-format since GROMACS 2020. It automatically applies the guidelines in *Guidelines for code formatting* (page 771) and in *Guidelines for #include directives* (page 772). Additionally, other Python scripts are used for a few other automatic formatting/checking tasks. This page provides more details for clang-format, clang-tidy and copyright scripts.

Our CI uses these same scripts (in particular, `clang-format.sh`, `copyright.sh`, `clang-tidy.sh` and the `check-source` target) to enforce that the code stays invariant under such formatting.

#### Setting up clang-format

GROMACS formatting is enforced with clang-format 18.1.8. **`clang-format`** is one of the core *clang* tools. It may be included in a *clang* or *llvm* package from your favorite packaging system or you may find a standalone *clang-format* package, but you should confirm that the provided command is correct (other 18.1.x versions might be okay too). Example:

```
$ clang-format --version
clang-format version 18.1.8
```

If you use a different version of clang-format, you will likely get different formatting results than the GROMACS continuous integration testing system, and the commits that you push will fail the automated tests.

**Note**

Refer to LLVM for source and binary downloads. If downloading sources, note that you will need to download both the *LLVM source code* and the *Clang source code*. As per the clang INSTALL.txt, place the expanded clang source into a `tools/clang` subdirectory within the expanded llvm archive, then run CMake against the llvm source directory.

In order to use the installed version of clang-format for `clang-format.sh` and for the pre-commit hook, you also need to run this in each of your GROMACS repositories:

```
git config hooks.clangformatpath /path/to/clang-format
```

Alternatively, if you just want to use `clang-format.sh`, you can set the `CLANG_FORMAT` environment variable to `/path/to/clang-format`.

Using the pre-commit hook or git filters needs additional setup; see the respective sections below.

clang-format discovers which formatting rules to apply from the `.clang-format` configuration file(s) in project directories, which will be automatically updated (if necessary) when you **`git pull`** from the GROMACS repository. For more about the tool and the `.clang-format` configuration file, visit https://releases.llvm.org/18.1.8/tools/clang/docs/ClangFormat.html

### What is automatically formatted?

To identify which files are subject to automatic formatting, the scripts use git filters, specified in `.gitattributes` files. Only files that have the attribute `filter` set to one of the below values are processed:

- **`filter=complete_formatting`: Performs all formatting. Uses clang-format for code formatting.**
  Files included here are also passed to the clang-tidy code checker.
- `filter=clangformat`: clang-format is run. Again also runs clang-tidy.
- `filter=includesort`: include order is enforced and copyright headers are checked.
- `filter=copyright`: only copyright headers are checked.

Other files are ignored by `clang-tidy.sh`, `clang-format.sh`, `copyright.sh` and `reformat_all.sh` scripts (see below).

### Setting up clang-tidy

GROMACS source code tidiness checking is enforced with clang-tidy provided alongside *clang* compiler version 18. **`clang-tidy`** is one of the core *clang* tools. It may be included in a *clang* or *llvm* package from your favorite packaging system or you may find a standalone *clang-tidy* or *clang-tools* package, but you should confirm that the provided command is version 18. Example:

```
$ clang-tidy --version
  LLVM (http://llvm.org/):
    LLVM version 18.1.8
```

If you use a different version of clang-tidy, you will likely get different checking results than the GROMACS continuous integration testing system, and the commits that you push will fail the automated tests.

> **Note**
>
> Refer to LLVM for source and binary downloads. If downloading sources, note that you will need to download both the *LLVM source code* and the *Clang source code*. As per the clang INSTALL.txt, place the expanded clang source into a `tools/clang` subdirectory within the expanded llvm archive, then run CMake against the llvm source directory.

In order to use the installed version of clang-tidy for `clang-tidy.sh` and for the pre-commit hook, you also need to run this in each of your GROMACS repositories:

```
git config hooks.runclangtidypath /path/to/run-clang-tidy.py
```

Alternatively, if you just want to use `clang-tidy.sh`, you can set the `RUN_CLANG_TIDY` environment variable to `/path/to/run-clang-tidy.py`.

As above, see the sections below for using the pre-commit hook or git filters.

clang-tidy discovers which formatting rules to apply from the `.clang-tidy` configuration file(s) in project directories, which will be automatically updated (if necessary) when you **`git pull`** from the GROMACS repository. For more about the tool and the `.clang-tidy` configuration file, visit https://releases.llvm.org/18.1.8/tools/clang/tools/extra/docs/clang-tidy/index.html.

### Tools

#### `copyright.py`

This script provides low-level functionality to check and update copyright headers in C/C++ source files, as well as in several other types of files like CMake and Python scripts.

This file is also used as a loadable Python module for kernel generators, and provides the functionality to generate conformant copyright headers for such scripts.

You should rarely need to run this directly, but instead the bash scripts below use it internally. You can run the script with `--help` option if you want to see what all options it provides if you need to do some maintenance on the copyright headers themselves.

#### `copyright.sh`

This script runs `copyright.py` on modified files and reports/applies the results. By default, the current HEAD commit on the source branch is compared to the work tree, and files that

1. are different between these two trees and
2. change under have outdated copyright header

are reported. This behavior can be changed by

1. Specifying an `--rev=REV` argument, which uses `REV` instead of HEAD as the base of the comparison. A typical use case is to specify `--rev=HEAD^` to check the HEAD commit.

2. Specifying `--copyright=<mode>`, which alters the level of copyright checking is done:

   **off**
   : does not check copyright headers at all

   **year**
   : only update copyright year in new-format copyright headers

   **add**
   : in addition to `year`, add copyright headers to files that do not have any

   **update**
   : in addition to `year` and `add`, also update new-format copyright headers if they are broken or outdated

   **replace**
   : replace any copyright header with a new-format copyright header

   **full**
   : do all of the above

By default, `update-*` refuses to update dirty files (i.e., that differ between the disk and the index) to make it easy to revert the changes. This can be overridden by adding a `-f`/`--force` option.

#### clang-format.sh

This script runs `clang-format` on modified files and reports/applies the results. By default, the current HEAD commit on the source branch is compared to the work tree, and files that

1. are different between these two trees and
2. change under clang-format

are reported. This behavior can be changed by

1. Specifying an `--rev=REV` argument, which uses `REV` instead of HEAD as the base of the comparison. A typical use case is to specify `--rev=HEAD^` to check the HEAD commit.
2. Specifying an action:
   - `check-*`: reports the files that clang-format changes
   - `diff-*`: prints the actual diff of what would change
   - `update-*`: applies the changes to the repository
   - `*-workdir`: operates on the working directory (files on disk)
   - `*-index`: operates on the index of the repository

   For convenience, if you omit the workdir/index suffix, workdir is assumed (i.e., `diff` equals `diff-workdir`).
3. Specifying `--format=off`, which does not run clang-format.

By default, `update-*` refuses to update dirty files (i.e., that differ between the disk and the index) to make it easy to revert the changes. This can be overridden by adding a `-f`/`--force` option.

Since the behaviour of clang-format can change between versions even when using the same options, only clang-format from Clang 18 will give correct results. The path to the correct `clang-format`

binary can be specified via `CLANG_FORMAT` environment variable or by running `git config hooks.clangformatpath /path/to/clang-format-18` in the repository root.

#### clang-tidy.sh

This script runs the `clang-tidy` source code checker on modified files and either reports or applies resulting changes. By default, the current HEAD commit on the source branch is compared to the work tree, and files that

1. are different between these two trees and
2. change when applying clang-tidy

are reported. This behavior can be changed by

1. Specifying an `--rev=REV` argument, which uses `REV` instead of HEAD as the base of the comparison. A typical use case is to specify `--rev=HEAD^` to check the HEAD commit.
2. Specifying an action:
   - `check-*`: reports the files that clang-format changes
   - `diff-*`: prints the actual diff of what would change
   - `update-*`: applies the changes to the repository
   - `*-workdir`: operates on the working directory (files on disk)
   - `*-index`: operates on the index of the repository

   For convenience, if you omit the workdir/index suffix, workdir is assumed (i.e., `diff` equals `diff-workdir`).
3. Specifying `--tidy=off`, which does not run clang-tidy.

By default, `update-*` refuses to update dirty files (i.e., that differ between the disk and the index) to make it easy to revert the changes. This can be overridden by adding a `-f`/`--force` option.

### Black

The Black tool reformats Python files in place, by default. To check and update the entire repository, use the `.black.toml` config file in the root directory of the repository:

```
pip install black
black --config .black.toml .
```

### git pre-commit hook

If you want to run `copyright.sh`, `clang-tidy.sh` and/or `clang-format.sh` automatically for changes you make, you can configure a pre-commit hook using `admin/git-pre-commit`:

1. Copy the `git-pre-commit` script to .git/hooks/pre-commit.
2. Specify the paths to `run-clang-tidy` and `clang-format` for the hook if you have not already done so:

```
git config hooks.runclangtidypath /path/to/run-clang-tidy.py
git config hooks.clangformatpath /path/to/clang-format
```

3. Set the operation modes for the hook:

```
git config hooks.clangtidymode check
git config hooks.clangformatmode check
git config hooks.copyrightmode  update
```

With this configuration, all source files modified in the commit are run through the code formatting tool, are checked with clang-tidy and also checked for correct copyright headers. If any file would be changed by `clang-tidy.sh`, `clang-format.sh` or `copyright.sh`, the names of those files are reported and the commit is prevented. The issues can be fixed by running the scripts manually.

To disable the hook without removing the `pre-commit` file, you can set

```
git config hooks.clangtidymode off
git config hooks.copyrightmode off
git config hooks.clangformatmode off
```

To disable it temporarily for a commit, set NO_FORMAT_CHECK environment variable. For example,

```
NO_FORMAT_CHECK=1 git commit -a
```

You can also run `git commit --no-verify`, but that also disables other hooks.

Note that when you run `git commit --amend`, the hook is only run for the changes that are getting amended, not for the whole commit. During a rebase, the hook is not run.

The actual work is done by the `admin/clang-tidy.sh`, `admin/clang-format.sh` and `admin/copyright.sh` scripts, which get run with the `check-index` action, and with `--copyright` and `--format` getting set according to the `git config` settings.

#### reformat_all.sh

This script runs clang-format, `copyright.py`, or the include sorter for all applicable files in the source tree. See `reformat_all.sh -h` for the invocation.

The script can also produce the list of files for which these commands would be run. To do this, specify `list-files` on the command line and use `--filter=<type>` to specify which command to get the file list for. This can be used together with, e.g., `xargs` to run other scripts on the same set of files.

For all the operations, it is also possible to apply patters (of the same style that various git commands accept, i.e., `src/*.cpp` matches all `.cpp` files recursively under `src/`). The patterns can be specified with `--pattern=<pattern>`, and multiple `--pattern` arguments can be given.

`-f`/`--force` is necessary if the working tree and the git index do not match.

#### Using git filters

An alternative to using a pre-commit hook to automatically apply clang-format on changes is to use a git filter (does not require either of the scripts, only the `.gitattributes` file). You can run

```
git config filter.clangformat.clean \
    "/path/to/clang-format -i"
```

To configure a filter for all files that specify `filter=complete_formatting` attribute that indicates that all formatting steps should be performed.

The pre-commit hook + manually running the scripts gives better/more intuitive control (with the filter, it is possible to have a work tree that is different from HEAD and still have an empty `git diff`) and provides better performance for changes that modify many files. It is the only way that currently also checks the copyright headers.

The filter allows one to transparently merge branches that have not been run through the source checkers, and is applied more consistently (the pre-commit hook is not run for every commit, e.g., during a rebase).

#### Hiding formatting commits from `git blame`

A large-scale code reformatting, for example, when switching to a new clang-format version, might make the output of `git blame`/`git praise` hard to parse, since many lines will be touched by reformatting without any functional change.

A manually-managed list of such formatting-only commits is kept in the `.git-blame-ignore-revs` file. Please run the following command in the repository root to instruct Git to "skip" the listed commits and instead show the earlier commit from which the line originates

```
git config blame.ignoreRevsFile .git-blame-ignore-revs
```

To temporarily disable this option, use `git blame --ignore-revs-file=` (without any argument).

### 8.8.5 Unit testing

The main goal of unit tests in GROMACS is to help developers while developing the code. They focus on testing functionality of a certain module or a group of closely related modules. They are designed for quick execution, such that they are easy to run after every change to check that nothing has been broken.

### Finding, building and running

As described in *Source code organization* (page 746), `src/gromacs/` is divided into modules, each corresponding to a subdirectory. If available, unit tests for that module can be found in a `tests/` subdirectory under the top-level module directory. Typically, tests for code in *`file`*`.h` in the module is in a corresponding `tests/`*`file`*`.cpp`. Not all files have corresponding tests, as it may not make sense to test that individual file in isolation. Focus of the tests is on functionality exposed outside the module. Some of the tests, in particular for higher-level modules, are more like integration tests, and test the functionality of multiple modules. Shared code used to implement the tests is in `src/external/googletest/` and `src/testutils/` (see below).

The tests are built if `BUILD_TESTING=ON` (the default) and `GMX_BUILD_UNITTESTS=ON` (the default) in CMake. Each module produces at least one separate unit test binary (*`module`*`-test`) under `bin/`, which can execute tests for that module.

The tests can be executed in a few different ways:

- Build the `test` target (e.g., `make test`): This runs all the tests using CTest. This includes also the regression tests if CMake has been told where to find them (regression tests are not discussed further on this page). If some of the tests fail, this only prints basic summary information (only a pass/fail status for each test binary or regression test class). You can execute the failing test binaries individually to get more information on the failure. Note that `make test` does not rebuild the test binaries if you have changed the source code, so you need to separately run `make` or `make tests`. The latter only builds the test binaries and their dependencies.
- Build the `check` target (e.g., `make check`): This behaves the same as the `test` target, with a few extensions:
  1. Test binaries are rebuilt if they are outdated before the tests are run.
  2. If a test fails, the output of the test binary is shown.
  3. If unit tests and/or regression tests are not available, a message is printed.
- The implementation of `make check` calls CTest via the `ctest` binary to run all the individual test binaries. More fine-grained control is available there, e.g. filtering by test name or label, or increasing verbosity.
- Directly executing a test binary. This provides the most useful output for diagnosing failures, and allows debugging test failures. The output identifies the individual test(s) that fail, and shows the results of all failing assertions. Some tests also add extra information to failing assertions to make it easier to identify the reason. Some tests are skipped because they cannot run with the number of MPI ranks or GPU devices detected. Explicit information about such cases can be obtained by using the `-echo-reasons` flag to the test binary. It is possible to control which tests are run using command line options. Execute the binary with `--help` to get additional information.

When executed using CTest, the tests produce XML output in `Testing/Temporary/`, containing the result of each test as well as failure messages. This XML is used by GitLab CI for reporting the test status for individual tests. Note that if a test crashes or fails because of an assert or a gmx_fatal() call, no XML is produced for the binary, and CI does not report anything for the test binary. The actual error is only visible in the console output.

### Unit testing framework

The tests are written using Google Test, which provides a framework for writing unit tests and compiling them into a test binary. Most of the command line options provided by the test binaries are implemented by Google Test. See the Google Test Primer for an introduction. Google Test is included in the source tree under `src/external/googletest/`, and is compiled as part of the unit test build.

`src/testutils/` contains GROMACS-specific shared test code. This includes a few parts:

- CMake macros for declaring test binaries. These take care of providing the `main()` method for the test executables and initializing the other parts of the framework, so that the test code in modules can focus on the actual tests. This is the only part of the framework that you need to know to be able to write simple tests: you can use `gmx_add_unit_test()` in CMake to create your test binary and start writing the actual tests right away. See `src/testutils/TestMacros.cmake` and existing CMake code for examples how to use them.
- Generic test fixtures and helper classes. The C++ API is documented on Doxygen page for testutils. Functionality here includes locating test input files from the source directory and constructing temporary files, adding custom command line options to the test binary, some custom test assertions for better exception and floating-point handling, utilities for constructing command line argument arrays, and test fixtures for tests that need to test long strings for correctness and for tests that execute legacy code where `stdin` reading etc. cannot be easily mocked.
- Some classes and functions to support the above. This code is for internal use of the CMake machinery to build and set up the test binaries, and to customize Google Test to suit our environment.
- Simple framework for building tests that check the results against reference data that is generated by the same test code. This can be used if it is not easy to verify the results of the code with C/C++ code alone, but manual inspection of the results is manageable. The general approach is documented on the Doxygen page on using the reference data.

In addition to `src/testutils/`, some of the module test directories may provide reusable test code that is used in higher-level tests. For example, the `src/gromacs/analysisdata/tests/` provides test fixtures, a mock implementation for `gmx::IAnalysisDataModule`, and some helper classes that are also used in `src/gromacs/trajectoryanalysis/tests/`. These cases are handled using CMake object libraries that are linked to all the test binaries that need them.

### Getting started with new tests

To start working with new tests, you should first read the Google Test documentation to get a basic understanding of the testing framework, and read the above description to understand how the tests are organized in GROMACS. It is not necessary to understand all the details, but an overall understanding helps to get started.

Writing a basic test is straightforward, and you can look at existing tests for examples. The existing tests have a varying level of complexity, so here are some pointers to find tests that use certain functionality:

- `src/gromacs/utility/tests/stringutil.cpp` contains very simple tests for functions. These do not use any fancy functionality, only plain Google Test assertions. The only thing required for these tests is the `TEST()` macro and the block following it, plus headers required to make them compile.
- The same file contains also simple tests using the reference framework to check line wrapping (the tests for `gmx::TextLineWrapper`). The test fixture for these tests is in `src/testutils/`

`include/testutils/stringtest.h/.cpp`. The string test fixture also demonstrates how to add a custom command line option to the test binary to influence the test execution.

- `src/gromacs/selection/tests/` contains more complex use of the reference framework. This is the code the reference framework was originally written for. `src/gromacs/selection/tests/selectioncollection.cpp` is the main file to look at.
- For more complex tests that do not use the reference framework, but instead do more complex verification in code, you can look at `src/gromacs/selection/tests/nbsearch.cpp`.
- For complex tests with mock-up classes and the reference framework, you can look at `src/gromacs/analysisdata/tests/`.

Here are some things to keep in mind when working with the unit tests:

- Try to keep the execution time for the tests as short as possible, while covering the most important paths in the code under test. Generally, tests should take seconds instead of minutes to run, so that no one needs to hesitate before running the tests after they have done some changes. Long-running tests should go somewhere else than in the unit test set. Note that CI will run the tests in several build configuration and slow tests will significantly slow down the pipelines and can even cause them to timeout.
- Try to produce useful messages when a test assertion fails. The assertion message should tell what went wrong, with no need to run the *test itself* under a debugger (e.g., if the assertion is within a loop, and the loop index is relevant for understanding why the assertion fails, it should be included in the message). Even better if even a user can understand what goes wrong, but the main audience for the messages is the developer who caused the test to fail.

#### MPI tests

If your test makes specific requirements on the number of MPI ranks, or needs a communicator as part of its implementation, then there are GROMACS-specific extensions that make normal-looking GoogleTests work well in these cases. Use `GMX_TEST_MPI(RankRequirement)` and declare the test with `gmx_add_mpi_unit_test` to teach `CTest` how to run the test regardless of whether the build is with thread-MPI or real MPI. See `src/testutils/include/mpitest.h` for details.

### 8.8.6 Physical validation

Physical validation tests check whether simulation results correspond to physical (or mathematical) expectations.

Unlike the existing tests, we are not be able to keep these tests in the "seconds, not minutes" time frame, rather aiming for "hours, not days". They should therefore be ran periodically, but probably not for every build.

Also, given the long run time, it will in many cases be necessary to separate running of the systems (e.g. to run it at a specific time, or on a different resource), such that the make script does give the option to

- prepare run files and an execution script,
- analyze already present simulations,
- or prepare, run and analyze in one go.

### Test description

Currently, simulation results are tested against three physically / mathematically expected results:

- *Integrator convergence*: A symplectic integrator can be shown to conserve a constant of motion (such as the energy in a micro-canonical simulation) up to a fluctuation that is quadratic in time step chosen. Comparing two or more constant-of-motion trajectories realized using different time steps (but otherwise unchanged simulation parameters) allows a check of the symplecticity of the integration. Note that lack of symplecticity does not necessarily imply an error in the integration algorithm, it can also hint at physical violations in other parts of the model, such as non-continuous potential functions, imprecise handling of constraints, etc.
- *Kinetic energy distribution*: The kinetic energy trajectory of a (equilibrated) system sampling a canonical or an isothermal-isobaric ensemble is expected to be Maxwell-Boltzmann distributed. The similarity between the physically expected and the observed distribution allows to validate the sampled kinetic energy ensemble.
- *Distribution of configurational quantities*: As the distribution of configurational quantities like the potential energy or the volume are in general not known analytically, testing the likelihood of a trajectory sampling a given ensemble is less straightforward than for the kinetic energy. However, generally, the ratio of the probability distribution between samples of the same ensemble at different state points (e.g. at different temperatures, different pressures) is known. Comparing two simulations at different state points therefore allows a validation of the sampled ensemble.

The physical validation included in GROMACS tests a range of the most-used settings on several systems. The general philosophy is to leave most settings to default values with the exception of the ones explicitly tested in order to be sensitive to changes in the default values. The test set will be enlarged as we discover interesting test systems and corner cases. Under double precision, some additional tests are run, and some other tests are run using a lower tolerance.

### Integrator convergence

All simulations performed under NVE on Argon (1000 atoms) and water (900 molecules) systems. As these tests are very sensitive to numerical imprecision, they are performed with long-range corrections for both Lennard-Jones and electrostatic interactions, with a very low pair-list tolerance (`verlet-buffer-tolerance = 1e-10`), and high LINCS settings where applicable.

**Argon**:

- *Integrators*: - `integrator = md` - `integrator = md-vv`
- *Long-range corrections LJ*: - `vdwtype = PME` - `vdwtype = cut-off`, `vdw-modifier = force-switch`, `rvdw-switch = 0.8`

**Water**:

- *Integrators*: - `integrator = md` - `integrator = md-vv`
- *Long-range corrections LJ*: - `vdwtype = PME` - `vdwtype = cut-off`, `vdw-modifier = force-switch`, `rvdw-switch = 0.8`
- *Long-range corrections electrostatics*: - `coulombtype = PME`, `fourierspacing = 0.05`
- *Constraint algorithms*: - `constraint-algorithm = lincs`, `lincs-order = 6`, `lincs-iter = 2` - `constraint-algorithm = none` - SETTLE

### Ensemble tests

The generated ensembles are tested with Argon (1000 atoms) and water (900 molecules, with SETTLE and PME) systems, in the following combinations:

- `integrator = md`, `tcoupl = v-rescale`, `tau-t = 0.1`, `ref-t = 87.0` (Argon) or `ref-t = 298.15` (Water)
- `integrator = md`, `tcoupl = v-rescale`, `tau-t = 0.1`, `ref-t = 87.0` (Argon) or `ref-t = 298.15` (Water), `pcoupl = parrinello-rahman`, `ref-p = 1.0`, `compressibility = 4.5e-5`
- `integrator = md-vv`, `tcoupl = v-rescale`, `tau-t = 0.1`, `ref-t = 87.0` (Argon) or `ref-t = 298.15` (Water)
- `integrator = md-vv`, `tcoupl = nose-hoover`, `tau-t = 1.0`, `ref-t = 87.0` (Argon) or `ref-t = 298.15` (Water), `pcoupl = mttk`, `ref-p = 1.0`, `compressibility = 4.5e-5`

All thermostats are applied to the entire system (`tc-grps = system`). The simulations run for 1ns at 2fs time step with Verlet cut-off. All other settings left to default values.

### Building and testing using the build system

Since these tests can not be ran at the same frequency as the current tests, they are kept strictly opt-in via `-DGMX_PHYSICAL_VALIDATION=ON`, with `-DGMX_PHYSICAL_VALIDATION=OFF` being the default. Independently of that, all previously existing build targets are unchanged, including `make check`.

If physical validation is turned on, a number of additional make targets can be used:

- `make check` is unchanged, it builds the main binaries and the unit tests, then runs the unit tests and, if available, the regression tests.
- `make check-phys` builds the main binaries, then runs the physical validation tests. **Warning**: This requires to simulate all systems and might take several hours on a average machine!
- `make check-all` combines `make check` and `make check-phys`.

As the simulations needed to perform the physical validation tests may take long, it might be advantageous to run them on an external resource. To enable this, two additional make targets are present:

- `make check-phys-prepare` prepares all simulation files under `tests/physicalvalidation` of the build directory, as well as a rudimentary run script in the same directory.
- `make check-phys-analyze` runs the same tests as `make check-phys`, but does not simulate the systems. Instead, this target assumes that the results can be found under `tests/physicalvalidation` of the build directory.

The intended usage of these additional targets is to prepare the simulation files, then run them on a different resource or at a different time, and later analyze them. If you want to use this, be aware *(i)* that the run script generated is very simple and might need (considerable) tuning to work with your setup, and *(ii)* that the analysis script is sensitive to the folder structure, so make sure to preserve it when copying the results to / from another resource.

Additionally to the mentioned make targets, a number of internal make targets are defined. These are not intended to be used directly, but are necessary to support the functionality described above, especially the complex dependencies. These internal targets include `run-ctest`, `run-ctest-nophys`, `run-ctest-phys` and `run-ctest-phys-analyze` running the different tests, `run-physval-sims` running the simulations for physical validation, and `missing-tests-notice`, `missing-tests-notice-all`, `missing-phys-val-phys`, `missing-phys-val-phys-analyze` and `missing-phys-val-all` notifying users about missing tests.

### Direct usage of the python script

The `make` commands mentioned above are calling the python script `tests/physicalvalidation/gmx_physicalvalidation.py`, which can be used independently of the make system. Use the `-h` flag for the general usage information, and the `--tests` for more details on the available physical validations.

The script requires a `json` file defining the tests as an input. Among other options, it allows to define the GROMACS binary and the working directory to be used, and to decide whether to only prepare the simulations, prepare and run the simulations, only analyze the simulations, or do all three steps at once.

### Adding new tests

The available tests are listed in the `systems.json` (tests standardly used for single precision builds) and `systems_d.json` (tests standardly used for double precision builds) files in the same directory, the GROMACS files are in the folder `systems/`.

The `json` files lists the different test. Each test has a `"name"` attribute, which needs to be unique, a `"dir"` attribute, which denotes the directory of the system (inside the `systems/` directory) to be tested, and a `"test"` attribute which lists the validations to be performed on the system. Additionally, the optional `"grompp_args"` and `"mdrun_args"` attributes allow to pass specific arguments to `gmx grompp` or `gmx mdrun`, respectively. A single test can contain several validations, and several independent tests can be performed on the same input files.

To add a new test to a present system, add the test name and the arguments to the `json` file(s). To use a new system, add a subfolder in the `systems/` directory containing `input/system.{gro,mdp,top}` files defining your system.

## 8.9 Known issues relevant for developers

This is a non-exhaustive list of known issues that have been observed and can be of interest for developers. These have not been solved because they are either outside the scope of the GROMACS project or are are simply too difficult or tedious to address ourselves.

### 8.9.1 Issues with GPU timer with OpenCL

When building using OpenCL in `Debug` mode, it can happen that the GPU timer state gets corrupted, leading to an assertion failure during the *mdrun* (page 258). This seems to be related to the load of other, unrelated tasks on the GPU.

### 8.9.2 GPU emulation does not work

The non-bonded GPU emulation mode does not work, at least for builds with GPU support; then a GPU setup call is called. Also dynamic pruning needs to be implemented for GPU emulation.

### 8.9.3 OpenCL on NVIDIA Volta and later broken

The OpenCL code produces incorrect results on Volta and Turing GPU architectures from NVIDIA (CC 7.0 and 7.5). This is an issue that affects certain flavors of the nonboded kernels, most likely a result of miscompilation, and there is no known workaround.

### 8.9.4 PME decomposition automated task assignment broken

When there are two or more ranks on a node doing combined PP and PME work (i.e no separate PME ranks) and more GPUs are detected than ranks, the automated task assignment fails and GROMACS aborts with "Error in user input" message. You can work around this by using `-gpu_id` or `GMX_GPU_ID` or limiting the number of visible GPUs.

Issue 4684

CHAPTER

NINE

# DOXYGEN DOCUMENTATION

The doxygen code documentation is available on the GROMACS webpage.

CHAPTER

TEN

# C++ API

## 10.1 Public C++ API

### 10.1.1 Overview

Trajectory analysis tools and pluggable MD extensions (such as code based on the sample_restraint example) use `gromacs/` headers supported by `libgromacs`.

Software that uses new public API facilities (such as *gmxapi* (page 718)) uses CMake and `find_package(gmxapi)` to configure a build system to use the `gmxapi/` headers and link to the library supporting the `::gmxapi` C++ namespace.

Currently, the `gmxapi` library conveys an indirect dependency on `libgromacs`. Due to a bug in CMake 3.24.0, `find_package(gmxapi)` must implicitly call `find_package(gromacs${GROMACS_SUFFIX})` to avoid a spurious error, even though client software does not generally need to explicitly use *Gromacs::libgromacs* (page 835) or its details.

### 10.1.2 Client build system support

GROMACS relies heavily on CMake to configure and manage the build system. The GROMACS installation directly supports CMake configured client software through configuration and "hints" files installed to `$GROMACS_ROOT/share/cmake/`.

**`gmx --version`** (or the appropriate `gmx$GROMACS_SUFFIX`) includes notes on the original build toolchain that may or may not be sufficient for configuring the client software build system.

#### Compiler toolchain

Though not explicitly required, it is highly recommended that client software build with a toolchain that closely matches that of the GROMACS build to avoid binary incompatibilities.

Each GROMACS installation (since 2022) provides a CMake "hints" file that can be used to initialize your **`cmake`** cache with the `-C` option.

For a GROMACS installation in `$GROMACS_ROOT/`, the hints file for a given *GROMACS_SUFFIX* (page 836) can be found at `$GROMACS_ROOT/share/cmake/gromacs$GROMACS_SUFFIX/gromacs-hints$GROMACS_SUFFIX.cmake`

The hints file is completely separate from the CMake configuration files that support `find_package(gmxapi)` and `find_package(GROMACS)`.

However, using `-C path/to/gromacs-hints$GROMACS_SUFFIX.cmake` in your client **`cmake`** configuration command line can help set appropriate compiler options so that you have a better chance of building a compatible binary. (I.e. it helps *`gromacs_check_compiler()`* (page 837) succeed.)

In addition to hints variables for CMake `enable_language` and `find_package`, the hints file sets `GMX_CMAKE_VERSION` in case the client build system needs to know the version of CMake that was used to build the GROMACS installation.

#### MPI support

GROMACS uses FindMPI (the module that supports CMake `find_package(MPI ...)`) to locate and configure compiler and linker options for MPI support. Client software is advised to do the same.

If software support for MPI was detected by GROMACS when built, the *gromacs-hints* file (see above) will define input variables to help `find_package` locate the same MPI installation.

#### Caveats

If GROMACS is installed from a package built in a different environment, the embedded toolchain information may be inaccurate. This could make the **`gmx --version`** output misleading and the *gromacs-hints* file useless. You may encounter spurious warnings when configuring the client build system, and the client software may or may not interact properly with the GROMACS installation.

In a computing environment with multiple toolchains available (such as a typical High Performance Computing (HPC) cluster), the toolchain may depend on environment variables for consistent behavior. If environment modules were used when setting up the GROMACS build environment (e.g. **`module load gcc openmpi/gcc`**), it may be necessary to load the same environment modules before building the client software.

### 10.1.3 `gmxapi` CMake package

The CMake configuration files installed with GROMACS support the "Config mode" of CMake find_package. Unlike the `gromacs$GROMACS_SUFFIX` packages, CMake configuration files only support a single `gmxapi` package name.

The `gmxapi` API and ABI hide most of the differences possible in `libgromacs` from different build options. However, the `gmxapi/mpi/resourceassignment.h` interface is affected by the original choice of *`GMX_MPI`* (page 753). A stable interface is available to MPI-enabled client software through the `gmxapi/mpi/gmxapi_mpi.h` template header.

Some GROMACS installations include multiple builds. For instance, there may be a `libgromacs.so`, `libgromacs_d.so`, `libgromacs_mpi.so`, and `libgromacs_mpi_d.so`, (according to build-time values of *`GMX_DOUBLE`* (page 753) and *`GMX_MPI`* (page 753)) any *one* of which might be provided by the `Gromacs::libgromacs` CMake target. Until resolution of Issue 4334, only one version of the `Gromacs::gmxapi` is importable from a GROMACS installation. Each GROMACS installation (with *`GMXAPI`* (page 755) `ON`) overwrites the CMake configuration files for the previously installed gmxapi support.

#### Imported target

**Gromacs::gmxapi**

The gmxapi package provides a single Gromacs::gmxapi target that conveys access to the installed gmxapi/ headers. The associated shared object library will be differently named, depending on the build system configuration options. (See *GMX_DOUBLE* (page 753) and *GMX_MPI* (page 753)).

### 10.1.4 gromacs (and gromacs$GROMACS_SUFFIX packages)

The CMake machinery to support find_package(GROMACS) has two parts: a FindGROMACS.cmake find module (found in share/gromacs/template/cmake/ in the installation and share/template/cmake/ in the source tree), and actual package configuration files (gromacs-config.cmake and supporting files installed to share/cmake/ from input files in src/gromacs/).

FindGROMACS.cmake is a simple wrapper over the package configuration files, providing a somewhat more convenient interface to the machinery that supports multiple suffixed GROMACS installations in the same installation prefix (see GROMACS_SUFFIX variable below). This file is intended to be version-agnostic and remain both forward- and backward-compatible even between major GROMACS releases. All version-specific information and the actual details about the compilation and linking settings is in the package configuration files. Build systems willing to utilize FindGROMACS.cmake can create a local copy of it and use it like it is used in the installed share/gromacs/template/CMakeLists.txt. The package configuration files can also be used directly if desired, bypassing FindGROMACS.cmake.

When using FindGROMACS.cmake, find_package(GROMACS) is able to find configurations for any of the gromacs, gromacs_d, gromacs_mpi, or gromacs_mpi_d CMake package names. Otherwise, you must use the exact package name that you are looking for. E.g. find_package(gromacs_d).

#### Imported targets

**Gromacs::libgromacs**

Provides access to the installed core GROMACS library and gromacs/ headers: target_link_libraries(foo PRIVATE Gromacs::libgromacs).

**Gromacs::gmx**

Represents the command line executable. For example, to set a local CMake variable _gmx_executable to the executable path (with the correct *GROMACS_SUFFIX* (page 836)) you can use get_target_property(_gmx_executable Gromacs::gmx LOCATION) in your CMakeLists.txt

### Input options

Input options for influencing what to find

**GROMACS_SUFFIX**

(only for `FindGROMACS.cmake`)

This CMake variable can be set before calling `find_package(GROMACS)` to specify the GROMACS suffix to search for. If not set, an unsuffixed version is searched for. If using the package configuration files directly, the suffix must be set using `find_package(GROMACS NAMES gromacs<suffix>)`.

**GROMACS_PREFER_STATIC**

This CMake variable can be set before calling `find_package(GROMACS)` to specify whether static or shared libraries are preferred if both are available. It does not affect which GROMACS installation is chosen, but if that installation has both static and shared libraries available (installed from two different builds with the same suffix), then this chooses the library to be returned in `GROMACS_LIBRARIES`.

**GROMACS_DIR**

This CMake (cache) variable is a standard mechanism provided by `find_package`, and can be used to specify a hint where to search for GROMACS. Also `CMAKE_PREFIX_PATH` can be used for this purpose; see CMake documentation for `find_package` for more details. `GROMACS_DIR` can also be set as an environment variable, and this is done by `GMXRC`.

### Output variables

Output variables that specify how the found `libgromacs` and header should be used:

**GROMACS_INCLUDE_DIRS**

List of include directories necessary to compile against the GROMACS headers. Currently, this includes the path to GROMACS headers.

**GROMACS_LIBRARIES**

List of libraries to link with to link against GROMACS. Under the hood, this uses imported CMake targets to represent `libgromacs`.

**GROMACS_DEFINITIONS**

List of compile definitions (with `-D` in front) that are required to compile the GROMACS headers.

**GROMACS_IS_DOUBLE**

Whether the found GROMACS was compiled in double precision.

**GROMACS_CXX_FLAGS**

Required compiler flags.

### Macros/functions

Declared macros/functions that can be used for checking for correctness of some settings:

**gromacs_check_double**(*GMX_DOUBLE*)

Checks that the found GROMACS is in the expected precision. The parameter GMX_DOUBLE should be the name of a cache variable that specified whether double-precision was requested.

**gromacs_check_compiler**(*LANG*)

Checks that the found GROMACS was compiled with the same compiler that is used by the current CMake system. Currently only LANG=CXX is supported.

Public C++ application programming interfaces are available for GROMACS installations depending on the detected environment and user options when the GROMACS build is configured with CMake.

- ***Public C++ API*** **(page 833)**
  - CMake target Gromacs::gmxapi, enabled by *GMXAPI* (page 755) (default, when BUILD_SHARED_LIBS on non-Windows platforms), provides gmxapi/ headers and ::gmxapi C++ namespace.
  - CMake target Gromacs::libgromacs, enabled by *GMX_INSTALL_LEGACY_API* (page 755) (default OFF), provides gromacs/ headers and ::gmx C++ namespace.
- *(Non-)Bonded LIBrary (NB-LIB) API* (page 734): Enabled by *GMX_INSTALL_NBLIB_API* (page 755). (default, when BUILD_SHARED_LIBS on non-Windows platforms)

CHAPTER

ELEVEN

# RELEASE NOTES

These release notes record the changes made in all major and patch releases of GROMACS. Major releases contain changes to the functionality supported, whereas patch releases contain only fixes for issues identified in the corresponding major releases.

Two version series of GROMACS are under active maintenance and within support lifetime at any time. In 2027, they are the 2027 series and the 2026 series. In the latter, only highly conservative fixes will be made, and only to address issues that affect scientific correctness. Naturally, some of those releases will be made after the year 2026 ends, but we keep the year of the original release in the version name so that users understand how up to date their version is. Such fixes will also be incorporated into the more recent release series, as appropriate. Around the time the 2028 release is made, the 2026 series will no longer be maintained.

Where issue numbers are reported in these release notes, more details can be found on the issue tracker at that issue number.

## 11.1 GROMACS 2027 series

### 11.1.1 Patch releases

### 11.1.2 Major release

#### Highlights

GROMACS 2027 was released on INSERT DATE HERE. Patch releases may have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

As always, we've got several useful performance improvements, with or without GPUs, all enabled and automated by default. In addition, several new features are available for running simulations. We are extremely interested in your feedback on how well the new release works on your simulations and hardware. The new features are:

-

### New and improved features

#### Multiple molecule types with identical SETTLE parameters are now supported

This can be useful when one, for instance, wants to apply position restraints to a subset of water molecules.

#### Periodic SCC-DFTB, GFN1-xTB and GFN2-xTB are available for CP2K QM/MM

The CP2K QM/MM interface can now generate periodic SCC-DFTB, GFN1-xTB and GFN2-xTB inputs using short-range regularized point-charge coupling with SPME electrostatics. For GFN1-xTB and GFN2-xTB CP2K should be compiled with support for the tblite interface. The user can select Gaussian Expansion of Electrostatic Potential (GEEP) for MM charges in tight-binding methods through the `qmmm-cp2k-dftb-electrostatic-coupling = gauss` MDP option, which requires a CP2K 2027.1 or higher, whereas the default is short-range regularized point-charge coupling `qmmm-cp2k-dftb-electrostatic-coupling = point-charge`. Regular DFT methods are still using GEEP (Gauss) for electrostatics.

### Performance improvements

#### Improved performance of systems that are inhomogeneous along z

The non-bonded kernels got very inefficient with systems that are have a very inhomogeneous atom distribution along the z-dimension. Now all three dimensions are considered when estimating atom density.

Issue 5622

### Changes to the API

#### Direct storage of `TimeControl` affects functions in `pargs.h` and `trxio.h`

During command line argument parsing, time control flags (`-b`, `-e` and `-dt`) are set to an input `gmx::TimeControl` struct (in `gromacs/fileio/timecontrol.h`) instead of to global variables. The following functions are affected:

`parse_common_args` takes an extra `gmx::TimeControl*` input, which can be `nullptr`. If time control is desired a non-`nullptr` object must be passed.

`read_first_frame` and `read_first_x` both take an extra `gmx::TimeControl*` input, which can be `nullptr` if time control is not desired.

`check_times` takes an extra `gmx::TimeControl&` input containing the time control parameters to check against.

### Improvements to GROMACS tools

#### `gmx pdb2gmx` now resolves generated hydrogen names from the residue topology

When `gmx pdb2gmx` adds hydrogens from the hydrogen database, it now looks up their names in the residue topology entry of the selected force field instead of relying on hard-coded numbering rules. This allows force fields that use different hydrogen naming schemes, such as original AMBER/IUPAC names for methylene hydrogens, to work without naming-specific code paths or command-line options.

### Bugs fixed

#### PDB with a box of size 1 Angstrom are interpreted as having no PBC

PDB files with structures not determined from X-ray should have a unit-cell with P=1 and dimensions of 1 Angstrom to indicate no periodic boundary conditions. GROMACS now interprets such structures as not having PBC.

Issue 4645, Issue 5679

#### Analysis tools now always process time values in double precision

This avoids picking incorrect frames when times values are large.

### Deprecated functionality

### Changes anticipated to GROMACS 2027 functionality

### Functionality deprecated in GROMACS 2027

#### Double-batched FFT (BBFFT) backend

The BBFFT backend has been deprecated in GROMACS following Intel's abandonment of the project. Users on Intel GPU hardware are advised to rely on the default oneMKL backend.

Issue 5658

### Removed functionality

#### Support for version 1 of the hardware locality library `hwloc` is removed

Building with `-DGMX_HWLOC=ON` now requires `hwloc` 2.0 or later, which has been supported in GROMACS for several years.

#### Google Native Client (NaCl) support is removed

The `GMX_NACL` build option has been removed.

Issue 5388

#### Portability

#### Miscellaneous

#### Renamed all NBNXN environment variables to NBNXM

All environment variables starting with `GMX_NBNXN` have been renamed to start with `GMX_NBNXM` for consistency with the algorithm's name.

#### Replaced usage of custom Bohr radius value in gmx spatial with the common value from units

The *gmx spatial* (page 322) command used to have its own definition of Bohr radius. For consistency with other parts of GROMACS, it now uses the definition of Bohr radius from the same source as the rest of the code. Notably, the value of the constant changed from `0.529177249` (IUPAC 1999) to `0.529177210903` (NIST 2018).

#### AMBER19SB and AMBER14SB force fields now use IUPAC standard hydrogen names

*gmx pdb2gmx* (page 281) now recognizes the hydrogen names from the force field rather than always starting from 1. This means that the AMBER19SB and AMBER14SB force fields now use the original hydrogen names, (i.e. HB2 and HB3 for methylene hydrogens instead of HB1 and HB2). This is more consistent with the naming in the original Amber force field files and with the IUPAC standard for hydrogen names.

## 11.2 GROMACS 2026 series

### 11.2.1 Patch releases

#### GROMACS 2026.4 release notes

This version was released on TODO, 2026. These release notes document the changes that have taken place in GROMACS since the previous 2026.3 version, to fix known issues. It also incorporates all fixes made in version 2025.4 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fix PME heap buffer overflow with multiple PME ranks

When the local PME grid required SIMD padding in the z dimension, major-dimension overlap communication could write beyond its communication buffer. This could corrupt memory or crash `mdrun` when PME used multiple PME ranks, most readily with `pme-order = 5`.

Issue 5669

#### Fixed MPI-parallel `mdrun -plumed` aborting at startup

Any MPI-parallel run using PLUMED aborted at startup with `MPI_ERR_COMM`: PLUMED reads the `setMPIComm` argument as a pointer, but the communicator was passed by value. Only 2026 was affected.

Issue 5671

#### Worked around SYCL miscompilation in PME by oneAPI 2026.x

With all oneAPI versions 2026.x (and until a fix arrives), the SYCL PME GPU gather kernel on sufficiently large systems is affected by a miscompilation that causes a crash. Earlier oneAPI versions are not affected. Results are correct if mdrun does not crash.

The issue is now worked around by always allocating some per-workgroup shared local memory, which can be more than is strictly required, but which is unlikely to affect performance noticeably.

Issue 5688

#### Removed incorrect load imbalance reporting with update GPU

Issue 5700

### Fixes for `gmx` tools

#### Fix stack overflow in `grompp` with large ifdef blocks

A recursion was replaced with a plain loop to avoid `gmx grompp` crashing with large ifdef blocks.

Issue 5683

### Increased precision of time processing in `gmx eneconv`

All internal time processing in `gmx eneconv` now preserves the full double precision of the time stamps. This avoids picking of incorrect frames when times are large and intervals are small.

Issue 5219

### Fixes that affect portability

### Fixed SYCL build with HeFFTe

### Work around icpx compiler crash in localtopology.cpp at RelWithAssert

The Intel LLVM compiler (icpx, observed with oneAPI 2026.0) segfaults in its frontend while optimizing `make_local_bondeds_excls()` in `domdec/localtopology.cpp` at `-O3` with assertions enabled (RelWithAssert configuration). The file is now compiled at `-O2` when building with IntelLLVM 2026.x in RelWithAssert mode; this has no meaningful performance impact as the file contains domain-decomposition setup code, not hot kernels.

Issue 5636

### Miscellaneous

### Stopped installing lmfit content

Some unnecessary CMake code which was problematic with IDEs and CMake version at least 4.3 is now removed.

Issue 5668

### Updated DOI to the preprint describing AMBER14SB and AMBER19SB conversion and validation

AMBER14SB and AMBER19SB force field *itp* (page 578) files contained a placeholder link the preprint describing the conversion and validation steps. Now that the preprint is uploaded on ChemRxiv, the placeholder was replaced with the DOI of the preprint, doi:10.26434/chemrxiv.15006112/v1.

Issue 5690

### Fix FLOPs-based dynamic load balancing

The FLOPs-based DLB algorithm (activated via `GMX_DLB_BASED_ON_FLOPS`) was not accounting for non-bonded kernels, making it ineffectual. The default timer-based DLB was not affected.

Issue 5698

### GROMACS 2026.3 release notes

This version was released on June 25th, 2026. These release notes document the changes that have taken place in GROMACS since the previous 2026.2 version, to fix known issues. It also incorporates all fixes made in version 2025.4 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

##### Avoid deadlock with NVSHMEM and `GMX_DISABLE_STAGED_GPU_TO_CPU_PMEPP_COMM`

Previously, GROMACS would freeze early on with this combination of enabled features.

Issue 5637

#### Fixes for `gmx` tools

##### Fix position restraint reference with pbc=xy and ref-coord-scaling=com

With pbc=xy and ref-coord-scaling=com, `gmx grompp` would incorrectly subtract the COM of the restraint group from the z-component of the reference coordinates. This should in most cases have caused obvious errors.

Issue 5630

##### Prevent buffer overflow in gmx angle

Previously, if `gmx angle` was used to measure a dihedral, an internal loop would execute exactly three times, regardless of how many angles were provided. If fewer than three angles were provided, this led to a heap buffer overflow.

Issue 5620

##### Fix gmx anaeig -over when number of eigenvectors differs

Previously, `gmx anaeig -over -v ev1.trr -v2 ev2.trr -first 1 -last 1` would exit with a fatal error if the two eigenvector files contained a different number of eigenvectors. Computing the subspace overlap of a set of eigenvectors should only require the two eigenvector files to have the same number of atoms, not the same number of stored eigenvectors.

See also this discussion thread

### Fixes that affect portability

### Fixed compilation error with Visual Studio and CUDA 13

### Fixed compilation error with HeFFTe and NVHPC compiler

### Fixed gmxapi build failures with CUDA

In some configurations, building gmxapi could have failed with `The link interface of target "Gromacs::libgromacs" contains:  OpenMP::OpenMP_CUDA but the target was not found.`

Issue 5656

### Miscellaneous

### Update documentation and warnings in gmx angle

Previously, the documentation and warning messages did not specify specific requirements to generate certain optional output files. Another warning message suggests the user uses an option from the legacy version of the tool. Documentation, warnings, and errors were updated for clarity.

Issue 5625

### GROMACS 2026.2 release notes

This version was released on May 5th, 2026. These release notes document the changes that have taken place in GROMACS since the previous 2026.1 version, to fix known issues. It also incorporates all fixes made in version 2025.4 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Fixed PME mixed mode with spread/gather on GPU and FFT and solve on CPU

With PME mixed mode, mdrun would exit with an error about infinite energies or with a CUDA error.

Issue 5336

### Fixed a RuntimeError within the NNPot interface

Fixed a “RuntimeError: Global alloc not supported yet” when running GPU inference on certain NNP models (e.g. Nutmeg) within the NNPot interface.

Issue 5586

### Correctly restore deltaH history from a checkpoint

Was reset to zero. This would only affect results when continuing from a checkpoint that was written at a step that is not a multiple of nstdhdl.

Issue 5584

### Correct cylinder pulling combined with AWH

When using AWH with pull-geometry=cylinder, the pull distance could be incorrect when distances between the pull group and some atoms in the cylinder reference group are above half the box size.

Issue 5629

### Fixed performance issue with GPU FEP kernel

Previously, using `-nbfe gpu` with Gapsys soft-core activated parts of GPU offloading support code while still doing the computation on the CPU, resulting in reduced performance and misleading log messages.

Now `mdrun` correctly identifies that the Gapsys soft-core flavor is not supported by GPU non-bonded force kernels, ensuring users revert to the supported CPU path or switch to Beutler soft-core.

Issue 5579

### Fixed `GMX_NO_CART_REORDER` environment variable

The `GMX_NO_CART_REORDER` environment variable did not disable Cartesian communicator rank reordering when set to 1 because the logic was inverted. This was broken since GROMACS 2020.

### Fixed `mdrun -ddorder cartesian` with PME-only ranks

With PME-only ranks, `mdrun -ddorder cartesian` could fail during domain decomposition setup.

Issue 5612

### Correct wallcycle table in the md.log file with single rank simulations

Issue 5614

### Fixed MPI communicator passing to CP2K

Corrected an issue with GROMACS-CP2K interface in case it is built with support for an external MPI library but `mdrun` runs as a single rank. The interface erroneously passed `MPI_COMM_NULL` to CP2K which caused an error.

### Fixes for `gmx` tools

### Better error reporting in `gmx x2top`

Properly print the filename when a forcefield parsing error is encountered.

