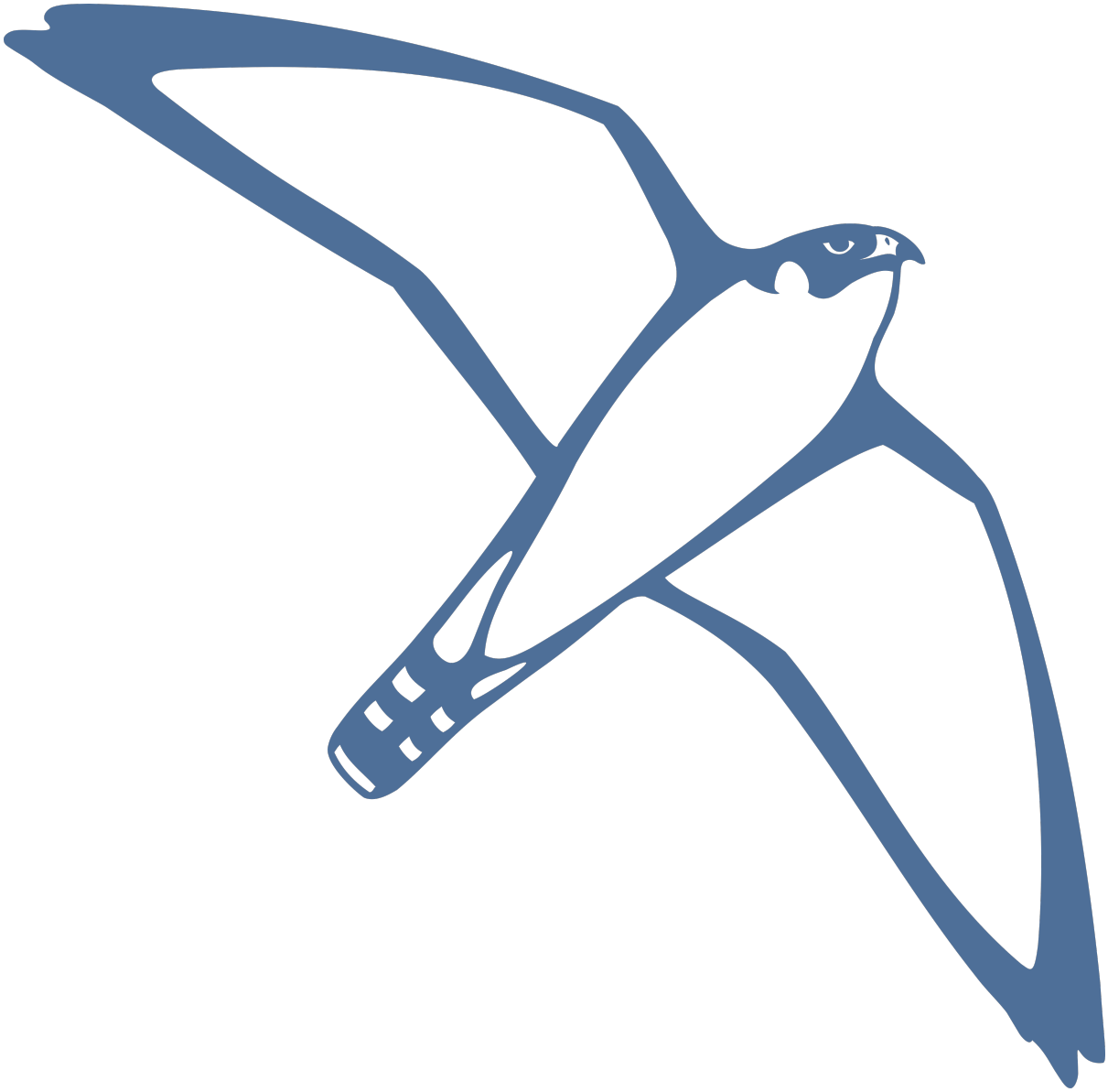


GROMACS Documentation

Release 2022-dev-20211004-152aefc

GROMACS development team

Oct 05, 2021

CONTENTS

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

2 Installation guide **3**
2.1 Introduction to building GROMACS . . . 3
2.1.1 Quick and dirty installation . . . 3
2.1.2 Quick and dirty cluster installation . . . 4
2.1.3 Typical installation . . . 4
2.1.4 Building older versions . . . 4
2.2 Prerequisites . . . 4
2.2.1 Platform . . . 4
2.2.2 Compiler . . . 5
2.2.3 Compiling with parallelization options . . . 5
2.2.4 CMake . . . 7
2.2.5 Fast Fourier Transform library . . . 7
2.2.6 Other optional build components . . . 8
2.3 Doing a build of GROMACS . . . 9
2.3.1 Configuring with CMake . . . 9
2.3.2 Compiling and linking . . . 17
2.3.3 Installing GROMACS . . . 17
2.3.4 Getting access to GROMACS after installation . . . 17
2.3.5 Testing GROMACS for correctness . . . 18
2.3.6 Testing GROMACS for performance . . . 19
2.3.7 Validating GROMACS for source code modifications . . . 19
2.3.8 Having difficulty? . . . 19
2.4 Special instructions for some platforms . . . 20
2.4.1 Building on Windows . . . 20
2.4.2 Building on Cray . . . 20
2.4.3 Building on Solaris . . . 20
2.4.4 Intel Xeon Phi . . . 20
2.5 Tested platforms . . . 21

3 User guide **22**
3.1 Known issues affecting users of GROMACS . . . 22
3.1.1 Unable to compile with CUDA 11.3 . . . 22
3.2 Getting started . . . 22
3.2.1 Flow Chart . . . 22
3.2.2 Setting up your environment . . . 24
3.2.3 Flowchart of typical simulation . . . 24
3.2.4 Important files . . . 24
3.2.5 Tutorial material . . . 26
3.2.6 Background reading . . . 26
3.3 System preparation . . . 26

3.3.1 Steps to consider . . . 26
3.3.2 Tips and tricks . . . 27
3.4 Managing long simulations . . . 28
3.4.1 Appending to output files . . . 28
3.4.2 Backing up your files . . . 29
3.4.3 Extending a .tpr file . . . 29
3.4.4 Changing mdp options for a restart . . . 29
3.4.5 Restarts without checkpoint files . . . 29
3.4.6 Are continuations exact? . . . 29
3.4.7 Reproducibility . . . 30
3.5 Answers to frequently asked questions (FAQs) . . . 31
3.5.1 Questions regarding GROMACS installation . . . 31
3.5.2 Questions concerning system preparation and preprocessing . . . 31
3.5.3 Questions regarding simulation methodology . . . 32
3.5.4 Parameterization and Force Fields . . . 33
3.5.5 Analysis and Visualization . . . 33
3.6 Force fields in GROMACS . . . 33
3.6.1 AMBER . . . 33
3.6.2 CHARMM . . . 34
3.6.3 GROMOS . . . 35
3.6.4 OPLS . . . 35
3.7 Molecular dynamics parameters (.mdp options) . . . 35
3.7.1 General information . . . 35
3.8 Useful mdrun features . . . 74
3.8.1 Re-running a simulation . . . 74
3.8.2 Running a simulation in reproducible mode . . . 75
3.8.3 Halting running simulations . . . 75
3.8.4 Running multi-simulations . . . 75
3.8.5 Controlling the length of the simulation . . . 76
3.9 Getting good performance from mdrun . . . 76
3.9.1 Hardware background information . . . 77
3.9.2 Work distribution by parallelization in GROMACS . . . 78
3.9.3 Parallelization schemes . . . 78
3.9.4 Running mdrun within a single node . . . 82
3.9.5 Running mdrun on more than one node . . . 85
3.9.6 Approaching the scaling limit . . . 87
3.9.7 Finding out how to run mdrun better . . . 87
3.9.8 Running mdrun with GPUs . . . 89
3.9.9 Running the OpenCL version of mdrun . . . 93
3.9.10 Performance checklist . . . 94
3.10 Common errors when using GROMACS . . . 95
3.10.1 Common errors during usage . . . 95
3.10.2 Errors in pdb2gmx . . . 96
3.10.3 Errors in grompp . . . 98
3.10.4 Errors in mdrun . . . 102
3.11 Command-line reference . . . 104
3.11.1 molecular dynamics simulation suite . . . 104
3.11.2 gmx anaeig . . . 110
3.11.3 gmx analyze . . . 112
3.11.4 gmx angle . . . 115
3.11.5 gmx awh . . . 116
3.11.6 gmx bar . . . 117
3.11.7 gmx bundle . . . 119
3.11.8 gmx check . . . 120
3.11.9 gmx chi . . . 122
3.11.10 gmx cluster . . . 124
3.11.11 gmx clustsize . . . 127
3.11.12 gmx confrms . . . 129

3.11.13 gmx convert-tpr . . . 130
3.11.14 gmx convert-trj . . . 130
3.11.15 gmx covar . . . 132
3.11.16 gmx current . . . 133
3.11.17 gmx density . . . 135
3.11.18 gmx densmap . . . 136
3.11.19 gmx densorder . . . 138
3.11.20 gmx dielectric . . . 139
3.11.21 gmx dipoles . . . 140
3.11.22 gmx disre . . . 143
3.11.23 gmx distance . . . 144
3.11.24 gmx do_dssp . . . 145
3.11.25 gmx dos . . . 147
3.11.26 gmx dump . . . 148
3.11.27 gmx dyecoupl . . . 149
3.11.28 gmx editconf . . . 150
3.11.29 gmx eneconv . . . 152
3.11.30 gmx enemat . . . 153
3.11.31 gmx energy . . . 155
3.11.32 gmx extract-cluster . . . 157
3.11.33 gmx filter . . . 159
3.11.34 gmx freevolume . . . 160
3.11.35 gmx gangle . . . 161
3.11.36 gmx genconf . . . 163
3.11.37 gmx genion . . . 164
3.11.38 gmx genrestr . . . 165
3.11.39 gmx grompp . . . 166
3.11.40 gmx gyrate . . . 168
3.11.41 gmx h2order . . . 169
3.11.42 gmx hbond . . . 170
3.11.43 gmx helix . . . 173
3.11.44 gmx helixorient . . . 174
3.11.45 gmx help . . . 176
3.11.46 gmx hydorder . . . 176
3.11.47 gmx insert-molecules . . . 177
3.11.48 gmx lie . . . 178
3.11.49 gmx make_edi . . . 179
3.11.50 gmx make_ndx . . . 182
3.11.51 gmx mdmat . . . 183
3.11.52 gmx mdrun . . . 183
3.11.53 gmx mindist . . . 188
3.11.54 gmx mk_angndx . . . 189
3.11.55 gmx msd . . . 190
3.11.56 gmx nmeig . . . 191
3.11.57 gmx nmens . . . 193
3.11.58 gmx nmr . . . 193
3.11.59 gmx nmtraj . . . 195
3.11.60 gmx nonbonded-benchmark . . . 196
3.11.61 gmx order . . . 197
3.11.62 gmx pairdist . . . 199
3.11.63 gmx pdb2gmx . . . 200
3.11.64 gmx pme_error . . . 203
3.11.65 gmx polystat . . . 204
3.11.66 gmx potential . . . 205
3.11.67 gmx principal . . . 206
3.11.68 gmx rama . . . 207
3.11.69 gmx rdf . . . 207
3.11.70 gmx report-methods . . . 209

3.11.71 gmx rms . . . 210
3.11.72 gmx rmsdist . . . 211
3.11.73 gmx rmsf . . . 213
3.11.74 gmx rotacf . . . 214
3.11.75 gmx rotmat . . . 215
3.11.76 gmx saltbr . . . 216
3.11.77 gmx sans . . . 217
3.11.78 gmx sasa . . . 218
3.11.79 gmx saxs . . . 220
3.11.80 gmx select . . . 221
3.11.81 gmx sham . . . 223
3.11.82 gmx sigeps . . . 225
3.11.83 gmx solvate . . . 225
3.11.84 gmx sorient . . . 227
3.11.85 gmx spatial . . . 228
3.11.86 gmx spol . . . 230
3.11.87 gmx tcaf . . . 231
3.11.88 gmx traj . . . 233
3.11.89 gmx trajectory . . . 234
3.11.90 gmx trjcat . . . 236
3.11.91 gmx trjconv . . . 237
3.11.92 gmx trjorder . . . 240
3.11.93 gmx tune_pme . . . 241
3.11.94 gmx vanhove . . . 245
3.11.95 gmx velacc . . . 246
3.11.96 gmx view . . . 248
3.11.97 gmx wham . . . 248
3.11.98 gmx wheel . . . 252
3.11.99 gmx x2top . . . 253
3.11.100 gmx xpm2ps . . . 254
3.11.101 Command-line interface and conventions . . . 256
3.11.102 Commands by name . . . 257
3.11.103 Commands by topic . . . 259
3.11.104 Special topics . . . 264
3.11.105 Command changes between versions . . . 272
3.12 Terminology . . . 277
3.12.1 Pressure . . . 277
3.12.2 Periodic boundary conditions . . . 277
3.12.3 Thermostats . . . 278
3.12.4 Energy conservation . . . 279
3.12.5 Average structure . . . 280
3.12.6 Blowing up . . . 280
3.12.7 Diagnosing an unstable system . . . 281
3.12.8 Molecular dynamics . . . 282
3.12.9 Force field . . . 283
3.13 Environment Variables . . . 283
3.13.1 Output Control . . . 283
3.13.2 Debugging . . . 284
3.13.3 Performance and Run Control . . . 284
3.13.4 OpenCL management . . . 287
3.13.5 Analysis and Core Functions . . . 288
3.14 Floating point arithmetic . . . 289
3.15 Security when using GROMACS . . . 289
3.16 Policy for deprecating GROMACS functionality . . . 290

4 Short How-To guides **291**
4.1 Beginners . . . 291
4.1.1 Resources . . . 291

4.2 Adding a Residue to a Force Field . . . 291
4.2.1 Adding a new residue . . . 291
4.2.2 Modifying a force field . . . 292
4.3 Water solvation . . . 292
4.4 Non water solvent . . . 292
4.4.1 Making a non-aqueous solvent box . . . 292
4.5 Mixed solvent . . . 293
4.6 Making Disulfide Bonds . . . 293
4.7 Running membrane simulations in GROMACS . . . 293
4.7.1 Running Membrane Simulations . . . 293
4.7.2 Adding waters with genbox . . . 294
4.7.3 External material . . . 294
4.8 Parameterization of novel molecules . . . 294
4.8.1 Exotic Species . . . 295
4.9 Potential of Mean Force . . . 296
4.10 Single-Point Energy . . . 296
4.11 Carbon Nanotube . . . 296
4.11.1 Robert Johnson's Tips . . . 296
4.11.2 Andrea Minoia's tutorial . . . 297
4.12 Visualization Software . . . 298
4.12.1 Topology bonds vs Rendered bonds . . . 298
4.13 Extracting Trajectory Information . . . 298
4.14 External tools to perform trajectory analysis . . . 299
4.15 Plotting Data . . . 299
4.15.1 Software . . . 299
4.16 Micelle Clustering . . . 300

5 Reference Manual **301**
5.1 Preface and Disclaimer . . . 301
5.1.1 Citation information . . . 302
5.1.2 GROMACS is *Free Software* . . . 302
5.2 Introduction . . . 303
5.2.1 Computational Chemistry and Molecular Modeling . . . 303
5.2.2 Molecular Dynamics Simulations . . . 304
5.2.3 Energy Minimization and Search Methods . . . 306
5.3 Definitions and Units . . . 308
5.3.1 Notation . . . 308
5.3.2 MD units . . . 308
5.3.3 Reduced units . . . 309
5.3.4 Mixed or Double precision . . . 310
5.4 Algorithms . . . 311
5.4.1 Periodic boundary conditions . . . 311
5.4.2 The group concept . . . 314
5.4.3 Molecular Dynamics . . . 315
5.4.4 Shell molecular dynamics . . . 338
5.4.5 Constraint algorithms . . . 338
5.4.6 Simulated Annealing . . . 342
5.4.7 Stochastic Dynamics . . . 342
5.4.8 Brownian Dynamics . . . 343
5.4.9 Energy Minimization . . . 343
5.4.10 Normal-Mode Analysis . . . 344
5.4.11 Free energy calculations . . . 345
5.4.12 Replica exchange . . . 347
5.4.13 Essential Dynamics sampling . . . 349
5.4.14 Expanded Ensemble . . . 349
5.4.15 Parallelization . . . 349
5.4.16 Domain decomposition . . . 350
5.5 Interaction function and force fields . . . 356

5.5.1 Non-bonded interactions . . . 356
5.5.2 Bonded interactions . . . 361
5.5.3 Restraints . . . 372
5.5.4 Polarization . . . 382
5.5.5 Free energy interactions . . . 383
5.5.6 Methods . . . 387
5.5.7 Virtual interaction sites . . . 388
5.5.8 Long Range Electrostatics . . . 393
5.5.9 Long Range Van der Waals interactions . . . 395
5.5.10 Force field . . . 399
5.6 Topologies . . . 402
5.6.1 Particle type . . . 403
5.6.2 Parameter files . . . 405
5.6.3 Molecule definition . . . 408
5.6.4 Constraint algorithms . . . 409
5.6.5 pdb2gmx input files . . . 410
5.6.6 File formats . . . 418
5.6.7 Force field organization . . . 432
5.7 File formats . . . 434
5.7.1 Summary of file formats . . . 434
5.7.2 File format details . . . 435
5.8 Special Topics . . . 451
5.8.1 Free energy implementation . . . 451
5.8.2 Potential of mean force . . . 452
5.8.3 Non-equilibrium pulling . . . 453
5.8.4 Collective variables: the pull code . . . 453
5.8.5 Adaptive biasing with AWH . . . 458
5.8.6 Enforced Rotation . . . 467
5.8.7 Electric fields . . . 477
5.8.8 Computational Electrophysiology . . . 478
5.8.9 Calculating a PMF using the free-energy code . . . 481
5.8.10 Removing fastest degrees of freedom . . . 481
5.8.11 Viscosity calculation . . . 484
5.8.12 Tabulated interaction functions . . . 485
5.8.13 Mixed Quantum-Classical simulation techniques . . . 487
5.8.14 MiMiC Hybrid Quantum Mechanical/Molecular Mechanical simulations . . . 488
5.8.15 Using VMD plug-ins for trajectory file I/O . . . 492
5.8.16 Interactive Molecular Dynamics . . . 492
5.8.17 Embedding proteins into the membranes . . . 493
5.8.18 Applying forces from three-dimensional densities . . . 494
5.9 Run parameters and Programs . . . 497
5.9.1 Online documentation . . . 497
5.9.2 File types . . . 497
5.9.3 Run Parameters . . . 498
5.10 Analysis . . . 499
5.10.1 Using Groups . . . 499
5.10.2 Looking at your trajectory . . . 502
5.10.3 General properties . . . 502
5.10.4 Radial distribution functions . . . 503
5.10.5 Correlation functions . . . 504
5.10.6 Curve fitting in GROMACS . . . 506
5.10.7 Mean Square Displacement . . . 507
5.10.8 Bonds/distances, angles and dihedrals . . . 508
5.10.9 Radius of gyration and distances . . . 509
5.10.10 Root mean square deviations in structure . . . 510
5.10.11 Covariance analysis . . . 511
5.10.12 Dihedral principal component analysis . . . 513
5.10.13 Hydrogen bonds . . . 513

5.10.14 Protein-related items . . . 514
5.10.15 Interface-related items . . . 516
5.11 Some implementation details . . . 518
5.11.1 Single Sum Virial in GROMACS . . . 518
5.11.2 Optimizations . . . 521
5.12 Averages and fluctuations . . . 523
5.12.1 Formulae for averaging . . . 523
5.12.2 Implementation . . . 524
5.13 Bibliography . . . 527

6 gmxapi Python package **536**
6.1 Python User Guide . . . 536
6.1.1 Full installation instructions . . . 536
6.1.2 Using the Python package . . . 546
6.1.3 gmxapi Python module reference . . . 551
6.2 Indices and tables . . . 560

7 NBLIB API **561**
7.1 Guide to Writing MD Programs . . . 561
7.1.1 Global Definitions . . . 561
7.1.2 Define Particle Data . . . 562
7.1.3 Defining Coordinates, Velocities and Force Buffers . . . 562
7.1.4 Writing the MD Program . . . 563

8 Developer Guide **568**
8.1 Contribute to GROMACS . . . 568
8.1.1 Checklist . . . 569
8.1.2 Preparing code for submission . . . 570
8.1.3 Alternatives . . . 570
8.1.4 Do you have more questions? . . . 570
8.1.5 Removing functionality . . . 571
8.2 Codebase overview . . . 571
8.2.1 Source code organization . . . 571
8.2.2 Documentation organization . . . 573
8.3 Build system overview . . . 574
8.3.1 Build types . . . 575
8.3.2 CMake cache variables . . . 576
8.3.3 External libraries . . . 579
8.3.4 Special targets . . . 579
8.3.5 Passing information to source code . . . 580
8.4 Change Management . . . 581
8.4.1 Getting started . . . 581
8.4.2 Labels . . . 583
8.4.3 Code Review . . . 583
8.4.4 More git tips . . . 584
8.5 Relocatable binaries . . . 587
8.5.1 Finding shared libraries . . . 587
8.5.2 Finding data files . . . 588
8.5.3 Known issues . . . 589
8.6 Documentation generation . . . 590
8.6.1 Building the GROMACS documentation . . . 590
8.6.2 Needed build tools . . . 590
8.7 Style guidelines . . . 591
8.7.1 Guidelines for code formatting . . . 591
8.7.2 Guidelines for #include directives . . . 593
8.7.3 Naming conventions . . . 594
8.7.4 Allowed language features . . . 597
8.7.5 Guidelines for creating meaningful issue reports . . . 600
8.7.6 Guidelines for formatting of git commits . . . 601

8.7.7 Error handling . . . 602
8.8 Development-time tools . . . 605
8.8.1 Using Doxygen . . . 605
8.8.2 Automation and Infrastructure . . . 618
8.8.3 Release engineering with GitLab . . . 628
8.8.4 Source tree checker scripts . . . 628
8.8.5 Automatic source code formatting . . . 631
8.8.6 Unit testing . . . 636
8.8.7 Physical validation . . . 638
8.8.8 Change management . . . 641
8.8.9 Build system . . . 641
8.8.10 Code formatting and style . . . 642
8.9 Known issues relevant for developers . . . 643
8.9.1 Issues with GPU timer with OpenCL . . . 643
8.9.2 GPU emulation does not work . . . 643
8.9.3 OpenCL on NVIDIA Volta and later broken . . . 643
8.9.4 MdrunIO tests fail on Power9 . . . 643

9 Doxygen documentation **644**

Python Module Index **645**

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-2022-dev.tar.gz
- As https https://ftp.gromacs.org/gromacs/gromacs-2022-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-2022.tar.gz
- (md5sum unknown)

CHAPTER

TWO

INSTALLATION GUIDE

2.1 Introduction to building GROMACS

These instructions pertain to building GROMACS 2022-dev. You might also want to check the up-to-date installation instructions.

2.1.1 Quick and dirty installation

1. Get the latest version of your C and C++ compilers.
2. Check that you have CMake version 3.16.3 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-2022-dev.tar.gz
cd gromacs-2022-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.

2.1.2 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.

2.1.3 Typical installation

As above, and with further details below, but you should consider using the following *CMake options* (page 10) with the appropriate value instead of `xxx` :

- `-DCMAKE_C_COMPILER=xxx` equal to the name of the C99 *Compiler* (page 5) you wish to use (or the environment variable `CC`)
- `-DCMAKE_CXX_COMPILER=xxx` equal to the name of the C++17 *compiler* (page 5) you wish to use (or the environment variable `CXX`)
- `-DGMX_MPI=on` to build using *MPI support* (page 6)
- `-DGMX_GPU=CUDA` to build with NVIDIA CUDA support enabled.
- `-DGMX_GPU=OpenCL` to build with OpenCL support enabled.
- `-DGMX_SIMD=xxx` to specify the level of *SIMD support* (page 10) 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 12)
- `-DCMAKE_INSTALL_PREFIX=xxx` to install GROMACS to a *non-standard location* (page 10) (default `/usr/local/gromacs`)
- `-DBUILD_SHARED_LIBS=off` to turn off the building of shared libraries to help with *static linking* (page 13)
- `-DGMX_FFT_LIBRARY=xxx` to select whether to use `fftw3`, `mkl` or `fftpack` libraries for *FFT support* (page 7)
- `-DCMAKE_BUILD_TYPE=Debug` to build GROMACS in debug mode

2.1.4 Building older versions

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

2.2 Prerequisites

2.2.1 Platform

GROMACS can be compiled for many operating systems and architectures. These include any distribution of Linux, Mac OS X or Windows, and architectures including x86, AMD64/x86-64, several PowerPC including POWER8, ARM v8, and SPARC VIII.

2.2.2 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 supported compiler versions are

- GNU (gcc/libstdc++) 7
- LLVM (clang/libc++) 5
- Microsoft (MSVC) 2017 15.7

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-2022-dev. We recommend to use the gcc compiler instead, as it is being extensively tested.

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 use the libstdc++ which comes with gcc as the default C++ library. For GROMACS, we require the compiler to support libstc++ version 7.1 or higher. To select a particular libstdc++ library, provide the path to g++ with `-DGMX_GPLUSPLUS_PATH=/path/to/g++`.

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

If you are running on Mac OS X, the best option is gcc. The Apple clang compiler provided by MacPorts will work, but does not support OpenMP, so will probably not provide best performance.

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

2.2.3 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 has excellent support for NVIDIA GPUs supported via CUDA. On Linux, NVIDIA CUDA toolkit with minimum version unknown is required, and the latest version is strongly encouraged. NVIDIA GPUs with at least NVIDIA compute capability unknown are required. You are strongly recommended to get the latest CUDA version and driver that supports your hardware, but beware of possible performance regressions in newer CUDA versions on older hardware. While some CUDA compilers (nvcc) might not officially support recent versions of gcc as the back-end compiler, we still recommend that you at least use a gcc version recent enough to get the best SIMD support for your CPU, since GROMACS always runs some code on the CPU. It is most reliable to use the same C++ compiler version for GROMACS code as used as the host compiler for nvcc.

To make it possible to use other accelerators, GROMACS also includes OpenCL support. The minimum OpenCL version required is unknown and only 64-bit implementations are supported. The current OpenCL implementation is recommended for use with GCN-based AMD GPUs, and on Linux we recommend the ROCm runtime. Intel integrated GPUs are supported with the Neo drivers. OpenCL is also supported with NVIDIA GPUs, but using the latest NVIDIA driver (which includes the NVIDIA OpenCL runtime) is recommended. Also note that there are performance limitations (inherent to the NVIDIA OpenCL runtime). It is not possible to configure both CUDA and OpenCL support in the same build of GROMACS, nor to support both Intel and other vendors' GPUs with OpenCL. A 64-bit implementation of OpenCL is required and therefore OpenCL is only supported on 64-bit platforms.

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 2.0 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.

CUDA-Aware MPI support

In simulations using multiple NVIDIA GPUs, an MPI implementation with CUDA support (also called "CUDA-aware") 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. 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.

With `GPU_MPI=ON`, GROMACS attempts to automatically detect CUDA 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 CUDA-aware support, in which case it can be manually enabled by setting environment variable `GMX_FORCE_CUDA_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).

2.2.4 CMake

GROMACS builds with the CMake build system, requiring at least version 3.16.3. 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.

2.2.5 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 14)). 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 *both* `--enable-sse2` and `--enable-avx` for FFTW-3.3.4 and earlier. From FFTW-3.3.5, you should also add `--enable-avx2` also. On Intel processors supporting 512-wide AVX, including KNL, add `--enable-avx512` also. 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 and IBM Power8 and later, you definitely want version 3.3.5 or later, and to compile it with `--enable-neon` and `--enable-vsx`, respectively, for SIMD support. If you are using a Cray, there is a special modified (commercial) version of FFTs using the FFTW interface which can be slightly faster.

Using MKL

Use MKL bundled with Intel compilers by setting up the compiler environment, e.g., through `source /path/to/compilervars.sh intel64` or similar before running CMake including setting `-DGMX_FFT_LIBRARY=mkl`.

If you need to customize this further, use

```
cmake -DGMX_FFT_LIBRARY=mkl \
      -DMKL_LIBRARIES="/full/path/to/libone.so;/full/path/to/
↪libtwo.so" \
      -DMKL_INCLUDE_DIR="/full/path/to/mkl/include"
```

The full list and order(!) of libraries you require are found in Intel's MKL documentation for your system.

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
```

2.2.6 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 14).
- The built-in GROMACS trajectory viewer `gmx view` requires X11 and Motif/Lesstif libraries and header files. You may prefer to use third-party software for visualization, such as VMD or PyMol.
- 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 ImageMagick, pdflatex, bibtex, doxygen, python 3.6, sphinx 1.6.1, and pygments.
- 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.

2.3 Doing a build of GROMACS

This section will cover a general build of GROMACS with *CMake* (page 7), 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.

2.3.1 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 xfz gromacs-2022-dev.tgz
cd gromacs-2022-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 gmx-users mailing list. 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.

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.
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 it will also enable Intel 3-way fused multiply-add instructions.
8. `AVX_512` Skylake-X desktop and Skylake-SP Xeon processors (2017); 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). Additionally, with GPU accelerated runs `AVX2_256` can also be faster on high-end Skylake CPUs with both 512-bit FMA units enabled.
9. `AVX_512_KNL` Knights Landing Xeon Phi processors
10. `IBM_VSX` Power7, Power8, Power9 and later have this.
11. `ARM_NEON_ASIMD` 64-bit ARMv8 and later.
12. `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.

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 mailing lists, because GROMACS can probably be ported for 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 enviroment 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 -DCUDA_TOOLKIT_ROOT_DIR=/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 `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 either with the `GMX_CUDA_TARGET_SM` or `GMX_CUDA_TARGET_COMPUTE` CMake variables, which take a semicolon delimited string with the two digit suffixes of CUDA (virtual) architectures names, for instance "35;50;51;52;53;60". For details, see the "Options for steering GPU code generation" section of the nvcc man / help or Chapter 6. of the nvcc manual.

The GPU acceleration has been tested on AMD64/x86-64 platforms with Linux, Mac OS X and Windows operating systems, but Linux is the best-tested and supported of these. Linux running on POWER 8 and ARM v8 CPUs also works well.

Experimental support is available for compiling CUDA code, both for host and device, using clang (version 6.0 or later). 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. Target architectures can be selected with `GMX_CUDA_TARGET_SM`, virtual architecture code is always embedded for all requested architectures (hence GMX_CUDA_TARGET_COMPUTE is ignored). Note that this is mainly a developer-oriented feature and it is not recommended for production use as the performance can be significantly lower than that of code compiled with nvcc (and it has also received less testing). However, note that since clang 5.0 the performance gap is only moderate (at the time of writing, about 20% slower GPU kernels), so this version could be considered in non performance-critical use-cases.

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 (e.g. from AMD), which must be installed in a path found in `CMAKE_PREFIX_PATH` (or via the environment variables `AMDAPPSDKROOT` or `CUDA_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, 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
```

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 necessary, most commonly when you are running a 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`.

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 aspects

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.

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 -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 oficial 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>"` 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.

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.

2.3.2 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 gmx-users, 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.

2.3.3 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.

2.3.4 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 toolchains file to help with building client software. For an installation at `/your/installation/prefix/here`, toolchain files will be installed at `/your/installation/prefix/here/share/cmake/gromacs${GMX_LIBS_SUFFIX}/gromacs-toolchain${GMX_LIBS_SUFFIX}.cmake` where `${GMX_LIBS_SUFFIX}` is *as documented above* (page 16).