Issue 5601

### gmx polystat computed internal distances between all particles

Now only particles in the index group are used.

Issue 5595

### Fixed a selection evaluation issue for numeric expressions

Fixed a selection issue that would lead to a crash when a numeric expression would reference a variable derived from a position expression.

Issue 5576

### Fixes that affect portability

### Facilitated configuring a oneAPI SYCL build with MKL from oneAPI 2026.0 and newer

A configure-time check that MKL can link in a SYCL build referred to a header deprecated in oneAPI 2025.0 that is removed in oneAPI 2026.0. The check now flexibly refers to the new header whenever it exists.

### Improved detection of GPU-awareness in MPI

Previously, GROMACS could fail to detect that the MPI library has extensions to query its GPU support, necessitating the use of `GMX_FORCE_GPU_AWARE_MPI=1` environment variable. Cray MPICH 9 with ROCm was affected.

### Miscellaneous

#### Adjusted fudge coefficients for AMBER14SB and AMBER19SB

AMBER itself uses value $\frac{1}{1.2} = 0.8\overline{3}$. The new value of `0.8333333333333333` for the Lennard-Jones fudge coefficient matches the original value better than the previously used `0.8333`.

Issue 5616

#### Fix SPC and SPC/E atom types in AMBER14SB and AMBER19SB force fields

The atom type for `OW` and `HW` was `OW_spc` and `HW_spc`, which are undefined. These are now corrected to use `OW_spce` and `HW_spce` atom types (respectively), which were converted from Amber ff14SB and ff19SB.

Issue 5615

#### Fix test failure with OMP_NUM_THREADS set

Fix test failure in NbnxmTests when `OMP_NUM_THREADS` was set to a value larger than 1.

Issue 5619

#### Hardened configuration with cuFFTMp and HeFFTe

Previously the CMake options for these distributed GPU FFT libraries would be ignored if `GMX_MPI` was not also set. This led to incorrect builds and confusing warnings from CMake. `GMX_MPI` is required, and now an error is given if support for such an FFT library is required and `GMX_MPI` is not also set.

Issue 5596 Issue 5598

#### Hardened configuration with NVSHMEM

Previously the CMake options for NVSHMEM would be ignored if `GMX_MPI` was not also set. This led to incorrect builds and confusing warnings from CMake. `GMX_MPI` is required, and now an error is given if NVSHMEM support is required and `GMX_MPI` is not also set.

Issue 5596 Issue 5598

### GROMACS 2026.1 release notes

This version was released on March 6th, 2026. These release notes document the changes that have taken place in GROMACS since the previous 2026.0 version, to fix known issues. It also incorporates all fixes made in version 2025.4 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Fixed a performance issue with thread-MPI

Way too many OpenMP threads (up to approximately (number_of_cpu_cores)^2) could be created, severely impacting performance.

Issue 5564

#### Fix segfault on exit when running NNP inference with CUDA

When running a simulation with the NNPot interface and performing inference on a CUDA device, a segfault would occur on program exit. This did not affect results, but should not happen.

Issue 5568

#### Running perturbed non-bonded interactions on a GPU could cause crashes

With atoms with small Lennard-Jones sigma parameter or no Lennard-Jones interactions, running perturbed non-bonded interactions on the GPU with soft-core would lead to extremely large forces. This would make simulations crash within a few steps.

Issue 5548

#### Incorrect results with perturbed non-bonded interactions on a GPU and many lambda states

When running perturbed non-bonded interactions on the GPU, the delta-H values, and thus free-energy differences, would be incorrect when more than 64 lambda states were used.

Issue 5551

#### The new AWH histogram tolerance parameter was not printed to the log file

Also added it to `gmx dump`.

Issue 5574

#### Ensemble averaging of orientation restraints gave a fatal error

Issue 5552

#### PME tuning never activated with a GPU and a single MPI rank

Issue 5578

### Fixes for `gmx` tools

#### `pdb2gmx` can now prints 7 decimals for charges and masses

`pdb2gmx` would print charges and masses with a maximum of 6 total decimals. Now this is 7, which matches the floating point precision for charges and masses.

Issue 5543

#### Fixed `gmx wham -ac` autocorrelation time calculation

Fixed `gmx wham -ac` incorrectly reporting very large integrated autocorrelation times (IACT) for long umbrella sampling trajectories when the pull coordinate was far from zero, due to precision loss in the calculation.

Issue 5547

### Fixes that affect portability

#### Fix device arch setup with newer version of amdclang

Newer versions of the compiler would fail to correctly configure the project due to not treating the test file as HIP input.

Issue 5558

### Miscellaneous

#### Do not write user names to H5MD files

`mdrun` now always writes the string "N/A" instead of the user name to the /h5md/author/name metadata field in H5MD files.

Issue 5557

#### Correctly detect warp uniform variable in CUDA kernel

NVCC was failing to detect warp uniform variables leading to performance degradation in the non-bonded force kernel. A workaround was implemented.

Issue 5546

#### QM/MM CP2K MDModule will check orthogonality of the cell

QM/MM simulations in CP2K require orthogonal periodic cells. The MDModule will now check for orthogonality of the cell and abort if the cell is non-orthogonal. In case of user-provided CP2K input files, a warning will be issued.

Issue 5529

#### New DFT functionals and D3 dispersion corrections for QM/MM with CP2K

QM/MM simulations with CP2K could now use hybrid DFT functionals B3LYP, PBE0 and range-separated hybrid functionals CAM-B3LYP, wB97X, wB97X-D3 in combination with all-electrons 6-31G* basis set. In addition, the option of using D3 dispersion corrections has been added to all functionals.

Issue 5529

#### Use correct stream priorities with AdaptiveCpp

In GROMACS 2026.0 built with AdaptiveCpp 25.02 or earlier, the GPU stream priorities were inverted. This could have led to decreased performance.

Issue 5575

### 11.2.2 Major release

#### Highlights

GROMACS 2026 was released on January 19th, 2026. Patch releases may have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

As always, we've got several useful performance improvements, with or without GPUs, all enabled and automated by default. In addition, several new features are available for running simulations. We are extremely interested in your feedback on how well the new release works on your simulations and hardware. The new features are:

- Two protein force fields from AMBER, ff14SB and ff19SB, have been ported to GROMACS as AMBER14SB and AMBER19SB. The ports also include the new OPC and OPC3 water models as well as several others.
- Expanded support for running simulations with Neural Network Potential models, now including link atom treatment for NNP/MM, pairlist input, and electrostatic embedding models.

- Experimental support for H5MD as a trajectory output format for `mdrun`.
- Full support for using HIP as the GPU backend for AMD devices.

### New and improved features

#### Collective variables module (Colvars) update

The (Colvars) library for enhanced sampling simulations included in GROMACS has been updated to version 2025-10-13.

This update brings many improvements, including:

- performance improvements for moving frame of reference calculations
- improved OPES implementation
- improved moving restraint logic, allowing for moving harmonic walls
- more flexible definitions of grids on CVs

A complete list of changes can be found here.

For documentation see *this section* (page 646) of the GROMACS doc along with the Colvars documentation page dedicated to GROMACS 2026. Messages about Colvars in the GROMACS discussion forum should be tagged with the colvars keyword for easier consultation.

#### Added LEaP-compatible dihedral reordering

AMBER LEaP reorders (improper) dihedrals when processing the topology, alphabetically by atom type. In order for GROMACS to be able to match the topology produced by LEaP, *gmx grompp* (page 230) was extended LEaP-compatible dihedrals reordering. This functionality is intended for usage with the ports of recent AMBER force fields (e.g. ff14SB and ff19SB), but it is enabled on any force field that defines the `_FF_AMBER_LEAP_ATOM_REORDERING` macro.

This feature has been validated on all supported dipeptides and tripeptides as well as select tetrapeptides and pentapeptides. Systematic validation on a broader set of scientifically relevant biomolecules is still pending.

Issue 4998

#### Added the port of AMBER's ff14SB and ff19SB protein force fields

The latest AMBER force field for proteins, ff19SB, has been ported to GROMACS. The port encompasses all functional forms, including per-amino-acid CMAPs, which are a new feature of ff19SB. Furthermore, the port also includes the recommended optimal 4-point and 3-point charge water models, better known as OPC and OPC3, and matching ions.

Older AMBER force field, ff14SB, has also been ported. For it, the TIP3P and TIP4PEW are considered to be the recommended water models.

Due to the limited usage for production simulations thus far and possible bugs in GROMACS features that these force fields use, these ports are considered validation pending.

Issue 4998

### Changed the behaviour of `pdb2gmx -rtpres`

In order to support force fields like AMBER19SB, *pdb2gmx* (page 281) can now automatically rename residues to have the name of the chosen residue type, so that *grompp* (page 230) can assign the CMAP terms correctly. This is handled by a new field in the `[ bondedtypes ]` description in the *residue types database* (page 582) that specifies that such force fields must rename the residues accordingly. `gmx pdb2gmx -rtpres` now defaults to `auto`, which means to preserve the residue name unless that residue-type-database setting requires renaming to match the residue type. It may also be set manually to `no` or `yes`.

### Improved implementation of the neural network potential interface

The neural network potential (NNP) interface has been extended to support new model inputs. Firstly, it is now possible to use a list of atom pairs within the NNP region filtered by a user-specified cutoff, as well as corresponding periodic shift vectors, as an input to the NNP. This enables the use of model architectures such as SchNet or MACE and re-uses the pair-list generated by GROMACS. Secondly, we also add support to pass positions and charges of the MM region around the NNP region to the model, enabling the use of electrostatic embedding schemes such as implemented in EMLE. Furthermore, a link atom scheme is now employed to treat covalent bonds between the NNP and MM regions. In addition, there have been minor performance improvements, and the interface now supports both CUDA and ROCm/HIP-builds of LibTorch. For more details on the supported features and usage, please see the *NNPot section in the reference manual* (page 647).

### Added support for H5MD trajectory file format

Trajectory data can now be written to the H5MD file format by *gmx mdrun* (page 258). Data written to this file format includes positions, velocities, forces, and the simulation box. Chemical bonds are also written to the connectivity group. For this release, only lossless output is supported.

Note that support for this file format in GROMACS tools is experimental. Analysis tools are able to read all files but bugs should be expected. H5MD output from *gmx trjconv* (page 334) and *gmx trjcat* (page 333) is not implemented.

### Fast Multipole Method (FMM) interface support

Support for simulations with external FMM libraries has been added. Short-range electrostatics can be computed by either GROMACS or the FMM library, while long-range electrostatics are handled by the FMM library when an FMM backend is selected. Although the FMM interface allows short-range electrostatics to be configured for either GROMACS or the FMM backend, short-range FMM interactions support in GROMACS is planned but not part of this release. MDP options are available for the well-known FMM libraries i.e., ExaFMM and FMSolvr, and these options are validated internally by the interface.

For more details, please see *the FMM section in the reference manual* (page 650).

### Add two new performance metrics

*gmx mdrun* (page 258) reports two additional time step-independent metrics to measure the performance of a simulation on specific hardware. The ms/step is the average wall-clock time (in milliseconds) required to compute a single timestep. As a direct measure of computational cost per timestep, it is useful for profiling code or hardware performance for a specific system. Matom*steps/s considers the number of atoms and measures the throughput of actions (number of steps * number of atoms) per second. This metric is suitable when comparing performance across systems and hardware. A similar metric can also be found from PLUMED.

Issue 5374

### QM/MM CP2K interface updates

QM/MM simulations with CP2K interface could now use GROMACS checkpoint files for saving QM subsystem centering translation. It greatly improves stability of QM/MM simulations during long trajectories and upon restarting from checkpoint.

Issue 5529

### Performance improvements

### Free-energy kernels on GPU with CUDA for perturbed non-bonded interactions

Free-energy calculations can now run faster on hardware with CUDA GPUs. The benefits are largest when the CPU is relatively weak and/or there are many perturbed non-bonded interactions. This feature has been contributed by "MetaX Integrated Circuits Technology". Note that this feature is currently marked as experimental.

### Optimized PME force transfers for multi-GPU runs with GPU-aware MPI

When using a GPU build with an (external) GPU-aware MPI library, the transfer of PME force data between GPUs has been optimized to use non-blocking MPI calls, improving performance by up to around 5%.

### More permissive AWH histogram equilibration check

The AWH histogram equilibration check, which is recommended to be used with multiple walkers, used a too strict tolerance of 20% maximum deviation. This tolerance can now be set by the user and the default value is 30%. This value provide significantly faster convergence when using multiple walkers. Since it does not deteriorate the convergence with a single walkers, the histogram equilibration check is now turned on by default.

### Bugs fixed

#### Collective variables module (Colvars) update

The (Colvars) library for enhanced sampling simulations included in GROMACS has been updated to version 2025-10-13.

This update brings many fixes, including some errors in gradient calculations, mostly affecting rarely used cases (see this page for details). A complete list of changes can be found here.

#### Tpr-file versioning fixed

Older installed GROMACS versions are intended to be able to do at least basic reading of .tpr files written by newer versions of GROMACS, unless made impossible by changes to the .tpr format. This happened in GROMACS 2025, but was not accompanied an appropriate increment of the version numbers, resulting in incorrect and misleading error messages. This cannot be fixed retroactively for installed GROMACS versions and extant .tpr files, but at least the problem conditions will no longer continue to arise for newly created .tpr files.

Issue 5334

#### Fixed missing non-bonded interactions with direct halo communication

When using the experimental direct halo communication feature combined with 8-wide SIMD non-bonded kernels and OpenMP threading, non-bonded interactions could be missing.

Issue 5509

#### Allow atoms involved intermolecular-exclusion to be perturbed

Issue 5527

#### Added check for constructing atoms of virtual sites

Constructing atoms for virtual sites can themselves be virtual sites, but only when those constructing atoms are virtual sites are higher up in the function type list (i.e. simpler constructions). This was documented in the manual. Now``grompp`` and `mdrun` will throw an error when these restrictions are violated.

Issue 5535

### Deprecated functionality

### Changes anticipated to GROMACS 2026 functionality

### Functionality deprecated in GROMACS 2026

### Generating angle constraints with grompp is deprecated

### Google Native Client (NaCl) build deprecated

Last release of the sandbox has been 10 years ago and the support for NaCl has been removed from LLVM 22. Building option `GMX_NACL` is now deprecated in GROMACS.

### Functionality deprecated in GROMACS 2025

### MTTK pressure coupling is deprecated

Issue 5072

### The TNG trajectory format is deprecated

The TNG file format will be removed in a future release. It will be replaced by the more widely used HDF5-based H5MD format. There will be at least one version of GROMACS supporting both H5MD and TNG to allow conversions between the two formats.

Issue 5225

### Functionality deprecated in GROMACS 2024

### The analysis tool `gmx gyrate-legacy` deprecated

`gmx gyrate` has been partly re-implemented in the modern GROMACS analysis framework. The old implementation is still available as `gmx gyrate-legacy`. Please plan to update to the new version. Please let the GROMACS developers know of desired functionality missing from, or broken in, the new implementation.

Issue 4927

### The analysis tool `gmx hbond-legacy` deprecated

`gmx hbond` has been partly re-implemented in the modern GROMACS analysis framework. The old implementation is still available as `gmx hbond-legacy`. Please plan to update to the new version. Please let the GROMACS developers know of desired functionality missing from, or broken in, the new implementation.

Issue 4927

### The analysis tools `gmx sans` and `gmx saxs` deprecated

`gmx sans` and `gmx saxs` has been partly re-implemented under new name `gmx scattering` in the modern GROMACS analysis framework. The old implementations are still available as `gmx sans-legacy` and `gmx saxs-legacy`. Please plan to update to the new version. Please let the GROMACS developers know of desired functionality missing from, or broken in, the new implementation.

Issue 4927

## Functionality deprecated in GROMACS 2022

### GMX_OPENCL_NB_CLUSTER_SIZE CMake variable deprecated in favor of GMX_GPU_NB_CLUSTER_SIZE

Both OpenCL and SYCL support different cluster sizes, so GMX_GPU_NB_CLUSTER_SIZE should be used going forward.

### Guessing masses and atomic radii from atom names is deprecated

When atom masses or van-der-Waals radii are needed, we suggest building a proper GROMACS topology instead of using PDB files directly, even if the tool supports it.

Issue 3368 Issue 4288

## Functionality deprecated in GROMACS 2021

### `mdrun -deffnm` to be removed

This functionality is convenient when running very simple simulations, because it permits grouping of a set of files that then differ only their suffix. However, it does not work in the wider case of an `mdrun` module (or modules) writing multiple `.xvg` output files. The resulting filenames collide. That, and its interaction with checkpointing and appending, have led to quite a few bug reports.

Because users can use a folder to group files (a standard mechanism that they understand from experience outside of GROMACS), we can build and test better software for them if we remove the erstwhile convenience of `mdrun -deffnm`. Please update your workflows accordingly.

Issue 3818

### OpenCL to be removed as a GPU framework

Since there are no prospects of an emerging GPU vendor in HPC needing OpenCL support, we intend to focus our efforts on CUDA, SYCL, and HIP for GPU acceleration in GROMACS. OpenCL support has been deprecated for a while, and will be removed in a future release.

Issue 3818

### Support for version 1 of the hardware locality library `hwloc`

Version 2 has been supported in GROMACS for several years. The capabilities of newer hardware and hardware-support APIs are of most interest for GROMACS moving forward, so we should minimize our testing work and encourage clusters to upgrade older `hwloc` installations.

Issue 3818

### Legacy API

The legacy installed headers have been deprecated for a while, however we wish to state more broadly that all headers found within the `src` directory tree of GROMACS are intended for internal consumption only, and are thus subject to change without notice. Further, the form and contents of the `libgromacs` library and related CMake targets may change as we move towards building APIs and supporting machinery that can be stable and supported in the long term.

Issue 3818

### Functionality deprecated in GROMACS 2019

### Generation of virtual sites to replace aromatic rings in standard residues

These are thought to produce artefacts under some circumstances (unpublished results), were never well tested, are not widely used, and we need to simplify pdb2gmx.

Issue 3254

### Benchmarking options only available with `gmx benchmark`

Options such as `-confout`, `-resethway`, `-resetstep` are not intended for use by regular mdrun users, so making them only available with a dedicated tool is more clear. Also, this permits us to customize defaults for e.g. writing files at the end of a simulation part in ways that suit the respective mdrun and benchmark use cases, so `-confout` will no longer be required.

Issue 3255

#### `gmx mdrun -nsteps`

The number of simulation steps described by the .tpr file can be changed with `gmx convert-tpr`, or altered in .mdp file before the call to `gmx grompp`. The convenience of this mdrun option was outweighed by the doubtful quality of its implementation, no clear record in the log file, and lack of maintenance.

Issue 3256

### Removed functionality

#### Support for Xeon Phi processors is removed

Remaining support for Knights Landing (KNL) hardware and SIMD instructions have been removed.

Issue 4740

### Portability

#### MPI 3.0 is now required, 2.0 no longer supported

When building with MPI library, GROMACS 2026 requires it to be conformant with MPI 3.0. This should not be a problem as long as you are using MPI library from the past decade (e.g., OpenMPI 1.8+, MPICH 3.0+).

Issue 4933

#### Full support for HIP as GPU backend

The HIP backend can now be used for offloading all GPU kernels to AMD devices. Ported kernels include direct GPU-to-GPU communication and GPU-only run path.

Issue 4947

#### MKL can now be found more easily from non-oneAPI distributions

The CMake support for finding and using MKL is now more flexible, including now being able to find and link MKL when it was installed from Debian's repositories.

Issue 5508

### Miscellaneous

#### Status of MDModules that require an external library is now logged

Support status of CP2K QM/MM, Colvars, Plumed & Neural Network Potential modules along with the version of libraries they use are now reported in `md.log` and `gmx --version`.

#### Modular simulator reports why it cannot be used

When a Velocity-verlet integrator simulation cannot use the modular simulator, the reasons are logged. If the modular simulator was required by the user, a fatal error is given along with the reasons why it cannot be run.

Issue 5339

#### Invalid `-nstlist` value is now an error instead of a warning

An invalid (e.g., too large) `-nstlist` value now triggers a fatal error instead of falling back to the fixed value from the TPR.

Issue 5365

#### FFTW3 now builds with `CMAKE_C_COMPILER`

Previously, the cmake flag `-DGMX_BUILD_OWN_FFTW=ON -DGMX_FFT_LIBRARY=fftw3` leads to FFTW3 being compiled with the system default compiler. The `CMAKE_C_COMPILER` is now passed to the FFTW3 configure command.

Issue 5344

#### Removed duplicate dihedral type definitions in the CHARMM27 force field

The CHARMM27 force fields files contained duplicates of definitions of dihedral types for atom types CPH1 and CPH2. This did not affect correctness, but was confusing and prevented strict error checking by `gmx grompp`.

Issue 5333

#### Refactored CMake CUDA configuration to use standard CMake mechanisms

The custom CUDA configuration logic has been replaced with CMake's built-in FindCUDAToolkit and native CUDA language support. This makes GROMACS configuration more consistent with other CUDA-enabled CMake projects.

The following CMake variables have been replaced with standard CMake equivalents:

- `GMX_CUDA_TARGET_SM` and `GMX_CUDA_TARGET_COMPUTE` are replaced by `CMAKE_CUDA_ARCHITECTURES`
- `CUDA_TOOLKIT_ROOT_DIR` is replaced by `CUDAToolkit_ROOT`

- `CUDA_HOST_COMPILER` is replaced by `CMAKE_CUDA_HOST_COMPILER`

## 11.3 Older (unmaintained) GROMACS series

## 11.4 GROMACS 2025 series

### 11.4.1 Patch releases

#### GROMACS 2025.5 release notes

This version was released on August 31st, 2026. These release notes document the changes that have taken place in GROMACS since the previous 2025.4 version, to fix known issues. It also incorporates all fixes made in version 2024.6 and earlier, which you can find described in the *Release notes* (page 838).

##### Fixes where mdrun could behave incorrectly

##### Fixed missing non-bonded interactions with direct halo communication

When using the experimental direct halo communication feature combined with 8-wide SIMD non-bonded kernels and OpenMP threading, non-bonded interactions could be missing.

Issue 5509

##### Prevent incorrect energy accumulation with `GMX_ENABLE_NBNXM_CPU_VECTORIZATION`

When building with the experimental `-DGMX_ENABLE_NBNXM_CPU_VECTORIZATION=on` flag for RISC-V vector unit and using multiple energy groups, the reported short-range non-bonded energy could have been incorrect.

Issue 5534

##### Fixed pressure drift with virtual sites and OpenMP threading

When virtual sites were used with more than one OpenMP thread per rank, an accumulation buffer for the virial was not always cleared. This could cause the pressure to drift while other thermodynamic quantities looked normal. This issue has only been observed while other thermodynamic quantities looked normal. This issue only occured with more than 200000 atoms per domain, i.e. MPI rank.

Issue 5661

### Fixes for `gmx` tools

### Fixes that affect portability

### Fixed compilation of tests on FreeBSD

If incompatible versions of GoogleTest and GoogleMock can be found in system include directories, the build process could in some cases use their headers and later fail during linking. This release includes a workaround that ensures that GROMACS-supplied versions of those libraries are always picked up before the system versions.

Issue 5141

### Miscellaneous

### GROMACS 2025.4 release notes

This version was released on November 21st, 2025. These release notes document the changes that have taken place in GROMACS since the previous 2025.3 version, to fix known issues. It also incorporates all fixes made in version 2024.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### The Parrinello-Rahman barostat can give too large volume fluctutations

The increase of the default *nstpcouple* (page 67) value in the GROMACS 2023 release can cause too large fluctuations of the volume of the simulation box. This, in turn, can lead to instabilities, in particular when anisotropic pressure coupling is used. The average volume is not affected. This issue is solved by requiring at least 200 integration steps per period (tau_p) for the Parrinello-Rahman barostat.

Issue 5424

### The C-rescale barostat can give too large volume fluctutations

The increase of the default *nstpcouple* (page 67) value in the GROMACS 2023 release can cause too large fluctuations of the volume of the simulation box. The average volume is not affected. The is only an issue when tau_p is smaller than the recommended value of 5 ps, in combination with a timestep of 2 fs or larger. This is solved by requiring at least 25 integration steps per period (tau_p) for the C-rescale barostat.

Issue 5425

### PME tuning could exit with an assertion failure

When using OpenMP threading and an unlucky combination of PME grid dimensions, mdrun could exit during PME tuning with an assertion failure for gridSize.

Issue 5406 Issue 5417

### mdrun would report incorrect kinetic energy values when interrupted

When mdrun was interrupted and did not stop at a step that was a multiple of *nstcalcenergy* (page 56), the kinetic energy, temperature, pressure and energy drift reported for the last step were incorrect when using the leap-frog integrator. Note that such an incorrect frame was removed when the simulation was continued with concatenation of output files.

Issue 5433

### Direct halo communication could cause mdrun to exit with an MPI error

The direct halo communication in mdrun, which needs to be enabled by setting an environment variable, could cause mdrun to stop with an MPI error. This is very unlikely to have led to incorrect results.

Issue 5418

### Expanded ensemble replica exchange output could contain concatenated numbers

With 100 or more replicas, spaces would be missing between values and indices. Also other large values could lead to no whitespace being printed.

Issue 5248

### Fixes for `gmx` tools

### Fix DSSP tool

Fixed legend formatting in `gmx dssp` tool. Data for plotting a graph from .xvg format is now compatible with XMGrace.

Issue 5409

### Fixes that affect portability

### Fix CUDA build on Windows

Building GROMACS with CUDA on Windows would sometimes fail at CMake stage with `Could NOT find OpenMP_CUDA` error. Added a workaround for that.

Issue 5431

### Enable `-Wpointer-arith` flag for LLVM-based compilers

Building GROMACS with LLVM-based compilers would get the `-Wpointer-arith` flag tested, but the result of the test wasn't used. This is now fixed.

Issue 5447

### Fix SurfaceAreaTest failure when ARM SVE SIMD is used

Some versions of GCC miscompile test code when targeting ARM machines with SVE support, leading to false-positive failures in `SurfaceAreaTest.Computes100PointsWithTriclinicPBC`. This has been observed with GCC 13 and 14 on systems with 256-bit and 512-bit SVE vectors. As a workaround, we disable compiler optimizations for the affected test function. This only affects test code and does not impact simulation performance.

Issue 5440

### Fixes to the Colvars library

This release fixes several bugs in the Colvars library, affecting the correctness of applied forces for certain variables.

The following is a list of bugfix pull requests in the`Colvars repository <https://github.com/Colvars/colvars>`_:

- Tests and fixes to multiple gradients (Colvars PR 838)
- Fixing the orientation sign (Colvars PR)
- Fix the gradients of the eigenvector CVC (Colvars PR 828)
- Fix a potential issue of "fit gradients" for non-scalar components with only centering but not rotating (Colvars PR 824)
- Fix the calculation of bias factor with off-grid hills (Colvars PR 808)
- Fix indentation of 'Init CVC' message in standard output (Colvars PR 755)

## Miscellaneous

### Disable floating point exceptions in CUDA builds

With CUDA 13 / NVIDIA driver 580, a floating-point exception can occur in the PTX JIT compiler. This is harmless in itself but would cause program termination if the floating-point exceptions are enabled. So, we don't enable them for CUDA builds anymore.

Issue 5493

### GROMACS 2025.3 release notes

This version was released on August 29th, 2025. These release notes document the changes that have taken place in GROMACS since the previous 2025.2 version, to fix known issues. It also incorporates all fixes made in version 2024.6 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### AWH for free-energy lambda coordinate was not active

When using AWH for a free-energy lambda coordinate (and no other dimension), the lambda state would remain at the initial state and all resulting free energies would be zero. Affected versions: 2024.4, 2024.5, 2025.0, 2025.1 and 2025.2.

Issue 5371

#### AWH bias sharing caused error without separate PME ranks

Using AWH in conjuction with bias sharing between several simulations caused mdrun to segfault if more than one rank per simulation was used but no separate PME ranks were present. This could not produce incorrect results. Affected versions: 2024.5, 2025.0, 2025.1 and 2025.2.

Issue 5394

#### `gmx mdrun` now throws with input inconsistent with the modular simulator

By default, `gmx mdrun` can choose the modular simulator if possible and fall back on the legacy simulator if needed. The behaviour can be specified with the use of documented environment variables. After this fix, the use of `GMX_USE_MODULAR_SIMULATOR` environment variable is properly honored. If the simulation cannot use the modular simulator, an error is given.

Issue 5339

#### Fixes for `gmx` tools

#### Fixes that affect portability

#### SYCL build now works with BUILD_SHARED_LIBS=off

Previously `gmx` and some test binaries failed to link to `libgromacs.a` because the required information to link to the SYCL runtime library is missing in that case. Linking of executables in SYCL builds is now more robust.

Issue 5341

### Fix build with ROCm 7 and newer

Building and linking GROMACS would fail due to changes in the linking behaviour of the main HIP library.

Issue 5380

Building GROMACS would fail due to `__AMDGCN_WAVEFRONT_SIZE` macro removal in ROCm 7.

Issue 5259

### Improved configuration error message with SYCL and VkFFT

Configuring GROMACS with SYCL and VkFFT, targeting Intel devices, would result in an error meessage implying using VkFFT with SYCL wasn't suported at all. The message is now clarified to include the statement that this lack of support only applies to Intel devices.

Issue 5401

### Fix detection of unusable MKL GPU offloading library during configuration

Configuration would previously happily proceed even if development headers of MKL GPU offloading library were not found, causing the compilation to fail. This now results in an error.

Issue 5400

### Fix build for NVIDIA sm_101 / sm_110 devices

Issue 5366

### Miscellaneous

### Documented that PME tuning in multi-replica simulations can give suprising results

PME tuning chooses between nominally equivalent model physics, yet there is no guarantee of consistency between (say) the tunings chosen for different replicas in a multi-replica simulation, so the fine details of comparisons of potential energy can be surprising. Disabling PME tuning for e.g. testing Hamiltonian replica-exchange simulations with PLUMED is advised.

Issue 5278

### mdrun reported double the number of domain decompositions when using GPUs

When using GPUs, the cycle count table at the end of the log file reported a number of calls for Domain decomp. that was too high by a factor of one plus the number of dimensions that were decomposed. The times and cycle counts were correct.

Issue 5369

### Support added for compilation with NVIDIA CUDA 13 toolkit

Addition of a compilation option necessary to avoid CUDA build failures when using the NVCC compiler from version 13 of the CUDA toolkit.

Issue 5389

### Avoid installing 3rd party header files

Fixed a regression where GROMACS installed muParser headers, causing conflicts with the separate muParser package if installed.

Issue 5282

### libgromacs now only links to MPI libraries when explicitly required

Formerly `gmxapi` could have erroneously triggered GROMACS to compile and link with an MPI library. This is now fixed.

Issue 5395

### Declared gmxapi option before finding MPI package

Formerly `gmxapi` was set up after MPI libraries, which caused a subsequent run of CMake to decide to find MPI libraries to help gmxapi even when `libgromacs` didn't need those libraries. Now the CMake cache does not vary between successive runs in this way.

Issue 5395

### Improve heuristics for choice of NBNXM kernel flavor for AMD gfx94x

Analytical Ewald kernels are used since these perform better in most common use-cases. The exception is *vdw-modifier=Force-switch* (page 61) with SYCL backend, where we still use tabulated kernels.

Issue 5329

### GROMACS 2025.2 release notes

This version was released on May 12th, 2025. These release notes document the changes that have taken place in GROMACS since the previous 2025.1 version, to fix known issues. It also incorporates all fixes made in version 2024.5 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Fix assertion failure with CUDA and HIP builds with empty domains

mdrun could exit with an assertion failure when a domain contained zero atoms.

Issue 5322

#### Fix mdrun crashes with position restraints + flat-bottomed position restraints + p-coupling with COM refcoord-scaling + OpenMP

This combination can lead to (too) high position restraint forces when OpenMP threads run slightly out of sync. Simulations will have very likely crashed due to too high forces, unless they are very short.

Issue 5337

#### Fixes for `gmx` tools

#### Fix parsing of pdb files with multiple cyclic and acyclic chains

pdb2gmx computed incorrect coordinate distances when checking for cyclicity of the second and later chains in the pdb input file.

Issue 5091

#### grompp now warns about identical dihedraltype 9 definitions

grompp would silently ignore double, identical dihedraltype definitions with function type 9. Now such entries are not ignored and a warning is generated.

Issue 5333

#### Fix out-of-boundary read in gmx sorient

The outermost (maximum radius) bin in the histograms created for the -o and -no output files from this tool added a value which was read from unallocated memory. Only values in this bin were affected.

### Fixes that affect portability

#### Fixed cross-test for Windows on Linux with Wine

Cross-testing of GROMACS 2025.1 for Windows on Linux with Wine, by setting the `CMAKE_CROSSCOMPILING_EMULATOR` variable to `wine` or `wine64`, could fail due to our tests using CMake generator expressions and ignoring the variable setting. The tests are now executed using the emulator set in the variable.

Issue 5342

### Miscellaneous

#### Intel Xe2/Battlemage documentation

Expanded the documentation and rephrased a misleading error message.

Issue 5330

### GROMACS 2025.1 release notes

This version was released on March 11th. These release notes document the changes that have taken place in GROMACS since the previous 2025.0 version, to fix known issues. It also incorporates all fixes made in version 2024.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Prevent "Need at least one thread" error from domain decomposition

Running domain decomposition with 9 or more ranks could cause mdrun to exit with an assertion failure.

Issue 5289

#### Fix force correction for affine transformations in density fitting module

The force calculation was not correctly updated when doing the transformation. Only simulations with non-identity `density-guided-simulation-transformation-matrix` were affected.

Issue 5298

### Fix PME-PP direct GPU communication with threadMPI

We observed crashes in runs using threadMPI and GPU-direct MPI communication with GROMACS 2024, and because of this direct GPU communication for PME-PP coordinate and force transfers were disabled in GROMACS 2025.0 (where GPU-direct communication became the default).

The original issue was caused by missing synchronization with multiple PP ranks and a PME rank, only affected systems with 23 thousand particles or less. In such cases, after a few hundred steps simulation crashed with thread-MPI errors, so no wrong physics ensued. For such small systems PP decomposition is not beneficial, so the crashes were unlikely to be triggered, hence these went unnoticed until the direct GPU communication became the default. This issue has now been fixed and direct GPU communication is enabled by default, making the performance on par or slightly better than GROMACS 2024 with `GMX_ENABLE_DIRECT_GPU_COMM` and much better than GROMACS 2024 running with the default settings.

Issue 5283 Issue 5306

### Fix NVSHMEM enabled mdrun hang with a varying number of communication pulses

A hang is encountered when the domain decomposition involves increasing and decreasing the number of communication pulses involving 1 PME and more than one PP ranks.

Issue 5303

### Fix host buffer pinning with shell/flexcon and PME on CPU

In edge cases it would be possible to crash GROMACS when running code with flexible constraints when PME was performed on the CPU (or not used) and GPU buffer operations are used.

Issue 5317

### Fix segmentation fault in NNPot with domain decomposition

When using the NNPot module with domain decomposition, a segmentation fault could occur.

### Fixes for `gmx` tools

### Fix parsing of CMAP types with arbitrary number of spaces between elements

The CMAP type parser assumed there is only a single space between consecutive elements, while some tools prefer to add extra whitespace when formatting. Now we happily accept arbitrary number of spaces or tabs.

Issue 5312

### Fixes that affect portability

#### Improve handling of muParser

When building with `DGMX_USE_MUPARSER` set to `EXTERNAL` or `NONE`, GROMACS 2025.0 could have trouble compiling. This is now fixed.

Issue 5290

#### Fix build issues with MSVC and CUDA

Building GROMACS on Windows using MSVC + CUDA (OpenMP is enabled by default) now works.

Issue 5294

#### Fix build with SYCL and HeFFTe

GROMACS 2025.0 was failing to build with AdaptiveCpp and HeFFTe due to incomplete refactoring. This is now fixed.

Issue 5314

#### Fixed cross-compile for Windows with MinGW GCC on Linux

Cross-compiling of GROMACS 2025.0 for Windows with MinGW GCC on Linux could fail due to a missing include and Linux-incompatible case sensitivity of Windows-specific includes. This is now fixed.

Issue 5088

### Miscellaneous

#### Silence compiler warning when building with ARM SVE

Silence a few harmless warnings when building with a recent compiler for ARM SVE.

#### Fix CMake issues when building with Plumed

To activate Plumed during compilation CMake had to be run twice. This has now been fixed and CMake only requires a single invocation for the build.

Issue 5292

#### No longer suggest SVE over NEON for CPU builds on Neoverse-v2 at run time

On Neoverse-v2, the most performant SIMD instruction depends on exact run time configuration so the user is now directed to the install guide when running on this architecture.

Issue 5296

### 11.4.2 Major release

#### Highlights

GROMACS 2025.0 was released on February 11th, 2025. Patch releases may have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

As always, we've got several useful performance improvements, with or without GPUs, all enabled and automated by default. In addition, several new features are available for running simulations. We are extremely interested in your feedback on how well the new release works on your simulations and hardware. The new features are:

- A feature-limited version of PLUMED can be used, on a non-Windows build of GROMACS, without needing to apply a patch.
- Basic support for running simulations with Neural Network Potential (NNP) models trained in PyTorch.
- Extended OpenMP parallelization of pair search and domain decomposition can improve the performance, especially relevant with fast GPUs.
- Added support for using AMD HIP as GPU backend. This is currently limited to the NBNxM (nonbonded interactions within cut-off) kernels.
- Support for continuing expanded ensemble equilibration across simulations by enabling initialization of *`init-lambda-counts`* (page 91) and *`init-wl-histogram-counts`* (page 91) through mdp options.
- GPU-direct communication is now used by default when the MPI library supports it.
- Enhanced PP halo exchange using GPU kernel-initiated communication implemented using NVSHMEM, improving performance when scaling to multiple NVIDIA GPUs.

#### New and improved features

#### A feature-limited version of the PLUMED interface is available

A basic version of the PLUMED interface is now bundled by default in GROMACS. With a non-Windows installation, it is possible to invoke PLUMED directly from the command line using the `-plumed` option of the `gmx mdrun` command, followed by the path to a PLUMED input file. This can be done **without the need to apply a patch** as in previous GROMACS versions. Importantly, this interface is not feature complete, see *the section in the manual* (page 647) for the details.

### Support for amino-acid-specific energy correction maps (CMAPs)

Previously, energy correction map (CMAP) types could only be specified using atom types and were applied to atoms in an amino-acid-agnostic way. CMAP types can now be specified using both atom types and residue types, which enables support for recent Amber force fields (ff19SB and later versions).

Issue 4430

### Neural Network Potential support

Basic support has been added to perform simulations with Neural Network Potentials (NNPs). These models can be trained to reproduce forces and energies at *ab initio* levels of accuracy, based on training data from electronic structure calculations with e.g. DFT or CCSD(T). Importantly, GROMACS does *not* include any pretrained models, so users need to train their own models or load pre-trained models from external sources. As of now, the interface supports NNP models trained in PyTorch. For details on usage and building GROMACS with LibTorch support, please see the *NNPot section in the reference manual* (page 647).

### Add Custom Improper Dihedrals in specbond.dat

This change allows users to specify an improper dihedral resulting for a special bond (i.e. a thioester connection resulting in a SP2 group) in the specbond.dat file which is read during the pdb2gmx preprocessing step.

Issue 5113

### Performance improvements

### Instant-submission mode enabled by default when building with AdaptiveCpp

In GROMACS 2024, one had to manually enable the instant-submission mode of AdaptiveCpp when building GROMACS (`-DSYCL_CXX_FLAGS_EXTRA=-DHIPSYCL_ALLOW_INSTANT_SUBMISSION=1`). Now it is enabled by default, improving performance by up to 20% when running on GPU and slightly reducing the CPU usage when using SYCL/AdaptiveCpp backend.

### More OpenMP parallelization

OpenMP multithreading is now used for position restraints, GPU LINCS initialization, domain decomposition state vector sorting and a few other places. This is particularly relevant for runs where a fast GPU is bottlenecked by DD and other CPU tasks.

Issue 5100

### GPU buffer operations enabled by default

This is generally faster on modern GPUs, but, for very small systems, this can worsen performance by a few percent. In this case, the old behavior can be restored by setting `GMX_GPU_DISABLE_BUFFER_OPS=1` environment variable.

### NVSHMEM enabled PP Halo Exchange of Forces and Coordinates

Introduced pack/unpack kernels fused with data transfers enabled via NVSHMEM for particle-particle (PP) halo exchange, optimizing performance for all simulation input types. This feature enhances scalability by leveraging NVSHMEM's low-overhead, asynchronous communication capabilities and is supported on NVIDIA Volta and newer architectures. Using this functionality requires NVSHMEM to be enabled at build-time (`-DGMX_NVSHMEM=ON`) as well as explicitly requested at runtime with the `GMX_ENABLE_NVSHMEM=1` environment variable. It should be noted that this is an early implementation, and performance improvements for this feature are planned for a future release.

Issue 5103

### Improvements to GROMACS tools

#### `gmx grompp` now checks dihedral coefficients sum

The sum of dihedral parameters of type 3 (i.e Ryckaert-Bellemans or Fourier dihedrals) is now checked during preprocessing. In free energy simulations, this sum must be equal in both states as it affects final results through dH/dl. Additionally, this sum should be zero when comparing potential energy values with other force field ports and simulation codes, but a non-zero sum does not otherwise affect the simulation (a simple note is emitted).

If `gmx grompp` rejects an interaction that was previously accepted, then change the first coefficient to make the total zero. This leading coefficient has no effect on the derivative of the energy (ie. the forces), and thus no effect on the dynamics.

No parameters in default force fields in GROMACS were affected, so none have changed.

Issue 4253

#### Added support for DSSP v.2 in `gmx dssp`

There is now the ability to choose between two different output modes: with polyproline helices search enabled (option "-polypro", default and corresponds to the output of DSSP v.4) and with polyproline helices search disabled (option "-nopolypro", corresponds to the output of DSSP v.2).

### The number of averaging blocks for `-evisco` is now chosen by the user

The number of averaging blocks used when computing viscosity using Einstein formula was previously hard-coded to 4. This number can now be chosen by the user via the `-einstein_blocks` flag of `gmx energy`. When computing viscosity from very long trajectories, it may be preferable to have several averaging blocks to obtain a more accurate average and avoid the long-time diffusive behavior of the pressure integral.

Issue 5114

### New features in `gmx hbond`

A set of additional functionality has been added: dynamic selections are now available (as well as detailed output for them in an atomic index file); added the ability to customize the geometric criterion for hydrogen bond formation using custom distances and angles (via "-hbr" and "-hba" flags respectively); added the ability to add any atomic elements as donors or acceptors of hydrogen bonds (via "-de" and "-ea" flags respectively).

### Improve reading performance of large .gro files

`gmx grompp` is now up to 30% faster when reading system coordinates from a large *gro* (page 577) file.

### Smaller sampling windows in gmx msd

Trajectories that save frames at higher than average frequencies can now be analyzed at more frequent sampling windows using `-trestart` < 1.0 picosecond, down to 1 femtosecond.

Issue 4694

### Bugs fixed

### Fix double counting of dispersion correction with test particle insertion

The dispersion correction of the test particle with the system was double counted.

Issue 5231

### `gmx grompp` now allows test particle insertion of an identical molecule

`gmx grompp` merges molecule blocks in the topology with the same molecule type. With TPI this prevented insertion of a molecule into a liquid of the same molecules.

#### Fix rare issue with missing perturbed exclusions error

In rare cases for system with perturbed pair interactions that are zero, e.g. a state with decoupled charges such that some hydrogens are non-interacting, `gmx mdrun` could exit with an error message stating that excluded, perturbed interactions are beyond the cut-off, whereas this was actually not the case.

Issue 5267

### Deprecated functionality

### Changes anticipated to GROMACS 2025 functionality

### Functionality deprecated in GROMACS 2025

#### MTTK pressure coupling is deprecated

Issue 5072

#### The TNG trajectory format is deprecated

The TNG file format will be removed in a future release. It will be replaced by the more widely used HDF5-based H5MD format. There will be at least one version of GROMACS supporting both H5MD and TNG to allow conversions between the two formats.

Issue 5225

### Functionality deprecated in GROMACS 2024

#### The analysis tool `gmx gyrate-legacy` deprecated

`gmx gyrate` has been partly re-implemented in the modern GROMACS analysis framework. The old implementation is still available as `gmx gyrate-legacy`. Please plan to update to the new version. Please let the GROMACS developers know of desired functionality missing from, or broken in, the new implementation.

Issue 4927

#### The analysis tool `gmx hbond-legacy` deprecated

`gmx hbond` has been partly re-implemented in the modern GROMACS analysis framework. The old implementation is still available as `gmx hbond-legacy`. Please plan to update to the new version. Please let the GROMACS developers know of desired functionality missing from, or broken in, the new implementation.

Issue 4927

### The analysis tools `gmx sans` and `gmx saxs` deprecated

`gmx sans` and `gmx saxs` has been partly re-implemented under new name `gmx scattering` in the modern GROMACS analysis framework. The old implementations are still available as `gmx sans-legacy` and `gmx saxs-legacy`. Please plan to update to the new version. Please let the GROMACS developers know of desired functionality missing from, or broken in, the new implementation.

Issue 4927

### The Xeon Phi support will be removed

Intel Xeon Phi series of accelerators has been discontinued in 2020, and most supercomputers using it are now retired. The support for Xeon Phi is deprecated in GROMACS 2024.

Issue 4740

### Functionality deprecated in GROMACS 2022

### GMX_OPENCL_NB_CLUSTER_SIZE CMake variable deprecated in favor of GMX_GPU_NB_CLUSTER_SIZE

Both OpenCL and SYCL support different cluster sizes, so GMX_GPU_NB_CLUSTER_SIZE should be used going forward.

### Guessing masses and atomic radii from atom names is deprecated

When atom masses or van-der-Waals radii are needed, we suggest building a proper GROMACS topology instead of using PDB files directly, even if the tool supports it.

Issue 3368 Issue 4288

### Functionality deprecated in GROMACS 2021

### `mdrun -deffnm` to be removed

This functionality is convenient when running very simple simulations, because it permits grouping of a set of files that then differ only their suffix. However, it does not work in the wider case of an `mdrun` module (or modules) writing multiple `.xvg` output files. The resulting filenames collide. That, and its interaction with checkpointing and appending, have led to quite a few bug reports.

Because users can use a folder to group files (a standard mechanism that they understand from experience outside of GROMACS), we can build and test better software for them if we remove the erstwhile convenience of `mdrun -deffnm`. Please update your workflows accordingly.

Issue 3818

### OpenCL to be removed as a GPU framework

Issue 3818 Work is underway for ports to AMD and Intel GPUs, and it is likely that those ports will not be based on the current GROMACS OpenCL port. Nvidia GPUs are targeted by the CUDA port, and no changes are expected there. The core team can't maintain, test, and extend up to 4 ports with current resource levels. Since there are no prospects of an emerging GPU vendor in HPC needing OpenCL support, we will remove the OpenCL port once AMD and Intel support is established in other ways.

### Support for version 1 of the hardware locality library `hwloc`

Issue 3818 Version 2 has been supported in GROMACS for several years. The capabilities of newer hardware and hardware-support APIs are of most interest for GROMACS moving forward, so we should minimize our testing work and encourage clusters to upgrade older `hwloc` installations.

### Legacy API

Issue 3818 The legacy installed headers have been deprecated for a while, however we wish to state more broadly that all headers found within the `src` directory tree of GROMACS are intended for internal consumption only, and are thus subject to change without notice. Further, the form and contents of the `libgromacs` library and related CMake targets may change as we move towards building APIs and supporting machinery that can be stable and supported in the long term.

### Functionality deprecated in GROMACS 2019

### Generation of virtual sites to replace aromatic rings in standard residues

Issue 3254 These are thought to produce artefacts under some circumstances (unpublished results), were never well tested, are not widely used, and we need to simplify pdb2gmx.

### Benchmarking options only available with `gmx benchmark`

Issue 3255 Options such as `-confout`, `-resethway`, `-resetstep` are not intended for use by regular mdrun users, so making them only available with a dedicated tool is more clear. Also, this permits us to customize defaults for e.g. writing files at the end of a simulation part in ways that suit the respective mdrun and benchmark use cases, so `-confout` will no longer be required.

### `gmx mdrun -nsteps`

Issue 3256 The number of simulation steps described by the .tpr file can be changed with `gmx convert-tpr`, or altered in .mdp file before the call to `gmx grompp`. The convenience of this mdrun option was outweighed by the doubtful quality of its implementation, no clear record in the log file, and lack of maintenance.

### Removed functionality

### Removed support for planetary simulations

`GMX_DO_GALACTIC_DYNAMICS` setting is removed. Setting a negative *epsilon-r* (page 60) value now always raises an error.

Issue 5223

### Portability

### Added support to compile GROMACS using AMD HIP as GPU backend

It is now possible to use AMD HIP directly as the GPU backend to run simulations on AMD devices. For now only the NBNxM kernels are can be offloaded to the device using this backend.

Issue 4947

### Added support for the oneMath interface library for GPU FFTs

This enables cross-vendor support for GPU FFTs to the GROMACS SYCL backend. Either cuFFT or rocFFT can now be used with Intel DPC++ and Codeplay's plugins for NVIDIA and AMD GPUs.

Issue 4744

### Update of required build tool versions

Updated minimal required versions:

- CMake 3.28,
- Compilers: GCC 11, Clang 14 (using Clang 18 for code formatting),
- GPU toolkits: CUDA 12.1, oneAPI 2024.0, AdaptiveCpp 23.10, ROCm 5.2.

Issue 5014

### Miscellaneous

### Internal build of FFTW now uses version 3.3.10

When the `-DGMX_BUILD_OWN_FFTW=ON` option is enabled, we now compile FFTW 3.3.10 instead of the previous 3.3.8 version. For more details, please refer to the FFTW release notes.

#### Increased AWH parameter 'awh-nsamples-update' default value from 10 to 100

This is to decrease the overhead of updating the bias, in particular with multiple walkers.

#### Support for continuing expanded ensemble equilibration across simulation parts

The mdp options *init-lambda-counts* (page 91) and *init-wl-histogram-counts* (page 91) can now initialize the number of counts at each sampled lambda state and the Wang-Landau histograms used to determine simulation equilibration. These are most useful when running short simulation parts, so that the information about how the system is equilibrating can be carried over between simulations. Otherwise, chains of short simulation parts would never converge when using expanded-ensemble methods. The information is now also carried over through the checkpoint file.