2.3.5 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-2022.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 too, 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| users 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=
```

2.3.6 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`.

2.3.7 Validating GROMACS for source code modifications

When building GROMACS from a release tarball, the build process automatically checks if any file contributing to the build process have been modified since they have been packed in the archive. This results in the marking of the version as either `MODIFIED` (if the source files have been modified) or `UNCHECKED` (if no validation was possible, e.g. if no Python installation was found). The actual checking is performed by comparing a checksum stored in the release tarball against one generated by the `createFileHash.py` Python script during the build configuration. When running a GROMACS binary, the checksum is also printed in the log file, together with a message if there is a mismatch or no validation has been possible.

This allows users to check whether the binary they are using was built from source code that is identical to the source code released by the GROMACS team. Thus unintentional modifications to the source code for building binaries that are used for running production simulations are easily detectable. Additionally, by manually setting a version tag using the GMX_VERSION_STRING_OF_FORK cmake option, users can mark a modified GROMACS release code with their custom version string suffix.

2.3.8 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 users emailing list for information on the error. Adding `site:https://mailman-1.sys.kth.se/pipermail/gromacs.org_gmx-users` to a Google search may help filter better results.
3. Search the internet using a search engine such as Google.
4. Post to the GROMACS users emailing list gmx-users for assistance. 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.

2.4 Special instructions for some platforms

2.4.1 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 7)) 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 8).

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.

2.4.2 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`).

2.4.3 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. At least version 1.11.8 of hwloc is recommended.

Oracle Developer Studio is not a currently supported compiler (and does not currently compile GROMACS correctly, perhaps because the thread-MPI atomics are incorrectly implemented in GROMACS).

2.4.4 Intel Xeon Phi

Xeon Phi processors, hosted or self-hosted, are supported. The Knights Landing-based Xeon Phi processors behave like standard x86 nodes, but support a special SIMD instruction set. When cross-compiling for such nodes, use the `AVX_512_KNL` SIMD flavor. Knights Landing processors support so-called "clustering modes" which allow reconfiguring the memory subsystem for lower latency. GROMACS can benefit from the quadrant or SNC clustering modes. Care needs to be taken to correctly pin threads. In particular, threads of an MPI rank should not cross cluster and NUMA boundaries. In addition to the main DRAM memory, Knights Landing has a high-bandwidth stacked memory called MCDRAM. Using it offers performance benefits if it is ensured that `mdrun` runs entirely from this memory; to do so it is recommended that MCDRAM is configured in "Flat mode"

and `mdrun` is bound to the appropriate NUMA node (use e.g. `numactl --membind 1` with quadrant clustering mode).

2.5 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 7 and 8, clang versions 8 and 9, and a version of oneAPI containing Intel's clang-based compiler. For this testing, we use Ubuntu 18.04 or 20.04 operating system. Other compiler, library, and OS versions are tested less frequently. For details, you can have a look at the continuous integration server used by GROMACS, which uses GitLab runner on a local k8s x86 cluster with NVIDIA and AMD GPU support.

We test irregularly on ARM v8, Cray, Power8, Power9, Google Native Client and other environments, and with other compilers and compiler versions, too.

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 301) of the documentation. If you have questions not answered by these resources, please visit the GROMACS users 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 Unable to compile with CUDA 11.3

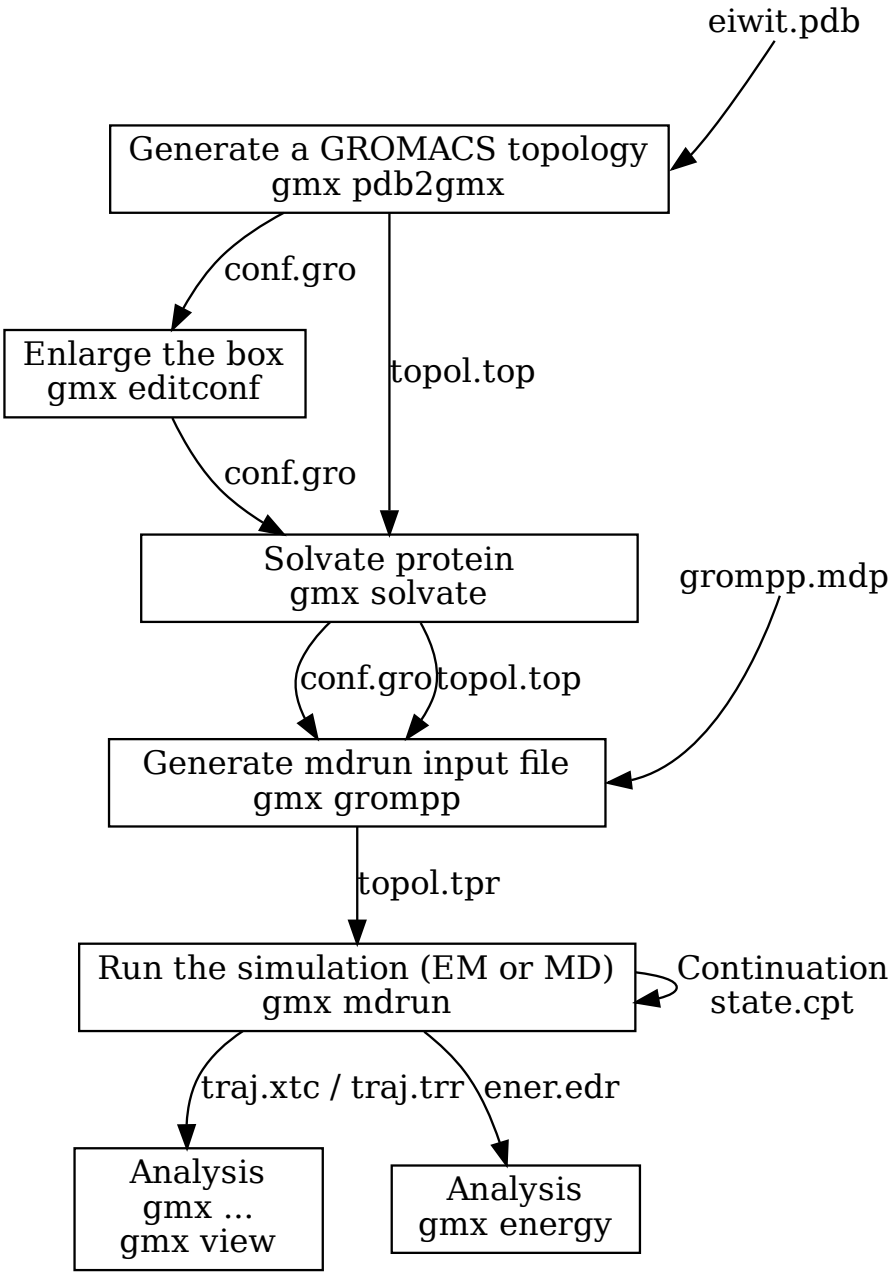
Due to a bug in the nvcc compiler, it is currently not possible to compile NVIDIA GPU-enabled GROMACS with version 11.3 of the CUDA compiler. We recommend using CUDA 11.2 or earlier until the compiler fix is released.

Issue 4037

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. A more detailed example is available in *Getting started* (page 22). Several steps of energy minimization may be necessary, these consist of cycles: *gmx grompp* (page 166) -> *gmx mdrun* (page 183).



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 17).

3.2.3 Flowchart of typical simulation

A typical simulation workflow with GROMACS is *illustrated here* (page 22).

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 200). *gmx pdb2gmx* (page 200) translates a *pdb* (page 442) 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 445) 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 35) in the *mdp* (page 440) 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 shouldn't 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 445) 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 440) file to choose whether or not you want position restraints on your solvent, rather than your *top* (page 445) file. Note that preprocessor variables are not the same as shell environment variables.

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

When *gmx pdb2gmx* (page 200) is executed to generate a molecular topology, it also translates the structure file (*pdb* (page 442) file) to a GROMOS structure file (*gro* (page 437) file). The main difference between a *pdb* (page 442) file and a gromos file is their format and that a *gro* (page 437) file can also hold velocities. However, if you do not need the velocities, you can also use a *pdb* (page 442) file in all programs. To generate a box of solvent molecules around the peptide, the program *gmx solvate* (page 225) is used. First the program *gmx editconf* (page 150) should be used to define a box of appropriate size around the molecule. *gmx solvate* (page 225) solvates a solute molecule (the peptide) into any solvent (in this case, water). The output of *gmx solvate* (page 225) is a gromos structure file of the peptide solvated in water. *gmx solvate* (page 225) also changes the molecular topology file (generated by *gmx pdb2gmx* (page 200)) to add solvent to the topology.

Molecular Dynamics parameter file (`.mdp`)

The Molecular Dynamics Parameter (*mdp* (page 440)) 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* (page 440) file. A *sample mdp file* (page 440) is available.

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 437) file), topology (*top* (page 445) file) MD-parameters (*mdp* (page 440) file) and (optionally) the index file (*ndx* (page 441)) to generate a run input file (*tpr* (page 446) extension). This file contains all information needed to start a simulation with GROMACS. The *gmx grompp* (page 166) program processes all input files and generates the run input *tpr* (page 446) 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 183). The only input file of *gmx mdrun* (page 183) that you usually need in order to start a run is the run input file (*tpr* (page 446) file). The typical output files of *gmx mdrun* (page 183) are the trajectory file (*trr* (page 447) file), a logfile (*log* (page 439) file), and perhaps a checkpoint file (*cpt* (page 435) file).

3.2.5 Tutorial material

There are several third-party tutorials available that cover aspects of using GROMACS. Further information can also be found in the *How to* (page 291) 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! Consider now 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 225), *gmx insert-molecules* (page 177) and *gmx genconf* (page 163) solve frequent problems.
5. Obtain or generate the topology file for the system, using (for example) *gmx pdb2gmx* (page 200), *gmx x2top* (page 253), SwissParam (for CHARMM forcefield), PRODRG (for GROMOS96 43A1), Automated Topology Builder (for GROMOS96 53A6), MKTOP (for

OPLS/AA) or your favourite text editor in concert with chapter 5 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 150)) whose size is appropriate for the eventual density you would like, fill it with solvent (e.g. using *gmx solvate* (page 225)), and add any counter-ions needed to neutralize the system (e.g. using *gmx grompp* (page 166) and *gmx insert-molecules* (page 177)). 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 166) and *gmx mdrun* (page 183)). 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 440) 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 (which should be done with Berendsen until after the density is stabilized, before a further switch to a barostat that produces the correct ensemble), 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 280), 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 166) and *gmx mdrun* (page 183), then *gmx energy* (page 155) and trajectory visualization tools).
10. Select the appropriate simulation parameters for the production simulation (defined in *mdp* (page 440) 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 104)) 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 183) 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 437) file with velocities, but since the *gro* (page 437) 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 183) during the run. The interval is given by the `-cpt` flag to *gmx mdrun* (page 183). When *gmx mdrun* (page 183) attemps 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 183) can be halted in several ways:

- the number of simulation `nsteps` (page 37) 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 183) 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 120) and *gmx dump* (page 148).

3.4.1 Appending to output files

By default, *gmx mdrun* (page 183) 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 37) option can be used to set this number manually in *gmx grompp* (page 166), 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 183) 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 446) file has completed and should be extended, use the *gmx convert-tpr* (page 130) 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 440) file may have generated velocities, but that is a one-time operation within *gmx grompp* (page 166) 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 440) file or topology, and then call

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

to instruct *gmx grompp* (page 166) to copy the full-precision coordinates in the checkpoint file into the new *tpr* (page 446) file. You should consider your choices for `tinit` (page 37), `init-step` (page 37), `nsteps` (page 37) and `simulation-part` (page 37). You should generally not regenerate velocities with `gen-vel` (page 50), and generally select `continuation` (page 51) 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 shouldn't 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 wall-clock 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-reprodudible 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 203)), only *mdrun* (page 183) is able to use the *MPI* (page 6) parallelism. So you only need to use the `-DGMX_MPI=on` flag when *configuring* (page 9) for a build intended to run the main simulation engine *mdrun* (page 183). 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 20).

3.5.2 Questions concerning system preparation and preprocessing

1. Where can I find a solvent *coordinate file* (page 434) for use with *solvate* (page 225)?

   Suitable equilibrated boxes of solvent *structure files* (page 434) can be found in the `$GMXDIR/share/gromacs/top` directory. That location will be searched by default by *solvate* (page 225), for example by using `-cs spc216.gro` as an argument. Other solvent boxes can be prepared by the user as described on the manual page for *solvate* (page 225) and elsewhere. Note that suitable topology files will be needed for the solvent boxes to be useful in *grompp* (page 166). These are available for some force fields, and may be found in the respective subfolder of `$GMXDIR/share/gromacs/top`.

2. How to prevent *solvate* (page 225) 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 280). You can either remove those waters by hand (and do the accounting for molecule types in the *topology* (page 445) 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 439) file or the *topology* (page 445) 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 437) file?

   The *gro* (page 437) file is used in GROMACS as a unified *structure file* (page 434) format that can be read by all utilities. The large majority of GROMACS routines can also use other file types such as *pdb* (page 442), with the limitations that no velocities are available in *this case*

(page 25). If you need a text-based format with more digits of precision, the *g96* (page 437) format is suitable and supported.

3. Do I always need to run *pdb2gmx* (page 200) when I already produced an *itp* (page 439) file elsewhere?

   You don't need to prepare additional files if you already have all *itp* (page 439) and *top* (page 445) files prepared through other tools.

   Examples for those are CHARMM-GUI, ATB (Automated Topology Builder), pmx. and PRODRG.

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 97) 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 289) 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 278) and more specifically in *What not to do* (page 279), 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 37) and `init-step` (page 37).

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 165).

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

   Please see the section on *Managing long simulations* (page 28). You can either prepare a new *mdp* (page 440) file, or extend the simulation time in the original *tpr* (page 446) file using *convert-tpr* (page 130).

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

   This is best achieved with the `-rerun` option to *mdrun* (page 183). See the *Re-running a simulation* (page 74) 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 282) 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 283) 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 283) 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 445) file. What matters is the information encoded in there. If the software has read a *tpr* (page 446) file, then the information is in reliable agreement with the topology you supplied to *grompp* (page 166).

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 277), and are not a reason for concern. You can fix the visualization using *trjconv* (page 237) 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 289), 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 283) for the simulation of biomolecules and a package of molecular simulation programs.

GROMACS versions higher than 4.5 support the following AMBER force fields natively:

- AMBER94
- AMBER96
- AMBER99
- AMBER99SB

- AMBER99SB-ILDN
- AMBER03
- AMBERGS

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 acpypi.py), but they do require an AMBER installation to work.

Older GROMACS versions need a separate installation of the ffamber ports:

- Using AMBER Force Field in GROMACS - known as the "ffamber ports," a number of AMBER force fields, complete with documentation.
- Using the ffamber ports with GROMACS requires that the input structure files adhere to the AMBER nomenclature for residues. Problematic residues involve termini (prefixed with N and C), lysine (either LYN or LYP), histidine (HID, HIE, or HIS), and cysteine (CYN or CYX). Please see the ffamber documentation.

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 282) simulations and analysis. Includes united atom (CHARMM19) and all atom (CHARMM22, CHARMM27, CHARMM36) *force fields* (page 283). The CHARMM27 force field has been ported to GROMACS and is officially supported as of version 4.5. 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 5.0 and newer, please use the following settings in the *mdp* (page 440) file:

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

Anyone using very old versions of GROMACS may find this script useful:

CHARMM to GROMACS - perl scripts intended to facilitate calculations using GROMACS programs and CHARMM forcefields (needed for GROMACS versions < 4.5). (link)

3.6.3 GROMOS

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 283), 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 440) 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 445) 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 36) 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 36) leap-frog integrator. The kinetic energy, which 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 off extra computation, especially with constraints and extra communication in parallel. Note that for nearly all production simulations the *integrator=md* (page 36) integrator is accurate enough.

md-vv-avek

A velocity Verlet algorithm identical to *integrator=md-vv* (page 36), except that the kinetic energy is determined as the average of the two half step kinetic energies as in the *integrator=md* (page 36) 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 47)) is set with *ref-t* (page 47), the inverse friction constant for each group is set with *tau-t* (page 47). The parameters *tcoupl* (page 46) and *nsttcouple* (page 47) are ignored. The random generator is initialized with *ld-seed* (page 39). When used as a thermostat, an appropriate value for *tau-t* (page 47) 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 47).

bd

An Euler integrator for Brownian or position Langevin dynamics, the velocity is the force divided by a friction coefficient (*bd-fric* (page 39)) plus random thermal noise (*ref-t* (page 47)). When *bd-fric* (page 39) is 0, the friction coefficient for each particle is calculated as mass/ *tau-t* (page 47), as for the integrator *integrator=sd* (page 36). The random generator is initialized with *ld-seed* (page 39).

steep

A steepest descent algorithm for energy minimization. The maximum step size is *emstep* (page 39), the tolerance is *emtol* (page 39).

cg

A conjugate gradient algorithm for energy minimization, the tolerance is *emtol* (page 39). CG is more efficient when a steepest descent step is done every once in a while, this is determined by *nstcgsteep* (page 39). 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 446) 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 37) times in each frame at random locations and with random orientiations of the molecule. When `nstlist` (page 40) is larger than one, `nstlist` (page 40) insertions are performed in a sphere with radius `rtpi` (page 39) 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 39). The temperature for the Boltzmann weighting is set with `ref-t` (page 47), 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 37), 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 39) sets the radius for the sphere around this location. Neighbor searching is done only once per frame, `nstlist` (page 40) is not used. Parallel `integrator=tpic` (page 37) gives identical results to single-rank `integrator=tpic` (page 37).

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 70)

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 37) + `dt` (page 37) * (`init-step` (page 37) + i). The free-energy lambda is calculated as: lambda = `init-lambda` (page 63) + `delta-lambda` (page 63) * (`init-step` (page 37) + 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 37) to the step number of the restart frame. *gmx convert-tpr* (page 130) 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 38) every *mts-level2-factor* (page 38) integration steps. All other forces are evaluated at every step. MTS is currently only supported with *integrator=md* (page 36).

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 38) 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

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 38) steps. This is useful for cases where an acceleration is expected on the center of mass which is nearly constant over *nstcomm* (page 38) 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] frequency 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 39) is 0, the friction coefficient for each particle is calculated as mass/ *tau-t* (page 47).

ld-seed
(-1) [integer] used to initialize random generator for thermal noise for stochastic and Brownian dynamics. When *ld-seed* (page 39) 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 39) 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] frequency 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 39), or a maximum number of iterations *niter* (page 39) has been reached. Minimization is converged when the maximum force is smaller than *emtol* (page 39). 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 37) and *integrator=tpic* (page 37)

Output control

nstxout
(0) [steps] number of steps that elapse between writing coordinates to the output trajectory file (*trr* (page 447)), the 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 447)), the 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 447)), the 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 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 energy file, the last energies are always written, should be a multiple of `nstcalcenergy` (page 40). Note that the exact sums and fluctuations over all MD steps modulo `nstcalcenergy` (page 40) are stored in the energy file, so *gmx energy* (page 155) can report exact energy averages and fluctuations also when `nstenergy` (page 40) > 1

nstxout-compressed
(0) [steps] number of steps that elapse between writing position coordinates using lossy compression (*xtc* (page 448) file), 0 for not writing 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 40) > 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 41), unless this is set to -1, in which case `rlist` (page 42) 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
Frequency to update the neighbor list. When dynamics and *verlet-buffer-tolerance* (page 41) set, *nstlist* (page 40) is actually a minimum value and *gmx mdrun* (page 183) 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 40) to 0. For best performance without cut-offs on a single MPI rank, set *nstlist* (page 40) to zero and `ns-type=simple`.

xy
Use periodic boundary conditions in x and y directions only. This works only with `ns-type=grid` and can be used in combination with *walls* (page 52). 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 42). As both *nstlist* (page 40) 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 40) -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 42). 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 40) -1) * *dt* (page 37) 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 41) =-1 and set `rlist` (page 42) manually.

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 41) option and the value of `rlist` (page 42) 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 166) with an NVT setup with the expected temperature and copy the resulting value of `rlist` (page 42) to the NVE setup.

Electrostatics

coulombtype

Cut-off

Plain cut-off with pair list radius `rlist` (page 42) and Coulomb cut-off `rcoulomb` (page 43), where `rlist` (page 42) >= `rcoulomb` (page 43).

Ewald

Classical Ewald sum electrostatics. The real-space cut-off `rcoulomb` (page 43) should be equal to `rlist` (page 42). Use *e.g.* `rlist` (page 42) =0.9, `rcoulomb` (page 43) =0.9. The highest magnitude of wave vectors used in reciprocal space is controlled by `fourierspacing` (page 45). The relative accuracy of direct/reciprocal space is controlled by `ewald-rtol` (page 45).

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 45) and the interpolation order with `pme-order` (page 45). 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 43), where `rlist` (page 42) >= `rvdw` (page 44). The dielectric constant beyond the cut-off is `epsilon-rf` (page 43). The dielectric constant can be set to infinity by setting `epsilon-rf` (page 43) =0.

User

Currently unsupported. *gmx mdrun* (page 183) 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 183) 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 43) 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 45) 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 43) is allowed to be smaller than *rlist* (page 42).

PME-User
Currently unsupported. A combination of PME and user tables (see above). *rcoulomb* (page 43) is allowed to be smaller than *rlist* (page 42). The PME mesh contribution is subtracted from the user table by *gmx mdrun* (page 183). 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 183) 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 42) and VdW cut-off *rvdw* (page 44), where *rlist* (page 42) >= *rvdw* (page 44).

PME
Fast smooth Particle-mesh Ewald (SPME) for VdW interactions. The grid dimensions

are controlled with *fourierspacing* (page 45) in the same way as for electrostatics, and the interpolation order is controlled with *pme-order* (page 45). The relative accuracy of direct/reciprocal space is controlled by *ewald-rtol-lj* (page 45), and the specific combination rules that are to be used by the reciprocal routine are set using *lj-pme-comb-rule* (page 45).

Shift
This functionality is deprecated and replaced by using *vdwtype=Cut-off* (page 43) with *vdw-modifier=Force-switch* (page 44). The LJ (not Buckingham) potential is decreased over the whole range and the forces decay smoothly to zero between *rvdw-switch* (page 44) and *rvdw* (page 44).

Switch
This functionality is deprecated and replaced by using *vdwtype=Cut-off* (page 43) with *vdw-modifier=Potential-switch* (page 44). The LJ (not Buckingham) potential is normal out to *rvdw-switch* (page 44), after which it is switched off to reach zero at *rvdw* (page 44). 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 user for *coulombtype* (page 42). The function value at zero is not important. When you want to use LJ correction, make sure that *rvdw* (page 44) corresponds to the cut-off in the user-defined function. When *coulombtype* (page 42) 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 44) and *rvdw* (page 44). 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 44) and *rvdw* (page 44). 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
don't 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 40), separated by underscores. For example, if `energygrps = Na Cl Sol` and `energygrp-table = Na Na Na Cl`, *gmx mdrun* (page 183) 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 45) 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 43) by the PME tuning in *gmx mdrun* (page 183).

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 45) 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 43) by the PME tuning in *gmx mdrun* (page 183).

pme-order
(4) Interpolation order for PME. 4 equals cubic interpolation. You might try 6/8/10 when running in parallel and simultaneously decrease grid dimension.

ewald-rtol
(10^{-5}) The relative strength of the Ewald-shifted direct potential at *rcoulomb* (page 43) is given by *ewald-rtol* (page 45). 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 45) is used to control the relative strength of the dispersion potential at *rvdw* (page 44) in the same way as *ewald-rtol* (page 45) 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 shouldn't 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

tcoupl

no
No temperature coupling.

berendsen
Temperature coupling with a Berendsen thermostat to a bath with temperature *ref-t* (page 47), with time constant *tau-t* (page 47). Several groups can be coupled separately, these are specified in the *tc-grps* (page 47) field separated by spaces.

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 47) 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 47) is the average time between randomization of each molecule. Inhibits particle dynamics somewhat, but 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 47) is the time between randomization of all molecules. Inhibits particle dynamics somewhat, but 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 47), but the stochastic term ensures that a proper canonical ensemble is generated. The random seed is set with `ld-seed` (page 39). This thermostat works correctly even for `tau-t` (page 47) =0. For NVT simulations the conserved energy quantity is written to the energy and log file.

nsttcouple
(-1) The frequency for coupling the temperature. The default value of -1 sets `nsttcouple` (page 47) equal to 10, or fewer steps if required for accurate integration. Note that the default value is not 1 because additional computation and communication is required for obtaining the kinetic energy. For velocity Verlet integrators `nsttcouple` (page 47) 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 36) integrator only supports 1. Data for the NH chain variables is not printed to the *edr* (page 436) file by default, but can be turned on with the `print-nose-hoover-chain-variables` (page 47) 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 47)), -1 means no temperature coupling

ref-t
[K] reference temperature for coupling (one for each group in `tc-grps` (page 47))

Pressure coupling

pcoupl

no
No pressure coupling. This means a fixed box size.

Berendsen
Exponential relaxation pressure coupling with time constant `tau-p` (page 48). The box is scaled every `nstpcouple` (page 48) steps. It has been argued that this does not yield a correct thermodynamic ensemble, but it is the most efficient way to scale a box at the beginning of a run.

C-rescale
Exponential relaxation pressure coupling with time constant `tau-p` (page 48), including a stochastic term to enforce correct volume fluctuations. The box is scaled every `nstpcouple` (page 48) 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 48) 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. 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 36) or *integrator=md-vv-avek* (page 36), very similar to Parrinello-Rahman. As for Nose-Hoover temperature coupling the time constant *tau-p* (page 48) 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. Currently (as of version 5.1), it only supports isotropic scaling, and only works without constraints.

pcoupltype
Specifies the kind of isotropy of the pressure coupling used. Each kind takes one or more values for *compressibility* (page 48) and *ref-p* (page 48). Only a single value is permitted for *tau-p* (page 48).

isotropic
Isotropic pressure coupling with time constant *tau-p* (page 48). One value each for *compressibility* (page 48) and *ref-p* (page 48) 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 48) and *ref-p* (page 48) 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 48) 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 48) 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 frequency for coupling the pressure. The default value of -1 sets *nstpcouple* (page 48) equal to 10, or fewer steps if required for accurate integration. Note that the default value is not 1 because additional computation and communication is required for obtaining the virial. For velocity Verlet integrators *nstpcouple* (page 48) is set to 1.

tau-p
(1) [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 48).

ref-p
[bar] The reference pressure for coupling. The number of required values is implied by *pcoupltype* (page 48).

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 373) 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 373) 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 49).

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 49).

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 166) 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 166) according to a Maxwell distribution at temperature `gen-temp` (page 50), with random seed `gen-seed` (page 50). This is only meaningful with `integrator=md` (page 36).

gen-temp
(300) [K] temperature for Maxwell distribution

gen-seed
(-1) [integer] used to initialize random generator for random velocities, when `gen-seed` (page 50) 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.

h-angles
Convert all bonds to constraints and convert the angles that involve H-atoms to bond-constraints.

all-angles
Convert all bonds to constraints and all angles to bond-constraints.

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 51), 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 51). The root mean square relative constraint deviation is printed to the log file every `nstlog` (page 40) steps. If a bond rotates more than `lincs-warnangle` (page 51) 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 51), 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 usally 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 coninuation 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 an order of 8 or more might be required. With domain decomposition, the cell size is limited by the distance spanned by *lincs-order* (page 51) +1 constraints. When one wants to scale further than this limit, one can decrease *lincs-order* (page 51) and increase *lincs-iter* (page 51), since the accuracy does not deteriorate when (1+ *lincs-iter* (page 51))* *lincs-order* (page 51) 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 you might want to increase it to 2.

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
Pairs of energy groups for which all non-bonded interactions are excluded. An example: if you have two energy groups `Protein` and `SOL`, specifying `energygrp-excl = Protein Protein SOL SOL` would give only the non-bonded interactions between the protein and the solvent. This is especially useful for speeding up energy calculations with `mdrun -rerun` and for excluding interactions within frozen groups.

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 41) =`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 52). Energy groups `wall0` and `wall1` (for *nwall* (page 52) =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 45) 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 52) =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 52) =2, where one should use *ewald-geometry* (page 46) =`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 54)) and multiple related *mdp* (page 440) variables will exist accordingly. Documentation references to things like *pull-coord1-vec* (page 56) 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 440) 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 55)

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

pull-print-ref-value

no
do not print the reference value for each pull coordinate

yes
print the reference value for each pull coordinate

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 56)

pull-nstxout
(50) frequency for writing out the COMs of all the pull group (0 is never)

pull-nstfout
(50) frequency for writing out the force of all the pulled group (0 is never)

pull-pbc-ref-prev-step-com

no
Use the reference atom (`pull-group1-pbcatom` (page 54)) 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 54)), 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.

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 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 54). A value of 0 means that the middle atom (number wise) is used, which is only safe for small groups. *gmx grompp* (page 166) 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 55) 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 56) and *pull-coord1-rate* (page 56) are not used.

flat-bottom
At distances above *pull-coord1-init* (page 56) a harmonic potential is applied, otherwise no potential is applied.

flat-bottom-high
At distances below *pull-coord1-init* (page 56) 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* (page 54) is external-potential.

pull-coord1-geometry

distance
Pull along the vector connecting the two groups. Components can be selected with *pull-coord1-dim* (page 56).

direction
Pull in the direction of *pull-coord1-vec* (page 56).

direction-periodic
As *pull-coord1-geometry=direction* (page 55), 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 55), 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 56). 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 56). From the first of the two groups in *pull-coord1-groups* (page 56) a cylinder is selected around the axis going through the COM of the second group with direction *pull-coord1-vec* (page 56) with radius *pull-cylinder-r* (page 53). Weights of the atoms decrease continously to zero as the radial distance goes from 0 to *pull-cylinder-r* (page 53) (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 55) but the second vector is given by *pull-coord1-vec* (page 56). 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 56). Pull coordinates of lower indices 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. The mathematical expression are evaluated using muParser. Only relevant if *pull-coord1-geometry* (page 55) is set to `transformation`.

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* (page 55) is set to `transformation`.

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 can be 0, in which case an absolute reference of *pull-coord1-origin* (page 56) 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 53) = *pull-coord1-start=yes* (page 56). With *pull-coord1-geometry* (page 55) = *pull-coord1-geometry=distance* (page 55), 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 56) 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 166) normalizes the vector.

pull-coord1-start

no
do not modify *pull-coord1-init* (page 56)

yes
add the COM distance of the starting conformation to *pull-coord1-init* (page 56)

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 56) and *pull-coord1-rate* (page 56) 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 56), but for state B. This is only used when *free-energy* (page 63)

is turned on. The force constant is then (1 - lambda) * *pull-coord1-k* (page 56) + lambda * *pull-coord1-kB* (page 56).

AWH adaptive biasing

awh

no
No biasing.

yes
Adaptively bias a reaction coordinate using the AWH method and estimate the corresponding PMF. The PMF and other AWH data are written to energy file at an interval set by *awh-nstout* (page 57) 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. This is only allowed if *pull-coord1-type=external-potential* (page 55) and *pull-coord1-potential-provider* (page 55) = `awh` for the concerned pull coordinate indices. Pull geometry 'direction-periodic' is 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 57) 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 60). This option is not compatible with using the free energy lambda state as an AWH reaction coordinate.

umbrella
The potential bias is applied by controlling the position of an harmonic potential using Monte-Carlo sampling. The force constant is set with *awh1-dim1-force-constant* (page 60). The umbrella location is sampled using Monte-Carlo every *awh-nstsample* (page 57) steps. This is option is required when using the free energy lambda state as an AWH reaction coordinate. 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 183) option `-multidir`. The biases will be independent.

yes
With *gmx mdrun* (page 183) and option `-multidir` the bias and PMF estimates for biases with *awh1-share-group* (page 59) >0 will be shared across simulations with the biases with the same *awh1-share-group* (page 59) value. The simulations should have the same AWH settings for sharing to make sense. *gmx mdrun* (page 183) 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 57).

awh-nstout
(100000) Number of steps between printing AWH data to the energy file, should be a multiple of *nstenergy* (page 40).

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
(10) The number of coordinate samples used for each AWH update. The update interval in steps is *awh-nstsample* (page 57) 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 the given diffusion constant(s) *awh1-dim1-diffusion* (page 60) determine the initial biasing rate. The error is obviously not known *a priori*. Only a rough estimate of *awh1-error-init* (page 58) is needed however. As a general guideline, leave *awh1-error-init* (page 58) 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 59) option) then *awh1-error-init* (page 58) should reflect that knowledge.

awh1-growth

exp-linear

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. 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 57)). 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 58) but skip the initial stage. This may be useful if there is *a priori* knowledge (see *awh1-error-init* (page 58)) which eliminates the need for an initial stage. This is also the setting compatible with *awh1-target=local-boltzmann* (page 59).

awh1-equilibrate-histogram

no
Do not equilibrate histogram.

yes
Before entering the initial stage (see *awh1-growth=exp-linear* (page 58)), 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 20%). This option would typically only be used when *awh1-share-group* (page 59) > 0 and the initial configurations poorly represent the target distribution.

awh1-target

constant
The bias is tuned towards a constant (uniform) coordinate distribution in the defined sampling interval (defined by [*awh1-dim1-start* (page 60), *awh1-dim1-end* (page 60)]).

cutoff
Similar to *awh1-target=constant* (page 58), but the target distribution is proportional to 1/(1 + exp(F - *awh1-target=cutoff* (page 58))), 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 59). *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 59) 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 58), since for *awh1-growth=exp-linear* (page 58) histograms are globally rescaled in the initial stage.

awh1-target-beta-scaling
(0) For *awh1-target=boltzmann* (page 59) and *awh1-target=local-boltzmann* (page 59) it is the unitless beta scaling factor taking values in (0,1).

awh1-target-cutoff
(0) [kJ mol^{-1}] For *awh1-target=cutoff* (page 58) 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 58) = 1, *gmx mdrun* (page 183) will expect a file `awhinit.xvg` to be present in the run directory. For multiple biases, *gmx mdrun* (page 183) expects files `awhinit1.xvg`, `awhinit2.xvg`, etc. The file name can be changed with the `-awh` option. The first *awh1-ndim* (page 59) columns of each input file should contain the coordinate values, such that each row defines a point in coordinate space. Column *awh1-ndim* (page 59) + 1 should contain the PMF value (in kT) for each point. The target distribution column can either follow the PMF (column *awh1-ndim* (page 59) + 2) or be in the same column as written by *gmx awh* (page 116).

awh1-share-group

0
Do not share the bias.

positive
Share the bias and PMF estimates within and/or between simulations. Within a simulation, the bias will be shared between biases that have the same *awh1-share-group* (page 59) index (note that the current code does not support this). With *awh-share-multisim=yes* (page 57) and *gmx mdrun* (page 183) option `-multidir` the bias will also be shared across simulations. Sharing may increase convergence initially, although the starting configurations can be critical, especially when sharing between many biases. Currently, positive group values should start at 1 and increase by 1 for each subsequent bias that is shared.

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 lambda state is the reaction coordinate for this dimension. The lambda states to use are specified by *fep-lambdas* (page 63), *vdw-lambdas* (page 64), *coul-lambdas* (page 64) etc. This is not compatible with delta-lambda. It also requires calc-lambda-neighbors to be -1. With multiple time-stepping, AWH should be in the slow level. This option requires *awh-potential=umbrella* (page 57).

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 [rad] 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 55)). For dihedral geometries *awh1-dim1-start* (page 60) greater than *awh1-dim1-end* (page 60) 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 [rad] End value defining the sampling interval together with *awh1-dim1-start* (page 60).

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 [rad] 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 58)). 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 59)>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 440) 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 440) file, they unfortunately generate warnings).

yes
Apply the rotation potential specified by *rot-type0* (page 61) to the group of atoms given under the *rot-group0* (page 61) 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 frequency (in steps) for the angle of the rotation group, as well as for the torque and the rotation potential energy.

rot-nstsout
(1000) Output frequency 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 62) 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 65)), or the Hamiltonian differences with respect to other lambda values (as specified with foreign lambda) to the energy file and/or to `dhdl.xvg`, where they can be processed by, for example *gmx bar* (page 117). The potentials, bond-lengths and angles are interpolated linearly as described in the manual. When `sc-alpha` (page 64) 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 63)). In other cases, `init-lambda-state` (page 63) should be specified instead. 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 columm of the lambda vector (`coul-lambdas` (page 64), `vdw-lambdas` (page 64), `bonded-lambdas` (page 64), `restraint-lambdas` (page 64), `mass-lambdas` (page 64), `temperature-lambdas` (page 64), `fep-lambdas` (page 63)) should be used. This is a zero-based index: `init-lambda-state` (page 63) 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 65) steps. Values must be between 0 and 1. Free energy differences between different lambda values can then be determined with *gmx bar* (page 117).

fep-lambdas (page 63) is different from the other -lambdas keywords because all components of the lambda vector that are not specified will use *fep-lambdas* (page 63) (including *restraint-lambdas* (page 64) and therefore the pull code restraints).

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 65) steps. 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 65) steps. 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 65) steps. Values must be between 0 and 1. 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 65) steps. Values must be between 0 and 1. 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 65) steps. Values must be between 0 and 1. 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 65) steps. Values must be between 0 and 1. Only the temperatures 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 63) has been set. A positive value will limit the number of lambda points calculated to only the nth neighbors of *init-lambda-state* (page 63): for example, if *init-lambda-state* (page 63) 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 117), a value of 1 is sufficient, while for MBAR -1 should be used.

sc-alpha
(0) the soft-core alpha parameter, a value of 0 results in linear interpolation of the LJ and Coulomb interactions

sc-r-power
(6) power 6 for the radial term in the soft-core equation.

sc-coul
(no) Whether to apply the soft-core free energy interaction transformation to the Columbic interaction of a molecule. Default is no, as it is generally more efficient to turn off the Coulomic interactions linearly before turning off the van der Waals interactions. Note that it is only taken into account when lambda states are used, not with *couple-lambda0* (page 65) / *couple-lambda1* (page 65), and you can still turn off soft-core interactions by setting *sc-alpha* (page 64) to 0.

sc-power
(0) the power for lambda in the soft-core function, only the values 1 and 2 are supported

sc-sigma
(0.3) [nm] the soft-core sigma for particles which have a C6 or C12 parameter equal to zero or a sigma smaller than *sc-sigma* (page 64)

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 63) 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 65) and *couple-lambda1* (page 65). 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 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-lambda1* (page 65), but for lambda=1

couple-intramol

no
All intra-molecular non-bonded interactions for moleculetype *couple-moltype* (page 65) 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 intra-molecular non-bonded interactions might lead to kinetically trapped vacuum conformations. The 1-4 pair interactions are not turned off.

nstdhdl
(100) the frequency for writing dH/dlambda and possibly Delta H to dhdl.xvg, 0 means no ouput, should be a multiple of *nstcalcenergy* (page 40).

dhdl-derivatives
(yes)

If yes (the default), the derivatives of the Hamiltonian with respect to lambda at each *nstdhdl* (page 65) step are written out. These values are needed for interpolation of linear energy differences with *gmx bar* (page 117) (although the same can also be achieved with the right foreign lambda 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 the foreign lambda and *dhdl-derivatives* (page 65) settings) are written out to a separate file, with the default name `dhdl.xvg`. This file can be used directly with *gmx bar* (page 117).

no
The free energy values are written out to the energy output file (`ener.edr`, in accumulated blocks at every *nstenergy* (page 40) steps), where they can be extracted with *gmx energy* (page 155) or used directly with *gmx bar* (page 117).

dh-hist-size
(0) If nonzero, specifies the size of the histogram into which the Delta H values (specified with foreign lambda) 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 40) step. Be aware that incorrect histogram settings (too small size or too wide bins) can introduce errors. Do not use histograms unless you're 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 66). This size limits the accuracy with which free energies can be calculated. Do not use histograms unless you're 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 40), but can be greater or less than *nstdhdl* (page 65).

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 critera 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 critera 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 67) 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 47) 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 67). If we define the Nratio = (number of samples at each histogram) / (average number of samples at each histogram). *wl-ratio* (page 67) 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 67). 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 67) is true, then when the incrementor becomes less than 1/N, where N is the mumber 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 67) is ignored, but the weights will still stop updating when the equilibration criteria set in *lmc-weights-equil* (page 68) 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 68) means that only states plus or minus *lmc-gibbsdelta* (page 68) 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 68) steps at each state before moving on to the next lambda state. If *lmc-forced-nstart* (page 68) 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) Frequency 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 66) 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 68) 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 63), except the weights can be any floating point number. Units are kT. 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 64) part of the lambda vector.

linear
Linearly interpolates the temperatures using the values of *temperature-lambdas* (page 64), *i.e.* if *sim-temp-low* (page 69) =300, *sim-temp-high* (page 69) =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 69) and *sim-temp-high* (page 69). The i:th state has temperature *sim-temp-low* (page 69) * (*sim-temp-high* (page 69) / *sim-temp-low* (page 69)) 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 69) and *sim-temp-high* (page 69). The i:th state has temperature *sim-temp-low* (page 69) + (*sim-temp-high* (page 69) - *sim-temp-low* (page 69))*((exp(*temperature-lambdas* (page 64) (i))-1)/(exp(1.0)-i)).

Non-equilibrium MD

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 69) specifies for which dimension(s) the freezing applies. To avoid spurious contributions to the virial and pressure due to large forces between completely frozen atoms you need to use energy group exclusions, this also saves computing time. Note that coordinates of frozen atoms are not scaled by pressure-coupling algorithms.

freezedim
dimensions for which groups in *freezegrps* (page 69) 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 69) 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 69) is non-zero are calculated as: box(ts)+(t-ts)*deform, off-diagonal elements are corrected for periodicity. The coordinates are transformed accordingly. Frozen degrees of freedom are (purposely) also transformed. 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.

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 70) (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 477) and in ref. *146* (page 534).

Mixed quantum/classical molecular dynamics

QMMM-grps

groups to be descibed 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
Channel #1 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-group #1.

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-width
(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 183) 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 θ degress by using 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 *Mixed Quantum-Classical simulation techniques* (page 487).

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.

BLYP
DFT using BLYP functional and DZVP-MOLOPT basis set.

INPUT
Provide an external input file for CP2K when running *gmx grompp* (page 166) with the `-qmi` command-line option. External input files are subject to the limitations that are described in *Mixed Quantum-Classical simulation techniques* (page 487).

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.

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 440) and *tpr* (page 446) files cannot be mistakenly misused, we still parse this option. *gmx grompp* (page 166) and *gmx mdrun* (page 183) will issue a fatal error if this is set.

adress

(no)

implicit-solvent

(no)

3.8 Useful mdrun features

This section discusses features in *gmx mdrun* (page 183) that don't 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 446) 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 130) and *gmx trjconv* (page 237)). 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 40), since *gmx mdrun* (page 183) 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 183) 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 183) 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 183) 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 183) 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 183) ranks is sufficient, this signal should not be sent to mpirun or the *gmx mdrun* (page 183) 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.

This feature requires *configuring |Gromacs| with an external MPI library* (page 6) 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.

There are two ways of organizing files when running such simulations. All of the normal mechanisms work in either case, including `-deffnm`.

-multidir You must create a set of `n` directories for the `n` simulations, place all the relevant input files in those directories (e.g. named `topol.tpr`), and run 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, then 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 ...`. This option is generally the most convenient to use. `gmx mdrun -table` for the group cutoff-scheme works only in this mode.

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.

3.8.5 Controlling the length of the simulation

Normally, the length of an MD simulation is best managed through the *mdp* (page 440) option *nsteps* (page 37), however there are situations where more control is useful. `gmx mdrun -nsteps 100` overrides the *mdp* (page 440) 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 packages. The information presented should help choosing appropriate parallelization options, run configuration, as well as acceleration options to achieve optimal simulation performance.

The GROMACS build system and the *gmx mdrun* (page 183) 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 183), the automatic machinery works well enough. But to get the most from your hardware to maximize your scientific quality, read on!

3.9.1 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 processors often have more than one 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 laptop or desktop computer 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, 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 183) 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 183) 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 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) and Intel integrated GPUs; NVIDIA hardware is also supported.

SIMD A type of CPU instruction by which modern CPU cores can execute multiple floating-point instructions in a single cycle.

3.9.2 Work distribution by parallelization in GROMACS

The algorithms in *gmx mdrun* (page 183) and their implementations are most relevant when choosing how to make good use of the hardware. For details, see the *Reference Manual* (page 301). 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 183) will make extensive use of the SIMD capabilities of the core. There are various *command-line options* (page 80) 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.

3.9.3 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 10) 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 183) should be compiled be compiled separately with the highest (latest) `native` SIMD instruction set of the target architecture (supported by GROMACS).

Recent 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 183) 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 assumed to be known at CMake time). Consequently, the SVE vector length must be 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 554) 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 554) might crash before reaching the check (that would abort with a user-friendly error message).

Process(-or) level parallelization via OpenMP

GROMACS *mdrun* (page 183) 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 1.1 specification, based on the system threading support. Both POSIX pthreads and Windows threads are supported, thus providing great portability to most UNIX/Linux and Windows operating systems. Acting as a drop-in replacement for MPI, thread-MPI enables compiling and running *mdrun* (page 183) on a single machine (i.e. not across a network) without MPI. Additionally, it not only provides a convenient way to use computers with multicore CPU(s), but thread-MPI does in some cases make *mdrun* (page 183) run slightly faster than with MPI.

Thread-MPI is included in the GROMACS source and it is the default parallelization since version 4.5, practically rendering the serial *mdrun* (page 183) deprecated. Compilation with thread-MPI is controlled by the `GMX_THREAD_MPI` CMake variable.

Thread-MPI is compatible with most *mdrun* (page 183) features and parallelization schemes, including OpenMP, GPUs; it is not compatible with MPI and multi-simulation runs.

By default, the thread-MPI mdrun 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 6).

The native GPU acceleration can be turned on or off, either at run-time using the *mdrun* (page 183) `-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 GPUs used has to match the number of 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 183) 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 iterations of tuning are executed during the first 100 to 1000 MD steps. These iterations 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 43) =`rvdw` *mdp* (page 440) 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 don't allow scaling to shorter cut-off as we don't want to change `rvdw` 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 faster than the GPU 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 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 78) with dynamic load balancing. To parallelize simulations across multiple machines (e.g. nodes of a cluster) *mdrun* (page 183) 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 183) 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 183) 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 183) 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 183) 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 multi-threaded 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 mutithreading 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 241) tool.

The number of PME ranks can be set manually on the *mdrun* (page 183) 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.

3.9.4 Running mdrun within a single node

gmx mdrun (page 183) 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, 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 183) within a single node, but runs more slowly than the thread-MPI version.

By default, *gmx mdrun* (page 183) 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 183) 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 64. 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" or "off" to control whether *mdrun* (page 183) will attempt to set the affinity of threads to cores. Defaults to "auto," which means that if *mdrun* (page 183) detects that all the cores on the node are being used for *mdrun* (page 183), then it should behave like "on," and attempt to set the affinities (unless they are already set by something else).

-pinoffset If `-pin on`, specifies the logical core number to which *mdrun* (page 183) should pin the first thread. When running more than one instance of *mdrun* (page 183) on a node, use this option to to avoid pinning threads from different *mdrun* (page 183) instances to the same core.

-pinstride If `-pin on`, specifies the stride in logical core numbers for the cores to which *mdrun* (page 183) should pin its threads. When running more than one instance of *mdrun* (page 183) on a node, use this option to avoid pinning threads from different *mdrun* (page 183) instances to the same core. Use the default, 0, to minimize the number of threads per physical core - this lets

mdrun (page 183) manage the hardware-, OS- and configuration-specific details of how to map logical cores to physical cores.

-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", "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", "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", "gpu." Defaults to "auto," which uses a compatible CUDA 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 interactions 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", "gpu." Defaults to "auto," which currently always uses the CPU. Setting "gpu" requires that a compatible CUDA GPU is available, the simulation uses a single rank. Update and constraints on a GPU is currently not supported with mass and constraints free-energy perturbation, domain decomposition, virtual sites, Ewald surface correction, replica exchange, constraint pulling, orientation restraints and computational electrophysiology.

-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 183). 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 183) 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 183). 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 183), 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 91) 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 183) 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 89).

-pmefft Allows choosing whether to execute the 3D FFT computation on a CPU or GPU. Can be set to "auto", "cpu", "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 183) using all the available resources. *mdrun* (page 183) 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 183) 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 183) 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 183) 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 183) 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 183) 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 183) 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 183) 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 183) 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 183) 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 an *gmx mdrun* (page 183) 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 183) will be local to that node.

3.9.5 Running mdrun on more than one node

This requires configuring GROMACS to build with an external MPI library. By default, this *mdrun* (page 183) executable is run with `gmx_mpi mdrun`. All of the considerations for running single-node *mdrun* (page 183) 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 various aspects of the PME and DD algorithms, shifting load between ranks and/or GPUs to maximize throughput. Some *mdrun* (page 183) features are not compatible with this, and these ignore this option.

-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.

During the simulation *gmx mdrun* (page 183) 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 35), so that the period between communication phases is the least common denominator of `nstlist` (page 40), `nstcalcenergy` (page 40), `nsttcouple` (page 47), and `nstpcouple` (page 48).

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 241) utility is available to search a wider range of parameter space, including making safe modifications to the *tpr* (page 446) 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 183) 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 183) 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 183) with 16 ranks, as above, and require that 5 of them are dedicated to the PME component.

```
mpirun -np 11 gmx_mpi mdrun -ntomp 2 -npme 6 -ntomp_pme 1
```

Starts *gmx mdrun* (page 183) with 11 ranks, as above, and require that six of them are dedicated to the PME component with one OpenMP thread each. The remaining five 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 183) 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 183) 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 183) 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 183) 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 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 183) will suffer greatly.

```
mpirun -np 20 gmx_mpi mdrun -gpu_id 01
```

Starts *gmx mdrun* (page 183) 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.

3.9.6 Approaching the scaling limit

There are several aspects of running a GROMACS simulation that are important as the number of atoms per core approaches the current scaling limit of ~100 atoms/core.

One of these is that the use of `constraints = all-bonds` with P-LINCS sets an artificial minimum on the size of domains. You should reconsider the use of constraints to all bonds (and bear in mind possible consequences on the safe maximum for dt), or change lincs_order and lincs_iter suitably.

3.9.7 Finding out how to run mdrun better

The Wallcycle module is used for runtime performance measurement of *gmx mdrun* (page 183). 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 183) 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 precentage of the total runtime each row represents. Note that the *gmx mdrun* (page 183) 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

As performance data is collected for every run, they are essential to assessing and tuning the performance of *gmx mdrun* (page 183) performance. Therefore, they benefit both code 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 183) 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 574) for more information.

3.9.8 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 90), *PME* (page 90), *bonded interactions* (page 90) and *update and constraints* (page 91) 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 78).

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 90), and *PME* (page 90).

Please note that the solving of PME on GPU is still only the initial version supporting this behaviour, and comes with a set of limitations outlined further below.

Right now, we generally support short-range nonbonded offload with and without dynamic pruning on a wide range of GPU accelerators (both NVIDIA and AMD). 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 is a new feature that that has some restrictions in terms of feature and parallelization compatibility (please see the *section below* (page 90)).

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 now allows the 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.
- PME will run on a GPU only when exactly one rank has a PME task, ie. decompositions with multiple ranks doing PME are not supported.
- Only single precision is supported.
- Only dynamical integrators are supported (ie. leap-frog, Velocity Verlet, stochastic dynamics)
- LJ PME is not supported on GPUs.

GPU accelerated calculation of bonded interactions (CUDA only)

GROMACS now allows the offloading of the bonded part of the PP workload to a CUDA-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 only)

GROMACS makes it possible to also perform the coordinate update and (if requested) constraint calculation on a CUDA-compatible GPU. This allows executing all (supported) computation of a simulation step on the GPU. This feature is supported in single domain runs (unless using the experimental GPU domain decomposition feature), and needs to be explicitly requested by the user. This is a new parallelization mode where all force and coordinate data can be "GPU resident" for a number of steps, typically between neighbor searching steps. 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 this scheme it is however still possible for part of the computation to be executed 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.

It is possible to change the default behaviour by setting the `GMX_FORCE_UPDATE_DEFAULT_GPU` environment variable to a non-zero value. In this case simulations will try to run all parts by default on the GPU, and will only fall back to the CPU based calculation if the simulation is not compatible.

Using this parallelization mode is typically advantageous in cases where a fast GPU is used with a weak 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 183).

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, NB and PME. Each PP rank has a NB 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 NB 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.

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) 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 calculation of the NB.
3) With fast/modern GPUs and/or slow/old CPUs with few cores, it generally helps to have the GPU do PME.
4) 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 183) will default to no bonded offload when PME is offloaded. Typical cases where performance can be improvement 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).
5) It *is* possible to use multiple GPUs with PME offload by letting e.g. 3 MPI ranks use one GPU each for short-range interactions, while a fourth rank does the PME on its GPU.
6) The only way to know for sure what 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 long-range PME electrostatics). Asynchronous task launches are handled by GPU device driver and require CPU involvement. Therefore, the work of scheduling GPU tasks will incur an overhead that can in some cases significantly delay or interfere with the CPU execution.

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 overhead is measured by *gmx mdrun* (page 183) and reported in the performance summary section of the log file ("Launch GPU ops" row). A few percent of runtime spent in this category is normal, but in fast-iterating and multi-GPU parallel runs 10% or larger overheads can be observed. In general, a user can do little to avoid such overheads, but there are a few cases where tweaks can give performance benefits. In single-rank 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. The performance impact will be most significant on NVIDIA GPUs with CUDA, less on AMD and Intel with OpenCL. 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 second type of overhead, interference of the GPU driver with CPU computation, is caused by the scheduling and coordination of GPU tasks. A separate GPU driver thread can require CPU resources which may clash with the concurrently running non-offloaded tasks, potentially degrading the performance of PME or bonded force computation. This effect is most pronounced when using AMD GPUs with OpenCL with older driver releases (e.g. fglrx 12.15). To minimize the overhead it is recommended to leave a CPU hardware thread unused when launching *gmx mdrun* (page 183), especially on CPUs with high core counts and/or HyperThreading enabled. E.g. on a machine with a 4-core CPU and eight threads (via HyperThreading) and an AMD GPU, try `gmx mdrun -ntomp 7 -pin on`. This will leave free CPU resources for the GPU task scheduling reducing interference with CPU computation. Note that assigning fewer resources to *gmx mdrun* (page 183) CPU computation involves a tradeoff which may outweigh the benefits of reduced GPU driver overhead, in particular without HyperThreading and with few CPU cores.

3.9.9 Running the OpenCL version of mdrun

Currently supported hardware architectures are: - GCN-based AMD GPUs; - NVIDIA GPUs (with at least OpenCL 1.2 support); - 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. .. seealso:: Issue 3268 add more Intel driver recommendations

The minimum OpenCL version required is unknown. See also the *known limitations* (page 93).

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 183) 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 devices, or to define a mapping of GPUs to PP ranks, is used for OpenCL devices.

Some other *OpenCL management* (page 287) 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 and Xeon Phi are not supported.
- 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 an 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.

Limitations of interest to GROMACS developers:

- The current implementation requires a minimum execution with of 16; kernels compiled for narrower execution width (be it due to hardware requirements or compiler choice) will not be suitable and will trigger a runtime error.

3.9.10 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

- Don't use double precision unless you're absolute 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, especially for gcc (e.g. from version 5 to 6 the performance of the compiled code improved a lot).
- MPI library: OpenMPI usually has good performance and causes little trouble.
- Make sure your compiler supports OpenMP (some versions of Clang don't).
- If you have GPUs that support either CUDA or OpenCL, use them.
  - Configure with `-DGMX_GPU=CUDA `` or ``-DGMX_GPU=OpenCL`.
  - For CUDA, use the newest CUDA available for your GPU to take advantage of the latest performance enhancements.
  - Use a recent GPU driver.
  - Make sure you use an *gmx mdrun* (page 183) 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 78)).
  - 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 fs, use *constraints=h-bonds* (page 50) (and not *constraints=all-bonds* (page 50)), since this is faster, especially with GPUs, and most force fields have been parametrized with only bonds involving hydrogens constrained.
- You can increase the time-step to 4 or 5 fs when using virtual interaction sites (`gmx pdb2gmx -vsite h`).
- 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 241) 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.
- 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.
- If the neighbor searching takes a lot of time, increase nstlist. If a Verlet buffer tolerance is used, this is done automatically by *gmx mdrun* (page 183) and the pair-list buffer is increased to keep the energy drift constant.
  - 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.

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 200) water box that is 10^3 times larger than what they think it is (e.g. *gmx solvate* (page 225)).
- 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's 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 200) does not have an entry in the *residue database* (page 443) for XXX. The *residue database* (page 443) 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 200) to build a *top* (page 445) file. A *residue database* (page 443) 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 200) on a *PDB* (page 442) file they have, without consideration of the contents of the file. A *force field* (page 283) is not magical, it can only deal with molecules or residues (building blocks) that are provided in the *residue database* (page 443) or included otherwise.

If you want to use *pdb2gmx* (page 200) to automatically generate your topology, you have to ensure that the appropriate *rtp* (page 443) entry is present within the desired *force field* (page 283) and has the same name as the building block you are trying to use. If you call your molecule "HIS," then *pdb2gmx* (page 200) will try to build histidine, based on the `[ HIS ]` entry in the *rtp* (page 443) file, so it will look for the exact atomic entries for histidine, no more no less.

If you want a *topology* (page 445) for an arbitrary molecule, you cannot use *pdb2gmx* (page 200) (unless you build the *rtp* (page 443) entry yourself). You will have to build that entry by hand, or use another program (such as *x2top* (page 253) or one of the scripts contributed by users) to build the *top* (page 445) 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 443) and rename as appropriate,
- parameterize the residue / molecule yourself (lots of work, even for an expert),
- find a *topology file* (page 445) for the molecule, convert it to an *itp* (page 439) file and include it in your *top* (page 445) file,
- use another *force field* (page 283) 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.

Long bonds and/or missing atoms

There are probably atoms missing earlier in the *pdb* (page 442) file which makes *pdb2gmx* (page 200) go crazy. Check the screen output of *pdb2gmx* (page 200), as it will tell you which one is missing. Then add the atoms in your *pdb* (page 442) 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 434) fed to *pdb2gmx* (page 200), 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 200) expects certain atoms within the given residue, based on the entries in the force field *rtp* (page 443) 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 438) file is missing. More likely, the nomenclature of your hydrogen atoms simply does not match what is expected by the *rtp* (page 443) entry. In this case, use `-ignh` to allow *pdb2gmx* (page 200) 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 33), 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 442) file as `ALA`, but rather `NALA`, as specified in the ffamber instructions.
- Atoms are simply missing in the structure file provided to *pdb2gmx* (page 200); look for `REMARK 465` and `REMARK 470` entries in the *pdb* (page 442) 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 443) entries. If you find yourself using `-missing` in order to generate a topology for a protein or nucleic acid, don't; 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 200), the atom names are expected to match those found in the *rtp* (page 443) 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 434) 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 283), 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 283), 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 200) 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 17) 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 445) or *force field* (page 283) 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 445) file, *itp* (page 439), 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 445) file where you choose the *force field* (page 283). If you are trying to mix two *force fields* (page 283), then you are asking for trouble. If a molecule *itp* (page 439) 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 24) 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 445) or *force field* (page 283) files in the inappropriate location; the `[defaults]` section can only appear once and must be the first directive in the *topology* (page 445). The `[defaults]` directive is typically present in the *force field* (page 283) file (forcefield.itp), and is added to the *topology* (page 445) 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 283) 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 439) 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 165) 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 445). If *pdb2gmx* (page 200) 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 289) 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 535) (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 445) file for the system. You've not given enough parameters for one of the bonded definitions. Sometimes this also occurs if you've mangled the *Include File Mechanism* (page 24) 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 445) file, *top* (page 445), the total number of atoms or particles within the system does not match exactly with what is provided within the *coordinate file* (page 434), often a *gro* (page 437) or a *pdb* (page 442).

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 445) file must have a corresponding `[ moleculetype ]` section defined previously, either in the *top* (page 445) file or an *included* (page 24) *itp* (page 439) file. See the reference manual section 5.6.1 for the syntax description. Your *top* (page 445) file doesn't 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 278) (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 some 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 283) 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 445) 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 439) 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 24) with `#include` directives.

This error can also arise if you are using a water model that is not enabled for use with your chosen *force field* (page 283) by default. For example, if you are attempting to use the SPC water model with an *AMBER force field* (page 33), you will see this error. The reason is that, in `spc.itp`, there is no `#ifdef` statement defining atom types for any of the *AMBER force fields* (page 33). You can either add this section yourself, or use a different water model.

XXX non-matching atom names

This error usually indicates that the order of the *topology* (page 445) file does not match that of the *coordinate file* (page 434). When running *grompp* (page 166), the program reads through the *topology* (page 445), mapping the supplied parameters to the atoms in the *coordinate* (page 434) 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 434) file that has the old (pre-4.5) ion nomenclature. In this case, allowing *grompp* (page 166) 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 undoubtedly *blow up* (page 280).

The sum of the two largest charge group radii (X) is larger than rlist - rvdw/rcoulomb

This error warns that some combination of settings will result in poor energy conservation at the longest cutoff, which occurs when charge groups move in or out of pair list range. The error can have two sources:

- Your charge groups encompass too many atoms. Most charge groups should be less than 4 atoms or less.
- Your *mdp* (page 440) settings are incompatible with the chosen algorithms. For switch or shift functions, rlist must be larger than the longest cutoff (`rvdw` or `rcoulomb`) to provide buffer space for charge groups that move beyond the neighbor searching radius. If set incorrectly, you may miss interactions, contributing to poor energy conservation.

A similar error ("The sum of the two largest charge group radii (X) is larger than rlist") can arise under two following circumstances:

- The charge groups are inappropriately large or rlist is set too low.
- Molecules are broken across periodic boundaries, which is not a problem in a periodic system. In this case, the sum of the two largest charge groups will correspond to a value of twice the box vector along which the molecule is broken.

Invalid line in coordinate file for atom X

This error arises if the format of the *gro* (page 437) file is broken in some way. The most common explanation is that the second line in the *gro* (page 437) file specifies an incorrect number of atoms, causing *grompp* (page 166) to continue searching for atoms but finding box vectors.

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 orderof -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 183) may suddenly stop (perhaps after writing several *pdb* (page 442) files) after a series of warnings about the constraint algorithms (e.g. LINCS, SETTLE or SHAKE) are written to the *log* (page 439) file. These algorithms often used to constrain bond lengths and/or angles. When a system is *blowing up* (page 280) (i.e. exploding due to diverging forces), the constraints are usually the first thing to fail. This doesn't 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 281).

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 280). 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 doesn't help, check the validity of the parameters in your *topology* (page 445)!

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 76), 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 440) file) set to 0, this will suppress most output.
- Your disk might be full. Eventually this will lead to *mdrun* (page 183) crashing, but since output is buffered, it might take a while for mdrun to realize it can't write.

Can not do Conjugate Gradients with constraints

This means you can't 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 280). This can mean that the system isn't 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 48), 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 - start with the much more forgiving Berendsen method first, then switch to other algorithms.

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 280). 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 183) tells you your system is *blowing up* (page 280). 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 280) 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 183) tried to partition your simulation cell into chunks, it couldn't. 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 166) warned you about a large charge group, pay attention and reconsider its size. *mdrun* (page 183) prints a breakdown of how it computed this minimum size in the *log* (page 439) file, so you can perhaps find a cause there.

If you didn't 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 183) the `-ntmpi 1` command line option. Otherwise, you might be using an MPI-enabled GROMACS and not be aware of the fact.

3.11 Command-line reference

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 (yes) 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-extract-cluster(1) Allows extracting frames corresponding to clusters from trajectory

gmx-freevolume(1) Calculate free volume

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-select(1) Print general information about selections

gmx-trajectory(1) Print coordinates, velocities, and/or forces for selections

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 modifed run-input file

Viewing trajectories

gmx-nmtraj(1) Generate a virtual oscillating trajectory from an eigenvector

gmx-view(1) View a trajectory on an X-Windows terminal

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(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(1) Compute small angle neutron scattering spectra

gmx-saxs(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(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-do_dssp(1) Assign secondary structure and calculate solvent accessible surface area

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 132)) or of a Normal Modes analysis (*gmx nmeig* (page 191)).

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 112).

`-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 442) file with two or three structures (you can use `rasmol -nmrpdb` to view such a *.pdb* (page 442) 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 447) *cpt* (page 435) *tng* (page 444)

-v2 [<.trr/.cpt/...>] (eigenvec2.trr) (Optional) Full precision trajectory: *trr* (page 447) *cpt* (page 435) *tng* (page 444)

-f [<.xtc/.trr/...>] (traj.xtc) (Optional) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) (Optional) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 437) *g96* (page 437) *pdb* (page 442) brk ent esp

-filt [<.xtc/.trr/...>] (filtered.xtc) (Optional) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-extr [<.xtc/.trr/...>] (extreme.pdb) (Optional) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 170). 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 170). 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 dihedrals) as function of time, but this is probably only fun for a select few.

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 132).

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.

Options

Options to specify input files:

-f [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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. 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 440) 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 440) 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 436) files. These can contain either lists of energy differences (see the *.mdp* (page 440) option `separate_dhdl_file`), or a series of histograms (see the *.mdp* (page 440) 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 440) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 444), *.trr* (page 447) or *.xtc* (page 448)), an energy file (*.edr* (page 436)) or an index file (*.ndx* (page 441)) 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 446) 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 446)) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-f2` [<.xtc/.trr/...>] (traj.xtc) (Optional) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-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 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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>] [-[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 450) file (argument `-o`) and optionally as a *.pdb* (page 442) 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 450) 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 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

-f [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) files

-xvg <enum> (xmgrace) xvg plot formatting: xmgrace, xmgr, none

-r0 <int> (1) starting 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 447) plot

-bfact <real> (-1) B-factor value for *.pdb* (page 442) 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 considere 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

- Produces MANY output files (up to about 4 times the number of residues in the protein, twice that if autocorrelation functions are calculated). Typically several hundred files are output.
- phi and 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 207).
- `-r0` option does not work properly
- 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 447) 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 447) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 447) 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 447) file. The total number of clusters is written to an *.xvg* (page 450) 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 450) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 446) 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 447) 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 442) file the two structures will be written as separate models (use `rasmol -nmrpdb`). Also in a *.pdb* (page 442) 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 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

-f2 [<.gro/.g96/...>] (conf2.gro) Structure file: *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

-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 437) *g96* (page 437) *pdb* (page 442) brk ent esp

-no [<.ndx>] (match.ndx) (Optional) Index file

Other options:

-[no]w (no) View output *.xvg* (page 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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]zeroq]
```

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 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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]zeroq (no) Set the charges of a group (from the index) to zero

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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

-n [<.ndx>] (index.ndx) (Optional) Extra index groups

Options to specify output files:

-o [<.xtc/.trr/...>] (trajout.xtc) Output trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

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 110).

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 447) file.

Option `-xpma` writes the atomic covariance matrix to an *.xpm* (page 447) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 447) *cpt* (page 435) *tng* (page 444)

-av [<.gro/.g96/...>] (average.pdb) Structure file: *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

`-n` [<.ndx>] (index.ndx) (Optional) Index file

`-f` [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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] [-[no]relative]
```

Description

`gmx density` computes partial densities across the box, using an index file.

For the total density of NPT simulations, use *gmx energy* (page 155) 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. Option `-relative` performs the binning in relative instead of absolute box coordinates, and scales the final output with the average box dimension along the output axis. This can be used in combination with `-center`.

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.

Even the centered calculation will lead to some smearing out the output profiles, as lipids themselves are compressed and expanded. In most cases you probably want this (since it corresponds to macroscopic experiments), but if you want to look at molecular details you can use the `-relative` option to attempt to remove even more of the effects of volume fluctuations.

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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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.

-[no]relative (no) Use relative coordinates for changing boxes and scale output by average dimensions.

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 447) 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 436) file with `-od`, instead of the usual *.xpm* (page 447) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) (Optional) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 140). 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 447) 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 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 199).

Options

Options to specify input files:

-f [<.xtc/.trr/...>] (traj.xtc) (Optional) Input trajectory or single configuration: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 do_dssp

Synopsis

```
gmx do_dssp [-f [<.xtc/.trr/...>]] [-s [<.tpr/.gro/...>]] [-n [<.ndx>]]
            [-map [<.map>]] [-ssdump [<.dat>]] [-o [<.xpm>]]
            [-sc [<.xvg>]] [-a [<.xpm>]] [-ta [<.xvg>]]
            [-aa [<.xvg>]] [-b <time>] [-e <time>] [-dt <time>]
            [-tu <enum>] [-[no]w] [-xvg <enum>] [-sss <string>]
            [-ver <int>]
```

Description

`gmx do_dssp` reads a trajectory file and computes the secondary structure for each time frame calling the dssp program. If you do not have the dssp program, get it from http://swift.cmbi.ru.nl/gv/dssp. `gmx do_dssp` assumes that the dssp executable is located in `/usr/local/bin/dssp`. If this is not the case, then you should set an environment variable `DSSP` pointing to the dssp executable, e.g.:

```
setenv DSSP /opt/dssp/bin/dssp
```

Since version 2.0.0, dssp is invoked with a syntax that differs from earlier versions. If you have an older version of dssp, use the `-ver` option to direct do_dssp to use the older syntax. By default,

do_dssp uses the syntax introduced with version 2.0.0. Even newer versions (which at the time of writing are not yet released) are assumed to have the same syntax as 2.0.0.

The structure assignment for each residue and time is written to an *.xpm* (page 447) matrix file. This file can be visualized with for instance `xv` and can be converted to postscript with `xpm2ps`. Individual chains are separated by light grey lines in the *.xpm* (page 447) and postscript files. The number of residues with each secondary structure type and the total secondary structure (`-sss`) count as a function of time are also written to file (`-sc`).

Solvent accessible surface (SAS) per residue can be calculated, both in absolute values (A^2) and in fractions of the maximal accessible surface of a residue. The maximal accessible surface is defined as the accessible surface of a residue in a chain of glycines. **Note** that the program [gmx-sas] can also compute SAS and that is more efficient.

Finally, this program can dump the secondary structure in a special file `ssdump.dat` for usage in the program *gmx chi* (page 122). Together these two programs can be used to analyze dihedral properties as a function of secondary structure type.

Options

Options to specify input files:

`-f` [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

`-n` [<.ndx>] (index.ndx) (Optional) Index file

`-map` [<.map>] (ss.map) (Library) File that maps matrix data to colors

Options to specify output files:

`-ssdump` [<.dat>] (ssdump.dat) (Optional) Generic data file

`-o` [<.xpm>] (ss.xpm) X PixMap compatible matrix file

`-sc` [<.xvg>] (scount.xvg) xvgr/xmgr file

`-a` [<.xpm>] (area.xpm) (Optional) X PixMap compatible matrix file

`-ta` [<.xvg>] (totarea.xvg) (Optional) xvgr/xmgr file

`-aa` [<.xvg>] (averarea.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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) files

`-xvg` <enum> (xmgrace) xvg plot formatting: xmgrace, xmgr, none

`-sss` <string> (HEBT) Secondary structures for structure count

`-ver` <int> (2) DSSP major version. Syntax changed with version 2

3.11.25 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>] [-toler <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 447) *cpt* (page 435) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

-toler <real> (1e-06) [HIDDEN]Tolerance when computing the fluidicity using bisection algorithm

-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).

3.11.26 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 446)), a trajectory (*.trr* (page 447)/*.xtc* (page 448)/`tng`), an energy file (*.edr* (page 436)), a checkpoint file (*.cpt* (page 435)) or topology file (*.top* (page 445)) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 440) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 437), `.g96` or *.pdb* (page 442).

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 437) 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 442) 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 442) (.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 442) 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 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

-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 437) *g96* (page 437) *pdb* (page 442) brk ent esp

-mead [<.pqr>] (mead.pqr) (Optional) Coordinate file for MEAD

Other options:

-[no]w (no) View output *.xvg* (page 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 442) file when written. Can only be done when a topology is present

Known Issues

- For complex molecules, the periodicity removal routine may break down,
- in that case you can use *gmx trjconv* (page 237).

3.11.29 gmx eneconv

Synopsis

```
gmx eneconv [-f [<.edr> [...]]] [-o [<.edr>]] [-b <real>] [-e <real>]
            [-dt <real>] [-offset <real>] [-[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 <real> (-1) First time to use

-e <real> (-1) Last time to use

-dt <real> (0) Only write out frame when t MOD dt = offset

-offset <real> (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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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] [-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 450) 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 147) 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.

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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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`

-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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) (Optional) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

-n [<.ndx>] (index.ndx) (Optional) Index file

Options to specify output files:

-ol [<.xtc/.trr/...>] (lowpass.xtc) (Optional) Trajectory: *xtc* (page 448) *trr* (page 447) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-oh [<.xtc/.trr/...>] (highpass.xtc) (Optional) Trajectory: *xtc* (page 448) *trr* (page 447) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

-trj [<.xtc/.trr/...>] (traj.xtc) (Optional) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

Options to specify output files:

-o [<.gro/.g96/...>] (out.gro) Structure file: *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 177).

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 437) *g96* (page 437) *pdb* (page 442) 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 446) 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 442) 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 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

-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 183) (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 440) 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 440) 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 130) is more convenient than `gmx grompp`. You then supply the old checkpoint file directly to *gmx mdrun* (page 183) 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 440) file with `-f` is the recommended procedure. Actually preserving the ensemble (if possible) still requires passing the checkpoint file to *gmx mdrun* (page 183) `-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 148) program. *gmx check* (page 120) 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 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

`-r` [<.gro/.g96/...>] (restraint.gro) (Optional) Structure file: *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

`-rb` [<.gro/.g96/...>] (restraint.gro) (Optional) Structure file: *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

`-n` [<.ndx>] (index.ndx) (Optional) Index file

`-p` [<.top>] (topol.top) Topology file

`-t` [<.trr/.cpt/...>] (traj.trr) (Optional) Full precision trajectory: *trr* (page 447) *cpt* (page 435) *tng* (page 444)

`-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 447) *cpt* (page 435) *tng* (page 444)

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>]] [-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` 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.41 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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.

3.11.42 gmx hbond

Synopsis

```
gmx hbond [-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` 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.
- `-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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450) 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.

-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

Known Issues

- The option `-sel` that used to work on selected hbonds is out of order, and therefore not available for the time being.

3.11.43 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

Options to specify output files:

-cz [<.gro/.g96/...>] (zconf.gro) Structure file: *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.44 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-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.

3.11.45 gmx help

3.11.46 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.47 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>] [-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 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 150) `-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 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

-ci [<.gro/.g96/...>] (insert.gro) Configuration to insert: *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

-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 437) *g96* (page 437) *pdb* (page 442) 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

-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

3.11.48 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 436) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.49 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 132)) or from a normal modes analysis (*gmx nmeig* (page 191)). 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 450) 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 446) 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 450) 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 436) files were concatenated first. The constraints are applied in the order they appear in the *.edi* (page 436) file. Depending on what was specified in the *.edi* (page 436) 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 436) file under DELTA_F0 and EFL_NULL.

Options

Options to specify input files:

-f [<.trr/.cpt/...>] (eigenvec.trr) Full precision trajectory: *trr* (page 447) *cpt* (page 435) *tng* (page 444)

-eig [<.xvg>] (eigenval.xvg) (Optional) xvgr/xmgr file

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

-n [<.ndx>] (index.ndx) (Optional) Index file

-tar [<.gro/.g96/...>] (target.gro) (Optional) Structure file: *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

-ori [<.gro/.g96/...>] (origin.gro) (Optional) Structure file: *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

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 450) 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.

3.11.50 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 221) `-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 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

-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

3.11.51 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 254) to make a PostScript (tm) plot.

Options

Options to specify input files:

-f [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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

3.11.52 gmx mdrun

Synopsis

```
gmx mdrun [-s [<.tpr>]] [-cpi [<.cpt>]] [-table [<.xvg>]]
          [-tablep [<.xvg>]] [-tableb [<.xvg> [...]]]
          [-rerun [<.xtc/.trr/...>]] [-ei [<.edi>]]
          [-multidir [<dir> [...]]] [-awh [<.xvg>]]
          [-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>*] [**-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 191).