#### CTest can now run tests in parallel

Properly express the resource requirements of test binaries and their dependencies so that `ctest --parallel` can be used safely.

## 11.5 GROMACS 2024 series

### 11.5.1 Patch releases

#### GROMACS 2024.7 release notes

This version was released on TODO, 2025. These release notes document the changes that have taken place in GROMACS since the previous 2024.6 version, to fix known issues. It also incorporates all fixes made in version 2023.5 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Fixes for `gmx` tools

#### Fixes that affect portability

#### Miscellaneous

#### GROMACS 2024.6 release notes

This version was released on August 22, 2025. These release notes document the changes that have taken place in GROMACS since the previous 2024.5 version, to fix known issues. It also incorporates all fixes made in version 2023.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### AWH for free-energy lambda coordinate was not active

When using AWH for a free-energy lambda coordinate (and no other dimension), the lambda state would remain at the initial state and all resulting free energies would be zero. Affected versions: 2024.4, 2024.5.

Issue 5371

#### Fix memory error for NVIDIA Pascal and earlier GPUs

When running on Pascal (e.g., GeForce GTX 10xx, Tesla P100) and earlier NVIDIA GPUs, a memory error was present in the neighbor list pruning kernel. It has not been observed to cause any incorrect results, but could, theoretically, cause undefined behavior. All earlier GROMACS versions in the 2024 series are affected.

Issue 5385

### Fixes for `gmx` tools

### Fixes that affect portability

#### Fix CUDA PTX generation for new architectures

The functionality to JIT-compile GPU kernels for new architectures was broken in recent releases. Now it should work again.

Issue 5285

#### Avoid launch bounds PTX error for new NVIDIA RTX Blackwell GPUs

Fix compatibility issue preventing using GROMACS 2024 on new NVIDIA GPUs.

#### AWH bias sharing caused error without separate PME ranks

Using AWH in conjuction with bias sharing between several simulations caused mdrun to segfault if more than one rank per simulation was used but no separate PME ranks were present. This could not produce incorrect results. Affected versions: 2024.5, 2025.0, 2025.1 and 2025.2.

Issue 5394

### Miscellaneous

### GROMACS 2024.5 release notes

This version was released on January 24th, 2025. These release notes document the changes that have taken place in GROMACS since the previous 2024.4 version, to fix known issues. It also incorporates all fixes made in version 2023.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Fix crashes with AWH and multi-sim bias sharing

`gmx mdrun` multi-sim AWH with bias sharing might have crashed with either 1 PP rank and 1 PME rank, or multiple PP ranks.

### Fix issue with fully frozen atoms in T-coupling groups with non-frozen atoms

`gmx grompp` would, incorrectly, subtract 3 degrees of freedom for each fully frozen atom that was part of a temperature coupling group that also contained non-frozen atoms. This led to too high temperatures being reported and too low real temperatures when temperature coupling is used. Note that you can not detect the latter issue by looking at the temperature in the output, as this would match the temperature of the bath.

Issue 5261

### Fix crash with modular simulator, md integrator, DD and constraints

The combination of modular simulator, md integrator, domain decomposition and constraints caused illegal memory access and could crash before step 0.

Issue 5236

### Fixes for `gmx` tools

### Handle trajectory reading error in gmx tcaf

`gmx tcaf` now fails with informative error message when failing to read a trajectory.

Issue 5238

### Fixes that affect portability

#### Fix linking error on Windows with Intel oneAPI

Fix `LINK : fatal error LNK1181: cannot open input file 'm.lib'` error encountered when using `icx` compiler on Windows by using dynamically-detected name for the `m` library.

Issue 5240

#### Work around the interal compiler error in Visual Studio 2022

Visual Studio 2022 will fail when compiling `calc_verletbuf.cpp`. This is a bug in the compiler, but we introduce a workaround for it.

Issue 5252

### Miscellaneous

#### GROMACS 2024.4 release notes

This version was released on October 31st, 2024. These release notes document the changes that have taken place in GROMACS since the previous 2024.3 version, to fix known issues. It also incorporates all fixes made in version 2023.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fix missing non-bonded interactions close to cut-off with GPUs

The GPU rolling pruning employed with the dual pair list setup used an incorrect list part calculation. This caused a few parlist entries not to be updated after the initial pruning, leading to a small number of missing pair interactions. Unless the pair list lifetime `nstlist` was very large, these interactions were close to the cut-off and therefore the errors were small.

Affected simulations: all GPU accelerated runs using GROMACS versions starting with the 2018 release with dual pair list enabled. This is enabled automatically with default Verlet buffering, but it is disabled if the Verlet buffer is manually set (or if the `GMX_DISABLE_DYNAMICPRUNING` environment variable is set).

Impact: Our analysis shows that the error caused by the missing interactions does not have a measurable effect on either forces or energy conservation in most atomistic simulations with default simulation settings. While a manually increased `nstlist`, hence increased (outer) pair list lifetime, leads to an increase in the number of missing interactions, for useful value of `nstlist` the effect is still negligible. There can be measurable effects in systems with vacuum, gas, or with excessively large `nstlist` values. In our testing we could not detect artifacts apart from systems crashing, so the chance of undetected incorrect results is small.

Issue 5138

### Add effect of perturbed masses to foreign Hamiltonian differences

The effect of perturbed masses was missing from the foreign Hamiltonian differences. This meant that effects of perturbed masses were ignored by `gmx bar`. Note that perturbed masses did contribute to dH/dlambda.

Issue 5195

### Fix illegal memory access with more VCM than T-coupling groups

When there were more center of mass motion removal groups than temperature coupling groups, mdrun could crash due to illegal memory access. This bug did not affect the simulation results.

Issue 5167

### Fix crashes when some atoms are not part of a VCM group

Uninitialized memory was read when some atoms were not part of a center of mass motion removal group. Often this memory contained zero and everything would work as it should. But when this memory was not zero, the simulation would crash within a few steps.

Issue 5181

### Fix parameter handling for coarse-grained bonded potentials

Free-energy perturbation has never been supported for restraint angle, restraint dihedral, and combined bending-torsion potentials sometimes used in coarse-grained force fields. This led to the hypothetical B-state parameters being incorrectly handled in `gmx grompp` and then incorrectly serialized to/from the .tpr file.

This was generally benign, even when such functional forms were in use. However, when other interactions were perturbed, the incorrect parameter handling made it look like the interactions with wrongly implemented parameters were intended to be perturbed. This condition was sometimes achieved when using alchemical-embedding system-preparation workflows.

Now the hypothetical B-state parameters are handled correctly, so FEP-based workflows can succeed. Perturbation of the above-mentioned interaction types is still not supported, however.

Issue 5129 Issue 5144 Issue 5147

### Fix Colvars output files always written to the working directory.

Colvars output files are now written in the same folder as edr file.

Issue 5122

### Forbid the usage of triangle constraints with -update gpu

Prevent using update on GPU if there are triangle constraints.

Issue 5123

### `gmx_mpi mdrun` could hang when using GPUs and separate PME ranks

A logic error in task assignment for `-nb gpu -pme cpu` and separate PME ranks made the (default) `-ddorder interleave` hang. Now it works.

Issue 4345

### Dynamic load balancing was ineffective when special forces were present

The timing for DLB included the calculation of special forces, e.g. pull code and rotation. As these require communication, imbalance might not have been measured. Now special forces are excluded from the timings.

Issue 5188

### Fix incorrect memory access with perturbed non-bonded and OpenMP

Old, invalid indices could be flagged for reduction of force buffers over OpenMP threads for perturbed non-bonded interactions. This could never lead to incorrect results or crashes. But performance might be improved now.

Issue 5220

### Fixes for `gmx` tools

### grompp checked incorrect B-state charges with free-energy decoupling

When the *`couple-moltype`* (page 87) options was used, `gmx grompp` would check the A-state charges instead of the B-state charges. This would lead to incorrect or no warnings when B-state of the system had non-zero net charge.

Issue 5200

### grompp, and mdrun, could exit with large mass differences

When masses of atoms differed more than a factor 327, both `gmx grompp` and `gmx mdrun` could exit with an assertion failure about an infinite enery drift estimate.

Issue 5222

### Fix dump crash with Colvars values

Fix `gmx dump` crash when trying to output the binary Colvars state file.

Issue 5034

### Fix element and atom number deduction in during preprocessing

Two-letter atom names like BR or CL were not correctly handled in the preprocessing during `gmx editconf`, `gmx pdb2gmx` and `gmx grompp`, leading to incorrect element and atom number assignment. This could lead to incorrect element names in output files and possibly incorrect behaviour in QMMM simulations.

Issue 5182

### Fix hang observed with NVSHMEM enabled PME-PP force transfers

A hang in PME-PP force transfers during NVSHMEM runs was resolved, occurring in certain conditions. It should also be noted that the NVSHMEM enabled PME-PP force transfers does not support charge perturbation.

Issue 5136

### Fixes that affect portability

### Fix physical validation with Pymbar version 4

Pymbar version 4 has a different API compared to version 3. Now we support using either of those versions and internally handle the API differences.

Issue 5130

### Fix compatibility with VkFFT 1.3.5

Fix crash on GROMACS shutdown when external VkFFT 1.3.5 is used.

Issue 5184

### Miscellaneous

### Work around FetchContent warnings in CMake 3.30 and newer

CMake 3.30 began to complain about GROMACS use of FetchContent, so now we tell such new versions to use the old policy.

Issue 5140

### Fix various crashes when GMX_USE_COLVARS=NONE

Colvars MDModule did not defined the Colvars custom mdp variables when the Colvars library was not compiled, preventing tools read correctly a tpr file generated with a GROMACS-Colvars version. Prevent also the creation of a tpr file if Colvars module is activated whereas GROMACS was not compiled with Colvars. Proper exit if a Colvars simulation is launched whereas GROMACS was not compiled with Colvars.

Issue 5055

### Fix reading cgroups in some kubernetes containers

Modern versions of kubernetes/docker do not appear to mount /etc/mtab in the containers, and if we did not find cgroups we would not detect CPU limits set through cgroups. Fixed by reading /proc/mounts instead. This will only influence performance when running in (some) containers.

Issue 5148

### Collected fixes in the Colvars library

Several bugs, both recent and long-standing, have recently been fixed in the Colvars library: in the list below, the links corresponds to issues or pull request in the Colvars repository.

- Fixed undefined behavior when getting the current working directory from `std::filesystem`, which could have affected multiple-walkers metadynamics runs (Colvars PR 728).
- Fixed the gradients and the metric functions of collective variables of `distanceDir` type (Colvars PR 724).
- Fixed the definition of an `orientation` type collective variable in a rotated frame of reference (Colvars PR 715).
- Implemented the contribution of fitting to the forces applied onto of variables with vector values defined in a rotated fram (Colvars PR 713).
- Fixed a crash in certain metadynamics simulations without using interpolating grids (Colvars PR 706).
- More consistent behavior when defining multiple biases and running with more than one thread per task (Colvars PR 694).
- Prevented the creation of spurious output files for runtime histograms (Colvars PR 675).

### Enable NVCC flags checks for Windows

When building GROMACS for Windows with CUDA support, the checks for testing compatible compute architectures were disabled. Hence, GROMACS was trying to compile for all of them which can induced failed builds when CUDA is not compatible anymore with old compute architectures.

Issue 5152

### GROMACS 2024.3 release notes

This version was released on August 29th, 2024. These release notes document the changes that have taken place in GROMACS since the previous 2024.2 version, to fix known issues. It also incorporates all fixes made in version 2023.5 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Remove double dispersion correction with MTTK barostat

When the MTTK barostat was used with dispersion correction, the pressure correction was applied twice. This caused the density to be too high.

Issue 5072

#### Correct zero Verlet buffer for system with only large masses

For system with most masses much larger than zero, the Verlet buffer estimate could be zero. Coarse-grained simulations could be affected by this issue.

Issue 5098

#### Fix incorrect energy group pair assignment of pair interactions

Energies for bonded pair interactions, typically 1-4 interactions, would be assigned to incorrect energy group pairs. With free-energy calculations this could lead to illegal memory access.

Issue 5109

#### Correct AWH metric with free-energy and Beutler soft-core

The AWH metric was incorrect for free-energy dimensions when Beutler soft-core interactions were used. The free-energies themselves were unaffected by this issue.

Issue 5107

#### Implement Lennard-Jones force-switch in free-energy kernel

Perturbed non-bonded Lennard-Jones interactions would use plain cut-off with potential shift when force-switch was specified. The errors this causes on the potential and force are negligible with the typical Charmm setup. When (de)coupling a complete, dense system there is a measurable error in the pressure, but this is again negligible when coupling a single molecule.

Issue 5016

### mdrun now computes the initial bonded distances with periodic molecules

When using periodic molecules, mdrun could quit with an error message about multiple assignments of bonded interactions telling the user to report the issue. The actual source of this was that mdrun did not determine the communication distance required for bonded interactions. Now this distance is determined.

Issue 5086

### Avoid using invalid device streams

Issue has only been seen via assertions in the GCC 13 standard library that guard against null-pointer dereferences. The dereferenced null pointers were never used in actual simulations.

Issue 5087

### Fix masks and alignment checks in Arm SVE SIMD

The double precision SVE implementation contained an incorrect mask for SVE implementations of 256 bits or more. This would lead to instantaneous extreme forces/crashes on the first step, so it should never have influenced any production simulations unnoticed. The alignment checks for some SVE routines have also been modified to correspond to the SIMD width, although this is formally not required on Arm, so it will not have caused any errors.

Issue 5080 Issue 5120

### Added an assert to ensure valid cutoff when calculating atom density

Simulations with *rvdw* (page 62) and *rcoulomb* (page 60) set to 0 could crash with a segmentation fault when calculating the effective atom density. This is now handled by asserting valid values and halting with a proper explanation.

Issue 5095

### Fixes for `gmx` tools

### Permit `gmx tune_pme` to understand `md.log` files

The format of `md.log` files changed in 2024 which broke the ability of `gmx tune_pme` to understand how previous runs worked by parsing that file. Now `gmx tune_pme` understands better.

### Avoid crash of convert-tpr on velocity-less tpr files

The convert-tpr now works on tpr files that do not contain velocities, as generated for, e.g., energy minimization.

Issue 5080

### Avoid premature termination when using `gmx hbond`

Fixed an error that caused gmx hbond to prematurely terminate if there were no donors or acceptors of hydrogen bonds in the chosen selection.

Issue 5059 Issue 4985

### Add grompp warning about zero AWH cover diameter when sharing a bias

### Fix TPR generation with Andersen Massive thermostat and constraints

A restriction to not generate TPR files for the Andersen thermostat with constraints was incorrectly applied to also the Andersen Massive thermostat.

Issue 5093

### Fixed filenames of generated CP2K files in QMMM MdModule

If there were no files provided with `grompp -qmi` option, module used `topol/_cp2k.inp` instead of `topol_cp2k.inp`

### Added a check in `gmx grompp` to ensure that cutoffs are valid

At least one of *rvdw* (page 62) and *rcoulomb* (page 60) must be > 0 when using the Verlet cutoff scheme. This is now checked by `gmx grompp`.

Issue 5095

### Fix `gmx dump` with options `-sys` and `-orgir`

These `gmx dump` options had no effect. This has now been fixed.

Issue 5124

### Fixes that affect portability

#### Fix build when GoogleTest is installed system-wide

In some cases (e.g., on FreeBSD), GROMACS build system can mix up different versions of GoogleTest leading to compilation errors. Now we always prefer the bundled version.

Issue 5046

### Miscellaneous

#### Fix detection of rocFFT installed in non-standard locations

#### Fix performance regression on PVC with oneAPI 2024.2

oneAPI 2024.2 enabled automatic GRF selection by the device compiler, causing up to 50% worse NBNxM kernel performance on Intel Data Center GPU Max (Ponte Vecchio) devices. We now force the use of small GRF mode on PVC to avoid that. Arc GPUs were not affected.

Issue 5105

## GROMACS 2024.2 release notes

This version was released on May 10th, 2024. These release notes document the changes that have taken place in GROMACS since the previous 2024.1 version, to fix known issues. It also incorporates all fixes made in version 2023.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Prevent crashes with AdaptiveCpp/hipSYCL

In some cases, GROMACS could randomly crash on neighbor-search steps with `hip_queue: hipMemsetAsync() failed (error code = HIP:1)` error. This was caused by a GPU buffer re-allocation while an operation on the buffer was pending. We add an explicit synchronization to avoid it.

Only AdaptiveCpp/hipSYCL builds are affected. This should not have caused any incorrect physics, only the `mdrun` crash.

Issue 5078

### Fixes for `gmx` tools

#### Improve errors and warnings to avoid poor results with gmx msd

Fail properly if `-trestart` is not divisible by `-dt` or if `-dt` > `-trestart`. A warning is generated if the two values are equal.

Issue 5051

#### Improve error messages from editconf and trjconv

Fail properly if a tpr file is not supplied when using the -conect option. Also avoid a "File input/output error:" when no tpr file is supplied when it is required for -pbc options.

Issue 5032

#### Avoid floating point exception with corrupted xtc files

Reading a corrupted xtc file could cause a floating point exception. Now the gmx binary will exit with an error message.

Issue 5037

#### Make Colvars file backup scheme consistent with GROMACS

Colvars output files in GROMACS versions 2024.0 and 2024.1 were backed up only once, regardless of the value of `GMX_MAXBACKUP`. This is corrected in 2024.2 and later releases, by letting Colvars back up existing output files consistently with other files produced by GROMACS itself.

Issue 5071

### Fixes that affect portability

#### Enable usage of Intel compiler with AVX-512 SIMD on AMD Zen 4

The fix is to use newer style compiler flags for Intel oneAPI DPC++/C++ Compiler and AVX-512. Older style flag `-xCORE-AVX512` supports only Intel CPUs, while newer style `-march=skylake-avx512` additionally supports AMD CPUs. This fixes building with Intel compiler on AMD CPUs with Zen 4 cores.

Issue 5043

### Warn about unsupported compilers sooner

Now CMake reports that a user's compiler is broken or unsupported before running into other fatal issues that would have obscured a critical problem.

Issue 5056

### Enable specification of custom install rpath during build configuration

CMake build and installation steps will now append to the `CMAKE_INSTALL_RPATH` value provided by the user at the configuration time (if any) instead of overwriting it.

Issue 5064

### Permit `dpcpp` to appear in the path to the compiler

The compiler named `dpcpp` is no longer supported, so CMake prohibits its use. However previously we inadvertently also prohibited any compiler whose full path also contained the string "dpcpp," which was not intended.

Issue 4716

### Miscellaneous

### NBLIB always used geometric Lennard-Jones combination rules

Now NBLIB detects the Lennard-Jones combination rules (or none) from the Lennard-Jones parameters set by the user.

Issue 5015

### Fix cuFFTMp compilation issue

The cuFFTMp build is now more likely to compile because the compiler now sees the correct path.

### Fix performance regression for some cases on latest NVIDIA GPUs

In version 2024.0 we included optimizations to non-bonded force calculations on NVIDIA GPUs which improved performance for a range of cases, through loop unrolling tuning. Here, we tweak the optimization to fix a regression in a specific case, which was discovered to be caused by this tuning.

Issue 4867

### Address performance regression on ARM with clang and SVE

The non-bonded CPU SIMD kernels contained loops which were not unrolled, in particular with the LLVM backend of clang on ARM SVE. Other combinations of compilers and architectures might also have been affected. Note that there are still other performance regressions in certain versions of these kernels on multiple platforms.

Issue 5036

### Fix crash in NbnxmSetupTest.CanCreateNbnxmGPU

`NbnxmSetupTest.CanCreateNbnxmGPU` could crash in GPU builds with an error in `freeDeviceBuffer` function during pairlist deinitialization.

Issue 4888

### Fix incorrect mdp keyword for mass repartitioning in documentation

The mdp keyword for mass repartitioning was incorrect in the list of mdp options in the documentation.

Issue 5007

### Report AdaptiveCpp/hipSYCL compiler path in `gmx -version`

In 2024.0 and 2024.1 it was empty.

Issue 5045

### Prevented CTest from running racy test cases concurrently

Issue 4654 Issue 4975

### GROMACS 2024.1 release notes

This version was released on February 28th, 2024. These release notes document the changes that have taken place in GROMACS since the previous 2024.0 version, to fix known issues. It also incorporates all fixes made in version 2023.4 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Avoid rounding errors with Parrinello-Rahman coupling of triclinic boxes

With triclinic boxes and isotropic or diagonal Parrinello-Rahman pressure coupling, the off-diagonal scaling matrix arguments could be non-zero. This was harmless, apart from causing an assertion failure when update was executed on a GPU.

Issue 4974

### Fixes for `gmx` tools

#### Fixed parsing of phases in gmx nmtraj

Issue 4968

### Fixes that affect portability

#### Fix compiler flag check with oneAPI DPC++

Some compiler warning flags were not tested properly when compiling for NVIDIA/AMD GPUs using Intel oneAPI. No user-facing impact expected.

Issue 5005

### Miscellaneous

#### Address mdrun performance regression with non-bonded interactions on CPU

The heuristics for choosing the inner pair-list interval has been improved to regain or improve the performance on non-bonded interactions on the CPU. Correctness is unaffected.

Issue 4988

#### Fixed nbnxm-test failure when bounds checking was enabled for stdlib

Issue 4966 Issue 4973

#### Increase tolerance of mdrun continuation tests in double precision

The mdrun continuation tests could fail in double precision with errors just above the tolerance.

Issue 4788 Issue 4931

#### Fix missing OpenMP initialization in grompp

The problem only manifested as a pop-up message when running `gmx grompp` built with Visual Studio. Since this message was blocking the application, it could mess up with test suite or automated scripts. No other deleterious effects noted.

Issue 4961

#### Fix grompp & mdrun crash when GMX_USE_COLVARS=NONE

Colvars MDModule was not properly constructed when the Colvars library was not compiled. Colvars mdp fields handler was returning a nullptr.

Issue 4980

#### Fix broken completion script on zsh-5.9 (Mac OS Sonoma)

The compinit command must be loaded by the script before loading bashcompinit, or there will be an error message when sourcing GMXRC on some zsh versions.

Issue 4986

### 11.5.2 Major release

#### Highlights

GROMACS 2024.0 was released on January 30th, 2024. Patch releases may have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

As always, we've got several useful performance improvements, with or without GPUs, all enabled and automated by default. In addition, several new features are available for running simulations. We are extremely interested in your feedback on how well the new release works on your simulations and hardware. The new features are:

- The Colvars library can now be used natively from GROMACS. This simplifies the use of advanced enhanced sampling simulations.
- Reduced artifacts from Lennard-Jones pair interactions on the pressure by a configurable increase of the Verlet buffer. Can lead to a slight performance loss, especially for coarse-grained systems.
- Corrected several aspects of the deform option. Now simulations with box deformation behave correctly under high shear or when a solid or membrane fractures. This also means that the deform option is now suitable for computing viscosities.
- New option for hydrogen mass repartitioning in `grompp` enables easy access to performance improvements.
- Improvements to AWH, such as better control of the histogram growth factor as well as enabling automatic scaling of the target distribution based on the AWH friction metric.
- Configurable HeFFTe multi-GPU FFT options lets users fine-tune the settings for specific use-cases.

### New and improved features

#### The AWH exponential histogram growth can now be controlled

The accelerated weight histogram growth factor during the initial phase was hard-coded to 3. Now this value can be controlled by the user. It is set to 2 by default for increased stability.

If the TPR was generated with an earlier GROMACS version, the old default value of 3 will be used.

#### Added support for instrumentation based on wallcycle regions using NVTX/ROCTX/ITT

Basic support has been added for GPU tracing libraries so wallcycle main and sub-regions will show up in tracing timelines which can help with performance analysis. The tracing instrumentation support can be enabled with one of the following CMake variables: `GMX_USE_NVTX`, `GMX_USE_ROCTX`, `GMX_USE_ITT`.

Issue 4446

#### Collective variables (Colvars) module support

The collective variables (Colvars) library for enhanced sampling simulations has a new and improved interface, which simplifies greatly its use and distribution with current and future GROMACS releases. The new interface requires *no patching* and supports a full integration of the Colvars input and of its restart data with the GROMACS TPR and CPT files, respectively.

For documentation and details, please see *this section* (page 646) of the GROMACS doc along with the Colvars doc page for GROMACS. Additionally, messages in the GROMACS discussion forum can also be tagged with the colvars keyword for easier consultation.

#### Automatic metric scaled AWH target distribution

The AWH target distribution can now be automatically scaled by sqrt(AWH friction metric). Regions with higher friction (slower diffusion) will get a higher target distribution. This should generally lower the statistical error of the estimated free energy landscape. The new option is called 'awh1-target-metric-scaling' and can be applied to further modify all AWH target distributions and/or AWH user input, but is not recommended in general in combination with Boltzmann or Local-Boltzmann target distributions, due to the risk of feedback loops between the two adaptive update mechanisms.

### Performance improvements

#### Improved performance for inhomogeneous systems

The performance of systems with a lot of empty space is improved by optimizing the pair search grid for the effective atom density.

Issue 4805

### Flexible hydrogen mass repartitioning using grompp

Instead of using pdb2gmx, which modifies the topology, a flexible scheme for hydrogen mass repartitioning is now available in `grompp` through the `mass-repartition-factor` `mdp` option. This provides easy access to a performance improvement of close to a factor two.

Issue 4866

### Small performance regression to achieve more accurate pressure

To lower the effect on Lennard-Jones pair interaction on the pressure, the Verlet buffer has been increased for most simulations using default mdp settings. This can lead to a few percent performance loss, in particular when using GPUs. The effect will be strongest for systems with no or weak electrostatics, which includes most coarse-grained systems.

Issue 4861

### Reduced grompp and mdrun setup time for systems with many atom types

The Verlet buffer calculation could take many minutes for systems with thousands of different Verlet buffer atom types (different atom type and charge). Such times have now been reduced to seconds.

Issue 4892

### With wall potentials, bonded interactions can now be run on GPUs

### HeFFTe multi-GPU FFT plan options are now configurable

New environment variables `GMX_HEFFTE_RESHAPE_ALGORITHM`, `GMX_HEFFTE_USE_GPU_AWARE`, `GMX_HEFFTE_USE_PENCILS`, and `GMX_HEFFTE_USE_REORDER` permit the HeFFTe plan options to be configured at run time. The performance obtained can vary with the quality of implementation of e.g. the GPU-aware MPI library, as well as the layout and number of the GPUs participating in the 3D-FFT. Users can now find and use the best settings for their case. See the HeFFTe documentation for more details.

### Improvements to GROMACS tools

### Improved Einstein viscosity calculation in gmx energy

Viscosity calculation using the Einstein formula is convenient as this does not require extremely frequent pressure tensor data. However, the implementation of the calculation was inconvenient for long simulations and could take hours to complete. Improved stepping through the data reduces the computational time to minutes and provides much clearer output.

### XVG output from `gmx rdf` now uses 6 decimal places

The output from `gmx rdf` now uses more decimal places in order to avoid rounding issues. These issues led to perceived erroneous shifts in the results.

Issue 4647

### Handle CYX-CYX disulfide bonds in `gmx pdb2gmx`

Naming CYS residues as CYX shows that they should form a disulfide bond. `gmx pdb2gmx` will now correctly interpret them as disulfide bond forming residues.

Issue 4929

## Bugs fixed

### The deform option was unsuited for flow simulations

The deform option now only deforms the box and does not modify atom positions anymore. In contrast to previous versions, it instead corrects the velocities of particles when they are shifted by a periodic box vector. Now, deform is also useful for shear flows. Applications where the system was stretched until some interactions broke were probably not affected measurably by these issues. Note that a velocity profile should be generated when using deform with the current or later versions. An mdp option has been added to let `grompp` do this.

Issue 4607

### mdrun now checks for excluded pairs beyond the cut-off with reaction-field and FEP

With reaction-field electrostatics and free-energy calculations, excluded atom pairs are not allowed to be beyond the Coulomb cut-off distance. Now mdrun checks for this and throws an error when this occurs.

Issue 4667

### Limit pressure deviations due to missing Lennard-Jones interactions

For systems dominated by Lennard-Jones interactions, i.e. with no or very weak electrostatics, e.g. most coarse-grained systems, the Verlet buffer was often set such that missing Lennard-Jones interactions could lead to the pressure increasing by more than 1 bar over the lifetime of the pair list. Now an mdp parameter has been added to limit the deviation in the average pressure. The default tolerance is 0.5 bar.

Issue 4861

#### enemat now prints correct headers when using `-free` or `-eref` options

Fixed a long-standing bug when `gmx enemat` would output incorrect headers to XVG.

Issue 4812

#### gmxapi.commandline_operation implicitly converts *input_files* to absolute paths

Relative paths in the *input_files* mapping are now explicitly relative to the working directory of the caller.

Issue 4827

### Deprecated functionality

### Changes anticipated to GROMACS 2024 functionality

#### The analysis tool `gmx chi` no longer deprecated

Given the community interest, the decision was made to keep `gmx chi`.

Issue 4108

### Functionality deprecated in GROMACS 2024

#### The analysis tool `gmx gyrate-legacy` deprecated

`gmx gyrate` has been partly re-implemented in the modern GROMACS analysis framework. The old implementation is still available as `gmx gyrate-legacy`. Please plan to update to the new version. Please let the GROMACS developers know of desired functionality missing from, or broken in, the new implementation.

Issue 4927

#### The analysis tool `gmx hbond-legacy` deprecated

`gmx hbond` has been partly re-implemented in the modern GROMACS analysis framework. The old implementation is still available as `gmx hbond-legacy`. Please plan to update to the new version. Please let the GROMACS developers know of desired functionality missing from, or broken in, the new implementation.

Issue 4927

### The analysis tools `gmx sans` and `gmx saxs` deprecated

`gmx sans` and `gmx saxs` has been partly re-implemented under new name `gmx scattering` in the modern GROMACS analysis framework. The old implementations are still available as `gmx sans-legacy` and `gmx saxs-legacy`. Please plan to update to the new version. Please let the GROMACS developers know of desired functionality missing from, or broken in, the new implementation.

Issue 4927

### The Xeon Phi support will be removed

Intel Xeon Phi series of accelerators has been discontinued in 2020, and most supercomputers using it are now retired. The support for Xeon Phi is deprecated in GROMACS 2024 and will be removed in the next release.

Issue 4740

### Functionality deprecated in GROMACS 2022

#### GMX_OPENCL_NB_CLUSTER_SIZE CMake variable deprecated in favor of GMX_GPU_NB_CLUSTER_SIZE

Both OpenCL and SYCL support different cluster sizes, so GMX_GPU_NB_CLUSTER_SIZE should be used going forward.

### Guessing masses and atomic radii from atom names is deprecated

When atom masses or van-der-Waals radii are needed, we suggest building a proper GROMACS topology instead of using PDB files directly, even if the tool supports it.

Issue 3368 Issue 4288

### Functionality deprecated in GROMACS 2021

#### `mdrun -deffnm` to be removed

This functionality is convenient when running very simple simulations, because it permits grouping of a set of files that then differ only their suffix. However, it does not work in the wider case of an `mdrun` module (or modules) writing multiple `.xvg` output files. The resulting filenames collide. That, and its interaction with checkpointing and appending, have led to quite a few bug reports.

Because users can use a folder to group files (a standard mechanism that they understand from experience outside of GROMACS), we can build and test better software for them if we remove the erstwhile convenience of `mdrun -deffnm`. Please update your workflows accordingly.

Issue 3818

### OpenCL to be removed as a GPU framework

Issue 3818 Work is underway for ports to AMD and Intel GPUs, and it is likely that those ports will not be based on the current GROMACS OpenCL port. Nvidia GPUs are targeted by the CUDA port, and no changes are expected there. The core team can't maintain, test, and extend up to 4 ports with current resource levels. Since there are no prospects of an emerging GPU vendor in HPC needing OpenCL support, we will remove the OpenCL port once AMD and Intel support is established in other ways.

### Support for version 1 of the hardware locality library `hwloc`

Issue 3818 Version 2 has been supported in GROMACS for several years. The capabilities of newer hardware and hardware-support APIs are of most interest for GROMACS moving forward, so we should minimize our testing work and encourage clusters to upgrade older `hwloc` installations.

### Legacy API

Issue 3818 The legacy installed headers have been deprecated for a while, however we wish to state more broadly that all headers found within the `src` directory tree of GROMACS are intended for internal consumption only, and are thus subject to change without notice. Further, the form and contents of the `libgromacs` library and related CMake targets may change as we move towards building APIs and supporting machinery that can be stable and supported in the long term.

### Functionality deprecated in GROMACS 2019

### Generation of virtual sites to replace aromatic rings in standard residues

Issue 3254 These are thought to produce artefacts under some circumstances (unpublished results), were never well tested, are not widely used, and we need to simplify pdb2gmx.

### Benchmarking options only available with `gmx benchmark`

Issue 3255 Options such as `-confout`, `-resethway`, `-resetstep` are not intended for use by regular mdrun users, so making them only available with a dedicated tool is more clear. Also, this permits us to customize defaults for e.g. writing files at the end of a simulation part in ways that suit the respective mdrun and benchmark use cases, so `-confout` will no longer be required.

#### `gmx mdrun -nsteps`

Issue 3256 The number of simulation steps described by the .tpr file can be changed with `gmx convert-tpr`, or altered in .mdp file before the call to `gmx grompp`. The convenience of this mdrun option was outweighed by the doubtful quality of its implementation, no clear record in the log file, and lack of maintenance.

#### Portability

#### Always use the Boost version bundled with GROMACS

Boost 1.83 is known to have compatibility issues when using Clang compiler on FreeBSD and Linux. This changes to always use the bundled Boost version, even if another version is present on the system.

Issue 4893

#### Miscellaneous

#### Changed default value of tau-p to 5 ps

The new default value for the pressure coupling time of 5 ps is larger than tau-t, as is recommended, and should work well for most simulations.

## 11.6 GROMACS 2023 series

### 11.6.1 Patch releases

#### GROMACS 2023.5 release notes

This version was released on May 3rd, 2024. These release notes document the changes that have taken place in GROMACS since the previous 2023.4 version, to fix known issues. It also incorporates all fixes made in version 2022.6 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### The Verlet buffer estimate could be wrong with initial coordinates out of the box

Due to incorrect periodic boundary treatment in the density estimate for the Verlet buffer calculation, the Verlet buffer could be off when many atoms were outside the rectangular unit cell. In nearly all cases this would lead to a slightly too large buffer and almost never to a too short buffer.

Issue 5002

#### With energy minimization, dispersion correction energies were double counted

The only affected the reported energies, as dispersion correction does not affect the forces on the atoms.

Issue 5031

### Correct forces with virtual_sites1

Virtual sites with a single constructing atom did not have their force distributed to the constructing atom. This likely did not cause silent errors, as this completely defeats the purpose of such virtual sites.

Issue 4978

### Avoid rounding errors with Parrinello-Rahman coupling of triclinic boxes

With triclinic boxes and isotropic or diagonal Parrinello-Rahman pressure coupling, the off-diagonal scaling matrix arguments could be non-zero. This was harmless, apart from causing an assertion failure when update was executed on a GPU.

Issue 4974

### Add checks for the combination of multiple time stepping and SD integrator

The documentation says that this combination is not supported, but the check for this in grompp was lost. Now both grompp and mdrun will exit with an error.

Issue 3775 Issue 5053

### Correct dH/dlambda with MTS and perturbed constraints

When doing free-energy calculations with multiple time stepping and perturbed constraint lengths, the slow force contributions to dH/dlambda were incorrectly multiplied by the MTS factor. The effect of this is usually rather small when only integrating the PME grid part less frequently, but can be substantial when the non-bonded pair interactions are not integrated at every step.

Issue 5060

### Fixes for `gmx` tools

### Fix grompp crash with atoms far out of the box

The Verlet buffer tolerance code did not put atoms correctly in the box, which could lead to illegal memory access in grompp

Issue 5002

### Fix LJ combination rules mixup in nonbonded-benchmark tool

The gmx nonbonded-benchmark tool would use geometric LH combination rules instead of Lorentz-Berthelot and Lorentz-Berthelot instead of no combination rules.

Issue 4963

### Fix `make_ndx` behavior with splitres

Since GROMACS 2023, `gmx make_ndx` would only output the first atom in each residue when using `splitres` command. Now the old behavior is restored, where all atoms in the residue are printed.

Issue 5049

### Fix invalid memory access in `gmx make_ndx` when using splitres/splitat

Could manifest in application crashes or garbled output data (for example, the output group names had empty prefix).

Issue 5050

### Fixed references to old tool names in some commands

There were references to old tool names, e.g., g_energy and g_bar in some commands.

Issue 5074

### Fixes that affect portability

### Use RDTIME instead of RDCYCLE on RISC-V

Starting with Linux 6.6, the `rdcycle` instruction cannot be called from userland for security reasons, making GROMACS fail with `SIGILL`. Now it is replaced with `rdtime`.

Issue 4967

### Miscellaneous

### Fix crashes with GPU direct communication for some atypical run configurations

When GPU direct communication was used in combination with atypical run conditions (e.g. many thread-MPI tasks on each GPU), crashes could occur when the tasks became very out of sync. This has now been made more robust.

Issue 5024

### GROMACS 2023.4 release notes

This version was released on January 24th, 2024. These release notes document the changes that have taken place in GROMACS since the previous 2023.3 version, to fix known issues. It also incorporates all fixes made in version 2022.6 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Freezing AWH simulations with multiple MPI ranks per walker

AWH simulations with biases shared by multiple walkers and using multiple MPI ranks per walker would hang when outputting energy, i.e., already at step 0.

There would not be any incorrect results.

Issue 4925

#### With AWH for FE, mdrun could exit with an assertion failure

The would happen e.g. when using soft-core only for LJ, whereas it was also needed for Coulomb or when decoupling large molecules. Now proper error messages are generated.

Issue 4938

#### Missing force contribution on neighbor search steps with GPU update

On a neighbor search step, when virial is _not_ computed (i.e., when `nstlist` (page 57) is not a multiple of `nstpcouple` (page 67)) and GPU update is used, there could have been a race between GPU buffer clearing and copying of forces computed on the CPU, leading to missing some of the force contribution on this step.

This should not have ever happened unless the GPU was heavily oversubscribed or hipSYCL/AdaptiveCpp was used, and even in those cases, the problem was very unlikely to manifest.

Issue 4937

#### Fixes for `gmx` tools

#### GROMACS error messages no longer include broken links

Two error messages used to contain links to the contents that are no longer available on the GROMACS website. These links have been replaced with the links to the relevant pages in the current documentation.

### Fixed a segmentation fault in grompp with AWH and FEP without FEP states

Issue 4932

### `gmx msd` now raises a proper error when given incompatible trajectory

`gmx msd` internally converts frame times to picoseconds. As a consequence, if the time cannot be expressed as an integral number of picoseconds (usually due to too frequent trajectory output), the tool would crash or produce an incorrect result.

Now, an error is printed in such cases.

Issue 4694

### Fixes that affect portability

### Fixed compilation with Boost 1.83

Issue 4909

### Miscellaneous

A grompp warning when using Parrinello-Rahman pressure coupling combining with position restraints suggested to use Berendsen instead. This suggestion has now been changed to C-rescale.

## GROMACS 2023.3 release notes

This version was released on October 19th, 2023. These release notes document the changes that have taken place in GROMACS since the previous 2023.2 version, to fix known issues. It also incorporates all fixes made in version 2022.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### mdrun now prints an estimate of the pressure deviation due to cut-off effects

Missing pair interactions due to using a constant pairlist over nstlist-1 steps cause a minor energy drift, but can in certain cases cause a measurable increase in pressure over the nstlist-1 steps due to missing Lennard-Jones interactions close to the cut-off. Now mdrun prints the average error in the pressure due to these missing LJ interactions.

Issue 4861

### The pressure deviation due to cut-off effects can now be controlled

As a temporary solution, the effect of missing Lennard-Jones interactions on the pressure can be limited by setting the environment variable GMX_VERLET_BUFFER_PRESSURE_TOLERANCE to the desired tolerance in bar.

Issue 4861

### Enable AWH to read 3D or higher-dimension user data

Mdrun could not start due to an error when reading in awh input data with dimensionality larger than two.

Issue 4828

### Allow restraining the rotation of a group of atoms during energy minimization

Avoid a segfault when using energy minimization together with enforced rotation.

Issue 4865

### Fix missing force buffer clearing with GPU DD and CPU bonded interactions

In simulations with domain decomposition using direct GPU communication for halo exchange (feature enabled with the GMX_ENABLE_DIRECT_GPU_COMM variable), a missing force buffer clearing prior to force halo exchange could lead to incorrect forces in cases where during a decomposition phase CPU-computed bonded interactions are not present whereas during the previous decomposition there were. Such errors are made significantly less likely by the lack of dynamic load balancing support with GPU-resident simulations that use GPU halo exchange.

Issue 4858

### Improve Verlet buffer estimation for systems with no or weak electrostatics

For systems which are dominated by LJ interactions, e.g. coarse-grained, the Verlet buffer estimate could be too small, because only the first derivative of the potential was taken into account. Now also the second and third derivatives are added. This can have a minor negative effect on performance.

Issue 4885

### Update virtual site velocities to avoid constraint instabilities

Virtual site velocities were only re-computed when writing velocities, but they are still integrated. This causes errors to accumulate. Now the velocities are updated at regular intervals in order to avoid (too) large velocities. This could cause runs to crash with a segmentation fault or domain decomposition error. Note that virtual site velocities are only used for output, they do not affect the positions.

Issue 4879

### Add workaround for OpenCL bug on AppleSilicon GPUs

After a resource leak was fixed in 2023.2, the OpenCL became broken on M1 Macs (and likely other AppleSilicon GPUs).

Issue 4852

### Fixes for `gmx` tools

### Fix legends in AWH data XVG extracted from AWH energy file

To avoid confusion, dimension legends, in the AWH data file XVG extracted from the AWH energy file with `gmx awh`, now start from the second dimension (the first column cannot have a legend). Dimension legends have also been modified to awh-dim%d (where %d is the dimension number).

Issue 4873

### Correctly dump VSITE2FD virtual sites

Systems with virtual site VSITE2FD (2 atoms with fixed distance) could previously not be handled by `gmx dump`. Note that this had no effect on simulation.

Issue 4845

### Fix DSSP tool

Fixed handling of Pi-helices in `gmx dssp` tool so now it produces output identical to original DSSP v4.1+.

Issue 4811

### Fix editconf -d with -noc

Unit cell vectors are now correctly output when running `gmx editconf -noc -d` which sets the box size to the largest dimensions of the system, but without centering.

Issue 4875

### Fix calculation of rotational kinetic energy in gmx traj

Computing the rotational kinetic energy using `gmx traj -ekr ekr.xvg` now returns correct results.

Issue 4889

### Fixes that affect portability

#### GROMACS can be compiled with Clang 16 with libc++ standard library

Unlike libstd++, libc++ follows C++ standards more strictly and therefore doesn't provide removed standard library classes. To enable compiling of GROMACS with Clang 16 and libc++, legacy symbols were replaced with modern C++17 equivalents in the bundled clFFT sources. The issue only affected OpenCL builds of GROMACS.

#### GROMACS automatically finds oneAPI libraries in new layout

MKL and SYCL-support libraries were reorganized in oneAPI 2023.2. Now GROMACS automatically finds the libraries it needs in both new and old layouts.

### Miscellaneous

#### Fix compilation for VMD plugin

This got broken with the changes for path handling.

Fixes Issue 4832

#### Unsupported anisotropic C-rescale rejected at preprocessing time

This pressure coupling setting previously resulted in an error at simulation runtime instead.

Fixes Issue 4847

#### CUDA Graph fixes related to neighbour search steps

When using the experimental CUDA Graphs feature, previously the code would crash for: runs where steps that involve virial calculations do not coincide with neighbour search (NS) steps - this issue is fixed by ensuring that CUDA Graphs are suitably updated on virial steps; runs where CUDA graphs are active on steps immediately preceding NS steps - this issue is fixed by adding necessary synchronization at the start of NS steps; and runs with an odd nstlist value - this is fixed by forcing graph reinstantiation rather than graph update in this case to correctly capture the odd/even pruning pattern into the graph.

Fixes Issue 4813

### Fix velocity vector copying in constr_vsiten

Fixed velocity vector copying in the function `constr_vsiten` because an energy minimization doesn't need a velocity vector. The fix avoids copying an empty vector and the corresponding SegFault.

Fixes Issue 4814

### Work around the performance regression on AMD MI250X with ROCm 5.5 or newer

With ROCm 5.5 and 5.6, some NBNXM kernels experienced up to 23% performance regression on MI250X compared to ROCm 5.3. We backported two patches from the 2024 branch that mostly mitigates this effect. There can still be a slowdown around 2% with ROCm 5.5+.

Issue 4874

## GROMACS 2023.2 release notes

This version was released on July 12th, 2023. These release notes document the changes that have taken place in GROMACS since the previous 2023.1 version, to fix known issues. It also incorporates all fixes made in version 2022.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Fix mdrun segfault when perturbing only LJ-14 parameters

Issue 4769

### Fix numerical instability with free-energy and LJ-PME

When combining free-energy calculation with PME for Lennard-Jones interactions, large rounding error would cause instabilities when two atoms were at short distance.

Issue 4780

### Fix mdrun domain decomposition setup limit of 715827882 atoms

An overflow in the domain decomposition grid setup limited the maximum number of atoms that could be simulated to max_int/3, unless the -dd option was specified.

Issue 4627

### Allow bonded pair interactions to be missing

With -noddcheck, mdrun did not allow bonded pair interactions to be missing. This is now allowed again.

Issue 4787

### Increase and check for atom count limits in output files

The maximum number of atoms in the checkpoint and trr files was 715 827 882 and has been increased to 1 431 655 765 atoms. Now mdrun exits with a clear error message when these limits are exceeded. For XTC files, the limit is raised by a separate fix.

Issue 4627

### Fix assertion failure with annealing and multiple T-coupling groups

Issue 4800

### Properly update the backup checkpoint

In 2023 and 2023.1, the `state_prev.cpt` file was never overwritten, and thus always contained the very first checkpoint for the run.

Issue 4810

### Fix domain decomposition with bonded interactions with long distances

mdrun would exit with an error about missing bonded interactions when domain decomposition was used and bonded interactions involved distances longer than the pairlist cutoff.

Issue 4818

### Fixes for `gmx` tools

### Avoid error in tools reading tpr files with energy minimization

Many tools would exit with an error "No v in input file" when reading a tpr file with integrator set to energy minimization, NM or TPI.

Issue 4774

### Tools now retain chain identifiers when reading PDB is used as input

Issue 4776

### The gmx hbond tool could produce random ouput

Because memory was not initialized, the gmx hbond tool could produce random output. This would not go unnoticed. This also fixes issues with the -ac and -life options.

Issue 4801

### Fixes that affect portability

### CMake config package

Depending on the build environment, GROMACS 2023 and 2023.1 could install a malformed `gromacs-config.cmake` file in *prefix*`/share/cmake/gromacs`*$SUFFIX/* that could cause failure of the `find_package(gromacs)` CMake command.

- `rocfft` is no longer ever a public dependency.
- The config package file now fully express the dependency on hipSYCL (Open SYCL), when relevant.

Issue 4793, Issue 4797

### Miscellaneous

### gmxapi.commandline_operation environment variable filtering

A new utility (*`gmxapi.runtime.filtered_mpi_environ()`* (page 726)) is available to remove MPI-related environment variables from `os.environ`, such as to prepare the subprocess environment of `gmxapi.commandline_operation`.

This is a follow-up to Issue 4423, for which the original fix appeared to be insufficient.

Issue 4736

### build-dependent checking for gmxapi runtime arguments

Depending on whether GROMACS was built with MPI support or thread-MPI support, some *gmx mdrun* (page 258) options are not defined. Such errors may only appear in the MD log file, and can thus be hard to identify in API use cases.

Additional checking has been added to `gmxapi.simulation.workflow.from_tpr()` to try to preempt user errors, and additional usage notes have been added to `gmxapi.mdrun`.

Issue 4771

### gmxapi.mdrun task uniqueness

Fix a bug in which all `gmxapi.mdrun` simulation tasks had the same ID (and working directory).

Issue 4795

### Fix for crash when CUDA Graphs are enabled on multi-GPU

A bug was introduced into version 2023.1 resulting in a crash when the non-default CUDA Graphs experimental feature was enabled on multi-GPU, caused by the introduction of extra synchronization which is not required for the CUDA graphs codepath. This version fixes the issue by avoiding this synchronization when graphs are in use.

Issue 4786

### Enable XTC support for gigantic systems

The (old) XTC format uses an internal char buffer whose size in bytes was stored as an integer in the file, which led to crashes when storing systems with more than roughly 300 million atoms. This version fixes the issue by introducing a 64-bit size only for large systems, and using a different magic number (2023) in the XTC header. This will only change the XTC format for large systems (which anyway would lead to crashes in the old version). Short-term the large system XTC files might not be readable by external tools (you will get errors about incorrect magic number), but we are working with external packages to update their implementations.

Issue 4628

### Fix resource leak in OpenCL

`gmx mdrun` built with OpenCL was slowly leaking memory when running on GPUs. That's fixed now.

Issue 4807

### Allow convert-tpr to assign initial velocities

To help ensemble projects like free energy that sometimes rely on thousands of simulations for each system, we now allow convert-tpr to assign a new set of random velocities rather than using grompp to regenerate the full tpr. This also fixes a bug where using 0 for velocity seed in the mdp file would have the same effect as -1 and cause a new seed to be generated from the OS.

Issue 4809

### Correct formulas for Nosé-Hoover thermostat

Several formulas describing Nosé-Hoover temperature coupling contained inconsistencies. The reference manual was updated to match the actual implementation.

Issue 4695

### Fix commandline-test on broken gcc versions

gcc 9.3.1 would fail generating proper paths for comparison, resulting in test failures.

Issue 4785

### Fix SIMD detection/recommendation on AMD Zen 4 / Genoa

Zen 4 provides a single AVX-512 unit, but in contrast to Intel chips it is still faster to use the single AVX-512 unit than the double AVX2 units, likely due to higher clock and lower instruction pressure. This change will select AVX-512 by default on Zen 4 (which can improve performance 5-10%), and it modifies the hardware detection so we only try to count AVX units on Intel CPUs. It also clarifies detection messages to make it clear it's based on expected performance rather than hardware support for a particular instruction set, and makes sure the stdout messages fit in a single line.

Issue 4715

## GROMACS 2023.1 release notes

This version was released on April 21st, 2023. These release notes document the changes that have taken place in GROMACS since the previous 2023 version, to fix known issues. It also incorporates all fixes made in version 2022.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Parallelization of TPI an normal modes working again

When running TPI or normal mode analysis with multiple MPI ranks mdrun would exit with an assertion failure.

Issue 4770

### The AWH metric could be incorrect for free-energy lambda dimensions

When different lambda components had different values for the same lambda point index, the AWH metric used dH/dlambda as input that used the derivative with respect to all lambda components. Note that this only affected the metric, not the sampling nor the free energy values.

Issue 4730

### Fix checkpointing of expanded ensemble simulations with domain decomposition

Expanded-ensemble simulations can now restart from a checkpoint when running multiple PP ranks.

Issue 4629

### Fix PME pipelining support in SYCL

When PME pipelining was used, long-range PME electrostatics were producing incorrect results in SYCL.

Only runs with >=3 GPUs and with direct GPU communication enabled (`GMX_ENABLE_DIRECT_GPU_COMM` env. var.) are affected.

Issue 4733

### Fix checkpointing of AWH friction metric for dimensions > 1.

The friction metric checkpoint i/o was wrong for dimensions > 1. This did not affect the AWH PMF or sampling, but would result in nonsense if the AWH friction tensors were used to calculate the diffusion in dimensions > 1.

Issue 4723

### Fixes for `gmx` tools

### Fix crash in `gmx solvate` when using solvent box in PDB format

Now a PDB file can be passed to the `-cs` option in `gmx solvate`. In previous releases (since at least 2016) this lead to a segfault.

### Fix creating index file from another index file

`gmx make_ndx` can again accept an index file alone as input, without associated structure file.

Issue 4717

### Allow selection of energy term by full name in `gmx energy`

It is now possible to select energy terms by full name. This is specifically helpful for terms starting with a number, such as "1/Viscosity" or "2CosZ*Vel-X", which could previously only be selected reliably by number.

Issue 4739

### Fix early crash in `gmx anaeig`

An internal change in GROMACS 2023 caused improper handling of optional program arguments leading to a crash in the program. This might have affected some other analysis tools.

Issue 4756

### Fixes that affect portability

### Fixed GMX_USE_TNG=off build

GROMACS can again be built without TNG support.

### Fixed abnormal termination during `gmx` startup

GROMACS made a call to `std::filesystem::equivalent` in a less than perfectly robust manner. This caused `gmx` to stop execution when linked against the (atypical) libc++ standard library when the build directory no longer existed.

Issue 4724

### Fixed CPU FFT with MKL 2023.0

Previously, GROMACS would fail during the initialization of CPU FFT when it was compiled with oneMKL 2023.0. This is fixed now.

Issue 4691

### Miscellaneous

### Workaround for strange compiler behavior to improve SYCL bonded kernel performance

For some SYCL targets (most notably, hipSYCL for AMD GPUs with ROCm 5.x), a very inefficient code was generated for bonded kernels. Now, bonded force calculation on GPU is expected to be up to 3 times faster.

Issue 3928

### Restored OpenMP acceleration of pulling routines

During internal code reorganization, OpenMP acceleration was accidentally disabled for pulling force calculation in GROMACS 2023. This is now fixed.

Issue 4747

#### Added support for new cuFFTMp interface

The interface to the cuFFTMp library has changed with its latest release in the NVIDIA HPC SDK version 23.3, which is required for NVIDIA Hopper GPU support. We have now added default support to the new interface, while retaining support for the legacy interface.

Issue 4731

#### Document workaround when MPI detection fails

MPI is an optional dependency of gmxapi even when building GROMACS without support for an MPI library. CMake's mechanism to find MPI can choke on broken MPI installations in ways that could be confusing. Now a work-around is documented for the convenience of a user who was not intending to use MPI.

Issue 4699

### 11.6.2 Major release

#### Highlights

GROMACS 2023 was released on February 6th, 2023. Patch releases may have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

As always, we've got several useful performance improvements, with or without GPUs, all enabled and automated by default. In addition, several new features are available for running simulations. We are extremely interested in your feedback on how well the new release works on your simulations and hardware. The new features are:

- The SYCL GPU implementation, which is the GPU portability layer that supports all major GPU platforms, has received major extensions in support for both platforms and features. To ensure portability in practice, the GROMACS GPU portability layer is actively developed with multiple SYCL implementations (hipSYCL, oneAPI DPC++, IntelLLVM) and regularly tested on multiple GPU backends.
  - SYCL supports more GPU offload features: bonded forces and direct GPU-GPU communication with GPU-aware MPI.
  - SYCL hardware support includes AMD (including RDNA support added here) and Intel for production as well as NVIDIA GPUs (not for production).
  - SYCL optimizations targeting important HPC platforms.
- PME decomposition has been optimized and extended to support offloading the entire PME calculation to multiple GPUs, including the FFT computation; when combined with cuFFTmp or heFFTe this enables much improved strong scaling (experimental feature).
- CUDA Graph support has been added to execute GPU-resident single-/multi-GPU simulations using thread-MPI entirely on the GPU to improve performance (experimental feature).
- Apple M1/M2 GPUs are now supported via the OpenCL GPU backend.
- New ensemble temperature mdp options allow setting the temperature of the ensemble for simulations without temperature coupling or with different reference temperatures.

- With *gmx dssp* (page 202), GROMACS now has a native implementation of the DSSP algorithm, which replaces `gmx do_dssp`.

### New and improved features

#### mdrun now also reports the conserved energy quantity with AWH bias sharing

#### Added option for setting the ensemble temperature

Several algorithms, such as pressure coupling and AWH, need the temperature of the system. When not all atoms are coupled to the (same) temperature, it is now possible to tell mdrun what the ensemble temperature is using two new mdp options.

Issue 3854

#### gmxapi.mdrun now publishes the simulation working directory path

`gmxapi.mdrun().output.directory` provides the (Future) path to the working directory/directories for the simulation(s). This can be useful in conjunction with *`gmxapi.utility.join_path()`* (page 724) to express data flow based on files known to the user to be produced by the simulation but not represented by other existing attributes of the *`OutputDataProxy`* (page 723).

Issue 4548

#### gmxapi.mdrun now captures STDOUT and STDERR

The GROMACS library prints a lot of output directly to standard out and standard error. Previously, this meant that simulator output that traditionally goes to the terminal would have to be caught from outside the Python interpreter. In mpi4py based ensembles, it could be challenging to catch the output at all, without manipulating the `mpiexec` command line.

`gmxapi.mdrun` now redirects STDERR and STDOUT during simulation, and provides paths to the resulting text files on new *stdout* and *stderr* outputs.

Reference Issue 4541

### Performance improvements

#### Update will run on GPU by default

The mdrun `-update auto` will by default map to GPU if supported. This gives a significant performance improvement with a single MPI rank.

### Increased default T- and P-coupling intervals

The default maximum values temperature and pressure coupling intervals have been increased from 10 to 100 steps. These values are used when the default value of -1 is specified in the mdp file and a lower value is used when required for accurate integration. The improves the performance of both GPU runs and parallel runs.

### The global communication frequency is independent of nstlist

The global communication frequency no longer depends on nstlist. This can improve performance in simulations using GPUs in particular.

### PME decomposition support with CUDA and SYCL backends

PME decomposition support has been added to CUDA and SYCL backends. With PME offloaded to the GPU, the number of PME ranks can now be configured with `-npme` option (previously limited to 1). The implementation requires building GROMACS with GPU-aware MPI and either with the *cuFFTMp library* (page 14) in a CUDA build configuration, or with *heFFTe* (page 14) in either a CUDA or SYCL build configuration.

GPU-based PME decomposition support still lacks substantial testing, hence is included in the current release as an experimental feature and should be used with caution (with results compared to those from equivalent runs using a single PME GPU). This feature can be enabled using the `GMX_GPU_PME_DECOMPOSITION` environment variable. The GROMACS development team welcomes any feedback to help mature this feature.

Issue 3884 Issue 4090

### CUDA Graphs for GPU-resident Steps

New CUDA functionality has been introduced, allowing GPU activities to be launched as a single CUDA graph on each step rather than multiple activities scheduled to multiple CUDA streams. It only works for those cases which already support GPU-resident steps (where all force and update calculations are GPU-accelerated). This offers performance advantages, especially for small cases, through reduction in both CPU and GPU side scheduling overheads. The feature can optionally be activated via the `GMX_CUDA_GRAPH` environment variable.

Issue 4277

### VkFFT support

For AMD GPUs, VkFFT has been integrated to provide performance improvements. Using this library is supported in all non-decomposed PME simulations (either single rank or single separate PME rank) and can be enabled with `-DGMX_GPU_FFT_LIBRARY=VKFFT` when using AdaptiveCpp (hipSYCL).

Issue 4052

### Changes to the API

#### Legacy aggregating headers have been removed.

Previously, some of the legacy API headers existed _only_ to aggregate `#include` lines for other installed headers. No guidance was provided regarding which header to include for a given feature. These redundant headers have been removed. Client software relying on `#include "gromacs/module.h"` will need to be updated with more specific `#include "gromacs/module/feature.h"` directives.

Issue 4487

### Improvements to GROMACS tools

#### `gmx do_dssp` replaced by native implementation of DSSP algorithm

`gmx do_dssp` replaced by native implementation of DSSP algorithm, version 4. Results for version 4 was formerly available with `gmx do_dssp -ver 4`. The new tool is called `gmx dssp`.

### Bugs fixed

#### Verlet buffer set correctly for inhomogeneous systems

The Verlet buffer estimation now uses an effective density for the system computing from the initial coordinates. This avoids underestimation of the buffer for (very) inhomogeneous systems.

Issue 4509

#### Fix segmentation fault for large atom and thread count

When the number of atoms times the number of OpenMP threads was larger than 2147483647, negative atom number could cause illegal memory access.

Issue 4628

#### Density-guided simulation normalization

With the `.mdp` option `density-guided-simulation-normalize-densities = yes`, the reference density and the simulated density values were previously divided by the sum of their values.

This lead to surprising behavior for reference densities with lots of negative voxel values: the density started to "repel" the protein structure instead of attracting it, if the total sum of voxel values was smaller than zero. The negative normalization constant lead to a sign change in voxel values.

To avoid this behavior, the reference density is now normalized so that the sum of *positive* values is unity, ensuring that the normalization constant is always positive.

Apart from avoiding the unexpected behavior, we expect that this also leads to smaller absolute differences between reference density and simulated density, with some small benefits for numerical stability.

This change affects all simulations where voxel values are negative (usually this excludes synthetic data) and that are run with `density-guided-simulation-normalize-densities = yes`, but only has a larger effect for: first, similarity measure `inner-product` as an effective force-constant scaling and, second, for all similarity measures where the sum of all voxel values was negative.

### gmxapi Python package avoids unnecessary MPI initialization

Delayed initialization of MPI (due to automatic behavior of `mpi4py`) avoids MPI initialization that previously occurred just by importing *gmxapi* (page 718). The previous behavior has been seen to cause strange interactions with resource management libraries like `libfabric` at unexpected times (such as during package installation) with *gmxapi* (page 718) version 0.3.

Issue 4693

### Fail-safe check for perturbed exclusions beyond rlist

With free-energy calculations, excluded non-bonded interactions involving at least one perturbed atom should not be beyond rlist when using PME. The check for this could have false negatives. Now the check is fail safe and will always trigger a fatal error when perturbed excluded pairs are beyond rlist.

Issue 3403 Issue 4321 Issue 4461

### Deprecated functionality

### Changes anticipated to GROMACS 2023 functionality

### Functionality deprecated in GROMACS 2023

### Functionality deprecated in GROMACS 2022

### GMX_OPENCL_NB_CLUSTER_SIZE CMake variable deprecated in favor of GMX_GPU_NB_CLUSTER_SIZE

Both OpenCL and SYCL support different cluster sizes, so GMX_GPU_NB_CLUSTER_SIZE should be used going forward.

### The analysis tool `gmx chi` will be removed

This tool has not been functional for a few years. Please comment at the linked issue if you have any interest in it.

Issue 4108

### Guessing masses and atomic radii from atom names is deprecated

When atom masses or van-der-Waals radii are needed, we suggest building a proper GROMACS topology instead of using PDB files directly, even if the tool supports it.

Issue 3368 Issue 4288

### Functionality deprecated in GROMACS 2021

#### `mdrun -deffnm` to be removed

This functionality is convenient when running very simple simulations, because it permits grouping of a set of files that then differ only their suffix. However, it does not work in the wider case of an `mdrun` module (or modules) writing multiple `.xvg` output files. The resulting filenames collide. That, and its interaction with checkpointing and appending, have led to quite a few bug reports.

Because users can use a folder to group files (a standard mechanism that they understand from experience outside of GROMACS), we can build and test better software for them if we remove the erstwhile convenience of `mdrun -deffnm`. Please update your workflows accordingly.

Issue 3818

### OpenCL to be removed as a GPU framework

Issue 3818 Work is underway for ports to AMD and Intel GPUs, and it is likely that those ports will not be based on the current GROMACS OpenCL port. Nvidia GPUs are targeted by the CUDA port, and no changes are expected there. The core team can't maintain, test, and extend up to 4 ports with current resource levels. Since there are no prospects of an emerging GPU vendor in HPC needing OpenCL support, we will remove the OpenCL port once AMD and Intel support is established in other ways.

### Support for version 1 of the hardware locality library `hwloc`

Issue 3818 Version 2 has been supported in GROMACS for several years. The capabilities of newer hardware and hardware-support APIs are of most interest for GROMACS moving forward, so we should minimize our testing work and encourage clusters to upgrade older `hwloc` installations.

### Legacy API

Issue 3818 The legacy installed headers have been deprecated for a while, however we wish to state more broadly that all headers found within the `src` directory tree of GROMACS are intended for internal consumption only, and are thus subject to change without notice. Further, the form and contents of the `libgromacs` library and related CMake targets may change as we move towards building APIs and supporting machinery that can be stable and supported in the long term.

### Functionality deprecated in GROMACS 2019

#### Generation of virtual sites to replace aromatic rings in standard residues

Issue 3254 These are thought to produce artefacts under some circumstances (unpublished results), were never well tested, are not widely used, and we need to simplify pdb2gmx.

#### Benchmarking options only available with `gmx benchmark`

Issue 3255 Options such as `-confout`, `-resethway`, `-resetstep` are not intended for use by regular mdrun users, so making them only available with a dedicated tool is more clear. Also, this permits us to customize defaults for e.g. writing files at the end of a simulation part in ways that suit the respective mdrun and benchmark use cases, so `-confout` will no longer be required.

#### `gmx mdrun -nsteps`

Issue 3256 The number of simulation steps described by the .tpr file can be changed with `gmx convert-tpr`, or altered in .mdp file before the call to `gmx grompp`. The convenience of this mdrun option was outweighted by the doubtful quality of its implementation, no clear record in the log file, and lack of maintenance.

### Removed functionality

#### The built-in viewer `gmx view` was removed

There is little use and no tests of this functionality, and it was deprecated in 2022.

Issue 4296

#### Some unmaintained utility scripts were removed

Several scripts in repository `scripts/` directory were not installed with the package, have not been maintained, and, as best we could tell, have not been used in a long time.

Issue 4639

### Portability

#### Full support for RISC-V

We now provide full support for RISC-V, including hardware cycle counters for efficient load balancing.

### Initial support for Apple silicon GPUs

We now recognize Apple-designed GPUs as a supported architecture in the OpenCL backend.

### VkFFT support for improved portability and performance on GPUs with OpenCL and SYCL

Support for the VkFFT GPU FFT library was added with two goals: improved portability across GPU platforms and better performance. VkFFT can be used with OpenCL and SYCL. For SYCL builds, VkFFT provides a portable backend for AMD and NVIDIA GPUs, and it is a better-performing alternative recommended at least on AMD with runs without PME decomposition (in non-HeFFTe builds). For OpenCL builds, VkFFT provides an alternative to ClFFT with much better performance and broader compiler support. It is the default on macOS and when building with Visual Studio. On other platforms, it can be enabled at build-time using `-DGMX_GPU_FFT_LIBRARY=VKFFT`.

Issue 4052

### PME GPU offload on macOS

Until now, PME calculations could not be offloaded to the GPU on macOS. They required the clFFT library, which silently crashed Apple's OpenCL drivers at runtime. To overcome this incompatibility, we replaced the clFFT backend with VkFFT on macOS.

### Increase of required versions

- GCC required version is now 9.
- oneMKL required version is now 2021.3.

### Miscellaneous

### Fix documentation issues for restricted bending potential

The actual implementation in the code was correct, but the manual section for the restricted bending potential had an extra factor 2 for the force inherited from the 2013 Bulacu JCTC paper, and the journals for two of the references had been swapped. No changes to any simulation results.

Issue 4568

### AWH friction metric is shared between AWH walkers

The friction metric now uses data from all walkers sharing the bias. In the AWH output only the shared friction output is written.

Issue 3842

#### Required `gmx grompp -maxwarn` to take a positive integer

Previously -1 was accepted and suppressed all warnings. Now a positive integer must be chosen.

## 11.7 GROMACS 2022 series

### 11.7.1 Patch releases

#### GROMACS 2022.6 release notes

This version was released on July 11th, 2023. These release notes document the changes that have taken place in GROMACS since the previous 2022.5 version, to fix known issues. It also incorporates all fixes made in version 2021.7 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Fix CUDA PME spread in multi-GPU runs (with >=3 GPUs)

Due to incorrect indexing of the second grid, spread could give incorrect with when separate PME ranks are used and pipelining is enabled. Runs with >=3 GPUs with direct GPU communication enabled (`GMX_ENABLE_DIRECT_GPU_COMM` env. var.) are affected.

Issue 4732

#### Fix missing synchronization in GPU PME pipelining

A missing synchronization could have caused incorrect long-range PME electrostatics forces/energies to be produced when GPU PME pipelining was used.

Only runs with >=3 GPUs and with direct GPU communication enabled (`GMX_ENABLE_DIRECT_GPU_COMM` env. var.) are affected.

Issue 4734

#### Fixes for `gmx` tools

#### Fixes that affect portability

#### Miscellaneous

#### GROMACS 2022.5 release notes

This version was released on February 3rd, 2023. These release notes document the changes that have taken place in GROMACS since the previous 2022.4 version, to fix known issues. It also incorporates all fixes made in version 2021.7 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fix perturbed exclusion handling across PBC in small systems

Systems with up to a few hundred atoms where molecules with excluded perturbed atoms were split over periodic boundary conditions could have double counted Coulomb and LJ-PME interactions over long distances. This led to very large errors in the energy and forces with PME and reaction-field, which likely did not go unnoticed. With a plain Coulomb cut-off the error is small and might not be noticed.

Issue 4665

#### Add missing net charge term when running PME on a GPU

When PME was running on a GPU, the term due to a net charge of the system was missing. In normal runs this only changed the potential energy by a constant, which is usually not relevant. In free-energy calculations where the net charge of the systen changes, the would lead to incorrect dV/dlambda and Delta lambda values (but one should anyhow avoid changing the net charge of a system with free-energy calculations).

Issue 4668

#### Foreign energy differences with the Gapsys soft-core function were zero

This caused all BAR and AWH non-bonded free-energy output to be zero, so it is unlikely that incorrect results would go unnoticed.

Issue 4705

#### Fix mdrun -rerun energy output with update groups

In the, unlikely, case of using mdrun -rerun with domain decomposition or GPU on a trajectory produced without domain decompostion and GPU, the reported energies could be incorrect when update groups were used. Update groups are typically used with biomolecular force fields with h-bonds constrained only and domain decomposition on GPU.

Issue 5067

#### Fix checkpointing of expanded ensemble simulations with modular simulator

Expanded ensemble simulations would fail to write checkpoint files when the modular simulator was used (which is the default setting for expanded ensemble in GROMACS 2022). The investigation lead to the discovery of another bug, which was also fixed: When restarting from checkpoint, a successful MC step in lambda space that had happened on the step on which the checkpoint was written would have been ignored. Since the checkpoint failed to be written in the first place, this is unlikely to have lead to wrong results in GROMACS 2022.

The bug dropping successful MC steps on checkpoint steps upon simulation restarts is also present in the legacy code path, which was the default in GROMACS 2021.7 and earlier. Simulations using the legacy code path no longer write checkpoint files, and inform about this behavior in their log file.

Issue 4629

### Fixes for `gmx` tools

### Fixes that affect portability

### Miscellaneous

### Improve muParser detection and bumped internal version to v2.3.4

Update internal muParser version that include all changes made by us. Use muParser's CMake config to detect external muParser. Update required version for external muParser to match the internal version.

Issue 4614

### GROMACS 2022.4 release notes

This version was released on November 16th, 2022. These release notes document the changes that have taken place in GROMACS since the previous 2022.3 version, to fix known issues. It also incorporates all fixes made in version 2021.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Incorrect foreign energy differences for 1-4 interactions on a GPU

When running free-energy calculations using a GPU without domain decomposition, the foreign energy differences for 1-4 interactions would be incorrect when the atoms involved had only charges perturbed and not the atom types. This issue did not affect free-energy calculations using `couple-moltype`. This issue did not affect dV/dlambda. If you have used BAR or AWH for free-energy calculations with atoms with charge manually perturbed in the topoogy and non-perturbed atom type, we suggest to rerun those with the corrected code to check if your results were affected by this issue.

Issue 4616

### The deform option caused parallel mdrun to exit

With the deform mdp option mdrun would exit with an MPI error when invoked with more than one MPI rank.

Issue 4604

### Pulling with average force output made mdrun exit at checkpoint writing

When `pull_fout_average` was set to `yes`, *gmx mdrun* (page 258) would exit with an assertion failure when trying to write a checkpoint file.

Issue 4636

### AMD RDNA devices are now properly marked as "unsupported" with OpenCL

AMD RDNA GPUs (Radeon RX 5000, 6000, and 7000 series) never worked correctly with OpenCL, usually with a simulation crashing quickly. We now properly mark those devices as incompatible.

AMD GCN (e.g., RX Vega 64) and CDNA/CDNA2 (e.g., Instinct MI100) devices are supported.

Issue 4521

## Fixes for `gmx` tools

### Programs reading a tpr file of a large system would exit with random errors

The tpr file writing and reading code contained a bug that caused reading of systems with more than 100 million atoms to exit with random error messages.

Issue 4628

### With flexible constraint grompp and mdrun exited with an assertion failure

Issue 4605

### Corrected gmx awh friction metric legend

The output of friction metric with *gmx awh* (page 160) -more is the sqrt of the metric while `sqrt` was missing from the legend. This has been added now. Note that the output of *gmx awh* (page 160) -fric, correctly, does not involve sqrt.

Issue 4598

## Fixes that affect portability

## Miscellaneous

### Fixed argument checking with nvcc

Incorrect configure-time checks of flags passed to nvcc resulted in several performance-related flags never being used. Simulations using Nvidia GPUs were correct but may not have achieved best possible performance.

### Added compilation support for new CUDA architectures

The list of NVIDIA CUDA architectures for which code is directly generated for, with a default cmake configuration, and when support exists in the compiler in use, has been updated to include the latest Ada Lovelace and Hopper architectures.

## GROMACS 2022.3 release notes

This version was released on September 2nd, 2022. These release notes document the changes that have taken place in GROMACS since the previous 2022.2 version, to fix known issues. It also incorporates all fixes made in version 2021.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Energy minimization would not converge with GPU and without DD

The steepest descent and conjugate gradient minimizers would not converge when using a GPU for the nonbonded interactions and not using domain decomposition.

Issue 4533

#### Settings of lambda-only replica exchange is mistakenly routed to temperature+lambda branch

When replica exchange of lambda-only mode is enabled, all reference temperatures ref-t are the same among replicas. However, another branch temperature+lambda is used according to log messages.

Issue 4580

### Fixes for `gmx` tools

#### Fix pdb2gmx vsite assignment for tetrahedral secondary amines

The pdb2gmx vsite assignment code would always select a planar vsite for structures like secondary amines based solely on the number of atoms, without taking geometry into account. Changed to take geometry into account.

Issue 4573

### Disallow C-rescale barostat without temperature coupling

The C-rescale barostat needs a reference temperature which is currently taken from the thermostat. A check is added to grompp for temperature coupling or BD/SD being present. Also a warning is generated when the reference temperature are not all equal.

Issue 4495

### Prevent hbond merging in gmx hbond for non-compatible options

The option to merge multiple hbonds from one donor-acceptor pair does not work with other options that rely on analysing all hydrogen bonds, or changing the search for donor-acceptor pairs. Due to this, the tool will stop if known incompatible combinations are used.

When using `-hbn` to export the hydrogen bonding information with merging enabled, previously indices of arbitrary hydrogens were printed. Now, to avoid confusion, the index file contains `-1` instead. Use `-nomerge` option if full hydrogen bond information is needed.

Issue 3837

### Fix generation of duplicate groups in index generation

Both *gmx make_ndx* (page 255) and *gmx select* (page 312) would duplicate index groups for molecules not defined as Protein or Nucleotides due to a logic error in the code, where any extra group defined in the system would cause the detection to be run again and generate the duplicate groups.

Issue 4524

### Fixes that affect portability

### Further fix to nvcc flag detection

Version 2022.1 fixed a nvcc flag detection issue for gcc version 11, but this fix resulted in issues for older gcc versions. Version 2022.2 resolved this for gcc version 7, but issues remained for other gcc versions. This release provides a fix that should prevent this issue for all gcc versions.

Issue 4539

### Document incompatibility of gcc-11 and nvcc 11.6.1

A known incompatibility between the default gcc and nvcc compiler on Ubuntu 22.04 has been documented, with a guide how to circumvent the issue.

Issue 4574

### Miscellaneous

#### Improve mdrun log file energy output formatting

The mdrun log file now prints more energy-field names in abbreviated forms so as to avoid exceeding the maximum column lengths available.

#### Improve warning message about AVX_128_FMA SIMD

This instruction set only works on early AMD CPUs that support their FMA4 extensions. Since Zen1, AMD has instead moved to support FMA3 (similar to Intel hardware), which unfortunately means SIMD instruction sets are NOT incremental. We now detect this for parallel runs and stick to vanilla 256-bit AVX, and properly detect when the FMA4 instruction set is used on non-compatible hardware and warn why the run is likely to crash.

Issue 4526

#### Remove unnecessary memory re-allocations for GPU update runs

An issue has been fixed where GPU memory allocations were repeatedly being performed unecessarily for runs with GPU update enabled, significantly impacting performance in some cases. The memory allocations are now only performed when necessary.

### GROMACS 2022.2 release notes

This version was released on June 16th, 2022. These release notes document the changes that have taken place in GROMACS since the previous 2022.1 version, to fix known issues. It also incorporates all fixes made in version 2021.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fix incorrect constraining in highly parallel runs

With domain decomposition, OpenMP and connected constraints (so not only bonds with hydrogens constrained), constraint correction could be applied when a domain would not have any constraints while it had constraints before. This is unlikely to have gone unnoticed in longer runs, as the chance is high that atoms would clash and the system would become unstable. But short runs might not crash and could therefore produce incorrect results. The correctness can be judged from the drift in the conserved energy quantity, which is reported at the end of the log file, and which will be one to two orders of magnitude larger than for a correct run.

Issue 4476

### Fix missing CPU-GPU synchronization when doing free-energy calculations

When GPU halo exchange with direct communication is enabled, CPU based free-energy kernels were being run without waiting on non-local co-ordinates to be available on host. This resulted in use of stale data on CPU side. This issue can cause incorect output when GPU direct communication is enabled using GMX_ENABLE_DIRECT_GPU_COMM environment variable and simulation contains free-energy calculations.

Issue 4471

### Fix missing PME mesh dV/dlambda with PME on GPU on a seperate PME rank

When doing free-energy calculations with PME running on GPU on a separate PME rank, the dV/dlambda contribution for the PME mesh part was missing. The same contribution was missing from the foreign lambda energy differences. Note that the energies and forces were correct.

Issue 4474

### Removed the (incorrect) output file size note/warning with mdrun -rerun

Issue 4484

### Wait for PME coordinate padding clearing to complete after reinitialization

As part of the irregular reinitialization of PME coordinate buffers, the padding area of the buffer on the GPU must be set to zero. Previously, a dependency was missing such that, with GPU direct communications enabled via the GMX_ENABLE_DIRECT_GPU_COMM, it was possible that the PME kernel could preceed before this initialization had completed, and this has now been fixed. The incorrect ordering is only expected to have occured in extreme benchmarking cases, causing an obvious crash.

Issue 4482

### Note known issues with Verlet buffer estimation

There are known issues with the Verlet buffer estimate for imhonogeneous systems as well as for potentials with only the repulsive part of the Lennard Jones potential. These issues and workarounds are listed in the *known issues* (page 34) section.

Issue 4509

### Fixes for `gmx` tools

#### Clarified pdb2gmx error message for undefined atom type in terminal database

Issue 4481

#### Lower severity of grompp exclusion distance issues with energy minimization

With energy minimization, the error/warning about pair distances beyond/close to the cut-off has been changed to a warning/note, as energy minimization might fix such issues.

Issue 4480

#### Fixed missing cmap torsion correction for periodic pepdites

When using pdb2gmx with periodic peptides and the CHARMM27 force field, the CMAP correction across the periodic boundary was missing (but not the torsion itself). Reprocess your PDB file with pdb2gmx from version 2022.2 or later to get a correct topology, or add it manually to the topology.

#### Avoid crash in gmx bar with invalid input

gmx bar could try to read an invalid input data file and would fail with a hard crash instead of a helpful error message.

#### Fix printing of incorrect data from analysis tools

A change on how we handle selections caused the analysis tools in the trajectoryanalysis framework to print data for wrong selections.

Issue 4508

#### Remove broken charge zeroing function in convert-tpr

This had been broken for a long time and no discernable use.

Issue 4226

### Fixes that affect portability

#### Warn when using gcc version 7 with CUDA builds

Different versions of gcc 7 behave differently in a way that makes it hard for GROMACS to check whether CUDA's `nvcc` compiler will accept compiler flags. GROMACS 2022 and 2022.1 sometimes would spuriously detect flags as invalid, avoid using them, and so produce slow CUDA kernels. Now GROMACS assumes all nvcc flags are valid in this case and the build system warns when this is occurring. If you then experience build failures, please use a newer version of gcc.

Issue 4478

### Fix external tinyXML version to below 7

Newer versions are not compatible with GROMACS.

Issue 4477

### Fixed a possible software build error with OpenMP linking

In some situations, software builds could have errors linking `omp` symbols. A minor update to the CMake configuration helps the `muparser` component to find the same OpenMP dependency used by the rest of the library.

Issue 4499

### Miscellaneous

### Fixed detection of external TinyXML-2

Updated the code to properly detect the presence and the version of external TinyXML-2 (only relevant if `-DGMX_EXTERNAL_TINYXML2=ON` is used).

Issue 4477

### Fixed warnings when using module-specific OpenMP thread-count environment variables

One of the arrays used to construct message strings wasn't updated properly, so sometimes an information message was wrong or may have printed garbage.

### GROMACS 2022.1 release notes

This version was released on April 22th, 2022. These release notes document the changes that have taken place in GROMACS since the previous 2022 version, to fix known issues. It also incorporates all fixes made in version 2021.5 and earlier, which you can find described in the *Release notes* (page 838).

### Note to developers and package maintainers

### Next release (GROMACS 2022.2) will rename master branch to main

At the date of the next release we will rename the master branch to main to moving away from master / slave terminology.

After the GROMACS 2022.2 patch release, developers are advised to delete their local master branch and fetch the remote main branch as in `git branch -d master; git fetch; git checkout main`

### Fixes where mdrun could behave incorrectly

### Fixed incorrect pairlist buffer with test particle insertion

With TPI the pairlist cut-off did not take into account rtpi and the radius of the molecule to insert.

Issue 4458

### Remove false positives for missing exclusions in free energy kernels

Free energy calculations good stop with a fatal error stating that excluded atoms pairs were beyond the pairlist cut-off while this actually was not the case.

Issue 4321

### Fix crash when steering FEP with AWH without PME or with separate PME rank

There would be a segfault when deciding whether early PME results are needed.

Issue 4413

### Fix bug with reporting energies for groups

When different molecules in a molecule block in the topology use different energy group assignments for atoms, the energy group assignment of the first molecule would be repeated for all other molecules in the block. Note that the reported energies for the whole system were correct.

Issue 4462

### Fix missing B State pinning for PME GPU

The PME memory is now correctly pinned when using GPU PME.

Issue 4408

### Only allow 1D PME GPU decomposition

Due to correctness issues in PME grid reduction with 1D decomposition, this feature could produce incorrect results. This would however in most real-world cases be masked by an overly large halo size. 0D decomposition cases are unaffected and only such setups will be allowed in the current release (0D PME decomposition can be forced using the GMX_PMEONEDD env var).

### Fixed exact continuation with the -reprod option

With the leap-frog integrator, kinetic energy terms were often not stored in the checkpoint file. This caused minor difference in the computed kinetic energy (due to different operation order), which could cause a run continued from checkpoint to diverge from a run without interuption, even when using the -reprod option.

Issue 4240

### Fixes for `gmx` tools

### Use correct scattering length for hydrogens in `gmx sans`

The floating-point comparison was always false, leading to all atoms with atomic number 1 having scattering length of deuterium (6.6710 fm) instead of -3.7406 fm for plain hydrogens.

### Fix C-terminal residue patch for charmm

One of the atom types names in the Charmm27 force field C-terminal COOH patch was incorrect, and would have triggered a crash or error in pdb2gmx, which was identified when Fedora ran our unit tests with additional checking flags.

Issue 4414

### Add polyproline helix coloring to DSSP maps

DSSP-4.0 can detect polyproline type-2 helices, so we now also have a dark blue-green color entry for this in the generated maps.

Issue 4410

### Remove option -unsat from gmx order and document deficiencies

This hasn't properly worked since it was added.

Issue 1166

### Fix g96 file writing

The g96 file writing could violate the file format when residue or atom names got longer than 5 characters.

Issue 4456

### Rerun will no longer abort when encountering too high forces

Issue 4352

### Allow incomplete index files for extract-cluster

Issue 4420

### Fixes that affect portability

### Fix nvcc flag detection

Issue 4415

### Fix issue in GMXRC.bash

Issue 4450

### Miscellaneous

### Fixed regression test download URL for forks of GROMACS

Users of forks of GROMACS (eg PLUMED) can now also use the feature to download the regression tests automatically.

### Fix internal nblib test failure

The nblib internal tests used incorrect indices, which triggered a crash when Fedora ran our unit tests with additional checking flags. This will not have influenced any actual clients merely using nblib.

Issue 4414

### Workaround for nested MPI-aware code

*gmxapi* (page 718) scripts containing `gmxapi.commandline_operation` tasks could be unusable if a task executable automatically detects MPI resources and the script is invoked with an MPI launcher.

The workaround is to increase the isolation of the task environment from the parent process by explicitly setting the task environment variables. This is now possible with a new *env* key word argument to *commandline_operation()* (page 718), which is simply passed along to `subprocess.run`.

Issue 4421

### Accurately checks when FEP lambda might go over 1 or under 0

The checks that verify that the FEP lambda does not go out of range used to trigger incorrectly when the delta-lambda and number of step was exactly right.

Issue 4442

### Correct free-energy (de)coupling integrator check

With free-energy (de)coupling calculations, grompp would only warn with the md integrator that sd should be used. Now this warning is extended to the md-vv integrators.

### Density-guided simulation affine transformation force correction

Forces were not calculated correctly when using affine transformations with density-guided-simulation-transformation-matrix, e.g., rotations and projections of structures, before calculating forces for density guided simulations.

The reason for this error was the missing multiplication with the transpose of the affine transformation matrix. This is needed to account for the coordinate transformation when calculating the force as the derivative of the energy, according to the chain rule of calculus.

Affects simulations where density-guided-simulation-transformation-matrix is set and not trivial. If the matrices were diagonal, forces were wrongly scaled. If a rotation matrix was set, the effect was a mis-rotation of forces, leading to an overall undesired torque on the structure.

Issue 4455

### Clarified Coulomb self terms in the reference manual

Issue 4451

### Correct formula for SD integrator

The formula in the reference manual was different from the implementation, even though both have been mathematically equivalent.

### Adjust test tolerances for double precision testing

Some tests could fail on different hardware when using double precision builds due to too strict tolerances. This mainly affected test simulations that could diverge due to the limited precision of some SIMD instructions (44 bits when using invsqrt).

Issue 4414

### 11.7.2 Major release

#### Highlights

GROMACS 2022 was released on February 22st, 2022. Patch releases may have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

As always, we've got several useful performance improvements, with or without GPUs, all enabled and automated by default. In addition, several new features are available for running simulations. We are extremely interested in your feedback on how well the new release works on your simulations and hardware. The new features are:

- Free-energy kernels are accelerated using SIMD, which make free-energy calculations up to three times as fast when using GPUs
- A new formulation of the soft-cored non-bonded interactions for free-energy calculations allows for a finer control of the alchemical transformation pathways
- New transformation pull coordinate allows arbitrary mathematical transformations of one of more other pull coordinates
- New interface for multi-scale Quantum Mechanics / Molecular Mechanics (QM/MM) simulations with the CP2K quantum chemistry package, supporting periodic boundary conditions.
- grompp performance improvements
- Cool quotes music playlist
- Additional features were ported to modular simulator
- Added AMD GPU support with SYCL via AdaptiveCpp (hipSYCL)
- More GPU offload features supported with SYCL (PME, GPU update).
- Improved parallelization with GPU-accelerated runs using CUDA and extended GPU direct communication to support multi-node simulation using CUDA-aware MPI.

#### New and improved features

#### Hybrid Quantum-Classical simulations (QM/MM) with CP2K interface

Simulations of chemical reactions pathways can provide an atomistic insight into many biological and chemical processes. To perform such kind of modelling in complex systems, that includes solvent and/or proteins Multi-scale Quantum Mechanics / Molecular Mechanics (QM/MM) approaches are often used. Here we introduce a whole new interface to perform QM/MM simulations in fully periodic systems using MDModule that couples GROMACS with CP2K quantum chemistry package. This enables hybrid simulations of systems in systems where chemical reactions occurs. The interface supports most of the simulations techniques available in GROMACS including energy minimization, classical MD and enhanced sampling methods such as umbrella sampling and accelerated weight histogram method.

### Transformation pull coordinate for mathematical transformations of pull coordinates

A new pull coordinate type named transformation has been added. This enables mathematical transformation of previously defined pull coordinates using a user supplied formula in a string. This allows for example non-linear transformation of a distance, e.g. a contact coordinate or (non-)linear combinations of multiple pull coordinates. This is a powerful tool for defining complex reaction coordinates and it can be combined with the Accelerated Weight Histogram Method to enhance sampling.

### Replica-exchange molecular dynamics simulations with GPU update

Replica-exchange molecular dynamics now works with GPU update.

### A new formulation of soft-core interactions for free energy calculations

With this addition GROMACS allows to choose from two schemes to soften non-bonded interactions during alchemical perturbations: Beutler *et al.*[100] (page 688) and Gapsys *et al.*[183] (page 692) soft-core functions.

### More flexible sharing of biases in AWH

With the accelerated weight histogram method, biases can now be shared between subsets of all simulations, without restrictions. The allows for more flexible ensemble simulation setups, as well as simpler launches of sets of simulations.

### More features implemented in modular simulator

Several features were added to the modular simulator, including all temperature and pressure coupling algorithms available in the legacy simulator, expanded ensemble and pull.

### Free energy calculations now support all non-perturbed bonded interactions

Previously GROMACS did not permit any usage of a few more special bonded interactions (restricted angles/dihedrals or combined bending-torsion potentials) in free energy calculations. These are now allowed, as long as the interaction itself is not perturbed.

Issue 3691

### Adapt number of threads to actually permitted hardware

Previously, GROMACS would attempt to start as many threads as there are processors in the system, and try to pin threads on processing units. This would fail whenever we are not allowed to use all those processors, e.g. when Slurm only provides part of a node to a job, or on A64fx where some processors are reserved for the system. We would also start far too many threads in container environments. As part of improved hardware detection, we now only detect processors on which we are allowed to run, and adapt the number of threads whenever there is a cpu limit set, which will improve performance both

for containers and make GROMACS do the right thing when Slurm or other queue systems allocate part of a node.

### Enable use of more OpenMP threads

The thread-force-reduction code in GROMACS will now allow up to 128 OpenMP threads by default, and we have changed the internal logic so we just limit the number of threads rather than refuse to run. This only applies within each rank; you can use an unlimited number of threads by combining OpenMP threading with multiple ranks. For large machines with many cores this is usually faster since the domain decomposition used with multiple ranks is better adapted to non-uniform memory access hardware.

Issue 4370

### Centering and symmetrization supported in gmx potential

gmx potential now supports the same centering and symmetrization options as gmx density, which is particularly useful for membranes.

Issue 3579

### Performance improvements

### GPU direct communication with CUDA-aware MPI

Direct GPU communication support has been extended to simulations that use a CUDA-aware library-MPI when running on NVIDIA GPUs. Detection of CUDA-aware MPI is performed both at cmake-time and runtime. The feature has been tested primarily with OpenMPI but any CUDA-aware MPI implementation should be suitable, and it is also possible to use with the thread-MPI implementation in GROMACS. CUDA-aware MPI support still lacks substantial testing, hence it is included in the current release as a development feature and should be used with caution. Hence, even if a suitable MPI is detected, direct communication is not used by default, but it can be enabled using the GMX_-ENABLE_DIRECT_GPU_COMM environment variable.

Issue 3960 Issue 2915

### Dynamic pairlist generation for energy minimization

With energy minimization, the pairlist, and domain decomposition when running in parallel, is now performed when at least one atom has moved more than the half the pairlist buffer size. The pairlist used to be constructed every step.

### Nonbonded free-energy kernels use SIMD

Free energy calculation performance is improved by making the nonbonded free-energy kernels SIMD accelerated. On AVX2-256 these kernels are 4 to 8 times as fast. This should give a noticeable speed-up for most systems, especially if the perturbed interaction calculations were a bottleneck. This is particularly the case when using GPUs, where the performance improvement of free-energy runs is up to a factor of 3.

Issue 2875 Issue 742

### PME-PP GPU Direct Communication Pipelining

For multi-GPU runs with direct PME-PP GPU communication enabled, the PME rank can now pipeline the coordinate transfers with computation in the PME Spread and Spline kernel (where the coordinates are consumed). The data from each transfer is handled separately, allowing computation and communication to be overlapped. This is expected to have most benefit on systems where hardware communication interfaces are shared between multiple GPUs, e.g. PCIe within multi-GPU servers or Infiniband across multiple nodes.

Issue 3969

### Domain decomposition with single MPI rank

When running with a single MPI rank with PME and without GPU, mdrun will now use the domain decomposition machinery to reorder particles. This can improve performance, especially for large systems. This behavior can be controlled with the environment variable GMX_DD_SINGLE_RANK.

### Restricted GPU support with multiple time stepping

GPUs can be used in combination with MTS, but for now this is limited to the setup where only the long-range nonbonded force is applied in longer timesteps (and computed on the CPU), while all other components are are calculated every step (which can be on the GPU).

### `gmx grompp` now runs 20-50% faster

After a series of improvements, the loops in the parameter- and atom-lookup code in `gmx grompp` have been transformed to run faster while using simpler, standard code idioms.

### PME decomposition support in mixed mode with CUDA and process-MPI

PME decomposition is supported now in mixed mode with CUDA backend. This is supported only if GROMACS is compiled with external process-MPI and underlying MPI implementation is CUDA-aware. This feature lacks substantial testing and has been disabled by default but can be enabled by setting GMX_GPU_PME_DECOMPOSITION=1 environment variable.

### Performance improvements when running on Ampere-class Nvidia GPUs

Improved performance of the short-ranged non-bonded kernels by up to 12%.

Issue 3872

## Changes to the API

### Remove physical constant conversion functions

Legacy conversion functions for physical constants from and to the GROMACS representation have been removed as they didn't see any use in the library.

## Improvements to GROMACS tools

### `gmx msd` has been migrated to the trajectoryanalysis framework

The tool now uses the GROMACS selection syntax. Rather than piping selections via stdin, selections are now made using the "-sel" option. There is a new option called `-maxtau`, which limits maximum time delta between frames to compare for calculating MSDs. This will allow users who otherwise would run into out-of-memory errors and slow execution with large systems to restrict sampling to useful tau values.

This migration comes with about a 20% speedup in execution time.

Some rarely used features have yet to be migrated, including:

- The -tensor option is not yet implemented.
- System COM removal with -rmcomm has not yet been implemented.
- B-factor writing using the -pdb option is not yet supported.

A slight behavior change is the removal of the -mw option. `gmx msd` with `-mol` will take the MSD of the center-of-mass of of molecules, while no mass-weighting is done when `-mol` is not selected. In previous GROMACS versions, `-mw` was on by default, and `-nomw` was silently ignored when `-mol` was chosen. This change will only cause different results when performing MSD calculations on a non-homogenous group of particles without `-mol` set.

Issue 2368

### `gmx lie` now reads energy files from reruns

This tool formerly relied on the presence of a pressure field in the .edr file, and that field will be missing if the .edr came from a rerun. However it was never necessary to rely on the presence of the pressure field, so now the tool just works correctly.

Issue 4070

### `gmx chi` no longer needs `residuetypes.dat` entries for custom residues

The need to add the names of custom residues to `residuetypes.dat` has been removed, because it served no purpose. This makes `gmx chi` easier to use.

### `gmx wham` has had minor improvements to its text output

Reporting about file handling and input-file column contents are easier to follow.

### `gmx do_dssp` supports DSSP version 4

The newer DSSP version 4 program can be used by `do_dssp` by specifying option `-ver 4` and setting the DSSP environement variable to the `mkdssp` executable path (e.g. `setenv DSSP /opt/dssp/mkdssp`)

Issue 4129

### `gmx trjconv -dump` now works reliably

The frame nearest the dump time is now always written, even if the time is before or after the range present in the trajectory file. To get the last frame of a trajectory file whose frames are in temporal order, you can request the dump of any time larger than the time of any frame in the trajectory, like `gmx trjconv -dump 9999999`.

Issue 2873

### `gmx trjconv` handles selections in TNG files better

When writing TNG files the whole system was written even if the user requested only a selection of atoms. Now only the selected atoms should be written. If the selection name matches a molecule type and the selected atoms are all present in that molecule then the molecule will be written as expected with the correct molecule count etc. If the selection only matches some atoms in a molecule or atoms from multiple molecules then the TNG file will contain a single molecule instance containing all those atoms.

Issue 2785

### `gmx pdb2gmx` no longer accepts charged glutamine (QLN) with the OPLS-AA forcefield

A torsion angle definition was missing from the (non-standard) charged glutamine residue. Grompp would use the default torsion angle instead. To avoid silent errors the charged glutamine residue was removed from the OPLS-AA forcefield.

Issue 3054

#### `gmxapi.commandline_operation` isolates working directories.

Subprocesses launched for wrapped command line operations now run in unique subdirectories. Users who rely on the *output_files* input and *file* output mapping should not be affected. Users who rely on assumptions about where wrapped commands are executed will need to adjust their scripts.

The *stderr*, *stdout*, and *file* output members are still the primary supported means to access command output. Additionally, a new *directory* output gives the filesystem path that was used for the subprocess. See *gmxapi.commandline_operation()* (page 718) for details.

Issue 3130

### Bugs fixed

#### Fixed slight inaccuracies when using virtual sites with pressure coupling

Virtual sites were reconstructed after the system was propagated, but before scaling due to pressure coupling. For virtual site types which are not a linear combination of other atoms, this is not completely correct. Since the scaling due to pressure coupling is very small in healthy simulations, the resulting inaccuracies are expected to have been extremely minor, and in most cases undetectable.

Issue 3866

#### Correct dVremain/dl when nstdhdl > nstcalcenergy

When nstcalcenergy was not a multiple of nstdhdl, incorrect dVremain/dl terms were written in the energy file. Note that all dH/dl output in both dhdl.xvg and the energy file, which is used by e.g. gmx bar, was correct.

#### Removed velocity output for acceleration groups

The reported velocity in the energy file for acceleration groups was always zero. Now their velocity is no longer reported in the energy file.

Issue 1354

#### Use correct c0 parameter in Me2PO4 in OPLSAA

OPLSAA torsions must sum to 0, but the parameters for Me2PO4 did not do so. Changed the c0 parameter to the correct value.

Issue 4075

### Allow function type Fourier Dihedral with free energy perturbations

The Fourier Dihedral (dihedral interaction type 3) could not be used in free energy perturbation simulations. Under the hood the dihedral parameters were anyhow converted to Ryckaert-Bellemans parameters, so now the checks for perturbations are the same for the two functions.

Issue 2606

### Do not scale coordinates of frozen atoms during Parrinello-Rahman pressure coupling

When Parrinello-Rahman pressure coupling was used, the box scaling was applied to all the atoms, causing frozen atoms to shift. The effect is more drastic towards the sides of the box and when the pressure is changed significantly during the simulations. Now, the frozen atoms will be ignored by the coupling and atoms with frozen dimensions shall keep such values.

Issue 3075

### Avoid non-uniform rotation with Test Particle Insertion in anisotropic systems

With anisotropic systems the random angles would not get a uniform distribution.

Issue 3558

### Allow free energy calculations with a linear angle potential

Free energy calculations with a linear angle potential were not explicitly allowed by grompp.

Issue 3456

### Fixed progress display in trjconv and trjcat

The progress information (frame number and time) shown during trajectory operations in trjconv and trjcat is now correctly displayed.

Issue 4320

### Fixed GROMOS dihedral generation for disulfide bridges

The pdb2gmx functionality now generates correct dihedrals for disulfide bridges with the GROMOS force field series.

Issue 4188

### Fixed energy term naming for periodic improper dihedrals

Those used the same name internally as the non-periodic version for printing to energy files and reading from them. This could cause tools being confused when trying to compare terms from files where the terms where written in a different order.

### gmx density now always uses relative coordinates

There is no realistic use case for using absolute coordinates in binning when the box dimension is changing, so gmx density now always uses relative coordinates internally. This also avoids issues with output scaling to the last instead of average box size when users forget this option, ensures the output is always correct, and gets rid of occassional segfaults.

Issue 3830

## Deprecated functionality

## Changes anticipated to GROMACS 2022 functionality

## Functionality deprecated in GROMACS 2022

### GMX_OPENCL_NB_CLUSTER_SIZE CMake variable deprecated in favor of GMX_GPU_NB_CLUSTER_SIZE

Both OpenCL and SYCL support different cluster sizes, so GMX_GPU_NB_CLUSTER_SIZE should be used going forward.