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 436) file. The *.edi* (page 436) 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 450) 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 440) 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 440) 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 446) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-ei [<.edi>] (sam.edi) (Optional) ED sampling input

-multidir [<dir> [...]] (rundir) (Optional) Run directory

-awh [<.xvg>] (awhinit.xvg) (Optional) xvgr/xmgr 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 447) *cpt* (page 435) *tng* (page 444)

-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 437) *g96* (page 437) *pdb* (page 442) 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, 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 PP task on each node to a device

-[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

-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) Try to avoid optimizations that affect binary reproducibility

-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

3.11.53 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 144) and *gmx pairdist* (page 199) calculate distances.

Options

Options to specify input files:

-f [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) (Optional) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 448) *trr* (page 447) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.54 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

3.11.55 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>] [-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`.

Options

Options to specify input files:

`-f` [<.xtc/.trr/...>] (traj.xtc) (Optional) Input trajectory or single configuration: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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)

-beginfit <real> (-1) Time point at which to start fitting.

-endfit <real> (-1) End time for fitting.

3.11.56 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 183). 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 110). An ensemble of structures can be generated from the eigenvectors with *gmx nmens* (page 193). 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 447) *cpt* (page 435) *tng* (page 444)

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

3.11.57 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 447) *cpt* (page 435) *tng* (page 444)

-e [<.xvg>] (eigenval.xvg) xvgr/xmgr file

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

-n [<.ndx>] (index.ndx) (Optional) Index file

Options to specify output files:

-o [<.xtc/.trr/...>] (ensemble.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

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)

3.11.58 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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`

3.11.59 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 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

-v [<.trr/.cpt/...>] (eigenvec.trr) Full precision trajectory: *trr* (page 447) *cpt* (page 435) *tng* (page 444)

Options to specify output files:

-o [<.xtc/.trr/...>] (nmtraj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

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

3.11.60 gmx nonbonded-benchmark

Synopsis

```
gmx nonbonded-benchmark [-o [<.csv>]] [-size <int>] [-nt <int>]
             [-simd <enum>] [-coulomb <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 12 relevant combinations of options. The combinations double to 24 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` 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

-[no]table (no) Use lookup table for Ewald correction instead of analytical

-combrule <enum> (geometric) The LJ combination rule: geometric, lb, none

-[no]halflj (no) Use optimization for LJ on half of the atoms

-[no]energy (no) Compute energies in addition to forces

-[no]all (no) Run all 12 combinations of options for coulomb, halflj, combrule

-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

3.11.61 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]unsat] [-[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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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]unsat (no) Calculate order parameters for unsaturated carbons. Note that this cannot be mixed with normal order parameters.

-[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

3.11.62 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 144) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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

3.11.63 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]chargegrp] [-[no]cmap] [-[no]renum] [-[no]rtpres]
```

Description

`gmx pdb2gmx` reads a *.pdb* (page 442) (or *.gro* (page 437)) file, reads some database files, adds hydrogens to the molecules and generates coordinates in GROMACS (GROMOS), or optionally *.pdb* (page 442), 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 442) 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 442) 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 442) 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 437) and `.g96` file formats do not support chain identifiers. Therefore it is useful to enter a *.pdb* (page 442) file name at the `-o` option when you want to convert a multi-chain *.pdb* (page 442) 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 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

Options to specify output files:

`-o` [<.gro/.g96/...>] (conf.gro) Structure file: *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 437) *g96* (page 437) *pdb* (page 442) 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, spc, spce, tip3p, tip4p, 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]chargegrp (yes) Use charge groups in the *.rtp* (page 443) file

-[no]cmap (yes) Use cmap torsions (if enabled in the *.rtp* (page 443) file)

-[no]renum (no) Renumber the residues consecutively in the output

-[no]rtpres (no) Use *.rtp* (page 443) entry names as residue names

3.11.64 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 446) 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

3.11.65 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.66 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]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.

Options

Options to specify input files:

-f [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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]correct (no) Assume net zero charge of groups to improve accuracy

Known Issues

- Discarding slices for integration should not be necessary.

3.11.67 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) files

-xvg <enum> (xmgrace) xvg plot formatting: xmgrace, xmgr, none

-[no]foo (no) Dummy option to avoid empty array

3.11.68 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) files

-xvg <enum> (xmgrace) xvg plot formatting: xmgrace, xmgr, none

3.11.69 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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

3.11.70 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 specfied 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 120).

Options

Options to specify input files:

-s [<.tpr/.gro/...>] (topol.tpr) Run input file for report: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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

3.11.71 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 447) 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 254).

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 446) file masses will be taken from there, otherwise the masses will be deduced from the `atommass.dat` file in `GMXLIB`. This is fine for proteins, but not necessarily for other molecules. A default mass of 12.011 amu (carbon) is assigned to unknown atoms. 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 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

`-f` [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-f2` [<.xtc/.trr/...>] (traj.xtc) (Optional) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.72 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 210). 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.73 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 442) 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 442) 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 442) 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 442) 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 442) 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 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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.

3.11.74 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.75 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.76 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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)

3.11.77 gmx sans

Synopsis

```
gmx sans [-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>]
         [-bin <real>] [-mode <enum>] [-mcover <real>]
         [-method <enum>] [-[no]pbc] [-grid <real>] [-startq <real>]
         [-endq <real>] [-qstep <real>] [-seed <int>] [-nt <int>]
```

Description

`gmx sans` computes SANS spectra using Debye formula. It currently uses topology file (since it need to assigne 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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

-bin <real> (0.2) [HIDDEN]Binwidth (nm)

-mode <enum> (direct) Mode for sans spectra calculation: direct, mc

-mcover <real> (-1) Monte-Carlo coverage should be -1(default) or (0,1]

-method <enum> (debye) [HIDDEN]Method for sans spectra calculation: debye, fft

-[no]pbc (yes) Use periodic boundary conditions for computing distances

-grid <real> (0.05) [HIDDEN]Grid spacing (in nm) for FFTs

-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> (48) Number of threads to start

3.11.78 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 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 442) 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)

3.11.79 gmx saxs

Synopsis

```
gmx saxs [-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` 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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)

3.11.80 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 234).

Options

Options to specify input files:

-f [<.xtc/.trr/...>] (traj.xtc) (Optional) Input trajectory or single configuration: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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

3.11.81 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 450) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.82 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.83 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 150) 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 150)).

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 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

`-cs` [<.gro/.g96/...>] (spc216.gro) (Library) Structure file: *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

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 437) *g96* (page 437) *pdb* (page 442) 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.

3.11.84 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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.

3.11.85 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 237) steps.

Usage:

1. Use *gmx make_ndx* (page 182) 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 237) 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 237). 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 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

`-f` [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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> (4) Number of additional bins to ensure proper memory allocation

Known Issues

- When the allocated memory is not large enough, a segmentation fault may occur.
- This is usually detected
- and the program is halted prior to the fault while displaying a warning message
- suggesting the use of the `-nab` (Number of Additional Bins)
- option. However, the program does not detect all such events. If you encounter a
- segmentation fault, run it again
- with an increased `-nab` value.

3.11.86 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.87 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 447) *cpt* (page 435) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.88 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 442) 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 450) 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 234) for plotting similar data for selections.

Options

Options to specify input files:

`-f` [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 442) output, 0 is autoscale

3.11.89 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional) Input structure: *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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

3.11.90 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 450) file. The *.xvg* (page 450) 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 450) file then the program tries to be smart. Beware.

Options

Options to specify input files:

-f [<.xtc/.trr/...> [...]] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 448) *trr* (page 447) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

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

3.11.91 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 450) 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 236) is better suited for concatenating multiple trajectory files.

The following formats are supported for input and output: *.xtc* (page 448), *.trr* (page 447), *.gro* (page 437), `.g96` and *.pdb* (page 442). The file formats are detected from the file extension. The precision of the *.xtc* (page 448) output is taken from the input file for *.xtc* (page 448), *.gro* (page 437) and *.pdb* (page 442), 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 447) 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 447), *.gro* (page 437) and `.g96` files.

Option `-sep` can be used to write every frame to a separate `.gro, .g96` or *.pdb* (page 442) file. By default, all frames all written to one file. *.pdb* (page 442) 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 448) format for your analysis to save disk space and to have portable files.

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 159) 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 447) 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, but only works reliably if the time interval between frames is uniform.

Option `-drop` reads an *.xvg* (page 450) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 448) *trr* (page 447) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 442) files. Useful for visualization of non-standard molecules, e.g. coarse grained ones

3.11.92 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 442) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) brk ent

-n [<.ndx>] (index.ndx) (Optional) Index file

Options to specify output files:

-o [<.xtc/.trr/...>] (ordered.xtc) (Optional) Trajectory: *xtc* (page 448) *trr* (page 447) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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

3.11.93 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 183) 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 446) file to `gmx tune_pme` together with other options for *gmx mdrun* (page 183) as needed.

`gmx tune_pme` needs to call *gmx mdrun* (page 183) 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 183) 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 183) 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 446) 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 446) 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 446) 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 446) 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 183) 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-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 447) *cpt* (page 435) *tng* (page 444)

-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 437) *g96* (page 437) *pdb* (page 442) 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 447) *cpt* (page 435) *tng* (page 444)

-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 437) *g96* (page 437) *pdb* (page 442) 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 446) 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 446) 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 446), add *.cpt* (page 435) steps)

-[no]launch (no) Launch the real simulation after optimization

-[no]bench (yes) Run the benchmarks or just create the input *.tpr* (page 446) 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)

3.11.94 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 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr/.gro/...>] (topol.tpr) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.95 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 440) 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 447) *cpt* (page 435) *tng* (page 444)

`-s` [<.tpr/.gro/...>] (topol.tpr) (Optional) Structure+mass(db): *tpr* (page 446) *gro* (page 437) *g96* (page 437) *pdb* (page 442) 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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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

3.11.96 gmx view

Synopsis

```
gmx view [-f [<.xtc/.trr/...>]] [-s [<.tpr>]] [-n [<.ndx>]] [-b <time>]
         [-e <time>] [-dt <time>]
```

Description

`gmx view` is the GROMACS trajectory viewer. This program reads a trajectory file, a run input file and an index file and plots a 3D structure of your molecule on your standard X Window screen. No need for a high end graphics workstation, it even works on Monochrome screens.

The following features have been implemented: 3D view, rotation, translation and scaling of your molecule(s), labels on atoms, animation of trajectories, hardcopy in PostScript format, user defined atom-filters runs on MIT-X (real X), open windows and motif, user friendly menus, option to remove periodicity, option to show computational box.

Some of the more common X command line options can be used: `-bg`, `-fg` change colors, `-font fontname` changes the font.

Options

Options to specify input files:

-f [<.xtc/.trr/...>] (traj.xtc) Trajectory: *xtc* (page 448) *trr* (page 447) *cpt* (page 435) *gro* (page 437) *g96* (page 437) *pdb* (page 442) *tng* (page 444)

-s [<.tpr>] (topol.tpr) Portable xdr run input file

-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)

Known Issues

- Balls option does not work
- Some times dumps core without a good reason

3.11.97 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 446) files), AND, with option `-ix`, a file which contains file names of the pullx `mdrun` output files. The *.tpr* (page 446) and pullx files must be in corresponding order, i.e. the first *.tpr* (page 446) 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 446) 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 112) 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.

3.11.98 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 436) 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

3.11.99 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 443) entry, which you can then add to the *.rtp* (page 443) 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 437) *g96* (page 437) *pdb* (page 442) brk ent esp *tpr* (page 446)

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 442) 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.

3.11.100 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 do_dssp* (page 145), *gmx rms* (page 210) or *gmx mdmat* (page 183).

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 450), *.xpm* (page 447), *.eps* (page 437) and *.pdb* (page 442) 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 447) matrix where axis label is zero

-legoffset <int> (0) Skip first N colors from *.xpm* (page 447) 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

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've installed an MPI version of GROMACS, by default the **gmx** binary is called **gmx_mpi** and you should adapt accordingly.

3.11.101 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 104) 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 264).

3.11.102 Commands by name

- *gmx* (page 104) - molecular dynamics simulation suite
- *gmx anaeig* (page 110) - Analyze eigenvectors/normal modes
- *gmx analyze* (page 112) - Analyze data sets
- *gmx angle* (page 115) - Calculate distributions and correlations for angles and dihedrals
- *gmx awh* (page 116) - Extract data from an accelerated weight histogram (AWH) run
- *gmx bar* (page 117) - Calculate free energy difference estimates through Bennett's acceptance ratio
- *gmx bundle* (page 119) - Analyze bundles of axes, e.g., helices
- *gmx check* (page 120) - Check and compare files
- *gmx chi* (page 122) - Calculate everything you want to know about chi and other dihedrals
- *gmx cluster* (page 124) - Cluster structures
- *gmx clustsize* (page 127) - Calculate size distributions of atomic clusters
- *gmx confrms* (page 129) - Fit two structures and calculates the RMSD
- *gmx convert-tpr* (page 130) - Make a modifed run-input file
- *gmx convert-trj* (page 130) - Converts between different trajectory types
- *gmx covar* (page 132) - Calculate and diagonalize the covariance matrix
- *gmx current* (page 133) - Calculate dielectric constants and current autocorrelation function
- *gmx density* (page 135) - Calculate the density of the system
- *gmx densmap* (page 136) - Calculate 2D planar or axial-radial density maps
- *gmx densorder* (page 138) - Calculate surface fluctuations
- *gmx dielectric* (page 139) - Calculate frequency dependent dielectric constants
- *gmx dipoles* (page 140) - Compute the total dipole plus fluctuations
- *gmx disre* (page 143) - Analyze distance restraints
- *gmx distance* (page 144) - Calculate distances between pairs of positions
- *gmx do_dssp* (page 145) - Assign secondary structure and calculate solvent accessible surface area
- *gmx dos* (page 147) - Analyze density of states and properties based on that
- *gmx dump* (page 148) - Make binary files human readable
- *gmx dyecoupl* (page 149) - Extract dye dynamics from trajectories
- *gmx editconf* (page 150) - Convert and manipulates structure files
- *gmx eneconv* (page 152) - Convert energy files
- *gmx enemat* (page 153) - Extract an energy matrix from an energy file
- *gmx energy* (page 155) - Writes energies to xvg files and display averages
- *gmx extract-cluster* (page 157) - Allows extracting frames corresponding to clusters from trajectory
- *gmx filter* (page 159) - Frequency filter trajectories, useful for making smooth movies
- *gmx freevolume* (page 160) - Calculate free volume
- *gmx gangle* (page 161) - Calculate angles
- *gmx genconf* (page 163) - Multiply a conformation in 'random' orientations

- *gmx genion* (page 164) - Generate monoatomic ions on energetically favorable positions
- *gmx genrestr* (page 165) - Generate position restraints or distance restraints for index groups
- *gmx grompp* (page 166) - Make a run input file
- *gmx gyrate* (page 168) - Calculate the radius of gyration
- *gmx h2order* (page 169) - Compute the orientation of water molecules
- *gmx hbond* (page 170) - Compute and analyze hydrogen bonds
- *gmx helix* (page 173) - Calculate basic properties of alpha helices
- *gmx helixorient* (page 174) - Calculate local pitch/bending/rotation/orientation inside helices
- *gmx help* (page 176) - Print help information
- *gmx hydorder* (page 176) - Compute tetrahedrality parameters around a given atom
- *gmx insert-molecules* (page 177) - Insert molecules into existing vacancies
- *gmx lie* (page 178) - Estimate free energy from linear combinations
- *gmx make_edi* (page 179) - Generate input files for essential dynamics sampling
- *gmx make_ndx* (page 182) - Make index files
- *gmx mdmat* (page 183) - Calculate residue contact maps
- *gmx mdrun* (page 183) - Perform a simulation, do a normal mode analysis or an energy minimization
- *gmx mindist* (page 188) - Calculate the minimum distance between two groups
- *gmx mk_angndx* (page 189) - Generate index files for 'gmx angle'
- *gmx msd* (page 190) - Compute mean squared displacements
- *gmx nmeig* (page 191) - Diagonalize the Hessian for normal mode analysis
- *gmx nmens* (page 193) - Generate an ensemble of structures from the normal modes
- *gmx nmr* (page 193) - Analyze nuclear magnetic resonance properties from an energy file
- *gmx nmtraj* (page 195) - Generate a virtual oscillating trajectory from an eigenvector
- *gmx nonbonded-benchmark* (page 196) - Benchmarking tool for the non-bonded pair kernels.
- *gmx order* (page 197) - Compute the order parameter per atom for carbon tails
- *gmx pairdist* (page 199) - Calculate pairwise distances between groups of positions
- *gmx pdb2gmx* (page 200) - Convert coordinate files to topology and FF-compliant coordinate files
- *gmx pme_error* (page 203) - Estimate the error of using PME with a given input file
- *gmx polystat* (page 204) - Calculate static properties of polymers
- *gmx potential* (page 205) - Calculate the electrostatic potential across the box
- *gmx principal* (page 206) - Calculate principal axes of inertia for a group of atoms
- *gmx rama* (page 207) - Compute Ramachandran plots
- *gmx rdf* (page 207) - Calculate radial distribution functions
- *gmx report-methods* (page 209) - Write short summary about the simulation setup to a text file and/or to the standard output.
- *gmx rms* (page 210) - Calculate RMSDs with a reference structure and RMSD matrices
- *gmx rmsdist* (page 211) - Calculate atom pair distances averaged with power -2, -3 or -6
- *gmx rmsf* (page 213) - Calculate atomic fluctuations

- *gmx rotacf* (page 214) - Calculate the rotational correlation function for molecules
- *gmx rotmat* (page 215) - Plot the rotation matrix for fitting to a reference structure
- *gmx saltbr* (page 216) - Compute salt bridges
- *gmx sans* (page 217) - Compute small angle neutron scattering spectra
- *gmx sasa* (page 218) - Compute solvent accessible surface area
- *gmx saxs* (page 220) - Compute small angle X-ray scattering spectra
- *gmx select* (page 221) - Print general information about selections
- *gmx sham* (page 223) - Compute free energies or other histograms from histograms
- *gmx sigeps* (page 225) - Convert c6/12 or c6/cn combinations to and from sigma/epsilon
- *gmx solvate* (page 225) - Solvate a system
- *gmx sorient* (page 227) - Analyze solvent orientation around solutes
- *gmx spatial* (page 228) - Calculate the spatial distribution function
- *gmx spol* (page 230) - Analyze solvent dipole orientation and polarization around solutes
- *gmx tcaf* (page 231) - Calculate viscosities of liquids
- *gmx traj* (page 233) - Plot x, v, f, box, temperature and rotational energy from trajectories
- *gmx trajectory* (page 234) - Print coordinates, velocities, and/or forces for selections
- *gmx trjcat* (page 236) - Concatenate trajectory files
- *gmx trjconv* (page 237) - Convert and manipulates trajectory files
- *gmx trjorder* (page 240) - Order molecules according to their distance to a group
- *gmx tune_pme* (page 241) - Time mdrun as a function of PME ranks to optimize settings
- *gmx vanhove* (page 245) - Compute Van Hove displacement and correlation functions
- *gmx velacc* (page 246) - Calculate velocity autocorrelation functions
- *gmx view* (page 248) - View a trajectory on an X-Windows terminal
- *gmx wham* (page 248) - Perform weighted histogram analysis after umbrella sampling
- *gmx wheel* (page 252) - Plot helical wheels
- *gmx x2top* (page 253) - Generate a primitive topology from coordinates
- *gmx xpm2ps* (page 254) - Convert XPM (XPixelMap) matrices to postscript or XPM

3.11.103 Commands by topic

Trajectory analysis

gmx gangle **(page 161)** Calculate angles

gmx convert-trj **(page 130)** Converts between different trajectory types

gmx distance **(page 144)** Calculate distances between pairs of positions

gmx extract-cluster **(page 157)** Allows extracting frames corresponding to clusters from trajectory

gmx freevolume **(page 160)** Calculate free volume

gmx msd **(page 190)** Compute mean squared displacements

gmx pairdist **(page 199)** Calculate pairwise distances between groups of positions

gmx rdf **(page 207)** Calculate radial distribution functions

gmx sasa **(page 218)** Compute solvent accessible surface area

gmx select **(page 221)** Print general information about selections

gmx trajectory **(page 234)** Print coordinates, velocities, and/or forces for selections

Generating topologies and coordinates

gmx editconf **(page 150)** Edit the box and write subgroups

gmx x2top **(page 253)** Generate a primitive topology from coordinates

gmx solvate **(page 225)** Solvate a system

gmx insert-molecules **(page 177)** Insert molecules into existing vacancies

gmx genconf **(page 163)** Multiply a conformation in 'random' orientations

gmx genion **(page 164)** Generate monoatomic ions on energetically favorable positions

gmx genrestr **(page 165)** Generate position restraints or distance restraints for index groups

gmx pdb2gmx **(page 200)** Convert coordinate files to topology and FF-compliant coordinate files

Running a simulation

gmx grompp **(page 166)** Make a run input file

gmx mdrun **(page 183)** Perform a simulation, do a normal mode analysis or an energy minimization

gmx convert-tpr **(page 130)** Make a modifed run-input file

Viewing trajectories

gmx nmtraj **(page 195)** Generate a virtual oscillating trajectory from an eigenvector

gmx view **(page 248)** View a trajectory on an X-Windows terminal

Processing energies

gmx enemat **(page 153)** Extract an energy matrix from an energy file

gmx energy **(page 155)** Writes energies to xvg files and display averages

gmx mdrun **(page 183)** (Re)calculate energies for trajectory frames with -rerun

Converting files

gmx editconf **(page 150)** Convert and manipulates structure files

gmx eneconv **(page 152)** Convert energy files

gmx sigeps **(page 225)** Convert c6/12 or c6/cn combinations to and from sigma/epsilon

gmx trjcat **(page 236)** Concatenate trajectory files

gmx trjconv **(page 237)** Convert and manipulates trajectory files

gmx xpm2ps **(page 254)** Convert XPM (XPixelMap) matrices to postscript or XPM

Tools

gmx analyze **(page 112)** Analyze data sets

gmx awh **(page 116)** Extract data from an accelerated weight histogram (AWH) run

gmx filter **(page 159)** Frequency filter trajectories, useful for making smooth movies

gmx lie **(page 178)** Estimate free energy from linear combinations

gmx pme_error **(page 203)** Estimate the error of using PME with a given input file

gmx sham **(page 223)** Compute free energies or other histograms from histograms

gmx spatial **(page 228)** Calculate the spatial distribution function

gmx traj **(page 233)** Plot x, v, f, box, temperature and rotational energy from trajectories

gmx tune_pme **(page 241)** Time mdrun as a function of PME ranks to optimize settings

gmx wham **(page 248)** Perform weighted histogram analysis after umbrella sampling

gmx check **(page 120)** Check and compare files

gmx dump **(page 148)** Make binary files human readable

gmx make_ndx **(page 182)** Make index files

gmx mk_angndx **(page 189)** Generate index files for 'gmx angle'

gmx trjorder **(page 240)** Order molecules according to their distance to a group

gmx xpm2ps **(page 254)** Convert XPM (XPixelMap) matrices to postscript or XPM

gmx report-methods **(page 209)** Write short summary about the simulation setup to a text file and/or to the standard output.

Distances between structures

gmx cluster **(page 124)** Cluster structures

gmx confrms **(page 129)** Fit two structures and calculates the RMSD

gmx rms **(page 210)** Calculate RMSDs with a reference structure and RMSD matrices

gmx rmsf **(page 213)** Calculate atomic fluctuations

Distances in structures over time

gmx mindist **(page 188)** Calculate the minimum distance between two groups

gmx mdmat **(page 183)** Calculate residue contact maps

gmx polystat **(page 204)** Calculate static properties of polymers

gmx rmsdist **(page 211)** Calculate atom pair distances averaged with power -2, -3 or -6

Mass distribution properties over time

gmx gyrate **(page 168)** Calculate the radius of gyration

gmx polystat **(page 204)** Calculate static properties of polymers

gmx rdf **(page 207)** Calculate radial distribution functions

gmx rotacf **(page 214)** Calculate the rotational correlation function for molecules

gmx rotmat **(page 215)** Plot the rotation matrix for fitting to a reference structure

gmx sans **(page 217)** Compute small angle neutron scattering spectra

gmx saxs **(page 220)** Compute small angle X-ray scattering spectra

gmx traj **(page 233)** Plot x, v, f, box, temperature and rotational energy from trajectories

gmx vanhove **(page 245)** Compute Van Hove displacement and correlation functions

Analyzing bonded interactions

gmx angle **(page 115)** Calculate distributions and correlations for angles and dihedrals

gmx mk_angndx **(page 189)** Generate index files for 'gmx angle'

Structural properties

gmx bundle **(page 119)** Analyze bundles of axes, e.g., helices

gmx clustsize **(page 127)** Calculate size distributions of atomic clusters

gmx disre **(page 143)** Analyze distance restraints

gmx hbond **(page 170)** Compute and analyze hydrogen bonds

gmx order **(page 197)** Compute the order parameter per atom for carbon tails

gmx principal **(page 206)** Calculate principal axes of inertia for a group of atoms

gmx rdf **(page 207)** Calculate radial distribution functions

gmx saltbr **(page 216)** Compute salt bridges

gmx sorient **(page 227)** Analyze solvent orientation around solutes

gmx spol **(page 230)** Analyze solvent dipole orientation and polarization around solutes

Kinetic properties

gmx bar **(page 117)** Calculate free energy difference estimates through Bennett's acceptance ratio

gmx current **(page 133)** Calculate dielectric constants and current autocorrelation function

gmx dos **(page 147)** Analyze density of states and properties based on that

gmx dyecoupl **(page 149)** Extract dye dynamics from trajectories

gmx principal **(page 206)** Calculate principal axes of inertia for a group of atoms

gmx tcaf **(page 231)** Calculate viscosities of liquids

gmx traj **(page 233)** Plot x, v, f, box, temperature and rotational energy from trajectories

gmx vanhove **(page 245)** Compute Van Hove displacement and correlation functions

gmx velacc **(page 246)** Calculate velocity autocorrelation functions

Electrostatic properties

gmx current **(page 133)** Calculate dielectric constants and current autocorrelation function

gmx dielectric **(page 139)** Calculate frequency dependent dielectric constants

gmx dipoles **(page 140)** Compute the total dipole plus fluctuations

gmx potential **(page 205)** Calculate the electrostatic potential across the box

gmx spol **(page 230)** Analyze solvent dipole orientation and polarization around solutes

gmx genion **(page 164)** Generate monoatomic ions on energetically favorable positions

Protein-specific analysis

gmx do_dssp **(page 145)** Assign secondary structure and calculate solvent accessible surface area

gmx chi **(page 122)** Calculate everything you want to know about chi and other dihedrals

gmx helix **(page 173)** Calculate basic properties of alpha helices

gmx helixorient **(page 174)** Calculate local pitch/bending/rotation/orientation inside helices

gmx rama **(page 207)** Compute Ramachandran plots

gmx wheel **(page 252)** Plot helical wheels

Interfaces

gmx bundle **(page 119)** Analyze bundles of axes, e.g., helices

gmx density **(page 135)** Calculate the density of the system

gmx densmap **(page 136)** Calculate 2D planar or axial-radial density maps

gmx densorder **(page 138)** Calculate surface fluctuations

gmx h2order **(page 169)** Compute the orientation of water molecules

gmx hydorder **(page 176)** Compute tetrahedrality parameters around a given atom

gmx order **(page 197)** Compute the order parameter per atom for carbon tails

gmx potential **(page 205)** Calculate the electrostatic potential across the box

Covariance analysis

gmx anaeig **(page 110)** Analyze the eigenvectors

gmx covar **(page 132)** Calculate and diagonalize the covariance matrix

gmx make_edi **(page 179)** Generate input files for essential dynamics sampling

Normal modes

gmx anaeig **(page 110)** Analyze the normal modes

gmx nmeig **(page 191)** Diagonalize the Hessian for normal mode analysis

gmx nmtraj **(page 195)** Generate a virtual oscillating trajectory from an eigenvector

gmx nmens **(page 193)** Generate an ensemble of structures from the normal modes

gmx grompp **(page 166)** Make a run input file

gmx mdrun **(page 183)** Find a potential energy minimum and calculate the Hessian

3.11.104 Special topics

The information in these topics is also accessible through `gmx help` *`topic`* on the command line.

Selection syntax and usage

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 promted 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)
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;
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.

3.11.105 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 264) 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 **??**).

Version 2020

gmx convert-trj

new

gmx convert-trj (page 130) has been introduced as a selection-enabled alternative for exchanging trajectory file format (previously done in *gmx trjconv* (page 237)).

gmx extract-cluster

new

gmx extract-cluster (page 157) 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 237) has been removed.

Version 2018

gmx trajectory

new

gmx trajectory (page 234) has been introduced as a selection-enabled version of *gmx traj* (page 233). 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 177) 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 199) has been introduced as a selection-enabled replacement for *gmx mindist* (page 188) (`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 207) 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 144).

You can provide your existing index file to *gmx distance* (page 144), 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 144) (for most options) or *gmx select* (page 221) (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 144) 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 161) 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 115) 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 200). 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 218) (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 161) (for angle calculation) and *gmx distance* (page 144) (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 144) 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 225) and *gmx insert-molecules* (page 177).

tpbconv

This tool has been renamed *gmx convert-tpr* (page 130).

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 occuring, 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 that the molecule(s) move from where they were initially located within the box is (for the vast majority of simulations) they are 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 237) 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 237) follows the same lines. Some complex cases needing more than one operation will require more than one invocation of *gmx trjconv* (page 237) in order to work.

For further information, see the corresponding section in the *Reference Manual* (page 311).

Suggested workflow

Fixing periodicity effects with *gmx trjconv* (page 237) 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's 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 don't 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 don't use any PBC related option afterwards.

With point three, the issue is that *gmx trjconv* (page 237) 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's 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's 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'd 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's 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 326) 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 279)). 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 279).

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 278) and *pressure coupling* (page 277).
- 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's important to remember that an average structure isn't necessarily meaningful. By way of analogy, suppose I alternate holding a ball in my left hand, then in my right hand. What's 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 doesn't 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's 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 can't 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're the first thing to crash. Similarly, in simulations with domain decomposition, you may see messages about particles being more than a cell length out of the domain decomposition cell of their charge group, which are symptomatic of your underlying problem, and not the domain decomposition algorithm itself. 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 didn't 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 best 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 183).

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 don't get that large, or
- use a smaller timestep.

Better system preparation is a way to make sure that forces don't 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 haven't equilibrated thoroughly first
3. Monitor various components of the system's energy using *gmx energy* (page 155). If an intramolecular term is spiking, that may indicate improper bonded parameters, for example.
4. Make sure you haven't been ignoring error messages (missing atoms when running *gmx pdb2gmx* (page 200), mismatching names when running *gmx grompp* (page 166), etc.) or using work-arounds (like using `gmx grompp -maxwarn` when you shouldn't be) to make sure your topology is intact and being interpreted correctly.
5. Make sure you are using appropriate settings in your *mdp* (page 440) 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 200) (see below), or maybe *gmx genion* (page 164) 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 200)? If so, don't. Reconstruct missing coordinates rather than ignoring them.
2. Did you override long/short bond warnings by changing the lengths? If so, don't. 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 315) 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 a 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 278) for more on *constant* temperature).
- NPT - number of particles (N), system pressure (P) and temperature (T) are constant / conserved. (See *pressure coupling* (page 277) 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 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.

3.13.1 Output Control

GMX_DUMP_NL Neighbour list dump level; default 0.

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_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 442) file that have an interaction energy less than the value set in this environment variable.

GMX_VIEW_XPM `GMX_VIEW_XVG`, `GMX_VIEW_EPS` and `GMX_VIEW_PDB`, commands used to automatically view *xvg* (page 450), *xpm* (page 447), *eps* (page 437) and *pdb* (page 442) file types, respectively; they default to `xv`, `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.

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_LOGO_COLOR set display color for logo in *gmx view* (page 248).

GMX_PRINT_LONGFORMAT use long float format when printing decimal values.

GMX_COMPELDUMP Applies for computational electrophysiology setups only (see reference manual). The initial structure gets dumped to *pdb* (page 442) file, which allows to check whether multimeric channels have the correct PBC representation.

GMX_TRAJECTORY_IO_VERBOSITY Defaults to 1, which prints frame count e.g. when reading trajectory files. Set to 0 for quiet operation.

GMX_ENABLE_GPU_TIMING Enables GPU timings in the log file for CUDA. 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_DISABLE_GPU_TIMING Disables GPU timings in the log file for OpenCL.

3.13.2 Debugging

GMX_PRINT_DEBUG_LINES when set, print debugging info on line numbers.

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 40).

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 40).

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_NPULSE over-ride the number of DD pulses used (default 0, meaning no over-ride). Normally 1 or 2.

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!

3.13.3 Performance and Run Control

GMX_DO_GALACTIC_DYNAMICS planetary simulations are made possible (just for fun) by setting this environment variable, which allows setting *epsilon-r* (page 43) to -1 in the *mdp* (page 440) file. Normally, *epsilon-r* (page 43) must be greater than zero to prevent a fatal error. See webpage for example input files for a planetary simulation.

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_GPU_NB_EWALD_TWINCUT force the use of twin-range cutoff kernel even if *rvdw* (page 44) equals *rcoulomb* (page 43) 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_DISABLE_CUDA_TIMING Deprecated. Use `GMX_DISABLE_GPU_TIMING` instead.

GMX_GPU_DD_COMMS perform domain decomposition halo exchange communication operations (on coordinate and force buffers) directly on GPU memory spaces, without the staging of data through CPU memory, where possible.

GMX_GPU_PME_PP_COMMS when the simulation uses a separate PME rank, perform communication operations between PP and PME rank (for coordinate and force buffers) directly on GPU memory spaces, without the staging of data through CPU memory, where possible.

GMX_GPU_SYCL_NO_SYNCHRONIZE disable synchronizations between different GPU streams in SYCL build, instead relying on SYCL runtime to do scheduling based on data dependencies. 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_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_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_DD_RECORD_LOAD record DD load statistics for reporting at end of the run (default 1, meaning on)

GMX_DETAILED_PERF_STATS when set, print slightly more detailed performance information to the *log* (page 439) 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 439) files or standard output.

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_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.

GMX_DISABLE_GPU_DETECTION when set, disables GPU detection even if *gmx mdrun* (page 183) was compiled with GPU support.

GMX_GPU_APPLICATION_CLOCKS setting this variable to a value of "0", "ON", or "DISABLE" (case insensitive) allows disabling the CUDA GPU allication clock support.

GMX_DISRE_ENSEMBLE_SIZE the number of systems for distance restraint ensemble averaging. Takes an integer value.

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_ENX_NO_FATAL disable exiting upon encountering a corrupted frame in an *edr* (page 436) file, allowing the use of all frames up until the corruption.

GMX_FORCE_UPDATE update forces when invoking `mdrun -rerun`.

GMX_FORCE_UPDATE_DEFAULT_GPU Force update to run on the GPU by default, overriding the `mdrun -update auto` option. Works similar to setting `mdrun -update gpu`, but (1) falls back to the CPU code-path, if set with input that is not supported and (2) can be used to run update on GPUs in multi-rank cases. The latter case should be considered experimental since it lacks substantial testing. Also, GPU update is only supported with the GPU direct communications and `GMX_FORCE_UPDATE_DEFAULT_GPU` variable should be set simultaneously with

GMX_GPU_DD_COMMS and GMX_GPU_PME_PP_COMMS environment variables in multi-rank case. Does not override `mdrun -update cpu`.

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_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_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_IGNORE_FSYNC_FAILURE_ENV allow *gmx mdrun* (page 183) 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 183) 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_NBLISTCG use neighbor list and kernels based on charge groups.

GMX_NBNXN_CYCLE when set, print detailed neighbor search cycle counting.

GMX_NBNXN_EWALD_ANALYTICAL force the use of analytical Ewald non-bonded kernels, mutually exclusive of GMX_NBNXN_EWALD_TABLE.

GMX_NBNXN_EWALD_TABLE force the use of tabulated Ewald non-bonded kernels, mutually exclusive of GMX_NBNXN_EWALD_ANALYTICAL.

GMX_NBNXN_SIMD_2XNN force the use of 2x(N+N) SIMD CPU non-bonded kernels, mutually exclusive of GMX_NBNXN_SIMD_4XN.

GMX_NBNXN_SIMD_4XN force the use of 4xN SIMD CPU non-bonded kernels, mutually exclusive of GMX_NBNXN_SIMD_2XNN.

GMX_NOOPTIMIZEDKERNELS deprecated, use GMX_DISABLE_SIMD_KERNELS instead.

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_LJ_COMB_RULE force the use of LJ paremeter lookup instead of using combination rules in the non-bonded kernels.

GMX_NO_INT, GMX_NO_TERM, GMX_NO_USR1 disable signal handlers for SIGINT, SIGTERM, and SIGUSR1, respectively.

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_PULL_PARTICIPATE_ALL disable the default heuristic for when to use a separate pull MPI communicator (at >=32 ranks).

GMX_NOPREDICT shell positions are not predicted.

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_NSCELL_NCG the ideal number of charge groups per neighbor searching grid cell is hard-coded to a value of 10. Setting this environment variable to any other integer value overrides this hard-coded value.

GMX_PME_NUM_THREADS set the number of OpenMP or PME threads; overrides the default set by *gmx mdrun* (page 183); 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_REQUIRE_SHELL_INIT require that shell positions are initiated.

GMX_REQUIRE_TABLES require the use of tabulated Coulombic and van der Waals interactions.

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_USE_GRAPH use graph for bonded interactions.

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.

MPIRUN the `mpirun` command used by *gmx tune_pme* (page 241).

MDRUN the *gmx mdrun* (page 183) command used by *gmx tune_pme* (page 241).

GMX_DISABLE_DYNAMICPRUNING disables dynamic pair-list pruning. Note that *gmx mdrun* (page 183) will still tune nstlist to the optimal value picked assuming dynamic pruning. Thus for good performance the -nstlist option should be used.

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 40) – `1`.

3.13.4 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_NOGENCACHE If set, disable caching for OpenCL kernel builds. Caching is normally useful so that future runs can re-use the compiled kernels from previous runs. Currently, caching is always disabled, until we solve concurrency issues.

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_DISABLE_FASTMATH Prevents the use of `-cl-fast-relaxed-math` 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_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.

`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 (see `GMX_OCL_NOGENCACHE`).

- 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_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_NOOPT Disable optimisations. Adds the option `cl-opt-disable` to the compiler options.

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_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_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_SHOW_DIAGNOSTICS Use Intel OpenCL extension to show additional runtime performance diagnostics.

3.13.5 Analysis and Core Functions

DSSP used by *gmx do_dssp* (page 145) to point to the `dssp` executable (not just its path).

GMX_DIPOLE_SPACING spacing used by *gmx dipoles* (page 140).

GMX_MAXRESRENUM sets the maximum number of residues to be renumbered by *gmx grompp* (page 166). 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 443) entries that are normally renamed. Setting this environment variable disables this renaming.

GMX_FONT name of X11 font used by *gmx view* (page 248).

GMXTIMEUNIT the time unit used in output files, can be anything in fs, ps, ns, us, ms, s, m or h.

MULTIPROT name of the `multiprot` executable, used by the contributed program `do_multiprot`.

NCPUS number of CPUs to be used for Gaussian QM calculation

GMX_TOTAL name of the `total` executable used by the contributed `do_shift` program.

GMX_ENER_VERBOSE make *gmx energy* (page 155) and *gmx eneconv* (page 152) loud and noisy.

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.

GMX_USE_XMGR sets viewer to `xmgr` (deprecated) instead of `xmgrace`.

3.14 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.

3.15 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 won't 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 don't 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.

3.16 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* (page **??**) of deprecated functionality.

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 282) 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 301) - very detailed document that can also act as a very good introduction for *MD* (page 282) in general.
- *Flow Chart* (page 22)- 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 200), or modify an existing one, there are several files you will need to modify. You must consult the *Reference Manual* (page 301) for description of the required format. Follow these steps:

1. Add the residue to the *rtp* (page 443) 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 443) format.
2. If you need hydrogens to be able to be added to your residue, create an entry in the relevant *hdb* (page 438) 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 439) file containing a `[moleculetype]` (for example, by modifying the *top* (page 445) produced by some parameterization server), and inserting an `#include` of that *itp* (page 439) file into a *top* (page 445) 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 200) 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 200) `-ff` option the local version will override the system version.

4.3 Water solvation

When using *solvate* (page 225) to generate a box of solvent, you need to supply a pre-equilibrated box of a suitable solvent for *solvate* (page 225) 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 445) file, either with the `-water` flag to *pdb2gmx* (page 200), or by editing your *top* (page 445) file appropriately by hand.

For information about how to use solvents other than pure water, please see *Non-Water Solvation* (page 292) or *Mixed Solvents* (page 293).

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 225) 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 150) to place a box of that size around your single molecule. Then use *editconf* (page 150) to move the molecule a little bit off center. Then use *genconf* (page 163) `-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 225) `-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 177) 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 225) or *gmx insert-molecules* (page 177).
- Edit your *topology* (page 445) to `#include` the appropriate *itp* (page 439) 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 200). You may find *pdb2gmx* (page 200) `-ss yes` is useful. The sulfur atoms will need to be in the same unit that *pdb2gmx* (page 200) is converting to a `moleculetype`, so invoking *pdb2gmx* (page 200) `-chainsep`' correctly may be required. See *pdb2gmx* (page 200) `-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 200).

Otherwise, editing your *top* (page 445) 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 225) 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 225) 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 225) 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 437) files and to account for any changes in the *topology* (page 445)), or
- use a script someone wrote to remove them.

4.7.3 External material

- Membrane simulations slides , membrane simulations video - (Erik Lindahl).
- GROMACS 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 283) 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 26).

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* (page 356) 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 users emailing list, 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 248) 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 183) `-rerun` mechanism, which applies the model physics in the *tpr* (page 446) 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 446) file with *grompp* (page 166). The configuration you supplied to *grompp* (page 166) 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 183) 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.

- Be absolutely sure that the "terminal" carbon atoms are sharing a bond in the topology file.
- Use `periodic_molecules = yes` in your *mdp* (page 440) file for input in *gmx grompp* (page 166).
- 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 253) 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 253). Here, *x2top* (page 253) will recognize that the terminal C atoms actually share a chemical bond. Thus, when you use *grompp* (page 166) 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://www.webcitation.org/66u2xJJ3O) 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 439)
  - atomnames2types.n2t from the file in section *n2t* (page 442)
  - aminoacids.rtp from the file in section *rtp* (page 443)
- generate a topology with the custom forcefield (the cnt_oplsaa.ff directory must be in the same directory as where the *gmx x2top* (page 253) command is run or it must be found on the GMXLIB path), `-noparam` instructs *gmx x2top* (page 253) 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 253) 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 440) and md.*mdp* (page 440)):

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␣
↪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 447) & *xtc* (page 448) 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.

Also if appropriate libraries were found at configure-time, *gmx view* (page 248) can useful.

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 (except when you give *gmx view* (page 248) a *tpr* (page 446) file). Thus it's not using your topology which is described in either your *top* (page 445) file or your *tpr* (page 446) 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 447), *xtc* (page 448), *tng* (page 444)) files.

- use the GROMACS trajectory analysis utilities
- use *gmx traj* (page 233) to write a *xvg* (page 450) 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 148) 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 450) 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 450) 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 450) 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.
- 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 228) 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 277) 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 237) `-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 166) to make a new *tpr* (page 446) file based on the frame that was output from the step above.
- use *trjconv* (page 237) `-pbc nojump` to produce the desired trajectory using the newly produced *tpr* (page 446) 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 22).

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 568) and continue right away with coding for GROMACS.

5.1 Preface and Disclaimer

GROMACS - 2022-dev

Contributions from:

Emile Apol, Rossen Apostolov, Paul Bauer, Herman J.C. Berendsen, Pär Bjelkmar, Christian Blau, Viacheslav Bolnykh, Kevin Boyd, Aldert van Buuren, Rudi van Drunen, Anton Feenstra, Gerrit Groenhof, Anca Hamuraru, Vincent Hindriksen, M. Eric Irrgang, Aleksei Iupinov, Christoph Junghans, Joe Jordan, Dimitrios Karkoulis, Peter Kasson, Jiri Kraus, Carsten Kutzner, Per Larsson, Justin A. Lemkul, Viveca Lindahl, Magnus Lundborg, Erik Marklund,Pascal Merz, Pieter Meulenhoff, Teemu Murtola, Szilárd Páll, Sander Pronk, Roland Schulz, Michael Shirts, Alexey Shvetsov, Alfons Sijbers, Peter Tieleman, Teemu Virolainen, Christian Wennberg, Maarten Wolf, and Artem Zhmurov.

Mark Abraham, Berk Hess, David van der Spoel, and Erik Lindahl.

© 1991 – 2000:

> Department of Biophysical Chemistry, University of Groningen. Nijenborgh 4, 9747 AG Groningen, The Netherlands.

© 2001 – 2021:

> 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 one of the mailing lists (see our webpage or this section on how to *contribute* (page 568)), 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 527)
- *Berendsen et al. (1995)* (page 527)
- *Lindahl et al. (2001)* (page 527)
- *van der Spoel at al. (2005)* (page 527)
- *Hess et al. (2008)* (page 527)
- *Pronk et al. (2013)* (page 527)
- *Pall et al. (2015)* (page 527)
- *Abraham et al. (2015)* (page 527)

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's 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 527), 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 527).

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 ν approximates or exceeds $k_B T/h$. Now 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 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	libration	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 ν are: *11* (page 527)

$$U^{QM} = U^{cl} + kT\left(\frac{1}{2}x - 1 + \frac{x}{e^x - 1}\right) \tag{5.3}$$

$$C_V^{QM} = C_V^{cl} + k\left(\frac{x^2 e^x}{(e^x - 1)^2} - 1\right) \tag{5.4}$$

  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 527). 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 527).

Electrons are in the ground state In MD 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 In this version, GROMACS always 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 527), *15* (page 527).

Boundary conditions are unnatural Since system size is small (even 10,000 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 sec. *Computational Chemistry and Molecular Modeling* (page 303), in many cases energy minimization is required. GROMACS provides a number of methods for local energy minimization, as detailed in sec. *Energy Minimization* (page 343).

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 527). 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 527)), 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 527), 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 527)) 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	μ	$\mathrm{e\ nm}$
electric potential	Φ	$\mathrm{kJ\ mol^{-1}\ e^{-1}} = 0.010\,364\,269\,19$ 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 Φ 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 Φ 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 ≈ 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	$6.022\,141\,29 \times 10^{23}\ \mathrm{mol^{-1}}$
R	gas constant	$8.314\,462\,1 \times 10^{-3}\ \mathrm{kJ\ mol^{-1}\ K^{-1}}$
k_B	Boltzmann's constant	*idem*
h	Planck's constant	$0.399\,031\,271\ \mathrm{kJ\ mol^{-1}\ ps}$
$\hbar$	Dirac's constant	$0.063\,507\,799\,3\ \mathrm{kJ\ mol^{-1}\ ps}$
c	velocity of light	$299\,792.458\ \mathrm{nm\ ps^{-1}}$

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^*	$k_B T\ \epsilon^{-1}$
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	ρ^*	$N\sigma^3\ V^{-1}$

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 396)). 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 311)) and the *group concept* (sec. *The group concept* (page 314)). The MD algorithm is described in sec. *Molecular Dynamics* (page 315): 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 343). 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 356). 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 499).

5.4.1 Periodic boundary conditions

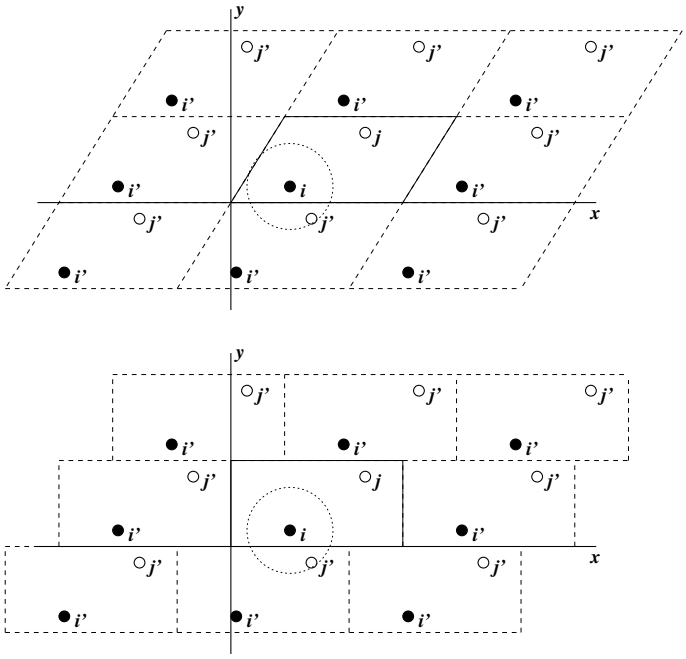


Fig. 5.1: Periodic boundary conditions in two dimensions.

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 527) 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 528). 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 237) 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.

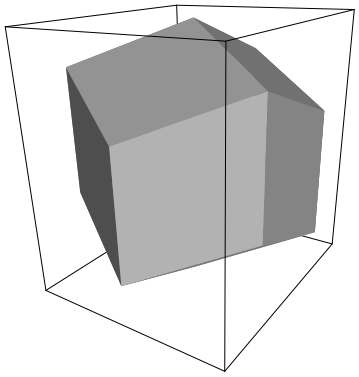


Fig. 5.2: A rhombic dodecahedron (arbitrary orientation).

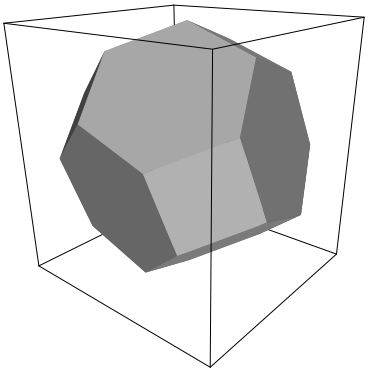


Fig. 5.3: A truncated octahedron (arbitrary orientation).

Some useful box types

Table 5.6: Overview over different box types

box type	image distance	box volume	box vectors			box vector angles		
			a	**b**	**c**	$\angle$ **bc**	$\angle$ **ac**	$\angle$ **ab**
cubic	d	d^3	d	0	0	90°	90°	90°
			0	d	0			
			0	0	d			
rhombic dodcahdron (xy-square)	d	$\frac{1}{2}\sqrt{2}\,d^3$ $0.707\,d^3$	d	0	$\frac{1}{2}\,d$	60°	60°	60°
			0	d	$\frac{1}{2}\,d$			
			0	0	$\frac{1}{2}\sqrt{2}\,d$			
rhombic dodcahdron (xy- hexagon)	d	$\frac{1}{2}\sqrt{2}\,d^3$ $0.707\,d^3$	d	$\frac{1}{2}\,d$	$\frac{1}{2}\,d$	60°	60°	60°
			0	$\frac{1}{2}\sqrt{3}\,d$	$\frac{1}{6}\sqrt{3}\,d$			
			0	0	$\frac{1}{3}\sqrt{6}\,d$			
truncated octahedron	d	$\frac{4}{9}\sqrt{3}\,d^3$ $0.770\,d^3$	d	$\frac{1}{3}\,d$	$-\frac{1}{3}\,d$	71.53°	109.47°	71.53°
			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 150) 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 321)). 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 322)) 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.

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 448) or *tng* (page 444)), 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 499).

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
$\Downarrow$

---

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.
$\Downarrow$
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
$\frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} = \mathbf{v}_i; \quad \frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = \frac{\mathbf{F}_i}{m_i}$
$\Downarrow$
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 22) in Sections *System preparation* (page 26) and *Getting started* (page 22).

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 356) and *top* (page 445), respectively. All this information is static; it is never modified during the run.

Coordinates and velocities

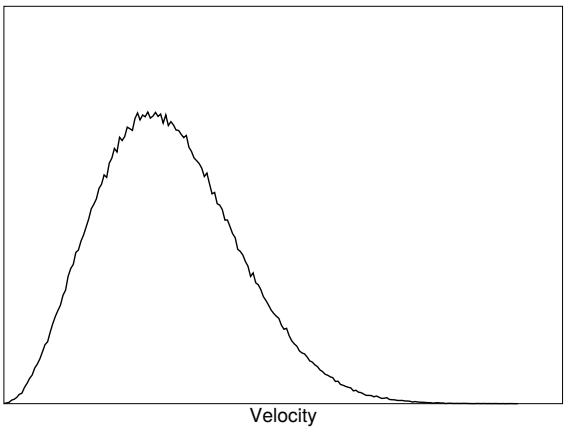


Fig. 5.4: A Maxwell-Boltzmann velocity distribution, generated from random numbers.

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 Δt (see *The leap-frog integrator* (page 322)), 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:

$$p(v_i) = \sqrt{\frac{m_i}{2\pi kT}} \exp\left(-\frac{m_i v_i^2}{2kT}\right) \tag{5.16}$$

where k is Boltzmann's constant (see chapter *Definitions and Units* (page 308)). 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 356), 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 311)). 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 stream computing, using e.g. SSE, AVX or CUDA on 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/ns 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/ns per particle, so it doesn't 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 535).

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 is overlapped with the integration on the CPU, so it is free in most 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_ℓ 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 r_b the Verlet buffer size, 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 ρ_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:

$$\begin{aligned}\langle \Delta V \rangle \approx \int_{-\infty}^{r_c} \int_{r_\ell}^{\infty} 4\pi r_0^2 \rho_2 \Big[& V'(r_c)(r_t - r_c) + \\ & V''(r_c)\frac{1}{2}(r_t - r_c)^2 + \\ & V'''(r_c)\frac{1}{6}(r_t - r_c)^3 + \\ & O\big((r_t - r_c)^4\big) \Big] G\left(\frac{r_t - r_0}{\sigma}\right) dr_0\, dr_t\end{aligned}$$

Replacing the factor r_0^2 by $(r_\ell + \sigma)^2$, which results in a slight overestimate, allows us to calculate the integrals analytically:

$$\begin{aligned}
\langle \Delta V \rangle \approx & \quad 4\pi(r_\ell+\sigma)^2\rho_2 \int_{-\infty}^{r_c}\int_{r_\ell}^{\infty} \Big[V'(r_c)(r_t-r_c)+ \\
& V''(r_c)\frac{1}{2}(r_t-r_c)^2+ \\
& V'''(r_c)\frac{1}{6}(r_t-r_c)^3\Big] G\Big(\frac{r_t-r_0}{\sigma}\Big)\, dr_0\, dr_t \\
= & \quad 4\pi(r_\ell+\sigma)^2\rho_2 \Big\{\frac{1}{2}V'(r_c)\Big[r_b\sigma G\Big(\frac{r_b}{\sigma}\Big) - (r_b^2+\sigma^2)E\Big(\frac{r_b}{\sigma}\Big)\Big] + \\
& \frac{1}{6}V''(r_c)\Big[\sigma(r_b^2+2\sigma^2)G\Big(\frac{r_b}{\sigma}\Big) - r_b(r_b^2+3\sigma^2)E\Big(\frac{r_b}{\sigma}\Big)\Big] + \\
& \frac{1}{24}V'''(r_c)\Big[r_b\sigma(r_b^2+5\sigma^2)G\Big(\frac{r_b}{\sigma}\Big) \\
& - (r_b^4+6r_b^2\sigma^2+3\sigma^4)E\Big(\frac{r_b}{\sigma}\Big)\Big]\Big\}
\end{aligned}$$

where $G(x)$ is a Gaussian distribution with 0 mean and unit variance and $E(x) = \frac{1}{2}\mathrm{erfc}(x/\sqrt{2})$. We always want to achieve small energy error, so σ will be small compared to both r_c and r_ℓ, 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 don't 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.

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 σ^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 σ 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 4×4 list, but the list can also be 4×8 (GPU CUDA kernels and AVX 256-bit single precision kernels) or 4×2 (SSE double-precision kernels). This means that the pair-list is effectively much larger than the corresponding 1×1 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.

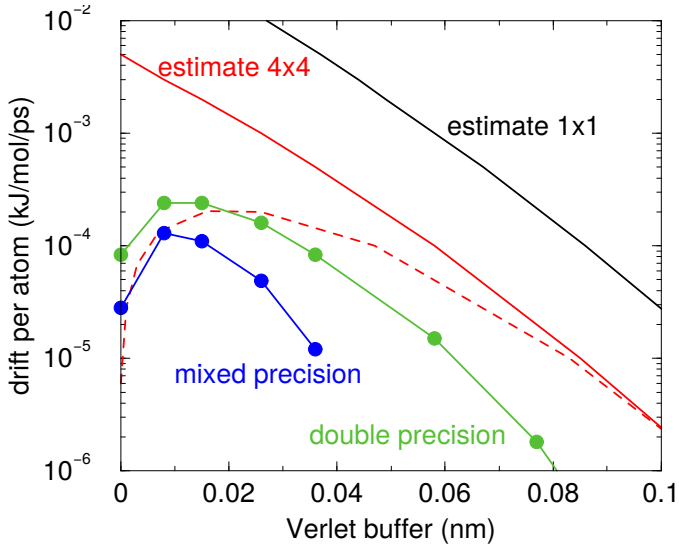


Fig. 5.5: Energy drift per atom for an SPC/E water system at 300K with a time step of 2 fs and a pair-list update period of 10 steps (pair-list life time: 18 fs). PME was used with `ewald-rtol` set to 10^{-5}; this parameter affects the shape of the potential at the cut-off. Error estimates due to finite Verlet buffer size are shown for a 1×1 atom pair list and 4×4 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 don't move freely over 18 fs, but rather vibrate.

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 359)). One then has a buffer with the size equal to the neighbor list cut-off less the longest interaction cut-off.

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:

$$\begin{aligned}
\mathbf{r}''' &= \mathbf{r}_j - \mathbf{r}_i \\
\mathbf{r}'' &= \mathbf{r}''' - \mathbf{c} * \mathrm{round}(r'''_z / c_z) \\
\mathbf{r}' &= \mathbf{r}'' - \mathbf{b} * \mathrm{round}(r''_y / b_y) \\
\mathbf{r}_{ij} &= \mathbf{r}' - \mathbf{a} * \mathrm{round}(r'_x / a_x)
\end{aligned} \tag{5.19}$$

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.

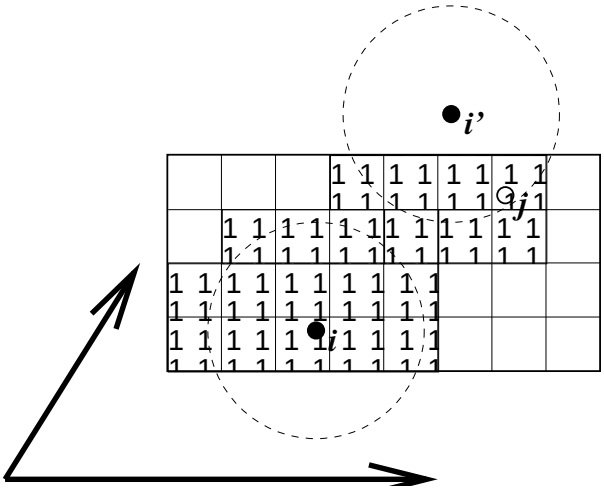


Fig. 5.6: Grid search in two dimensions. The arrows are the box vectors.

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 314)).

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_{kin} and the virial $\boldsymbol{\Xi}$:

$$\mathbf{P} = \frac{2}{V}(\mathbf{E}_{\mathrm{kin}} - \boldsymbol{\Xi}) \tag{5.25}$$

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:

$$P = \mathrm{trace}(\mathbf{P})/3$$

The virial $\boldsymbol{\Xi}$ tensor is defined as:

$$\boldsymbol{\Xi} = -\frac{1}{2}\sum_{i<j} \mathbf{r}_{ij} \otimes \mathbf{F}_{ij} \tag{5.26}$$

The GROMACS implementation of the virial computation is described in sec. *Virial and pressure* (page 396)

The leap-frog integrator

The default MD integrator in GROMACS is the so-called *leap-frog* algorithm *22* (page 528) 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 323)). The leap-frog algorithm uses positions $\mathbf{r}$ at time t and

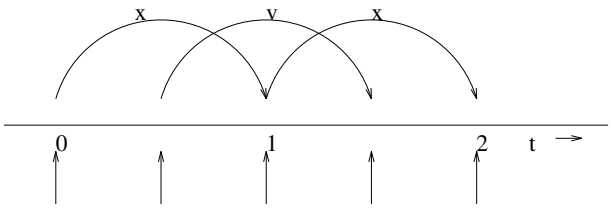


Fig. 5.7: The Leap-Frog integration method. The algorithm is called Leap-Frog because $\mathbf{r}$ and $\mathbf{v}$ are leaping like frogs over each other's backs.

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 528) 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 528) 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 528) 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

$$\begin{aligned} \Gamma(t) = &\quad \exp(iLt)\Gamma(0) \\ iL = &\quad \dot{\Gamma} \cdot \nabla_\Gamma, \end{aligned}$$

where L is the Liouville operator, and Γ 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

$$\begin{aligned} iL_1 = &\quad \sum_{i=1}^{N} \frac{1}{m_i}\mathbf{F}(r_i) \cdot \nabla_{\mathbf{v}_i} \\ iL_2 = &\quad \sum_{i=1}^{N} \mathbf{v}_i \cdot \nabla_{\mathbf{r}_i} \end{aligned}$$

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 Δt 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

$$\begin{aligned} \exp(iLn\Delta t) \approx &\quad \left(\exp(iL_2\frac{1}{2}\Delta t)\exp(iL_1\Delta t)\exp(iL_2\frac{1}{2}\Delta t)\right)^n \\ \approx &\quad \exp(iL_2\frac{1}{2}\Delta t)\left(\exp(iL_1\Delta t)\exp(iL_2\Delta t)\right)^{n-1} \\ &\qquad \exp(iL_1\Delta t)\exp(iL_2\frac{1}{2}\Delta t) \end{aligned}$$

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

$$\exp(iLn\Delta t) = \quad \exp(iL_1\Delta t)\exp(iL_2\Delta t) + \mathcal{O}(\Delta t^3). \tag{5.34}$$

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

$$\exp(iLn\Delta t) \approx \quad \left(\exp(iL_1\Delta t)\exp(iL_2\Delta t)\right)^n. \tag{5.35}$$

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 440) file option `integrator=md-vv-avek` (page 36)). 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 36) and `integrator=md-vv-avek` (page 36)) 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 36)). 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

Several other simulation packages uses multiple time stepping for bonds and/or the PME mesh forces. In GROMACS we have not implemented this (yet), since we use a different philosophy. Bonds can be constrained (which is also a more sound approximation of a physical quantum oscillator), which allows the smallest time step to be increased to the larger one. This not only halves the number of force calculations, but also the update calculations. For even larger time steps, angle vibrations involving hydrogen atoms can be removed using virtual interaction sites (see sec. *Removing fastest degrees of freedom* (page 481)), which brings the shortest time step up to PME mesh update frequency of a multiple time stepping scheme.

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 528), stochastic randomization through the Andersen thermostat *27* (page 528), the extended ensemble Nosé-Hoover scheme *28* (page 528), *29* (page 528), or a velocity-rescaling scheme *30* (page 528) 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 528) 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 τ. 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 528) described below.

The heat flow into or out of the system is affected by scaling the velocities of each particle every step, or every n_{TC} steps, with a time-dependent factor λ, 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 τ_T is close, but not exactly equal, to the time constant τ 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 τ/τ_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 τ_T. **Note** that in practice the scaling factor λ 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, λ 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 528) 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 528). 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 Δt is the timestep and τ_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 τ_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 528).

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 528) and later modified by Hoover *29* (page 528). 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, ξ. This friction parameter (or *heat bath* variable) is a fully dynamic quantity with its own momentum (p_ξ) 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 315) 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 ξ is:

$$\frac{\mathrm{d}p_\xi}{\mathrm{d}t} = (T - T_0)\,. \tag{5.43}$$

The reference temperature is denoted T_0, while T is the current instantaneous temperature of the system. 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

$$H = \sum_{i=1}^{N} \frac{\mathbf{p}_i}{2m_i} + U\left(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N\right) + \frac{p_\xi^2}{2Q} + N_f kT\xi, \tag{5.44}$$

where N_f is the total number of degrees of freedom.

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 to let the GROMACS user work instead with the period τ_T of the oscillations of kinetic energy between the system and the reservoir instead. It is directly related to Q and T_0 via:

$$Q = \frac{\tau_T^2 T_0}{4\pi^2}. \tag{5.45}$$

This provides a much more intuitive way of selecting the Nosé-Hoover coupling strength (similar to the weak-coupling relaxation), and in addition τ_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

[1] Note that some derivations, an alternative notation $\xi_{\mathrm{alt}} = v_\xi = p_\xi/Q$ is used.

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 *33* (page 528). 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 528):

$$
\begin{aligned}
\frac{\mathrm{d}^2\mathbf{r}_i}{\mathrm{d}t^2} &= \frac{\mathbf{F}_i}{m_i} - \frac{p_{\xi_1}}{Q_1}\frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} \\
\frac{\mathrm{d}p_{\xi_1}}{\mathrm{d}t} &= (T - T_0) - p_{\xi_1}\frac{p_{\xi_2}}{Q_2} \\
\frac{\mathrm{d}p_{\xi_{i=2\ldots N}}}{\mathrm{d}t} &= \left(\frac{p^2_{\xi_{i-1}}}{Q_{i-1}} - kT\right) - p_{\xi_i}\frac{p_{\xi_{i+1}}}{Q_{i+1}} \\
\frac{\mathrm{d}p_{\xi_N}}{\mathrm{d}t} &= \left(\frac{p^2_{\xi_{N-1}}}{Q_{N-1}} - kT\right)
\end{aligned}
$$

The conserved quantity for Nosé-Hoover chains is

$$
H = \sum_{i=1}^{N}\frac{\mathbf{p}_i}{2m_i} + U\left(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N\right) + \sum_{k=1}^{M}\frac{p^2_{\xi_k}}{2Q'_k} + N_f kT\xi_1 + kT\sum_{k=2}^{M}\xi_k \tag{5.46}
$$

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 440) option the values of all the positions and velocities of all Nosé-Hoover particles in the chain are written to the *edr* (page 436) file. Leap-frog simulations currently can only have Nosé-Hoover chain lengths of 1, but this will likely be updated in 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 528)), we split the Liouville operator as

$$
iL = iL_1 + iL_2 + iL_{\mathrm{NHC}}, \tag{5.47}
$$

where

$$
\begin{aligned}
iL_1 &= \sum_{i=1}^{N}\left[\frac{\mathbf{p}_i}{m_i}\right]\cdot\frac{\partial}{\partial\mathbf{r}_i} \\
iL_2 &= \sum_{i=1}^{N}\mathbf{F}_i\cdot\frac{\partial}{\partial\mathbf{p}_i} \\
iL_{\mathrm{NHC}} &= \sum_{i=1}^{N} -\frac{p_\xi}{Q}\mathbf{v}_i\cdot\nabla_{\mathbf{v}_i} + \frac{p_\xi}{Q}\frac{\partial}{\partial\xi} + (T - T_0)\frac{\partial}{\partial p_\xi}
\end{aligned}
$$

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 528).

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 440) 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 528).

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 τ_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 528).

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 528) that scales coordinates and box vectors every step, the extended-ensemble Parrinello-Rahman approach *38* (page 528), *39* (page 528), and for the velocity Verlet variants, the Martyna-Tuckerman-Tobias-Klein (MTTK) implementation of pressure control *35* (page 528). 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 5.1 afterwards, 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_{PC} steps, with a matrix μ, 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 μ 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, β 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 β 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 7.6×10^{-4} MD units (see chapter *Definitions and Units* (page 308)). 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 μ' 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 δ_{ij} is the Kronecker delta and $\mathbf{\Xi}$ 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 it is not yet clear exactly what errors this approximation may yield.

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_{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

$$\mathrm{d}\epsilon = -\frac{\beta}{\tau_p}(P_0 - P)\mathrm{d}t + \sqrt{\frac{2k_B T\beta}{V\tau_p}}\mathrm{d}W \tag{5.52}$$

Here, β is the isothermal compressibility of the system. It suffices to take a rough estimate because the value of β 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 7.6×10^{-4} MD units (see chapter *Definitions and Units* (page 308)). 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:

$$\mathrm{d}\epsilon_{xy} = -\frac{2\beta}{3\tau_p}(P_0 - \frac{\gamma}{L} - \frac{P_{xx} + P_{yy}}{2})\mathrm{d}t + \sqrt{\frac{4k_B T\beta}{3V\tau_p}}\mathrm{d}W_{xy} \tag{5.53}$$

$$\mathrm{d}\epsilon_z = -\frac{\beta}{3\tau_p}(P_0 - P_{zz})\mathrm{d}t + \sqrt{\frac{2k_B T\beta}{3V\tau_p}}\mathrm{d}W_z \tag{5.54}$$

Here γ 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 535).

Parrinello-Rahman pressure coupling

In cases where the fluctuations in pressure or volume are important *per se* (*e.g.* to calculate thermodynamic properties), especially for small systems, it may be a problem that the exact ensemble is not well defined for the weak-coupling scheme, and that it does not simulate the true NPT ensemble.

GROMACS also supports constant-pressure simulations using the Parrinello-Rahman approach *38* (page 528), *39* (page 528), 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 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. The matrices and $_{ref}$ are the current and reference pressures, respectively.

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 ficticious, 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

[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.

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 528) that we will try to implement at some point in the future together with support for external stress.

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 β and the pressure time constant τ_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 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 scheme 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 μ_{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 γ_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 β_{zz} is more critical than with normal pressure coupling. Normally an incorrect compressibility will just scale τ_p, but with surface tension coupling it affects the convergence of the surface tension. When β_{zz} is set to zero (constant box height), ΔP_{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 528) and Tuckerman *41* (page 528) 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 ϵ is $v_\epsilon = p_\epsilon/W = \dot{\epsilon} = \dot{V}/3V$, and define $\alpha = 1 + 3/N_{dof}$ (see Ref *41* (page 528))

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 α are required to make phase space incompressible *41* (page 528). The ϵ 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 η 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

$$
H = \sum_{i=1}^{N}\frac{\mathbf{p}_i}{2m_i} + U(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) + \frac{p_\epsilon^2}{2W} + PV +
$$

$$
\sum_{k=1}^{M}\frac{p_{\xi_k}^2}{2Q_k} + \sum_{k=1}^{M}\frac{p_{\eta_k}^2}{2Q'_k} + N_f kT\xi_1 + kT\sum_{i=2}^{M}\xi_k + kT\sum_{k=1}^{M}\eta_k
$$

Returning to the Trotter decomposition formalism, for pressure control and temperature control *35* (page 528) we get:

$$
iL = iL_1 + iL_2 + iL_{\epsilon,1} + iL_{\epsilon,2} + iL_{\text{NHC-baro}} + iL_{\text{NHC}} \tag{5.66}
$$

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

$$\begin{aligned}\exp(iL\Delta t) = \quad & \exp\left(iL_{\text{NHC}-\text{baro}}\Delta t/2\right)\exp\left(iL_{\text{NHC}}\Delta t/2\right)\\ & \exp\left(iL_{\epsilon,2}\Delta t/2\right)\exp\left(iL_2\Delta t/2\right)\\ & \exp\left(iL_{\epsilon,1}\Delta t\right)\exp\left(iL_1\Delta t\right)\\ & \exp\left(iL_2\Delta t/2\right)\exp\left(iL_{\epsilon,2}\Delta t/2\right)\\ & \exp\left(iL_{\text{NHC}}\Delta t/2\right)\exp\left(iL_{\text{NHC}-\text{baro}}\Delta t/2\right) + \mathcal{O}(\Delta t^3)\end{aligned}$$

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_ϵ 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

$$\begin{aligned}\exp(iL\Delta t) = \quad & \exp\left(iL_{\text{NHC}-\text{baro}}\Delta t/2\right)\exp\left(iL_{\epsilon,2}\Delta t/2\right)\exp\left(iL_2\Delta t/2\right)\\ & \exp\left(iL_{\text{NHC}}\Delta t/2\right)\exp\left(iL_{\epsilon,1}\Delta t\right)\exp\left(iL_1\Delta t\right)\exp\left(iL_{\text{NHC}}\Delta t/2\right)\\ & \exp\left(iL_2\Delta t/2\right)\exp\left(iL_{\epsilon,2}\Delta t/2\right)\exp\left(iL_{\text{NHC}-\text{baro}}\Delta t/2\right) + \mathcal{O}(\Delta t^3)\end{aligned}$$

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 529). 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 528).

Standard velocity Verlet with Nosé-Hoover temperature control has a Trotter expansion

$$\begin{aligned}\exp(iL\Delta t) \approx \quad & \exp\left(iL_{\text{NHC}}\Delta t/2\right)\exp\left(iL_2\Delta t/2\right)\\ & \exp\left(iL_1\Delta t\right)\exp\left(iL_2\Delta t/2\right)\exp\left(iL_{\text{NHC}}\Delta t/2\right).\end{aligned}$$

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

$$\begin{aligned}\exp(iL\Delta t) \approx \quad & \left(\exp\left(iL_{\text{NHC}}(n\Delta t/2)\right)\left[\exp\left(iL_2\Delta t/2\right)\right.\right.\\ & \left.\exp\left(iL_1\Delta t\right)\exp\left(iL_2\Delta t/2\right)\right]^n\exp\left(iL_{\text{NHC}}(n\Delta t/2)\right).\end{aligned}$$

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 λ and μ 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 337) 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 314)). This vector contains only zero (freeze) or one (don't 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. 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.

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 440) file to have *mdrun* (page 183) use this feature.

5.4.4 Shell molecular dynamics

GROMACS can simulate polarizability using the shell model of Dick and Overhauser *43* (page 529). 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 529) and water*45* (page 529).

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 529) 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

$$\sigma_k(\mathbf{r}_1 \ldots \mathbf{r}_N) = 0; \quad k = 1 \ldots K. \tag{5.80}$$

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 λ_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 339)

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 529) (sec. *Constraint algorithms* (page 409)). 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 rouding 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 529).

LINCS

The LINCS algorithm

LINCS is an algorithm that resets bonds to their correct lengths after an unconstrained update *49* (page 529). 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 352).

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 529). Only a short description of the first step is given here.

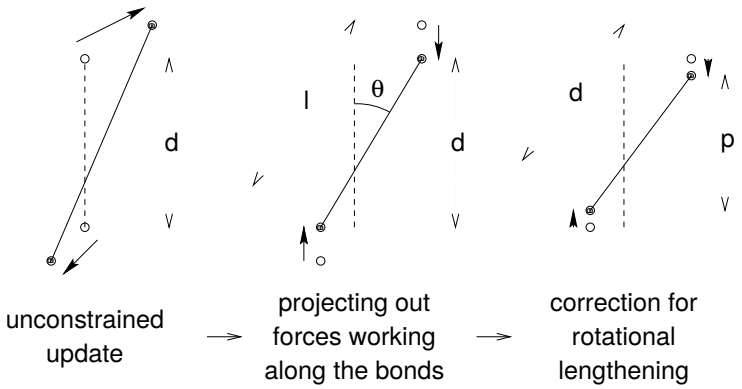


Fig. 5.8: The three position updates needed for one time step. The dashed line is the old bond of length d, the solid lines are the new bonds. $l = d\cos\theta$ and $p = (2d^2 - l^2)^{\frac{1}{2}}$.

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 529).

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 ϕ 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 529).

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 440) 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 440) 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 γ_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 529). It can be described as:

$$\begin{aligned}
\mathbf{v}' &= \mathbf{v}(t - \frac{1}{2}\Delta t) + \frac{1}{m}\mathbf{F}(t)\Delta t \\
\Delta\mathbf{v} &= -\alpha\, \mathbf{v}'(t + \frac{1}{2}\Delta t) + \sqrt{\frac{k_B T}{m}(1-\alpha^2)}\, \mathbf{r}^G{}_i \\
\mathbf{r}(t+\Delta t) &= \mathbf{r}(t) + \left(\mathbf{v}' + \frac{1}{2}\Delta\mathbf{v}\right)\Delta t
\end{aligned} \tag{5.94}$$

$$\begin{aligned}
\mathbf{v}(t + \frac{1}{2}\Delta t) &= \mathbf{v}' + \Delta\mathbf{v} \\
\alpha &= 1 - e^{-\gamma \Delta t}
\end{aligned} \tag{5.95}$$

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 implented 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.

When using SD as a thermostat, an appropriate value for γ 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 γ_i is the friction coefficient [amu/ps] and $\mathring{\mathbf{r}}_i(t)$ is a noise process with $\langle \mathring{r}_i(t)\mathring{r}_j(t+s)\rangle = 2\delta(s)\delta_{ij}k_BT/\gamma_i$. In GROMACS the equations are integrated with a simple, explicit scheme

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \frac{\Delta t}{\gamma_i}\mathbf{F}_i(\mathbf{r}(t)) + \sqrt{2k_BT\frac{\Delta t}{\gamma_i}}\,\mathbf{r}^G{}_i, \tag{5.97}$$

where $\mathbf{r}^G{}_i$ is Gaussian distributed noise with $\mu = 0$, $\sigma = 1$. The friction coefficients γ_i can be chosen the same for all particles or as $\gamma_i = m_i\,\gamma_i$, where the friction constants γ_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 is an option of the *mdrun* (page 183) program.

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). EM is just an option of the *mdrun* (page 183) 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

$$\mathbf{r}_{n+1} = \mathbf{r}_n + \frac{\mathbf{F}_n}{\max(|\mathbf{F}_n|)}h_n, \tag{5.98}$$

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 ϵ. 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 ϵ can be estimated from the root mean square force f a harmonic oscillator would exhibit at a temperature T. This value is

$$f = 2\pi\nu\sqrt{2mkT}, \tag{5.99}$$

where ν 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 ϵ 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. In GROMACS conjugate gradient can not be used with constraints, including the SETTLE algorithm for water *47* (page 529), as this has not been implemented. If water is present it must be of a flexible model, which can be specified in the *mdp* (page 440) file by `define = -DFLEXIBLE`.

This is not really a restriction, since the accuracy of conjugate gradient is only required for minimization prior to a normal-mode analysis, which cannot be performed with constraints. 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 529), *53* (page 529), 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 529)*56* (page 529) can be performed using GROMACS, by diagonalization of the mass-weighted Hessian H:

$$\begin{aligned} R^T M^{-1/2} H M^{-1/2} R = & \quad \text{diag}(\lambda_1, \ldots, \lambda_{3N}) \\ \lambda_i = & \quad (2\pi\omega_i)^2 \end{aligned} \tag{5.100}$$

where M contains the atomic masses, R is a matrix that contains the eigenvectors as columns, λ_i are the eigenvalues and ω_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:

$$H_{ij} = \frac{\partial^2 V}{\partial x_i \partial x_j} \tag{5.101}$$

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:

$$\begin{aligned} H_{ij} = & \quad -\frac{f_i(\mathbf{x} + h\mathbf{e}_j) - f_i(\mathbf{x} - h\mathbf{e}_j)}{2h} \\ f_i = & \quad -\frac{\partial V}{\partial x_i} \end{aligned} \tag{5.102}$$

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 183). Then, *mdrun* (page 183) 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 191) does the diagonalization and the sorting of the normal modes according to their frequencies. Both *mdrun* (page 183) and *gmx nmeig* (page 191) should be run in double precision. The normal modes can be analyzed with the program *gmx anaeig* (page 110). Ensembles of structures at any temperature and for any subset of normal modes can be generated with *gmx nmens* (page 193). An overview of normal-mode analysis and the related principal component analysis (see sec. *Covariance analysis* (page 511)) can be found in *57* (page 529).

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:

$$\Delta G_1 - \Delta G_2 = \Delta G_3 - \Delta G_4 \tag{5.103}$$

If we are interested in the left-hand term we can equally well compute the right-hand term.

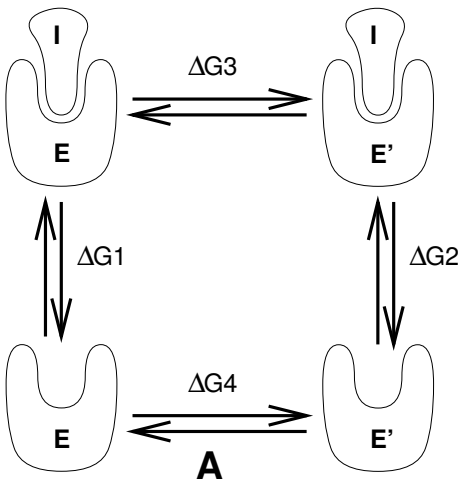


Fig. 5.9: Free energy cycles. **A:** to calculate ΔG_{12}, the free energy difference between the binding of inhibitor **I** to enzymes **E** respectively **E**′.

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 (5.103) 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

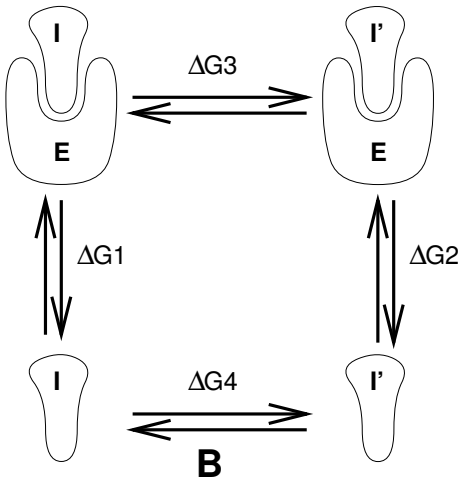


Fig. 5.10: Free energy cycles. **B:** to calculate ΔG_{12}, the free energy difference for binding of inhibitors **I** respectively **I**′ to enzyme **E**.

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* λ : $H = H(p, q; \lambda)$ in such a way that $\lambda = 0$ describes system A and $\lambda = 1$ describes system B:

$$H(p, q; 0) = H^{\mathrm{A}}(p, q); \quad H(p, q; 1) = H^{\mathrm{B}}(p, q). \tag{5.104}$$

In GROMACS, the functional form of the λ-dependence is different for the various force-field contributions and is described in section sec. *Free energy interactions* (page 383).

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 Δ of an N, p, T ensemble, which is assumed to be the equilibrium ensemble generated by a MD simulation at constant pressure and temperature:

$$\begin{aligned}
A(\lambda) &= -k_B T \ln Q \\
Q &= c \iint \exp[-\beta H(p, q; \lambda)]\, dp\, dq \\
G(\lambda) &= -k_B T \ln \Delta \\
\Delta &= c \iiint \exp[-\beta H(p, q; \lambda) - \beta p V]\, dp\, dq\, dV \\
G &= A + pV,
\end{aligned} \tag{5.105}$$

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 λ as an ensemble average:

$$\frac{dA}{d\lambda} = \frac{\iint (\partial H/\partial \lambda) \exp[-\beta H(p, q; \lambda)]\, dp\, dq}{\iint \exp[-\beta H(p, q; \lambda)]\, dp\, dq} = \left\langle \frac{\partial H}{\partial \lambda} \right\rangle_{NVT;\lambda}, \tag{5.106}$$

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 λ:

$$A^{\mathrm{B}}(V, T) - A^{\mathrm{A}}(V, T) = \int_0^1 \left\langle \frac{\partial H}{\partial \lambda} \right\rangle_{NVT;\lambda} d\lambda \tag{5.107}$$

$$G^{\mathrm{B}}(p,T) - G^{\mathrm{A}}(p,T) = \int_0^1 \left\langle \frac{\partial H}{\partial \lambda} \right\rangle_{NpT;\lambda} d\lambda. \tag{5.108}$$

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:

$$G^{\mathrm{B}}(p) - G^{\mathrm{A}}(p) = A^{\mathrm{B}}(V) - A^{\mathrm{A}}(V) - \int_p^{p^{\mathrm{B}}} [V^{\mathrm{B}}(p') - V]\, dp' \tag{5.109}$$

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 ΔV is the volume change at p and κ 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 $-\frac{3}{2}k_B T \ln(m^{\mathrm{B}}/m^{\mathrm{A}})$. **Note** that this is only true in the absence of constraints.

Thermodynamic integration

GROMACS offers the possibility to integrate (5.107) or eq. (5.108) 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 λ. 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 529) for calculating values of ΔG for transformations from state A to state B using the program *gmx bar* (page 117). The same data can also be used to calculate free energies using MBAR *59* (page 529), though the analysis currently requires external tools from the external pymbar package.

The λ-dependence for the force-field contributions is described in detail in section sec. *Free energy interactions* (page 383).

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:

$$P(1 \leftrightarrow 2) = \min\left(1, \exp\left[\left(\frac{1}{k_B T_1} - \frac{1}{k_B T_2}\right)(U_1 - U_2)\right]\right) \tag{5.110}$$

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 529), *61* (page 529). 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:

$$U_1 - U_2 = N_{df} \frac{c}{2} k_B (T_1 - T_2) \tag{5.111}$$

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:

$$P(1 \leftrightarrow 2) = \exp\left(-\frac{\epsilon^2 c\, N_{df}}{2(1+\epsilon)}\right) \approx \exp\left(-\epsilon^2 \frac{c}{2} N_{df}\right) \tag{5.112}$$

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 ϵ 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 529), which should be taken into account. The GROMACS website features a so-called `REMD calculator`, that lets you type in the temperature range and the number of atoms, and based on that proposes a set of temperatures.

An extension to the REMD for the isobaric-isothermal ensemble was proposed by Okabe et al. *63* (page 529). In this work the exchange probability is modified to:

$$P(1 \leftrightarrow 2) = \min\left(1, \exp\left[\left(\frac{1}{k_B T_1} - \frac{1}{k_B T_2}\right)(U_1 - U_2) + \left(\frac{P_1}{k_B T_1} - \frac{P_2}{k_B T_2}\right)(V_1 - V_2)\right]\right) \tag{5.113}$$

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:

$$P(1 \leftrightarrow 2) = \min\left(1, \exp\left[\left(\frac{1}{k_B T_1} - \frac{1}{k_B T_2}\right)((U_1(x_2) - U_1(x_1)) + (U_2(x_1) - U_2(x_2)))\right]\right) \tag{5.114}$$

The separate Hamiltonians are defined by the free energy functionality of GROMACS, with swaps made between the different values of λ defined in the mdp file.

Hamiltonian and temperature replica exchange can also be performed simultaneously, using the acceptance criteria:

$$P(1 \leftrightarrow 2) = \min\left(1, \exp\left[\left(\frac{1}{k_B T_1} - \frac{1}{k_B T_2}\right)(\frac{U_1(x_2) - U_1(x_1)}{k_B T_1} + \frac{U_2(x_1) - U_2(x_2)}{k_B T_2})\right]\right) \tag{5.115}$$

Gibbs sampling replica exchange has also been implemented in GROMACS *64* (page 530). 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 options of the *mdrun* (page 183) 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 183) on how to use these multinode features.

5.4.13 Essential Dynamics sampling

The results from Essential Dynamics (see sec. *Covariance analysis* (page 511)) 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 530), *66* (page 530). 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 530).

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 132)) or normal-mode analysis (*gmx nmeig* (page 191)). Then, this information is fed into *make_edi* (page 179), which has many options for selecting vectors and setting parameters, see `gmx make_edi -h`. The generated *edi* (page 436) input file is then passed to *mdrun* (page 183).

5.4.14 Expanded Ensemble

In an expanded ensemble simulation *68* (page 530), 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:

$$P(\vec{x}, k) \propto \exp\left(-\beta_k U_k + g_k\right), \tag{5.116}$$

where $\beta_k = \frac{1}{k_B T_k}$ is the β corresponding to the kth thermodynamic state, and g_k is a user-specified weight factor corresponding to the kth 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 λ 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 530) and the midpoint *70* (page 530) 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 527).

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 independenly. 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.

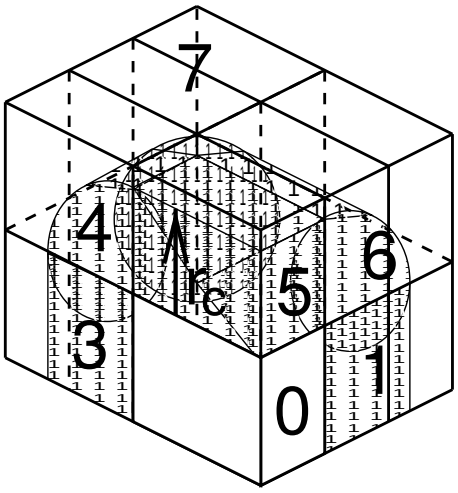


Fig. 5.11: A non-staggered domain decomposition grid of 3×2×2 cells. Coordinates in zones 1 to 7 are communicated to the corner cell that has its home particles in zone 0. r_c is the cut-off radius.

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 527) for details on determining which non-bonded and bonded forces should be calculated on which rank.

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 a charge group needs to be communicated: a non-bonded distance and a bonded distance check.

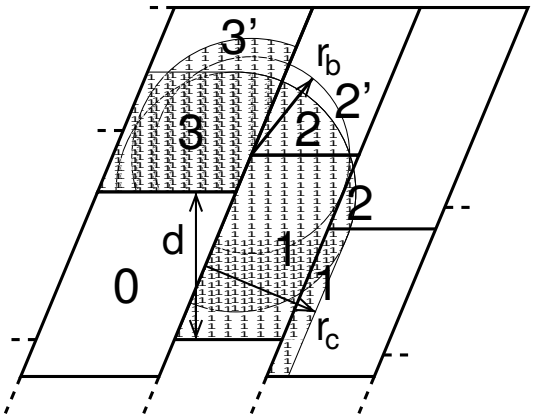


Fig. 5.12: The zones to communicate to the rank of zone 0, see the text for details. r_c and r_b are the non-bonded and bonded cut-off radii respectively, d is an example of a distance between following, staggered boundaries of cells.

By default, *mdrun* (page 183) 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 183).

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 imbalance 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 183) 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 529), which is the parallel version of the LINCS algorithm *49* (page 529) (see *The LINCS algorithm* (page 339)). 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.

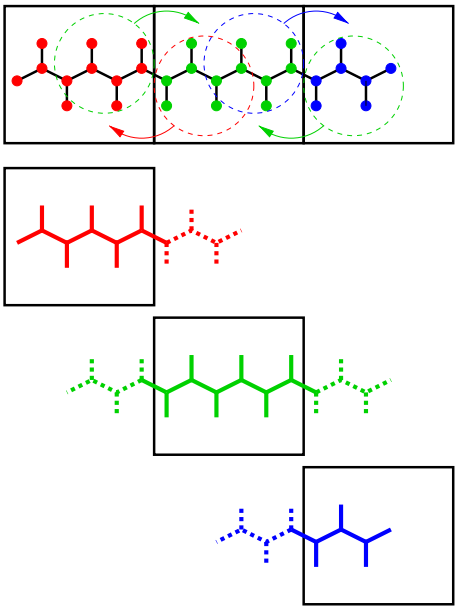


Fig. 5.13: Example of the parallel setup of P-LINCS with one molecule split over three domain decomposition cells, using a matrix expansion order of 3. The top part shows which atom coordinates need to be communicated to which cells. The bottom parts show the local constraints (solid) and the non-local 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 183) 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_{list},r_{VdW},r_{Coul})	*mdp* (page 440) file	
two-body bonded	max(r_{mb},r_c)	*mdrun* (page 183) `-rdd`	starting conf. + 10%
multi-body bonded	r_{mb}	*mdrun* (page 183) `-rdd`	starting conf. + 10%
constraints	r_{con}	*mdrun* (page 183) `-rcon`	est. from bond lengths
virtual sites	r_{con}	*mdrun* (page 183) `-rcon`	0

In most cases the defaults of *mdrun* (page 183) 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 charge-groups in the starting configuration, with 10% added for headroom. For the constraints, the value of r_{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_{con}, but by the minimum cell size L_C, which has the following lower limit:

$$L_C \geq \max(r_{\text{mb}}, r_{\text{con}}) \tag{5.117}$$

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 $r_{\text{mb}} > r_c$, *mdrun* (page 183) employs a smart algorithm to reduce the communication. Simply communicating all charge groups within r_{mb} would increase the amount of communication enormously. Therefore only charge-groups that are connected by bonded interactions to charge 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_{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 394)). 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 527). 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.

mdrun (page 183) 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 183) `-ddorder pp_pme`. For

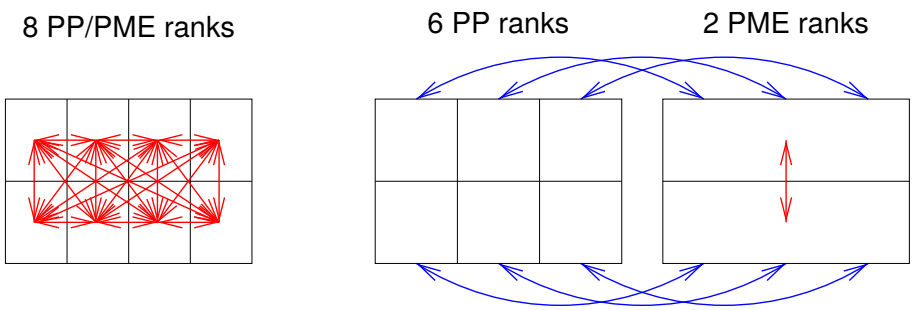


Fig. 5.14: Example of 8 ranks without (left) and with (right) MPMD. The PME communication (red arrows) is much higher on the left than on the right. For MPMD additional PP - PME coordinate and force communication (blue arrows) is required, but the total communication complexity is lower.

machines with a real 3-D torus and proper communication software that assigns the ranks accordingly one should use *mdrun* (page 183) `-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 166) will print an estimate for this load at the end and also *mdrun* (page 183) 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 440) settings and/or the number of PME ranks with the `-npme` option of *mdrun* (page 183). 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 183) 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 183) 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.

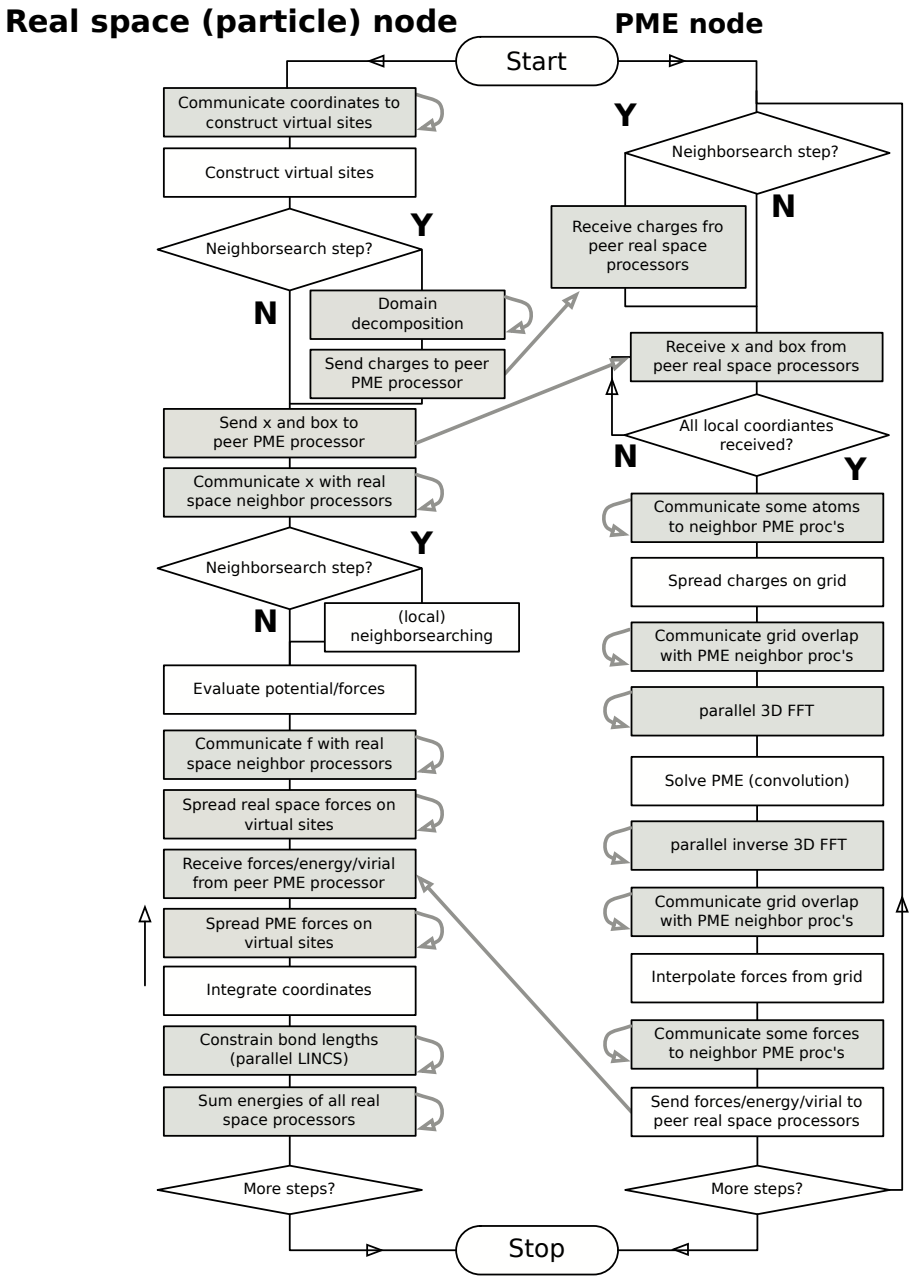


Fig. 5.15: Flow chart showing the algorithms and communication (arrows) for a standard MD simulation with virtual sites, constraints and separate PME-mesh ranks.

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 356)), 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 451).

5.5.1 Non-bonded interactions

Non-bonded interactions in GROMACS are pair-additive:

$$V(\mathbf{r}_1, \ldots \mathbf{r}_N) = \sum_{i<j} V_{ij}(\mathbf{r}_{ij}); \tag{5.118}$$

$$\mathbf{F}_i = -\sum_j \frac{dV_{ij}(r_{ij})}{dr_{ij}} \frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.119}$$

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:

$$V_{LJ}(r_{ij}) = \frac{C_{ij}^{(12)}}{{r_{ij}}^{12}} - \frac{C_{ij}^{(6)}}{{r_{ij}}^{6}} \tag{5.120}$$

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:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = -\left(12\,\frac{C_{ij}^{(12)}}{{r_{ij}}^{13}} - 6\,\frac{C_{ij}^{(6)}}{{r_{ij}}^{7}}\right)\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.121}$$

The LJ potential may also be written in the following form:

$$V_{LJ}(\mathbf{r}_{ij}) = 4\epsilon_{ij}\left(\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right) \tag{5.122}$$

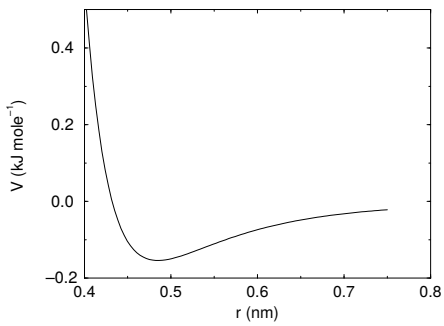


Fig. 5.16: The Lennard-Jones interaction.

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):

$$\begin{aligned} C_{ij}^{(6)} &= \left(C_{ii}^{(6)}\, C_{jj}^{(6)}\right)^{1/2} \\ C_{ij}^{(12)} &= \left(C_{ii}^{(12)}\, C_{jj}^{(12)}\right)^{1/2} \end{aligned} \tag{5.123}$$

or, alternatively the Lorentz-Berthelot rules can be used. An arithmetic average is used to calculate σ_{ij}, while a geometric average is used to calculate ϵ_{ij} (type 2):

$$\begin{aligned} \sigma_{ij} &= \tfrac{1}{2}(\sigma_{ii} + \sigma_{jj}) \\ \epsilon_{ij} &= (\epsilon_{ii}\, \epsilon_{jj})^{1/2} \end{aligned} \tag{5.124}$$

finally an geometric average for both parameters can be used (type 3):

$$\begin{aligned} \sigma_{ij} &= (\sigma_{ii}\, \sigma_{jj})^{1/2} \\ \epsilon_{ij} &= (\epsilon_{ii}\, \epsilon_{jj})^{1/2} \end{aligned} \tag{5.125}$$

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. The potential form is:

$$V_{bh}(r_{ij}) = A_{ij} \exp(-B_{ij} r_{ij}) - \frac{C_{ij}}{{r_{ij}}^6} \tag{5.126}$$

See also Fig. 5.17. The force derived from this is:

$$\mathbf{F}_i(r_{ij}) = \left[A_{ij} B_{ij} \exp(-B_{ij} r_{ij}) - 6 \frac{C_{ij}}{{r_{ij}}^7}\right] \frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.127}$$

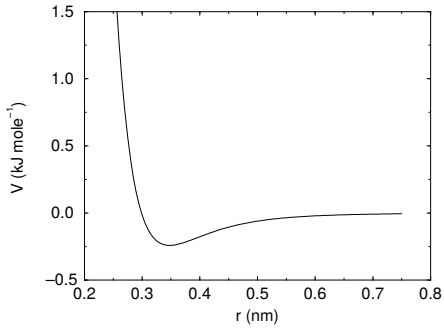


Fig. 5.17: The Buckingham interaction.

Coulomb interaction

The Coulomb interaction between two charge particles is given by:

$$V_c(r_{ij}) = f\frac{q_i q_j}{\varepsilon_r r_{ij}} \tag{5.128}$$

See also Fig. 5.18, where $f = \frac{1}{4\pi\varepsilon_0} = 138.935\,458$ (see chapter *Definitions and Units* (page 308))

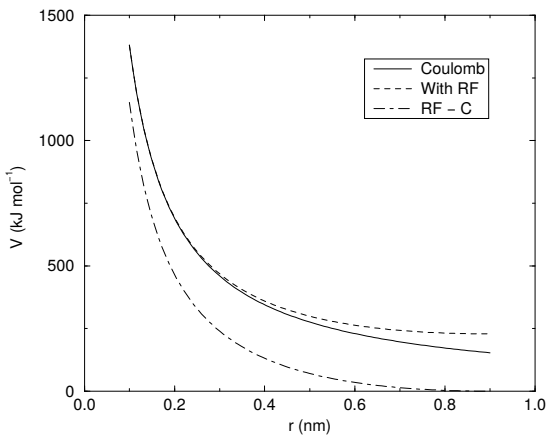


Fig. 5.18: The Coulomb interaction (for particles with equal signed charge) with and without reaction field. In the latter case ε_r was 1, ε_{rf} was 78, and r_c was 0.9 nm. The dot-dashed line is the same as the dashed line, except for a constant.

The force derived from this potential is:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = -f\frac{q_i q_j}{\varepsilon_r {r_{ij}}^2}\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.129}$$

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 ε_r may be set in the in the input for *grompp* (page 166).

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 ε_{rf}. The interaction then reads:

$$V_{crf} = f\frac{q_i q_j}{\varepsilon_r r_{ij}}\left[1+\frac{\varepsilon_{rf}-\varepsilon_r}{2\varepsilon_{rf}+\varepsilon_r}\frac{{r_{ij}}^3}{r_c^3}\right] - f\frac{q_i q_j}{\varepsilon_r r_c}\frac{3\varepsilon_{rf}}{2\varepsilon_{rf}+\varepsilon_r} \tag{5.130}$$

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 (5.130) for simplicity as

$$V_{crf} = f\frac{q_i q_j}{\varepsilon_r}\left[\frac{1}{r_{ij}} + k_{rf}\,{r_{ij}}^2 - c_{rf}\right] \tag{5.131}$$

with

$$k_{rf} = \frac{1}{r_c^3}\frac{\varepsilon_{rf}-\varepsilon_r}{(2\varepsilon_{rf}+\varepsilon_r)} \tag{5.132}$$

$$c_{rf} = \frac{1}{r_c} + k_{rf}\,r_c^2 = \frac{1}{r_c}\frac{3\varepsilon_{rf}}{(2\varepsilon_{rf}+\varepsilon_r)} \tag{5.133}$$

For large ε_{rf} the k_{rf} goes to ${r_c}^{-3}/2$, while for ε_{rf} = ε_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:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = -f\frac{q_i q_j}{\varepsilon_r}\left[\frac{1}{{r_{ij}}^2} - 2k_{rf}r_{ij}\right]\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.134}$$

The reaction-field correction should also be applied to all excluded atoms pairs, including self pairs, in which case the normal Coulomb term in (5.130) and (5.134) is absent.

Modified non-bonded interactions

In GROMACS, the non-bonded potentials can be modified by a shift function, also called a force-switch function, since it switches the force to zero at the cut-off. The purpose of this is to replace 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. But note that for Lennard-Jones interactions these errors are usually smaller than other errors, such as integration errors at the repulsive part of the potential. For Coulomb interactions we advise against using a shifted potential and for use of a reaction field or a proper long-range method such as PME.

There is *no* fundamental difference between a switch function (which multiplies the potential with a function) and a shift function (which adds a function to the force or potential) *72* (page 530). The switch function is a special case of the shift function, which we apply to the *force function* $F(r)$, related to the electrostatic or van der Waals force acting on particle i by particle j as:

$$\mathbf{F}_i = c\,F(r_{ij})\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.135}$$

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:

$$\begin{array}{rcll} F_s(r) & = & F_\alpha(r) & r < r_1 \\ F_s(r) & = & F_\alpha(r) + S(r) & r_1 \le r < r_c \\ F_s(r) & = & 0 & r_c \le r \end{array} \tag{5.136}$$

When $r_1 = 0$ this is a traditional shift function, otherwise it acts as a switch function. The corresponding shifted potential function then reads:

$$V_s(r) = \int_r^\infty F_s(x)\,dx \tag{5.137}$$

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:

$$\begin{array}{rcl} S_F(r_1) & = & 0 \\ S'_F(r_1) & = & 0 \\ S_F(r_c) & = & -F_\alpha(r_c) \\ S'_F(r_c) & = & -F'_\alpha(r_c) \end{array} \tag{5.138}$$

A 3rd degree polynomial of the form

$$S_F(r) = A(r-r_1)^2 + B(r-r_1)^3 \tag{5.139}$$

fulfills these requirements. The constants A and B are given by the boundary condition at r_c:

$$\begin{array}{rcl} A & = & -\alpha \dfrac{(\alpha+4)r_c \; - \; (\alpha+1)r_1}{r_c^{\alpha+2}\,(r_c-r_1)^2} \\ B & = & \alpha \dfrac{(\alpha+3)r_c \; - \; (\alpha+1)r_1}{r_c^{\alpha+2}\,(r_c-r_1)^3} \end{array} \tag{5.140}$$

Thus the total force function is:

$$F_s(r) = \frac{\alpha}{r^{\alpha+1}} + A(r-r_1)^2 + B(r-r_1)^3 \tag{5.141}$$

and the potential function reads:

$$V_s(r) = \frac{1}{r^\alpha} - \frac{A}{3}(r-r_1)^3 - \frac{B}{4}(r-r_1)^4 - C \tag{5.142}$$

where

$$C = \frac{1}{r_c^\alpha} - \frac{A}{3}(r_c-r_1)^3 - \frac{B}{4}(r_c-r_1)^4 \tag{5.143}$$

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:

$$\begin{array}{rcl} S_V(r_1) & = & 1 \\ S'_V(r_1) & = & 0 \\ S''_V(r_1) & = & 0 \\ S_V(r_c) & = & 0 \\ S'_V(r_c) & = & 0 \\ S''_V(r_c) & = & 0 \end{array} \tag{5.144}$$

The fifth-degree polynomial that has these properties is

$$S_V(r; r_1, r_c) = \frac{1 - 10(r-r_1)^3(r_c-r_1)^2 + 15(r-r_1)^4(r_c-r_1) - 6(r-r_1)}{(r_c-r_1)^5} \tag{5.145}$$

This implementation is found in several other simulation packages,*73* (page 530)*75* (page 530) but differs from that in CHARMM.*76* (page 530) 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 530) 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:

$$V(r) = f\frac{\mathrm{erfc}(\beta r_{ij})}{r_{ij}} q_i q_j, \tag{5.146}$$

where β is a parameter that determines the relative weight between the direct space sum and the reciprocal space sum and $\mathrm{erfc}(x)$ is the complementary error function. For further details on long-range electrostatics, see sec. *Long Range Electrostatics* (page 393).

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:

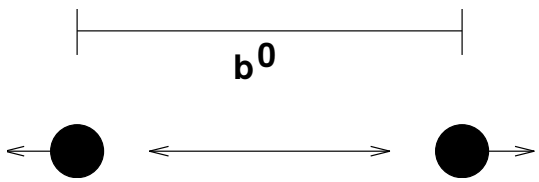


Fig. 5.19: Principle of bond stretching (left), and the bond stretching potential (right).

$$V_b\left(r_{ij}\right) = \frac{1}{2} k_{ij}^b (r_{ij} - b_{ij})^2 \tag{5.147}$$

See also Fig. 5.19, with the force given by:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = k_{ij}^b (r_{ij} - b_{ij}) \frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.148}$$

Fourth power potential

In the GROMOS-96 force field *77* (page 530), the covalent bond potential is, for reasons of computational efficiency, written as:

$$V_b\left(r_{ij}\right) = \frac{1}{4} k_{ij}^b \left({r_{ij}}^2 - b_{ij}^2\right)^2 \tag{5.149}$$

The corresponding force is:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = k_{ij}^b ({r_{ij}}^2 - b_{ij}^2)\, \mathbf{r}_i j \tag{5.150}$$

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 361)) as

$$2 k^b b_{ij}^2 = k^{b,\mathrm{harm}} \tag{5.151}$$

The force constants are mostly derived from the harmonic ones used in GROMOS-87 *78* (page 530). 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 530) 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:

$$V_{morse}(r_{ij}) = D_{ij}[1 - \exp(-\beta_{ij}(r_{ij} - b_{ij}))]^2, \tag{5.152}$$

See also Fig. 5.20, and the corresponding force is:

$$\begin{aligned} \mathbf{F}_{morse}(\mathbf{r}_{ij}) &= 2D_{ij}\beta_{ij}\exp(-\beta_{ij}(r_{ij} - b_{ij}))* \\ &\quad [1 - \exp(-\beta_{ij}(r_{ij} - b_{ij}))]\frac{\mathbf{r}_{ij}}{r_{ij}}, \end{aligned} \tag{5.153}$$

where D_{ij} is the depth of the well in kJ/mol, β_{ij} defines the steepness of the well (in nm^{-1}), and b_{ij} is the equilibrium distance in nm. The steepness parameter β_{ij} can be expressed in terms of the reduced mass of the atoms *i* and *j*, the fundamental vibration frequency ω_{ij} and the well depth D_{ij}:

$$\beta_{ij} = \omega_{ij}\sqrt{\frac{\mu_{ij}}{2D_{ij}}} \tag{5.154}$$

and because $\omega = \sqrt{k/\mu}$, one can rewrite β_{ij} in terms of the harmonic force constant k_{ij}:

$$\beta_{ij} = \sqrt{\frac{k_{ij}}{2D_{ij}}} \tag{5.155}$$

For small deviations $(r_{ij} - b_{ij})$, one can approximate the exp-term to first-order using a Taylor expansion:

$$\exp(-x) \approx 1 - x \tag{5.156}$$

and substituting (5.155) and (5.156) in the functional form:

$$\begin{aligned} V_{morse}(r_{ij}) &= D_{ij}[1 - \exp(-\beta_{ij}(r_{ij} - b_{ij}))]^2 \\ &= D_{ij}[1 - (1 - \sqrt{\tfrac{k_{ij}}{2D_{ij}}}(r_{ij} - b_{ij}))]^2 \\ &= \tfrac{1}{2}k_{ij}(r_{ij} - b_{ij}))^2 \end{aligned} \tag{5.157}$$

we recover the harmonic bond stretching potential.

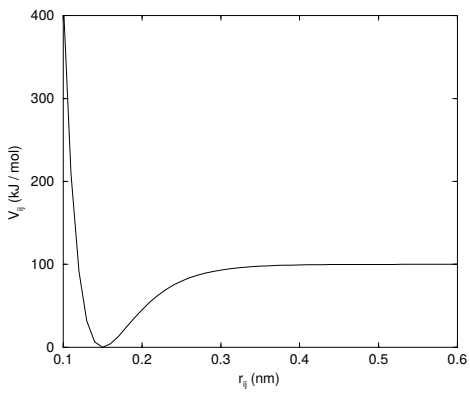


Fig. 5.20: The Morse potential well, with bond length 0.15 nm.

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:

$$V_b\left(r_{ij}\right) = k_{ij}^b(r_{ij} - b_{ij})^2 + k_{ij}^b k_{ij}^{cub}(r_{ij} - b_{ij})^3 \tag{5.158}$$

A flexible water model (based on the SPC water model *80* (page 530)) including a cubic bond stretching potential for the O-H bond was developed by Ferguson *81* (page 530). 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: over-stretching leads to infinitely low energies. The integration timestep is therefore limited to 1 fs.

The force corresponding to this potential is:

$$\mathbf{F}_i(\mathbf{r}_{ij}) = 2k_{ij}^b(r_{ij} - b_{ij})\,\frac{\mathbf{r}_{ij}}{r_{ij}} + 3k_{ij}^b k_{ij}^{cub}(r_{ij} - b_{ij})^2\,\frac{\mathbf{r}_{ij}}{r_{ij}} \tag{5.159}$$

FENE bond stretching potential

In coarse-grained polymer simulations the beads are often connected by a FENE (finitely extensible nonlinear elastic) potential *82* (page 530):

$$V_{\text{FENE}}(r_{ij}) = -\frac{1}{2}k_{ij}^b b_{ij}^2 \log\left(1 - \frac{{r_{ij}}^2}{b_{ij}^2}\right) \tag{5.160}$$

The potential looks complicated, but the expression for the force is simpler:

$$F_{\text{FENE}}(\mathbf{r}_{ij}) = -k_{ij}^b\left(1 - \frac{{r_{ij}}^2}{b_{ij}^2}\right)^{-1}\mathbf{r}_{ij} \tag{5.161}$$

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 is also represented by a harmonic potential on the angle θ_{ijk}

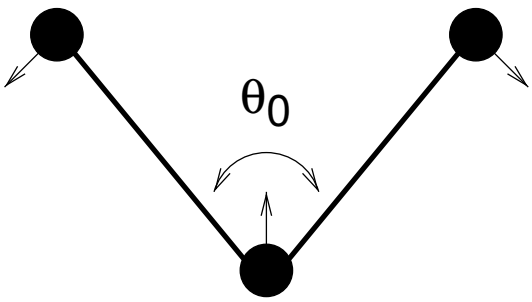


Fig. 5.21: Principle of angle vibration (left) and the bond angle potential.

$$V_a(\theta_{ijk}) = \frac{1}{2}k_{ijk}^{\theta}(\theta_{ijk} - {\theta_{ijk}}^0)^2 \tag{5.162}$$

As the bond-angle vibration is represented by a harmonic potential, the form is the same as the bond stretching (Fig. 5.19).

The force equations are given by the chain rule:

$$\begin{aligned}\mathbf{F}_i &= -\frac{dV_a(\theta_{ijk})}{d\mathbf{r}_i} \\ \mathbf{F}_k &= -\frac{dV_a(\theta_{ijk})}{d\mathbf{r}_k} \qquad \text{where} \quad \theta_{ijk} = \arccos\frac{(\mathbf{r}_{ij}\cdot\mathbf{r}_{kj})}{r_{ij}r_{kj}} \\ \mathbf{F}_j &= -\mathbf{F}_i - \mathbf{F}_k\end{aligned} \tag{5.163}$$

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:

$$V_a(\theta_{ijk}) = \frac{1}{2}k^{\theta}_{ijk}\left(\cos(\theta_{ijk}) - \cos({\theta_{ijk}}^0)\right)^2 \tag{5.164}$$

where

$$\cos(\theta_{ijk}) = \frac{\mathbf{r}_{ij}\cdot\mathbf{r}_{kj}}{r_{ij}r_{kj}} \tag{5.165}$$

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 363)) by:

$$k^{\theta}\sin^2({\theta_{ijk}}^0) = k^{\theta,\mathrm{harm}} \tag{5.166}$$

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 530) prevents the bending angle θ from reaching the $180°$ 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 $180°$ value, the bending potential (5.164) is divided by a $\sin^2\theta$ factor:

$$V_{\mathrm{ReB}}(\theta_i) = \frac{1}{2}k_\theta\frac{(\cos\theta_i - \cos\theta_0)^2}{\sin^2\theta_i}. \tag{5.167}$$

Figure 5.22 shows the comparison between the ReB potential, (5.167), and the standard one (5.164).

The wall of the ReB potential is very repulsive in the region close to $180°$ 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:

$$F_{\mathrm{ReB}}(\theta_i) = \frac{2k_\theta}{\sin^4\theta_i}(\cos\theta_i - \cos\theta_0)(1 - \cos\theta_i\cos\theta_0)\frac{\partial\cos\theta_i}{\partial\vec{r}_k}. \tag{5.168}$$

Due to its construction, the restricted bending potential cannot be used for equilibrium θ_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 have to be 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).

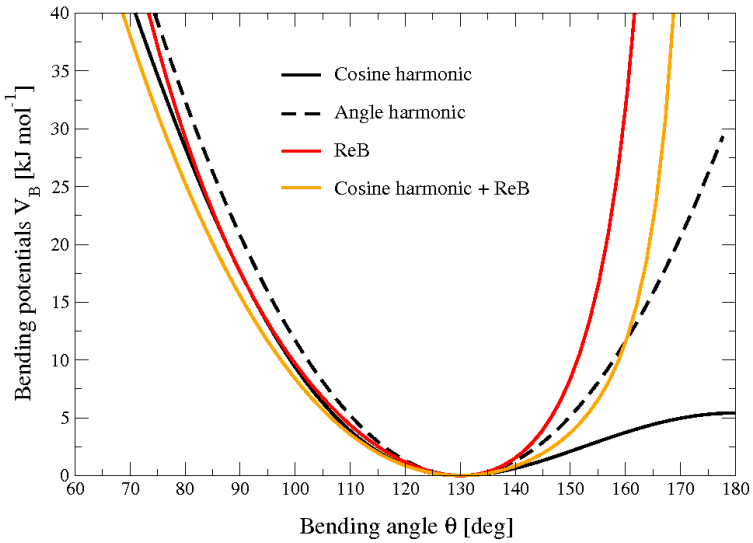


Fig. 5.22: Bending angle potentials: cosine harmonic (solid black line), angle harmonic (dashed black line) and restricted bending (red) with the same bending constant $k_\theta = 85$ kJ mol^{-1} and equilibrium angle $\theta_0 = 130°$. The orange line represents the sum of a cosine harmonic ($k = 50$ kJ mol^{-1}) with a restricted bending ($k = 25$ kJ mol^{-1}) potential, both with $\theta_0 = 130°$.

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 θ_{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 530). The energy is given by:

$$V_a(\theta_{ijk}) = \frac{1}{2}k^{\theta}_{ijk}(\theta_{ijk} - {\theta_{ijk}}^0)^2 + \frac{1}{2}k^{UB}_{ijk}(r_{ik} - r^0_{ik})^2 \tag{5.169}$$

The force equations can be deduced from sections *Harmonic potential* (page 361) and *Harmonic angle potential* (page 363).

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 531):

$$V_{rr'} \;=\; k_{rr'}\left(|\mathbf{r}_i - \mathbf{r}_j| - r_{1e}\right)\left(|\mathbf{r}_k - \mathbf{r}_j| - r_{2e}\right) \tag{5.170}$$

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:

$$\mathbf{F}_i = -k_{rr'}\left(|\mathbf{r}_k - \mathbf{r}_j| - r_{2e}\right)\frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} \tag{5.171}$$

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 531):

$$V_{r\theta} \;=\; k_{r\theta}\left(|\mathbf{r}_i - \mathbf{r}_k| - r_{3e}\right)\left(|\mathbf{r}_i - \mathbf{r}_j| - r_{1e} + |\mathbf{r}_k - \mathbf{r}_j| - r_{2e}\right) \tag{5.172}$$

where $k_{r\theta}$ is the force constant, r_{3e} is the $i - k$ distance, and the other constants are the same as in (5.170). The force associated with the potential on atom i is:

$$\mathbf{F}_i \;=\; -k_{r\theta}\left[\left(|\mathbf{r}_i - \mathbf{r}_k| - r_{3e}\right)\frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} + \left(|\mathbf{r}_i - \mathbf{r}_j| - r_{1e} + |\mathbf{r}_k - \mathbf{r}_j| - r_{2e}\right)\frac{\mathbf{r}_i - \mathbf{r}_k}{|\mathbf{r}_i - \mathbf{r}_k|}\right] \tag{5.173}$$

Quartic angle potential

For special purposes there is an angle potential that uses a fourth order polynomial:

$$V_q(\theta_{ijk}) \;=\; \sum_{n=0}^{5} C_n(\theta_{ijk} - {\theta_{ijk}}^0)^n \tag{5.174}$$

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.

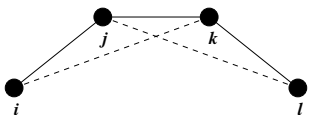


Fig. 5.23: Principle of improper dihedral angles. Out of plane bending for rings. The improper dihedral angle ξ is defined as the angle between planes (i,j,k) and (j,k,l).

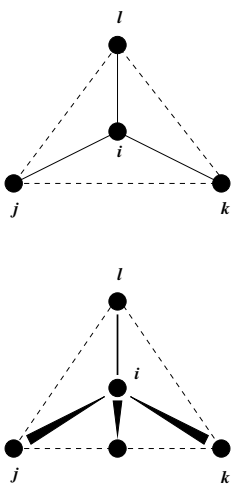


Fig. 5.24: Principle of improper dihedral angles. Out of tetrahedral angle. The improper dihedral angle ξ is defined as the angle between planes (i,j,k) and (j,k,l).

Improper dihedrals: harmonic type

The simplest improper dihedral potential is a harmonic potential; it is plotted in Fig. 5.25.

$$V_{id}(\xi_{ijkl}) = \frac{1}{2}k_\xi(\xi_{ijkl} - \xi_0)^2 \tag{5.175}$$

Since the potential is harmonic it is discontinuous, but since the discontinuity is chosen at 180° distance from ξ_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.

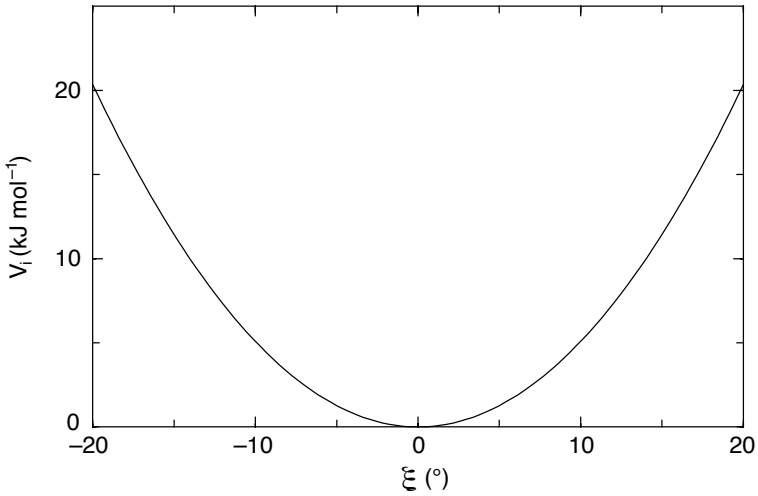


Fig. 5.25: Improper dihedral potential.

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 200) in this case.

Proper dihedrals: periodic type

Proper dihedral angles are defined according to the IUPAC/IUB convention, where ϕ 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.

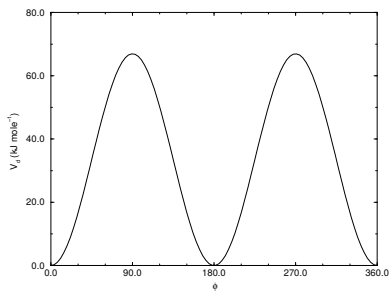


Fig. 5.26: Principle of proper dihedral angle (left, in *trans* form) and the dihedral angle potential (right).

$$V_d(\phi_{ijkl}) = k_\phi(1 + \cos(n\phi - \phi_s)) \tag{5.176}$$

Proper dihedrals: Ryckaert-Bellemans function

For alkanes, the following proper dihedral potential is often used (see Fig. 5.27):

$$V_{rb}(\phi_{ijkl}) = \sum_{n=0}^{5} C_n(\cos(\psi))^n, \tag{5.177}$$

where $\psi = \phi - 180°$.

Note: A conversion from one convention to another can be achieved by multiplying every coefficient C_n by $(-1)^n$.

An example of constants for C is given in Table 5.8.

Table 5.8: Constants for Ryckaert-Bellemans potential (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 ψ is defined according to the "polymer convention" ($\psi_{trans} = 0$).)

The RB dihedral function can also be used to include Fourier dihedrals (see below):

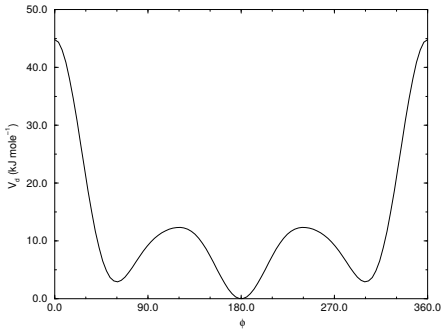


Fig. 5.27: Ryckaert-Bellemans dihedral potential.

$$V_{rb}(\phi_{ijkl}) \;=\; \frac{1}{2}\left[F_1(1+\cos(\phi)) + F_2(1-\cos(2\phi)) + F_3(1+\cos(3\phi)) + F_4(1-\cos(4\phi))\right] \tag{5.178}$$

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:

$$\begin{array}{rcl} C_0 & = & F_2 + \frac{1}{2}(F_1 + F_3) \\ C_1 & = & \frac{1}{2}(-F_1 + 3\,F_3) \\ C_2 & = & -F_2 + 4\,F_4 \\ C_3 & = & -2\,F_3 \\ C_4 & = & -4\,F_4 \\ C_5 & = & 0 \end{array} \tag{5.179}$$

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 531) or four *87* (page 531) cosine terms of a Fourier series. In GROMACS the four term function is implemented:

$$V_F(\phi_{ijkl}) \;=\; \frac{1}{2}\left[C_1(1+\cos(\phi)) + C_2(1-\cos(2\phi)) + C_3(1+\cos(3\phi)) + C_4(1-\cos(4\phi))\right], \tag{5.180}$$

Internally, GROMACS uses the Ryckaert-Bellemans code to compute Fourier dihedrals (see above), because this is more efficient.

Note: Mind the conversion from *k*cal 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 364)), a restricted torsion/dihedral potential is introduced:

$$V_{\mathrm{ReT}}(\phi_i) = \frac{1}{2} k_\phi \frac{(\cos\phi_i - \cos\phi_0)^2}{\sin^2\phi_i} \tag{5.181}$$

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 ϕ_0 value should not be too close to the interval limits (as for the restricted bending potential, described in *Restricted bending potential* (page 364), 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 364)) 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:

$$V_{\mathrm{CBT}}(\theta_{i-1}, \theta_i, \phi_i) = k_\phi \sin^3\theta_{i-1} \sin^3\theta_i \sum_{n=0}^{4} a_n \cos^n\phi_i. \tag{5.182}$$

This combined bending-torsion (CBT) potential has been proposed by *88* (page 531) for polymer melt simulations and is extensively described in *83* (page 530).

This potential has two main advantages:

- it does not only depend on the dihedral angle ϕ_i (between the $i-2$, $i-1$, i and $i+1$ beads) but also on the bending angles θ_{i-1} and θ_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 531) and theoretically discussed by *90* (page 531), cancel the torsion potential and force when either of the two bending angles approaches the value of $180°$.
- its dependence on ϕ_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 θ_{i-1} and θ_i may have any form. It is also possible to leave out the two angle bending terms (θ_{i-1} and θ_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:

$$\frac{\partial V_{\mathrm{CBT}}(\theta_{i-1}, \theta_i, \phi_i)}{\partial \vec{r}_l} = \frac{\partial V_{\mathrm{CBT}}}{\partial \theta_{i-1}} \frac{\partial \theta_{i-1}}{\partial \vec{r}_l} + \frac{\partial V_{\mathrm{CBT}}}{\partial \theta_i} \frac{\partial \theta_i}{\partial \vec{r}_l} + \frac{\partial V_{\mathrm{CBT}}}{\partial \phi_i} \frac{\partial \phi_i}{\partial \vec{r}_l} \tag{5.183}$$

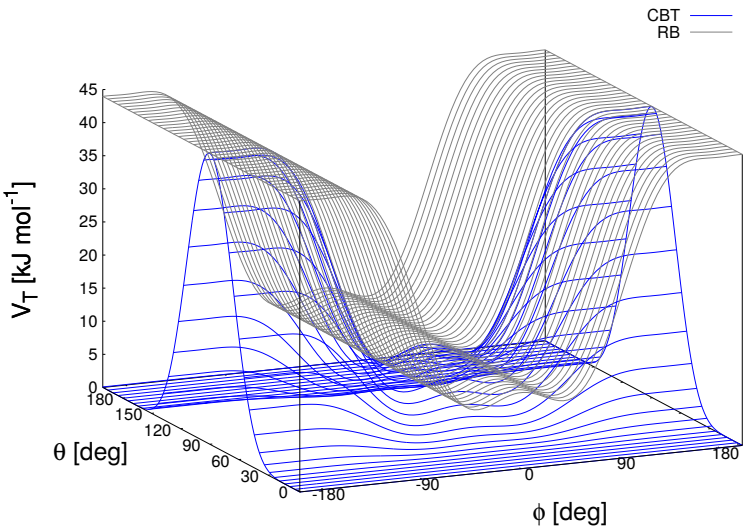


Fig. 5.28: Blue: surface plot of the combined bending-torsion potential ((5.182) with $k = 10$ kJ mol^{-1}, $a_0 = 2.41$, $a_1 = -2.95$, $a_2 = 0.36$, $a_3 = 1.33$) when, for simplicity, the bending angles behave the same ($\theta_1 = \theta_2 = \theta$). Gray: the same torsion potential without the $\sin^3\theta$ terms (Ryckaert-Bellemans type). ϕ 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 (θ_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 θ and ϕ 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 (θ is not $180°$ and ϕ is very close to $180°$).

Tabulated bonded interaction functions

For full flexibility, any functional shape can be used for bonds, angles and dihedrals through user-supplied tabulated functions. The functional shapes are:

$$\begin{aligned} V_b(r_{ij}) &= k\, f_n^b(r_{ij}) \\ V_a(\theta_{ijk}) &= k\, f_n^a(\theta_{ijk}) \\ V_d(\phi_{ijkl}) &= k\, f_n^d(\phi_{ijkl}) \end{aligned} \tag{5.184}$$

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 485). 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 183) 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 183) 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:

$$V_{pr}(\mathbf{r}_i) = \frac{1}{2} k_{pr} |\mathbf{r}_i - \mathbf{R}_i|^2 \tag{5.185}$$

The potential is plotted in Fig. 5.29.

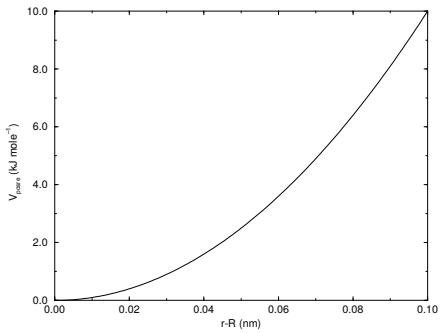


Fig. 5.29: Position restraint potential.

The potential form can be rewritten without loss of generality as:

$$V_{pr}(\mathbf{r}_i) = \frac{1}{2} \left[k_{pr}^x (x_i - X_i)^2 \, \hat{\mathbf{x}} + k_{pr}^y (y_i - Y_i)^2 \, \hat{\mathbf{y}} + k_{pr}^z (z_i - Z_i)^2 \, \hat{\mathbf{z}} \right] \tag{5.186}$$

Now the forces are:

$$\begin{aligned} F_i^x &= -k_{pr}^x \, (x_i - X_i) \\ F_i^y &= -k_{pr}^y \, (y_i - Y_i) \\ F_i^z &= -k_{pr}^z \, (z_i - Z_i) \end{aligned} \tag{5.187}$$

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 200) 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 49), 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):

$$V_{\mathrm{fb}}(\mathbf{r}_i) = \frac{1}{2} k_{\mathrm{fb}} [d_g(\mathbf{r}_i; \mathbf{R}_i) - r_{\mathrm{fb}}]^2 \, H[d_g(\mathbf{r}_i; \mathbf{R}_i) - r_{\mathrm{fb}}], \tag{5.188}$$

where $\mathbf{R}_i$ is the reference position, r_{fb} is the distance from the center with a flat potential, k_{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.

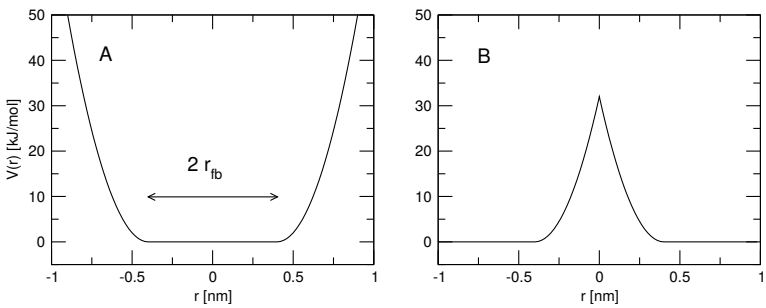


Fig. 5.30: Flat-bottomed position restraint potential. (A) Not inverted, (B) inverted.

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:

$$d_g(\mathbf{r}_i; \mathbf{R}_i) = |\mathbf{r}_i - \mathbf{R}_i| \tag{5.189}$$

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 446) 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:

$$d_g(\mathbf{r}_i; \mathbf{R}_i) = \sqrt{(x_i - X_i)^2 + (y_i - Y_i)^2} \tag{5.190}$$

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.

$$d_g(\mathbf{r}_i; \mathbf{R}_i) = |x_i - X_i|, \quad \text{or} \quad d_g(\mathbf{r}_i; \mathbf{R}_i) = |y_i - Y_i|, \quad \text{or} \quad d_g(\mathbf{r}_i; \mathbf{R}_i) = |z_i - Z_i|. \tag{5.191}$$

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_{fb}. That feature is switched on by defining a negative r_{fb} in the topology. The following potential is used (Figure 5.30 B):

$$V_{\text{fb}}^{\text{inv}}(\mathbf{r}_i) = \frac{1}{2} k_{\text{fb}} [d_g(\mathbf{r}_i; \mathbf{R}_i) - |r_{\text{fb}}|]^2 \, H[-(d_g(\mathbf{r}_i; \mathbf{R}_i) - |r_{\text{fb}}|)]. \tag{5.192}$$

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:

$$V_{ar}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k, \mathbf{r}_l) = k_{ar}(1 - \cos(n(\theta - \theta_0))), \quad \text{where} \;\; \theta = \arccos\left(\frac{\mathbf{r}_j - \mathbf{r}_i}{\|\mathbf{r}_j - \mathbf{r}_i\|} \cdot \frac{\mathbf{r}_l - \mathbf{r}_k}{\|\mathbf{r}_l - \mathbf{r}_k\|}\right) \tag{5.193}$$

For one pair of atoms and the z-axis:

$$V_{ar}(\mathbf{r}_i, \mathbf{r}_j) = k_{ar}(1 - \cos(n(\theta - \theta_0))), \quad \text{where} \;\; \theta = \arccos\left(\frac{\mathbf{r}_j - \mathbf{r}_i}{\|\mathbf{r}_j - \mathbf{r}_i\|} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\right) \tag{5.194}$$

A multiplicity (n) of 2 is useful when you do not want to distinguish between parallel and anti-parallel vectors. The equilibrium angle θ 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 ϕ defined by four particles as in an improper dihedral (sec. *Improper dihedrals* (page 366)) but with a slightly modified potential. Using:

$$\phi' = (\phi - \phi_0) \;\; \text{MOD} \; 2\pi \tag{5.195}$$

where ϕ_0 is the reference angle, the potential is defined as:

$$V_{dihr}(\phi') \;=\; \begin{cases} \frac{1}{2} k_{dihr} (\phi' - \Delta\phi)^2 & \text{for} \quad \|\phi'\| \;>\; \Delta\phi \\ 0 & \text{for} \quad \|\phi'\| \;\leq\; \Delta\phi \end{cases} \tag{5.196}$$

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 409)).
- 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).

$$V_{dr}(r_{ij}) = \begin{cases} \frac{1}{2}k_{dr}(r_{ij}-r_0)^2 & \text{for} \quad r_{ij} < r_0 \\ 0 & \text{for} \quad r_0 \leq r_{ij} < r_1 \\ \frac{1}{2}k_{dr}(r_{ij}-r_1)^2 & \text{for} \quad r_1 \leq r_{ij} < r_2 \\ \frac{1}{2}k_{dr}(r_2-r_1)(2r_{ij}-r_2-r_1) & \text{for} \quad r_2 \leq r_{ij} \end{cases} \tag{5.197}$$

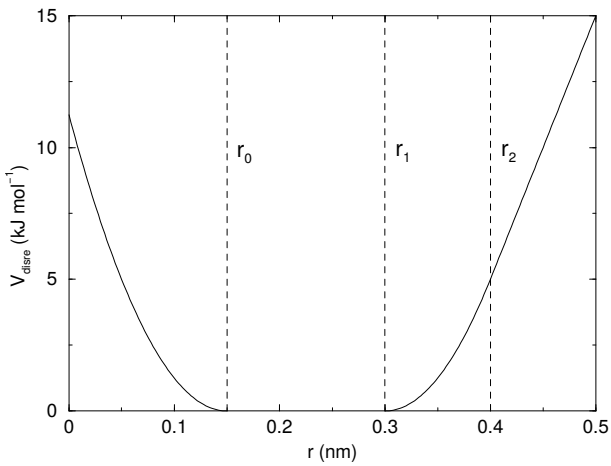


Fig. 5.31: Distance Restraint potential.

The forces are

$$\mathbf{F}_i = \begin{cases} -k_{dr}(r_{ij}-r_0)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_{ij} < r_0 \\ 0 & \text{for} \quad r_0 \leq r_{ij} < r_1 \\ -k_{dr}(r_{ij}-r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_1 \leq r_{ij} < r_2 \\ -k_{dr}(r_2-r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_2 \leq r_{ij} \end{cases} \tag{5.198}$$

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 418). 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 531). The forces with time averaging are:

$$\mathbf{F}_i = \begin{cases} -k^a_{dr}(\bar{r}_{ij} - r_0)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad \bar{r}_{ij} < r_0 \\ 0 & \text{for} \quad r_0 \leq \bar{r}_{ij} < r_1 \\ -k^a_{dr}(\bar{r}_{ij} - r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_1 \leq \bar{r}_{ij} < r_2 \\ -k^a_{dr}(r_2 - r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_2 \leq \bar{r}_{ij} \end{cases} \tag{5.199}$$

where $\bar{r}_{ij}$ is given by an exponential running average with decay time τ:

$$\bar{r}_{ij} = < r_{ij}^{-3} >^{-1/3} \tag{5.200}$$

The force constant k^a_{dr} is switched on slowly to compensate for the lack of history at the beginning of the simulation:

$$k^a_{dr} = k_{dr}\left(1 - \exp\left(-\frac{t}{\tau}\right)\right) \tag{5.201}$$

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 χ 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 183) program is done in the following fashion:

$$\begin{aligned} \overline{r^{-3}}_{ij}(0) &= r_{ij}(0)^{-3} \\ \overline{r^{-3}}_{ij}(t) &= \overline{r^{-3}}_{ij}(t - \Delta t)\ \exp\left(-\tfrac{\Delta t}{\tau}\right) + r_{ij}(t)^{-3}\left[1 - \exp\left(-\tfrac{\Delta t}{\tau}\right)\right] \end{aligned} \tag{5.202}$$

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:

$$\mathbf{F}_i = \begin{cases} k^a_{dr}\sqrt{(r_{ij} - r_0)(\bar{r}_{ij} - r_0)}\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_{ij} < r_0 \quad \text{and} \quad \bar{r}_{ij} < r_0 \\ -k^a_{dr}\min\left(\sqrt{(r_{ij} - r_1)(\bar{r}_{ij} - r_1)}, r_2 - r_1\right)\frac{\mathbf{r}_{ij}}{r_{ij}} & \text{for} \quad r_{ij} > r_1 \quad \text{and} \quad \bar{r}_{ij} > r_1 \\ 0 & \text{otherwise} \end{cases} \tag{5.203}$$

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 interatomic distance. Thus, when combining atom pairs, a fixed list of N restraints may be taken together, where the apparent "distance" is given by:

$$r_N(t) = \left[\sum_{n=1}^{N} \bar{r}_n(t)^{-6}\right]^{-1/6} \tag{5.204}$$

where we use r_{ij} or (5.200) 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:

$$\begin{aligned} r_1 &= r_1 * M^{-1/6} \\ r_2 &= r_2 * M^{-1/6} \end{aligned} \tag{5.205}$$

where M is the number of molecules. The GROMACS preprocessor *grompp* (page 166) can do this automatically when the appropriate option is given. The resulting "distance" is then used to calculate the scalar force according to:

$$\mathbf{F}_i = \begin{cases} 0 & r_N < r_1 \\ k_{dr}(r_N - r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & r_1 \leq r_N < r_2 \\ k_{dr}(r_2 - r_1)\frac{\mathbf{r}_{ij}}{r_{ij}} & r_N \geq r_2 \end{cases} \tag{5.206}$$

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
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 375), 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 (5.197). 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 193).

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 531).

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:

$$\delta_i = \frac{2}{3}\text{tr}(\mathbf{S}\mathbf{D}_i) \tag{5.207}$$

where $\mathbf{S}$ is the dimensionless order tensor of the molecule. The tensor $\mathbf{D}_i$ is given by:

$$\mathbf{D}_i = \frac{c_i}{\|\mathbf{r}_i\|^\alpha}\begin{pmatrix} 3xx-1 & 3xy & 3xz \\ 3xy & 3yy-1 & 3yz \\ 3xz & 3yz & 3zz-1 \end{pmatrix} \tag{5.208}$$

$$\text{with:} \quad x = \frac{r_{i,x}}{\|\mathbf{r}_i\|}, \quad y = \frac{r_{i,y}}{\|\mathbf{r}_i\|}, \quad z = \frac{r_{i,z}}{\|\mathbf{r}_i\|} \tag{5.209}$$

For a dipolar coupling $\mathbf{r}_i$ is the vector connecting the two nuclei, $\alpha = 3$ and the constant c_i is given by:

$$c_i = \frac{\mu_0}{4\pi}\gamma_1^i\gamma_2^i\frac{\hbar}{4\pi} \tag{5.210}$$

where γ_1^i and γ_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:

$$\mathbf{T}^T\mathbf{S}\mathbf{T} = s\begin{pmatrix} -\frac{1}{2}(1-\eta) & 0 & 0 \\ 0 & -\frac{1}{2}(1+\eta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.211}$$

where $-1 \leq s \leq 1$ and $0 \leq \eta \leq 1$. s is called the order parameter and η 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 δ_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:

$$\mathbf{D}_i^c(t) = \mathbf{R}(t)\mathbf{D}_i(t)\mathbf{R}^T(t) \tag{5.212}$$

The calculated orientation for vector i is given by:

$$\delta_i^c(t) = \frac{2}{3}\text{tr}(\mathbf{S}(t)\mathbf{D}_i^c(t)) \tag{5.213}$$

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:

$$MSD(t) = \left(\sum_{i=1}^{N} w_i\right)^{-1} \sum_{i=1}^{N} w_i(\delta_i^c(t) - \delta_i^{exp})^2 \tag{5.214}$$

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 δ_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 (5.212)). The time-averaged tensors are calculated using an exponentially decaying memory function:

$$\mathbf{D}_i^a(t) = \frac{\displaystyle\int_{u=t_0}^{t} \mathbf{D}_i^c(u) \exp\left(-\frac{t-u}{\tau}\right) \mathrm{d}u}{\displaystyle\int_{u=t_0}^{t} \exp\left(-\frac{t-u}{\tau}\right) \mathrm{d}u} \tag{5.215}$$

Assuming that the order tensor $\mathbf{S}$ fluctuates slower than the $\mathbf{D}_i$, the time-averaged orientation can be calculated as:

$$\delta_i^a(t) = \frac{2}{3}\mathrm{tr}(\mathbf{S}(t)\mathbf{D}_i^a(t)) \tag{5.216}$$

where the order tensor $\mathbf{S}(t)$ is calculated using expression (5.214) 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:

$$V = \frac{1}{2}k \sum_{i=1}^{N} w_i(\delta_i^c(t) - \delta_i^{exp})^2 \tag{5.217}$$

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:

$$\begin{aligned}
\mathbf{F}_i(t) &= -\frac{\mathrm{d}V}{\mathrm{d}\mathbf{r}_i} \\
&= -kw_i(\delta_i^c(t) - \delta_i^{exp})\frac{\mathrm{d}\delta_i(t)}{\mathrm{d}\mathbf{r}_i} \\
&= -kw_i(\delta_i^c(t) - \delta_i^{exp})\frac{2c_i}{\|\mathbf{r}\|^{2+\alpha}}\left(2\mathbf{R}^T\mathbf{S}\mathbf{R}\mathbf{r}_i - \frac{2+\alpha}{\|\mathbf{r}\|^2}\mathrm{tr}(\mathbf{R}^T\mathbf{S}\mathbf{R}\mathbf{r}_i\mathbf{r}_i^T)\mathbf{r}_i\right)
\end{aligned} \tag{5.218}$$

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:

$$V = M\frac{1}{2}k\sum_{i=1}^{N} w_i \langle \delta_i^c(t) - \delta_i^{exp}\rangle^2 \tag{5.219}$$

The force on vector $\mathbf{r}_{i,m}$ in subsystem m is given by:

$$\mathbf{F}_{i,m}(t) = -\frac{\mathrm{d}V}{\mathrm{d}\mathbf{r}_{i,m}} = -kw_i\langle \delta_i^c(t) - \delta_i^{exp}\rangle \frac{\mathrm{d}\delta_{i,m}^c(t)}{\mathrm{d}\mathbf{r}_{i,m}} \tag{5.220}$$

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:

$$V = M\frac{1}{2}k^a\sum_{i=1}^{N} w_i \langle \delta_i^a(t) - \delta_i^{exp}\rangle^2 \tag{5.221}$$

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:

$$k^a = k\,\frac{1}{\tau}\int_{u=t_0}^{t} \exp\left(-\frac{t-u}{\tau}\right)\mathrm{d}u \tag{5.222}$$

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:

$$\mathbf{F}_{i,m}(t) = \begin{cases} 0 & \text{for} \quad a\,b \le 0 \\ k^a w_i \dfrac{a}{|a|}\sqrt{a\,b}\,\dfrac{\mathrm{d}\delta_{i,m}^c(t)}{\mathrm{d}\mathbf{r}_{i,m}} & \text{for} \quad a\,b > 0 \end{cases} \tag{5.223}$$

$$\begin{aligned} a = &\quad \langle \delta_i^a(t) - \delta_i^{exp}\rangle \\ b = &\quad \langle \delta_i^c(t) - \delta_i^{exp}\rangle \end{aligned} \tag{5.224}$$

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 α that is used in (5.208)) 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 166) *mdp* (page 440) file, for a study of the effect of different force constants and averaging times and ensemble averaging see *92* (page 531). Information for each restraint is stored in the energy file and can be processed and plotted with *gmx nmr* (page 193).

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 α (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 529):

$$k_{cs} = \frac{q_s^2}{\alpha} \tag{5.225}$$

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 531) it was found that some particles can overpolarize and this was fixed by introducing a higher order term in the polarization energy:

$$\begin{aligned} V_{pol} &= \frac{k_{cs}}{2} r_{cs}^2 & r_{cs} \leq \delta \\ &= \frac{k_{cs}}{2} r_{cs}^2 + k_{hyp}(r_{cs} - \delta)^4 & r_{cs} > \delta \end{aligned} \tag{5.226}$$

where δ is a user-defined constant that is set to 0.02 nm for anions in the Drude force field *94* (page 531). Since this original introduction it has also been used in other atom types *93* (page 531).