### The built-in viewer `gmx view` will be removed

There is little use and no tests of this functionality, so it is not worth attempting to maintain moving forward.

Issue 4296

### The analysis tool `gmx chi` will be removed

This tool has not been functional for a few years. Please comment at the linked issue if you have any interest in it.

Issue 4108

### Guessing masses and atomic radii from atom names is deprecated

When atom masses or van-der-Waals radii are needed, we suggest building a proper GROMACS topology instead of using PDB files directly, even if the tool supports it.

Issue 3368 Issue 4288

### Functionality deprecated in GROMACS 2021

#### `mdrun -deffnm` to be removed

This functionality is convenient when running very simple simulations, because it permits grouping of a set of files that then differ only their suffix. However, it does not work in the wider case of an `mdrun` module (or modules) writing multiple `.xvg` output files. The resulting filenames collide. That, and its interaction with checkpointing and appending, have led to quite a few bug reports.

Because users can use a folder to group files (a standard mechanism that they understand from experience outside of GROMACS), we can build and test better software for them if we remove the erstwhile convenience of `mdrun -deffnm`. Please update your workflows accordingly.

Issue 3818

### OpenCL to be removed as a GPU framework

Issue 3818 Work is underway for ports to AMD and Intel GPUs, and it is likely that those ports will not be based on the current GROMACS OpenCL port. Nvidia GPUs are targeted by the CUDA port, and no changes are expected there. The core team can't maintain, test, and extend up to 4 ports with current resource levels. Since there are no prospects of an emerging GPU vendor in HPC needing OpenCL support, we will remove the OpenCL port once AMD and Intel support is established in other ways.

### Support for version 1 of the hardware locality library `hwloc`

Issue 3818 Version 2 has been supported in GROMACS for several years. The capabilities of newer hardware and hardware-support APIs are of most interest for GROMACS moving forward, so we should minimize our testing work and encourage clusters to upgrade older `hwloc` installations.

### Legacy API

Issue 3818 The legacy installed headers have been deprecated for a while, however we wish to state more broadly that all headers found within the `src` directory tree of GROMACS are intended for internal consumption only, and are thus subject to change without notice. Further, the form and contents of the `libgromacs` library and related CMake targets may change as we move towards building APIs and supporting machinery that can be stable and supported in the long term.

### Functionality deprecated in GROMACS 2019

#### Generation of virtual sites to replace aromatic rings in standard residues

Issue 3254 These are thought to produce artefacts under some circumstances (unpublished results), were never well tested, are not widely used, and we need to simplify pdb2gmx.

#### Benchmarking options only available with `gmx benchmark`

Issue 3255 Options such as `-confout`, `-resethway`, `-resetstep` are not intended for use by regular mdrun users, so making them only available with a dedicated tool is more clear. Also, this permits us to customize defaults for e.g. writing files at the end of a simulation part in ways that suit the respective mdrun and benchmark use cases, so `-confout` will no longer be required.

#### `gmx mdrun -nsteps`

Issue 3256 The number of simulation steps described by the .tpr file can be changed with `gmx convert-tpr`, or altered in .mdp file before the call to `gmx grompp`. The convenience of this mdrun option was outweighted by the doubtful quality of its implementation, no clear record in the log file, and lack of maintenance.

### Removed functionality

#### Removed mdrun-only build configuration

The need for the mdrun-only build of GROMACS has expired, as it has the same set of dependencies as regular GROMACS. It was deprecated in GROMACS 2021. Removing it will simplify maintenance, testing, documentation, installation, and teaching new users.

Issue 3808

#### Removed support for x86 MIC, ARMv7, Sparc64 HPC-ACE, and IBM VMX SIMD

These platforms are dead in HPC and so are no longer supported. The KNL platform is unaffected by this change.

Issue 3891

#### Removed deprecated environment variables

The following environment variables were removed after being deprecated in favor of better-named alternatives:

- `GMX_CUDA_NB_ANA_EWALD` and `GMX_OCL_NB_ANA_EWALD` (use `GMX_GPU_NB_ANA_EWALD`)
- `GMX_CUDA_NB_TAB_EWALD` and `GMX_OCL_NB_TAB_EWALD` (use `GMX_GPU_NB_TAB_EWALD`)

- GMX_CUDA_NB_EWALD_TWINCUT and GMX_OCL_NB_EWALD_TWINCUT (use GMX_GPU_NB_EWALD_TWINCUT)

Issue 3803

### Removed the ability for gmx wham to read .pdo files

Files in .pdo format were written by GROMACS versions prior to 4.0. That is so long ago that being able to read them is no longer relevant, so this capability was deprecated in version 2021. If you do need to read such files, please use an older version of GROMACS.

### Removed 32bit support

We deprecated 32bit support in 2020 and have had no way to test it ourselves for a while before that. Those architectures are no longer relevant in HPC, so we officially no longer support building GROMACS on them.

### Portability

### Intel classic compiler (icc/icpc) no longer supported

We now support the Intel clang-based compiler from oneAPI (icx/icpx) instead. Please use it, or gcc.

Issue 3893

### Provisional: Initialize GMX_INSTALL_NBLIB_API and GMXAPI build options from BUILD_SHARED_LIBS

CMake options GMXAPI and GMX_INSTALL_NBLIB_API produce shared object libraries, so their default values are now initialized from BUILD_SHARED_LIBS. Pending movement on Issue 3605 and related issues, the coupling between these options is subject to change, but users generally should not need to manually set GMXAPI and GMX_INSTALL_NBLIB_API.

Issue 4053

### Updates to pybind11 dependency

pybind11 is no longer bundled with GROMACS.

The gmxapi 0.3 Python package build system relies on PEP 517/518 build requirements to get pybind11 header dependencies through the Python packaging system. Package managers like pip will download dependencies automatically. Package managers that do not automatically fulfill dependencies should still report the missing dependency to the user.

The sample_restraint sample project (bundled in python_packaging/sample_restraint) still has a primitive CMake-only build procedure. If you fork a project from this source, you may choose to modernize the build system (similarly to that of gmxapi) or to bundle the pybind11 sources. Within the GROMACS repository, the sample_restraint option default is now GMXAPI_EXTENSION_DOWNLOAD_PYBIND=ON.

Issue 4092

### CMake toolchain file replaced with cache file

The `gromacs-toolchain.cmake` file (previously installed to `$CMAKE_INSTALL_PREFIX/share/cmake/gromacs/`) is no longer provided. Instead a partial CMake cache file is installed to `$CMAKE_INSTALL_PREFIX/share/cmake/gromacs${SUFFIX}/gromacs-hints.cmake`.

Client software may get CMake hints by configuring with `-C /path/to/gromacs-hints.cmake`, instead of forcing a cross-compiling CMake configuration with `-DCMAKE_TOOLCHAIN_FILE` or `--toolchain`.

Client software bundled with GROMACS (the gmxapi Python package) no longer requires the toolchain file. See *Full installation instructions* (page 694) for details.

Issue 4208

### Bundle muparser

GROMACS now bundles MuParser version 2.3. It is also possible to link to an external provided library.

### Miscellaneous

### grompp no longer modifies nstcomm

grompp will no longer set nstcomm, the interval for center of mass motion removal, equal to nstcalcenergy when nstcomm < nstcalcenergy. A note is still printed in that case.

### Bonded atom types names can now start with a digit

Bonded atom types names in topologies were not allowed to start with a number. Now all names are supported that contain at least one non-digit character.

Issue 4120

### grompp now warns when exclusion forces might be missing

When using PME, exclusions between non-perturbed,atom pairs should be within the cut-off distance, otherwise mdrun might not compute grid correction forces and energies. grompp now computes these distance for the starting structure and warns when they are beyond 90% of the cut-off distance and generates an error when they are beyond the cut-off distance.

Issue 4051

### The AWH cover diameter for angles now has units degrees

Using old tpr files that apply AWH to angles or dihedrals and have a non-zero cover diameter results in an error with the suggestion to regenerate the tpr file.

Issue 4367

### Core spin-up code is removed

Formerly, on non-x86 and non-PowerPC platforms, mdrun ran some multi-threaded code to try to wake up any cores that the OS might have powered down. This caused problems on some Arm platforms, and does not seem to suit a significant number of platforms for use of GROMACS. So now it is removed.

If required, please manually spin-up the cores with, e.g., `stress --cpu $(nproc --all)`.

Issue 4074

### Add documentation for linear angle potential

Added documentation and reference for the linear angle potential. Also added please_cite entry, but there is no call to reference it yet.

Issue 4286

### gmxapi.mdrun guarantees trajectory output

gmxapi simulations now always run with full-precision trajectory output (`-o`) in order to guarantee the availability of a usable output trajectory through the `mdrun.output.trajectory` result.

Issue 4285

### gmxapi.mdrun accepts arbitrary runtime arguments

Arbitrary mdrun arguments can be passed through gmxapi with the new *runtime_args* key word argument, accepting a dictionary of flags and values.

Issue 4284

### Improved MPI awareness and task uniqueness for gmxapi Python runner

Previously, only the Python components in *gmxapi.simulation* (page 722) reacted to the presence of an MPI context. This could result in duplicate work or even invalid file access.

*gmxapi.commandline_operation()* (page 718) now executes tasks in unique working directories.

For all gmxapi operations, tasks are only launched from one process (per ensemble member). If mpi4py is available, the MPI environment is inspected. If multiple ranks are discovered, the `ResourceManager` instances on the various ranks coordinate to make sure that `update` is

only called for each member of each task once. Results are broadcast to all ranks from the `ResourceManager` where the work occurred.

These changes merely constitute a bug-fix. Additional development is needed for more optimal use of resources and to reduce unnecessary data transfers.

Issue 3138

#### Further discouraged use of Berendsen coupling algorithms

Those algorithms have been proven to cause incorrect sampling of their respective distributions and are mainly provided as a means to provide backwards compatibility for older simulations. This is why their use has been further discouraged by changing the current notes about their use to actual warnings at grompp time.

## 11.8 GROMACS 2021 series

### 11.8.1 Patch releases

#### GROMACS 2021.7 release notes

This version was released on January 31st, 2023. These release notes document the changes that have taken place in GROMACS since the previous 2021.6 version, to fix known issues. It also incorporates all fixes made in version 2020.7 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Add missing net charge term when running PME on a GPU

When PME was running on a GPU, the term due to a net charge of the system was missing. In normal runs this only changed the potential energy by a constant, which is usually not relevant. In free-energy calculations where the net charge of the systen changes, the would lead to incorrect dV/dlambda and Delta lambda values (but one should anyhow avoid changing the net charge of a system with free-energy calculations).

Issue 4668

#### Fixes for `gmx` tools

#### Fixes that affect portability

#### Miscellaneous

#### GROMACS 2021.6 release notes

This version was released on July 8th, 2022. These release notes document the changes that have taken place in GROMACS since the previous 2021.5 version, to fix known issues. It also incorporates all fixes made in version 2020.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Extend error message for free-energy exclusion beyond rlist

With free-energy decoupling simulations an error for exclusions beyond rlist can occur due to a too small box. This cause is now added to the error message.

Issue 3403 Issue 3808

### Fix running with LJ PME only

Simulations with only LJ PME but not electrostatic PME would fail to run due to an error in task assignment.

Issue 4362

### Fix missing synchronization in CUDA update kernels

When using GPU update with SETTLE or LINCS constraints, virial calculations could have been incorrect on Volta and newer NVIDIA GPUs, which in turn would lead to incorrect pressure. The GPU update is not enabled by default, so the error can only appear in simulations where it was manually selected, and even in this case the error might be rare since we have not observed it in practice in the testing we have performed.

To check whether your runs could have been affected, please examine your mdrun log file:

- Look for the line "GPU support: CUDA";
- Look for the line "PP task will update and constrain coordinates on the GPU";
- Check whether any GPU the value of "compute cap." 7.0 or higher in the "GPU Info:" section.

If all three are present, than the bug could have perturbed the virial calculation and, in turn, led to incorrect pressure coupling. All GROMACS version prior to 2021.6 and 2022.0 that allow offloading of the update and constraint calculations to GPUs are affected.

Issue 4393

### Fixes for `gmx` tools

### Do not try to guess atom names in `gmx rms` unless needed

Guessing atomic masses based on atom names may sometimes fail. When `-nomw` switch is used, atom masses are not needed, but `gmx rms` was trying to guess them anyway, throwing a fatal error when an unknown element was encountered. Now, the error is only raised when masses are actually needed.

Issue 4356

### Fixes that affect portability

### Miscellaneous

### Corrects units for AWH interval in the user guide

When applying AWH to angles or dihedrals, the units of bonds of the sampling interval listed in the mdp section of the user guide are now stated to be in degrees. The guide incorrectly stated that there were in radians, whereas the code interprets the user input as degrees.

Issue 4367

### Fix distance restraint force calculation in case of negative prefactor

When calculating distance restraint forces, the quadratic regime for weak restraint violation and the linear regime for strong restraint violation were interchanged in case of a negative force constant.

Issue 4347

### GROMACS 2021.5 release notes

This version was released on January 14th, 2022. These release notes document the changes that have taken place in GROMACS since the previous 2021.4 version, to fix known issues. It also incorporates all fixes made in version 2020.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Do not scale coordinates of frozen atoms during Parrinello-Rahman pressure coupling

When Parrinello-Rahman pressure coupling was used, the box scaling was applied to all the atoms, causing frozen atoms to shift. The effect is more drastic towards the sides of the box and when the pressure is changed significantly during the simulations. Now, the frozen atoms will be ignored by the coupling and atoms with frozen dimensions shall keep such values.

Issue 3075

### Properly account for DeltaH contribution from PME when running AWH with FEP

The linear dHdL contribution from PME, when PME was calculated on GPU or on a separate PME rank, was calculated too late to be taken into account for FEP steered by AWH. Please verify your simulation results from simulations running FEP steered by AWH with PME on GPU or using a separate PME rank.

Issue 4294

### Fix reading of AWH user PMF reading with large PMF values

The reading of user supplied AWH input in mdrun with PMF values larger than 88 kT would cause mdrun to exit with an assertion failure. Now values up to 700 kT are allowed and exceeding those causes an exit with a clear error message.

Issue 4299

### Fixes for `gmx` tools

#### `gmx make_edi` now closes its output file properly

Previously the file was not closed explicitly, leaving the result up to the runtime environment. Now it will work with all environments.

### Out-of-bounds, overflow and incorrect outputs fixes in `gmx spatial`

There were several issues with memory management in `gmx spatial`, which were addressed: 1. Out-of-bound memory writes. 2. Confusing error message when the coordinate is exactly on the boundary (happens with .xtc files). 3. Norm could become negative due to integer overflow. 4. Having negative `-ign` (default `-1`) led to incorrect number of grid points 5. The coordinates of the grid points were incorrect especially when `-ign` is non-zero. 6. Norm calculation was incorrect. 7. Default `-nab` value is increased from 4 to 16.

Issue 3214

### Fixes that affect portability

### Miscellaneous

### Performance improvements when running on Ampere-class Nvidia GPUs

Improved performance of the short-ranged non-bonded kernels by up to 12%.

Issue 3873

## GROMACS 2021.4 release notes

This version was released on November 5th, 2021. These release notes document the changes that have taken place in GROMACS since the previous 2021.3 version, to fix known issues. It also incorporates all fixes made in version 2020.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fixed crash for large system with virtual sites

When large system with virtual sites were ran with domain decomposition and OpenMP threading, mdrun would crash when the number of atoms in a domain and its halo were more than 200000.

Issue 4167

#### Fixed bug with GPU LINCS occasionally shifting atoms in wrong direction

Due to missing blocking synchronizations in the CUDA version of LINCS, the shared memory was occasionally overwritten with the new data. This may slightly affect the final coordinates of the shifted atoms.

Issue 4199

#### Disabled the use of PME Mixed mode for FEP simulations

The use of Mixed mode PME (`-pme gpu -pmefft cpu`) led to incorrect computation of $\frac{\partial V}{\partial \lambda}$ in FEP simulations.

Mixed mode is only used when explicitly requested by the user.

Issue 4190

#### Fixed spurious nan in AWH free energy output when running FEP with other dimensions

When running AWH with alchemical free energy perturbations as one of multiple dimensions the free energy output could contain nan entries due to failing log operations. This did not affect the AWH bias, which means that the simulations were not affected as such, but the output was.

Issue 4180

#### Made mdrun work without MPI

When configured with neither of MPI or thread-MPI, mdrun would terminate with an assertion failure.

Issue 4264

### Fixes for `gmx` tools

#### Fix `gmx convert-tpr -s -o`

Formerly, this combination could be used when supplying an index file. Now this combination can also be used with default index groups when not supplying an index file.

### grompp now prints a note again when combining center of mass motion removal and position restraints

Issue 4128

### Static selections of large index groups now work

Commands like `gmx distance -f traj.xtc -n ndx.ndx -select "group \"Contacts\""` only worked if the size of `Contacts` was less than the number of atoms. This restriction was a bug, and has been fixed so that `Contacts` make take any size.

Other similar uses of static selections derived from index groups will also now work.

Issue 4148

### Static selections of index groups with repeated indices now work

Static groups from index files referenced in selections (e.g. `gmx tool -select "group \"Contacts\""`) only worked correctly if they never had adjacent repeats of the same index within the group. Repeating the same index can be meaningful e.g. in lists of inter-atomic distances to analyze with `gmx distance` to analyze `"1 2 2 3"`. Previously, the index group had to be written like `"2 3 1 2"` in order to work.

Issue 4149

### Fixes that affect portability

### Miscellaneous

### Fix a bug affecting re-run gmxapi scripts

A typo may have prevented gmxapi simulations from continuing from checkpoints after being interrupted. Fixed in version 0.2.3 of the gmxapi Python package.

Issue 4267

### GROMACS 2021.3 release notes

This version was released on August 18th, 2021. These release notes document the changes that have taken place in GROMACS since the previous 2021.2 version, to fix known issues. It also incorporates all fixes made in version 2020.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fix mdrun -ddorder pp_pme

When rank ordering PP-PME, mdrun would deadlock during the initialization phase.

Issue 4114

### Fixed gmxapi MD plugin binding

Molecular Dynamics extension code was not properly handled when added to a simulation through the gmxapi Python interface. This meant that restraint potentials would silently fail to be applied with gmxapi versions >= 0.1. Updates have been applied internally to gmxapi.

The gmxapi 0.2.2 Python package supports the updated GROMACS API and will issue errors if a simulation attempts to bind external plugin code with a compatible-but-broken API (GROMACS 2021 through 2021.2).

Third party code should not need to be updated, but developers will note an additional "null restraint" in https://gitlab.com/gromacs/gromacs/-/tree/master/python_packaging/sample_restraint (for illustration and testing purposes).

Issue 4078 and Issue 4102

### Fixed multi-rank restarts from checkpoints written by single-rank simulations

Currently a single-rank simulation never uses update groups, however a multi-rank run can do so. This fix ensures that the atoms within update groups always start in the same periodic image, which was not guaranteed if the checkpoint was written by a single-rank simulation.

Issue 4016

### Fixes for `gmx` tools

#### Fix gmx nmr -viol option

The tool would previously fail with a cryptic error. Also enforces that this option is exclusive with other analysis modes.

Issue 4060

### Fixed gmx dipoles -quad option

The tool now reports correct values.

Issue 4080

### Make sure gmx convert-tpr -until works

This got broken during reworking the internals of the tool and didn't calculate the number of remaining steps correctly.

Issue 4056

### Fixed dihedral transition counting in gmx chi and gmx angle

When a trajectory of only 1 frame is passed, transition counting is avoided (formerly it was attempted and crashed).

When a trajectory of multiple frames is passed, transition counting is correct (formerly it did not take place).

### Fixed possible crash in gmx chi histogramming

Formerly an invalid reference to a temporary string was used for a residue name, which might have caused a crash.

### Fixed gmx chi -chi_prod

Formerly it could crash or produce garbage results when the number of relevant dihedrals differed from the number of residues with dihedrals.

### Fixes that affect portability

### Check that necessary python modules are available

The source code validation could otherwise fail a build with cryptic errors.

Issue 3985

#### Ensure that NB-LIB and gmxapi can be build even without tests enabled

Could otherwise lead to cryptic build errors.

### Miscellaneous

#### Removed performance loss in the mdrun domain decomposition

With 16 or more so-called PP MPI ranks, the domain decomposition repartitioning could incur large performance overheads due to a sub-optimally sized hash table. This has now been fixed.

Issue 4054

## GROMACS 2021.2 release notes

This version was released on May 5th, 2021. These release notes document the changes that have taken place in GROMACS since the previous 2021.1 version, to fix known issues. It also incorporates all fixes made in version 2020.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Removed a potential race condition with GPU update

Fixed possible (but so far unobserved) race condition in coordinate copy when using GPU update with dipole moment calculation.

Issue 4024

#### Avoided issues with global reduction observed with md-vv

The new implementation for md-vv in the modular simulator could produce floating-point exceptions computing values on non-master ranks that were never used. This is now fixed by avoiding that computation. The other integrators were unaffected because they over-wrote the values computed.

Issue 4031

#### Prohibited SETTLE interactions for atoms with perturbed masses

Older implementations produced varying degrees of wrong results because this has never been implemented. Now both `mdrun` and `grompp` refuse to handle such a system, suggesting using normal constraints.

Issue 3959

### Rerun now writes pull output correctly

Refactoring omitted to preserve that pullf.xvg and pullx.xvg files should be written during a rerun. All 2019 and 2020 versions were affected, as well as 2021 and 2021.1. The pull output files are now written as they used to be in 2018 and earlier.

Issue 4043

### Fixes for `gmx` tools

### Fix incorrect behaviour with single residue chains in pdb2gmx

The code for chcking for cyclic molecules could lead to single residue chains incorrectly to be assigned as circular molecules.

Issue 4029

### Fix grompp check for position restraints with absolute reference

Fixed that grompp with position restraints would always issue a warning about using an absolute reference, even when an absolute reference was not used.

Issue 3996

### Fix error when using VMD plugin

Tools would crash with a C++ library assertion because the plugin loading code incorrectly tried to construct a string from nullptr.

Issue 3055

### Fix file permissions with `gmx solvate` and `gmx genion`

These used to write temporary files with Unix permissions 0600. Now they respect the umask of the process (typically 0644).

Issue 4040

### Fixes that affect portability

### Support for Intel oneAPI compiler 2021.2

Fixed compiler infinity math and MKL flags.

### Fix Apple OpenCL build

Issue 4008

### Fixed compilation issue with GCC 11

Issue 4039

### Miscellaneous

### Fix bond type in GROMOS force fields

The bond type for C and +N in [ACE] was incorrect.

Issue 3995

### Allow PME on CPU in runs with domain decomposition and GPU update

Relaxed a limitation which prevented running parallel runs with domain decomposition and GPU update to use the CPU for PME (as long as combined PP-PME ranks are used). This allows parallel runs to scale when the CPU resources are sufficient for PME.

Issue 4035

## GROMACS 2021.1 release notes

This version was released on March 8th, 2021. These release notes document the changes that have taken place in GROMACS since the previous 2021 version, to fix known issues. It also incorporates all fixes made in version 2020.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Fix MiMiC with virtual sites

It is likely that MiMiC with virtual sites has not worked correctly because the call to construct the sites was placed after the call that uses the sites. Now it should work, but we have not tested that it does.

Issue 3866

### Fix mass perturbation to dH/dlambda

The contribution for perturbed mass was missing in dH/dlambda. Note that this contribution was not missing from the foreign energy differences used for the Bennett acceptance ratio method.

Issue 3943

### Running AWH with a convolved potential and an FEP dimension gives wrong results.

The output PMF is wrong when using awh-potential = convolved with a pull dimension combined with an FEP dimension. The FEP dimension always uses an umbrella potential and the combination does not work properly. This has been disabled in grompp.

Issue 3946

### Remove velocity from partially frozen atoms in md-vv

md-vv would add some velocity to the frozen dimensions of partially frozen atoms during constraining. This did not lead to wrong trajectories, as the frozen dimensions of the positions are kept fixed during propagation. The non-zero velocities were, however, reported in trajectories and final configurations. They might also have lead to slightly wrong kinetic energies, since the reported kinetic energy is calculated after the velocities are constrained. All effects are expected to be relatively small, since they did not accumulate, as the velocities were regularly reset to zero once per step.

Issue 3849

### Fixes for `gmx` tools

### Fix periodic boundary conditions in analysis framework tools

There was a bug in the trajectory analysis framework which caused molecules that were broken over PBC not to be made whole. This would usually lead to obviously incorrect outliers in analysis output.

Issue 3900

### Fix range checking bug in `gmx covar`

A check was inverted causing range checking to be applied wrong.

Issue 3902

#### Fix various bugs in `gmx xpm2ps`

Numerous minor issues were introduced in refactoring since GROMACS 5.1, now fixed.

Issue 3881

#### Fixes that affect portability

#### Fixed compilation on Cygwin

A GROMACS header file was not including the necessary standard header. A problem with the `M_PI` math constant defined only by POSIX and not by C++ was also worked around.

Issue 3890

#### Improve grompp checks of AWH settings when sampling an FEP dimension

Ensure that the AWH sampling interval is compatible with nstcalcenergy when sampling an FEP dimension using AWH. This avoids crashes in the first AWH sampling step (step > 0) if the settings were not correct.

Issue 3922

#### Miscellaneous

- Updated GROMACS logos

### 11.8.2 Major release

#### Highlights

GROMACS 2021 was released on January 28th, 2021. Patch releases may have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

As always, we've got several useful performance improvements, with or without GPUs, all enabled and automated by default. In addition, several new features are available for running simulations. We are extremely interested in your feedback on how well the new release works on your simulations and hardware. The new features are:

- Support for multiple time stepping, allowing for simple near doubling of simulation speed and is intended to replace the virtual site treatment
- Ability to use stochastic cell rescaling barostat for equilibration and production simulations
- Preliminary support for using SYCL as accelerator framework
- Support for performing free energy perturbation with AWH
- Support PME offloading to GPU for free energy simulations

- Support for ARM SVE and Fujitsu A64FX (contribution by Research Organization for Information Science and Technology (RIST))
- New nonbonded interaction API with NB-LIB (in collaboration with PRACE)
- New GROMACS logo!

### New and improved features

#### Virtual site with single constructing atom

Added a virtual site that is constructed on top if its single constructing atom. This can be useful for free-energy calculations.

#### Density-guided simulations can apply matrix multiplication and shift vector to structures

The new mdp option "density-guided-simulation-shift-vector" defines a shift vector that shifts the density-guided simulation group before the density forces are evaluated. With a known shift vector that aligns structure and input density, this feature enables structure refinement to non-aligned densities without the need to manipulate the input density data or structure. The mdp option "density-guided-simulation-transformation-matrix" allows to define a matrix with which to multiply the structure coordinates, before the shift vector is applied. This allows arbitrary rotation, skewing and scaling of input structures with respect to the input densities. A typical use case are membrane-embedded proteins which cannot easily be shifted and rotated within membranes.

#### Lower energy drift due to SETTLE

GROMACS already applied an improvement to the center of mass calculation in SETTLE to reduce energy drift in single precision. Now the center of mass calculation is completely avoided, which significantly reduces the energy drift when large coordinate values are present. This allows for accurate simulations of systems with SETTLE up to 1000 nm in size (but note that constraining with LINCS and SHAKE still introduces significant drift, which limits the system size to 100 to 200 nm).

#### mdrun now reports energy drift

With conservative integrators, mdrun now reports the drift of the conserved energy quantity in the log file.

### FEP using AWH

It is now possible to control the lambda state of a free energy perturbation simulation using the Accelerated Weight Histogram method. This can be used as one of multiple AWH dimensions, where the other(s) are coupled to pull coordinates.

### Support for cyclic molecules in pdb2gmx

It is now possible to process cyclic molecules in pdb2gmx and generate GROMACS topology files for them.

### Stochastic cell rescaling barostat

Implementation of the stochastic cell rescaling barostat. This is a first-order, stochastic barostat, that can be used both for equilibration and production.

### Performance improvements

### Added support for multiple time-stepping

A two-level multiple time-stepping scheme has been implemented. Any combination of five different force groups can be selected to evaluate less frequently, thereby improving performance.

### Extend supported use-cases for GPU version of update and constraints

GPU version of update and constraints can now be used for FEP, except mass and constraints free-energy perturbation.

### Reduce time spent in grompp with large numbers of distance restraints

The time *gmx grompp* (page 230) spent processing distance restraint has been changed from quadratic in the number of restraints to linear.

Issue 3457

### Support for offloading PME to GPU when doing Coulomb FEP

PME calculations can be offloaded to GPU when doing Coulomb free-energy perturbations.

### CPU SIMD accelerated implementation of harmonic bonds

SIMD acceleration for bonds slightly improves performance for systems with H-bonds only constrained or no constraints. This gives a significant improvement with multiple time stepping.

### Allow offloading GPU update and constraints without direct GPU communication

Allow domain-decomposition and separate PME rank parallel runs to offload update and constraints to a GPU with CUDA without requiring the (experimental) direct GPU communication features to be also enabled.

### Tune CUDA short-range nonbonded kernel parameters on NVIDIA Volta and Ampere A100

Recent compilers allowed re-tuning the nonbonded kernel defaults on NVIDIA Volta and Ampere A100GPUs which improves performance of the Ewald kernels, especially those that also compute energies.

### Improvements to GROMACS tools

### Bugs fixed

### Fixed exported libgromacs CMake target

Update the exported libgromacs CMake target to not depend on non- existing include paths and add GMX_DOUBLE define to interface definitions. The target now gets exported into the GROMACS namespace.

Issue 3468

### Fixed unsolicited changing of atom names in pdb file

Remove functions to change atoms names when reading and writing pdb files. This affected naming of H atoms in particular.

Issue 3469

### pdb2gmx handles ASPH and GLUH inputs better

The default is to treat all such residues as the unprotonated form, and not to try to infer the protonation from the residue name in the input. Protonated forms are only available via the interactive selection options. Now pdb2gmx reports when it is converting such input residues automatically. It also ensures that the output configuration and topology are naming such residues correctly in both the default and interactive-selection cases.

Issue 2480

### Correct excluded perturbed interactions beyond the non-bonded cut-off distance

With free-energy calculations without coupling of intermolecular interactions, non-bonded pair interactions at distance longer than the cut-off distance can be excluded. These interactions would still have PME long-range contributions. The contributions are now removed. In addition, mdrun will stop with a fatal error when interactions beyond the pair-list cut-off are present.

Issue 3403 Issue 3808

### Corrected AWH initial histogram size

The initial histogram size for AWH biases depended (weakly) on the force constant. This dependence has been removed, which increases the histogram size by a about a factor of 3. In practice this has only a minor effect on the time to solution. For multiple dimensions, the histogram size was underestimated, in particular with a combination of slower and faster dimensions. The, now simplified, formula for the initial histogram size is given in the reference manual.

Issue 3751

### Fixed default for tick-mark spacing in gmx xpm2ps

This was inadvertently changed many years ago, leading to the intended default of automatic tick-mark spacing being replaced with an unsuitable fixed value.

Issue 3881

### Fixed LJ Ewald exclusions when used with cut-off electrostatics

The exclusion forces in CUDA and OpenCL kernels were computed incorrectly if LJ Ewald was used together with cut-off electrostatics.

Issue 3840

### Deprecated functionality

The core GROMACS team wants to let users and downstream developers know about impending changes so that disruption is minimized. Do get in touch if you feel something inappropriate is planned!

Deprecated functionality often remains in GROMACS for a year or more, but this should not be relied upon.

### Changes anticipated to GROMACS 2021 functionality

#### `gmx mdrun -membed`

The feature for embedding a protein in a membrane will be retained, but probably in a different form, such as `gmx membed`.

#### `gmx mdrun -rerun`

The feature for computing potential energy quantities from a trajectory will be retained, but probably in a different form, such as `gmx rerun` and `gmx test-particle-insertion`.

### Integrator .mdp options will only contain dynamical integrators

Energy minimization will be accessed in a diffent form, perhaps with `gmx minimize` and interpret an .mdp field for which minimizer to use. Normal-mode analysis may be accessed with e.g. `gmx normal-modes`. The command-line help for these tools will then be better able to document which functionality is supported when.

### Much functionality in `trjconv`, `editconf`, `eneconv` and `trjcat`

The functionality in such tools is being separated to make it available in composable modules, that we plan to make available as simpler tools, and eventually via the GROMACS API that is under development.

### `gmx do_dssp` to be replaced

This tool is deprecated, because it is problematic for some users to obtain and install a separate DSSP binary, so we plan to replace the implementation at some point with a native implementation, likely based upon xssp, and make it available under a new gmx tool name.

### Functionality deprecated in GROMACS 2021

#### `mdrun -deffnm` to be removed

This functionality is convenient when running very simple simulations, because it permits grouping of a set of files that then differ only their suffix. However, it does not work in the wider case of an `mdrun` module (or modules) writing multiple `.xvg` output files. The resulting filenames collide. That, and its interaction with checkpointing and appending, have led to quite a few bug reports.

Because users can use a folder to group files (a standard mechanism that they understand from experience outside of GROMACS), we can build and test better software for them if we remove the erstwhile convenience of `mdrun -deffnm`. Please update your workflows accordingly.

Issue 3818

### OpenCL to be removed as a GPU framework

Issue 3818 Work is underway for ports to AMD and Intel GPUs, and it is likely that those ports will not be based on the current GROMACS OpenCL port. Nvidia GPUs are targeted by the CUDA port, and no changes are expected there. The core team can't maintain, test, and extend up to 4 ports with current resource levels. Since there are no prospects of an emerging GPU vendor in HPC needing OpenCL support, we will remove the OpenCL port once AMD and Intel support is established in other ways.

### Intel KNC (MIC) support

Issue 3818 This architecture is nearly extinct in HPC. Note that KNL support will continue and is not affected by this deprecation.

### Sparc64 HPC ACE

This architecture is nearly extinct in HPC.

### Legacy SIMD architecture support

Issue 3818 We occasionally need to extend the GROMACS SIMD framework, and so should slowly remove older architectures that are difficult or impossible to test. The following implementations are deprecated and will not support new functionality in future.

- Power 7
- ARMv7 (this platform was deprecated in GROMACS 2020)
- x86 MIC (this platform was deprecated in GROMACS 2021)
- Sparc64 HPC ACE (this platform was deprecated in GROMACS 2021)

### The mdrun-only build of GROMACS

Issue 3808 Before GROMACS had the `gmx` wrapper binary, the `mdrun` binary could be built independently of the many other binary tools that were built by default. That was useful for installing on compute clusters because dependencies for `mdrun` were minimized. However, we now manage such dependencies better with CMake, and an mdrun-only build is no longer easier to build. The mdrun-only build is also harder to test, and introduces complexity into documenting GROMACS and teaching users to use it. So it is time to remove that build.

### Support for version 1 of the hardware locality library `hwloc`

Issue 3818 Version 2 has been supported in GROMACS for several years. The capabilities of newer hardware and hardware-support APIs are of most interest for GROMACS moving forward, so we should minimize our testing work and encourage clusters to upgrade older `hwloc` installations.

### Legacy API

Issue 3818 The legacy installed headers have been deprecated for a while, however we wish to state more broadly that all headers found within the `src` directory tree of GROMACS are intended for internal consumption only, and are thus subject to change without notice. Further, the form and contents of the `libgromacs` library and related CMake targets may change as we move towards building APIs and supporting machinery that can be stable and supported in the long term.

### Constant-acceleration MD

Issue 1354 This has been broken for many years, and will be removed as nobody has been found with interest to fix it.

### Reading .pdo files in `gmx wham`

The pull code in GROMACS before version 4.0 wrote files in `.pdo` format. Analyses of such files are likely no longer relevant, and if they are, using any older GROMACS version will work. `gmx wham` will be simpler to maintain and extend if we no longer support reading `.pdo` files.

## Functionality deprecated in GROMACS 2020

### Support for 32bit architectures

Issue 3252 There are no current or planned large scale resources using 32bit architectures, and we have no ability to properly test and evaluate them.

### Free-energy soft-core power 48

Issue 3253 Free-energy soft-core power 48 is almost never used and is therefore deprecated.

### Support for Armv7

Issue 2990 There are several issues with current code for the architecture, and we don't have the resources for support and fix issues related to it. As the architecture has no large HPC impact it is thus deprecated.

### Functionality deprecated in GROMACS 2019

#### Generation of virtual sites to replace aromatic rings in standard residues

Issue 3254 These are thought to produce artefacts under some circumstances (unpublished results), were never well tested, are not widely used, and we need to simplify pdb2gmx.

#### Benchmarking options only available with `gmx benchmark`

Issue 3255 Options such as `-confout`, `-resethway`, `-resetstep` are not intended for use by regular mdrun users, so making them only available with a dedicated tool is more clear. Also, this permits us to customize defaults for e.g. writing files at the end of a simulation part in ways that suit the respective mdrun and benchmark use cases, so `-confout` will no longer be required.

#### `gmx mdrun -nsteps`

Issue 3256 The number of simulation steps described by the .tpr file can be changed with `gmx convert-tpr`, or altered in .mdp file before the call to `gmx grompp`. The convenience of this mdrun option was outweighted by the doubtful quality of its implementation, no clear record in the log file, and lack of maintenance.

### Removed functionality

#### Removed GMX_SCSIGMA_MIN environment variable

This was used to reproduce free-energy soft-core behavior of GROMACS versions before 4.5.

### Portability

#### Python environment

Where Python is required, CPython versions 3.6 to 3.8 are supported.

CMake now detects Python using FindPython3. If you previously used `PYTHON_EXECUTABLE` to hint the location of the Python interpreter, you should instead specify the Python "root" or "prefix" path (the directory containing `./bin/python3`) with CMake variable `Python3_ROOT_DIR` or `CMAKE_PREFIX_PATH`. As other infrastructure evolves, `PYTHON_EXECUTABLE` may cease to have the desired effect without warning.

### CMake

Updated required CMake version to 3.13.

### C++ standard

GROMACS has updated the required C++ standards compliance from C++14 to C++17, and requires 2017 standard library features. See the install guide for details.

### Cygwin

GROMACS now builds on Cygwin with both gcc and clang compilers.

### Windows

GROMACS now builds correctly on Windows with MSVC even when the path to the source or build directory has a space in it.

Builds with MSVC 2019 correctly detect the proper static linking setup during CMake configuration.

### RDTSCP usage and reporting

GROMACS now defaults always on x86 to use the RDTSCP machine instruction for lower latency timing. Very old machines might need to configure with `GMX_USE_RDTSCP=off`. Non-x86 platforms are unaffected, except that they will no longer report that RDTSCP is disabled (because that is self-evident).

### armv8+sve support (ARM_SVE)

Support for ARM Scalable Vector Extensions (SVE) has been added. GROMACS supports SVE vector length fixed at CMake configure time (typically via the -msve-vector-bits=<len> compiler option), which is at the time of the release supported in GNU GCC 10 and later, and will supported soon by LLVM 12 and compilers based on this. The default is to detect the default vector length at CMake configure time, and that can be changed with the `GMX_SIMD_ARM_SVE_LENGTH=<bits>` option. Supported values are 128, 256, 512 and 1024. Note that the nonbonded kernels have not been optimized for ARM_SVE as of yet. ARM_SVE support is contributed by the Research Organization for Science Information and Technology (RIST)

### Miscellaneous

#### Default values for temperature and pressure coupling intervals are now 10

With the default mdp input value of -1 for nsttcouple and nstpcouple, grompp would set these values to nstlist. Now these are set to 10 and thus independent of nstlist (note that grompp may choose smaller values when needed for accurate integration).

#### Uniform and manual CMake GPU-support configuration

The GPU accelerations setup has been changed to be uniform for CUDA and OpenCL. Either option is now enabled by setting GMX_GPU to CUDA or OpenCL in the CMake configuration. To simplify the CMake code, we have also moved away from automated option selection based on the build host. In particular, this means that CUDA will not be enabled unless the GMX_GPU option is explicitly enabled, and CMake will no longer perform the extra steps of trying to detect hardware and propose to install CUDA if hardware is available. Apart from the simplification, this should also make it easier to handle multiple different accelerator APIs targeting e.g. NVIDIA hardware.

#### Configuration-time trivalue options changed from autodetection to boolean on/off

To simplify the CMake configuration and avoid having multiple settings that change outside of the users direct control we have removed the support for automatically setting booleans. GMX_BUILD_HELP and GMX_HWLOC are now disabled by default, while GMX_LOAD_PLUGINS is enabled by default.

#### gmxapi C++ interface

`gmxapi::Context` is now created with `gmxapi::createContext()`, which allows the client to provide an MPI communicator for the library to use instead of its default (e.g MPI_COMM_WORLD). MPI-enabled clients may use the `gmxapi/mpi/gmxapi_mpi.h` template header and the `assignResource()` helper to generate the argument to `createContext`.

#### Unification of several CUDA and OpenCL environment variables

The environment variables that had exactly the same meaning in OpenCL and CUDA were unified:

- GMX_CUDA_NB_ANA_EWALD and GMX_OCL_NB_ANA_EWALD into GMX_GPU_NB_ANA_EWALD
- GMX_CUDA_NB_TAB_EWALD and GMX_OCL_NB_TAB_EWALD into GMX_GPU_NB_TAB_EWALD
- GMX_CUDA_NB_EWALD_TWINCUT and GMX_OCL_NB_EWALD_TWINCUT into GMX_GPU_NB_EWALD_TWINCUT

#### Dysfunctional parts of the QMMM interface has been removed

Currently, GROMACS supports QM/MM officially only via MiMiC; a new CP2K QM/MM interface is being developed within BioExcel. All other QM/MM support has been untested and likely dysfunctional for years and has now been removed from .mdp input and output, resulting in smaller .mdp output files from grompp.

## 11.9 GROMACS 2020 series

### 11.9.1 Patch releases

#### GROMACS 2020.7 release notes

This version was released on February 3rd, 2022. These release notes document the changes that have taken place in GROMACS since the previous 2020.6 version, to fix known issues.

#### Fixes where mdrun could behave incorrectly

#### Fixed bug with GPU LINCS occasionally shifting atoms in wrong direction

Due to missing blocking synchronizations in the CUDA version of LINCS, the shared memory was occasionally overwritten with the new data. This may slightly affect the final coordinates of the shifted atoms.

Issue 4199

#### Fix calculation of restraint potential for large restraint deviations

The calculation in the code did not follow the description of the potential in the manual but the potential continued to grow quadratically instead of linearly as it should.

Issue 4346

#### Fixes for `gmx` tools

#### Fixes that affect portability

#### Miscellaneous

#### GROMACS 2020.6 release notes

This version was released on March 4th, 2021. These release notes document the changes that have taken place in GROMACS since the previous 2020.5 version, to fix known issues.

### Fixes where mdrun could behave incorrectly

### Cosine acceleration failed to abort if it could not be run

Cosine acceleration is only compatible with the leap-frog integrator (`integrator = md`). GROMACS did, however, accept input files requesting cosine acceleration for other integration algorithms, and did report viscosity-related quantities from these simulations. Since the cosine acceleration was never applied in these cases, any results obtained from simulations with enabled cosine acceleration and integrators other than `md` should be regarded as invalid.

Issue 3903

### Fixes for `gmx` tools

### Fix range checking bug in `gmx covar`

A check was inverted causing range checking to be applied wrong.

Issue 3902

### Fixes that affect portability

### Miscellaneous

### GROMACS 2020.5 release notes

This version was released on January 6th, 2021. These release notes document the changes that have taken place in GROMACS since the previous 2020.4 version, to fix known issues. It also incorporates all fixes made in version 2019.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Fix mdrun writing zero dH/dlambda and foreign lambda energies before checkpointing

With free-energy runs with separate-dhdl-file=no and nstdhdl not a multiple of nstenergy, mdrun would write zeros for dH/dlambda and foreign energies to the energy file for steps between the last energy frame and the checkpoint. This would lead to errors in free-energy estimates which could go unnoticed as values only deviate for a few steps.

Issue 3763

### Fixed bugs with COM pulling and domain decompostion with weight or >32 ranks

When using COM pulling and domain decomposition, the results would be incorrect when using relative weights per atom or when using more than 32 DD MPI ranks. This would usually lead to crashes or obviously wrong results.

Issue 3750

### Fix incorrect AWH free-energies when multiple walkers share a bias

The AWH free-energy output was incorrect when multiple walkers shared an AWH bias. The error went up quadratically with the free-energy update interval, as well as with the number of walkers. The error decreases as update size decreases with time. This meant that with default AWH settings the error was negligible. With a free-energy update interval of 2 ps, we observed an error about equal to the statistical error with 32 walkers for a rather fast reaction coordinate. For slower coordinates the error will be smaller than the statistical error.

Issue 3828

### Fixed conserved energy for MTTK

When using *pcoupl=MTTK* (page 66) and *tcoupl=nose-hoover* (page 64), the calculated conserved energy was incorrect due to two errors dating back to GROMACS 4.6 and 2018, respectively. As a result, all reported conserved energies using this combination of temperature and pressure coupling algorithms in any GROMACS version since GROMACS 4.6 are likely to be wrong. Note that these errors did not impact the dynamics, as the conserved energy is only reported, but never used in calculations. Also note that this bug only affects this exact combination of temperature / pressure coupling algorithms.

Issue 3796

### Fixed conserved energy for Nose-Hoover

When using *tcoupl=nose-hoover* (page 64) and one or more temperature groups with non-integer number of degrees of freedom, the calculated conserved energy was incorrect due to an error dating back to GROMACS 2018. Reported conserved energies using Nose-Hoover temperature coupling and non-integer number of degrees of freedom since GROMACS 2018 are likely to be slightly off. Note that this error does not impact the dynamics, as the conserved energy is only reported, but never used in calculations. Also note that this will only be noticeable when using small temperature groups or small systems.

Issue 3831

### Fixed kinetic energy and temperature reporting for MTTK

When using *pcoupl=MTTK* (page 66) and *tcoupl=nose-hoover* (page 64), the reported kinetic energy and temperature were very slightly off. The integration of the temperature coupling trailed the reporting by half a time step. Note that these errors did not impact the dynamics, as the quantities were correctly integrated and only wrongly reported. Also note that the difference is so small that it is unlikely to have been significant for any application except for rigorous algorithm validation. Finally, note that this bug only affects this exact combination of temperature / pressure coupling algorithms.

Issue 3832

### Fix pull error message with angles and dihedrals

The COM pull code could print incorrect pull group indices when mdrun exited with an error about a too long pull distance in angle and dihedral geometries.

Issue 3613

### Fix numerical issues in expanded ensemble

When performing simulated tempering or expanded ensemble simulations with changes in the Hamiltonian that were too large, then Monte Carlo proposals to states that were sufficiently unlikely would underflow, causing division by zero errors. This was fixed by numerically hardening the logical flow so that such proposals would be rejected instead.

Issue 3304

### Fix incorrect electric field strength with applied electric field

The electric field generated by the electric field module would be incorrect when used together with domain decomposition due to an error with indexing the field to all atoms instead of just those on the current domain.

In overlap regions between domains, which have the thickness of the pairlist cut-off distance, the electric field would be doubled (or more with 2D or 3D domain decomposition).

To validate if a simulation has been affected by the issue, users should calculate the actual potential across the simulation box using the Poisson equation. If this potential agrees with the one provided as the input, a simulation was not affected.

Issue 3800

### Fixes for `gmx` tools

#### Improve CHARMM support in gmx do_dssp

Issue 3568

#### Fix non-funtioning gmx h2order -d option

The gmx h2order tool would always take the normal along the z-axis.

Issue 3820

#### Fix pull group index handling

The pull code would not validate its index groups correctly, leading to infinite loops or assertions being triggered at grompp time.

Issue 3810

### Fixes that affect portability

#### Fix building on OSX

The code wouldn't compile due to a missing include.

Issue 3730

### Miscellaneous

#### GROMACS 2020.4 release notes

This version was released on October 6th, 2020. These release notes document the changes that have taken place in GROMACS since the previous 2020.3 version, to fix known issues. It also incorporates all fixes made in version 2019.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Bug fix for the GPU version of LINCS in multiple domain case

Increase in the maximum number of coupled constraints in the domain did not trigger memory re-allocation, which is now fixed. This can happen, e.g. when big molecule enters the domain, previously occupied by smaller molecules. The bug does not affect the single domain case.

### Fix index handling of N-body virtual sites with domain decomposition

Incorrect indexing would be used to handle N-body virtual sites in the domain decomposition code. This would usually lead to crashes due to illegal or incorrect memory usage.

Issue 3635

### Fix assertion failure with LJ-PME and dispersion correction

With vdw-type=PME and dispersion correction, mdrun would exit with an assertion failure during PME tuning.

Issue 3677

### Bug fix for FEP calculations with modular simulator and domain decomposition

When using the modular simulator, domain decomposition and free energy calculations with perturbed masses, the simulation would always be performed using the masses at lambda=0 instead of the actual lambda value.

### Added workaround for RDRAND not always returning random numbers on Ryzen

On AMD Ryzen 3000 series CPUs, the hardware random number generator (RDRAND) can behave incorrectly, always returning -1 (0xFFFFFFFF). When this hardware bug is detected at runtime, GROMACS will switch to its software-based pseudo-random number generator instead.

While many motherboard vendors have been distributing firmware updates that contain microcode fixes and most motherboards are sold with these factory-installed, there can still be some systems affected that didn't receive the updates.

In case you ran simulations on one of these systems, in theory all random number seeding could be affected (see below for algorithms), since it would mean the same seed is used. Even this should be fine for virtually all individual simulations since the generated numbers are still random. The most likely case that would be seriously affected is if you use identical starting conformations and start many simulations with different random seeds generated automatically (instead of manually selecting your seeds) - then the Ryzen hardware bug could mean all your simulations actually get the same generated initial velocities, or the same stochastic changes, etc. depending which algorithms you are using.

A list of affected algorithms can be found below:

1. Seeding in *gmx grompp* (page 230) is affected if no user supplied seed is used (e.g. if `-1` is used to ask GROMACS to generate a seed). This can affect Langevin/Stochastic dynamics, v-rescale thermostat, anything Monte-Carlo related and the generation of random velocities.
2. Decision when to exchange replicas during replica exchange simulations.
3. Simulations using the random components from `AWH`.
4. Some analysis and preparation tools might be affected, e.g. free volume calculation, ion placement, WHAM, normal mode analysis and PME error estimates.

Diagnosing: to aid detecting the error, run `gmx mdrun -debug 1` with GROMACS 2020.4 or later, which will produce a debug log, typically called `gmx.debug`. This file will contain the following message if the processor the program ran on is affected:

- Hardware random number generator (RDRAND) returned -1 (0xFFFFFFFF) twice in a row. This may be due to a known bug in AMD Ryzen microcode. Will use pseudo-random number generator (PRNG) rather than hardware device.

Earlier releases will fail SeedTest.makeRandomSeed test from unit tests suite on the affected systems. To check, run `make check` in your build folder. You can also find a sample testing code at the link below.

For more information on the issue, please check this website.

### Fixes for `gmx` tools

### Fix default output with gmx trjcat -demux

Files would not be written when using default file name output.

Issue 3653

### Fixes that affect portability

### CUDA 11.0 supported

A build with CUDA 11.0 now configures and passes tests. Building with CUDA 11.0 means that hardware with CC 3.0 is no longer supported, while CC 8.0 can now be used.

Issue 3632

### Fix building with MSVC

The build would fail due to a missing header.

Issue 3669

### Only check for RDTSCP on x86 platforms

### Miscellaneous

### Fix crash of grompp when the whole system is frozen

When the whole system would be frozen, grompp would crash with a segmentation fault.

Issue 3683

### Fixes the unexpected change in molecule indexing in output after simulation

Molecule indices of repeat molecules are now again numbered consecutively as expected (instead of all `1`).

Issue 3575

### Fix `INTERFACE_INCLUDE_DIRECTORIES` for `libgromacs` CMake target

`libgromacs.cmake` was malformed, referencing non-existent directories.

Issue 3592

### GROMACS 2020.3 release notes

This version was released on July 9th, 2020. These release notes document the changes that have taken place in GROMACS since the previous 2020.2 version, to fix known issues. It also incorporates all fixes made in version 2019.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Fixes for `gmx` tools

### Fix incorrect reading of certain older tpr files

Certain older tpr files could be read incorrectly, usually leading to an exit with a memory allocation error.

### Fix segmentation fault with gmx lie

The tool would crash due to mismatching sizes of energy terms in the file and F_NRE.

Issue 3547

### Fix matrix reading in gmx xpm2ps

The tool would fail to read a matrix if no second matrix was provided.

Issue 3551

### Fix uninitialized variable warnings in gmx hbond

Tool would produce garbage due to using uninitialized memory.

Issue 3550

### Actually fix gmx do_dssp

The tool was still broken and gave incorrect results after the previous fix.

Issue 3444

### Allow configuration of dssp default path

Users can configure the default path for dssp using GMX_DSSP_PROGRAM_PATH.

Issue 3520

### Avoid segmentation fault in gmx genrestr

The tool could fail when running simple inputs due to memory access errors caused by accessing free'd memory.

Issue 3582

### Fixes that affect portability

### Update MSVC SIMD flags

Newly supported SIMD flags may improve performance on recent x86 running Windows.

### Fix error with tinyxml2 linking

The signature for linking the external library was wrong.

### Miscellaneous

### Updated message on using GPU with non-dynamical integrator

The GPU implementation of PME and bonded forces requires dynamical integrator. The message that informs user why using GPU for PME or bonded forces is not supported with non-dynamical integrator was made more clear.

### GROMACS 2020.2 release notes

This version was released on April 30th, 2020. These release notes document the changes that have taken place in GROMACS since the previous 2020.1 version, to fix known issues. It also incorporates all fixes made in version 2019.6 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Ewald dipole correction incorrect without domain decomposition

Ewald dipole correction (epsilon-surface != 0) is now disabled when not using domain decomposition. With domain decomposition, it only works when each molecule consists of a single update group (e.g. water). This will be fixed in release-2021.

Issue 3441

#### Expanded ensemble simulations restarted from checkpoints

When restarting expanded ensemble simulations from checkpoints, expanded ensemble would silently refuse to run, and simulations would remain in their original lambda state.

Issue 3465

#### Fixed free energy calculations with LJ PME

Fixed an issue that calculated wrong long-range corrections when using free energy perturbation with `vdwtype = pme`. This affected forces, energies, lambda derivatives and foreign lambdas.

Issue 3470

#### The velocities of the center of mass are now removed correctly in case of -update gpu

When the center of mass motion is removed, the velocities are updated in the CPU memory. In case of GPU update, they should be copied back to the GPU memory after they were updated on the CPU. This affected most runs where the velocity of the center of mass has to be removed, in particular these where this velocity is large in the beginning of the run.

Issue 3508

#### Fix checkpoint restart with non-zero initial step

When restarting from the checkpoint, the init-step mdp parameter was ignored while checking if the simulation is already finished. As a result, this check only worked properly when init-step was 0 or was not specified.

Issue 3489

### Fixes for `gmx` tools

#### Time output unit fixes

When selecting a time unit of microseconds or larger, `gmx tool -tu` now produces the correct string in .xvg and particularly .xvgr plots

#### Fix do_dssp

The tool would fail with a segmentation fault.

Issue 3444

### Fixes that affect portability

#### Give clearer message about not detecting IBM_VSX support in gcc > 9

CMake would fail with a confusing error message.

Issue 3380

### Miscellaneous

#### Fixed initial DLB state reporting

The initial DLB state was reported incorrectly in the log file when the either "on" or "auto" value was the chosen at mdrun startup.

## GROMACS 2020.1 release notes

This version was released on March 3rd, 2020. These release notes document the changes that have taken place in GROMACS since the previous 2020 version, to fix known issues. It also incorporates all fixes made in version 2019.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fix fatal error with mdrun -multidir with more than 1 rank per simulation

Issue 3296

### Fix deadlock in mdrun runs with multiple ranks and separate PME ranks

When multiple PP ranks as well as separate PME ranks are used, mdrun could deadlock before starting the PP-PME balancing.

Issue 3335

### Avoid mdrun assertion failure when running with shells and update on a GPU

A check for shells has been added in the mdrun task assignment code, so that mdrun falls back to CPU or produces a clear error message when attempting to run with shells and update on a GPU.

Issue 3303

### Allow large prime factors in the mdrun MPI rank count

The domain decomposition would refuse to run with large prime factors in the MPI rank count even when the grid was specified by the user.

Issue 3336

### Actually fix PME forces with FE without perturbed q/LJ

PME would incorrectly ignore the mesh forces on perturbed atoms when no charges or LJ atom types were actually perturbed. Note that this is a rather uncommon scenario.

Issue 2640 Issue 3359

### Avoid deadlock when checking for missing DD interactions

When missing bonded interactions after domain decomposition were detected, mdrun was deadlocking instead of exiting with a failure.

Issue 3373

### Fix checkpoint restarts using Parrinello-Rahman and md-vv

Checkpoints using Parrinello-Rahman and md-vv (only implemented in the new modular simulator approach) could not be read.

Issue 3377

### Avoid overzealous program abort with orientation restraints

It could happen that mdrun would abort on checking orientation restraints in multiple molecules even though no restraints where applied to them.

Issue 3375

### Add fatal error for mdrun -multidir when simulations sharing state start at different step

When (re)starting mdrun -multidir for simulations sharing state data (e.g., replica exchange, AWH with bias sharing or NMR ensemble averaging) having a different initial step only caused a note to be printed, which could lead to simulations getting out of sync. Now a fatal error is issued in this situation.

Issue 2440 Issue 3990

### Correct skewed box using modular simulator without DD

Using modular simulator without DD, it was not checked whether the box was getting overly skewed when using pressure control.

Issue 3383

### Fix NMR restraints using modular simulator

Using NMR restraints (distance or orientation restraints) under modular simulator did not work as expected. All orientation restraint simulations would fail with a segmentation fault, as would distance restraint simulations using time averaging. All other distance restraint simulations would run correctly, but output to the energy trajectory would only occur if it coincided with general energy writing steps.

Issue 3388

### Avoid integer overflow when using dispersion correction

A change in the integer type storing the index meant that the value could overflow and turn negative, leading to wrong lookup and unphysical values.

Issue 3391

### Fix too small pairlist buffer on Intel GPUs

The pairlist buffer generated for Intel GPUs was slightly too small, because it assumed a 4x4 atom-cluster pair kernel instead of 4x2.

Issue 3407

### Fix checkpoint files getting out of sync with simulations sharing data

When simulations share data, e.g., replica exchange, AWH with bias sharing or NMR ensemble averaging, MPI barrier have now been added before renaming the checkpointing files to avoid that checkpoints files from the simulations can get out of sync. Now in very unlikely cases some checkpoint files might have temporary names, but all content will be in sync.

Issue 2440

### Fix simulations using graph and modular simulations

Simulations using modular simulator and a graph object would fail with a segmentation fault.

Issue 3389

### Fix center of mass motion removal with frozen atoms

When frozen atoms were part of center of mass motion removal groups, they would still contribute to the mass of those groups. This meant that the COM velocity correction was (slightly) too small. Now completely frozen atoms are removed from COM removal groups by grompp. When atoms are only frozen along one or two dimensions and part of a COM removal group, grompp now issues a warning.

Issue 2553

### Fix temperature calculation when center of mass motion is removed for part of the system

In the uncommon case where the center of mass motion is removed for part of the system but not the whole system, the number of degrees of freedom for the part without COMM removal would be incorrectly lowered by 3.

Issue 3406

### Fix possible issue with picking undefined NB kernel types

The CPU reference implementations for the NB kernels were missing some definitions for specific kernel types. This only affected installations that have SIMD explicitly turned off, something that is unlikely to happen in production environments.

Issue 2728

#### Fixes for `gmx` tools

#### Fixes that affect portability

#### Add support for ICC NextGen

Add support for Intel Compiler based on LLVM technology. To compile GROMACS with this compiler use `CXX=icpc CXXFLAGS=-qnextgen cmake`.

#### Document known issues with OpenCL on Volta and Turing

Issue 3125

#### Miscellaneous

#### Fix check for modified source files in release tarballs

It could happen that modifications to the source tree were not picked up if they happened after the build directory had been generated.

Issue 3302

### 11.9.2 Major release

#### Highlights

GROMACS 2020 was released on January 1, 2020. Patch releases may have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

As always, we've got several useful performance improvements, with or without GPUs, all enabled and automated by default. In addition, several new features are available for running simulations. We are extremely interested in your feedback on how well the new release works on your simulations and hardware. The new features are:

- Density-guided simulations allow "fitting" atoms into three-dimensional density maps.
- Inclusion of gmxapi 0.1, an API and user interface for managing complex simulations, data flow, and pluggable molecular dynamics extension code.
- New modular simulator that can be built from individual objects describing different calculations happening at each simulation step.
- Parrinello-Rahman pressure coupling is now also available for the md-vv integrator.
- Running almost the entire simulation step on a single CUDA compatible GPU for supported types of simulations, including coordinate update and constraint calculation.

### New and improved features

### Density-guided simulations

Users can now apply additional forces from three dimensional reference densities. These forces can be used to "fit" atoms into the densities by increasing the similarity of a simulated density to the reference density.

Multiple protocols are available for how to calculate simulated densities as well as how the similarity between a reference and a simulated density is evaluated.

### Virtual site on the line through two atoms at fixed distance

This is use useful for e.g. halogens in the CHARMM force field.

Issue 2451

### gmxapi Python support

Data flow driven simulation and analysis from Python is now available in a default GROMACS installation when users install the *gmxapi* Python package. See *gmxapi Python package* (page 694).

### New modular simulator

A new approach for how to combine individual calculation steps during a single simulation step is introduced, with focus on extensibility and modularization. This simulator is the default now for simulations using velocity-verlet in NVE, NVT (v-rescale thermostat only), NPT (v-rescale thermostat and Parrinello-Rahman barostat only), or NPH (Parrinello-Rahman barostat only), with or without free energy perturbation.

### Performance improvements

### Up to a factor 2.5 speed-up of the non-bonded free-energy kernel

The non-bonded free-energy kernel is a factor 2.5 faster with non-zero A and B states and a factor 1.5 with one zero state. This especially improves the run performance when non-perturbed non-bondeds are offloaded to a GPU. In that case the PME-mesh calculation now always takes the most CPU time.

### Proper dihedrals of Fourier type and improper dihedrals of periodic type are SIMD accelerated

### Avoid configuring the own-FFTW with AVX512 enabled when GROMACS does not use AVX512

Previously if GROMACS was configured to use any AVX flavor, the internally built FFTW would be configured to also contain AVX512 kernels. This could cause performance loss if the (often noisy) FFTW auto-tuner picks an AVX512 kernel in a run that otherwise only uses AVX/AVX2 which could

run at higher CPU clocks without AVX512 clock speed limitation. Now AVX512 is only used for the internal FFTW if GROMACS is also configured with the same SIMD flavor.

### Update and constraints can run on a GPU

For standard simulations (see the user guide for more details), update and constraints can be offloaded to a GPU with CUDA. Thus all compute intensive parts of a simulation can be offloaded, which provides better performance when using a fast GPU combined with a slow CPU. By default, update will run on the CPU, to use GPU in single rank simulations, one can use new '-update gpu' command line option. For use with domain decomposition, please see below.

### GPU Direct Communications

When running on multiple GPUs with CUDA, communication operations can now be performed directly between GPU memory spaces (automatically routed, including via NVLink where available). This behaviour is not yet enabled by default: the new codepaths have been verified by the standard GROMACS regression tests, but (at the time of release) still lack substantial "real-world" testing. They can be enabled by setting the following environment variables to any non-NULL value in your shell: GMX_GPU_DD_COMMS (for halo exchange communications between PP tasks); GMX_GPU_PME_PP_COMMS (for communications between PME and PP tasks); GMX_FORCE_UPDATE_DEFAULT_GPU can also be set in order to combine with the new GPU update feature (above). The combination of these will (for many common simulations) keep data resident on the GPU across most timesteps, avoiding expensive data transfers. Note that these currently require GROMACS to be built with its internal thread-MPI library rather than any external MPI library, and are limited to a single compute node. We stress that users should carefully verify results against the default path, and any reported issues will be gratefully received to help us mature the software.

### Bonded kernels on GPU have been fused

Instead of launching one GPU kernel for each listed interaction type there is now one GPU kernel that handles all listed interactions. This improves the performance when running bonded calculations on a GPU.

### Delay for ramp-up added to PP-PME tuning

Modern CPUs and GPUs can take a few seconds to ramp up their clock speeds. Therefore the PP-PME load balancing now starts after 5 seconds instead of after a few MD steps. This avoids sub-optimal performance settings.

### Improvements to GROMACS tools

#### Fixed bug in gmx order -calcdist

The reference position for the distance calculation was calculated wrongly.

#### Improved grompp usability by rejecting more invalid .mdp lines

Lines like

```
ref-t 298 = 0.1 =
```

are now all rejected with a descriptive message, which will help prevent some kinds of errors in constructing .mdp inputs. Note that an .mdp parameter name with a missing value is still accepted, and leads to the default behavior for that parameter.

#### Added convert-trj

A new tool *convert-trj* (page 179) has been added to allow users to interchange trajectory formats without having to use legacy *gmx trjconv* (page 334). Supported actions include the generation of slimmed down output trajectories, as well as the replacement of particle information in individual frames with data from a structure file. The new tool allows the usage of command line selections, meaning it is no longer necessary to write *index* (page 580) files to select certain atoms. It is part of the drive to split up the *trjconv* (page 334) tool into smaller parts.

#### Added extract-cluster

Added a dedicated tool to extract trajectory frames corresponding to different clusters obtained from *gmx cluster* (page 171). The new *extract-cluster* (page 218) tool generates new trajectories that contain only those frames that correspond to the correct cluster. The corresponding option **-sub** in *gmx trjconv* (page 334) has been removed.

#### Changed behaviour of genion

Functionality of genion was altered to prevent swapping ions for solvent closer than -rmin from any other non-solvent atom. This improvement prevents situations where an ion could be placed at the core of a protein, which would potentially render the folded protein less stable or may require long equilibration times.

### Bugs fixed

#### gmx mdrun -append now requires that a checkpoint is found

Previously `gmx mdrun -append` would start from the .tpr configuration (and thus not append) when the checkpoint file was missing.

#### The Verlet buffer now correctly handles perturbed constraints

With free-energy calculations with perturbed constraints, the Verlet buffer could be underestimated when constraint lengths were perturbed. As usually only very few constraints are perturbed, the effect is very small and much smaller than the overestimate of the buffer due to approximations, so the results of most runs with perturbed constraints will not have been affected.

Issue 4395

### Deprecated functionality

The core GROMACS team wants to let users and downstream developers know about impending changes so that disruption is minimized. Do get in touch if you feel something inappropriate is planned!

Deprecated functionality often remains in GROMACS for a year or more, but this should not be relied upon.

#### Changes anticipated to GROMACS 2020 functionality

##### gmx mdrun -membed

The feature for embedding a protein in a membrane will be retained, but probably in a different form, such as `gmx membed`.

##### gmx mdrun -rerun

The feature for computing potential energy quantities from a trajectory will be retained, but probably in a different form, such as `gmx rerun` and `gmx test-particle-insertion`.

#### Integrator .mdp options will only contain dynamical integrators

Energy minimization will be accessed in a diffent form, perhaps with `gmx minimize` and interpret an .mdp field for which minimizer to use. Normal-mode analysis may be accessed with e.g. `gmx normal-modes`. The command-line help for these tools will then be better able to document which functionality is supported when.

#### Much functionality in `trjconv`, `editconf`, `eneconv` and `trjcat`

The functionality in such tools is being separated to make it available in composable modules, that we plan to make available as simpler tools, and eventually via the GROMACS API that is under development.

#### `gmx do_dssp` to be replaced

This tool is deprecated, because it is problematic for some users to obtain and install a separate DSSP binary, so we plan to replace the implementation at some point with a native implementation, likely based upon xssp, and make it available under a new gmx tool name.

### Functionality deprecated in GROMACS 2020

#### Support for 32bit architectures

Issue 3252 There are no current or planned large scale resources using 32bit architectures, and we have no ability to properly test and evaluate them.

#### Free-energy soft-core power 48

Issue 3253 Free-energy soft-core power 48 is almost never used and is therefore deprecated.

#### Support for Armv7

Issue 2990 There are several issues with current code for the architecture, and we don't have the resources for support and fix issues related to it. As the architecture has no large HPC impact it is thus deprecated.

### Functionality deprecated in GROMACS 2019

#### Generation of virtual sites to replace aromatic rings in standard residues

Issue 3254 These are thought to produce artefacts under some circumstances (unpublished results), were never well tested, are not widely used, and we need to simplify pdb2gmx.

### Benchmarking options only available with `gmx benchmark`

Issue 3255 Options such as `-confout`, `-resethway`, `-resetstep` are not intended for use by regular mdrun users, so making them only available with a dedicated tool is more clear. Also, this permits us to customize defaults for e.g. writing files at the end of a simulation part in ways that suit the respective mdrun and benchmark use cases, so `-confout` will no longer be required.

#### `gmx mdrun -nsteps`

Issue 3256 The number of simulation steps described by the .tpr file can be changed with `gmx convert-tpr`, or altered in .mdp file before the call to `gmx grompp`. The convenience of this mdrun option was outweighted by the doubtful quality of its implementation, no clear record in the log file, and lack of maintenance.

### Removed functionality

### Group cut-off scheme

The group cut-off scheme has been removed. Several kinds of simulation that depend on it no longer work.

- Simulations under vacuum conditions are not supported.
- User supplied tables for short-range nonbonded interactions are not supported.
- Switched short-range nonbonded interactions with PME are not supported.
- Membrane embedding is deactivated.
- QMMM is not supported.

Issue 1852

### Generalized reaction-field

This only worked correctly with the group scheme. Note that generalized reaction-field simulations can still be performed using standard reaction field and computing the dielectric constant manually.

### gmx anadock

The gmx anadock tool was removed since it does not belong in GROMACS (it analyzes AutoDock outputs).

### gmx dyndom

The gmx dyndom tool was removed since it does not belong in GROMACS (it analyzes DynDom outputs).

### gmx morph

The gmx morph tool was removed since it yields non-physical structures that can easily be done by a script.

### gmx mdrun -gcom

This feature sometimes overrode the effects of various .mdp settings in a way that was difficult to understand and report. A user who wants to do communication between PP ranks less often should choose their `nst*` mdp options accordingly.

## Portability

### Added support for Hygon Dhyana CPU architecture

Support for hardware detection and related heuristics has been implemented for the Hygon Dhyana derived from the first-gen AMD Zen which it shares most of its architectural details with.

### Enabled PME offload support with OpenCL on NVIDIA and Intel GPUs

Thanks to portability improvements, the previously disabled PME OpenCL offload is now enabled also on NVIDIA and Intel GPUs.

### Fixed building on Solaris with GCC

GROMACS now builds on Solaris with GCC (tested on illumos distribution openindiana, "Hipster" rolling release, using GCC 5, 6, 7, and 8).

## Miscellaneous

### grompp now warns if macros in mdp "define" field are unused in topology

Macros defined in the mdp (with e.g. -DPOSRES) now cause a warning in grompp if they are not encountered while parsing the topology file

Issue 1975

#### Introduced CMake variable GMX_VERSION_STRING_OF_FORK

To help users and developers understand which version of GROMACS is being used, anybody providing a forked version of GROMACS shall set GMX_VERSION_STRING_OF_FORK in the source code (or if necessary when running CMake). It will then appear in the log file and users will know which version and fork of the code produced the result.

#### Provide checksum to validate release tarballs

Released versions of GROMACS will now provide a checksum calculated from the files participating in building the binaries. When building GROMACS from the tarball, the files will be checksummed again and compared against the checksum generated during the release build. If the checksums don't match, the version string is modified to indicate that the source tree has been modified, and the information is printed in the log files for the users. If checksumming has not been possible (either due to missing Python during installation, or because the original checksum file is missing), this is indicated through a different version string.

Issue 2128

#### Updated physical constants to CODATA 2018

Reproducibility of computed quantities is easiest to achieve if software stays up to date with the standards. The values for standard units have thus been updated to conform with the data available here.

#### Change grompp warning about decoupling without SD to a note

The warning that grompp issues when decoupling a molecule without the use of the SD integrator has been changed to a note, since there are valid use cases for using normal MD when not running in the completely decoupled state.

Issue 2767

## 11.10 GROMACS 2019 series

### 11.10.1 Patch releases

#### GROMACS 2019.6 release notes

This version was released on February 28th, 2020. These release notes document the changes that have taken place in GROMACS since the previous 2019.5 version, to fix known issues. It also incorporates all fixes made in version 2018.8 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Actually fix PME forces with FE without perturbed q/LJ

PME would incorrectly ignore the mesh forces on perturbed atoms when no charges or LJ atom types were actually perturbed. Note that this is a rather uncommon scenario.

Issue 2640 Issue 3359

#### Avoid overzealous program abort with orientation restraints

It could happen that mdrun would abort on checking orientation restraints in multiple molecules even though no restraints where applied to them.

Issue 3375

#### Calculate Coulomb and LJ reciprocal terms in rerun

Reruns would not calculate Coulomb and LJ reciprocal terms, leading to wrong potential energies. This bug only showed up if GROMACS was compiled without GPU support.

Issue 3400

### Fixes for `gmx` tools

Added check for inconsistent input of distance restraint labels in gmx disre.

### Fixes that affect portability

#### Fix compiler errors with Intel compiler

Fix compiler error with Intel compiler 2019 update 5 and 2020 initial release. Compilation was failing with `mcpcom: core dumped` for the file `pullutil.cpp`.

### Miscellaneous

#### Avoid cryptic GPU detection errors when devices are unavailable or out of memory

Issue 3178 Issue 3399

### GROMACS 2019.5 release notes

This version was released on December 23rd, 2019. These release notes document the changes that have taken place in GROMACS since the previous 2019.4 version, to fix known issues. It also incorporates all fixes made in version 2018.8 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Fix use of uninitialized data on PME only ranks

When building GPU enabled versions of GROMACS with clang as either host only or host and device side compiler, PME datastructures could be left uninitialized, leading to the use of random values for LJ PME energies, virial and pressure.

The effect of this bug was that the potential and total energy could be wrong, but not the Coulomb mesh energy. This didn't affect sampling. The pressure could also be wrong, which would affect sampling when pressure coupling is used, but likely the simulation would explode after a few steps.

This doesn't seem to have affected versions of GROMACS built with gcc as the host side compiler.

Issue 3120

#### Fix out of range memory access with free-energy calculations

With free-energy calculations not using lambda states, an output buffer would be accessed one element beyond it's allocated size. We don't expect this to have caused incorrect results, but a memory checker would complain.

Issue 3173

#### Work around broken Apple Clang compiler in Mac OS Catalina

In Mac OS Catalina, the default XCode compilers checks and enforces stack alignment. This would have been a good idea if Apple itself did not ship a C library that violates the stack alignment with AVX instructions are enabled.

Issue 3199

#### Fix error with intermolecular interactions and domain decomposition

With intermolecular interactions at distances longer than the cutoff and domain decomposition, mdrun could exit with an error message about missing interactions.

Issue 3204

### Fix issues with AWH with pull-geometry 'direction' to be periodic

Removed fatal error with AWH with periodic pull-geometry 'direction' when the distance was within 2% of half the box size. Changed an assertion failure when the AWH interval was larger than the box size to a fatal error. Clarified the documentation for pull geometry 'direction-periodic'.

Issue 2946

### Remove assertion failure with AWH when not using the initial stage

Issue 3217

### Fixes for `gmx` tools

### Make histogram output clearer

Output gave number of events included in histogram bar as *a.u.*, which was not clear for users.

### Fix dihedral angle calculation near 180 degree boundary

The analysis tools could incorrectly calculate properties of torsion angles and their averages when close to the -180 or 180 degree boundary.

Issue 3225

### Remove problematic output of gmx angle tool

It could happen that the calculation of the standard deviation for angles caused a divide by zero error for empty populations. Because this standard deviation was meaningless, it has been removed.

Issue 3206

### Fixes that affect portability

### Check that libhwloc headers and runtime match

It could happen that the libhwloc headers and library detection would lead to a mismatch at compile or runtime that could cause cryptic crashes while using mdrun.

Issue 3200

### Miscellaneous

#### Fix .gro file formatting with large boxes

The GROMACS manual says the box components in the .gro file format are separated by spaces. But no space was printed when a box component, except for the first, was 1000 nm or larger or an off-diagonal component was -100 nm or smaller. Now at least one space is always printed. Content that was written in a way that already had at least one space between components is unchanged. Existing parsers that conform to the documentation and expect whitespace separation will continue to work in all cases.

Issue 3176

#### Fix duplicate PDB CONECT record output

PDB "CONECT" record output was duplicated in some instances. Since GROMACS does not use these anywhere, analysis was not affected. The behavior is now fixed.

Issue 3206

#### Fix performance issue with bonded interactions in wrong GPU stream

This could lead to a significant loss in performance.

Issue 3241

### GROMACS 2019.4 release notes

This version was released on October 2nd, 2019. These release notes document the changes that have taken place in GROMACS since the previous 2019.3 version, to fix known issues. It also incorporates all fixes made in version 2018.7 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fix incorrect pressure when atoms in CMAP cross a box boundary

The virial calculation and thus the pressure would be incorrect when the second and third atom involved in a CHARMM CMAP correction term would reside in different periodic images. This can happen when a protein is positioned over a box boundary. Note that the energy and forces were correct, but sampling was affected when pressure coupling was applied when a protein crossed a box boundary.

Issue 2845 Issue 2867

### Fix incorrect LJ cut-off on GPU when rvdw < rcoulomb

When rvdw was chosen by the user to be smaller than rcoulomb in the mdp file, the LJ cut-off would initially be set to the Coulomb cut-off for computing non-bonded interactions on the GPU. This only affected energy minimization, mdrun -rerun and the first 2*nstlist steps of a normal MD run, since the correct LJ cut-off is set when PME tuning (on by default) starts after 2*nstlist steps (unless PME tuning was disabled with -notunepme).

Issue 3056

### Fix (unlikely) missing bonded forces with CUDA GPUs and domain decomposition

Forces could be missing for bonded interactions computed on CUDA GPUs with domain decomposition when there are non-local bonded interactions, but no non-local non-bonded interactions between two domains. Note that this is extremely unlikely to happen, since the distance between the bonded atoms needs to be larger than the pair-list cut-off distance and there should be no other non-local atoms within the pair-list cut-off distance.

Issue 3063

### Fix incorrect reporting of final kinetic energy and temperature

With the leap-frog integrator, the kinetic energy and temperature reported for the last step were incorrect when the last step was not divisible by nstcalcenergy, nsttcouple or nstpcouple.

Issue 2950

### Fix segmentation fault in grompp and mdrun with cosine COM pulling

Issue 3023

### Fixes for `gmx` tools

### Fix grompp not adding angle constraints between constraints

When using the mdp option constraints=all-angles, angles involving bonds supplied as constraints in the topology would be removed, but not replaced by angle constraints.

Issue 3067

### Fix gmx wham with angle and dihedral geometries

gmx wham would apply an incorrect radian to degree unit conversion, leading to no overlap or not-a-number output.

Issue 2609 Issue 3094

### Fix bug in gmx xpm2ps

The tool would fail when not being provided with a library file to read in.

Issue 3012

### Fix bug in gmx anaeig

An issue was noted when reading a second set set of eigenvectors that could lead to problems when the number of eigenvectors was less than the three times the number of atoms.

Issue 2972

### Fix issue with demux.pl script

The trajectories could become discontinuous with simulations longer than 100ns and exchange strides that are not a multiple of 1 ps. This only affected the post-processing of trajectories generated from replica exchange simulations.

### Made gmx disre work with non-consecutively labeled restraints

Issue 2953

### Fixed writing of gro files with index groups

An output `.gro` file from from e.g. `gmx editconf -f -n` would take the atom names for the output file in order from the atoms in the input file, rather than in order from the atoms indicated by the indices in the index file.

Issue 3107

### Made gmx make_ndx keep chain IDs

Old style structure file reading caused the chain IDs to be overwritten with default values.

Issue 3070

### Fixes that affect portability

#### Disable PME OpenCL on Apple

The Apple OpenCL compilers fail to produce a functional clFFT build. The OpenCL PME support is therefore disabled on Apple platforms.

Issue 2941

### Miscellaneous

#### Added AMD Zen 2 detection

The AMD Zen 2 architecture is now detected as different from Zen 1 and uses 256-bit wide AVX2 SIMD instructions (GMX_SIMD=AVX2_256) by default. Also the non-bonded kernel parameters have been tuned for Zen 2. This has a significant impact on performance.

## GROMACS 2019.3 release notes

This version was released on June 14, 2019. These release notes document the changes that have taken place in GROMACS since the previous 2019.2 version, to fix known issues. It also incorporates all fixes made in version 2018.7 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fix missing interactions with domain decomposition

When running with domain decomposition, any interactions described by the rarely-used topology file section `[ intermolecular_interactions ]` were ignored. This did not affect normal non-bonded or intra-molecular interactions.

Issue 2953

#### Fix possible floating point exception during minimization.

It was possible that very small forces during minimization could lead to a crash due to a divide by zero error. Fixed by introducing a check.

Issue 2917

### Fix segmentation fault when using membrane embedding

Issue 2947

### Allow AWH with pull-geometry 'direction' to be periodic

When applying AWH to a pull coordinate with geometry 'direction' with a AWH interval length of more than 95% of the box size, the dimension is now made periodic.

Issue 2946

### Fixes for `gmx` tools

### Fixed residue and molecule indexing in selections

Issue 2951

### Fix PQR formatting

The formatting was incorrect for some tools that use PQR files.

Issue 2955

### Fix gmx wham with angle geometries

gmx wham would mix up degree and radian units leading to no overlap or not-a-number output. **Note**: this fix is not correct, a correct fix is applied in the next patch release.

Issue 2609 Issue 3094

### Add some information for grompp error with wrong line endings

### Give meaningful error with too large grid in hbond

When using a grid that is too large *hbond* (page 237) could try to allocate enough memory to cause a crash.

Issue 2962

### Add some information for syntax errors with include delimiters in grompp

Issue 2911

### Fixes that affect portability

### Fixed wider reference SIMD setups

The reference SIMD builds could use a too small memory alignment, leading to mdrun exiting with an alignment error

Issue 2952

### Fixed build failure with Apple Clang

Builds would fail because of qsort being undefined.

### Miscellaneous

### Removed non-existent mdp option awh1-dim1-period from user guide

Issue 2940

### Add checks for too many interactions during memory allocation

Issue 2932

### GROMACS 2019.2 release notes

This version was released on April 16th, 2019. These release notes document the changes that have taken place in GROMACS since the previous 2019.1 version, to fix known issues. It also incorporates all fixes made in version 2018.6 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Fix L-BGFS minimizer

The minimizer could fail on a number of systems.

Issue 2641

### Disallow pull geometry direction-periodic with AWH

This could lead to incorrect behavior or a cryptic error message.

Issue 2923

### Fixed mdrun -nsteps option

Fixed that the, deprecated, mdrun option -nsteps only allowed extension of the simulation under certain conditions.

Issue 2881

### Fixes for `gmx` tools

### gmx cluster -clndx indices now correct

The reported indices of trajectory frames in clusters were too small by one.

Issue 2926

### gmx editconf -f in.pdb -o out.pdb again preserves chain IDs

This had been inadvertently broken and is now fixed.

Issue 2900

### Tools again accept .tpr files as input

The pdb2gmx, solvate, and insert-molecules tools could no longer accept input configurations contained in .tpr format files. This is now fixed.

Issue 2900

### Fix segmentation fault when preparing simulated annealing inputs

grompp was unable to prepare tpr files for inputs containing simulated annealing procedures. The code has been fixed to allow the generation of those files again.

Issue 2871

### Fixes that affect portability

#### Fix error in AVX 512 detection code

The CMake detection code had a typo that could lead to wrong detection results.

### Miscellaneous

#### Added warning with the use of GROMOS force fields

grompp now warns when a GROMOS force field is used. The GROMOS force fields have been parametrized with a physically incorrect multiple-time-stepping scheme for a twin-range cut-off. When used with a single-range cut-off, physical properties, such as the density, might be off from the intended values.

Issue 2884

#### Prevented internal build of FFTW with clang and AVX-512 SIMD

Prevented the internal build of FFTW with clang from attempting to configure FFTW to compile with AVX-512 support. That SIMD level is not supported by FFTW with the clang compiler, and compilation fails.

Issue 2892

#### Updated performance guide for recent Intel processors with AVX512 instruction support

Noted the tradeoffs between CPU frequency and SIMD throughput and advising users to prefer AVX2 over AVX512 in GPU-offload or highly parallel MPI cases.

#### Updated release notes for 2019.1

A *fix* (page 1011) made to GPU kernels in 2019.1 was thought to resolve Issue 2845 but further investigation suggests that the real cause is not yet known.

#### GROMACS 2019.1 release notes

This version was released on February 15, 2019. These release notes document the changes that have taken place in GROMACS since the initial version 2019, to fix known issues. It also incorporates all fixes made in version 2018.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fix error with 2D/3D dynamic load balancing

With 2D or 3D domain decomposition with dynamic load balancing, mdrun would exit with the error "The domain decomposition grid has shifted too much .." when a cell size was limited.

Issue 2830

#### Fix incorrect LJ repulsion force switching on GPUs

When using a CUDA or OpenCL GPU, the coefficient for the second order term for the LJ repulsion in the force switching function, called 'A' in the manual, had the wrong sign. This lead to very small errors in the forces and the pressure. Note that the dispersion force switching was correct. The effects of this bug on any physical results seems to be negligible. Note that force switching is usually only used in combination with the CHARMM force field.

Issue 2845

#### Fix segmentation fault in mdrun with domain decomposition

Issue 2813

#### Fix segmentation fault with energy minimization with the group scheme

Using energy minimization in combination with the group cutoff scheme and domain decomposition could lead to a segmentation fault.

Issue 2813

#### Correct free-energy Delta H output with mass lambda's

When separate lambda parameters were used for perturbed mass free-energy contributions, these contributions were double counted in the Delta H output used for BAR calculations. Note that dH/dlambda was always correct

Issue 2703 Issue 2849

#### Prevent mdrun -rerun from writing incorrect free-energy output

Now mdrun -rerun exits with a fatal error when masses or constraints are perturbed. Their contributions to Hamiltonian differences and derivatives were incorrectly set to zero in version 2019.

Issue 2849

### Fix possible division by zero in enforced-rotation code

Issue 1431

### Fixes for `gmx` tools

### Fix trjconv -ndec

This only works for writing .xtc files. The code and documentation now works correctly with .gro files, which was changed in 2016 release series so that it would only write fixed-width columns.

Issue 2824 Issue 2037

### Fix using index file groups when .tpr file not supplied

Selections that use groups from a supplied index file can again be used even when a .tpr file is not supplied.

Issue 2847

### Fix tune_pme

The tool did not work due to a file reading error that is fixed now.

Issue 2827

### Fixes that affect portability

### With MSVC, disabled internal clFFT fallback used for OpenCL support

GROMACS requires MSVC 2017, and the GROMACS OpenCL build requires clFFT. If clFFT is found on the user's system, then all may be well, but the version of clFFT bundled within GROMACS cannot be built because only MSVC 2010 is supported by clFFT at this time. A configure-time fatal error is now issued in this case.

Issue 2500

### Explicitly require 64-bit platforms for OpenCL

A 64-bit OpenCL runtime is required by GROMACS. All known OpenCL implementations on 64-bit platforms are 64-bit (and there are no known 32-bit platforms with 64-bit OpenCL), hence we require a 64-bit platform at configure-time in OpenCL builds. A known unsupported 32-bit platform is ARMv7.

#### Miscellaneous

#### Improved docs for applying electric fields

### 11.10.2 Major release

#### Highlights

GROMACS 2019 was released on December 31st, 2018. Patch releases may have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

As always, we've got several useful performance improvements, with or without GPUs, all enabled and automated by default. We are extremely interested in your feedback on how well this worked on your simulations and hardware. They are:

- Simulations now automatically run using update groups of atoms whose coordinate updates have only intra-group dependencies. These can include both constraints and virtual sites. This improves performance by eliminating overheads during the update, at no cost.
- Intel integrated GPUs are now supported with OpenCL for offloading non-bonded interactions.
- PME long-ranged interactions can now also run on a single AMD GPU using OpenCL, which means many fewer CPU cores are needed for good performance with such hardware.

#### New and improved features

#### GROMACS build is now more reproducible

The build system no longer embeds information about who built the binary and where. We used to include this information to help troubleshoot problems and ensure checkpoint continuations are exact where possible, but this does not seem necessary. This makes the build closer to `reproducible by default` which is useful for projects that offer distributions of reproducible software, including GROMACS.

#### Update gmx cluster to write correct PDB files and index files with cluster frames

*PDB* (page 581) files from gmx cluster were missing the CRYST header for box information, making it more difficult than needed to use them with our GROMACS tools. Also, the *index* (page 580) files needed for *gmx trjconv* (page 334) to split up trajectories into frames corresponding to the clusters were not written. This adds support for writing this *index* (page 580) file as well as proper *PDB* (page 581) files.

### Allow using COM of previous step as PBC reference

Added an option (`pull-pbc-ref-from-prev-step-com`), when pulling, to use the COM of the group of the previous step, to calculate PBC jumps instead of a reference atom, which can sometimes move a lot during the simulation. With this option the PBC reference atom is only used at initialization. This can be of use when using large pull groups or groups with potentially large relative movement of atoms.

### Transitional external API headers and library

Library access to GROMACS is transitioning to new infrastructure. gmxapi 0.0.7 provides abstractions for execution environment and simulation work, as well as development tools for extending MD simulation code without patching the GROMACS source. Client code may be built against a GROMACS installation. MD plugin code may apply externally calculated forces (see restraint module) or issue simulation stop signals through session resources available at run time to registered plugins. For more project information and use cases, refer to the tracked Issue 2585 and to DOI 10.1093/bioinformatics/bty484. For a few examples of building on and extending GROMACS, refer to the Python package and sample restraint plugin repository.

### Restraint module for gmxapi MD extension code

Provides functionality that was previously accessed by modifying the "pull" code in the GROMACS source. Client software may be built against an unmodified GROMACS installation. Separately compiled MD extensions can be registered with the new Restraint functionality at run time using simulation client code built with the new `gmxapi` tools. (See above.)

### Enable output of average pull forces and positions

Normally the pull module writes instantaneous output of positions and forces, however now it can write the average of these values over the period since the last output. This works correctly even if a checkpoint restart intervened. This is enabled using the new options `pull-fout-average` and `pull-xout-average`.

## Performance improvements

### Implemented update groups

Domain decomposition can now be based on so-called update groups. These are groups of atoms with dependencies during the update, which can be constraints and virtual sites. Update groups can typically be used when only bonds involving hydrogens are constrained and are enabled automatically when possible. This improves performance by eliminating MPI and OpenMP communication for constraints and virtual sites.

### PME on GPU when running free energy perturbations not involving charges

PME can now be run on a GPU when doing free energy perturbations that do not involve perturbing charges.

### PME long-ranged interaction GPU offload now available with OpenCL

On supported devices from all supported vendors (AMD, Intel, NVIDIA), it is now possible to offload PME tasks to the GPU using OpenCL. This works in the same way as the former CUDA offload. A single GPU can now be used to accelerate the computation of the long-ranged PME interactions. This feature means that only 2-4 CPU cores per GPU will be about as fast as the 2018 version that needed many more CPU cores to balance the GPU. Performance on hardware that had good balance of GPU and CPU also shows minor improvements, and the capacity for hardware with strong GPUs to run effective simulations is now greatly improved.

### Intel integrated GPUs are now supported for GPU offload with OpenCL

On Intel CPUs with integrated GPUs, it is now possible to offload nonbonded tasks to the GPU the same way as offload is done to other GPU architectures. This can have performance benefits, in particular on modern desktop and mobile Intel CPUs this offload can give up to 20% higher simulation performance.

### Bonded interactions are now supported for CUDA GPU offload

Common types of bonded and LJ-14 interactions found can now run on NVIDIA GPUs with CUDA, with and without domain decomposition. Interactions with perturbed parameters are not supported.

### Added code generation support for NVIDIA Turing GPUs

With CUDA 10.0 NVIDIA Turing GPUs can be directly targeted by the nvcc compiler. We now generate the appropriate set of flags for the Turing architecture by default when using CUDA 10 (or later).

### Improvements to GROMACS tools

### pdb2gmx writes total charge differently

pdb2gmx notes the total charge for each residue in the `[atoms]` field of the topology file it produces. The fact that this should generally be an integer can be used for troubleshooting issues in system or force field preparation. This printing is now done only once per residue, rather than for every atom.

### nmeig does thermochemistry

The nmeig tool that analyzes the Hessian matrix from a normal mode analysis now generates thermochemical properties like standard entropy, heat capacity at constant volume, internal thermal energy and zero-point energy. The analysis is based on the harmonic approximation that is the same as what is used in quantum chemistry.

### Implement writing of LaTeX methods in report-methods

Added a new tool report-methods to write a summary of methods used to set up a simulation to an output file or to standard output.

## Bugs fixed

### Fix type bug in triclinic DD code

Fix bug with unusual off-diagonal elements communicating too few atoms.

### Ensure domains are large enough for atom motion

Domain decomposition now makes sure that domains will always be large enough so that atoms will not move across additional domains.

Issue 2614

### Velocity Verlet integrators output energy averages from correct steps

Velocity Verlet integrators would accumulate energies for writing averages to the energy file when step-1 was a multiple of nstcalcenergy. This has now been corrected to step being a multiple of nstcalcenergy. Note that although this (slightly) changes the reported averages, the averages were not incorrect.

Issue 2718

### Fix chainsep behaviour of pdb2gmx

Issue 2577

### grompp correctly checks nstexpanded against nstcalcenergy

With expanded ensemble, but without free-energy perturbation, grompp would not check if nstexpanded was a multiple of nstcalcenergy. If the latter was not the case, results might have been incorrect.

Issue 2714

### Issue with do_dssp and unknown residues

The do_dssp tool would fail with unknown residues, as well as have issues on Windows.

Issue 2599

### Deprecated functionality

### Changes anticipated to GROMACS 2019 functionality

#### `gmx mdrun -membed`

The feature for embedding a protein in a membrane will be retained, but probably in a different form, such as `gmx membed`.

#### `gmx mdrun -rerun`

The feature for computing potential energy quantities from a trajectory will be retained, but probably in a different form, such as `gmx rerun` and `gmx test-particle-insertion`.

### Integrator .mdp options will only contain dynamical integrators

Energy minimization will be accessed in a differt form, perhaps with `gmx minimize` and interpret an .mdp field for which minimizer to use. Normal-mode analysis may be accessed with e.g. `gmx normal-modes`. The command-line help for these tools will thenx be better able to document which functionality is supported when.

### Much functionality in `trjconv`, `editconf`, `eneconv` and `trjcat`

The functionality in such tools is being separated to make it available in composable modules, that we plan to make available as simpler tools, and eventually via the GROMACS API that is under development.

### `gmx do_dssp` to be replaced

This tool is deprecated, because it is problematic for some users to obtain and install a separate DSSP binary, so we plan to replace the implementation at some point with a native implementation, likely based upon xssp, and make it available under a new gmx tool name.

### Functionality deprecated in GROMACS 2019

#### Generation of virtual sites to replace aromatic rings in standard residues

Issue 3254 These are thought to produce artefacts under some circumstances (unpublished results), were never well tested, are not widely used, and we need to simplify pdb2gmx.

#### `gmx mdrun -gcom`

This feature sometimes overrides the effects of various .mdp settings in a way that is difficult to understand and report. A user who wants to do communication between PP ranks less often should choose their `nst*` mdp options accordingly.

#### Benchmarking options only available with `gmx benchmark`

Issue 3255 Options such as `-confout`, `-resethway`, `-resetstep` are not intended for use by regular mdrun users, so making them only available with a dedicated tool is more clear. Also, this permits us to customize defaults for e.g. writing files at the end of a simulation part in ways that suit the respective mdrun and benchmark use cases, so `-confout` will no longer be required.

#### `gmx mdrun -nsteps`

Issue 3256 The number of simulation steps described by the .tpr file can be changed with `gmx convert-tpr`, or altered in .mdp file before the call to `gmx grompp`. The convenience of this mdrun option was outweighted by the doubtful quality of its implementation, no clear record in the log file, and lack of maintenance.

### Functionality deprecated before GROMACS 2019

This functionality has been declared as deprecated in previous versions of GROMACS, but has not yet been removed.

#### The group cutoff scheme

All remaining aspects of the group cutoff scheme will be removed, once a few remaining features (e.g. tabulated interactions, energy-group exclusions, and vacuum simulations) are available with the Verlet scheme. Deprecated in GROMACS 5.1

#### QM/MM support for ORCA, GAMESS and MOPAC

These interfaces are untested, and no maintainer has been found for them. Deprecated in GROMACS 2018.

### Removed functionality

#### NVML support removed on NVIDIA GPUs

NVML support (for reporting GPU application clocks or changing these for higher throughput) is no longer available. It was only ever supported on high-end hardware and changing clocks is on recent generations of hardware only useful when root permissions were available to the user. It may become less useful as GROMACS evolves, complicated the GROMACS code, and wasn't regularly tested or maintained. It might return if some of these conditions change.

#### Support for CUDA compute capability 2.x removed

The Fermi-era GPUs (cira 2010) are no longer in widespread use, are not tested in Jenkins, complicated the code, and are no longer supported.

#### Contrib directory removed

This code had not been maintained in years, so likely didn't work, and has been removed. The git history retains its memory if anybody needs it.

#### BlueGene support removed

As most of these machines are now retired, and the ports have not been actively maintained since GROMACS 5.1, the support for BlueGene and QPX SIMD has been removed.

#### Implicit solvent support removed

Since GROMACS-4.6, the SIMD and multi-threading support has been mostly broken. Since nobody wants to fix it, the feature has been removed. Old force field files with parameters for such simulations can still be read, but the values are ignored.

#### Removed `gmx mdrun -multi`

The implementation of `gmx mdrun -multidir` is more reliable and works with more features. Nobody was willing to maintain the duplicate functionality.

### Portability

#### Increased the minimum CUDA version required

We now require CUDA 7.0, whose features help keep the code more maintainable.

#### Increased the minimum MSVC version required

We now require MSVC 2017, so we can rely on full C++11 support and the highest quality implementations. On this platform, we now also require CUDA 9.0.

#### Updated the OpenCL requirement to version 1.2

We now require at least OpenCL version 1.2 both for API and kernels. All currently targeted vendors' libraries do support it, so this is not a restriction in any way.

#### Preliminary support for ARM Performance Libraries

The ARM Performance Libraries can now be used for FFT transforms through the FFTW compatiblity layer. This can provide performance benefits over using a vanilla FFTW3 on recent ARMv8 architectures.

### Miscellaneous

#### grompp discourages use of constraints=all-bonds

Common force fields, including AMBER, CHARMM and OPLS/aa, are parametrized with bonds involving hydrogen constrained. Constraining all bonds should be avoided, for correctness. grompp now issues a note when constraints=all-bonds is used with these force fields when time steps are smaller than 2.6 fs and hydrogens are not replaced by virtual sites. Using constraints=h-bonds will also improve performance.

#### Documentation changed to reStructuredText

The complete documentation has been moved to the reStructuredText markup format to allow building it together for display as html or as pdf.

## 11.11 GROMACS 2018 series

### 11.11.1 Patch releases

#### GROMACS 2018.7 release notes

This version was released on May 29, 2019. These release notes document the changes that have taken place in GROMACS since version 2018.6, to fix known issues. It also incorporates all fixes made in previous versions, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Reverted broken change made in 2018.1

Reverted a change made in 2018.1 that broke simulations that used the SHAKE constraint algorithm.

Issue 2879

#### Work around gcc 7 AVX512 compiler bug

With gcc version 7 a compiler bug caused a large part of non-bonded interactions to be ignored when compiled with AVX512 and running on more than 16 OpenMP threads.

Issue 2762

#### Fixes for `gmx` tools

#### Fixes to improve portability

#### Miscellaneous

#### Updated release notes for 2018.6

A *fix* (page 1022) made to GPU kernels in 2018.6 was thought to resolve Issue 2845 but further investigation suggests that the real cause is not yet known.

#### GROMACS 2018.6 release notes

This version was released on February 22, 2019. These release notes document the changes that have taken place in GROMACS since version 2018.5, to fix known issues. It also incorporates all fixes made in previous versions, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Correct free-energy Delta H output with mass lambda's

When separate lambda parameters were used for perturbed mass free-energy contributions, these contributions were double counted in the Delta H output used for BAR calculations. Note that dH/dlambda was always correct

Issue 2703 Issue 2849

#### Fix incorrect LJ repulsion force switching on GPUs

When using a CUDA or OpenCL GPU, the coefficient for the second order term for the LJ repulsion in the force switching function, called 'A' in the manual, had the wrong sign. This lead to very small errors in the forces and the pressure. Note that the dispersion force switching was correct. The effects of this bug on any physical results seems to be negligible. Note that force switching is usually only used in combination with the CHARMM force field.