```
[ 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 529).

Thole polarization

Based on early work by Thole *95* (page 531), Roux and coworkers have implemented potentials for molecules like ethanol *96* (page 531)*98* (page 531). 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:

$$V_{thole} = \frac{q_i q_j}{r_{ij}} \left[1 - \left(1 + \frac{\bar{r}_{ij}}{2} \right) \exp^{-\bar{r}_{ij}} \right] \tag{5.227}$$

Note that there is a sign error in Equation 1 of Noskov *et al.* *98* (page 531):

$$\bar{r}_{ij} = a\frac{r_{ij}}{(\alpha_i\alpha_j)^{1/6}} \tag{5.228}$$

where a is a magic (dimensionless) constant, usually chosen to be 2.6 *98* (page 531); α_i and α_j are the polarizabilities of the respective shell particles.

5.5.5 Free energy interactions

This section describes the λ-dependence of the potentials used for free energy calculations (see sec. *Free energy calculations* (page 345)). 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, λ 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 440) 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.

$$\begin{aligned}
V_b = & \frac{1}{2}\left[(1-\lambda)k_b^A + \lambda k_b^B\right]\left[b - (1-\lambda)b_0^A - \lambda b_0^B\right]^2 \\
\frac{\partial V_b}{\partial \lambda} = & \frac{1}{2}(k_b^B - k_b^A)\left[b - (1-\lambda)b_0^A + \lambda b_0^B\right]^2 + \\
& (b_0^A - b_0^B)\left[b - (1-\lambda)b_0^A - \lambda b_0^B\right]\left[(1-\lambda)k_b^A + \lambda k_b^B\right]
\end{aligned}$$

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:

$$\begin{aligned}
V_d = & \left[(1-\lambda)k_d^A + \lambda k_d^B\right]\left(1 + \cos\left[n_\phi\phi - (1-\lambda)\phi_s^A - \lambda\phi_s^B\right]\right) \\
\frac{\partial V_d}{\partial \lambda} = & (k_d^B - k_d^A)\left(1 + \cos\left[n_\phi\phi - (1-\lambda)\phi_s^A - \lambda\phi_s^B\right]\right) + \\
& (\phi_s^B - \phi_s^A)\left[(1-\lambda)k_d^A - \lambda k_d^B\right]\sin\left[n_\phi\phi - (1-\lambda)\phi_s^A - \lambda\phi_s^B\right]
\end{aligned}$$

Note: that the multiplicity n_ϕ 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:

$$\begin{aligned} V &= \left((1-\lambda)k^A + \lambda k^B\right) f \\ \frac{\partial V}{\partial \lambda} &= \left(k^B - k^A\right) f \end{aligned} \tag{5.229}$$

Coulomb interaction

The Coulomb interaction between two particles of which the charge varies with λ is:

$$\begin{aligned} V_c &= \frac{f}{\varepsilon_{rf} r_{ij}} \left[(1-\lambda) q_i^A q_j^A + \lambda\, q_i^B q_j^B\right] \\ \frac{\partial V_c}{\partial \lambda} &= \frac{f}{\varepsilon_{rf} r_{ij}} \left[-q_i^A q_j^A + q_i^B q_j^B\right] \end{aligned} \tag{5.230}$$

where $f = \frac{1}{4\pi\varepsilon_0} = 138.935\,458$ (see chapter *Definitions and Units* (page 308)).

Coulomb interaction with reaction field

The Coulomb interaction including a reaction field, between two particles of which the charge varies with λ is:

$$\begin{aligned} V_c &= f\left[\frac{1}{r_{ij}} + k_{rf}\, {r_{ij}}^2 - c_{rf}\right] \left[(1-\lambda) q_i^A q_j^A + \lambda\, q_i^B q_j^B\right] \\ \frac{\partial V_c}{\partial \lambda} &= f\left[\frac{1}{r_{ij}} + k_{rf}\, {r_{ij}}^2 - c_{rf}\right] \left[-q_i^A q_j^A + q_i^B q_j^B\right] \end{aligned} \tag{5.231}$$

Note that the constants k_{rf} and c_{rf} are defined using the dielectric constant ε_{rf} of the medium (see sec. *Coulomb interaction with reaction field* (page 359)).

Lennard-Jones interaction

For the Lennard-Jones interaction between two particles of which the *atom type* varies with λ we can write:

$$\begin{aligned} V_{LJ} &= \frac{(1-\lambda) C_{12}^A + \lambda\, C_{12}^B}{{r_{ij}}^{12}} - \frac{(1-\lambda) C_6^A + \lambda\, C_6^B}{{r_{ij}}^6} \\ \frac{\partial V_{LJ}}{\partial \lambda} &= \frac{C_{12}^B - C_{12}^A}{{r_{ij}}^{12}} - \frac{C_6^B - C_6^A}{{r_{ij}}^6} \end{aligned} \tag{5.232}$$

It should be noted that it is also possible to express a pathway from state A to state B using σ and ϵ (see (5.122)). It may seem to make sense physically to vary the force field parameters σ and ϵ 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 531)):

$$\begin{aligned} E_k &= \frac{1}{2}\frac{\mathbf{p}^2}{(1-\lambda)m^A + \lambda m^B} \\ \frac{\partial E_k}{\partial \lambda} &= -\frac{1}{2}\frac{\mathbf{p}^2(m^B - m^A)}{((1-\lambda)m^A + \lambda m^B)^2} \end{aligned} \tag{5.233}$$

after taking the derivative, we *can* insert $\mathbf{p}$ = m $\mathbf{v}$, such that:

$$\frac{\partial E_k}{\partial \lambda} = -\frac{1}{2}\mathbf{v}^2(m^B - m^A) \tag{5.234}$$

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

$$g_k = |\mathbf{r}_k| - d_k \tag{5.235}$$

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 λ dependency by

$$d_k = (1-\lambda)d_k^A + \lambda d_k^B \tag{5.236}$$

Thus the λ-dependent constraint equation is

$$g_k = |\mathbf{r}_k| - \left((1-\lambda)d_k^A + \lambda d_k^B\right). \tag{5.237}$$

The (zero) contribution G to the Hamiltonian from the constraints (using Lagrange multipliers λ_k, which are logically distinct from the free-energy λ) is

$$\begin{aligned} G &= \sum_k^K \lambda_k g_k \\ \frac{\partial G}{\partial \lambda} &= \frac{\partial G}{\partial d_k}\frac{\partial d_k}{\partial \lambda} \\ &= -\sum_k^K \lambda_k \left(d_k^B - d_k^A\right) \end{aligned} \tag{5.238}$$

For SHAKE, the constraint equations are

$$g_k = \mathbf{r}_k^2 - d_k^2 \tag{5.239}$$

with d_k as before, so

$$\frac{\partial G}{\partial \lambda} = -2\sum_k^K \lambda_k \left(d_k^B - d_k^A\right) \tag{5.240}$$

Soft-core interactions

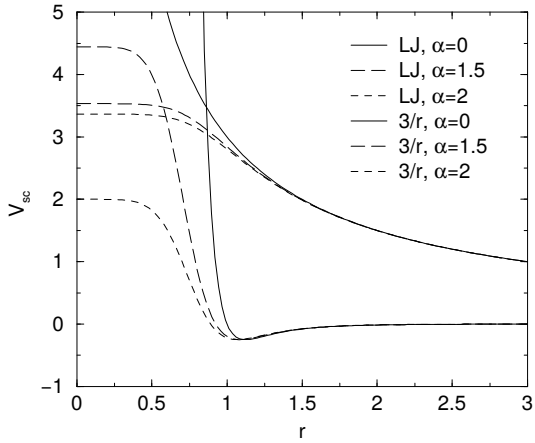


Fig. 5.32: Soft-core interactions at $\lambda = 0.5$, with $p = 2$ and $C_6^A = C_{12}^A = C_6^B = C_{12}^B = 1$.

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 (5.232) and (5.231) may lead to poor convergence. When the particles have nearly disappeared, or are close to appearing (at λ 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 λ).

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 λ 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:

$$\begin{aligned} V_{sc}(r) = &\quad (1-\lambda)V^A(r_A) + \lambda V^B(r_B) \\ r_A = &\quad \left(\alpha\sigma_A^6\lambda^p + r^6\right)^{\frac{1}{6}} \\ r_B = &\quad \left(\alpha\sigma_B^6(1-\lambda)^p + r^6\right)^{\frac{1}{6}} \end{aligned} \tag{5.241}$$

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, α is the soft-core parameter (set with `sc_alpha` in the *mdp* (page 440) file), p is the soft-core λ power (set with `sc_power`), σ 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.

For intermediate λ, 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:

$$F_{sc}(r) = -\frac{\partial V_{sc}(r)}{\partial r} = (1-\lambda)F^A(r_A)\left(\frac{r}{r_A}\right)^5 + \lambda F^B(r_B)\left(\frac{r}{r_B}\right)^5 \tag{5.242}$$

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 531) 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 σ is set with `sc_sigma` in the *mdp* (page 440)

file. These hydrogens produce peaks in $\partial H/\partial\lambda$ at λ 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 using the *mdp* (page 440) option `sc-coul` to `yes`. Note that the `sc-coul` is only taken into account when lambda states are used, not with `couple-lambda0` / `couple-lambda1`, 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`.

Recently, a new formulation of the soft-core approach has been derived that in most cases gives lower and more even statistical variance than the standard soft-core path described above *101* (page 531), *102* (page 531). Specifically, we have:

$$\begin{aligned} V_{sc}(r) = &\quad (1-\lambda)V^A(r_A) + \lambda V^B(r_B) \\ r_A = &\quad \left(\alpha\sigma_A^{48}\lambda^p + r^{48}\right)^{\frac{1}{48}} \\ r_B = &\quad \left(\alpha\sigma_B^{48}(1-\lambda)^p + r^{48}\right)^{\frac{1}{48}} \end{aligned} \tag{5.243}$$

This "1-1-48" path is also implemented in GROMACS. Note that for this path the soft core α should satisfy $0.001 < \alpha < 0.003$, rather than $\alpha \approx 0.5$.

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. 5.33). 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**.

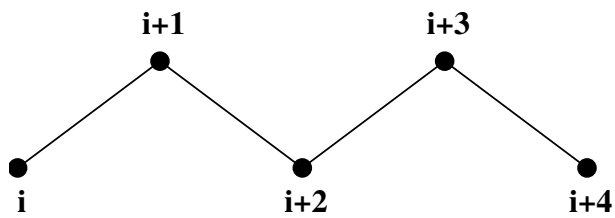


Fig. 5.33: Atoms along an alkane chain.

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 531), 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.

Charge Groups

In principle, the force calculation in MD is an $O(N^2)$ problem. Therefore, we apply a cut-off for non-bonded force (NBF) calculations; only the particles within a certain distance of each other are interacting. This reduces the cost to $O(N)$ (typically $100N$ to $200N$) of the NBF. It also introduces an error, which is, in most cases, acceptable, except when applying the cut-off implies the creation of charges, in which case you should consider using the lattice sum methods provided by GROMACS.

Consider a water molecule interacting with another atom. If we would apply a plain cut-off on an atom-atom basis we might include the atom-oxygen interaction (with a charge of -0.82) without the compensating charge of the protons, and as a result, induce a large dipole moment over the system. Therefore, we have to keep groups of atoms with total charge 0 together. These groups are called *charge groups*. Note that with a proper treatment of long-range electrostatics (e.g. particle-mesh Ewald (sec. *PME* (page 394)), keeping charge groups together is not required.

Treatment of Cut-offs in the group scheme

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.