Issue 2845

### Fixes for `gmx` tools

### Fixes to improve portability

#### Fix compiler flags for Power8

A compiler flag for Power8 processors lead to errors in the code and was removed.

Issue 2747 Issue 2746 Issue 2734

### Miscellaneous

## GROMACS 2018.5 release notes

This version was released on January 22, 2019. These release notes document the changes that have taken place in GROMACS since version 2018.4, to fix known issues. A complete list of fixes and their descriptions can be found in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

#### Fixed numerical derivative for normal-mode analysis with shell code

Due to higher precision needs when using polarizable shell particles the normal mode code did not work reproducibly with shells. In order to fix this the step-size used for numerically computing the Hessian was reduced to near machine precision. The change does not affect the results for non-polarizable systems, such as proteins or small molecules.

### Make large PME grids work on GPU

PME grids with size along Z larger than 511 would make mdrun exit with a cryptic CUDA error.

Issue 2779

### Fix LINCS accuracy with OpenMP when constraint triangles are present

Constraint triangles, which usually only occur when replacing hydrogens by virtual interaction sites in CH3 and NH3 groups, need double the number of iterations as normal constraints. With OpenMP this would only happen when the last OpenMP thread has at least one such triangle. This would cause a slight loss of accuracy in inhomogeneous systems.

Issue 2808

### Fix acceleration with `cos-acceleration`

A factor of 2 was missing from the acceleration value, leading to incorrect results when e.g. calculating viscosities.

Issue 2572

### Fix checkpoint restart of tpr with infinite step count

An issue was introduced that caused *mdrun* (page 258) to refuse to start when using infinite step counts. Now *mdrun* (page 258) properly accepts those files again.

Issue 2757

### Fix energy history file reading

A check was missing when reading energy files from a simulation that was stopped before the first value had been written to the file.

Issue 2781

### Fixes for `gmx` tools

### Fix trjconv -ndec

This only works for writing .xtc files. The code and documentation now works correctly with .gro files, which was changed in 2016 release series so that it would only write fixed-width columns.

Issue 2813 Issue 2037

### Fixes to improve portability

### Miscellaneous

### Warn for problematic coupling times with Nose-Hoover and Parrinello-Rahman

When combining the Nose-Hoover and Parrinello-Rahman coupling algorithms, resonances in the kinetic energy and pressure/volume can appear when the two coupling times involved are similar. Now grompp warns when `tau-p` is less than two times `tau-t`.

Issue 2749

### Fixed efficiency issue with shell code minimization

Code cleanup touching unnecessarily complex code created an efficiency issue. Both the issue and some of the complexity are now fixed.

Issue 2705

### Added code generation support for NVIDIA Turing GPUs

With CUDA 10.0 NVIDIA Turing GPUs can be directly targeted by the nvcc compiler. We now generate the appropriate set of flags for the Turing architecture by default when using CUDA 10 (or later).

### GROMACS 2018.4 release notes

This version was released on November 12, 2018. These release notes document the changes that have taken place in GROMACS since version 2018.3, to fix known issues. It also incorporates all fixes made in version 2016.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Correct PME forces with free energy without perturbed charges/LJ

With free-energies calculations with lambda not set to zero and no actual perturbed charges or atom types for Lennard-Jones, the Coulomb or LJ PME mesh forces would be scaled with lambda. Note that this bug did not affect the, usual, setup where charges or atom types are actually perturbed.

Issue 2640

### Add constraint contribution to foreign Hamiltonian differences

The contribution of perturbed constraints was missing from the foreign Hamiltonian values. This is important for free energy calculations, such as BAR.

Issue 2703

### Add mass contribution to foreign Hamiltonian differences

For free energy calculations with perturbed masses, the kinetic energy contribution was missing from the foreign Hamiltonian values.

Issue 2703

### Work around bugs with expanded ensemble runs

With expanded ensemble runs, the energies would be outdated or zero with the velocity Verlet integrator with nstcalcenergy>1 or with other integrators when nstexpanded was not a multiple of nstcalcenergy. In these cases mdrun now sets nstcalcenergy to 1.

Issue 2714 Issue 2718

### Checkpoint continuations require suitable .tpr files

The step of a checkpoint file used for an mdrun restart must now be less than the number of steps in the .tpr. Formerly, the step in the checkpoint could be any number, and mdrun -nsteps could be used to get a particular result, but the use of that option is already deprecated. Use gmx grompp or gmx convert-tpr to make a .tpr file that expresses the intent.

Issue 2717

### Fixes for `gmx` tools

### Fix mindist output file checks

mindist would not check if the output file needed to print residue names and residue contacts over time was actually defined, leading to errors with empty file name strings.

Issue 2653

### Fix gmx helix segmentation faults

The .tpr file is now read correctly, and the helix analysis correctly handles selections that include proline residues.

Issue 2701

### Fix bug in entropy calculation in gmx anaeig

When gmx anaeig received an inconsistent number of atoms and eigenvectors (fewer eigenvectors than three times the number of atoms) the entropy calculations would use uninitialized values.

Issue 2668

### Fixes to improve portability

### Miscellaneous

Fixed an issue where the log file could sometimes report an incorrect initial dynamic load balancing state

Issue 2631

### Fix Bromine parameters in amber forcefield files

The forcefield entries for Bromine and Iron were missing the actual values to define sigma and epsilon. The proper values have been included from parm99.dat for Bromine. As Iron has no corresponding parameters, the entry has been removed.

Issue 2711

### Made normal-mode analysis work for more than one molecule

Fixed an issue where normal mode analysis would only consider the first copy of each molecule in a system. Also fixed issues with vsites or shells in normal modes.

Issue 2720

### Disallow rerun using same filename as output file

When using identical filenames for `-rerun` a cryptic error was thrown because the same file would be used for reading and writing. Now *mdrun* (page 258) will give a helpful error message to get around this.

Issue 2634

### Fix issue when building GROMACS without TNG

Some compiler errors have been resolved that could show when building GROMACS without TNG support enabled.

### GROMACS 2018.3 release notes

This version was released on August 23, 2018. These release notes document the changes that have taken place in GROMACS since version 2018.2, to fix known issues. It also incorporates all fixes made in version 2016.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Multi-domain GPU runs can no longer miss pair interactions

With systems with empty space in the unit cell, GPU runs with domain decomposition would not compute LJ and Coulomb interactions between domains when there we no interactions between domains on a rank at some point in time.

- This bug only affects simulations running on GPUs with domain decomposition and containing empty regions of space that can lead to domains being empty.
- Possible observations of this bug may have been seemingly random failures of calculations that where not reproducible when restarting a simulation from a checkpoint file, as the domain would then again be filled properly if interactions are present at the beginning.
- It is unlikely that this bug will have unnoticed effects on all but very short simulations, as the missing interactions will inevitable lead to simulation instability and crashes.
- If a simulation that crashed due to this bug is restarted it can contain a small region around the crash that will be unphysical due to some interactions not being calculated just before the crash itself.

**This is a critical fix and users of 2018.x series that run on GPUs should update to this point release**

Issue 2502

### Fix Conjugate Gradient assertion failure at end of minimization

When the final step coincided with a coordinate output step, conjugate gradient minimization would exit with an assertion failure instead of writing confout.gro.

Issue 2554

### Multi-domain Conjugate Gradient minimimization no longer segfaults.

Issue 2554

### Fix pairlist buffer with Brownian Dynamics

With Brownian Dynamics and bd-fric > 0, the Verlet pairlist buffer would be determined with incorrect masses for constrained atoms and virtual sites. This would lead to a too small buffer for typical atomistic systems with constraints.

Issue 2613

### Avoid "atom moved to far" errors

The introduction of the dual pair list has led to larger nstlist values, which leads to larger atom displacements between domain decomposition steps. This has made it more likely that the errors "An atom moved too far between two domain decomposition steps" and "N particles communicated to PME rank M are more than 2/3 times the cut-off out of the domain decomposition cell ..." appear for stable systems. Now atom displacements are correctly taken into account when determining the minimum cell size, so these errors should only appear for unstable systems.

Issue 2614

### grompp now checks that pull groups are not close to half the box size

Pull groups that use a reference atom for periodic boundary treatment should have all their atoms well within half the box size of this reference. When this is not the case, grompp will issue a warning.

Issue 2397

### Fixed segmentation fault in mdrun with QM/MM ONIOM scheme

Issue 2617

### Correctly specified that PME on GPUs is only supported for dynamical integrators

Previously PME on GPU support could run (but fail) for energy minimization and normal-mode analysis runs.

Issue 2578

### Fixes for `gmx` tools

### Fixed syntax error in make_gromos_rtp.py

Issue 2557

### Fix gmx solvate topology updating

Removed hard coded solvent names to allow updates to topology based on solvent molecule information. Also allows updating with multiple solvent types.

Issue 1929

### Fix bfactor output error caused by fix for Issue 2511

The fix for the PQR file output broke the output of bfactors from other tools.

Issue 2575

### Made sure that gmx rms can skip values

When requested to skip values, gmx rms would still output all values despite the option. Now it only outputs values that are requested to be processed.

Issue 2565

### Fix trjconv when not providing structure file

trjconv would fail with a segmentation violation when running without any structure file due to incomplete initialization of the topology data structure. This fix adds the missing checks that prevents the failure.

Issue 2619

### Fix enforced rotation energy output

### Fixes to improve portability

### Fix nvcc host compiler check triggering

Issue 2583

### Report up to date hwloc version information

Issue 2591

### Disable single compilation unit with CUDA 9.0

Issue 2561

### Miscellaneous

### Avoid aborting mdrun when GPU sanity check detects errors

Issue 2415

### Improve OpenCL kernel performance on AMD Vega GPUs

The OpenCL kernel optimization flags did not explicitly turn off denorm handling which could lead to performance loss. The optimization is now explicitly turned on both for consistency with CUDA and performance reasons. On AMD Vega GPUs (with ROCm) kernel performance improves by up to 30%.

## GROMACS 2018.2 release notes

This version was released on June 14, 2018. These release notes document the changes that have taken place in GROMACS since version 2018.1, to fix known issues. It also incorporates all fixes made in version 2016.5 and earlier, which you can find described in the *Release notes* (page 838).

### Fixes where mdrun could behave incorrectly

### Prevented OpenCL timing memory leak

When using OpenCL builds and timing, a memory leak would lead to all system memory being used up.

Issue 2470

### Fixed MPI error after constraint failure during energy minimization

Issue 2540

### Fixed moving frozen atoms with constraints

Frozen atoms which also had bond constraints could move.

Issue 2542

### Fixed COM removal moving frozen atoms

When frozen atoms were part of center of mass motion removal groups, they could accumulate momentum and move.

Issue 2551

### Fixed AWH too infrequent checks for covering

For multidimensional AWH grids with many points relative to the number of samples required for covering the grid, the detection of covering could be delayed because of too infrequent checks.

Issue 2487

### Fixed AWH continuation consistency checks

Some kinds of changes upon restarts are now disallowed, as intended.

### Fixed AWH awh-nsamples-update value checking

Zero is now forbidden, as intended.

Issue 2489

### Fixes for `gmx` tools

### Fixed buffer overflow in grompp warnings

When grompp issued a warning or error with a filename/path of more than 255 characters, a buffer overflow would occur. This could also happens during make check/test.

Issue 2465

### Fixed infinite loop in gmx solvate

When provided with a PDB file that had no box information for the solvent, gmx solvate could be stuck in an infinite loop. Fixed by disallowing empty boxes for solvent PDB files.

Issue 2523

### Fixed enemat when the .edr file had no matching energy groups

Issue 2508

### Fixed PQR file output

PQR files from gmx editconf violated the standard for the format because they were always written in fixed format. This commit fixes the issue by introducing a different output method for PQR files that follows the standard.

Issue 2511

### Fixed crash in gmx solvate

gmx solvate would crash due to memory corruption when using multiple solvent molecule types.

### Added check for unallowed periodic setups

Long distances between atoms in bonded interactions could lead to incorrect periodicity removal. In such cases an inconsistent shift message was printed, but the run or analysis was not terminated and other, confusing, errors could be issued. Now an informative fatal error is issued.

Issue 2549

### Fixes to improve portability

### Fixed CUDA compilation on Windows.

Issue 2509

### Fixed SIMD support for POWER systems in double precision with gcc 8

Issue 2421

### Fixed possible illegal instruction on KNL with Intel compiler

Issue 2504

### Miscellaneous

### Information message about OMP_NUM_THREADS now sent to log file

Made it easier to track this information by writing it to the log file in a clear way, rather than to stderr.

Issue 2472

### Fixed inadvertent disabling of SIMD version of the integrator

Fixed a bug so the SIMD version of the leap-frog integrator is chosen, when possible. This may improve performance.

Issue 2497

### Fixed own FFTW builds on certain AVX2/AVX512 hardware

Version 3.3.8 of FFTW fixes some known gcc-8 errors for AVX2 by removing the fast-math flag, and it also appears to fix an issue with failed unit tests on AVX512-capable hardware, so we have bumped the version we download to 3.3.8.

Issue 2541

### Switched to using more standard CMake variables for installing on GNU systems

GnuInstallDirs.cmake is a better approach.

### Several documentation and output improvements

- Updated top-level README file for latest GROMACS core publication.
- Reporting about GPU detection has improved.
- `gmx mindist -pi` docs improved.
- Docs for mdp options relating to bonds improved.
- Fixed various typos.
- Removed a leftover mention of the twin-range scheme.
- `gmx trjconv -ndec` docs improved.

### GROMACS 2018.1 release notes

This version was released on March 21, 2018. These release notes document the changes that have taken place in GROMACS since the initial version 2018, to fix known issues. It also incorporates all fixes made in version 2016.5 and earlier, which you can find described in the *Release notes* (page 838).

#### Fixes where mdrun could behave incorrectly

#### Fixed leap-frog integrator with Nose-Hoover T coupling and Parrinello-Rahman P coupling

With Parrinello-Rahman P coupling active, when applying Nose-Hoover T coupling at an MD step where no P coupling occured, the update phase could use outdated or garbage coupling data. Such simulations with `nsttcouple` (page 65) equal to `nstpcouple` (page 67) are unaffected by this issue, so few users will be impacted by this. Simulations using other coupling algorithms are unaffected.

Issue 2418

#### Used SIMD bondeds without perturbed interactions

In free-energy calculations that lacked bonded interactions between perturbed atom types, the SIMD-accelerated bonded functions were inadvertently disabled. This has been enabled, which will improve the performance of some kinds of free-energy calculations.

#### Fixed bonds whose displacement was zero

We should allow overlapping atoms in harmonic bonds. But the former code would cause a floating point exception and incorrect free-energy derivatives.

#### Fixed centre-of-mass motion removal on part of the system

COMM removal requested for part of the system acted on the whole system.

Issue 2381

#### Fixed multi-simulations with multiple ranks per simulation

These used to crash or hang mysteriously before the simulation would start.

Issue 2403

### Improved inter-simulation signalling implementation

Reduced communication overhead with either many simulations or many ranks per simulation.

### Fixed FEP calculations with SHAKE

All SHAKE + FEP calculations accumulated wrong values to dH/dl output, but in some cases the result will look the same.

Issue 2434

### Fixed handling of mdp `define` statement assigning preprocessor values

Now .mdp files can configure the topology with values, as originally intended, e.g. `"define = -DBOOL -DVAR=VALUE"`.

Issue 2392

### Prevented log file energy average printing dividing by zero

If very few simulation frames have computed energies, then there may be insufficient data for averages. If so, skip the average printing entirely.

Issue 2394

### Correctly set cutoff modifiers in forcerec

The cutoff modifiers were not copied from interaction_const_t to forcerec_t which meant only the generic kernels were used with the group scheme. This fix will restore the performance of the group scheme.

Issue 2399

### Fixed box scaling in PME mixed mode using both GPU and CPU

Issue 2385

### Re-enabled GPU support with walls and 1 energy group

With a single non-bonded energy group and walls, we can now use a GPU for non-bonded calculations.

### Removed tumbling ice-cube warning with SD integrator

With SD, there is friction, so ice cubes will not tumble.

### Fixed assertion failure in test-particle insertion

Erroneous logic in the TPI meant that it always failed without producing any result.

Issue 2398

### Avoided mdrun echoing "No option -multi"

mdrun would print as many messages "No option -multi" as there are MPI ranks to stderr. Also updated -multi to -multidir in an error message.

Issue 2377

### Improved mdrun handling when GPUs are present but unavailable

Issue 2415

### Fixed crash with AWH and awh1-equilibrate-histogram=yes

When running AWH with awh1-equilibrate-histogram=yes and multiple MPI ranks, the simulation would segmentation fault.

Issue 2436

### Fixed issues with AWH and bias sharing

When sharing AWH biases between multiple simulations, there were four issues. An MPI error would occur when an individual simulation would use more than one rank. The deconvoluted PMF would be garbage (but the sampling was correct). with more than 32 MPI ranks for an individual simulation, an error about a coordinate being 0 could occur. And continuation from checkpoints could be refused.

Issue 2433 Issue 2439 Issue 2441 Issue 2444

### Fixed virial with AWH and domain decomposiion

When running AWH with domain decomposition, the AWH/pull virial contribution would be multiplied with the number of MPI ranks.

### Fixed restart bug with pull geometry direction-periodic

With COM pulling with geometry direction-periodic, (only) at the step of continuing from checkpoint the closest PBC image would be used instead of the of the one closest to the reference value. This could lead to a sharp spike in the pull force at the continuation step.

Issue 2446

### Fixes for `gmx` tools

### Added check in grompp to avoid assertion failure

With an mdp file with a parameter present with both the current name and the old name which automatically gets replaced, an assertion would fail. Now a fatal error is issued.

Issue 2386

### Fixed grompp net charge check

Use of multiple non-consecutive blocks of a moleculetype now works correctly.

Issue 2407

### Fixed issue with adding selection groups for TNG output

When there were more molecule blocks than molecule types in the topology, the output was wrong.

### Fixed help text and functionality of `pdb2gmx -missing`

This now permits dangling bonds at termini, which is occasionally useful.

### Fixes to improve portability

### PME on Fermi-era GPUs on large systems now works

On older GPUs, it was possible to run into a hardware size limitation that has now been fixed.

Issue 2409

### GoogleTest death tests are now used in a more portable way

### Tests for GPU utility functionality are now more robust

Non-GPU builds, and GPU builds that find incompatible or otherwise unavailable devices will pass the tests in the manner intended.

Issue 2405

### Used more portable python shebangs

Per https://www.python.org/dev/peps/pep-0394/#recommendation, we should use env, and point it at python2. When we either make them 2/3 or just-3 compatible, this should change.

Some distros (notably Arch Linux) already point python at python3 so we should choose to be explicit, and thus somewhat portable.

Issue 2401

### Added work-around for GCC 5.3 targetting AVX512 hardware

GCC 5.3 has bug in overload resolution causing the AVX512 and scalar function to become ambiguous.

### Used isfinite unambiguously

Patch provdied by Veselin Kolev to quiet some compiler warnings.

Issue 2400

### Worked around gcc-6 bug in tabulated group non-bonded kernels

With the gcc-6 compiler, AVX and -O3, which is the default, the tabulated non-bonded kernels of the (deprecated) group cutoff-scheme produced incorrect energies and forces. The errors are so large that they could not have caused latent issues.

Issue 2424

### Detected correct AMD Zen SMT topology

On recent AMD Zen processors, hardware thread detection and pinning handling have been fixed, improving performance.

Issue 2388

#### Fixed POWER VSX SIMD usage for upcoming gcc version 8

Issue 2421

#### Fixed clang 6 with CUDA 9

Permits builds for sm_70 and may work around an issue with sm_37

Issue 2443

#### Miscellaneous

#### Made multi-atom TPI reproducible with different compilers

#### Documentation enhancements

In particular, for handling options to mdrun relating to GPUs and running mdrun with good performance.

### 11.11.2 Major release

#### Highlights

GROMACS 2018 was released on January 10, 2018. Patch releases may have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

As always, we've got several useful performance improvements, with or without GPUs, and all enabled and automated by default. We are extremely interested in your feedback on how well this worked on your simulations and hardware. They are:

- PME long-ranged interactions can now run on a single GPU, which means many fewer CPU cores are needed for good performance.
- Optimized SIMD support for recent CPU architectures: AMD Zen, Intel Skylake-X and Skylake Xeon-SP.

There are some new features available also:

- The AWH (Accelerated Weight Histogram) method is now supported, which is an adaptive biasing method used for overcoming free energy barriers and calculating free energies (see https://doi.org/10.1063/1.4890371).
- A new dual-list dynamic-pruning algorithm for the short-ranged interactions, that uses an inner and outer list to permit a longer-lived outer list, while doing less work overall and making runs less sensitive to the choice of the "nslist" parameter.
- A physical validation suite is added, which runs a series of short simulations, to verify the expected statistical properties, e.g. of energy distributions between the simulations, as a sensitive test that the code correctly samples the expected ensemble.
- Conserved quantities are computed and reported for more integration schemes - now including all Berendsen and Parrinello-Rahman schemes.

### New and improved features

### Added support for AWH biasing

The AWH (Accelerated Weight Histogram) method is an adaptive biasing method used for overcoming free energy barriers and calculating free energies (see https://doi.org/10.1063/1.4890371). Although AWH can in general bias any system parameter, this change only implements biasing of reaction coordinates. The actual force distribution and coordinate handling is taken care of by the pull code. AWH interacts with the pull code by registering itself as the external potential module for the coordinate that should be AWH biased. The AWH code sets the potential and force for those coordinates. See the reference manual for full documentation.

It includes a feature to compute the time-integrated force correlation, also known as the friction tensor (see e.g. https://doi.org/10.1103/PhysRevLett.108.190602). The friction tensor defines a metric on the coordinate space and the local volume element of this metric is a useful measure for determining which regions need more or less sampling.

### Dual pair-list buffer with dynamic pruning

The GROMACS simulation engine uses a new dual pair-list algorithm with dynamic pruning in cases where the Verlet buffer is determined automatically (which is the default). This allows further reducing the frequency of pair search (and domain decomposition) while avoiding large Verlet buffers and the previously inherent increased computational cost in the short-ranged nonbonded kernels. This is achieved by constructing an "outer" pair-list built infrequently, which includes many pairs in the list that are outside the cut-off range for most of the lifetime of the list. Such pairs can be pruned out very efficiently every few steps and with that building a smaller, "inner" pair-list with a shorter life-time, and importantly a correspondingly shorter Verlet buffer (still adhering to the specified tolerance), which is then used in the nonbonded kernels. Thanks to this, simulations runs are significantly less sensitive to tuning the search frequency parameter ("nstlist"). When short-ranged interactions are running on the GPU, the dynamic pruning is overlapped with the integration on the CPU, so is usually free. This feature improves all of simulation rate, hardware utilization, and power consumption.

### Added physical validation suite

These tests run series of short simulations and verify the expected statistical properties, e.g. of energy distributions between the simulations, as a sensitive test that the code correctly samples the expected ensemble.

To run everything locally (which can take a few hours!) use

```
cmake -DGMX_PHYSICAL_VALIATION=ON ..
make
make check-phys-run
```

Currently, the script is only running a few systems, checking convergence of energy conservation in NVE system with decaying timestep, and the ensembles generated by a few thermostating and barostating algorithms. Other systems and ensembles covering a broader combination of settings will be added over time.

### Added reporting of conserved quantities for coupling algorithms

The work that some more coupling algorithms (Berendsen pressure, Berendsen temperature, and Parrinello-Rahman pressure) applies on the system is calculated and integrated. Formulae are in the reference manual.

### Added acceleration correction VCM mode

New mdp option to remove the center of mass translational velocity and correct the center of mass position, assuming linear acceleration. This is useful e.g. for pulling on a group using an absolute reference.

### Changed handling of gmx mdrun -gpu_id

As more code is able to be offloaded to the GPU, task assignment has become more complex, and is likely to get more complex still. The -gpu_id command-line option now merely enables the user to restrict which of the detected GPUs are available to the automated task assignment scheme, somewhat like the `CUDA_VISIBLE_DEVICES` environment variable. For the rare cases where full control is needed, `gmx mdrun -gputasks` is available and documented in the user guide.

### Added log output for equivalent 1x1 pair-list setup

The GROMACS NxM setup can use a shorter pair-list buffer than other codes' simpler 1x1 scheme, so our log files now report the equivalent setup, to help people doing performance and correctness comparisons.

### New mdp input for electric fields

### EW3DC for non-neutral systems

Added the corrections to force and energy according to Ballenegger, Arnold, and Cerda, J. Chem. Phys. 131, 094107 2009 (https://doi.org/10.1063/1.3216473). Hinted that people read https://doi.org/10.1021/ct400626b to help make good choices. Added a warning in grompp for charged systems with Ewald.

### Reduce rounding errors in SETTLE

The parameters for SETTLE are now computed in double precision, which lowers the systematic error.

### Made `gmx mdrun -pforce` terminate with non-finite forces

The *gmx mdrun* (page 258) option -pforce used to print non-finite forces, but would not terminate the run if any were found. Now a fatal error is issued.

## Performance improvements

### Implemented support for PME long-ranged interactions on GPUs

A single GPU can now be used to accelerate the computation of the long-ranged PME interactions. This feature provides excellent performance improvements, in particular that only 2-4 CPU cores per GPU will be about as fast as the 2016 version that needed many more CPU cores to balance the GPU. Performance on hardware that had good balance of GPU and CPU also shows minor improvements, and the capacity for hardware with strong GPUs to run effective simulations is now greatly improved.

Currently, the GPU used for PME must be either the same GPU as used for the short-ranged interactions and in the same single rank of the simulation, or any GPU used from a PME-only rank. mdrun -pme gpu now requires that PME runs on a GPU, if supported. All CUDA versions and hardware generations supported by GROMACS can run this code path, including CUDA 9.0 and Volta GPUs. However, not all combinations of features are supported with PME on GPUs - notably FEP calculations are not yet available.

The user guide is updated to reflect the new capabilities, and more documentation will be forthcoming.

### Added more SIMD intrinsics support for PME spread and gather

Achieved speedup on Intel KNL processors of around 11% for PME spread/gather on typical simulation systems.

### Added SIMD intrinsics version of simple update

In the simple case of leap-frog without pressure coupling and with at most one temperature-coupling group, the update of velocities and coordinates is now implemented with SIMD intrinsics for improved simulation rate.

### Add SIMD intrinsics version of Urey-Bradley angle kernel

For steps where energies and shift forces are not required, this kernel improves performance, which can otherwise be rate limiting in GPU-accelerated runs, particularly with CHARMM force fields.

### Use OpenMP up to 16 threads with AMD Ryzen when automating run setup

AMD Ryzen appears to always perform slightly better with OpenMP than MPI, up to using all 16 threads on the 8-core die.

### 128-bit AVX2 SIMD for AMD Ryzen

While Ryzen supports 256-bit AVX2, the internal units are organized to execute either a single 256-bit instruction or two 128-bit SIMD instruction per cycle. Since most of our kernels are slightly less efficient for wider SIMD, this improves performance by roughly 10%.

### Choose faster nbnxn SIMD kernels on AMD Zen

On AMD Zen, tabulated Ewald kernels are always faster than analytical. And with AVX2_256 2xNN kernels are faster than 4xN. These faster choices are now made based on CpuInfo at run time.

Refs Issue 2328

### Enabled group-scheme SIMD with GMX_SIMD=AVX2_128

The group-scheme kernels can use AVX instructions from either the AVX_128_FMA and AVX_256 extensions. But hardware that supports the new AVX2_128 extensions also supports AVX_256, so we enable such support for the group-scheme kernels.

### Detect AVX-512 FMA units to choose best SIMD

Recent Intel x86 hardware can have multiple AVX-512 FMA units, and the number of those units and the way their use interacts with the way the CPU chooses its clock speed mean that it can be advantageous to avoid using AVX-512 SIMD support in GROMACS if there is only one such unit. Because there is no way to query the hardware to count the number of such units, we run code at CMake and mdrun time to compare the performance from using such units, and recommend the version that is best. This may mean that building GROMACS on the front-end node of the cluster might not suit the compute nodes, even when they are all from the same generation of Intel's hardware.

### Speed up nbnxn buffer clearing

### Tweaked conditional in the nonbonded GPU kernels

GPU compilers miss an easy optimization of a loop invariant in the inner-lop conditional. Precomputing part of the conditional together with using bitwise instead of logical and/or improves performance with most compilers by up to 5%.

### Improvements to GROMACS tools

#### Split off the NMR related analyses from gmx energy.

A new tool *gmx nmr* (page 272) is created by straight copying code from *gmx energy* (page 214) to a new tool. The reason is to reduce complexity.

A few cleanups are introduced to pass the valgrind memory test.

Added references the *gmx nmr* (page 272) in the manual.

#### Added selection-enabled gmx trajectory

For now, this tool only plots coordinates, velocities, and forces for selections, so it should provide a full replacement for -ox, -ov, -of, -com, and -mol from *gmx traj* (page 328).

#### Decreased memory usage in gmx traj and gmx trjconv

#### Made TNG writing work with multiple identical steps

Introduce a wrapper structure around TNG so we detect and correct for cases when writing multiple frames with the same step, or non-zero initial steps to TNG files. This will avoid frames overwriting each other, and make sure the time per frame is correct.

Issue 2189

#### Improved frame time/step handling in gmx trjconv

Store the exact step in PDB/GRO file headers, and be more careful about not claiming to have time or step information when it was not available. This change will avoid some of the problems described in Issue 2189, but it does not yet properly fix the issue in the TNG library.

Issue 2189

#### Fixed gmx trjconv to always dump at correct time

Set frame timestep before starting the loop by reading first two frames and rewinding, and make sure we always write something to the dump output based on best-guess (if there is at least one input frame present).

Issue 1832

### Clarified gmx editconf help text

It is possible that users can confuse -c with -center so this patch makes it clear that -center doesn't do anything unless the user really wants to shift the center of the system away from the middle of the box.

Fixes Issue 2171

### Added option -water tips3p to pdb2gmx.

Fixes Issue 2272

### Removed incorrect comment for CHARMM tips3p

Removed CHARMM tips3p performance warning in *gmx pdb2gmx* (page 281) input file, since the performance loss is negligible with the `cutoff-scheme=Verlet` (page 57).

### Avoided gmx grompp charge warning from merely rounding error

Even though the *gmx grompp* (page 230) total charge check uses double for summation, there are rounding errors for each charge when charges are stored in single precision. Now the charge check rounds the net charge of molecules to integer when the difference is less than the maximum possible sum of charge rounding errors.

Fixes Issue 2192

### Improved pdb2gmx for nonstandard residue types

If explicit non-blank chain identifiers are set, it will now be a hard error if the residue types in each chain do not match. For blank chain ID we still need to allow detection of non-chain parts, but this case too now provides more explicit output information.

Issue 2370

### Allowed empty lines in hdb files

Skip lines that consist only of whitespace. Not a universal solution for fixing hdb files, but better than the user getting very strange error messages that don't say anything about whitespace.

Issue 2028

### Changed to no longer require matching names between rtp and tdb files

This was only documented in the source. It's a remnant from the days when all force fields were in the same directory, and no longer necessary. With this change we will properly match all termini to all amino acids.

Issue 2026 Issue 2027

### Made duplicate atoms in bondeds an error in gmx grompp

Having duplicate atom indices in bonded interactions used to be only a warning. But since in nearly all cases this will lead to issues, this is now a error, except for angle restraints where it can be useful so there it is now a note.

Issue 2141

### Made gmx grompp -r obligatory with position restraints

With position restraints it would often occur that users accidentally used equilibrated coordinates instead of the original coordinates for position restraint coordinates due to -r defaulting to -c. Now -r always need to be supplied with position restraints, but using the same file name as with -c will reproduce the old behavior.

### Fixed gmx msd when using COM removal and molecules

Changed order of code to actually assign correct coordinates before copying the data, and modified data structure size when using COM removal and individual molecules.

Issue 2043

### Fixed index error in gmx chi

An error in the index construction could lead to segfaults. However, the actual indices were correct, so it should not have produced any incorrect results.

Issue 1814

### Fixed gmx grompp complexity for large exclusion orders

To avoid exploding computational complexity for highly connected molecules with large values for excluded neighbors, avoid adding a neighbor to the temporary nnb structure if it is already present as a lower-order neighbor.

Issue 2260

### Fixed gmx density for non-mass calculations

We now always use mass and never charge/electron density to center systems.

Issue 2230

### Fixed gmx check for tprs with different numbers of atoms

Fixes Issue 2279

### Bugs fixed

### Fixed multiple time stepping with Parrinello-Rahman and Nose-Hoover.

These now work in correct Trotter style, applied once and scaled by the correct number of steps.

Fixes Issue 2031 Fixes Issue 2032

### Applied Berendsen pressure coupling only at nstpcouple steps

Berendsen pressure coupling was mistakenly applied on successive steps. Since there is no need for this, this is changed to act only on nstpcouple steps. Note that this change prevents continuation from old checkpoint files for Berendsen pressuring-coupling runs, since the previous-step pressure is no longer stored.

### Added missing Ewald correction for PME-User

With *coulombtype=PME-User* (page 60), the Ewald mesh energy was not subtracted leading to (very) incorrect Coulomb energies and forces.

Fixes Issue 2286

### Fixed incorrect dV/dlambda for walls

The free-energy derivative dV/dlambda for walls, which can be perturbed by changing atom types of non-wall atoms, only contained the B-state contribution.

Fixes Issue 2267

### Supported OpenMP for orientation restraints

Previously this was broken, but has been fixed and is now tested and supported.

### Fixed orientation restraint reference

The resetting of the COM of the molecule with orientation restraints for fitting to the reference structure was done with the COM of the reference structure instead of the instantaneous structure. This does not affect the restraining (unless ensemble averaging is used), only the printed orientation tensor.

Fixes Issue 2219

### Used graph with orientation restraints

With the Verlet cut-off scheme by default molecules are not made whole. Now they are made whole when orientation restraints are used. Added checks and assertions for correct PBC treatment with orientation restraints.

Fixes Issue 2228

### Fixed Ekin at step 0 with COM removal

The kinetic energy at step 0 was computed from the velocities without the center of mass velocity removed. This could cause a relatively large jump in kinetic energy, especially for small systems. Now compute_globals is called twice with COM removal so we get the correct kinetic energy.

### Fixed gmx grompp with Andersen massive and no COM removal

Fixed a floating point exception leading to a segv. Also fixed possible different rounding for the interval for Andersen massive in *gmx grompp* (page 230) in mdrun for the common case where tau-t is a multiple of delta-t.

Fixes Issue 2256

### Improved Verlet buffer constraint estimate

The displacement estimate for a constrained atom (typically H) rotating around the COM with a partner atom is now derived and documented correctly. Note that we (still) use a Gaussian with matched variance, which results in a much larger buffer than necessary, since the tail of the displacement distribution sets the buffer size and the Gaussian has a long tail whereas the actual distribution has no tail.

### Fixed virtual site generation for water oxygens not named OW

*gmx pdb2gmx* (page 281) would break when generating virtual sites if water oxygens were not named OW. Now checking for the atomnumber instead.

Fixes Issue 2268

### Fixed thread-MPI rank choice for orientation restraints

Only a single rank is supported, so that must be what the thread-MPI code will choose. There's another check later on that catches the multi-rank MPI case.

### Fixed some incorrect behavior with gmx solvate

*gmx solvate* (page 319) cannot replicate non-rectangular solvent boxes correctly (there are several different places that assume a diagonal box matrix), so give a fatal error if that is attempted. To support some uses with triclinic boxes, skip the replication step if the solvent and target box sizes are already equal.

Support for general triclinic boxes can be added separately, and the check introduced here can be valuable even in that case: it keeps a pre-equilibrated solvent box intact if the target box size is the same.

Related to fix of Issue 2148

### Fixed DD exact continuation in reproducible node

With domain decomposition, the local atom density, used for setting the search grid for sorting particles, was based on the local atom count including atoms/charge groups that would be moved to neighboring cells. This lead to a different density value, which in turn could result in a different number of search grid cells and thus a different summation order during a run compared with continuing that run from a checkpoint, when no atoms would be moved. That difference violated the intention of `mdrun -reprod`, and is now fixed.

Refs Fixes Issue 2318

### Made mdrun only stop at nstlist steps with mdrun -reprod

Stopping mdrun with two INT or TERM signals (e.g. from Ctrl-C from the terminal shell) would always happen right after the first global communication step. But this breaks exact continuation. Now with `mdrun -reprod` a second signal will still stop at a pair-list generation step, like with the first signal, so we can still have exact continuation.

Fixes Issue 2318

### Added check for GPU detection support before detecting GPU devices

When a CUDA-enabled binary was run on a node with no CUDA driver available, a note was issued that the version of the CUDA driver is insufficient, which was wrong and now fixed.

Fixes Issue 2322

### Removed duplicated lines from OPLS ffbonded.itp

Identical lines have been removed, as identified with uniq.

Fixes Issue 1678.

### mdrun no longer warns about NVML clocks that are at max

If the clocks are already maxed out there is no point in echoing warnings about not being able to set them.

Fixes Issue 2313.

### Used reduced default tolerances for tpx comparison

The tolerances for gmx check are mainly intended for handling slight statistical deviations, but they can hide differences between tpr files, when the user likely wants exact checks on small quantities like Lennard-Jones parameters. This changes changes the default relative tolerance to 0.000001 and the absolute tolerance to zero, so that we only allow for any minor differences due to compiler optimization.

Fixes Issue 2024.

### Fixed return values of frame-reading functions

This function was based on read_first_x that returned the number of atoms, and was documented to do the same, but has always returned a logical boolean about whether a frame has been read. This led to aspects of `gmx spatial` and `gmx trjcat -demux` being broken.

Fixed by returning a proper bool, and fixing the remaining logic that used the return value in a non-boolean sense.

Refs Issue 2157

### Removed PBC before generating TPR with group scheme

Ensure that all molecules have been made whole before generating the run input file when using the group scheme, to avoid error messages for large charge groups when molecules are broken over PBC boundaries.

Fixes Issue 2339

### Fixed PBC error in gmx_spatial

Fixes Issue 2157.

### Documented power spectrum options of gmx velacc

Fixes Issue 2019.

### Changed to require .tpr file for gmx cluster

The program could crash without it, so it wasn't optional.

Fixes Issue 2170.

### Disallowed ascii formats for gmx trjcat

Since gmx trjcat (deliberately) does not use any .tpr file, the tool can't handle trajectory formats such as .gro or .pdb where atom/residue names are needed.

Fixes Issue 2225.

### Improved grompp missing-parameters error message

If an interaction entry had parameters but not the function type, then the error message has been confusing. Note that even when only one function type is implemented, the field is still required, which makes for ready extensibility.

Refs Issue 2144

### Checked for large energy at first step

Also added step number to fatal error message.

Fixes Issue 2333

### Disallowed combination of PME-user and verlet cutoff

Fixes Issue 2332

### Avoided confusing message at end of non-dynamical runs

Energy minimization, test-particle insertion, normal-mode analysis, etc. are not targets for performance optimization so we will not write performance reports. This commit fixes an oversight whereby we would warn a user when the lack of performance report is normal and expected.

Fixes Issue 2172

### Changed to require `-ntmpi` when setting `-ntomp` and using GPUs

With GPUs and thread-MPI, setting only `gmx mdrun -ntomp` could lead to oversubscription of the hardware threads. Now, with GPUs and thread-MPI the user is required to set `-ntmpi` when using `-ntomp`. Here we chose that to also require `-ntmpi` when the user specified both `-nt` and `-ntomp`; here we could infer the number of ranks, but it's safer to ask the user to explicity set `-ntmpi`. Note that specifying both `-ntmpi` and `-nt` has always worked correctly.

Fixes Issue 2348

## Removed features

### Removed hybrid GPU+CPU nonbonded mode

This mode was not very useful, since it ran the non-local non-bonded interactions on the CPU. The fraction of non-local interaction is set by the domain decomposition, so this is not flexible. Also this mode was not being tested.

### QM/MM: removed optimization and transition-state search

These functionalities used to only work with old versions of Orca, had very limited use and will possibly not work any longer now.

### Updated application clock handling on Pascal+ GPUs

Starting with Pascal (CC >= 6.0) it is no longer possible to change application clocks without root privileges. Application clocks are still reported for Pascal+, but there is no longer suggestions about changing them.

### Removed continuation from gmx convert-tpr

Removed the obsolete option of *gmx convert-tpr* (page 178) to write a tpr file for continuation using a trajectory and energy file. This is superseded by checkpointing.

### Portability

#### Enabled compiling CUDA device code with clang

clang can be used as a device compiler by setting GMX_CLANG_CUDA=ON. A CUDA toolkit (>=7.0) is also needed. Note that the resulting runtime performance is usually worse than that of binaries compiled by the official NVIDIA CUDA compiler (nvcc).

#### Increased the oldest cmake, compiler and CUDA versions required

We now require gcc-4.8.1, clang-3.3 and icc-17.0.1, so we can rely on full C++11 support. We now also require CUDA-6.5 and CMake-3.4.3.

#### Added check that CUDA available hardware and compiled code are compatible

Added an early check to detect when the *gmx mdrun* (page 258) binary does not embed code compatible with the GPU device it tries to use nor does it have PTX that could have been just-in-time compiled.

Additionally, if the user manually sets GMX_CUDA_TARGET_COMPUTE=20 and no later SM or COMPUTE but runs on >2.0 hardware, we'd be executing just-in-time-compiled Fermi kernels with incorrect host-side code assumptions (e.g amount of shared memory allocated or texture type). This change also prevents such cases.

Fixes Issue 2273

#### Disabled ARM Neon native rsqrt iteration used in short-ranged interactions

Fixes Issue 2261

#### Avoided FTZ triggering simd test failures

For very small arguments on platforms without FMA support, the Intel compiler's default usage of flush-to-zero for denormal values can lead to slight deviations. Since this is a range we really don't care about, and non-FMA platforms are anyway a thing of the past, just avoid testing a very small range around that threshold for non-FMA SIMD platforms.

Issue 2335

#### Fixed OpenCL compiles on Mac OS

Confirmed to work on Mac OS 10.13.2 running on a Macbook Pro with Radeon Pro 560.

Issue 2369

### Tested that nvcc/host compiler combination works

We now compile a trivial CUDA program during a run of CMake to catch both unsupported nvcc/host compiler version combinations and other unknown errors.

Issue 1616

### Added AVX_512 and KNC symbols to FFTW SIMD test

Otherwise the CMake code might complain loudly about FFTW not being accelerated on KNC or KNL hosts.

### Implemented changes for CMake policy 0068

CMake-3.9 introduced a changed behavior for RPATH vs. install_name options on OS X. This avoids relying on functionality that will be removed in future CMake versions.

### Miscellaneous

### Updated note in manual on stochastic dynamics integrator

The comment in the SD section about Berendsen was outdated. Added a few sentences on equilibration/damping of modes.

### Added grompp note for Parrinello-Rahman + position restraints

This combination can be unstable and is often not desirable, so grompp now issues a note to suggest alternatives to the user.

Refs Issue 2330

### Clarified the description of Fmax during energy minimization

### Improved vsite parallel checking

The vsite struct now stores internally whether it has been configured with domain decomposition. This allows for internal checks on valid commrec, which have now been added, and would have prevented Issue 2257.

### Added partial support for writing masses and partial charges with TNG files

Issue 2188

### Updated TNG to version 1.8.1

Added data block for atom masses.

Fixes Issue 2187 and Issue 2250 and other bugs and warnings.

### Added load balance fraction to DLB print

DLB can often be based on a small fraction of the total step time, especially with GPUs. Now this is printed to md.log and stderr.

### Added reference for dihedral function in OPLS.

The OPLS four-term dihedral function was not described in the reference listed earlier, so this was updated. Also updated the reference to the three term dihedral to an older paper.

### Updated many aspects of the documentation

Imported and updated more material from the wiki. Incorporated suggestions arising from many tracked issues. Updated user guide, developer guide, install guide, and reference manual.

### Updated mdrun signal help text

Updated mdrun help text on signal handling for old and recent changes to the behavior.

Fixes Issue 2324

### Changed to handle erroneous command line args better

Some gmx modules need to be able to accept non-option arguments, and some should not. Introduced enough functionality to support such behaviour, while giving useful error messages in cases where the command line is merely missing hyphens (which can happen e.g. when people copy-paste from inconveniently built PDF files for tutorials). Increased test coverage of relevant cases.

Removed some useless command-line argument strings from test cases that never needed them.

Also tested some behaviours of handling string options, and renamed some test input strings to reflect the intent.

Issue 2153

#### Changed to no longer allow multiple energy groups for GPU runs

Exit with a fatal error instead of only warning, since the latter leads to writing data for energy groups that is incorrect to the energy file.

Issue 1822

#### Removed duplications in GMXLIB search paths

Remove entries that are duplicated, or identical to the default search path, to avoid e.g. listing identical force fields multiple times.

Issue 1928

#### Changed to no longer write reference pull group 0 to log

This is an internal group used for absolute references, which cannot be set by users, so printing it just leads to confusion.

Issue 2143

## 11.12 GROMACS 2016 series

### 11.12.1 Patch releases

#### GROMACS 2016.5 Release Notes

This version was released on February 16, 2018. These release notes document the changes that have taken place in GROMACS since version 2016.4 to fix known issues. It also incorporates all fixes made in version 5.1.5 (which was the last planned release in that series).

#### Fixes where mdrun could behave incorrectly

#### Fixed triclinic domain decomposition bug

With triclinic unit-cells with vectors a,b,c, the domain decomposition would communicate an incorrect halo along dimension x when b[x]!=0 and vector c not parallel to the z-axis. The halo cut-off bound plane was tilted incorrect along x/z with an error approximately proportional to b[x]*(c[x] - b[x]*c[y]/b[y]). When c[x] > b[x]*c[y]/b[y], the communicated halo was too small, which could cause instabilities or silent errors. When c[x] < b[x]*c[y]/b[y], the communicated halo was too large, which could cause some communication overhead.

Issue 2125

### Required -ntmpi with setting -ntomp with GPUs

With GPUs and thread-MPI, setting only `-ntomp` could lead to oversubscription of the hardware threads. Now with GPUs and thread-MPI the user is required to set `-ntmpi` when using `-ntomp`. Here we chose that to also require `-ntmpi` when the user specified both `-nt` and `-ntomp`; here we could infer the number of ranks, but it's safer to ask the user to explicity set `-ntmpi`. Note that specifying both `-ntmpi` and `-nt` has always worked correctly.

Issue 2348

### Prevented dynamic load balancing activating immediately after exchange

Turning on DLB right after exchanging replicas caused an assertion failure and is also useless.

Issue 2298

### Avoided confusing message at end of non-dynamical runs

EM, TPI, NM, etc. are not targets for performance optimization so we will not write performance reports. This commit fixes and oversight whereby we would warn a user when the lack of performance report is normal and expected.

Issue 2172

### Changed to issue fewer messages when `-cpi` checkpoint file is not present

Removed duplicated message.

Issue 2173

### Disallowed combination of PME-user and Verlet cutoff

Issue 2332

### Added missing Ewald correction for pme-user

With `coulomb-type = pme-user`, the Ewald mesh energy was not subtracted leading to (very) incorrect Coulomb energies and forces.

Issue 2286

### Fixed thread-MPI rank choice for orientation restraints

Only a single rank is supported, so that must be what the thread-MPI code will choose. There's another check later on that catches the multi-rank MPI case.

### Fixed nstlist increase warning print

The log file warning message had a buggy conditional which this commit fixes.

### Removed incorrect comment for CHARMM tips3p

### Fixed incorrect dV/dlambda for walls

The free-energy derivative dV/dlambda for walls, which can be perturbed by changing atom types of non-wall atoms, only contained the B-state contribution.

Issue 2267

### Fixed warning for confout with periodic molecules

With periodic molecules, `gmx mdrun` would incorrectly attempt to make molecules whole for writing the final state to confout.

Issue 2275

### Fixed wrong megaflop accounting

Some nrnb index entries were missing in the interaction_function array, leading to that wrong megaflops accounting printed.

Issue 2274

### Fixes for `gmx` tools

### Fixed `gmx grompp` net charge check

The grompp check for the net charge would ignore molecule blocks at the end when molecule types are used in multiple, non consecutive molecule blocks.

Issue 2407

### Extended `gmx grompp` missing energy term message

Issue 2301

### Fixed `gmx genion` charge summation accuracy

`gmx genion` accumulated the charge is a float, which could cause underestimation of the net charge for highly charged systems.

Issue 2290

### Fixed `gmx check` for tprs with different #atoms

Issue 2279

### Fixed `gmx grompp` with Andersen massive and no COM removal

Fixed a floating point exception leading to a crash. Also fixed possible different rounding for the interval for Andersen massive in `gmx grompp` from `gmx mdrun` for the common case where `tau_t` is a multiple of `delta_t`.

Issue 2256

### Improved documentation

### Updated documention of Nose-Hoover output

The documentation of Nose-Hoover chain variable printing was (long) outdated.

Issue 2301

### Clarified docs for Fmax in EM

### GROMACS 2016.4 Release Notes

This version was released on September 15, 2017. These release notes document the changes that have taken place in GROMACS since version 2016.3 to fix known issues. It also incorporates all fixes made in version 5.1.4 and several since.

### Fixes where mdrun could behave incorrectly

### Disabled PME tuning with the group scheme

PME tuning with the group cut-off scheme did not work correctly. Interactions between charge-group pairs at distances between `rlist` and `rcoulomb` can go missing. The group scheme is deprecated, and this issue would require considerable effort to fix and test, so we have simply disabled PME tuning with the group scheme.

Issue 2200

### Fixed value of Ewald shift

In all the Ewald short-ranged kernel flavours, the value of the potential at the cutoff is subtracted from the potential at the actual distance, which was done incorrectly (failing to divide the shift value by cutoff distance). Fortunately, the value of that distance is often close to 1, and the inconsistent shifts often cancel in practice, and energy differences computed on neighbour lists of the same size will have the error cancel. The difference doesn't even show up in the regressiontests, but would if we had a unit test of a single interaction.

Issue 2215

### Fixed orientation restraint reference

The resetting of the COM of the molecule with orientation restraints for fitting to the reference structure was done with the COM of the reference structure instead of the instantaneous structure. This does not affect the restraining (unless ensemble averaging is used), only the printed orientation tensor.