In the group cut-off scheme, all of the interaction functions in Table 5.9 require that neighbor searching be done with a radius at least as large as the r_c specified for the functional form, because of the use of charge groups. The extra radius is typically of the order of 0.25 nm (roughly the largest distance between two atoms in a charge group plus the distance a charge group can diffuse within neighbor list updates).

Table 5.9: Parameters for the different functional forms of the non-bonded interactions.

Type		Parameters
Coulomb	Plain cut-off	r_c, ε_r
	Reaction field	r_c, ε_{rf}
	Shift function	r_1, r_c, ε_r
	Switch function	r_1, r_c, ε_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 531). We can write the potential energy as:

$$V = V(\mathbf{r}_s, \mathbf{r}_1, \ldots, \mathbf{r}_n) = V^*(\mathbf{r}_1, \ldots, \mathbf{r}_n) \tag{5.244}$$

The force on the particle i is then:

$$\mathbf{F}_i = -\frac{\partial V^*}{\partial \mathbf{r}_i} = -\frac{\partial V}{\partial \mathbf{r}_i} - \frac{\partial V}{\partial \mathbf{r}_s}\frac{\partial \mathbf{r}_s}{\partial \mathbf{r}_i} = \mathbf{F}_i^{direct} + \mathbf{F}_i \tag{5.245}$$

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:

$$\mathbf{F}_i = \begin{bmatrix} \frac{\partial x_s}{\partial x_i} & \frac{\partial y_s}{\partial x_i} & \frac{\partial z_s}{\partial x_i} \\ \frac{\partial x_s}{\partial y_i} & \frac{\partial y_s}{\partial y_i} & \frac{\partial z_s}{\partial y_i} \\ \frac{\partial x_s}{\partial z_i} & \frac{\partial y_s}{\partial z_i} & \frac{\partial z_s}{\partial z_i} \end{bmatrix} \mathbf{F}_s \tag{5.246}$$

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 531).

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 (5.246)) *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.

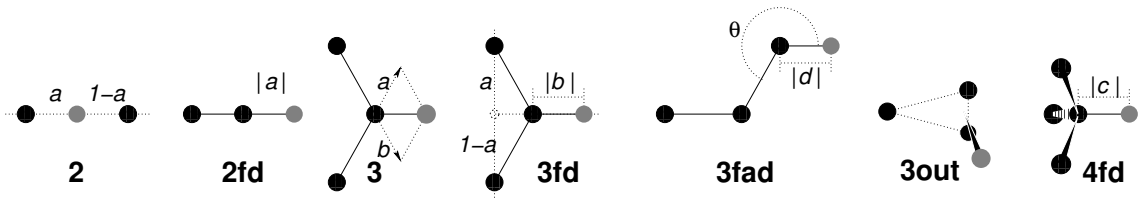


Fig. 5.34: The seven different types of virtual site construction. The constructing atoms are shown as black circles, the virtual sites in gray.

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. 5.34 depicts 6 of the available virtual site constructions. The conceptually simplest construction types are linear combinations:

$$\mathbf{r}_s = \sum_{i=1}^{N} w_i \, \mathbf{r}_i \tag{5.247}$$

The force is then redistributed using the same weights:

$$\mathbf{F}_i = w_i \, \mathbf{F}_s \tag{5.248}$$

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:

$$\mathbf{r}_s \; = \; \mathbf{r}_i \tag{5.249}$$

- The force is moved to the constructing atom:

$$\mathbf{F}_i \; = \; \mathbf{F}_s \tag{5.250}$$

- The velocity of the virtual site equals that of the constructing atom:

$$\mathbf{v}_s \;=\; \mathbf{v}_i \tag{5.251}$$

As a linear combination of two atoms (Fig. 5.34 2)

- The weights are calculated as

$$w_i = 1 - a\,, \;\; w_j = a \tag{5.252}$$

- 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. 5.34 2fd)

- The position is calculated as:

$$\mathbf{r}_s \;=\; \mathbf{r}_i + a\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} \tag{5.253}$$

- 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:

$$\begin{aligned} \mathbf{F}_i &= \mathbf{F}_s - \gamma(\mathbf{F}_{is} - \mathbf{p}) \\ \mathbf{F}_j &= \gamma(\mathbf{F}_s - \mathbf{p}) \end{aligned} \quad \text{where} \quad \begin{aligned} \gamma &= \frac{a}{|\mathbf{r}_{ij}|} \\ \mathbf{p} &= \frac{\mathbf{r}_{is}\cdot\mathbf{F}_s}{\mathbf{r}_{is}\cdot\mathbf{r}_{is}}\mathbf{r}_{is} \end{aligned} \tag{5.254}$$

- The velocity is calculated as:

$$\mathbf{v}_s = \mathbf{v}_i + \frac{a}{|\mathbf{r}_{ij}|}\left(\mathbf{v}_{ij} - \mathbf{r}_{ij}\frac{\mathbf{v}_{ij}\cdot\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|^2}\right) \tag{5.255}$$

As a linear combination of three atoms (Fig. 5.34 3)

- The weights are calculated as:

$$w_i = 1 - a - b\,, \;\; w_j = a\,, \;\; w_k = b \tag{5.256}$$

- 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. 5.34 3fd)

- The position is calculated as:

$$\mathbf{r}_s \;=\; \mathbf{r}_i + b\frac{\mathbf{r}_{ijk}}{|\mathbf{r}_{ijk}|} \;\; \text{where} \;\; \mathbf{r}_{ijk} \;=\; (1-a)\mathbf{r}_{ij} + a\mathbf{r}_{jk} \tag{5.257}$$

- 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:

$$\begin{aligned} \mathbf{F}_i &= \mathbf{F}_s - \gamma(\mathbf{F}_{is} - \mathbf{p}) \\ \mathbf{F}_j &= (1-a)\gamma(\mathbf{F}_s - \mathbf{p}) \\ \mathbf{F}_k &= a\gamma(\mathbf{F}_s - \mathbf{p}) \end{aligned} \quad \text{where} \quad \begin{aligned} \gamma &= \frac{b}{|\mathbf{r}_{ij} + a\mathbf{r}_{jk}|} \\ \mathbf{p} &= \frac{\mathbf{r}_{is}\cdot\mathbf{F}_s}{\mathbf{r}_{is}\cdot\mathbf{r}_{is}}\mathbf{r}_{is} \end{aligned} \tag{5.258}$$

- The velocity is calculated as:

$$\mathbf{v}_s \;=\; \mathbf{v}_i + \frac{b}{|\mathbf{r}_{ijk}|}\left(\dot{\mathbf{r}}_{ijk} - \mathbf{r}_{ijk}\frac{\dot{\mathbf{r}}_{ijk}\cdot\mathbf{r}_{ijk}}{|\mathbf{r}_{ijk}|^2}\right) \tag{5.259}$$

In the plane of three atoms, with a fixed angle and distance (Fig. 5.34 3fad)

- The position is calculated as:

$$\mathbf{r}_s \;=\; \mathbf{r}_i + d\cos\theta\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} + d\sin\theta\frac{\mathbf{r}_\perp}{|\mathbf{r}_\perp|} \;\text{ where }\; \mathbf{r}_\perp \;=\; \mathbf{r}_{jk} - \frac{\mathbf{r}_{ij}\cdot\mathbf{r}_{jk}}{\mathbf{r}_{ij}\cdot\mathbf{r}_{ij}}\mathbf{r}_{ij} \tag{5.260}$$

- 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 α with $\mathbf{r}_{ij}$. Atom k defines the plane and the direction of the angle. **Note** that in this case b and α must be specified, instead of a and b (see also sec. *Virtual sites* (page 404)). 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 (5.260)):

$$\begin{aligned}
\mathbf{F}_i &= \mathbf{F}_s - \frac{d\cos\theta}{|\mathbf{r}_{ij}|}\mathbf{F}_1 + \frac{d\sin\theta}{|\mathbf{r}_\perp|}\left(\frac{\mathbf{r}_{ij}\cdot\mathbf{r}_{jk}}{\mathbf{r}_{ij}\cdot\mathbf{r}_{ij}}\mathbf{F}_2 + \mathbf{F}_3\right)\\
\mathbf{F}_j &= \frac{d\cos\theta}{|\mathbf{r}_{ij}|}\mathbf{F}_1 - \frac{d\sin\theta}{|\mathbf{r}_\perp|}\left(\mathbf{F}_2 + \frac{\mathbf{r}_{ij}\cdot\mathbf{r}_{jk}}{\mathbf{r}_{ij}\cdot\mathbf{r}_{ij}}\mathbf{F}_2 + \mathbf{F}_3\right)\\
\mathbf{F}_k &= \frac{d\sin\theta}{|\mathbf{r}_\perp|}\mathbf{F}_2
\end{aligned} \tag{5.261}$$

where $\mathbf{F}_1 = \mathbf{F}_s - \frac{\mathbf{r}_{ij}\cdot\mathbf{F}_s}{\mathbf{r}_{ij}\cdot\mathbf{r}_{ij}}\mathbf{r}_{ij}$, $\mathbf{F}_2 = \mathbf{F}_1 - \frac{\mathbf{r}_\perp\cdot\mathbf{F}_s}{\mathbf{r}_\perp\cdot\mathbf{r}_\perp}\mathbf{r}_\perp$ 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

$$\begin{aligned}
\frac{\delta}{\delta t}\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} &= \frac{1}{|\mathbf{r}_{ij}|}\left(\mathbf{v}_{ij} - \mathbf{r}_{ij}\frac{\mathbf{v}_{ij}\cdot\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|^2}\right)\\
\frac{\delta}{\delta t}\frac{\mathbf{r}_\perp}{|\mathbf{r}_\perp|} &= \frac{1}{|\mathbf{r}_\perp|}\left(\dot{\mathbf{r}}_\perp - \mathbf{r}_\perp\frac{\dot{\mathbf{r}}_\perp\cdot\mathbf{r}_\perp}{|\mathbf{r}_\perp|^2}\right)\\
\dot{\mathbf{r}}_\perp &= \mathbf{v}_{jk} - \mathbf{r}_{ij}\frac{|\mathbf{r}_{ij}|^2(\mathbf{v}_{ij}\cdot\mathbf{r}_{jk} + \mathbf{r}_{ij}\cdot\mathbf{v}_{jk}) - (\mathbf{r}_{ij}\cdot\mathbf{r}_{jk})(2\mathbf{r}_{ij}\cdot\mathbf{v}_{ij})}{|\mathbf{r}_{ij}|^4} - \frac{\mathbf{r}_{ij}\cdot\mathbf{r}_{jk}}{|\mathbf{r}_{ij}|^2}\,\mathbf{v}_{ij}
\end{aligned} \tag{5.262}$$

As a non-linear combination of three atoms, out of plane (Fig. 5.34 3out)

- The position is calculated as:

$$\mathbf{r}_s \;=\; \mathbf{r}_i + a\mathbf{r}_{ij} + b\mathbf{r}_{ik} + c(\mathbf{r}_{ij}\times\mathbf{r}_{ik}) \tag{5.263}$$

- 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:

$$\begin{aligned}
\mathbf{F}_j &= \begin{bmatrix} a & -c\,z_{ik} & c\,y_{ik}\\ c\,z_{ik} & a & -c\,x_{ik}\\ -c\,y_{ik} & c\,x_{ik} & a \end{bmatrix}\mathbf{F}_s\\
\mathbf{F}_k &= \begin{bmatrix} b & c\,z_{ij} & -c\,y_{ij}\\ -c\,z_{ij} & b & c\,x_{ij}\\ c\,y_{ij} & -c\,x_{ij} & b \end{bmatrix}\mathbf{F}_s\\
\mathbf{F}_i &= \mathbf{F}_s - \mathbf{F}_j - \mathbf{F}_k
\end{aligned} \tag{5.264}$$

- The velocity is calculated as:

$$\mathbf{v}_s \;=\; \mathbf{v}_i + \frac{c}{|\mathbf{r}_m|}\left(\dot{\mathbf{r}}_m - \mathbf{r}_m\frac{\dot{\mathbf{r}}_m\cdot\mathbf{r}_m}{|\mathbf{r}_m|^2}\right) \tag{5.265}$$

From four atoms, with a fixed distance, see separate Fig. 5.35

- 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:

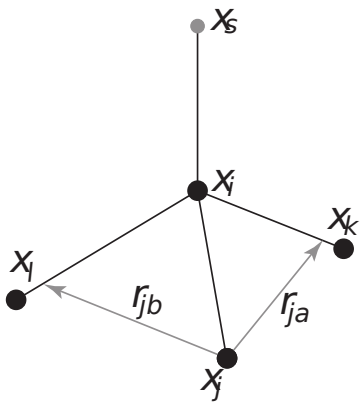


Fig. 5.35: The new 4fdn virtual site construction, which is stable even when all constructing atoms are in the same plane.

- 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:

$$\mathbf{v}_s = \mathbf{v}_i + \frac{c}{|\mathbf{r}_m|}\left(\dot{\mathbf{r}}_m - \mathbf{r}_m\frac{\dot{\mathbf{r}}_m \cdot \mathbf{r}_m}{|\mathbf{r}_m|^2}\right) \quad \text{where} \quad \dot{\mathbf{r}}_m = \dot{\mathbf{r}}_{ja} \times \mathbf{r}_{jb} + \mathbf{r}_{ja} \times \dot{\mathbf{r}}_{jb} \tag{5.266}$$

- 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_α 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:

$$w_i = a_i \left(\sum_{j=1}^{N} a_j \right)^{-1} \tag{5.267}$$

- 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

$$V = \frac{f}{2} \sum_{n_x} \sum_{n_y} \sum_{n_z*} \sum_{i}^{N} \sum_{j}^{N} \frac{q_i q_j}{\mathbf{r}_{ij,\mathbf{n}}}. \tag{5.268}$$

$(n_x, n_y, n_z) = \mathbf{n}$ 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 531). The idea is to convert the single slowly-converging sum (5.268) into two quickly-converging terms and a constant term:

$$\begin{aligned}
V &= V_{\text{dir}} + V_{\text{rec}} + V_0 \\
V_{\text{dir}} &= \frac{f}{2} \sum_{i,j}^{N} \sum_{n_x} \sum_{n_y} \sum_{n_z*} q_i q_j \frac{\text{erfc}(\beta r_{ij,\mathbf{n}})}{r_{ij,\mathbf{n}}} \\
V_{\text{rec}} &= \frac{f}{2\pi V} \sum_{i,j}^{N} q_i q_j \sum_{m_x} \sum_{m_y} \sum_{m_z*} \frac{\exp\left(-(\pi \mathbf{m}/\beta)^2 + 2\pi i \mathbf{m} \cdot (\mathbf{r}_i - \mathbf{r}_j)\right)}{\mathbf{m}^2} \\
V_0 &= -\frac{f\beta}{\sqrt{\pi}} \sum_{i}^{N} q_i^2,
\end{aligned} \tag{5.269}$$

where β 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.

Using Ewald

Don't 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 440) 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 527) 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 527), 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 353)).

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 440) 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 532). 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 532). 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 183) will automatically do a lot of PME optimization, and GROMACS also includes a special tool, *gmx tune_pme* (page 241), which automates the process of selecting the optimal number of PME-only ranks.

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:

$$V(r_{ij}) \;=\; -C_6\, {r_{ij}}^{-6} \tag{5.270}$$

and the corresponding force is:

$$\mathbf{F}_{ij} \;=\; -6\, C_6\, r_{ij}^{-8} \mathbf{r}_{ij} \tag{5.271}$$

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:

$$V_{lr} \;=\; \frac{1}{2} N \rho \int_0^\infty 4\pi r^2 g(r)\, (V(r) - V_c(r))\, \mathrm{d}r \tag{5.272}$$

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 (5.272), 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:

$$\begin{aligned} V_{lr} = \;& \frac{1}{2} N \left(\rho \int_0^{r_1} 4\pi r^2 g(r)\, C_6\, S\, \mathrm{d}r + \rho \int_{r_1}^{r_c} 4\pi r^2\, (V(r) - V_c(r))\, \mathrm{d}r + \rho \int_{r_c}^{\infty} 4\pi r^2 V(r)\, \mathrm{d}r \right) \\ = \;& \frac{1}{2} N \left(\left(\frac{4}{3} \pi \rho r_1^3 - 1 \right) C_6\, S + \rho \int_{r_1}^{r_c} 4\pi r^2\, (V(r) - V_c(r))\, \mathrm{d}r - \frac{4}{3} \pi N \rho\, C_6\, r_c^{-3} \right) \end{aligned} \tag{5.273}$$

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 532):

$$V_{lr} = \;\; -\frac{2}{3} \pi N \rho\, C_6\, r_c^{-3} \tag{5.274}$$

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*:

$$\langle C_6 \rangle = \frac{2}{N(N-1)} \sum_i^N \sum_{j>i}^N C_6(i,j) \tag{5.275}$$

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:

$$\Xi \;=\; -\frac{1}{2} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij} \;=\; 3\, C_6\, r_{ij}^{-6} \tag{5.276}$$

The pressure is given by:

$$P \;=\; \frac{2}{3\,V} \left(E_{kin} - \Xi \right) \tag{5.277}$$

The long-range correction to the virial is given by:

$$\Xi_{lr} = \frac{1}{2} N \rho \int_0^\infty 4\pi r^2 g(r)(\Xi - \Xi_c)\,\mathrm{d}r \tag{5.278}$$

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 532):

$$P_{lr} = -\frac{4}{3}\pi C_6\, \rho^2 r_c^{-3} \tag{5.279}$$

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 $\langle C_6 \rangle$ ((5.275)):

$$P_{lr} = -\frac{4}{3}\pi \langle C_6 \rangle \rho^2 r_c^{-3} \tag{5.280}$$

For inhomogeneous systems, (5.280) can be applied under the same restriction as holds for the energy (see sec. *Energy* (page 396)).

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

$$\begin{aligned}V &= V_{\mathrm{dir}} + V_{\mathrm{rec}} + V_0 \\ V_{\mathrm{dir}} &= -\frac{1}{2} \sum_{i,j}^{N} \sum_{n_x} \sum_{n_y} \sum_{n_z *} \frac{C_6^{ij}\, g(\beta r_{ij,\mathbf{n}})}{{r_{ij,\mathbf{n}}}^6}\end{aligned} \tag{5.281}$$

$$\begin{aligned}V_{\mathrm{rec}} &= \frac{\pi^{\frac{3}{2}} \beta^3}{2V} \sum_{m_x} \sum_{m_y} \sum_{m_z *} f(\pi |\mathbf{m}|/\beta) \times \sum_{i,j}^{N} C_6^{ij} \exp\left[-2\pi i \mathbf{m} \cdot (\mathbf{r_i} - \mathbf{r_j})\right] \\ V_0 &= -\frac{\beta^6}{12} \sum_{i}^{N} C_6^{ii}\end{aligned} \tag{5.282}$$

where $\mathbf{m} = (m_x, m_y, m_z)$, β 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 527), the functions f and g introduced above are defined as

$$\begin{aligned}f(x) &= 1/3\left[(1 - 2x^2)\exp(-x^2) + 2x^3\sqrt{\pi}\,\mathrm{erfc}(x)\right] \\ g(x) &= \exp(-x^2)(1 + x^2 + \frac{x^4}{2}).\end{aligned} \tag{5.283}$$

The above methodology works fine as long as the dispersion parameters can be combined geometrically ((5.123)) in the same way as the charges for electrostatics

$$C_{6,\mathrm{geom}}^{ij} = \left(C_6^{ii}\, C_6^{jj} \right)^{1/2} \tag{5.284}$$

For Lorentz-Berthelot combination rules ((5.124)), the reciprocal part of this sum has to be calculated seven times due to the splitting of the dispersion parameter according to

$$C_{6,\mathrm{L-B}}^{ij} = (\sigma_i + \sigma_j)^6 = \sum_{n=0}^{6} P_n \sigma_i^n \sigma_j^{(6-n)}, \tag{5.285}$$

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 532).

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

$$V_{\mathrm{dir}} = C_6^{\mathrm{dir}} r^{-6} - C_6^{\mathrm{recip}} [1 - g(\beta r)] r^{-6}. \tag{5.286}$$

This potential will reduce to the expression in (5.281) when $C_6^{\mathrm{dir}} = C_6^{\mathrm{recip}}$, and the total interaction is given by

$$\begin{aligned} V(r < r_c) &= \underbrace{C_6^{\mathrm{dir}} r^{-6} - C_6^{\mathrm{recip}} [1 - g(\beta r)] r^{-6}}_{\text{Direct space}} + \underbrace{C_6^{\mathrm{recip}} [1 - g(\beta r)] r^{-6}}_{\text{Reciprocal space}} \\ &= C_6^{\mathrm{dir}} r^{-6} \\ V(r > r_c) &= C_6^{\mathrm{recip}} [1 - g(\beta r)] r^{-6}. \end{aligned} \tag{5.287}$$

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

$$\left(-C_6^{\mathrm{dir}} + C_6^{\mathrm{recip}} [1 - g(\beta r_c)] \right) r_c^{-6} \tag{5.288}$$

is added to (5.287) in order to ensure that the potential is continuous at the cutoff. Note that, in the same way as (5.286), 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 440) 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 183).

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 2022-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.

All-hydrogen force field

The GROMOS-87-based all-hydrogen force field is almost identical to the normal GROMOS-87 force field, since the extra hydrogens have no Lennard-Jones interaction and zero charge. The only differences are in the bond angle and improper dihedral angle terms. This force field is only useful when you need the exact hydrogen positions, for instance for distance restraints derived from NMR measurements. When citing this force field please read the previous paragraph.

GROMOS-96

GROMACS supports the GROMOS-96 force fields *77* (page 530). 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 200).

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 532), 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 361))
- an angle potential based on the cosine of the angle (*Cosine based angle potential* (page 364))

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 437). 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 530) 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 532)
- AMBER96 *112* (page 532)
- AMBER99 *113* (page 532)
- AMBER99SB *114* (page 532)
- AMBER99SB-ILDN *115* (page 532)
- AMBER03 *116* (page 532)
- AMBERGS *117* (page 532)

CHARMM

GROMACS supports the CHARMM force field for proteins *118* (page 532), *119* (page 532), lipids *120* (page 532) and nucleic acids *121* (page 532), *122* (page 532). 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 532) – and the details and results are presented in the paper by Bjelkmar et al. *124* (page 533). 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 200) 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 ϕ and ψ 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 ϕ, and atoms 2-5 define ψ. 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 200), it is possible to use TopoTools *125* (page 533) 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 re-parametrized 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 533) is an example of a solvent-free protein-membrane model for which the membrane was derived from structure-based coarse-graining *127* (page 533). A GROMACS implementation can be found at code.google.com/p/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 356)) 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 445), 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 437) (positions and velocities), or trajectory file *trr* (page 447) (positions, velocities, forces).

This chapter describes the setup of the topology file, the *top* (page 445) 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 432) 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'd 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 529).

Table 5.10: Particle types in GROMACS

Particle	Symbol
atom	A
shell	S
virtual side	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 435) = **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. 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).

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 388).

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 388).

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 166) (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 403)), whereas the charge is listed in *rtp* (page 443) (*rtp* (page 443) = **r**esidue **t**opology **p**arameter file, see *rtp* (page 443)). 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 445) using the *pdb2gmx* (page 200) 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	ϵ	kJ/mol
sigma	σ	nm

Non-bonded parameters

The non-bonded parameters consist of the van der Waals parameters V (`c6` or σ, depending on the combination rule) and W (`c12` or ϵ), 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 408).

```
[ 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.
↪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 418)):

$$\begin{aligned} \text{for combination rule 1 :} \quad & \begin{array}{lllll} \mathrm{V}_{ii} & = & C_i^{(6)} & = & 4\,\epsilon_i\sigma_i^6 \quad [\text{ kJ mol}^{-1}\text{ nm}^6\text{]} \\ \mathrm{W}_{ii} & = & C_i^{(12)} & = & 4\,\epsilon_i\sigma_i^{12} \quad [\text{ kJ mol}^{-1}\text{ nm}^{12}\text{]} \end{array} \\ \text{for combination rules 2 and 3 :} \quad & \begin{array}{lll} \mathrm{V}_{ii} & = & \sigma_i \quad [\text{ nm]} \\ \mathrm{W}_{ii} & = & \epsilon_i \quad [\text{ kJ mol}^{-1}\text{]} \end{array} \end{aligned} \tag{5.289}$$

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:

$$\begin{aligned} \text{for combination rules 1 and 3 :} \quad & \begin{array}{lll} C_{ij}^{(6)} & = & \left(C_i^{(6)}\,C_j^{(6)}\right)^{\frac{1}{2}} \\ C_{ij}^{(12)} & = & \left(C_i^{(12)}\,C_j^{(12)}\right)^{\frac{1}{2}} \end{array} \\ \text{for combination rule 2 :} \quad & \begin{array}{lll} \sigma_{ij} & = & \frac{1}{2}(\sigma_i + \sigma_j) \\ \epsilon_{ij} & = & \sqrt{\epsilon_i\,\epsilon_j} \end{array} \end{aligned} \tag{5.290}$$

When σ and ϵ 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 σ 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 σ, and nothing more.

There is only one set of combination rules for Buckingham potentials:

$$\begin{array}{lll} A_{ij} & = & (A_{ii}\,A_{jj})^{1/2} \\ B_{ij} & = & 2/\left(\frac{1}{B_{ii}} + \frac{1}{B_{jj}}\right) \\ C_{ij} & = & (C_{ii}\,C_{jj})^{1/2} \end{array} \tag{5.291}$$

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 166) 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 440) 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 200) 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 409)).

```
[ 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 418)). When `gen-pairs` is set to `no`, *grompp* (page 166) 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 345)) 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 345) and sec. *Free energy implementation* (page 451) 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 166) 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 418)). 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.

When all non-bonded interactions within or between groups of atoms need to be excluded, is it more convenient and much more efficient to use energy monitor group exclusions (see sec. *The group concept* (page 314)).

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 409)). The distances are constrained using the LINCS or the SHAKE algorithm, which can be selected in the *mdp* (page 440) file. Both types of constraints can be perturbed in free-energy calculations by adding a second constraint distance (see *Constraint forces* (page 430)). Several types of bonds and angles (see Table 5.14) can be converted automatically to constraints by *grompp* (page 166). There are several options for this in the *mdp* (page 440) file.

We have also implemented the SETTLE algorithm *47* (page 529), 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 533). 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 200) generates a topology for the input coordinate file. Several formats are supported for that coordinate file, but *pdb* (page 442) is the most commonly-used format (hence the name *pdb2gmx* (page 200)). *pdb2gmx* (page 200) 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 200) 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 200), which indicates that a force field in a `xxx.ff` directory is desired. *pdb2gmx* (page 200) 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 200): an atom type file (extension *atp* (page 435), see *Atom types* (page 403)) 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 200) 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 443) 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 443) 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 200) 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 443). 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 200), but if changes are needed make them in the *top* (page 445) file (see *Topology file* (page 418)), or in a *rtp* (page 443) file in the working directory as explained in sec. *pdb2gmx input files* (page 410). Defining topologies of new small molecules is probably easier by writing an include topology file *itp* (page 439) directly. This will be discussed in section *Molecule.itp file* (page 426). When adding a new protein residue to the database, don't forget to add the residue name to the residuetypes.dat file, so that *grompp* (page 166), *make_ndx* (page 182) and analysis tools can recognize the residue as a protein residue (see *Default Groups* (page 500)).

The *rtp* (page 443) files are only used by *pdb2gmx* (page 200). As mentioned before, the only extra information this program needs from the *rtp* (page 443) 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 445) 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

 [ 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
```

(continues on next page)

(continued from previous page)

```
 [ 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 200) for adding missing hydrogens (see *hdb* (page 438)), the atom names defined in the *rtp* (page 443) 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 439) 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 443) files. These strings are copied to the topology file and can be replaced by force-field parameters by the C-preprocessor in *grompp* (page 166) using `#define` statements.

pdb2gmx (page 200) automatically generates all angles. This means that for most force fields the `[ angles ]` field is only useful for overriding *itp* (page 439) parameters. For the GROMOS-96 force field the interaction number of all angles needs to be specified.

pdb2gmx (page 200) 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 200) can add correction maps to the dihedrals using the default `-cmap` option. Please refer to *CHARMM* (page 401) for more information.

pdb2gmx (page 200) 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 444) files are used to convert standard residue names to the force-field build block names. If no *r2b* (page 444) 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 444) 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 200) code) through the *r2b* (page 444) file and `specbond.dat` files. This convention is only of importance when you are adding residue types to an *rtp* (page 443) 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 417)), which can subsequently be translated by the *r2b* (page 444) 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_δ protonated
HISE	neutral histidine with N_ϵ protonated
HISH	positive histidine with both N_δ and N_ϵ 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 435) 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 438) files. It contains information for the *pdb2gmx* (page 200) 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 443) file (see *rtp* (page 443)) must be the same. The format of the hydrogen database is as follows:

```
; res    # additions
         # H add type    H       i       j       k
ALA      1
```

(continues on next page)

(continued from previous page)

```
        1       1       H       N       -C      CA
ARG     4
        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 533).
  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°.
  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 530) 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 530) 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 533).
  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 533).
- 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 200) 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 443) 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 200). 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 438)), 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 443)), 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 443) file, see *rtp* (page 443).

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 447) file for each force field. Just as for the termini, there is one such file for each class of residues in the *rtp* (page 443) file.

The virtual site database is not really a very 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 200) 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 200) 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. The format of the lines is as follows:

```
resA atomA nbondsA resB atomB nbondsB length newresA newresB
```

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` $\pm$ 10% in the coordinate file supplied to *pdb2gmx* (page 200), 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.

5.6.6 File formats

Topology file

The topology file is built following the GROMACS specification for a molecular topology. A *top* (page 445) file can be generated by *pdb2gmx* (page 200). 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 166) for generating exclusions, and which bonded interactions can be converted to constraints by *grompp* (page 166).

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; m (u); q (e); particle type; V$^{(cr)}$; W$^{(cr)}$
	`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`)		
	`constraint`$^{(*)}$`types`	(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 406)
- $^{(*)}$ 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 422, 1]	func. type[Page 422, 2]	Order of parameters and their units	use in F.E.?[Page 422, 3]
bond	`bonds`[4,5]	2	1	b_0 (nm); k_b (kJ mol^{-1}nm^{-2}	all
G96 bond	`bonds`?,[Page 422, 5]	2	2	b_0 (nm); k_b (kJ mol^{-1}nm^{-4}	all
Morse	`bonds`?,[Page 422, 5]	2	3	b_0 (nm); D (kJ mol^{-1}; β (nm^{-1}	all
cubic bond	`bonds`?,[Page 422, 5]	2	4	b_0 (nm); $C_{i=2,3}$ (kJ mol^{-1}nm^{-i}	
connection	`bonds`?	2	5		
harmonic potential	`bonds`	2	6	b_0 (nm); k_b (kJ mol^{-1}nm^{-2}	all
FENE bond	`bonds`?	2	7	b_m (nm); k_b (kJ mol^{-1}nm^{-2}	
tabulated bond	`bonds`?	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

continues on next page

Table 5.14 – continued from previous page

Name of interaction	Topology file directive	num. atoms[Page 422, 1]	func. type[Page 422, 2]	Order of parameters and their units	use in F.E.?[Page 422, 3]
restraint potential	`bonds`	2	10	low, up$_{1,2}$ (nm); k_{dr} ((kJ mol^{-1}nm^{-2})	all
extra LJ or Coulomb	`pairs`	2	1	V[7]; W[Page 422, 7]	all
extra LJ or Coulomb	`pairs`	2	2	fudge QQ (); q_i; q_j (e), V[Page 422, 7]; W[Page 422, 7]	
extra LJ or Coulomb	`pairs_nb`	2	1	q_i; q_j (e); V[Page 422, 7]; W[Page 422, 7]	
angle	`angles`[?]	3	1	θ_0 (deg); k_θ (kJ mol^{-1}rad^{-2})	all
G96 angle	`angles`[?]	3	2	θ_0 (deg); k_θ (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`[?]	3	5	θ_0 (deg); k_θ (kJ mol^{-1}rad^{-2}); r_{13} (nm); k_{UB} ((kJ mol^{-1}nm^{-2})	all
quartic angle	`angles`[?]	3	6	θ_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
restricted bending potential	`angles`	3	10	θ_0 (deg); k_θ (kJ mol^{-1})	
proper dihedral	`dihedrals`	4	1	ϕ_s (deg); k_ϕ (kJ mol^{-1}); multiplicity	ϕ, k
improper dihedral	`dihedrals`	4	2	ξ_0 (deg); k_ξ (kJ mol^{-1}rad^{-2})	all

continues on next page

Table 5.14 – continued from previous page

Name of interaction	Topology file directive	num. atoms[Page 422, 1]	func. type[Page 422, 2]	Order of parameters and their units	use in F.E.?[Page 422, 3]
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	ϕ_s (deg); k_ϕ (kJ mol^{-1}); multiplicity	ϕ, 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)	`dihedrals`	4	9	ϕ_s (deg); k_ϕ (kJ mol^{-1}); multiplicity	ϕ, k
restricted dihedral	`dihedrals`	4	10	ϕ_0 (deg); k_ϕ (kJ mol^{-1})	
combined bending-torsion potential	`dihedrals`	4	11	a_0, a_1, a_2, a_3, a_4 (kJ mol^{-1})	
exclusions	`exclusions`	1		one or more atom indices	
constraint	`constraints`[?]	2	1	b_0 (nm)	all
constraint[?]	`constraints`	2	2	b_0 (nm)	all
SETTLE	`settles`	1	1	d_{OH}, d_{HH} (nm)	
1-body virtual site	`virtual_sites1`	2	0		
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	θ (deg); d (nm)	
3-body virtual site (out)	`virtual_sites3`	4	4	a, b (); c (nm^{-1})	
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	

continues on next page

Table 5.14 – continued from previous page

Name of interaction	Topology file directive	num. atoms[Page 422, 1]	func. type[Page 422, 2]	Order of parameters and their units	use in F.E.?[Page 422, 3]
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	ϕ_0 (deg); $\Delta\phi$ (deg); k_{dihr} (kJ mol^{-1}rad^{-2})	all
orientation restraint	`orientation_restraints`	2	1	exp.; label; α; c (U nm$^{\alpha}$; obs. (U); weight (U^{-1})	
angle restraint	`angle_restraints`	4	1	θ_0 (deg); k_c (kJ mol^{-1}); multiplicity	θ, k
angle restraint (z)	`angle_restraints_z`	2	1	θ_0 (deg); k_c (kJ mol^{-1}); multiplicity	θ, 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)

[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 166) for generating exclusions
[5] This interaction type can be converted to constraints by *grompp* (page 166)
[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 406)

  - 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
- Starting with GROMACS version 3.1.3, 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; starting with GROMACS version 3.1.3 *grompp* (page 166) generates a warning for parameter redefinitions with different values
- 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

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
```

(continues on next page)

(continued from previous page)

```
   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 ]
     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 166) 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)
- `comb-rule` is the number of the combination rule (see *Non-bonded parameters* (page 406)).
- `gen-pairs` is for pair generation. The default is 'no', *i.e.* get 1-4 parameters from the pair-types 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 183) 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 445) 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
```

(continues on next page)

(continued from previous page)

```
    1     2     3     4     3
    2     3     4     5     3
```

In the original implementation of the potential for alkanes *131* (page 533) 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 372)). The reference positions are read from a separate coordinate file by *grompp* (page 166).

[dihedral_restraints] : restrain selected dihedrals to a reference value. The implementation of dihedral restraints is described in section *Dihedral restraints* (page 374) 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 ϕ_0 in (5.195) and (5.196) of the manual.
- `dphi` is the value of $\Delta\phi$ in (5.196) of the manual.
- `fc` is the force constant k_{dihr} in (5.196) of the manual.

#include "tip3p.itp" : includes a topology file that was already constructed (see section *Molecule.itp file* (page 426)).

[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 445). 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 166). The blocks of molecules do not need to be contiguous, but some tools (e.g. *genion* (page 164)) 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 314) and sec. *Using Groups* (page 499)).

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 200) 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 ]
```

(continues on next page)

(continued from previous page)

```
    1       2
...
    6       8
[ dihedrals ]
;   ai    aj    ak    al funct  definition
     2     1     3     4   9
...
     6     1     3     5   9
[ dihedrals ]
     3     6     1     2   4
     1     4     3     5   4
     1     7     6     8   4
```

Using *itp* (page 439) files results in a very short *top* (page 445) 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 445) 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 445) 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 533) or by Van Buuren and Berendsen *133* (page 533). In fact, you can use much of the functionality of the C preprocessor, `cpp`, because *grompp* (page 166) 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 440) file (containing *grompp* (page 166) input parameters), or use the line `#define DeLoof` early in your *top* (page 445) or *itp* (page 439) file; and
- put the `#ifdef` statements in your *top* (page 445), as shown below:

```
...



[ atoms ]
```

(continues on next page)

(continued from previous page)

```
; nr       type      resnr     residu      atom       cgnr       charge
↪      mass
#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 200) to implement optional position restraints (*Position restraints* (page 372)) by `#include`-ing an *itp* (page 439) 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 345). 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 λ-dependence of the interactions is described in section sec. *Free energy interactions* (page 383). 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 166) 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 λ 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 ␣
↪14.027
  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
```

(continues on next page)

(continued from previous page)

```
[ 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

#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 453). 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 409)). 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 437) 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 200) 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)
- 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 418).

For each force field, there several files which are only used by *pdb2gmx* (page 200). These are: residue databases (*rtp* (page 443)) the hydrogen database (*hdb* (page 438)), two termini databases (`.n.tdb` and `.c.tdb`, see) and the atom type database (*atp* (page 435)), which contains only the masses. Other optional files are described in sec. *pdb2gmx input files* (page 410).

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 445) file under the `[ moleculetype ]` section (see *Topology file* (page 418) 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 440)** run parameters, input for *gmx grompp* (page 166) and *gmx convert-tpr* (page 130)

m2p **(page 439)** input for *gmx xpm2ps* (page 254)

Structure files

gro **(page 437)** GROMACS format

g96 **(page 437)** GROMOS-96 format

pdb **(page 442)** brookhaven Protein DataBank format

Structure+mass(db): ***tpr*** **(page 446),** ***gro*** **(page 437),** ***g96*** **(page 437), or** ***pdb*** **(page 442)** 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 445)** system topology (ascii)

itp **(page 439)** include topology (ascii)

rtp **(page 443)** residue topology (ascii)

ndx **(page 441)** index file (ascii)

n2t **(page 442)** atom naming definition (ascii)

atp **(page 435)** atom type library (ascii)

r2b **(page 444)** residue to building block mapping (ascii)

arn **(page 435)** atom renaming database (ascii)

hdb **(page 438)** hydrogen atom database (ascii)

vsd **(page 447)** virtual site database (ascii)

tdb **(page 444)** termini database (ascii)

Run Input files

tpr **(page 446)** system topology, parameters, coordinates and velocities (binary, portable)

Trajectory files

tng **(page 444)** Any kind of data (compressed, portable, any precision)

trr **(page 447)** x, v and f (binary, full precision, portable)

xtc **(page 448)** x only (compressed, portable, any precision)

gro **(page 437)** x and v (ascii, any precision)

g96 **(page 437)** x only (ascii, fixed high precision)

pdb **(page 442)** x only (ascii, reduced precision)

Formats for full-precision data: *tng* (page 444) or *trr* (page 447)

Generic trajectory formats: *tng* (page 444), *xtc* (page 448), *trr* (page 447), *gro* (page 437), *g96* (page 437), or *pdb* (page 442)

Energy files

***ene* (page 437)** energies, temperature, pressure, box size, density and virials (binary)

***edr* (page 436)** energies, temperature, pressure, box size, density and virials (binary, portable)

Generic energy formats: *edr* (page 436) or *ene* (page 437)

Other files

***dat* (page 436)** generic, preferred for input

***edi* (page 436)** essential dynamics constraints input for *gmx mdrun* (page 183)

***eps* (page 437)** Encapsulated Postscript

***log* (page 439)** log file

***map* (page 440)** colormap input for *gmx do_dssp* (page 145)

***mtx* (page 441)** binary matrix data

***out* (page 442)** generic, preferred for output

***tex* (page 444)** LaTeX input

***xpm* (page 447)** ascii matrix data, use *gmx xpm2ps* (page 254) to convert to *eps* (page 437)

***xvg* (page 450)** 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 183).

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.

dlg

The dlg file format is used as input for the *gmx view* (page 248) trajectory viewer. These files are not meant to be altered by the end user.

Sample

```
grid 39 18 {

group "Bond Options" 1 1 16 9 {
  radiobuttons { " Thin Bonds"  " Fat Bonds" " Very Fat Bonds" "␣
↪Spheres" }
        "bonds" "Ok" " F" "help bonds"
}

group "Other Options" 18 1 20 13 {
  checkbox " Show Hydrogens"      ""  "" "FALSE" "help opts"
  checkbox " Draw plus for atoms" ""  "" "TRUE"  "help opts"
  checkbox " Show Box"            ""  "" "TRUE"  "help opts"
  checkbox " Remove PBC"          ""  "" "FALSE" "help opts"
  checkbox " Depth Cueing"        ""  "" "TRUE"  "help opts"
  edittext "Skip frames: "        ""  "" "0"     "help opts"
}

simple 1 15 37 2 {
  defbutton "Ok" "Ok" "Ok" "Ok" "help bonds"
}

}
```

edi

Files with the edi file extension contain information for *gmx mdrun* (page 183) 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 155).

ene

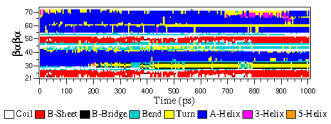
The ene file extension stands for binary energy file. It holds the energies as generated during your *gmx mdrun* (page 183).

The file can be transformed to a portable energy file (portable across hardware platforms), the *edr* (page 436) file using the program *gmx eneconv* (page 152).

See also *gmx energy* (page 155).

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 254) 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
```

(continues on next page)

(continued from previous page)

```
    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 200) 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 445) 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 254) program. All of these options are very easy to comprehend when you look at the PosScript(tm) output from *gmx xpm2ps* (page 254).

```
; 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 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
```

(continues on next page)

(continued from previous page)

```
y-minorticklen          = 4.0
y-label                 =
y-fontsize              = 16
y-font                  = Times-Roman
y-tickfontsize          = 10
y-tickfont              = Helvetica
```

map

This file maps matrix data to RGB values which is used by the *gmx do_dssp* (page 145) program.

The format of this file is as follow: first line number of elements in the colormap. Then for each line: The first character is a code for the secondary structure type. Then comes a string for use in the legend of the plot and then the R (red) G (green) and B (blue) values.

In this case the colors are (in order of appearance): white, red, black, cyan, yellow, blue, magenta, orange.

```
8
~   Coil            1.0       1.0     1.0
E   B-Sheet         1.0       0.0     0.0
B   B-Bridge        0.0       0.0     0.0
S   Bend            0.0       0.8     0.8
T   Turn            1.0       1.0     0.0
H   A-Helix         0.0       0.0     1.0
G   3-Helix         1.0       0.0     1.0
I   5-Helix         1.0       0.6     0.0
```

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 166) 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

cutoff-scheme           = Verlet

coulombtype             = PME
rcoulomb                = 1.0
```

(continues on next page)

(continued from previous page)

```
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 166) 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.

mtx

Files with the mtx file extension contain a matrix. The file format is identical to the *trr* (page 447) format. Currently this file format is only used for hessian matrices, which are produced with *gmx mdrun* (page 183) and read by *gmx nmeig* (page 191).

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, by the graphics program (*gmx view* (page 248)) and by the preprocessor (*gmx grompp* (page 166)). 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 182).

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 253), 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 442) 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 200) to make a GROMACS topology for a protein contained in a *pdb* (page 442) 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 439) 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 445) file, this is used for the GROMOS96 forcefield.

gmx pdb2gmx (page 200) automatically generates all angles, this means that the `[angles]` field is only useful for overriding *itp* (page 439) parameters.

gmx pdb2gmx (page 200) 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 200) 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

 [ 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
```

(continues on next page)

(continued from previous page)

```
;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 440) file options control which of these are written by *gmx mdrun* (page 183), 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 148).

gmx dump (page 148) `-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 120) `-f traj.tng`

top

The top file extension stands for topology. It is an ascii file which is read by *gmx grompp* (page 166) which processes it and creates a binary topology (*tpr* (page 446) 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
    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
```

(continues on next page)

(continued from previous page)

```
    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
#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 148) -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 120) -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 148):

```
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 120) -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 254). 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 g_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",
```

(continues on next page)

(continued from previous page)

```
"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",
"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

To read and write these files the following "C" routines are available:

```
/* All functions return 1 if successful, 0 otherwise */

extern int open_xtc(XDR *xd,char *filename,char *mode);
/* Open a file for xdr I/O */

extern void close_xtc(XDR *xd);
/* Close the file for xdr I/O */

extern int read_first_xtc(XDR *xd,char *filename,
                          int *natoms,int *step,real *time,
                          matrix box,rvec **x,real *prec);
/* Open xtc file, read xtc file first time, allocate memory for x␣
↪*/

extern int read_next_xtc(XDR *xd,
                         int *natoms,int *step,real *time,
                         matrix box,rvec *x,real *prec);
/* Read subsequent frames */

extern int write_xtc(XDR *xd,
                     int natoms,int step,real time,
                     matrix box,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 `-lgmx.$(CPU)`.

Using xtc in your FORTRAN programs

To read and write these in a FORTRAN program use the calls to `readxtc` and `writextc` as in the following sample program which reads and xtc file and copies it to a new one:

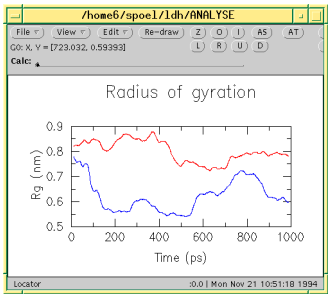
```
program testxtc

parameter (maxatom=10000,maxx=3*maxatom)
integer xd,xd2,natoms,step,ret,i
real    time,box(9),x(maxx)

call xdrfopen(xd,"test.xtc","r",ret)
print *,'opened test.xtc, ret=',ret
call xdrfopen(xd2,"testout.xtc","w",ret)
print *,'opened testout.xtc, ret=',ret

call readxtc(xd,natoms,step,time,box,x,prec,ret)

if ( ret .eq. 1 ) then
   call writextc(xd2,natoms,step,time,box,x,prec,ret)
else
   print *,'Error reading xtc'
endif

stop
end
```

To link your program use `-L$(GMXHOME)/lib/$(CPU) -lxtcf` on your linker command line.

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 440) file, through the use of the `couple-moltype`, `couple-lambda0`, `couple-lambda1`, and `couple-intramol` *mdp* (page 440) 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, λ 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 λ 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 λ variable, can be preserved by using only `fep-lambdas` to define the pathway.

Fig. 5.36 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 λ 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 λ 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:

```
restraint-lambdas       = 0.0 0.1 0.2 0.4 0.6 1.0
```

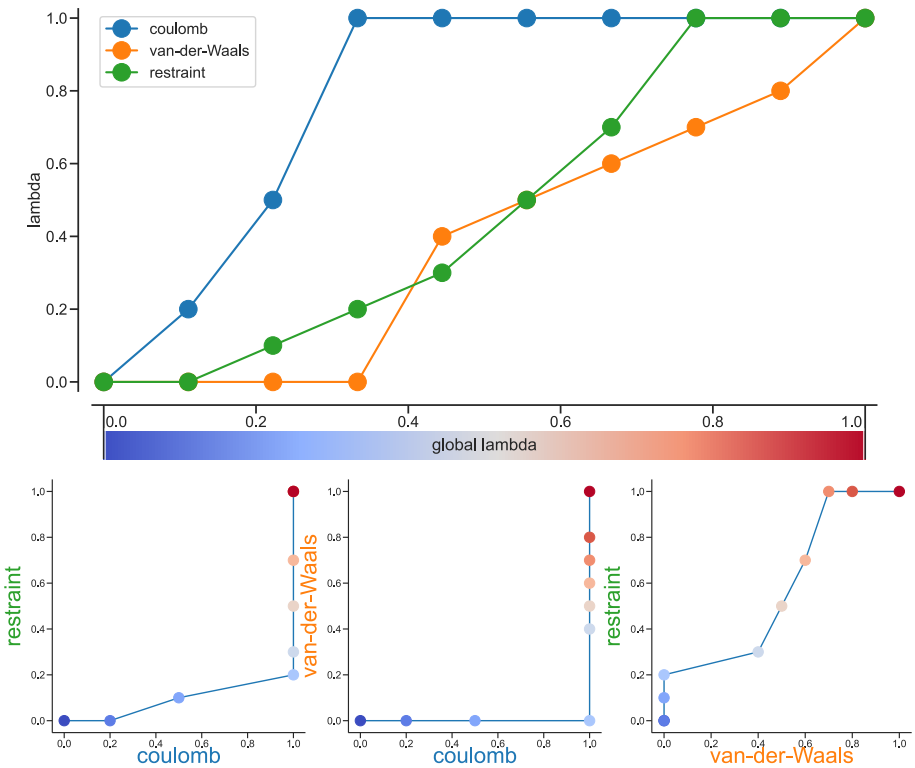


Fig. 5.36: Separate λ values for Coulomb, van-der-Waals and restraint interactions.

and all of the other components of the λ vector would be left in the A state.

To compute free energies with a vector λ using thermodynamic integration, then the TI equation becomes vector equation:

$$\Delta F = \int \langle \nabla H \rangle \cdot d\vec{\lambda} \tag{5.292}$$

or for finite differences:

$$\Delta F \approx \int \sum \langle \nabla H \rangle \cdot \Delta\lambda \tag{5.293}$$

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 533). For a system of two non-interacting masses the potential of mean force is:

$$V_{pmf}(r) = -(n_c - 1)k_B T \log(r) \tag{5.294}$$

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 533) to obtain the equilibrium free-energy difference ΔG between two distances from many non-equilibrium simulations:

$$\Delta G_{AB} = -k_B T \log \left\langle e^{-\beta W_{AB}} \right\rangle_A \tag{5.295}$$

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 457). 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.

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.

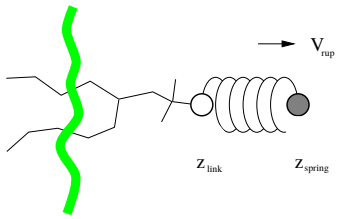


Fig. 5.37: Schematic picture of pulling a lipid out of a lipid bilayer with umbrella pulling. V_{rup} is the velocity at which the spring is retracted, Z_{link} is the atom to which the spring is attached and Z_{spring} is the location of the spring.

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 458)) 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 440) 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 440) 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):

$$\begin{aligned} w(r < r_{\text{cyl}}) &= \quad 1 - 2\left(\frac{r}{r_{\text{cyl}}}\right)^2 + \left(\frac{r}{r_{\text{cyl}}}\right)^4 \\ w(r \geq r_{\text{cyl}}) &= \quad 0 \end{aligned} \tag{5.296}$$

Note that the radial dependence on the weight causes a radial force on both cylinder group and the other pull 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.

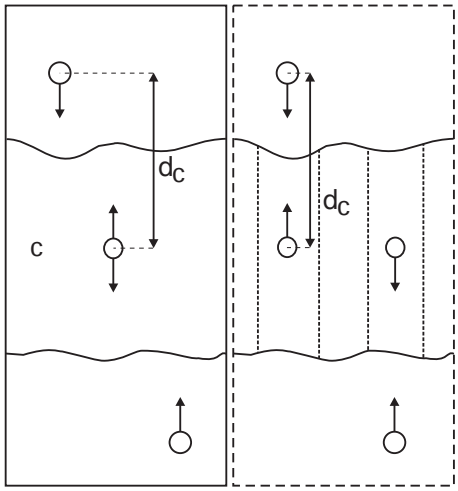


Fig. 5.38: Comparison of a plain center of mass reference group versus a cylinder reference group applied to interface systems. C is the reference group. The circles represent the center of mass of two groups plus the reference group, d_c is the reference distance.

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 533) 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:

$$w_i' = w_i \sum_{j=1}^{N} w_j \, m_j \Big/ \sum_{j=1}^{N} w_j^2 \, m_j \tag{5.297}$$

where m_j is the mass of atom j of the group. The mass of the group, required for calculating the constraint force, is:

$$M = \sum_{i=1}^{N} w_i' \, m_i \tag{5.298}$$

The definition of the weighted center of mass is:

$$\mathbf{r}_{com} = \sum_{i=1}^{N} w_i' \, m_i \, \mathbf{r}_i \Big/ M \tag{5.299}$$

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:

$$\mathbf{F}_i = \frac{w_i' \, m_i}{M} \, \mathbf{F}_{com} \tag{5.300}$$

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. 5.39. Since the potential now depends on the coordinates of two additional groups defining the orientation, the torque forces will work on these two groups.

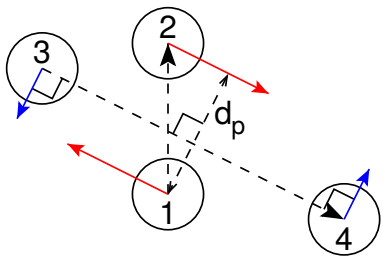


Fig. 5.39: The pull setup for geometry `direction-relative`. The "normal" pull groups are 1 and 2. Groups 3 and 4 define the pull direction and thus the direction of the normal pull forces (red). This leads to reaction forces (blue) on groups 3 and 4, which are perpendicular to the pull direction. Their 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 440) 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 440) 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 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 440) 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␣
↪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
```

(continues on next page)

(continued from previous page)

```
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 440) 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 481). 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 535) *137* (page 533) 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 533), local elevation *139* (page 533) and metadynamics *140* (page 533) 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

Rather than biasing the reaction coordinate $\xi(x)$ directly, AWH acts on a *reference coordinate* λ. The fundamentals of the method is based on the connection between atom coordinates and λ and is established through the extended ensemble *68* (page 530),

$$P(x,\lambda) = \frac{1}{\mathcal{Z}} e^{g(\lambda) - Q(\xi(x),\lambda) - V(x)}, \tag{5.301}$$

where $g(\lambda)$ is a bias function (a free variable) and $V(x)$ is the unbiased potential energy of the system. The distribution along λ 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

$$P(\lambda) = \int P(x,\lambda)dx = \frac{1}{\mathcal{Z}} e^{g(\lambda)} \int e^{-Q(\xi(x),\lambda) - V(x)} dx \equiv \frac{1}{\mathcal{Z}} e^{g(\lambda) - F(\lambda)}, \tag{5.302}$$

where $F(\lambda)$ is the free energy

$$F(\lambda) = -\ln \int e^{-Q(\xi(x),\lambda)-V(x)} dx. \tag{5.303}$$

The reaction coordinate $\xi(x)$ is commonly coupled to λ with a harmonic potential

$$Q(\xi, \lambda) = \frac{1}{2}\beta k(\xi - \lambda)^2, \tag{5.304}$$

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, λ 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. (5.302) 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 $\rho(\lambda)$.

It is also possible to directly control the λ state of, e.g., alchemical free energy perturbations *187* (page 535). In that case there is no harmonic potential and λ changes in discrete steps along the reaction coordinate depending on the biased free energy difference between the λ states. N.b., it is not yet possible to use AWH in combination with perturbed masses or constraints.

For a multidimensional reaction coordinate ξ, the sampling interval is the Cartesian product $I = \Pi_\mu I_\mu$ (a rectangular domain).

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 $\rho_n(\lambda)$,

$$g_n(\lambda) = \ln \rho_n(\lambda) + F_n(\lambda), \tag{5.305}$$

which is consistent with (5.302). Note that also the target distribution may be updated during the simulation (see examples in section *Choice of target distribution* (page 464)). Substituting this choice of $g = g_n$ back into (5.302) yields the simple free energy update

$$\Delta F_n(\lambda) = F(\lambda) - F_n(\lambda) = -\ln \frac{P_n(\lambda)}{\rho_n(\lambda)}, \tag{5.306}$$

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

$$\omega_n(\lambda|x) \equiv P_n(\lambda|x) = \frac{e^{g_n(\lambda)-Q(\xi(x),\lambda)}}{\sum_{\lambda'} e^{g_n(\lambda')-Q(\xi(x),\lambda')}}. \tag{5.307}$$

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 (5.306) 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,

$$\Delta F_n = -\ln \frac{W_n(\lambda) + \sum_t \omega_n(\lambda|x(t))}{W_n(\lambda) + \sum_t \rho_n(\lambda))}. \tag{5.308}$$

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 460)), is

$$W_{n+1}(\lambda) = W_n(\lambda) + \sum_t \rho_n(\lambda). \tag{5.309}$$

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 λ values, $\omega_n(\lambda) > \rho_n(\lambda)$ and $\Delta F_n(\lambda) < 0$, which by (5.305) implies $\Delta g_n(\lambda) < 0$ (assuming $\Delta\rho_n \equiv 0$). Thus, the probability to sample λ decreases after the update (see (5.302)). 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 (5.304)), 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 533),

$$|\Delta F_n(\lambda)| \propto \frac{1}{N_n} e^{-\frac{1}{2}\beta k(\xi(x)-\lambda)^2}. \tag{5.310}$$

When directly controlling the lambda state of the system, the shape of the update is instead

$$|\Delta F_n(\lambda)| \propto \frac{1}{N_n} P_n(\lambda|x). \tag{5.311}$$

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 533) (see also section *Reweighting and combining biased data* (page 465)) and will converge at the same rate as $F(\lambda)$, see Fig. 5.40. The PMF will be written to output (see section *Usage* (page 466)).

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 (5.307). This is also called Gibbs sampling or independence sampling. Alternatively, and by default in the code, the following *convolved bias potential* can be applied,

$$U_n(\xi) = -\ln \int e^{g_n(\lambda)-Q(\xi,\lambda)} d\lambda. \tag{5.312}$$

These two approaches are equivalent in the sense that they give rise to the same biased probabilities $P_n(x)$ (cf. (5.301)) 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 λ 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 (5.309), 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 ((5.309)). The AWH initial stage takes inspiration from the well-known Wang-Landau algorithm *138* (page 533), 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 (5.313) below. After a covering has occurred, N_n is scaled up by a constant "growth factor" γ, chosen heuristically as $\gamma = 3$. 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 $1/N$ ((5.310)) this leads to a close to exponential decay of the update size in the initial stage, see Fig. 5.40.

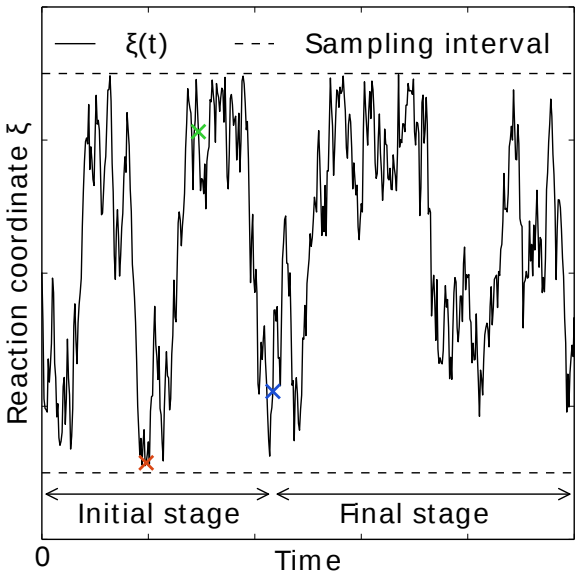


Fig. 5.40: AWH evolution in time for a Brownian particle in a double-well potential. The reaction coordinate $\xi(t)$ traverses the sampling interval multiple times in the initial stage before exiting and entering the final stage. In the final stage, the dynamics of ξ becomes increasingly diffusive.

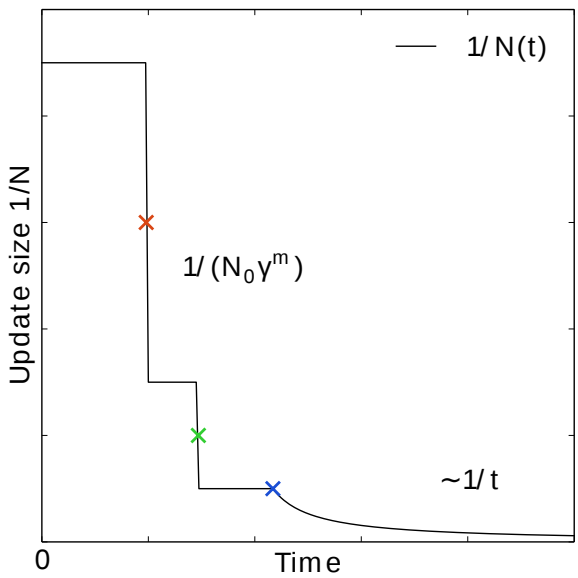


Fig. 5.41: In the final stage, the dynamics of ξ becomes increasingly diffusive. The times of covering are shown as $\times$-markers of different colors. At these times the free energy update size $\sim 1/N$, where N is the size of the weight histogram, is decreased by scaling N by a factor of $\gamma = 3$.

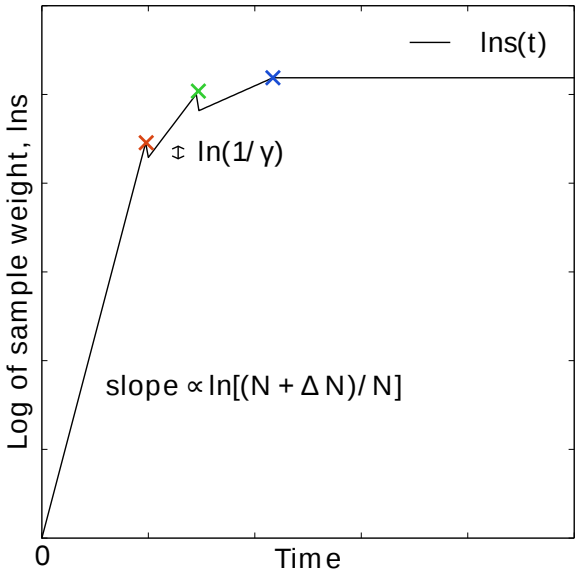


Fig. 5.42: In the final stage, N grows at the sampling rate and thus $1/N \sim 1/t$. The exit from the final stage is determined on the fly by ensuring that the effective sample weight s of data collected in the final stage exceeds that of initial stage data (note that $\ln s(t)$ is plotted).

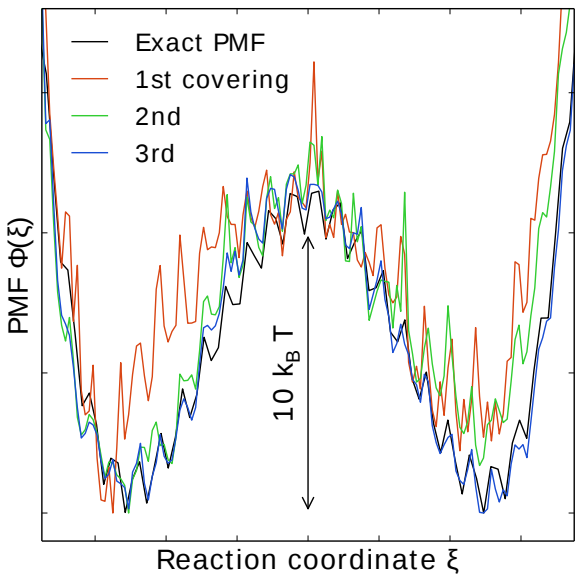


Fig. 5.43: An estimate of the PMF is also extracted from the simulation (bottom right), which after exiting the initial stage should estimate global free energy differences fairly accurately.

The update size directly determines the rate of change of $F_n(\lambda)$ and hence, from (5.305), 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. 5.40. 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 λ_μ in the one-dimensional sampling interval I_μ 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

$$\Delta W(\lambda^*) \geq w_{\text{peak}} \equiv \prod_\mu \frac{\Delta\lambda_\mu}{\sqrt{2\pi}\sigma_k}. \tag{5.313}$$

Here, $\Delta\lambda_\mu$ is the point spacing of the discretized I_μ and $\sigma_k = 1/\sqrt{\beta k_\mu}$ (where k_μ 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 533). 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 533). 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 "artifically" 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 γ 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. 5.40. 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 γ 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 ΔN 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 Δn 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 γ, 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 (5.309)). 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

$$\rho_{\mathrm{const}}(\lambda) = \mathrm{const}. \tag{5.314}$$

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

$$\rho_{\mathrm{cut}}(\lambda) \propto \frac{1}{1 + e^{F(\lambda) - F_{\mathrm{cut}}}}, \tag{5.315}$$

where F_{cut} is a free energy cutoff (relative to $\min_\lambda F(\lambda)$). Thus, regions of the sampling interval where $F(\lambda) > F_{\mathrm{cut}}$ 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 $0 < s_\beta < 1$,

$$\rho_{\mathrm{Boltz}}(\lambda) \propto e^{-s_\beta F(\lambda)}, \tag{5.316}$$

The parameter s_β determines to what degree the free energy landscape is flattened; the lower s_β, the flatter. Note that both $\rho_{\mathrm{cut}}(\lambda)$ and $\rho_{\mathrm{Boltz}}(\lambda)$ 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 (5.305)).

There is in fact an alternative approach to obtaining $\rho_{\mathrm{Boltz}}(\lambda)$ as the limiting target distribution in AWH, which is particular in the way the weight histogram $W(\lambda)$ and the target distribution ρ 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 533), see *137* (page 533) for details. Because of the popularity and success of well-tempered metadynamics, this is a special case worth considering. In this case ρ is a function of the reference weight histogram

$$\rho_{\mathrm{Boltz,loc}}(\lambda) \propto W(\lambda), \tag{5.317}$$

and the update of the weight histogram is modified (cf. (5.309))

$$W_{n+1}(\lambda) = W_n(\lambda) + s_\beta \sum_t \omega(\lambda | x(t)). \tag{5.318}$$

Thus, here the weight histogram equals the real history of samples, but scaled by s_β. 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 $\rho = \rho_{\mathrm{Boltz,loc}}$ there is no initial stage (section *The initial stage* (page 460)). The rescaling of the weight histogram applied in the initial stage is a global operation, which is incompatible $\rho_{\mathrm{Boltz,loc}}$ only depending locally on the sampling history.

Lastly, the target distribution can be modulated by arbitrary probability weights

$$\rho(\lambda) = \rho_0(\lambda) w_{\mathrm{user}}(\lambda). \tag{5.319}$$

where $w_{\mathrm{user}}(\lambda)$ is provided by user data and in principle $\rho_0(\lambda)$ can be any of the target distributions mentioned above.

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) indepedendent 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 460)) 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.

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 (5.313). 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 λ 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 λ_μ along dimension μ as visited, and shares this with the other walkers, also all points within a certain diameter D_{cover} should have been visited (i.e. fulfill (5.313)). Increasing D_{cover} increases robustness, but may slow down convergence. For the maximum value of D_{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.

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 ξ-biased data by reweighting ("unbiasing") the trajectory using the bias potential $U_{n(t)}$, see (5.312). 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 $e^{-U_{n(t)}(\xi(t))}$,

$$\hat{\Phi}(u) = -\ln \sum_t 1_u(u(t)) e^{U_{n(t)}(\xi(t)} \mathcal{Z}_{n(t)}. \tag{5.320}$$

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 ξ (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 (5.320) 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 (5.320) (using slightly different notation) can be found in *143* (page 533).

The friction metric

During the AWH simulation, the following time-integrated force correlation function is calculated,

$$\eta_{\mu\nu}(\lambda) = \beta \int_0^\infty \frac{\langle \delta\mathcal{F}_\mu(x(t),\lambda)\delta\mathcal{F}_\nu(x(0),\lambda)\omega(\lambda|x(t))\omega(\lambda|x(0))\rangle}{\langle \omega^2(\lambda|x)\rangle} dt. \tag{5.321}$$

Here $\mathcal{F}_\mu(x,\lambda) = k_\mu(\xi_\mu(x) - \lambda_\mu)$ is the force along dimension μ from an harmonic potential centered at λ 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 (5.307), reweight the samples. $\eta_{\mu\nu}(\lambda)$ is a friction tensor *186* (page 535) and *144* (page 533). Its matrix elements are inversely proportional to local diffusion coefficients. A measure of sampling (in)efficiency at each λ is given by

$$\eta^{\frac{1}{2}}(\lambda) = \sqrt{\det \eta_{\mu\nu}(\lambda)}. \tag{5.322}$$

A large value of $\eta^{\frac{1}{2}}(\lambda)$ indicates slow dynamics and long correlation times, which may require more sampling.

Usage

AWH stores data in the energy file (*edr* (page 436)) with a frequency set by the user. The data – the PMF, the convolved bias, distributions of the λ and ξ coordinates, etc. – can be extracted after the simulation using the *gmx awh* (page 116) tool. Furthermore, the trajectory of the reaction coordinate $\xi(t)$ is printed to the pull output file `pullx.xvg`. The log file (*log* (page 439)) 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 533), 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

$$\frac{1}{N_0} = \frac{1}{N_0(\varepsilon_0, D)} = \frac{1}{\Delta t_{\text{sample}}} \max_d \frac{L_d^2}{2D_d}\varepsilon_0^2 \tag{5.323}$$

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 (ε_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 ε_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 ε_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 obtaining from an AWH simulation with the *gmx awh* (page 116) tool. ε_0 on the other hand, should be a rough estimate of the initial error.

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 ξ and the reference coordinate λ, 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 neccessarily 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.

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 (5.315)) 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 465)), 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 466). $\eta_{\mu\nu}(\lambda)$ as well as the sampling efficiency measure $\eta^{\frac{1}{2}}(\lambda)$ ((5.322)) are written to the energy file and can be extracted with *gmx awh* (page 116). 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 533).

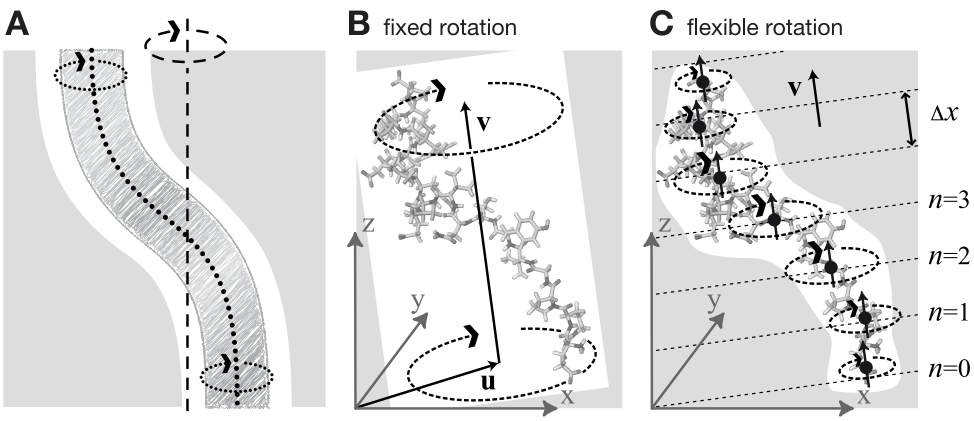


Fig. 5.44: Comparison of fixed and flexible axis rotation. A: Rotating the sketched shape inside the white tubular cavity can create artifacts when a fixed rotation axis (dashed) is used. More realistically, the shape would revolve like a flexible pipe-cleaner (dotted) inside the bearing (gray). B: Fixed 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 Δx.

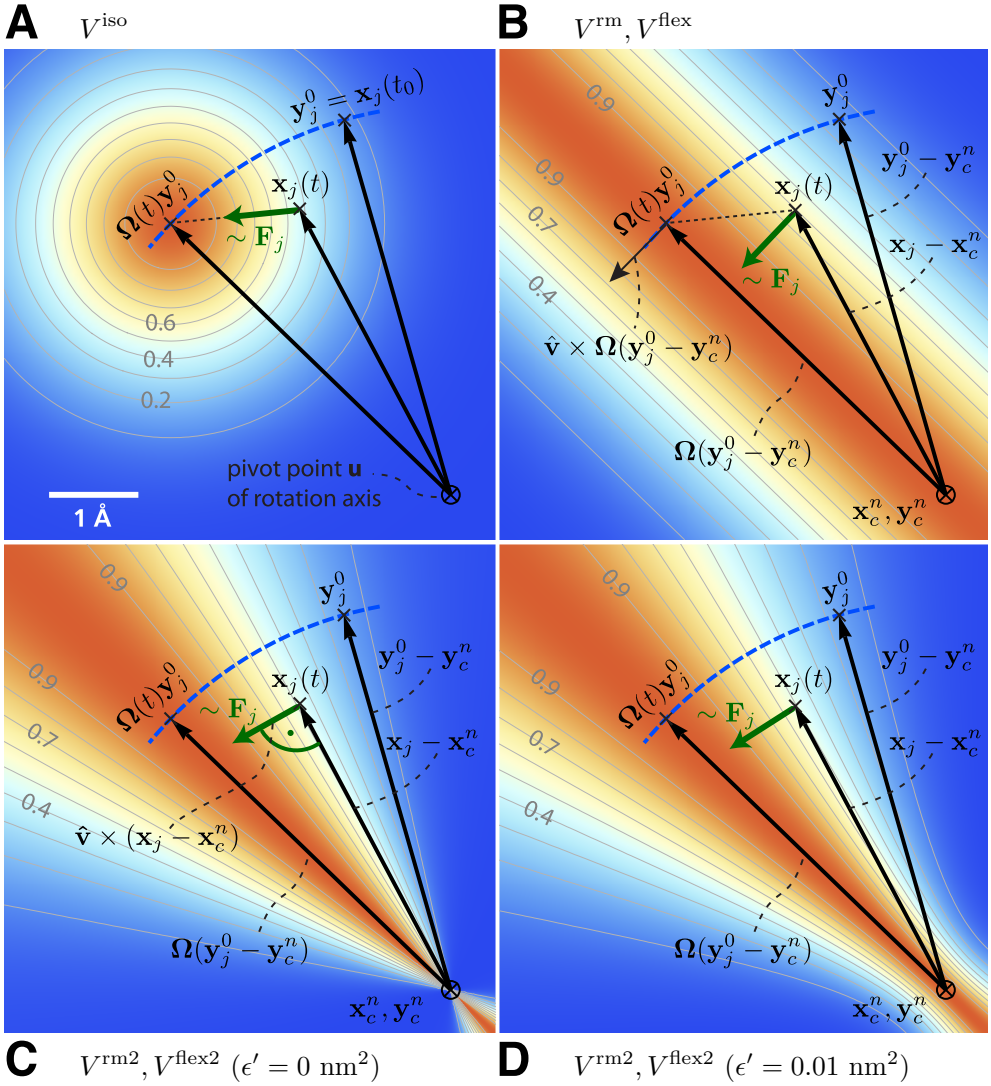


Fig. 5.45: Selection of different rotation potentials and definition of notation. All four potentials V (color coded) are shown for a single atom at position $\mathbf{x}_j(t)$. A: Isotropic potential V^{iso}, B: radial motion potential V^{rm} and flexible potential V^{flex}, C–D: radial motion2 potential V^{rm2} and flexible2 potential V^{flex2} for ϵ'=0nm^2 (C) and ϵ'=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.

Fixed Axis Rotation

Stationary Axis with an Isotropic Potential

In the fixed axis approach (see Fig. 5.44 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 ω 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,

$$V^{\mathrm{iso}} = \frac{k}{2} \sum_{i=1}^{N} w_i \left[\mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u}) - (\mathbf{x}_i - \mathbf{u})\right]^2 \tag{5.324}$$

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

$$\mathbf{\Omega}(t) = \begin{pmatrix} \cos\omega t + v_x^2\,\xi & v_x v_y\,\xi - v_z \sin\omega t & v_x v_z\,\xi + v_y \sin\omega t \\ v_x v_y\,\xi + v_z \sin\omega t & \cos\omega t + v_y^2\,\xi & v_y v_z\,\xi - v_x \sin\omega t \\ v_x v_z\,\xi - v_y \sin\omega t & v_y v_z\,\xi + v_x \sin\omega t & \cos\omega t + v_z^2\,\xi \end{pmatrix} \tag{5.325}$$

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. 5.45 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

$$\mathbf{F}_j^{\mathrm{iso}} = -\nabla_j V^{\mathrm{iso}} = k\, w_j \left[\mathbf{\Omega}(t)(\mathbf{y}_j^0 - \mathbf{u}) - (\mathbf{x}_j - \mathbf{u})\right] \tag{5.326}$$

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,

$$\mathbf{x}_c = \frac{1}{M} \sum_{i=1}^{N} m_i \mathbf{x}_i \mathrm{and} \mathbf{y}_c^0 = \frac{1}{M} \sum_{i=1}^{N} m_i \mathbf{y}_i^0\,, \tag{5.327}$$

which yields the "pivot-free" isotropic potential

$$V^{\mathrm{iso-pf}} = \frac{k}{2} \sum_{i=1}^{N} w_i \left[\mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^0) - (\mathbf{x}_i - \mathbf{x}_c)\right]^2, \tag{5.328}$$

with forces

$$\mathbf{F}_j^{\mathrm{iso-pf}} = k\, w_j \left[\mathbf{\Omega}(t)(\mathbf{y}_j^0 - \mathbf{y}_c^0) - (\mathbf{x}_j - \mathbf{x}_c)\right]. \tag{5.329}$$

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. (5.324) and (5.328)) 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^{iso}) or within the rotation group, in case of $V^{\mathrm{iso-pf}}$.

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

$$\mathbf{r}_i = \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u}) - (\mathbf{x}_i - \mathbf{u}) \tag{5.330}$$

onto the plane perpendicular to the rotation vector,

$$\mathbf{r}_i^{\perp} := \mathbf{r}_i - (\mathbf{r}_i \cdot \hat{\mathbf{v}})\hat{\mathbf{v}} \tag{5.331}$$

yielding

$$\begin{aligned} V^{\mathrm{pm}} = & \frac{k}{2}\sum_{i=1}^{N} w_i (\mathbf{r}_i^{\perp})^2 \\ = & \frac{k}{2}\sum_{i=1}^{N} w_i \left\{\mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u}) - (\mathbf{x}_i - \mathbf{u}) \right. \\ & \left. - \left\{\left[\mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u}) - (\mathbf{x}_i - \mathbf{u})\right] \cdot \hat{\mathbf{v}}\right\} \hat{\mathbf{v}}\right\}^2 \end{aligned}$$

and similarly

$$\mathbf{F}_j^{\mathrm{pm}} = k\, w_j\, \mathbf{r}_j^{\perp} \tag{5.332}$$

Pivot-Free Parallel Motion Potential

Replacing in eqn. (5.332) the fixed pivot $\mathbf{u}$ by the center of mass $\mathbf{x_c}$ yields the pivot-free variant of the parallel motion potential. With

$$\mathbf{s}_i = \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^0) - (\mathbf{x}_i - \mathbf{x}_c) \tag{5.333}$$

the respective potential and forces are

$$V^{\mathrm{pm-pf}} = \frac{k}{2}\sum_{i=1}^{N} w_i (\mathbf{s}_i^{\perp})^2 \tag{5.334}$$

$$\mathbf{F}_j^{\mathrm{pm-pf}} = k\, w_j\, \mathbf{s}_j^{\perp} \tag{5.335}$$

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^{rm} and V^{rm2}, drastically reduce or even eliminate this effect. The first variant, V^{rm} (Fig. 5.45 B), eliminates all force components parallel to the vector connecting the reference atom and the rotation axis,

$$V^{\mathrm{rm}} = \frac{k}{2}\sum_{i=1}^{N} w_i \left[\mathbf{p}_i \cdot (\mathbf{x}_i - \mathbf{u})\right]^2, \tag{5.336}$$

with

$$\mathbf{p}_i := \frac{\hat{\mathbf{v}} \times \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u})}{\|\hat{\mathbf{v}} \times \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u})\|}\,. \tag{5.337}$$

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

$$\mathbf{F}_j^{\mathrm{rm}} = -k\, w_j \left[\mathbf{p}_j \cdot (\mathbf{x}_j - \mathbf{u})\right]\, \mathbf{p}_j\,. \tag{5.338}$$

Pivot-Free Radial Motion Potential

Proceeding similar to the pivot-free isotropic potential yields a pivot-free version of the above potential. With

$$\mathbf{q}_i := \frac{\hat{\mathbf{v}} \times \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^0)}{\|\hat{\mathbf{v}} \times \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^0)\|}\,, \tag{5.339}$$

the potential and force for the pivot-free variant of the radial motion potential read

$$V^{\mathrm{rm-pf}} = \frac{k}{2}\sum_{i=1}^{N} w_i \left[\mathbf{q}_i \cdot (\mathbf{x}_i - \mathbf{x}_c)\right]^2\,, \tag{5.340}$$

$$\mathbf{F}_j^{\mathrm{rm-pf}} = -k\, w_j \left[\mathbf{q}_j \cdot (\mathbf{x}_j - \mathbf{x}_c)\right] \mathbf{q}_j + k\frac{m_j}{M}\sum_{i=1}^{N} w_i \left[\mathbf{q}_i \cdot (\mathbf{x}_i - \mathbf{x}_c)\right] \mathbf{q}_i\,. \tag{5.341}$$

Radial Motion 2 Alternative Potential

As seen in Fig. 5.45 B, the force resulting from V^{rm} still contains a small, second-order radial component. In most cases, this perturbation is tolerable; if not, the following alternative, V^{rm2}, fully eliminates the radial contribution to the force, as depicted in Fig. 5.45 C,

$$V^{\mathrm{rm2}} = \frac{k}{2}\sum_{i=1}^{N} w_i \frac{\left[(\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{u})) \cdot \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u})\right]^2}{\|\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{u})\|^2 + \epsilon'}\,, \tag{5.342}$$

where a small parameter ϵ' has been introduced to avoid singularities. For ϵ'=$0\mathrm{nm}^2$, 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 ϵ' (*e.g.*, ϵ'=$0.01\mathrm{nm}^2$, Fig. 5.45 D) in the denominator of eqn. (5.342) yields a well-defined potential and continuous forces also close to the rotation axis, which is not the case for ϵ'=$0\mathrm{nm}^2$ (Fig. 5.45 C). With

$$\begin{aligned}
\mathbf{r}_i &:= \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{u}) \\
\mathbf{s}_i &:= \frac{\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{u})}{\|\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{u})\|} \equiv \Psi_i\ \hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{u}) \\
\Psi_i^* &:= \frac{1}{\|\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{u})\|^2 + \epsilon'}
\end{aligned} \tag{5.343}$$

the force on atom j reads

$$\mathbf{F}_j^{\mathrm{rm2}} = -k\left\{w_j\,(\mathbf{s}_j \cdot \mathbf{r}_j)\left[\frac{\Psi_j^*}{\Psi_j}\mathbf{r}_j - \frac{\Psi_j^{*2}}{\Psi_j^3}(\mathbf{s}_j \cdot \mathbf{r}_j)\mathbf{s}_j\right]\right\} \times \hat{\mathbf{v}}. \tag{5.344}$$

Pivot-Free Radial Motion 2 Potential

The pivot-free variant of the above potential is

$$V^{\mathrm{rm2-pf}} = \frac{k}{2}\sum_{i=1}^{N} w_i \frac{\left[(\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c)) \cdot \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c)\right]^2}{\|\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c)\|^2 + \epsilon'}\,. \tag{5.345}$$

With

$$\begin{aligned}
\mathbf{r}_i &:= \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c) \\
\mathbf{s}_i &:= \frac{\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c)}{\|\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c)\|} \equiv \Psi_i\ \hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c) \\
\Psi_i^* &:= \frac{1}{\|\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c)\|^2 + \epsilon'}
\end{aligned} \tag{5.346}$$

the force on atom j reads

$$\mathbf{F}_j^{\mathrm{rm2-pf}} = -k \left\{ w_j \left(\mathbf{s}_j \cdot \mathbf{r}_j\right) \left[\frac{\Psi_j^*}{\Psi_j} \mathbf{r}_j - \frac{\Psi_j^{*2}}{\Psi_j^3} (\mathbf{s}_j \cdot \mathbf{r}_j) \mathbf{s}_j \right] \right\} \times \hat{\mathbf{v}}$$
$$+k \, \frac{m_j}{M} \left\{ \sum_{i=1}^{N} w_i \left(\mathbf{s}_i \cdot \mathbf{r}_i\right) \left[\frac{\Psi_i^*}{\Psi_i} \mathbf{r}_i - \frac{\Psi_i^{*2}}{\Psi_i^3} (\mathbf{s}_i \cdot \mathbf{r}_i) \, \mathbf{s}_i \right] \right\} \times \hat{\mathbf{v}} \, .$$

Flexible Axis Rotation

As sketched in Fig. 5.44 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. (5.340) and (5.345) will now be generalized towards a "flexible axis" as sketched in Fig. 5.44 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. 5.44 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^{flex} by a Gaussian function

$$g_n(\mathbf{x}_i) = \Gamma \exp\left(-\frac{\beta_n^2(\mathbf{x}_i)}{2\sigma^2} \right) , \tag{5.347}$$

centered at the midplane of the nth slab. Here σ is the width of the Gaussian function, Δx the distance between adjacent slabs, and

$$\beta_n(\mathbf{x}_i) := \mathbf{x}_i \cdot \hat{\mathbf{v}} - n \, \Delta x \, . \tag{5.348}$$

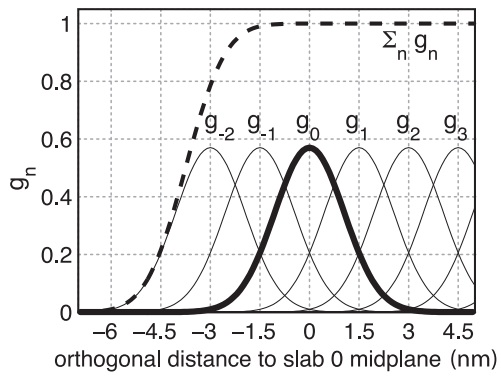


Fig. 5.46: Gaussian functions g_n centered at $n \, \Delta x$ for a slab distance $\Delta x = 1.5$ nm and $n \geq -2$. Gaussian function g_0 is highlighted in bold; the dashed line depicts the sum of the shown Gaussian functions.

A most convenient choice is $\sigma = 0.7 \Delta x$ and

$$1/\Gamma = \sum_{n \in Z} \exp\left(-\frac{(n - \frac{1}{4})^2}{2 \cdot 0.7^2} \right) \approx 1.75464 \, , \tag{5.349}$$

which yields a nearly constant sum, essentially independent of $\mathbf{x}_i$ (dashed line in Fig. 5.46), *i.e.*,

$$\sum_{n \in Z} g_n(\mathbf{x}_i) = 1 + \epsilon(\mathbf{x}_i) \, , \tag{5.350}$$

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 $\mathbf{x}_i$,

$$\mathbf{x}_c^n = \frac{\sum_{i=1}^N g_n(\mathbf{x}_i)\,m_i\,\mathbf{x}_i}{\sum_{i=1}^N g_n(\mathbf{x}_i)\,m_i}\,, \tag{5.351}$$

while the reference centers $\mathbf{y}_c^n$ are calculated from the reference positions $\mathbf{y}_i^0$,

$$\mathbf{y}_c^n = \frac{\sum_{i=1}^N g_n(\mathbf{y}_i^0)\,m_i\,\mathbf{y}_i^0}{\sum_{i=1}^N g_n(\mathbf{y}_i^0)\,m_i}\,. \tag{5.352}$$

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. (5.340) / Fig. 5.45 B), yielding as the contribution of slab n

$$V^n = \frac{k}{2}\sum_{i=1}^N w_i\,g_n(\mathbf{x}_i)\,[\mathbf{q}_i^n\cdot(\mathbf{x}_i-\mathbf{x}_c^n)]^2\,, \tag{5.353}$$

and a total potential function

$$V^{\text{flex}} = \sum_n V^n\,. \tag{5.354}$$

Note that the global center of mass $\mathbf{x}_c$ used in eqn. (5.340) is now replaced by $\mathbf{x}_c^n$, the center of mass of the slab. With

$$\begin{aligned} \mathbf{q}_i^n &:= \frac{\hat{\mathbf{v}}\times\mathbf{\Omega}(t)(\mathbf{y}_i^0-\mathbf{y}_c^n)}{\|\hat{\mathbf{v}}\times\mathbf{\Omega}(t)(\mathbf{y}_i^0-\mathbf{y}_c^n)\|} \\ b_i^n &:= \mathbf{q}_i^n\cdot(\mathbf{x}_i-\mathbf{x}_c^n)\,, \end{aligned} \tag{5.355}$$

the resulting force on atom j reads

$$\begin{aligned} \mathbf{F}_j^{\text{flex}} = & -k\,w_j\sum_n g_n(\mathbf{x}_j)\,b_j^n\left\{\mathbf{q}_j^n - b_j^n\frac{\beta_n(\mathbf{x}_j)}{2\sigma^2}\hat{\mathbf{v}}\right\} \\ & + k\,m_j\sum_n\frac{g_n(\mathbf{x}_j)}{\sum_h g_n(\mathbf{x}_h)}\sum_{i=1}^N w_i\,g_n(\mathbf{x}_i)\,b_i^n\left\{\mathbf{q}_i^n - \frac{\beta_n(\mathbf{x}_j)}{\sigma^2}\,[\mathbf{q}_i^n\cdot(\mathbf{x}_j-\mathbf{x}_c^n)]\,\hat{\mathbf{v}}\right\}. \end{aligned}$$

Note that for V^{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,

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{x}_c\,, \quad \text{and} \quad \tilde{\mathbf{y}}_i^0 = \mathbf{y}_i^0 - \mathbf{y}_c^0\,, \tag{5.356}$$

such that

$$V^{\text{flex-t}} = \frac{k}{2}\sum_n\sum_{i=1}^N w_i\,g_n(\tilde{\mathbf{x}}_i)\left[\frac{\hat{\mathbf{v}}\times\mathbf{\Omega}(t)(\tilde{\mathbf{y}}_i^0-\tilde{\mathbf{y}}_c^n)}{\|\hat{\mathbf{v}}\times\mathbf{\Omega}(t)(\tilde{\mathbf{y}}_i^0-\tilde{\mathbf{y}}_c^n)\|}\cdot(\tilde{\mathbf{x}}_i-\tilde{\mathbf{x}}_c^n)\right]^2. \tag{5.357}$$

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 $\mathbf{F}^{\text{flex-t}}$ have the same form as eqn. (5.356).

Flexible Axis 2 Alternative Potential

In this second variant, slab segmentation is applied to V^{rm2} (eqn. (5.345)), resulting in a flexible axis potential without radial force contributions (Fig. 5.45 C),

$$V^{\mathrm{flex2}} = \frac{k}{2} \sum_{i=1}^{N} \sum_{n} w_i \, g_n(\mathbf{x}_i) \frac{\left[(\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c^n)) \cdot \mathbf{\Omega}(t) (\mathbf{y}_i^0 - \mathbf{y}_c^n) \right]^2}{\|\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c^n)\|^2 + \epsilon'} \tag{5.358}$$

With

$$\begin{aligned}
\mathbf{r}_i^n &:= \mathbf{\Omega}(t)(\mathbf{y}_i^0 - \mathbf{y}_c^n) \\
\mathbf{s}_i^n &:= \frac{\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c^n)}{\|\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c^n)\|} \equiv \psi_i \;\; \hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c^n) \\
\psi_i^* &:= \frac{1}{\|\hat{\mathbf{v}} \times (\mathbf{x}_i - \mathbf{x}_c^n)\|^2 + \epsilon'} \\
W_j^n &:= \frac{g_n(\mathbf{x}_j) \, m_j}{\sum_h g_n(\mathbf{x}_h) \, m_h} \\
\mathbf{S}^n &:= \sum_{i=1}^{N} w_i \; g_n(\mathbf{x}_i) \; (\mathbf{s}_i^n \cdot \mathbf{r}_i^n) \left[\frac{\psi_i^*}{\psi_i} \mathbf{r}_i^n - \frac{\psi_i^{*2}}{\psi_i^3} (\mathbf{s}_i^n \cdot \mathbf{r}_i^n) \; \mathbf{s}_i^n \right]
\end{aligned} \tag{5.359}$$

the force on atom j reads

$$\begin{aligned}
\mathbf{F}_j^{\mathrm{flex2}} = \; & -k \left\{ \sum_n w_j \; g_n(\mathbf{x}_j) \; (\mathbf{s}_j^n \cdot \mathbf{r}_j^n) \; \left[\frac{\psi_j^*}{\psi_j} \mathbf{r}_j^n - \frac{\psi_j^{*2}}{\psi_j^3} (\mathbf{s}_j^n \cdot \mathbf{r}_j^n) \; \mathbf{s}_j^n \right] \right\} \times \hat{\mathbf{v}} \\
& + k \left\{ \sum_n W_j^n \, \mathbf{S}^n \right\} \times \hat{\mathbf{v}} - k \left\{ \sum_n W_j^n \; \frac{\beta_n(\mathbf{x}_j)}{\sigma^2} \frac{1}{\psi_j} \;\; \mathbf{s}_j^n \cdot \mathbf{S}^n \right\} \hat{\mathbf{v}} \\
& + \frac{k}{2} \left\{ \sum_n w_j \; g_n(\mathbf{x}_j) \frac{\beta_n(\mathbf{x}_j)}{\sigma^2} \frac{\psi_j^*}{\psi_j^2} (\mathbf{s}_j^n \cdot \mathbf{r}_j^n)^2 \right\} \hat{\mathbf{v}}.
\end{aligned}$$

Applying transformation (5.356) 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^{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 440) input file. The reference positions $\mathbf{y}_i^0$ are read from a special *trr* (page 447) file provided to *grompp* (page 166). If no such file is found, $\mathbf{x}_i(t = 0)$ are used as reference positions and written to *trr* (page 447) such that they can be used for subsequent setups. All parameters of the potentials such as k, ϵ', etc. (Table 5.16) are provided as *mdp* (page 440) 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			k	$\hat{\mathbf{v}}$	$\mathbf{u}$	ω	ϵ'	Δx	$g_n^{\min}$
mdp (page 440) input variable name			k	vec	pivot	rate	eps	slab-dist	min-gauss
unit			$\frac{\text{kJ}}{\text{mol}\cdot\text{nm}^2}$	–	nm	°/ps	nm^2	nm	–
fixed axis potentials:		eqn.							
isotropic	$\mathrm{V}^{\mathrm{iso}}$	(5.324)	**x**	**x**	**x**	**x**	–	–	–
— pivot-free	$\mathrm{V}^{\mathrm{iso-pf}}$	(5.328)	**x**	**x**	–	**x**	–	–	–
parallel motion	V^{pm}	(5.332)	**x**	**x**	**x**	**x**	–	–	–
— pivot-free	$\mathrm{V}^{\mathrm{pm-pf}}$	(5.334)	**x**	**x**	–	**x**	–	–	–
radial motion	V^{rm}	(5.336)	**x**	**x**	**x**	**x**	–	–	–
— pivot-free	$\mathrm{V}^{\mathrm{rm-pf}}$	(5.340)	**x**	**x**	–	**x**	–	–	–
radial motion 2	$\mathrm{V}^{\mathrm{rm2}}$	(5.342)	**x**	**x**	**x**	**x**	**x**	–	–
— pivot-free	$\mathrm{V}^{\mathrm{rm2-pf}}$	(5.345)	**x**	**x**	–	**x**	**x**	–	–
flexible axis potentials:		eqn.							
flexible	$\mathrm{V}^{\mathrm{flex}}$	(5.354)	**x**	**x**	–	**x**	–	**x**	**x**
— transl. tol	$\mathrm{V}^{\mathrm{flex-t}}$	(5.357)	**x**	**x**	–	**x**	–	**x**	**x**
flexible 2	$\mathrm{V}^{\mathrm{flex2}}$	(5.358)	**x**	**x**	–	**x**	**x**	**x**	**x**
— transl. tol	$\mathrm{V}^{\mathrm{flex2-t}}$	–	**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**
$\theta_{\mathrm{ref}}(t)$	degrees	$\theta_{\mathrm{ref}}(t) = \omega t$	`rotation`	**x**	**x**
$\theta_{\mathrm{av}}(t)$	degrees	(5.360)	`rotation`	**x**	–
$\theta_{\mathrm{fit}}(t)$, $\theta_{\mathrm{fit}}(t,n)$	degrees	(5.362)	`rotangles`	–	**x**
$\mathbf{y}_0(n)$, $\mathbf{x}_0(t,n)$	nm	(5.351),(5.352)	`rotslabs`	–	**x**
$\tau(t)$	kJ/mol	(5.363)	`rotation`	**x**	–
$\tau(t,n)$	kJ/mol	(5.363)	`rottorque`	–	**x**

Angle of Rotation Groups: Fixed Axis

For fixed axis rotation, the average angle $\theta_{\rm av}(t)$ 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,

$$\theta_{\rm av} = \sum_{i=1}^{N} r_i\,\theta_i \Bigg/ \sum_{i=1}^{N} r_i\,. \tag{5.360}$$

Here, r_i is the distance of the reference position to the rotation axis, and the difference angles θ_i are determined from the atomic positions, projected onto a plane perpendicular to the rotation axis through pivot point $\mathbf{u}$ (see eqn. (5.331) for the definition of $\perp$),

$$\cos\theta_i = \frac{(\mathbf{y}_i - \mathbf{u})^{\perp}\cdot(\mathbf{x}_i - \mathbf{u})^{\perp}}{\|(\mathbf{y}_i - \mathbf{u})^{\perp}\cdot(\mathbf{x}_i - \mathbf{u})^{\perp}\|}\,. \tag{5.361}$$

The sign of $\theta_{\rm av}$ is chosen such that $\theta_{\rm av} > 0$ 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 $\theta_{\rm fit}$ as the angle by which the reference has to be rotated around $\hat{\mathbf{v}}$ for the optimal fit,

$$\mathrm{RMSD}\big(\mathbf{x}_i,\ \mathbf{\Omega}(\theta_{\rm fit})\mathbf{y}_i^0\big) \stackrel{!}{=} \min\,. \tag{5.362}$$

To determine the local angle for each slab n, both reference and actual positions are weighted with the Gaussian function of slab n, and $\theta_{\rm fit}(t,n)$ is calculated as in eqn. (5.362) from the Gaussian-weighted positions.

For all angles, the *mdp* (page 440) 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

$$\tau(t) = \sum_{i=1}^{N} \mathbf{r}_i(t)\times\mathbf{f}_i^{\perp}(t), \tag{5.363}$$

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 τ_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:

$$E(t) = E_0 \exp\left[-\frac{(t-t_0)^2}{2\sigma^2}\right] \cos\left[\omega(t-t_0)\right] \tag{5.364}$$

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 σ is the width of the pulse. Special cases occur when σ = 0 (non-pulsed field) and for ω is 0 (static field). See `electric-field-x` (page 70) for more details.

This simulated laser-pulse was applied to simulations of melting ice *146* (page 534). A pulsed electric field may look ike Fig. 5.47. 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 478)).

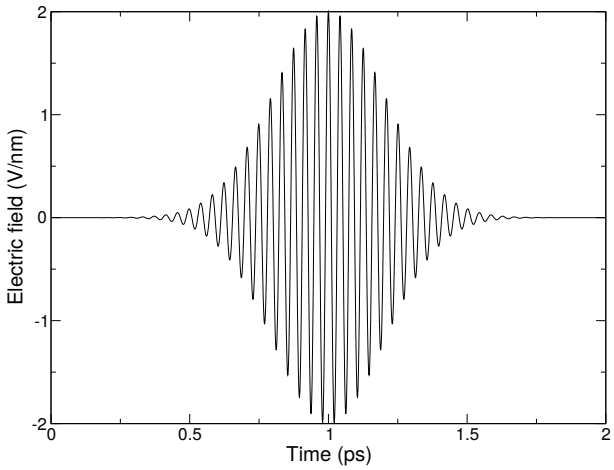


Fig. 5.47: A simulated laser pulse in GROMACS.

Electric fields are applied when the following options are specified in the *grompp* (page 166) *mdp* (page 440) file. You specify, in order, E_0, ω, t_0 and σ:

```
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, ω = 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, ω = 150/ps and t_0 = 5 ps and σ = 1 ps. Read more in ref. *146* (page 534). Note that the input file format is changed from the undocumented older version. A figure like Fig. 5.47 may be produced by passing the `-field` option to *gmx mdrun* (page 183).

5.8.8 Computational Electrophysiology

The Computational Electrophysiology (CompEL) protocol *147* (page 534) 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 Δq across the membrane, which gives rise to a potential difference ΔU according to the membrane capacitance:

$$\Delta U = \Delta q / C_{membrane} \tag{5.365}$$

The transmembrane electric field and concentration gradients are controlled by *mdp* (page 440) 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. 5.48 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.

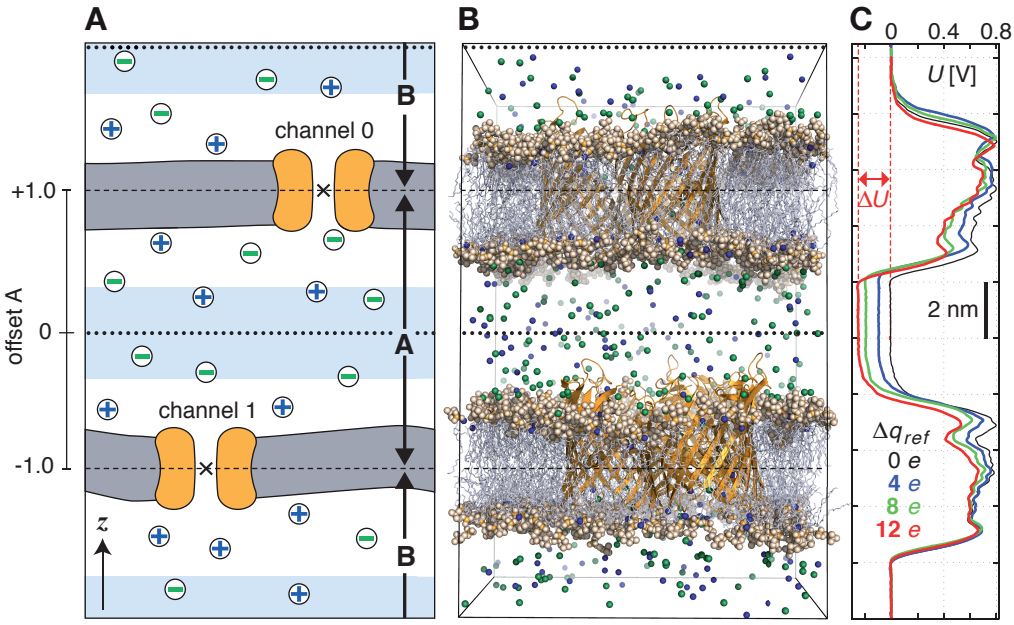


Fig. 5.48: Typical double-membrane setup for CompEL simulations (A, B). Ion/water molecule exchanges will be performed as needed between the two light blue volumes around the dotted black lines (A). Plot (C) shows the potential difference ΔU resulting from the selected charge imbalance Δq_{ref} between the compartments.

The potential difference ΔU across the membrane is easily calculated with the *gmx potential* (page 205) utility. By this, the potential drop along z or the pore axis is exactly known in each time interval of the simulation (Fig. 5.48 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 Δt is determined by:

$$G = \frac{\sum_i n_i q_i}{\Delta t \, \Delta U} \, . \tag{5.366}$$

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 440) 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. 5.48). 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. 5.48 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. 5.48 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 150) `-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 182) `-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 453)).

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 446) 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 442) file.

5.8.9 Calculating a PMF using the free-energy code

The free-energy coupling-parameter approach (see sec. *Free energy calculations* (page 345)) 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 409). 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 λ and time by setting delta-lambda in the *mdp* (page 440) 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 372)). These positions can be made a function of the coupling parameter λ. The positions for the A and the B states are supplied to *grompp* (page 166) 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 λ 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 388)). 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 482) and have previously been described in full detail *148* (page 534).

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 200). 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 166) 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 200) due to the complex nature of the construction needed for a whole aromatic group.

Hydrogen bond-angle vibrations

Construction of virtual sites

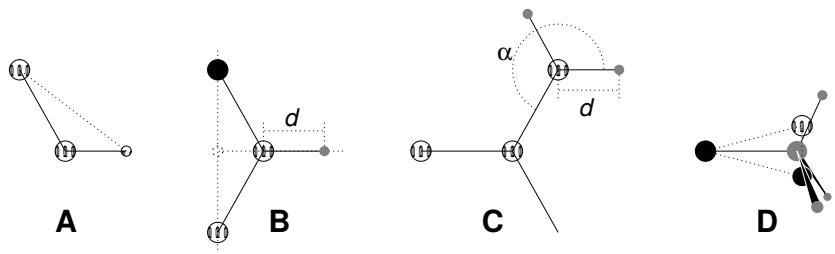


Fig. 5.49: The different types of virtual site constructions used for hydrogen atoms. The atoms used in the construction of the virtual site(s) are depicted as black circles, virtual sites as gray ones. Hydrogens are smaller than heavy atoms. A: fixed bond angle, note that here the hydrogen is not a 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. 5.49).

- *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. 5.49 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. 5.49 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. 5.49 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. 5.49 D.

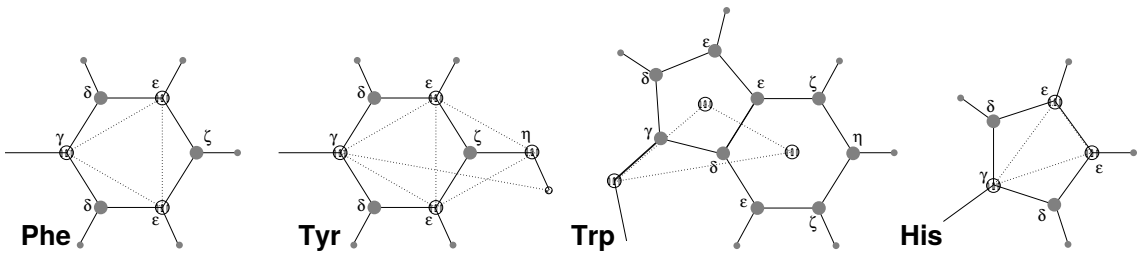


Fig. 5.50: The different types of virtual site constructions used for aromatic residues. The atoms used in the construction of the virtual site(s) are depicted as black circles, virtual sites as gray ones. Hydrogens are smaller than heavy atoms. A: phenylalanine; B: tyrosine (note that the hydroxyl hydrogen 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 388)) from these three. Each of the aromatic residues require a different approach:

- *Phenylalanine:* C_γ, $C_{\epsilon 1}$, 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. 5.49 A.
- *Tyrosine:* The ring is treated identically to the phenylalanine ring. Additionally, constraints are defined between $C_{\epsilon 1}$, $C_{\epsilon 2}$, and O_η. The original improper dihedral angles will keep both triangles (one for the ring and one with O_η) 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_γ and H_η. **Note** that the hydrogen is not treated as a virtual site. See Fig. 5.49 B.
- *Tryptophan:* C_β 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. 5.49 C.
- *Histidine:* C_γ, $C_{\epsilon 1}$ 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. 5.49 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:

$$\eta = \frac{1}{2}\frac{V}{k_B T}\lim_{t\to\infty}\frac{\mathrm{d}}{\mathrm{d}t}\left\langle\left(\int_{t_0}^{t_0+t} P_{xz}(t')\mathrm{d}t'\right)^2\right\rangle_{t_0} \tag{5.367}$$

This can be done with *gmx energy* (page 155). This method converges very slowly *149* (page 534), and as such a nanosecond simulation might not be long enough 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.

GROMACS also has a non-equilibrium method for determining the viscosity *149* (page 534). 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:

$$a_x(z) + \frac{\eta}{\rho}\frac{\partial^2 v_x(z)}{\partial z^2} = 0 \tag{5.368}$$

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:

$$a_x(z) = A\cos\left(\frac{2\pi z}{l_z}\right) \tag{5.369}$$

where l_z is the height of the box. The generated velocity profile is:

$$v_x(z) = V\cos\left(\frac{2\pi z}{l_z}\right) \tag{5.370}$$

$$V = A\frac{\rho}{\eta}\left(\frac{l_z}{2\pi}\right)^2 \tag{5.371}$$

The viscosity can be calculated from A and V:

$$\eta = \frac{A}{V}\rho\left(\frac{l_z}{2\pi}\right)^2 \tag{5.372}$$

In the simulation V is defined as:

$$V = \frac{\sum_{i=1}^{N} m_i v_{i,x} 2\cos\left(\frac{2\pi z}{l_z}\right)}{\sum_{i=1}^{N} m_i} \tag{5.373}$$

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:

$$\mathrm{sh}_{\max} = \max_z\left|\frac{\partial v_x(z)}{\partial z}\right| = A\frac{\rho}{\eta}\frac{l_z}{2\pi} \tag{5.374}$$

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 528), which can be written in terms of the shear rate as:

$$T_s = \frac{\eta\,\tau}{2\rho\,C_v}\mathrm{sh}_{\mathrm{max}}^2 \tag{5.375}$$

where τ 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}]sh$^2_{\mathrm{max}}$. 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 ((5.372)).

5.8.12 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:

$$V_s(x) = A_0 + A_1\,\epsilon + A_2\,\epsilon^2 + A_3\,\epsilon^3 \tag{5.376}$$

where the table spacing h and fraction ϵ are given by:

$$\begin{aligned} h &= x_{i+1} - x_i \\ \epsilon &= (x - x_i)/h \end{aligned} \tag{5.377}$$

so that $0 \leq \epsilon < 1$. From this, we can calculate the derivative in order to determine the forces:

$$-V_s'(x) \;=\; -\frac{\mathrm{d}V_s(x)}{\mathrm{d}\epsilon}\frac{\mathrm{d}\epsilon}{\mathrm{d}x} \;=\; -(A_1 + 2A_2\,\epsilon + 3A_3\,\epsilon^2)/h \tag{5.378}$$

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:

$$\begin{aligned} |V_s - V|_{max} &= V'''' \frac{h^4}{384} + O(h^5) \\ |V_s' - V'|_{max} &= V'''' \frac{h^3}{72\sqrt{3}} + O(h^4) \\ |V_s'' - V''|_{max} &= V'''' \frac{h^2}{12} + O(h^3) \end{aligned} \tag{5.379}$$

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 ($\epsilon = s\mathrm{r}_{ij} - n_0$) and ϵ^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

$$V(r_{ij}) \; = \; \frac{q_i q_j}{4\pi\epsilon_0} f(r_{ij}) + C_6\, g(r_{ij}) + C_{12}\, h(r_{ij}) \tag{5.380}$$

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 440) file:

```
rlist           = 1.0
coulombtype     = User
rcoulomb        = 1.0
vdwtype         = User
rvdw            = 1.0
```

mdrun (page 183) 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 183) 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 440) 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 440) 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.

5.8.13 Mixed Quantum-Classical simulation techniques

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 interfaces to several popular Quantum Chemistry packages (MOPAC *150* (page 534), GAMESS-UK *151* (page 534), Gaussian *152* (page 534) and CPMD *153* (page 534)).

GROMACS interactions between the two subsystems are either handled as described by Field et al. *154* (page 534) or within the ONIOM approach by Morokuma and coworkers *155* (page 534), *156* (page 534).

Overview

Two approaches for describing the interactions between the QM and MM subsystems are supported in this version:

1. **Electronic Embedding** The electrostatic interactions between the electrons of the QM region and the MM atoms and between the QM nuclei and the MM atoms are included in the Hamiltonian for the QM subsystem:

$$H^{QM/MM} = H_e^{QM} - \sum_i^n \sum_J^M \frac{e^2 Q_J}{4\pi\epsilon_0 r_{iJ}} + \sum_A^N \sum_J^M \frac{e^2 Z_A Q_J}{e\pi\epsilon_0 R_{AJ}},$$

   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. Bonded interactions between QM and MM atoms are described at the MM level by the appropriate force-field terms. Chemical bonds that connect the two subsystems are capped by 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.

2. **ONIOM** In the ONIOM approach, the energy and gradients are first evaluated for the isolated QM subsystem at the desired level of *ab initio* theory. Subsequently, the energy and gradients of the total system, including the QM region, are computed using the molecular mechanics force field and added to the energy and gradients calculated for the isolated QM subsystem. Finally, in order to correct for counting the interactions inside the QM region twice, a molecular mechanics calculation is performed on the isolated QM subsystem and the energy and gradients are subtracted. This leads to the following expression for the total QM/MM energy (and gradients likewise):

$$E_{tot} = E_I^{QM} + E_{I+II}^{MM} - E_I^{MM},$$

   where the subscripts I and II refer to the QM and MM subsystems, respectively. The superscripts indicate at what level of theory the energies are computed. The ONIOM scheme has the advantage that it is not restricted to a two-layer QM/MM description, but can easily handle more than two layers, with each layer described at a different level of theory.

Usage

QMMM is currently not supported in GROMACS.

5.8.14 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 *Mixed Quantum-Classical simulation techniques* (page 487). 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 descibing 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 535). 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:

$$E^{es}_{QM/MM} = -\sum_{a}^{N_{mm}} Q_a \int \rho(\mathbf{r}) \frac{r^4_{c,a} - |\mathbf{R_a} - \mathbf{r}|^4}{r^5_{c,a} - |\mathbf{R_a} - \mathbf{r}|^5} d\mathbf{r} + \sum_{a}^{N_{mm}} \sum_{n}^{N_{qm}} Q_a Z_n \frac{r^4_{c,a} - |\mathbf{R_a} - \mathbf{R_n}|^4}{r^5_{c,a} - |\mathbf{R_a} - \mathbf{R_n}|^5}$$

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 535).

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 535).

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 15), 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 440) file for a MiMiC simulation one needs to add two options:

1. `integrator=mimic` (page 37) 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 440) options relating to force calculation and output are active.

After setting up the *mdp* (page 440) file one can run *grompp* (page 166) as usual. *grompp* (page 166) 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

&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 exansion 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
```

5.8.15 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.