Issue 2219

### Fixed bugs with setup for orientation restraints

The orientation restraint initialization got moved to before the initialization of the domain decomposition, which made the check for domain decomposition fail. Also fixed orientation restraints not working with the whole system as fitting group.

### Worked around missing OpenMP implementation in orientation restraints

The orientation restraint code is not aware of OpenMP threads and uses some global information. By only running it on the master rank, results are now independent of number of threads used.

Issue 2223

### Enable group-scheme SIMD kernels on recent AVX extensions

The group-scheme code only runs using the feature set of AVX_256, but that is supported on the more recent hardware, so we should have the group scheme run with the maximum suitable SIMD. With previous releases, building AVX_256 binaries was required for best performance with the (deprecated) group scheme.

### Fix FEP state with rerun

When using FEP states with rerun, the FEP state was always 0.

Issue 2244

### Fixed COM pull force with SD

The reported COM pull force when using the SD integrator was random only. Now the pull force is summed over the systematic and random SD update components. A better solution is to not add the random force at all, but such a change should not be done in a release branch.

Issue 2201

### Fix PBC bugs in the swap code

Issue 2245

### Fixed flat-bottomed position restraints with multiple ranks

Reallocation was never done for flat-bottomed restraints, during domain decomposition, so the indexing could go out of range, leading to segfaults.

Issue 2236

### Fixed null pointer print in DD

Fixed a (rather harmless) print of a null pointer string during DD initialization. This would only show up with `gmx mdrun -dlb yes`.

### Improved the "files not present" error message

It's possible to use `gmx mdrun -deffnm` in restarts even if it wasn't used in the initial simulation. This can lead to absurd situations such as:

**Expected output files not present or named differently:**
pullx.xvg pullf.xvg

where `pullx.xvg` and `pullf.xvg` are present and named exactly as listed, but GROMACS expects them to be named as `-deffnm` requested.

The improved error message suggest to the user to check for that possibility.

Issue 942 (partial workaround)

### Fixed LJ-PME + switch grompp error

An error call was missing in grompp when LJ-PME was requested in combination with a force or potential switch modifier.

Issue 2174

### Fixed unused SIMD PME table kernel

The Verlet-scheme 2xNN PME kernel with tabulated correction had several issues. This kernel flavor could only be selected manually by setting an environment variable, so no user simulations should be affected.

Issue 2247

### Fixed bugs in most double-precision Simd4 implementations

The double precision version of reduce() and dotProduct() returned a float with AVX2_256, AVX_256, AVX_128_FMA, AVX_512, MIC and IBM_QPX. Only reduce() is used in double, in the PME force gather, and the difference is small.

Issue 2162

### Avoid inf in SIMD double sqrt()

Arguments > 0 and < float_min to double precision SIMD sqrt() would produce inf on many SIMD architectures. Now sqrt() will return 0 for arguments in this range, which is not fully correct, but should be unproblematic.

Issue 2164 Issue 2163

### Fix NVML error messages

These should refer to the API calls that failed, e.g. when users lack permissions to change clocks.

### Fixed IMD interface malfunctions

Issue 2206

### Fixed initial temperature reporting

When continuing a simulation from a checkpoint, mdrun could report double the intial temperature when `nstcalcenergy=1` or `nsttcoupl=1`. Note that this only affected reporting, the actual velocities were correct. Now the initial temperature is no longer reported for continuation runs, since at continuation there is no "initial" temperature.

Issue 2199

### Fix exception in SIMD LJ PME solve

Clear SIMD padding elements in solve helper arrays to avoid, otherwise harmles, fp overflow exceptions.

Issue 2242

### Fixes for `gmx` tools

### Fixed memory access issues in gmx solvate

**There was out-of-bounds access if**

1) the solvent configuration was given as a .pdb file, or
2) there was more than one type of residue in the solvent (which triggered sorting).

Also fix a memory leak in the sorting routine.

Should fix crashes mentioned in Issue 2148

### Fixed a consistency check in `gmx make_edi` for flooding

If one sets up a flooding .edi input file with `gmx make_edi`, the code should check that one does not use of the last 6 eigenvectors of the covariance matrix, which correspond to the rotational and translational degrees of freedom. The check that was in the code erroneously checked against the number of eigenvalues neig that was stored in the .xvg file, not against the total number of eigenvectors which depends on the number of atoms nav used in gmx covar. Thus the original check would always fail if the .xvg eigenvalue file contained 1-6 values only.

### Supported quiet trajectory-handling I/O

Permits `GMX_TRAJECTORY_IO_VERBOSITY=0` to be set to keep frame-reading code quiet, which is convenient for tools using libgromacs.

### Improved documentation

### Migrated much content from the wiki to the user guide

This includes * expanding the "Performance" section, * reworking extending simulations, doing restarts and reproducibility, * adding documentation for mdp option `simulation-part`. * adding documentation for issues relating to floating-point arithmetic * adding documentation for run-time errors

### Corrected the PDF manual to reflect that all tools are called `gmx <tool>`

There were still a few occurrences of the old-style `g_tool` naming, this patch removes. Deliberately left `g_membed` as is, because there was never a `gmx membed`, but instead it got incorporated into `gmx mdrun`.

### Clarified `gmx editconf` help text

It is possible that users can confuse `-c` with `-center` so this patch makes it clear that `-center` doesn't do anything unless the user really wants to shift the center of the system away from the middle of the box.

Issue 2171

### Added missing .mdp file documentation for the enforced rotation module

### Fixed parameter description for dihedral_restraints

The force-constant parameter for dihedral_restraints was not documented in the table of interaction types.

Issue 2144

### Replaced instance of "group" by "coord" in pull .mdp documentation

### Portability enhancements

### Supported CUDA 9/Volta for nonbonded kernels

Implemented production-quality support for Volta GPUs and CUDA 9.

The code was adapted to support changes to the nature of warp synchrony, without disturbing support for older GPUs and/or CUDA. Further improvements may be seen (e.g. in the 2017 release).

### Really enabled AVX512 in the GROMACS-managed build of FFTW

An earlier attempt to enable AVX512 on GCC 4.9 or newer and Clang 3.9 or newer was wrongly implemented. Now this works on all compilers we officially support (MSVC, GCC, clang, ICC).

### Fixed aspects for compiling and running on Solaris

### Fixed AVX512F compiler flags

Avoid using the MIC code generation flags for the Xeon code path.

### Fixed compiler flags for using MKL

### Fixes compilation issues with ARM SIMD

ARM_NEON has never supported double precision SIMD, so disabled it with GROMACS double-precision build.

The maskzR* functions used the wrong argument order in the debug-mode pre-masking (and sometimes in a typo-ed syntax).

In the shift operators, the clang-based compilers (including the armclang v6 compiler series) seem to check that the required immediate integer argument is given before inlining the call to the operator function. The inlining seems to permit gcc to recognize that the callers always use an immediate. In theory, the new code might generate code that runs a trifle slower, but we don't use it at the moment and the cost might be negligible if other effects dominate performance.

## GROMACS 2016.3 Release Notes

This version was released on March 14, 2017. These release notes document the changes that have taken place in GROMACS since version 2016.2 to fix known issues. It also incorporates all fixes made in version 5.1.4 and several since.

### Fixes where mdrun could behave incorrectly

### Fixed mdrun with separate PME ranks hanging upon exit

A recent fix for another issue led to mdrun hanging while communicating with PME ranks to coordinate end-of-run performance statistics.

Issue 2131

### Fixed handling of previous virials with md-vv integrator

These quantities get written to checkpoint files only for the Trotter pressure-coupling integrators that need them, but they were being copied in do_md for all Trotter integrators. This meant that an appending restart of md-vv plus nose-hoover plus no pressure coupling truncated off a correct edr frame and wrote one with zero virial and wrong pressure. And in the same case, a no-append restart writes a duplicate frame that does not agree with the one written before termination.

Issue 1793

### Fixed an incorrect check that nstlog != 0 for expanded ensembles

The original version was accidentally reversed, causing it to fail when nstlog was not equal to 0.

### Fixes for `gmx` tools

### Fixed `gmx tune_pme` detection of GPU support

### Fixed spacing in `gmx tune_pme` call to thread-MPI mdrun

### Fixed minor issues in `gmx traj -av -af`

Made the description of the xvg y-axis more useful. Also works for option `-af`.

Issue 2133

### Removed rogue printing to xvg file in gmx mindist

`gmx mindist -xvg` none is now adhered to, and printing is preceded by a comment.

Issue 2129

### Fixed bug in `gmx solvate -shell` if it yielded 0 SOL.

In the transition from genbox to solvate, some incorrect logic was introduced.

Issue 2119

### Corrected output of `gmx do_dssp -sc`

This code has always written a probability, and not a percentage, so fixed the label. It still fits within the expected 8-character field.

Issue 2120

### Improved documentation

#### Made several minor improvements to documentation and messages to users

Removed documentation of unimplemented `gmx trjconv -clustercenter`.

Introduced system preparation section to user guide, to create somewhere to document the use and limitations of vdwradii.dat. Enchanced documentation of solvate and insert-molecules, similarly.

Issue 2094

#### Documented that we now support AMD GCN on Mesa/LLVM

AMD GPUs using Mesa 17.0+ and LLVM 4.0+ run GROMACS using OpenCL.

#### Documented running Clang static analyzer manually

### Portability enhancements

#### Enabled avx512 in the GROMACS FFTW build only if the compiler supports it

Enabling avx512 requires GCC 4.9 or newer or Clang 3.9 or newer. Since we support compilers older than those, we can not afford to enable avx512 in `GMX_BUILD_OWN_FFTW=on` unconditionally.

#### Worked around false positives in SIMD test from bug in xlc 13.1.5

atan2(0,0) should return 0.0, which the GROMACS simd implementation does. However, since at least one compiler produces -nan for the standard library version it's better to compare with the known correct value rather than calling std:atan2(0,0).

Issue 2102

#### Fixed compile with icc of `GMX_SIMD=None`

ICC defines invsqrt in math.h

### GROMACS 2016.2 Release Notes

This version was released on February 7, 2016. These release notes document the changes that have taken place in GROMACS since version 2016.1 to fix known issues. It also incorporates all fixes made in version 5.1.4 and several since.

### Fixes where mdrun could behave incorrectly

#### Add grompp check for equipartition violation risk for decoupled modes

When atoms involved in an angle with constrained bonds have very different masses, there can be very weakly coupled dynamics modes. Default mdp settings are often not sufficiently accurate to obtain equipartitioning. This change adds a grompp check for this issue.

Part of Issue 2071

#### Disallow overwriting of dihedral type 9

It is no longer allowed to repeat blocks of parameter of multiple lines for dihedrals of type 9. It is also no longer allowed to override parameters or dihedrals of type 9. Both are too complex to properly check. It is still allowed to repeat parameters blocks consisting of a single line. Repeating a block with the same parameters would lead to incorrect dihedral potentials and forces.

Issue 2077

#### Fixed flat-bottom position restraints + DD + OpenMP

A (re)allocation was missing, causing a crash.

Issue 2095

#### Fixed multi-domain reruns

Old code cleanup led multi-domain rerun to crash because it failed to consider logic separated over two places.

Issue 2105

### Fixes for mdrun performance issues

#### Corrected CUDA sm_60 performance

The kernel launch now suits the SM size of the GP100 architecture.

### Fixes for `gmx` tools

#### Fixed some FFT handling in cross-corrrelation calculations

An array of complex number was created as an array of pointers and then passed to gmx_fft_1d. This does not work.

Issue 2109

### Fixed gmx rmsf -q -oq

This led to the PDB file containing B-factors using coordinates based on those from the -s file, rather than -q file. gmx rmsf -oq was otherwise fine.

### Fixed crash in gmx order

gmx order used a cumbersome floating point method to compute a histogram, leading to an index value that could be negative.

Issue 2104

### Fixed minor trjconv bug

gmx trjconv -novel -f in.pdb -o out.pdb probably works better now.

### Fixed time label print in gmx vanhove

### Handled issuing warnings correctly in xpm2ps and membed

The code should not (over)write the output file before checking for errors. For membed, it is useful to require the user to fix issues in their input file before we unilaterally over-write it.

### Corrected documentation about eigenvalue handling

Some file format docs were out of step with the implementation in eigio.cpp.

The behaviour of gmx anaeig -eig -eig2 was not properly documented.

### Made editconf B-factor attachment more useful in practice

B-factor values will be added to residues unless an index is larger than the number of residues or an option is specified. Protein residue indices can start from any number and, in case they start from a large number, there is no way to add B-factor values to residues.

This patch changes it to add B-factor values to residues unless the number of B-factor values is larger than the number of residues.

### Fixed possible memory error with long selections

If a selection was more than 1000 characters long and there was a whitespace exactly at the 1000 point, a buffer overflow could occur. Replaced the buffer with std::string, simplifying the code significantly.

Issue 2086

### Fixed use of position variables with plus/merge

If a selection contained a position variable (e.g., 'com of ...') that was used more than once, and at least one of those uses was with plus/merge, there were out-of-bounds memory writes. This was caused by the internal position structure not getting fully initialized. Incomplete initialization happens in all contexts with such variables, but only plus/merge (and possibly permute) actually use the values that remained uninitialized, which caused them to incorrectly compute the amount of memory required to store the result.

Issue 2086

## Improved documentation

### Made several minor improvements to documentation and messages to users

In particular, for selections:

- Explained resindex and resnr keywords in selection help.
- Explained how selection-enabled tools treat -s and -f input files.

Issue 2083

### Clarified use of tau-p and pcoupltype

grompp used to permit the erroneous "tau-p = 5 5". This does not reflect that only one time constant is permitted for pressure coupling (unlike group-based temperature coupling). The recent fix for Issue 1893 leads to the user receiving a grompp warning, so this improves the docs to make clear that pressure coupling is different.

Issue 1893

## Portability enhancements

### Fixed x86 conditional on IBM s390x

The CpuInfoTest.SupportLevel test fails on IBM s390x because wrong condition was used.

Fixes: https://bugzilla.redhat.com/show_bug.cgi?id=1390149

Issue 2072

### Build system enhancements

#### Fixed compilation with CMAKE_CXX_FLAGS="-Wall -Werror"

Issue 2073

#### Stopped trying to use objdump –reloc in the build system on Mac

Recent Xcode objdump does not support –reloc.

The warning that is based on the output of running objdump was only implemented to work on Linux-like things, so should not spam the cmake output on other platforms.

#### Improved the support for plugin loading in the build system

The mdrun-only and prefer-static-libs builds set the default for BUILD_SHARED_LIBS to off, which silently disabled plugin support for things like VMD-based I/O handling.

Converted GMX_LOAD_PLUGINS to tri-state ON/OFF/AUTO so that if the preconditions for support are not met we can have suitable behaviour in each case.

Issue 2082

#### Turn off hwloc support when static lib found

Hwloc dependencies are not resolved at CMake time when static libwloc.a is detected and in most of these cases link-time errors will prevent building GROMACS. As it is hard for a user to know how to solve such cryptic errors and hwloc is not a required dependency, we turn off hwloc support when a static lib is detected. The user can override this on the cmake command line.

Issue 1919

#### Fixed build with GMX_EXTERNAL_TNG=ON

### House-keeping that reduces users' problems

#### Mdrun prints invalid performance data less often

If mdrun finished before a scheduled reset of the timing information (e.g. from mdrun -resetstep or mdrun -resethway), then misleading timing information should not be reported.

Related, the default reset step for gmx tune_pme was increased to 1500.

Issue 2041

### Added a runtime check for number of threads in bonded code

Replaced a debug assertion on the number of OpenMP threads not being larger than GMX_OPENMP_-MAX_THREADS by fatal error. But since the listed-forces reduction is actually not required with listed forces, these are now conditional and mdrun can run with more than GMX_OPENMP_MAX_-THREADS threads.

Issue 2085

### Fixed integer narrowing in TNG reading for long trajectories

Reading of TNG trajectories with sufficiently large numbers of frames could truncate integers used for frame numbers. Fixed to use 64-bit integers as originally intended.

### Fixed logic of TRR reading

When reading a trr file, reaching the end of the file was indistinguishable from a reading error or a magic-number error. This is now fixed, restoring the intended behaviour in each case.

Issue 1926

### GROMACS 2016.1 Release Notes

This version was released on October 28, 2016. These release notes document the changes that have taken place in GROMACS since the initial version 2016 to fix known issues. It also incorporates all fixes made in version 5.1.4.

### Made distance restraints work with threads and DD

The NMR distance restraints use several buffers for summing distances that were indexed based on the index of the thread+domain local ilist force atoms. This gives incorrect results with OpenMP and/or domain decomposition. Using the type index for the restraint and a domain- local, but not thread-local index for the pair resolves these issues. The are now only two limitations left:

- Time-averaged restraint don't work with DD.
- Multiple copies of molecules in the same system without ensemble averaging does not work with DD.

Note that these fixes have not been made in any 5.1.x release.

Issue 1117 Issue 1989 Issue 2029

### Fixed Ewald surface+3DC corrections

Ewald surface and 3DC correction forces were only applied up to, but not including, the last atom with exclusions. With water at the end of the system only the last H would not be corrected. With ions at the end all ions would be missing. In addition, with the Verlet scheme and domain decomposition no force correction was applied at all.

Issue 2040

### Fixed opening of wall table files

Issue 2033

### Fixed bug in gmx insert-molecules.

With option -ip, and if all trials were unsuccessful, a molecule was eventually incorrectly placed at 0/0/0 due to a memory error when referencing to rpos[XX][mol].

### Made virial reproducible

OpenMP reduction was used to reduce virial contributions over threads, which does not have a defined order. This leads to different rounding, which makes runs non-reproducible (but still fully correct). Now thread local buffers are used. Also removed OpenMP parallezation for small count (e.g. shift forces).

### Updated to support FFTW 3.3.5

The auto-download of FFTW now gets FFTW 3.3.5 and builds it properly, including with `--enable-vsx` when GMX_SIMD is set to VSX, i.e. for Power8, and `--enable-avx512` when GMX_SIMD is any of the AVX flavours (which is safe on non-512 now, works on KNL, and is presumed useful for future AVX512 architectures).

### Permitted automatic load balancing to disable itself when it makes the run slower

Under certain conditions, especially with (shared) GPUs, DLB can decrease the performance. We now measure the cycles per step before turning on DLB. When the running average of cycles per step with DLB gets above the average without DLB, we turn off DLB. We then measure again without DLB. If without DLB the cycle count is still lower, we keep DLB off for the remainder of the run. Otherwise is can turn on again as before. This procedure ensures that the performance will never deteriorate due to DLB.

### Improved the accuracy of timing for dynamic load balancing with GPUs

With OpenCL, the time for the local non-bonded to finish on the GPU was ignored in the dynamic load balancing. This change lets OpenCL take the same code path as CUDA.

One internal heuristic parameter was far too small for both CUDA and OpenCL, which is now fixed.

### Corrected kernel launch bounds for Tesla P100 GPUs

This corrects our initial guess of kernel tuning parameters that resulted in reduced occupancy on sm_60 GPU, and thus improves performance.

### Improved logic handling if/when the run is terminated for SETTLE warnings

The code now honours that when the environment variable GMX_MAXCONSTRWARN is set to -1, there is no maximum number of warnings.

Issue 2058

### Fixed bug in gmx wham for reading pullx files.

Because the order of columns in the pullx files has changed recently, `gmx wham` did not pick the reaction coordinate from `pullx.xvg` if the COM of the pull groups were written. `gmx wham` was tested with various pull options and geometries.

### Fixed ouput bug in gmx wham

### Fixed deadlock with thread-MPI

With thread-MPI mdrun could deadlock while pinning threads.

Issue 2025

### Made error reporting in grompp more user friendly

This tool now always reports the file and line in user input files that lead to a condition such that subsequent parsing cannot continue.

### Fixed SIMD suggestion for VMX

### Fixed script xplor2gmx.pl to work with GMXDATA

### Fixed default nice level in mdrun-only build

Now an mdrun-only build should default to zero nice level, the same as `gmx mdrun` in a normal build.

### Fixed math-test false positive

Depending on the accuracy of the floating point division, the input of the test function could be 1ulp too large or too small. If it was too large the result of the test function wasn't within 4ulp and the test failed.

### Improved documentation

Approaches for reducing overhead for GPU runs are now documented.

The available wallcycle counters and subcounters reported in the md.log files are now listed and explained in the user guide, along with how to enable reporting of the subcounters.

Several install-guide sections have been improved, including those for OpenCL, mdrun-only, and "make check". A "quick and dirty" cluster installation section was added.

### OpenCL error string are now written, instead of cryptic error codes

### Fixed build with GMX_USE_TNG=off

### Removed variable-precision .gro writing

The precision used when writing .gro files is now fixed to 3, 4 and 5 decimal places for x, v and box respectively to ensure compatibility with other software. Variable-precision reading is still supported.

Issue 2037

### Fixed BG/Q platform files and install guide

Renamed the platform file to reflect normal practice and the install guide.

#### Reduced the memory required for free-energy simulations

Pair lists with atoms whose short-ranged parameters are perturbed now use less memory.

Issue 2014

### 11.12.2 Major release

#### Highlights

GROMACS 2016 was released on August 4, 2016. Patch releases have been made since then, please use the updated versions! Here are some highlights of what you can expect, along with more detail in the links below!

- As always, we've got several useful performance improvements, with or without GPUs. CPU-side SIMD and threading enhancements will make GPU-accelerated simulations faster even if we'd left the GPU code alone! Thanks to these and additional GPU kernel improvements, in GPU-accelerated runs expect around 15% improvement in throughput. (And not just for plain vanilla MD, either... the pull code now supports OpenMP threading throughout, and multi-simulations have less coupling between simulations.)
- We have a new C++11 portability layer permitting us to accelerate in SIMD on the CPU lots of minor routines. These will also often improve runs that use accelerators or many nodes through better load balancing. POWER8, ARM64, AVX512 (KNL), and more are fully SIMD accelerated now because they are supported in the new portability layer!
- We made further SIMD acceleration of bonded interactions which reduces their calculation time by about a factor of 2. This improves load balance at high parallelization by a factor of 2, and shows significantly better scaling.
- Similarly, SIMD acceleration of SETTLE reduces the time for constraints by a factor of 3 to 5 - which has a strong effect for GPU runs.
- OpenCL GPU support is now available with all combinations of MPI, thread-MPI and GPU sharing (ie. the same as CUDA). Kernel performance has improved by up to 60%. AMD GPUs benefit the most, OpenCL on NVIDIA is generally still slow.
- Tools in the new analysis framework can handle trajectories that are subsets of the simulation system.
- New pull coordinate geometries angle-axis, dihedral, and normal angle.
- Checkpoint restarts work only in the cases where the implementation can always do what the user wants.
- The version numbering has changed to be the year of the release, plus (in future) a patch number. GROMACS 2016 will be the initial release from this branch, then GROMACS 2016.1 will have the set of bugs that have been fixed in GROMACS 2016, etc.

### New and improved features

#### Changed to require a C++11 compiler

GROMACS now requires both a C++11 and C99 compiler. For details, see the install guide.

#### Changed to support only CUDA 5.0 and more recent versions

Issue 1831

#### Allowed rcoulomb > rvdw with PME

GROMACS has had kernels that support Coulomb PME + cut-off LJ with rcoulomb > rvdw for a while, but these were only available via PME load balancing. Now we allow this setup to be chosen also through mdp options.

#### Added optional support for portable hardware locality (hwloc)

Added CMake support to detect and build GROMACS with hwloc, which will improve GROMACS ability to recognize and take advantage of all the available hardware. If hwloc is unavailable, GROMACS will fall back on other detection routines.

#### Made normal-mode calculations work with shells and vsites

Implemented shells and vsites in normal-mode analysis in mdrun and in analysis of eigenvalues and frequencies. The normal-mode analysis is done on real atoms only and the shells are minimized at each step of the analysis.

Issue 879

#### Changed pull group count for coords stored in tpr file

Added a parameter ngroup to the pull coord parameters. This is now also stored in the tpr file. This makes the pull geometry forward compatible, which is useful since it avoid bumping the .tpr version with every new geometry, and we expect that users want to experiment with new geometries.

#### Added pull coordinate geometry angle-axis

The new geometry is described in the docs. Some checks in readpull.cpp where reorganized since adding new geometries made some old logic a bit convoluted.

### Added pull coordinate geometry dihedral (angle)

How to use the new geometry is explained in the docs.

### Added pull coordinate geometry angle

A new subsection was added to the docs explaining the new geometry.

### Replaced `pull-print-com1,2` mdp option with `pull-print-com`

Changes were made to the pull output order and naming.

### Added pull potential flat-bottom-high

Added the new pull coordinate type flat-bottom-high, which is a flat potential above the reference value and harmonic below.

### Added `gmx grompp` check for pull group

Added a check for valid pull groups in a pull coordinate. Using a pull group index that was out of range would cause invalid memory access.

### Added new swapping functionality to computational electrophysiology module

Support was added for ion/water position swapping for multiple ion types and polyatomic ions, including use of a user-defined number of ionic species, and (small) polyatomic ions.

Also added two extra .mdp file parameters 'bulk-offset' that allow the user to specify an offset of the swap layers from the compartment midplanes. This is useful for setups where e.g. a transmembrane protein extends far into at least one of the compartments. Without an offset, ions would be swapped in the vicinity of the protein, which is not wanted. Adding an extended water layer comes at the cost of performance, which is not the case for the offset solution.

Documentation and testing was improved.

### Fixed logic for DD missing-interactions check

The code that was intended to double check that the domain decomposition algorithm has not missed any interactions was inactive in several cases, and has been fixed.

Issue 1882, Issue 1793

#### Permitted forces and velocities to be written to compressed TNG

If there is no uncompressed coordinate output, write forces and velocities to the TNG file with compressed coordinate output. If there is uncompressed coordinate output to a TNG file, forces and velocities will be written to it.

Use a greatest common divisor to set the frequency of some TNG data output to ensure lambdas and box shape are written at least as often as anything else.

Issue 1863

#### Added new notes to the user when coupling algorithms are unavailable

mdrun will now give the user an explanatory note when pressure and/or temperature coupling is turned off.

#### Added mdrun check for finite energies

Added a check that the total potential energy is finite. This check is nearly free and can catch issues with incorrectly set up systems before users get a confusing constraint or PME error. Note that this check is only performed at steps where energies are calculated, so it will often not catch an exploding system.

#### Added `gmx grompp` check for unbound atoms

`gmx grompp` now prints a note for atoms that are not connected by a potential or constraint to any other atom in the same moleculetype, since this often means the user made a mistake.

Issue 1958

#### Improved multi-simulation signalling

Multi-simulations (including REMD) may have need to send messages between the simulations. For example, REMD needs to write a fully-consistent set of checkpoint files so that the restart works correctly, but normal multi-simulations are fine with decoupled progress and will simulate more efficiently if they can do so. Similarly, `gmx_mpi mdrun -maxh -multi` needs to synchronize only for REMD. The implementation has been upgraded so that such coupling happens only when an algorithm chosen by the user requires it.

Issue 860, Issue 692, Issue 1857, Issue 1942

Changed multi-simulation nsteps behaviour “”””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””””’-

It is unclear what the expected behaviour of a multi-simulation should be if the user supplies any of the possible non-uniform distributions of init_step and nsteps, sourced from any of .mdp, .cpt or command line. Previously mdrun adjusted the total number of stesps to run so that each run did the same number of steps, but now it reports on the non-uniformity and proceed, assuming the user knows what they are doing.

Issue 1857

### Added working directory to things reported in .log file

When running GROMACS via a batch script, it is useful to know which working directory is being used for relative paths (file names) in the command line. This is now written alongside other header information.

### Prevented fragile use cases involving checkpoint restarts and/or appending

All output files named in the checkpoint file (ie. that were used in the previous run) must be present before a checkpoint restart will be permitted. Thus, workflows where people used things like `gmx mdrun -s production -cpi equilibration` are no longer available to do a "continuous" restart. Instead, use `gmx grompp -t equilibration -o production`.

Issue 1777

### Removed warning after OpenMP core-count check

In many cases `gmx_mpi mdrun` issued a warning that compared the total core count with something different returned from OpenMP. This problem is caused by inappropriate management of thread affinity masks, but the wording of the message did not help the user realise this, so has been removed. `gmx_mpi mdrun -pin on` may help improve performance in such cases.

### Preparation for hardware detection might try to force offline cores to work

Hardware detection might be foiled by kernels that take cores offline when work is unavailable. We are not aware of any such platforms on which GROMACS is likely to be used, but we will probably start to see them soon. On such platforms, if the number of cores physically present differs from the count that are online, we try to force them online before deciding how GROMACS will use the online cores. For now, no x86 or PowerPC platforms need such code, so it will never run on those platforms. The good news is that we no longer have to risk making a confusing warning about such possibilities.

### Added new suggestion for users to try e.g. hyper-threading, if its disabled

GROMACS tends to perform best with several hardware threads available per core (e.g. hyper-threading turned on, on x86), and now the log file will note explicitly when such opportunities exist.

### Performance improvements

### GPU improvements

In addition to those noted below, overall minor improvements contribute up to 5% increase in CUDA performance, so depending on parameters and compilers an 5-20% GPU kernel performance increase is expected. These benefits are seen with CUDA 7.5 (which is now the version we recommend); certain older versions (e.g. 7.0) see even larger improvements.

Even larger improvements in OpenCL performance on AMD devices are expected, e.g. can be >50% with RF/plain cut-off and PME with potential shift with recent AMD OpenCL compilers.

Note that due to limitations of the NVIDIA OpenCL compiler CUDA is still superior in performance on NVIDIA GPUs. Hence, it is recommended to use CUDA-based GPU acceleration on NVIDIA hardware.

### Improved support for OpenCL devices

The OpenCL support is now fully compatible with all intra- and inter-node parallelization mode, including MPI, thread-MPI, and GPU sharing by PP ranks. (The previous limitations were caused by bugs in high-level GROMACS code.)

Additionally some prefetching in the short-ranged kernels (similar to that in the CUDA code) that had been disabled was found to be useful after all.

### Added Lennard-Jones combination-rule kernels for GPUs

Implemented LJ combination-rule parameter lookup in the CUDA and OpenCL kernels for both geometric and Lorentz-Berthelot combination rules, and enabled it for plain LJ cut-off. This optimization was already present in the CPU kernels. This improves performance with e.g. OPLS, GROMOS and AMBER force fields by about 10-15% (but does not help with CHARMM force fields because they use force-switched kernels).

### Added support for CUDA CC 6.0/6.1

Added build-system and kernel-generator support for the Pascal architectures announced so far (GP100: 6.0, GP104: 6.1) and supported by the CUDA 8.0 compiler.

By default we now generate binary as well as PTX code for both sm_60 and sm_61 and given the considerable differences between the two, we also generate PTX for both virtual arch. For now we don't add CC 6.2 (GP102) compilation support as we know nothing about it.

On the kernel-generation side, given the increased register file, for CC 6.0 the "wider" 128 threads/block kernels are enabled, on 6.1 and later the 64 threads/block remains.

### Improved GPU pair-list splitting to improve performance

Instead of splitting the GPU lists (to generate more work units) based on a maximum cut-off, we now generate lists as close to the target list size as possible. The heuristic estimate for the number of cluster pairs is now too high by 0-1% instead of 10%. This results in a few percent fewer pair lists, but still slightly more than requested.

### Improved CUDA GPU memory configuration

This makes use of the larger amount of L1 cache available for global load caching on hardware that supports it (K40, K80, Tegra K1, & CC 5.2) by passing the appropriate command line option ("-dlcm=ca").

Issue 1804

### Automatic nstlist changes were tuned for Intel Knight's Landing

### CPU improvements

These improvements to individual kernels will provide incremental improvements to CPU performance for simulations where they are active, but their value for simulations using GPU offload are much higher, because via the auto-tuning, they permit all kinds of resource utilization and throughput to increase.

### Optimized the bonded thread force reduction

The code for multi-threading of bonded interactions has to combine the forces afterwards. This reduction now uses fixed-size blocks of 32 atoms, and instead of dividing reduction of the whole range of blocks uniformly over the threads, now only used blocks are divided (uniformly) over the threads. This speeds up the reduction by a factor of the number of threads (!) for typical protein+water systems when not using domain decomposition. With domain decomposition, the speed up is up to a factor of 3.

### Used SIMD transpose-scatter in bonded force reduction

The angle and dihedral SIMD functions now use the SIMD transpose scatter functions for force reduction. This change gives a massive performance improvement for bondeds, mainly because the dihedral force update did a lot of vector operations without SIMD that are now fully replaced by SIMD operations.

### Added SIMD implementation of Lennard-Jones 1-4 interactions

The gives a few factors speed improvement. The main improvement comes from simplified analytical LJ instead of tables; SIMD helps a bit.

### Added SIMD implementation of SETTLE

On Haswell CPUs, this makes SETTLE a factor 5 faster.

### Added SIMD support for routines that do periodic boundary coordinate transformations

### Threading improvements

These improvements enhance the performance of code that runs over multiple CPU threads.

### Improved Verlet-scheme pair-list workload balancing

Implemented near perfect load-balancing for Verlet-scheme CPU pair-lists. This increases the search cost by 3%, but this is outweighed by the more balanced non-bonded kernel times, particularly for small systems.

### Improved the threading of virtual-site code

On many threads, a significant part of the vsites would end up in the separate serial task, thereby limiting scaling. Now two weakly dependent tasks are generated for each thread and one of them uses a thread-local force buffer, parts of which are reduced by different threads that are responsible for those parts.

Also the setup now runs multi-threaded.

### Add OpenMP support to more loops

Loops over number of atoms cause significant amount of serial time with large number of threads, which limits scaling.

### Add OpenMP parallelization for the pull code

The pull code could take up to a third of the compute time for OpenMP parallel simulation with large pull groups. Now all pull-code loops over atoms have an OpenMP parallel version.

### Other improvements

### Multi-simulations are coupled less frequently

For example, replica-exchange simulations communicate between simulations only at exchange attempts. Plain multi-simulations do not communicate between simulations. Overall performance will tend to improve any time the progress of one simulation might be faster than others (e.g. it's at a different pressure, or using a quieter part of the network).

### Improvements to GROMACS tools

### Supported replacing solvent in `gmx insert-molecules`

Make it possible to specify the solvent (or other set of atoms) with `-replace` (as a selection) for `gmx insert-molecules`, and make the tool replace residues from this set with the inserted molecules, instead of not inserting there. It is assumed that the solvent consists of single-residue molecules, since molecule information would require a tpr input, which might not be commonly available when preparing the system.

### Default random seeds have changed for some analysis tools

See individual tools documentation for their functionality. In some cases, the magic value to obtain a generated seed has changed (or is now documented.)

### Made `gmx solvate` and `gmx insert-molecules` work better with PDB inputs

When both `-f` and `-o` were .pdb files, the pdbinfo struct got out-of-sync when the atoms were added/removed.

Issue 1887

### Tools in the new analysis framework can read trajectory files with subsets

Make tools written for the new C++ analysis framework support analyzing trajectories that contain an arbitrary subset of atoms.

Issue 1861

### Made moleculetype name case sensitive

This is useful in case you have more than 36 chains in your system with chain IDs set. PDB allows using both uppercase letters, lowercase letters and numbers for chain identifiers. Now we can use the maximum of 62 chains.

### Added number density normalization option for `gmx rdf`

Add an option to `gmx rdf` that allows selecting a radial number density as the normalization for the output (in addition to current raw neighbor counts and the actual RDF).

### Simplified `gmx genconf` by removing `-block`, `-sort` and `-shuffle`

Option `-block` isn't useful since particle decomposition was removed. Options `-sort` and `-shuffle` were undocumented and don't seem very useful - these days they would be somebody's simple python script.

### Used macros for units and conversions in `gmx wham`

Also Issue 1841

### Improved `gmx sasa` error message

Print more information when an output group is not part of the group selected for calculation, which should help the user diagnosing the issue.

### Made `gmx vanhove` work without PBC

### Fix `gmx hbond` group overlap check

`gmx hbond` does not support partially overlapping analysis groups. The check in the code was broken and never caught this, resulting incorrect output that might OK at first sight. Also corrected bitmasks = enums that (intentionally?) seemed to give correct results by not using non power of 2 enum index entries.

### Made `gmx dos` work again.

Due to an error in the index handling `gmx dos` always stopped with a fatal error.

Issue 1996

### Add checks for too much memory in `gmx nmeig`

`gmx nmeig` could request storage for eigenvector output and matrices for more than `INT_MAX` elements, but nearly all loop variables are int. Now a fatal error is produced in this case. This also avoids the confusing error message when too much memory is requested; the allocation routine will get the correct size, but gmx_fatal prints it as a smaller integer. Added support for `-first` > 1 with sparse matrices.

## Bugs fixed

These document fixes for issues that have been fixed for the 2016 release, but which have not been back-ported to other branches.

### Fixed two problems related to restarts for velocity-Verlet

The first problem is more serious; in addition to causing problems with restarts in most cases for velocity-Verlet integrators plus either Berendsen or v-rescale temperature-coupling algorithms, the temperature coupling code was called twice. This made the distribution of kinetic energies too broad (but with the correct average). Other algorithm combinations were unaffected.

In the second problem, the initial step after restarts with velocity-Verlet integrators and either Berendsen or v-rescale temperature-coupling algorithms had too high a pressure because they used an empty virial matrix that was only filled with MTTK pressure control. The effects of this bug were very small; it only affected the volume integration for one step on restarts.

Issue 1883

### Fixed Verlet buffer calculation with nstlist=1

Under rare circumstances the Verlet buffer calculation code was called with nstlist=1, which caused a division by zero. The division by zero is now avoided. Furthermore, grompp now also determines and prints the Verlet buffer sizes with nstlist=1, which provider the user information and adds consistency checks.

Issue 1993

### Fixed large file issue on 32-bit platforms

At some point gcc started to issue a warning instead of a fatal error for the checking code; fixed to really generate an error now.

Issue 1834

### Avoided using abort() for fatal errors

This avoids situations that produce useless core dumps.

Issue 1866

### Fixed possible division by zero in polarization code

### Avoided numerical overflow with overlapping atoms in Verlet scheme

The Verlet-scheme kernels did not allow overlapping atoms, even if they were not interacting (in contrast to the group kernels). Fixed by clamping the interaction distance so it can not become smaller than ~6e-4 in single and ~1e-18 in double, and when this number is later multiplied by zero parameters it will not influence forces. The clamping should never affect normal interactions; mdrun would previously crash for distances that were this small. On Haswell, RF and PME kernels get 3% and 1% slower, respectively. On CUDA, RF and PME kernels get 1% and 2% faster, respectively.

Issue 1958

### Relax pull PBC check

The check in the pull code for COM distances close to half the box was too strict for directional pulling. Now dimensions orthogonal to the pull vector are no longer checked. (The check was actually not strict enough for directional pulling along x or y in triclinic units cells, but that is a corner case.) Furthermore, the direction-periodic hint is now only printed with geometry direction.

Issue 1962

### Add detection for ARMv7 cycle counter support

ARMv7 requires special kernel settings to allow cycle counters to be read. This change adds a cmake setting to enable/disable counters. On all architectures but ARMv7 it is enabled by default, and on ARMv7 we run a small test program to see if the can be executed successfully. When cross-compiling to ARMv7 counters will be disabled, but either choice can be overridden by setting a value for GMX_CYCLECOUNTERS in cmake.

Issue 1933

### Introduced fatal error for too few frames in `gmx dos`

To prevent `gmx dos` from crashing with an incomprehensible error message when there are too few frames, test for this.

Part of Issue 1813

### Properly reset CUDA application clocks

We now store the application clock values we read when starting mdrun and reset to these values, but only when clocks have not been changed (by another process) in the meantime.

Issue 1846

### Fixed replica-exchange debug output to all go to the debug file

When `mdrun -debug` was selected with replica exchange, some of the order description was printed to mdrun's log file, but it looks like the actual numbers were being printed to the debug log. This puts them both in the debug log.

### Fixed gmx mdrun -membed to always run on a single rank

This used to give a fatal error if default thread-MPI mdrun had chosen more than one rank, but it will now correctly choose to use a single rank.

### Fixed issues with using int for number of simulation steps

Mostly we use a 64-bit integer, but we messed up a few things.

During mdrun -rerun, edr writing complained about the negative step number, implied it might be working around it, and threatened to crash, which it can't do. Silenced the complaint during writing, and reduced the scope of the message when reading.

Fixed TNG wrapper routines to pass a 64-bit integer like they should.

Made various infrastructure use gmx_int64_t for consistency, and noted where in a few places the practical range of the value stored in such a type is likely to be smaller. We can't extend the definition of XTC or TRR, so there is no proper solution available. TNG is already good, though.

Issue 2006

### Fixed trr magic-number reading

The trr header-reading routine returned an "OK" value even if the magic number was wrong, which might lead to chaotic results everywhere. This led to problems if other code (e.g. cpptraj) mistakenly wrote a wrong-endian trr file, which was then used with GROMACS. (This should never be a thing for XDR files, which are defined to be big endian, but such code has existed.)

Issue 1926

### Changed to use only legal characters in OpenCL cache filename

The option to cache JIT-compiled OpenCL short-ranged kernels needed to be hardened, so that mdrun would write files whose names would usually be specific to the device, but also only contain filenames that would work everywhere, ie only alphanumeric characters from the current locale.

### Fixes for bugs introduced during development

These document fixes for issues that were identified as having been introduced into the release-2016 branch since it diverged from release-5-1. These will not appear in the final release notes, because no formal release is thought to have had the problem. Of course, the tracked issues remain available should further discussion arise.

### Fixed bug in v-rescale thermostat & replica exchange

Commit 2d0247f6 made random numbers for the v-rescale thermostat that did not vary over MD steps, and similarly the replica-exchange random number generator was being reset in the wrong place.

Issue 1968

### Fixed vsite bug with MPI+OpenMP

The recent commit b7e4f30d caused non-local virtual sites not be treated when using OpenMP. This means their coordinates lagged one step behind and their forces are not spread to the atoms, leading to small errors in the forces. Note that non-local virtual sites are only used when local virtual sites use them as a constructing atom; the most common case is a C/N in a CH3/NH3 group with vsite H's. Also added a check on the vsite count for debug builds.

Issue 1981

### Fixed some thread affinity cases

Fixed one deadlock in newly refactored thread-affinity code, which happened with automatic pinning, if only part of the nodes were full.

There is one deadlock still theoretically possible: if thread-MPI reports that setting the affinity is not possible only on a subset of ranks, the code deadlocks. This has always been there and might never happen, so it is not fixed here.

### Removed OpenMP overhead at high parallelization

Commit 6d98622d introduced OpenMP parallelization for for loops clearing rvecs of increasing rvecs. For small numbers of atoms per MPI rank this can increase the cost of the loop by up to a factor 10. This change disables OpenMP parallelization at low atom count.

### Removed std::thread::hardware_concurrency()

We should not use std::thread::hardware_concurrency() for determining the logical processor count, since it only provides a hint. Note that we still have 3 different sources for this count left.

### Added support for linking against external TinyXML-2

This permits convenient packaging of GROMACS by distributions, but it got lost from gerrit while rebasing.

Issue 1956

### Fixed data race in hwinfo with thread-MPI

Issue 1983

### Fixes for Power7 big-endian

Now compiles and passes all tests in both double and single precision with gcc 4.9.3, 5.4.0 and 6.1.0 for big-endian VSX.

The change for the code in incrStoreU and decrStoreU addresses an apparent regression in 6.1.0, where the compiler thinks the type returned by vec_extract is a pointer-to-float, but my attempts a reduced test case haven't reproduced the issue.

Added some test cases that might hit more endianness cases in future.

We have not been able to test this on little-endian Power8; there is a risk the gcc-specific permutations could be endian-sensitive. We'll test this when we have hardware access, or if somebody runs the tests for us.

Issue 1997 Issue 1988

### Reduce hwloc & cpuid test requirements

On some non-x86 linux platforms hwloc does not report caches, which means it will fail our strict test requirements of full topology support. There is no problem whatsoever with this, so we reduce the test to only require basic support from hwloc - this is still better than anything we can get ourselves. Similarly for CPUID, it is not an error for an architecture to not provide any of the specific flags we have defined, so avoid marking it as such.

Issue 1987

### Work around compilation issue with random test on 32-bit machines

gcc 4.8.4 running on 32-bit Linux fails a few tests for random distributions. This seems to be caused by the compiler doing something strange (that can lead to differences in the lsb) when we do not use the result as floating-point values, but rather do exact binary comparisions. This is valid C++, and bad behaviour of the compiler (IMHO), but technically it is not required to produce bitwise identical results at high optimization. However, by using floating-point tests with zero ULP tolerance the problem appears to go away.

Issue 1986

### Updated `gmx wham` for the new pull setup

This bring `gmx wham` up to date with the new pull setup where the pull type and geometry can now be set per coordinate and the pull coordinate has changed and is more configurable.

### Fix membed with partial revert of 29943f

The membrane embedding algorithm must be initialized before we call init_forcerec(), so it cannot trivially be moved into do_md(). This has to be cleaned up anyway for release-2017 since we will remove the group scheme be then, but for now this fix will allow us have the method working in release-2016.

Issue 1998

### Removed mdrun features

### Removed SD2 integrator

This integrator has known problems, and is in all ways inferior to sd. It has no tests, and was deprecated in GROMACS 5.0. There are no plans to replace it.

Issue 1137

### Removed the twin-range scheme

Only the (deprecated) group scheme supports this, and the Verlet scheme will not support it in the foreseeable future. There is now the explicit requirement that rlist >= max(rcoulomb,rvdw).

### Removed support for twin-range with VV integrators

Group-scheme twin-ranged non-bonded interactions never worked with velocity-Verlet integrators and constraints. There are no plans to make that combination work.

Issue 1137, Issue 1793

### Removed Reaction-Field-nec

The reaction-field no-exclusion correction option was only introduced for backward compatibility and a performance advantage for systems with only rigid molecules (e.g. water). For all other systems the forces are incorrect. The Verlet scheme does not support this option and even if it would, it wouldn't even improve performance.

### Removed AdResS module

This feature requires the (deprecated) group scheme, and there are no plans to port it to the Verlet scheme.

Issue 1852

### Removed mdrun -compact

It is too complicated to support multiple ways of analysing per-step data.

### Removed lambda printing from mdrun log file

Issue 1773

### Removed GMX_NOCHARGEGROUPS

This undocumented feature was only useful with the (deprecated) group scheme.

### Miscellaneous

### Various improvements to documentation and tests

In particular, the definition of pressure in the reference manual should be in bar, and a spurious r_ij in the force for the Morse potential was removed. Added documentation and literature references for membrane embedding. Improved template analysis program documentation. gmock was patched to work with gcc 6.

Issue 1932

### Improved make_ndx help text

Clarified the use of boolean operators. The old help text could incorrectly hint that AND, OR, and NOT would work as keywords. Added a reference to `gmx select` that in most cases can serve as a replacement.

Issue 1976

### Addded checks on number of items read in mdp statements

Added checks for the number of items read in all sscanf() statements processing data from the mdp file.

Issue 1945.

### Work around glibc 2.23 with CUDA

glibc 2.23 changed the behaviour of string.h in a way that broke all versions of CUDA with all gcc compiler versions. The GROMACS build system detects this glibc, and works around it by adding the _FORCE_INLINE preprocessor define to CUDA compilation.

Issue 1982

### Split NBNXN CUDA kernels into four compilation units

The CUDA nonbonded kernels are now built in four different compilation units when this is possible; ie. devices with compute capability >= 3.0. This can dramatically reduce compilation time.

Forcing the use of a single compilation unit can be done using the GMX_CUDA_NB_SINGLE_COMPILATION_UNIT cmake option.

Issue 1444

### Added stream flushes when not writing newline character

Some of our routines use the carriage return without a newline to keep writing the status e.g. on stderr. For some operating systems this seems to lead to the output being cached in the buffers, so this change adds an explicit fflush() for these print stamements.

Fixed Issue 1772

### Supported cmap with QMMM

Formerly, QMMM only supported bonded interactions using up to 4 atoms. Now any number is supported and some hard-coded assumptions have been removed.

### Upgraded support for lmfit library

Now based on lmfit 6.1. The CMake option GMX_EXTERNAL_LMFIT permits linking an external lmfit package, rather than the one bundled in GROMACS.

Issue 1957

### libxml2 is no longer a dependency

GROMACS used to use libxml2 for running its test code. This has been replaced by a bundled version of tinyxml2 (or optionally, a system version of that library).

### Disable automated FFTW3 builds on Windows

The FFTW distribution does not include configurations to build it automatically on windows, in particular not through the `./configure; make; make install` triad.

Issue 1961

### Remove warnings on checkpoint mismatch

mdrun now only warns for mismatch in minor version, build or number of ranks used when reproducibility is requested. Also added a separate message for not matching precision.

Issue 1992

### Report the filename and the line number on failure

Extend the call to gmx_fatal in fget_lines() to report the filename and the line number where the read failed.

### Handled constraint errors with EM

All energy minimizers could fail with random errors when constraining produced NaN coordinates. Steepest descents now rejects steps with a constraint error. All other minimizer produce a fatal error with the suggestion to use steepest descents first.

Issue 1955

### Disable static libcudart on OS X

Recent versions of CMake enable a static version of libcudart by default, but this breaks builds at least on the most recent version (10.11) of OS X, so we disable it on this platform.

### Fixed rare issue linking with clock_gettime

Misuse of preprocessing commands might have led to inappropriate use of clock_gettime().

Issue 1980

### Disabled NVIDIA JIT cache with OpenCL

The NVIDIA JIT caching is known to be broken with OpenCL compilation in the case when the kernel source changes but the path does not change (e.g. kernels get overwritten by a new installation). Therefore we disable the JIT caching when running on NVIDIA GPUs. AMD GPUs are unaffected.

Issue 1938

### Release notes for older GROMACS versions

Unfortunately, resources are finite and many versions of GROMACS are no longer actively maintained. This page records the release notes for all such versions, so that users can find a record of the changes made in all major and patch releases of GROMACS. Major releases contain changes to the functionality supported, whereas patch releases contain only fixes for issues identified in the corresponding major releases.

Where issue numbers are reported in these release notes, more details can be found at https://gitlab.com/gromacs/gromacs/-/issues at that issue number.

### GROMACS 5.1 series

TODO coming soon

# PYTHON MODULE INDEX

## g

gmxapi, 718
gmxapi._gmxapi, 730
gmxapi._logging, 726
gmxapi.exceptions, 727
gmxapi.runtime, 725
gmxapi.simulation, 722
gmxapi.utility, 724
gmxapi.version, 728

## s

scripted_gmx_docker_builds, 811

## u

utility, 813