5.8.16 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 440) option `IMD-group`. When `IMD-group` is empty, the IMD protocol is disabled and cannot be enabled via the switches in *mdrun* (page 183). To interact with the entire system, `IMD-group` can be set to `System`. When using *grompp* (page 166), a *gro* (page 437) file to be used as VMD input is written out (`-imd` switch of *grompp* (page 166)).

Starting the simulation

Communication between VMD and GROMACS is achieved via TCP sockets and thus enables controlling an *mdrun* (page 183) running locally or on a remote cluster. The port for the connection can be specified with the `-imdport` switch of *mdrun* (page 183), 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 183) 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 183). 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 183) are hidden by default and show up in the help text only with *gmx mdrun* (page 183) `-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^{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.s

5.8.17 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 534) and later implemented as g_membed tool *158* (page 534). 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.

5.8.18 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 *182* (page 535), and *183* (page 535).

Overview

The forces that are applied depend on:

- The forward model that describes how atom positions are translated into a simulated density, $\rho^{\mathrm{sim}}(\mathbf{r})$.
- The similarity measure that describes how close the simulated density is to the reference density, ρ^{ref}, $S[\rho^{\mathrm{ref}}, \rho^{\mathrm{sim}}(\mathbf{r})]$.
- The scaling of these forces by a force constant, k.

The resulting effective potential energy is

$$U = U_{\mathrm{forcefield}}(\mathbf{r}) - kS[\rho^{\mathrm{ref}}, \rho^{\mathrm{sim}}(\mathbf{r})] \,. \tag{5.381}$$

The corresponding density based forces that are added during the simulation are

$$\mathbf{F}_{\mathrm{density}} = k\nabla_{\mathbf{r}} S[\rho^{\mathrm{ref}}, \rho^{\mathrm{sim}}(\mathbf{r})] \,. \tag{5.382}$$

This derivative decomposes into a similarity measure derivative and a simulated density model derivative, summed over all density voxels $\mathbf{v}$

$$\mathbf{F}_{\mathrm{density}} = k \sum_{\mathbf{v}} \partial_{\rho^{\mathrm{sim}}_{\mathbf{v}}} S[\rho^{\mathrm{ref}}, \rho^{\mathrm{sim}}] \cdot \nabla_{\mathbf{r}} \rho^{\mathrm{sim}}_{\mathbf{v}}(\mathbf{r}) \,. \tag{5.383}$$

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

$$\rho_{\mathbf{v}} = \sum_i A_i \frac{1}{\sqrt{2\pi}^3 \sigma^3} \exp[-\frac{(\mathbf{r_i} - \mathbf{v})^2}{2\sigma^2}] \,. \tag{5.384}$$

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 σ. 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

$$\nabla_r \rho_{\mathbf{v}}^{\mathrm{sim}}(\mathbf{r}) = \sum_i -A_i \frac{(\mathbf{r_i} - \mathbf{v})}{\sigma} \frac{1}{\sqrt{2\pi}^3 \sigma^3} \exp[-\frac{(\mathbf{r_i} - \mathbf{v})^2}{2\sigma^2}] \,. \tag{5.385}$$

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,

$$S_{\mathrm{inner-product}}[\rho^{\mathrm{ref}}, \rho^{\mathrm{sim}}] = \frac{1}{N_{\mathrm{voxel}}} \sum_{v=1}^{N_{\mathrm{voxel}}} \rho_v^{\mathrm{ref}} \rho_v^{\mathrm{sim}} \,. \tag{5.386}$$

The negative relative entropy between two densities,

$$S_{\mathrm{relative-entropy}}[\rho^{\mathrm{ref}}, \rho^{\mathrm{sim}}] = \sum_{v=1, \rho^{\mathrm{ref}}>0, \rho^{\mathrm{sim}}>0}^{N_{\mathrm{voxel}}} \rho^{\mathrm{ref}} [\log(\rho_v^{\mathrm{sim}}) - \log(\rho_v^{\mathrm{ref}})] \,. \tag{5.387}$$

The cross correlation between two densities,

$$S_{\mathrm{cross-correlation}}[\rho^{\mathrm{ref}}, \rho^{\mathrm{sim}}] = \frac{\sum_v \left((\rho_v^{\mathrm{ref}} - \bar{\rho}^{\mathrm{ref}})(\rho_v^{\mathrm{sim}} - \bar{\rho}^{\mathrm{sim}}) \right)}{\sqrt{\sum_v (\rho_v^{\mathrm{ref}} - \bar{\rho}^{\mathrm{ref}})^2 \sum_v (\rho_v^{\mathrm{sim}} - \bar{\rho}^{\mathrm{sim}})^2}} . \tag{5.388}$$

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_{atoms}.
- Spreading range in multiples of Gaussian width, N_σ.
- The ratio of spreading width to the input density grid spacing, r_σ.
- The number of voxels of the input density, N_{voxel}.
- Frequency of force calculations, N_{force}.
- The communication cost when using multiple ranks, that is reflected in a constant c_{comm}.

The overall cost of the density-guided simulation is approximately proportional to

$$\frac{1}{N_{\mathrm{force}}} \left[N_{\mathrm{atoms}} \left(N_\sigma r_\sigma \right)^3 + c_{\mathrm{comm}} N_{\mathrm{voxel}} \right] \,. \tag{5.389}$$

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 π. 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 73) 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 $1 + \delta t_{\mathrm{density}}/tau$, where $\delta t_{\mathrm{density}}$ 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 $1+2\delta t_{\mathrm{density}}/tau$. 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

$$U = U_{\text{forcefield}}(\mathbf{r}) - kS[\rho^{\text{ref}}, \rho^{\text{sim}}(\mathbf{A}\mathbf{r} + \mathbf{v}_{\text{shift}})] \,. \tag{5.390}$$

$$\mathbf{F}_{\text{density}} = k\nabla_{\mathbf{r}} S[\rho^{\text{ref}}, \rho^{\text{sim}}(\mathbf{A}\mathbf{r} + \mathbf{v}_{\text{shift}})] \,. \tag{5.391}$$

Affine transformations may be used, amongst other things, to perform

- rotations, e.g., around the z-axis by an angle θ 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.

5.9 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 166) `-h`) to see the same information, as well as `gmx help grompp`. The list of all programs are available from *gmx help* (page 176).

5.9.2 File types

Information about different file types can be found in *File formats* (page 434).

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 440) parameters can be found at under the link above both in your local GROMACS installation, or *here* (page 35).

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 104). The output files are often produced as finished Grace/Xmgr graphs.

First, in sec. *Using Groups* (page 499), the group concept in analysis is explained. *Selections* (page 501) 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 311), it was explained how *groups of atoms* can be used in *mdrun* (page 183) (see sec. *The group concept* (page 314)). 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:

$$4\pi r^2 g_{AB}(r) \; = \; V \sum_{i \in A}^{N_A} \sum_{j \in B}^{N_B} P(r) \tag{5.392}$$

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 441) `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 182) or *gmx select* (page 221). To generate an index file with angles or dihedrals, use *gmx mk_angndx* (page 189). 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 166), 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_α atoms

Backbone

protein backbone atoms; N, C_α and C

MainChain

protein main chain atoms: N, C_α, C and O, including oxygens in C-terminus

MainChain+Cb

protein main chain atoms including C_β

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_α, 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 404); 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 221)

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 441) 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 221) 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 442) 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 176) selections. The help is divided into subtopics that can be accessed with, e.g., `help syntax`/ *gmx help* (page 176) `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 221) -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.

5.10.2 Looking at your trajectory

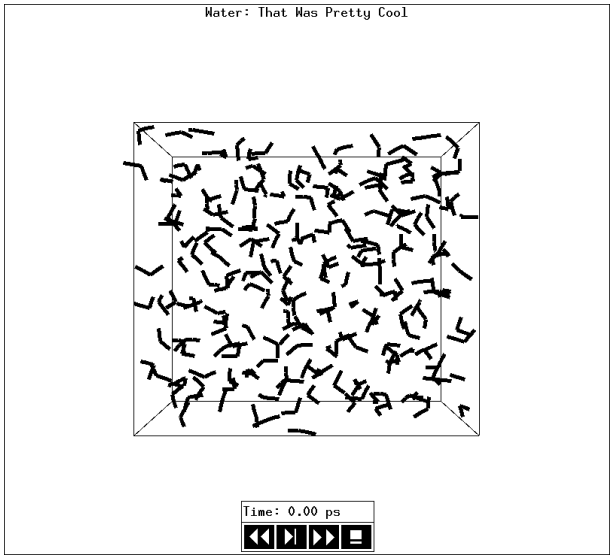


Fig. 5.51: The window of *gmx view* (page 248) showing a box of water.

gmx view (page 248)
Before analyzing your trajectory it is often informative to look at your trajectory first. GROMACS comes with a simple trajectory viewer *gmx view* (page 248); the advantage with this one is that it does not require OpenGL, which usually isn't present on *e.g.* supercomputers. It is also possible to generate a hard-copy in Encapsulated Postscript format (see Fig. 5.51). If you want a faster and more fancy viewer there are several programs that can read the GROMACS trajectory formats – have a look at our webpage for updated links.

5.10.3 General properties

gmx energy (page 155), *gmx traj* (page 233)
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 155). 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

$$\mathbf{v}_{com} = \frac{1}{M} \sum_{i=1}^{N} m_i \mathbf{v}_i \tag{5.393}$$

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 233) `-com -ov`. It is however recommended to remove the center-of-mass velocity every step (see chapter *Algorithms* (page 311))!

5.10.4 Radial distribution functions

gmx rdf (page 207)
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:

$$\begin{aligned} g_{AB}(r) &= \frac{\langle \rho_B(r) \rangle}{\langle \rho_B \rangle_{local}} \\ &= \frac{1}{\langle \rho_B \rangle_{local}} \frac{1}{N_A} \sum_{i \in A}^{N_A} \sum_{j \in B}^{N_B} \frac{\delta(r_{ij} - r)}{4\pi r^2} \end{aligned} \tag{5.394}$$

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).

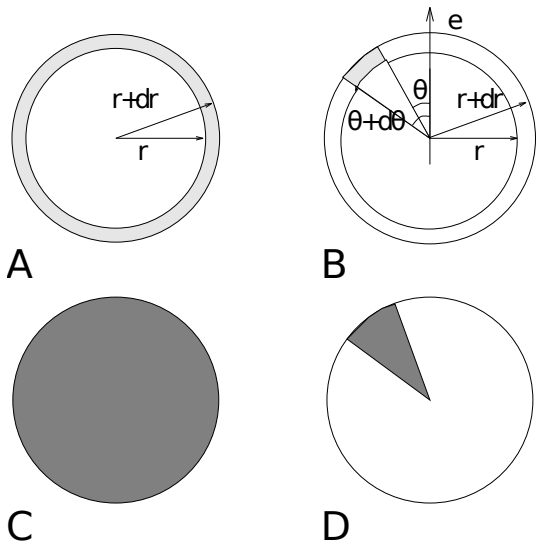


Fig. 5.52: Definition of slices in *gmx rdf* (page 207): A. $g_{AB}(r)$. B. $g_{AB}(r, \theta)$. The slices are colored gray. C. Normalization $\langle \rho_B \rangle_{local}$. D. Normalization $\langle \rho_B \rangle_{local,\,\theta}$. Normalization volumes are colored gray.

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 207) divides the system into spherical slices (from r to $r + dr$, see Fig. 5.52 A) and makes a histogram in stead of the δ-function. An example of the RDF of oxygen-oxygen in SPC water :ref:*80* (page 530) is given in Fig. 5.53

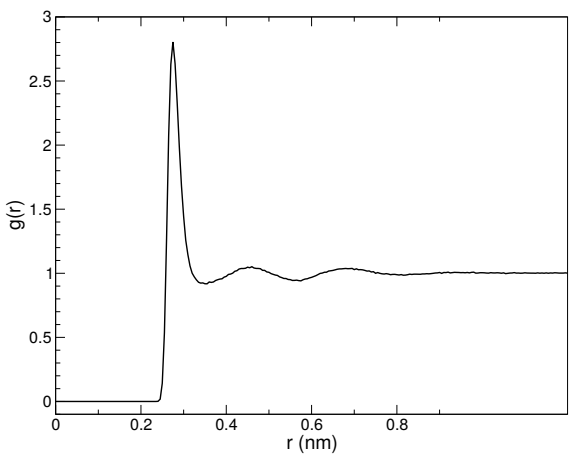


Fig. 5.53: $g_{OO}(r)$ for Oxygen-Oxygen of SPC-water.

With *gmx rdf* (page 207) it is also possible to calculate an angle dependent rdf $g_{AB}(r,\theta)$, where the angle θ is defined with respect to a certain laboratory axis **e**, see Fig. 5.52 B.

$$g_{AB}(r,\theta) = \frac{1}{\langle\rho_B\rangle_{local,\,\theta}} \frac{1}{N_A} \sum_{i\in A}^{N_A} \sum_{j\in B}^{N_B} \frac{\delta(r_{ij}-r)\delta(\theta_{ij}-\theta)}{2\pi r^2 sin(\theta)} \tag{5.395}$$

$$cos(\theta_{ij}) = \frac{\mathbf{r}_{ij}\cdot\mathbf{e}}{\|r_{ij}\|\,\|e\|} \tag{5.396}$$

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 θ 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 532). 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:

$$C_f(t) \;=\; \langle f(\xi)f(\xi+t)\rangle_\xi \tag{5.397}$$

where the notation on the right hand side indicates averaging over ξ, *i.e.* over time origins. It is also possible to compute cross-correlation function from two properties $f(t)$ and $g(t)$:

$$C_{fg}(t) \;=\; \langle f(\xi)g(\xi+t)\rangle_\xi \tag{5.398}$$

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 τ_f:

$$\tau_f \;=\; \int_0^\infty C_f(t)\mathrm{d}t \tag{5.399}$$

In practice, correlation functions are calculated based on data points with discrete time intervals Δt, so that the ACF from an MD simulation is:

$$C_f(j\Delta t) \;=\; \frac{1}{N-j}\sum_{i=0}^{N-1-j} f(i\Delta t)f((i+j)\Delta t) \tag{5.400}$$

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 Δt. 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 (5.400) 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:

$$C_f(j\Delta t) \;=\; \frac{1}{M}\sum_{i=0}^{N-1-M} f(i\Delta t)f((i+j)\Delta t) \tag{5.401}$$

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 ξ ((5.397)) 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 $kM \leq N$):

$$C_f(j\Delta t) \;=\; \frac{1}{k}\sum_{i=0}^{k-1} f(iM\Delta t)f((iM+j)\Delta t) \tag{5.402}$$

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 (5.400) is proportional to N^2, which is considerable. However, this can be improved by using fast Fourier transforms to do the convolution *108* (page 532).

Special forms of the ACF

There are some important varieties on the ACF, *e.g.* the ACF of a vector $\mathbf{p}$:

$$C_{\mathbf{p}}(t) \;=\; \int_0^\infty P_n(\cos\angle\left(\mathbf{p}(\xi),\mathbf{p}(\xi+t)\right)\,\mathrm{d}\xi \tag{5.403}$$

where $P_n(x)$ is the nth 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 1st and 2nd order Legendre polynomial ((5.403)). This can also be used for rotational autocorrelation (*gmx rotacf* (page 214)) and dipole autocorrelation (*gmx dipoles* (page 140)).

In order to study torsion angle dynamics, we define a dihedral autocorrelation function as *159* (page 534):

$$C(t) \;=\; \langle\cos(\theta(\tau)-\theta(\tau+t))\rangle_\tau \tag{5.404}$$

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:

$$C(t) \;=\; \langle\cos(\theta(\tau))\cos(\theta(\tau+t)) \;+\; \sin(\theta(\tau))\sin(\theta(\tau+t))\rangle_\tau \tag{5.405}$$

Some Applications

The program *gmx velacc* (page 246) calculates the *velocity autocorrelation function*.

$$C_{\mathbf{v}}(\tau) \;=\; \langle\mathbf{v}_i(\tau)\cdot\mathbf{v}_i(0)\rangle_{i\in A} \tag{5.406}$$

The self diffusion coefficient can be calculated using the Green-Kubo relation *108* (page 532):

$$D_A \;=\; \frac{1}{3}\int_0^\infty \langle\mathbf{v}_i(t)\cdot\mathbf{v}_i(0)\rangle_{i\in A}\;dt \tag{5.407}$$

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 507)), which can also be used for the computation of diffusion constants. However, Allen & Tildesley *108* (page 532) 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 140):

$$C_\mu(\tau) \;=\; \langle\mu_i(\tau)\cdot\mu_i(0)\rangle_{i\in A} \tag{5.408}$$

[1] $P_0(x)=1$, $P_1(x)=x$, $P_2(x)=(3x^2-1)/2$

with $\mu_i = \sum_{j \in i} \mathbf{r}_j q_j$. The dipole correlation time can be computed using (5.399). For some applications see (**???**).

The viscosity of a liquid can be related to the correlation time of the Pressure tensor $\mathbf{P}$ *160* (page 534), *161* (page 534). *gmx energy* (page 155) can compute the viscosity, but this is not very accurate *149* (page 534), 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 534) (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 534):

$$\varepsilon^2(t) = 2\alpha\tau_1 \left(1 + \frac{\tau_1}{t}\left(e^{-t/\tau_1} - 1\right)\right) + 2(1-\alpha)\tau_2 \left(1 + \frac{\tau_2}{t}\left(e^{-t/\tau_2} - 1\right)\right) \tag{5.409}$$

where τ_1 and τ_2 are time constants (with $\tau_2 \geq \tau_1$) and α 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 112) for error estimation using

$$\lim_{t \to \infty} \varepsilon(t) = \sigma \sqrt{\frac{2(\alpha\tau_1 + (1-\alpha)\tau_2)}{T}} \tag{5.410}$$

where σ is the standard deviation of the data set and T is the total simulation time *149* (page 534).

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

$$f(x) = \frac{a_0 + a_1}{2} - \frac{a_0 - a_1}{2} \mathrm{erf}\left(\frac{x - a_2}{a_3^2}\right) \tag{5.411}$$

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 534). This is implemented in *gmx densorder* (page 138).

Transverse current autocorrelation function

In order to establish the transverse current autocorrelation function (useful for computing viscosity *164* (page 534)) the following function is fitted:

$$f(x) \;=\; e^{-\nu}\left(\cosh(\omega\nu) + \frac{\sinh(\omega\nu)}{\omega}\right) \tag{5.412}$$

with $\nu = x/(2a_0)$ and $\omega = \sqrt{1-a_1}$. This is implemented in *gmx tcaf* (page 231).

Viscosity estimation from pressure autocorrelation function

The viscosity is a notoriously difficult property to extract from simulations *149* (page 534), *165* (page 534). It is *in principle* possible to determine it by integrating the pressure autocorrelation function *160* (page 534), 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 534):

$$f(t)/f(0) = (1-C)\cos(\omega t)e^{-(t/\tau_f)^{\beta_f}} + Ce^{-(t/\tau_s)^{\beta_s}} \tag{5.413}$$

where ω is the frequency of rapid pressure oscillations (mainly due to bonded forces in molecular simulations), τ_f and β_f are the time constant and exponent of fast relaxation in a stretched-exponential approximation, τ_s and β_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:

$$\eta = \frac{V}{k_B T}\int_0^\infty f(t)dt \tag{5.414}$$

This equation has been applied to computing the bulk and shear viscosity using different elements from the pressure tensor *167* (page 535).

5.10.7 Mean Square Displacement

gmx msd (page 190)
To determine the self diffusion coefficient D_A of particles of type A, one can use the Einstein relation *108* (page 532):

$$\lim_{t\to\infty}\langle\|\mathbf{r}_i(t)-\mathbf{r}_i(0)\|^2\rangle_{i\in A} \;=\; 6D_A t \tag{5.415}$$

This *mean square displacement* and D_A are calculated by the program *gmx msd* (page 190). 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 190) 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.

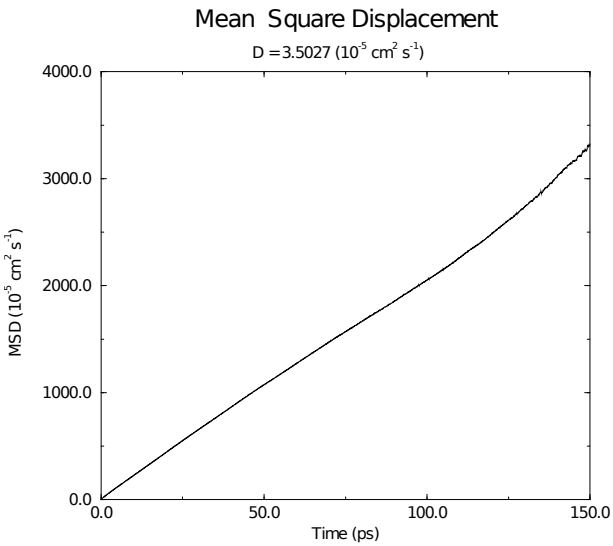


Fig. 5.54: Mean Square Displacement of SPC-water.

5.10.8 Bonds/distances, angles and dihedrals

gmx distance (page 144), *gmx angle* (page 115), *gmx gangle* (page 161)

To monitor specific *bonds* in your modules, or more generally distances between points, the program *gmx distance* (page 144) 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 115) 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
```

(continues on next page)

(continued from previous page)

```
[ 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.

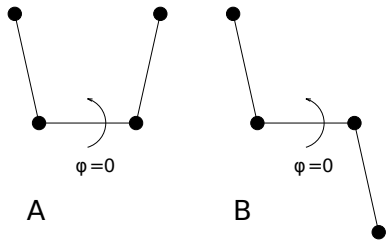


Fig. 5.55: Dihedral conventions: A. "Biochemical convention". B. "Polymer convention".

The program *gmx gangle* (page 161) 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 115), 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 161) uses the normal vector perpendicular to the plane. See Fig. 5.56 A, B, C) for the definitions.

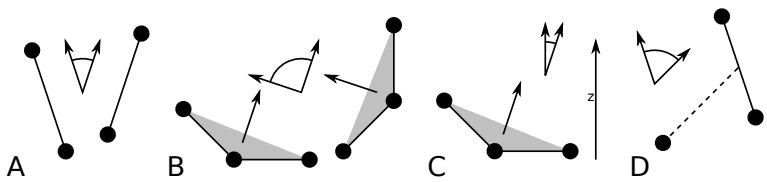


Fig. 5.56: Angle options of *gmx gangle* (page 161): A. Angle between two vectors. B. Angle between two planes. C. Angle between a vector and the z axis. D. Angle between a vector and the normal of a sphere. Also other combinations are supported: planes and vectors can be used interchangeably.

5.10.9 Radius of gyration and distances

gmx gyrate (page 168), *gmx distance* (page 144), *gmx mindist* (page 188), *gmx mdmat* (page 183), *gmx pairdist* (page 199), *gmx xpm2ps* (page 254)

To have a rough measure for the compactness of a structure, you can calculate the *radius of gyration* with the program *gmx gyrate* (page 168) as follows:

$$R_g = \left(\frac{\sum_i \|\mathbf{r}_i\|^2 m_i}{\sum_i m_i} \right)^{\frac{1}{2}} \tag{5.416}$$

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

$$R_{g,x} = \left(\frac{\sum_i \left(r_{i,y}^2 + r_{i,z}^2 \right) m_i}{\sum_i m_i} \right)^{\frac{1}{2}} \tag{5.417}$$

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 144), as explained in sec. *Bonds/distances, angles and dihedrals* (page 508).
- The *minimum distance* between two groups of atoms during time can be calculated with the program *gmx mindist* (page 188). It also calculates the *number of contacts* between these groups within a certain radius r_{max}.
- *gmx pairdist* (page 199) is a selection-enabled version of *gmx mindist* (page 188).
- To monitor the *minimum distances between amino acid residues* within a (protein) molecule, you can use the program *gmx mdmat* (page 183). This minimum distance between two residues A_i and A_j is defined as the smallest distance between any pair of atoms (i $\in$ A_i, j $\in$ 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 254) into a Postscript Fig. 5.57.

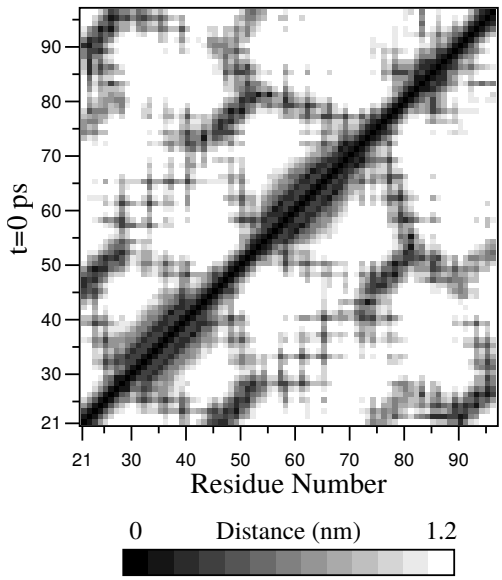


Fig. 5.57: A minimum distance matrix for a peptide *168* (page 535).

- 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 210), *gmx rmsdist* (page 211)

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 210) by least-square fitting the structure to the reference structure ($t_2 = 0$) and subsequently calculating the $RMSD$ ((5.418)).

$$RMSD(t_1, t_2) \;=\; \left[\frac{1}{M}\sum_{i=1}^{N} m_i \|\mathbf{r}_i(t_1) - \mathbf{r}_i(t_2)\|^2\right]^{\frac{1}{2}} \tag{5.418}$$

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_α, 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 (5.418) with a structure at time $t_2 = t_1 - \tau$. This gives some insight in the mobility as a function of τ. 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 211):

$$RMSD(t) \;=\; \left[\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} \|\mathbf{r}_{ij}(t) - \mathbf{r}_{ij}(0)\|^2\right]^{\frac{1}{2}} \tag{5.419}$$

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 535), can find correlated motions. It uses the covariance matrix C of the atomic coordinates:

$$C_{ij} = \left\langle M_{ii}^{\frac{1}{2}}(x_i - \langle x_i\rangle) M_{jj}^{\frac{1}{2}}(x_j - \langle x_j\rangle)\right\rangle \tag{5.420}$$

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:

$$R^T C R = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{3N}) \quad \text{where } \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{3N} \tag{5.421}$$

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 $p_i(t)$:

$$\mathbf{p}(t) = R^T M^{\frac{1}{2}}(\mathbf{x}(t) - \langle\mathbf{x}\rangle) \tag{5.422}$$

The eigenvalue λ_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:

$$\mathbf{x}^f(t) = \langle\mathbf{x}\rangle + M^{-\frac{1}{2}} R_{*i}\, p_i(t) \tag{5.423}$$

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 535). 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 λ_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:

$$A^{\frac{1}{2}} = R\,\mathrm{diag}(\lambda_1^{\frac{1}{2}}, \lambda_2^{\frac{1}{2}}, \ldots, \lambda_{3N}^{\frac{1}{2}})\,R^T \tag{5.424}$$

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:

$$\begin{aligned} d(A,B) &= \sqrt{\mathrm{tr}\left(\left(A^{\frac{1}{2}} - B^{\frac{1}{2}}\right)^2\right)} \\ &= \sqrt{\mathrm{tr}\left(A + B - 2A^{\frac{1}{2}}B^{\frac{1}{2}}\right)} \\ &= \left(\sum_{i=1}^{N}\left(\lambda_i^A + \lambda_i^B\right) - 2\sum_{i=1}^{N}\sum_{j=1}^{N}\sqrt{\lambda_i^A\lambda_j^B}\left(R_i^A \cdot R_j^B\right)^2\right)^{\frac{1}{2}} \end{aligned} \tag{5.425}$$

where tr is the trace of a matrix. We can now define the overlap s as:

$$s(A,B) = 1 - \frac{d(A,B)}{\sqrt{\mathrm{tr}A + \mathrm{tr}B}} \tag{5.426}$$

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:

$$\mathrm{overlap}(\mathbf{v}, \mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}(\mathbf{v}_i \cdot \mathbf{w}_j)^2 \tag{5.427}$$

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 535), *171* (page 535). The eigenvalues are proportional to the index to the power -2. The cosine content is defined as:

$$\frac{2}{T}\left(\int_0^T \cos\left(\frac{i\pi t}{T}\right) p_i(t)\mathrm{d}t\right)^2 \left(\int_0^T p_i^2(t)\mathrm{d}t\right)^{-1} \tag{5.428}$$

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 132). The principal components and overlap (and many more things) can be plotted and analyzed with *gmx anaeig* (page 110). The cosine content can be calculated with *gmx analyze* (page 112).

5.10.12 Dihedral principal component analysis

gmx angle (page 115), *gmx covar* (page 132), *gmx anaeig* (page 110)

Principal component analysis can be performed in dihedral space *172* (page 535) using GROMACS. You start by defining the dihedral angles of interest in an index file, either using *gmx mk_angndx* (page 189) or otherwise. Then you use the *gmx angle* (page 115) program with the `-or` flag to produce a new *trr* (page 447) file containing the cosine and sine of each dihedral angle in two coordinates, respectively. That is, in the *trr* (page 447) 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 447) file as input for the *gmx covar* (page 132) program and perform principal component analysis as usual. For this to work you will need to generate a reference file (*tpr* (page 446), *gro* (page 437), *pdb* (page 442) etc.) containing the same number of "atoms" as the new *trr* (page 447) 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 132) since the coordinates in the dummy reference file do not correspond in any way to the information in the *trr* (page 447) file. Analysis of the results is done using *gmx anaeig* (page 110).

5.10.13 Hydrogen bonds

gmx hbond (page 170)

The program *gmx hbond* (page 170) 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:

$$\begin{aligned} r &\leq r_{HB} &= 0.35 \text{ nm} \\ \alpha &\leq \alpha_{HB} &= 30^o \end{aligned} \tag{5.429}$$

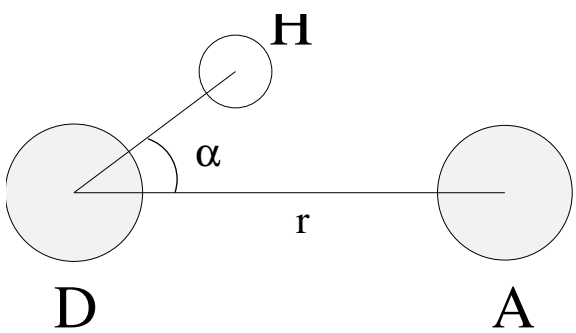


Fig. 5.58: Geometrical Hydrogen bond criterion.

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 170) 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 (α) 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 α-helices or β-turns or strands.

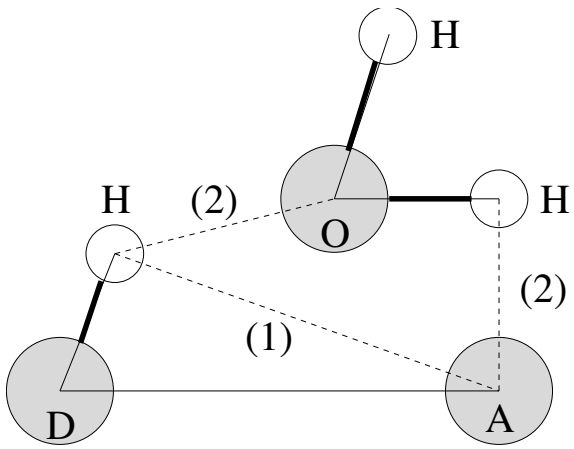


Fig. 5.59: Insertion of water into an H-bond. (1) Normal H-bond between two residues. (2) H-bonding bridge via a water molecule.

- 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:

$$C(\tau) \; = \; \langle s_i(t) \, s_i(t+\tau) \rangle \tag{5.430}$$

- 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 τ_{HB}:

$$\tau_{HB} \; = \; \int_0^\infty C(\tau) d\tau \tag{5.431}$$

- 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 τ_{HB}. A more complete analysis is given in ref. *173* (page 535); one of the more fancy option is the Luzar and Chandler analysis of hydrogen bond kinetics *174* (page 535), *175* (page 535).
- 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 do_dssp (page 145), *gmx rama* (page 207), *gmx wheel* (page 252)

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 509)), or calculate the RMSD (sec. *Root mean square deviations in structure* (page 510)).

You can also look at the changing of *secondary structure elements* during your run. For this, you can use the program *gmx do_dssp* (page 145), which is an interface for the commercial program `DSSP` *176* (page 535). For further information, see the `DSSP` manual. A typical output plot of *gmx do_dssp* (page 145) is given in Fig. 5.60.

One other important analysis of proteins is the so-called *Ramachandran plot*. This is the projection of the structure on the two dihedral angles ϕ and ψ of the protein backbone, see Fig. 5.61:

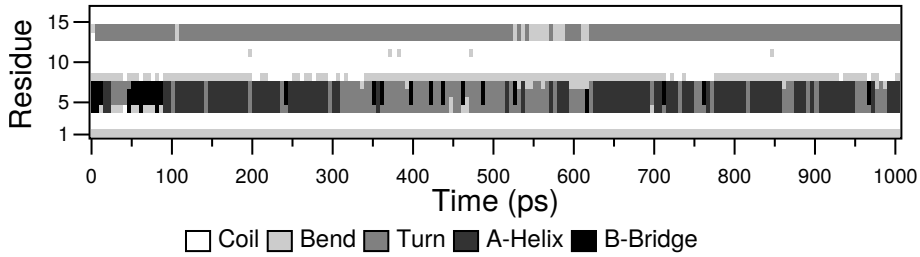


Fig. 5.60: Analysis of the secondary structure elements of a peptide in time.

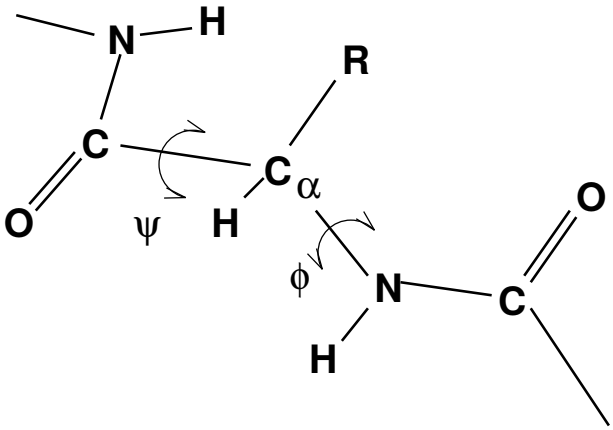


Fig. 5.61: Definition of the dihedral angles ϕ and ψ of the protein backbone.

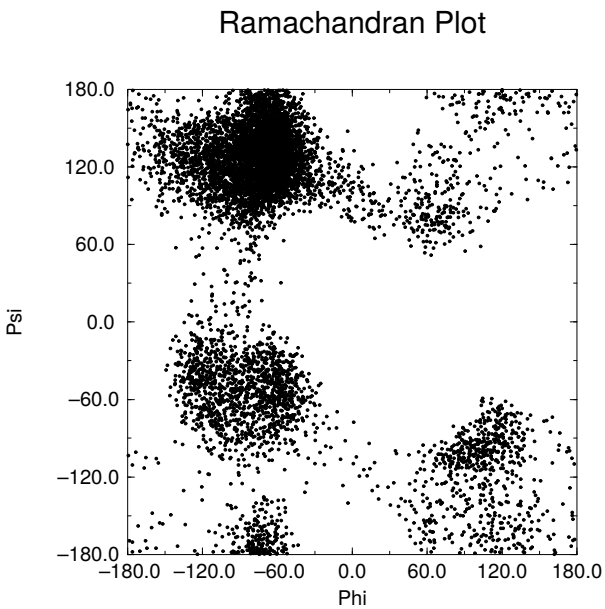


Fig. 5.62: Ramachandran plot of a small protein.

To evaluate this Ramachandran plot you can use the program *gmx rama* (page 207). A typical output is given in Fig. 5.62.

When studying α-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 252) program. Two examples are plotted in Fig. 5.63.

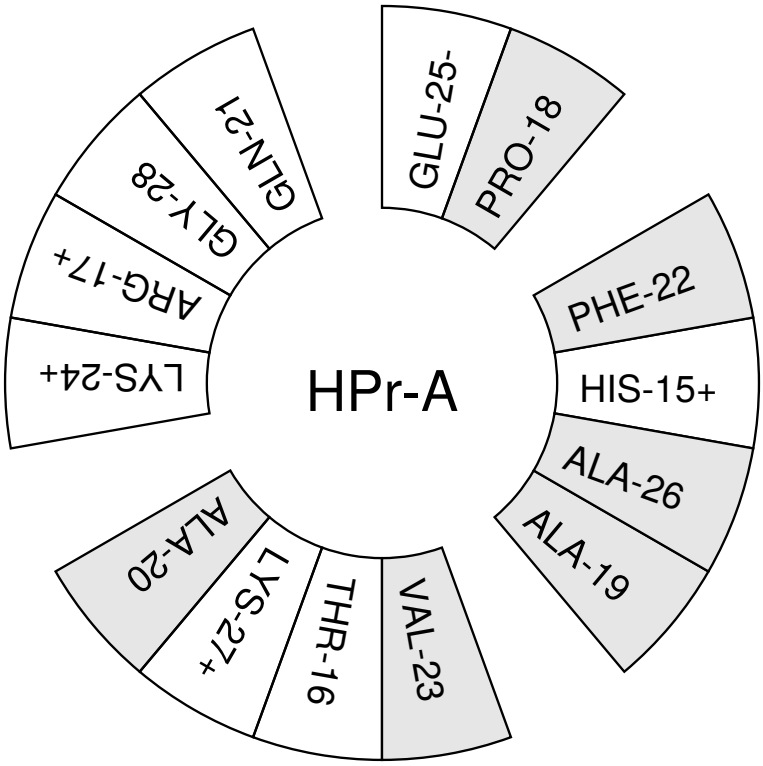


Fig. 5.63: Helical wheel projection of the N-terminal helix of HPr.

5.10.15 Interface-related items

gmx order (page 197), *gmx density* (page 135), *gmx potential* (page 205), *gmx traj* (page 233)

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 197) can calculate order parameters using the equation:

$$S_z = \frac{3}{2}\langle\cos^2\theta_z\rangle - \frac{1}{2} \tag{5.432}$$

where θ_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 (ψ) across the interface can be computed from a trajectory by evaluating

the double integral of the charge density ($\rho(z)$):

$$\psi(z) - \psi(-\infty) = -\int_{-\infty}^{z} dz' \int_{-\infty}^{z'} \rho(z'')dz''/\epsilon_0 \tag{5.433}$$

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 205) 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 233) 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 135) 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 Ξ can be written in full tensor form as:

$$\Xi = -\frac{1}{2} \sum_{i<j}^{N} \mathbf{r}_{ij} \otimes \mathbf{F}_{ij} \tag{5.434}$$

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 535).

Virial

In a system with periodic boundary conditions, the periodicity must be taken into account for the virial:

$$\Xi = -\frac{1}{2} \sum_{i<j}^{N} \mathbf{r}_{ij}^{n} \otimes \mathbf{F}_{ij} \tag{5.435}$$

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* δ_i to the position vector $\mathbf{r}_i$ of atom i. The difference vector $\mathbf{r}_{ij}^{n}$ is thus equal to:

$$\mathbf{r}_{ij}^{n} = \mathbf{r}_i + \delta_i - \mathbf{r}_j \tag{5.436}$$

or in shorthand:

$$\mathbf{r}_{ij}^{n} = \mathbf{r}_i^{n} - \mathbf{r}_j \tag{5.437}$$

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 δ_{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 $\xi_{\mathrm{alt}} = v_\xi = p_\xi/Q$ 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 δ_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

$$\begin{aligned} \mathbf{F}_i &= \sum_{j=1}^{N} \mathbf{F}_{ij} \\ \mathbf{F}_j &= \sum_{i=1}^{N} \mathbf{F}_{ji} \end{aligned} \tag{5.438}$$

which is the total force on i with respect to j. Because we use Newton's Third Law:

$$\mathbf{F}_{ij} = -\mathbf{F}_{ji} \tag{5.439}$$

we must, in the implementation, double the term containing the shift δ_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 Ξ into a component Ξ_0 that is a single sum over particles, and a second component Ξ_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:

$$\begin{aligned} b &= \|\mathbf{r}_{ij}^n\| \\ V_b &= \frac{1}{2} k_b (b - b_0)^2 \\ \mathbf{F}_i &= -\nabla V_b \\ &= k_b (b - b_0) \frac{\mathbf{r}_{ij}^n}{b} \\ \mathbf{F}_j &= -\mathbf{F}_i \end{aligned} \tag{5.440}$$

The virial contribution from the bonds then is:

$$\begin{aligned} \Xi_b &= -\frac{1}{2}(\mathbf{r}_i^n \otimes \mathbf{F}_i + \mathbf{r}_j \otimes \mathbf{F}_j) \\ &= -\frac{1}{2}\mathbf{r}_{ij}^n \otimes \mathbf{F}_i \end{aligned} \tag{5.441}$$

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:

$$\begin{aligned} \Delta\mathbf{r}_i &= \mathbf{r}_i(t+\Delta t) - [\mathbf{r}_i(t) + \mathbf{v}_i(t - \frac{\Delta t}{2})\Delta t + \frac{\mathbf{F}_i}{m_i}\Delta t^2] \\ \mathbf{G}_i &= \frac{m_i \Delta\mathbf{r}_i}{\Delta t^2 i} \end{aligned} \tag{5.442}$$

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:

$$\Xi = -\frac{1}{2}\sum_i^{N_c} \mathbf{r}_i \otimes \mathbf{F}_i \tag{5.443}$$

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 530), *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 356)) and Coulomb (sec. *Coulomb interaction* (page 358)) interactions.
4. Atoms two and three have only Coulomb interactions, and equal charges.

These loops also works for the SPC/E *178* (page 535) and TIP3P *128* (page 533) water models. And for four site water models similar to TIP4P *128* (page 533):

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 356)) interactions.
4. Atoms two and three have only Coulomb (sec. *Coulomb interaction* (page 358)) 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 535).

When analyzing a MD trajectory averages $\langle x \rangle$ and fluctuations

$$\left\langle (\Delta x)^2 \right\rangle^{\frac{1}{2}} \; = \; \left\langle [x - \langle x \rangle]^2 \right\rangle^{\frac{1}{2}} \tag{5.444}$$

of a quantity x are to be computed. The variance σ_x of a series of N$_x$ values, $\{x_i\}$, can be computed from

$$\sigma_x \; = \; \sum_{i=1}^{N_x} x_i^2 \; - \; \frac{1}{N_x} \left(\sum_{i=1}^{N_x} x_i \right)^2 \tag{5.445}$$

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

$$\sigma_x \; = \; \sum_{i=1}^{N_x} [x_i - \langle x \rangle]^2 \tag{5.446}$$

with

$$\langle x \rangle \; = \; \frac{1}{N_x} \sum_{i=1}^{N_x} x_i \tag{5.447}$$

Using (5.445) and (5.447) one has to go through the series of x_i values twice, once to determine $\langle x \rangle$ and again to compute σ_x, whereas (5.444) requires only one sequential scan of the series $\{x_i\}$. However, one may cast (5.445) in another form, containing partial sums, which allows for a sequential update algorithm. Define the partial sum

$$X_{n,m} \; = \; \sum_{i=n}^{m} x_i \tag{5.448}$$

and the partial variance

$$\sigma_{n,m} \; = \; \sum_{i=n}^{m} \left[x_i - \frac{X_{n,m}}{m-n+1} \right]^2 \tag{5.449}$$

It can be shown that

$$X_{n,m+k} \; = \; X_{n,m} + X_{m+1,m+k} \tag{5.450}$$

and

$$\begin{aligned} \sigma_{n,m+k} = \quad & \sigma_{n,m} + \sigma_{m+1,m+k} + \left[\frac{X_{n,m}}{m-n+1} - \frac{X_{n,m+k}}{m+k-n+1} \right]^2 * \\ & \frac{(m-n+1)(m+k-n+1)}{k} \end{aligned}$$

For $n = 1$ one finds

$$\sigma_{1,m+k} \; = \; \sigma_{1,m} + \sigma_{m+1,m+k} \; + \; \left[\frac{X_{1,m}}{m} - \frac{X_{1,m+k}}{m+k} \right]^2 \frac{m(m+k)}{k} \tag{5.451}$$

and for $n = 1$ and $k = 1$ (5.451) becomes

$$\begin{aligned} \sigma_{1,m+1} = \quad & \sigma_{1,m} + \left[\frac{X_{1,m}}{m} - \frac{X_{1,m+1}}{m+1} \right]^2 m(m+1) \\ = \quad & \sigma_{1,m} + \frac{[\, X_{1,m} - m x_{m+1} \,]^2}{m(m+1)} \end{aligned} \tag{5.452}$$

where we have used the relation

$$X_{1,m+1} \;=\; X_{1,m} + x_{m+1} \tag{5.453}$$

Using formulae (5.452) and (5.453) the average

$$\langle x \rangle \;=\; \frac{X_{1,N_x}}{N_x} \tag{5.454}$$

and the fluctuation

$$\left\langle (\Delta x)^2 \right\rangle^{\frac{1}{2}} = \left[\frac{\sigma_{1,N_x}}{N_x} \right]^{\frac{1}{2}} \tag{5.455}$$

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 436), 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 439) 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 (5.450) and (5.451) the average and standard deviation over part of the trajectory can be computed as:

$$\begin{aligned} X_{m+1,m+k} &= X_{1,m+k} - X_{1,m} \\ \sigma_{m+1,m+k} &= \sigma_{1,m+k} - \sigma_{1,m} - \left[\frac{X_{1,m}}{m} - \frac{X_{1,m+k}}{m+k} \right]^2 \frac{m(m+k)}{k} \end{aligned} \tag{5.456}$$

or, more generally (with $p \geq 1$ and $q \geq p$):

$$\begin{aligned} X_{p,q} &= X_{1,q} - X_{1,p-1} \\ \sigma_{p,q} &= \sigma_{1,q} - \sigma_{1,p-1} - \left[\frac{X_{1,p-1}}{p-1} - \frac{X_{1,q}}{q} \right]^2 \frac{(p-1)q}{q-p+1} \end{aligned} \tag{5.457}$$

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 (5.452) and (5.453):

$$\begin{aligned} X_{1,p-1} &= X_{1,p} - x_p \\ \sigma_{1,p-1} &= \sigma_{1,p} - \frac{[\, X_{1,p-1} - (p-1)x_p \,]^2}{(p-1)p} \end{aligned} \tag{5.458}$$

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:

$$X^{AB}_{1,n+m} \;=\; X^A_{1,n} + X^B_{1,m} \tag{5.459}$$

When we want to compute the partial variance from the two components we have to make a correction $\Delta\sigma$:

$$\sigma^{AB}_{1,n+m} \;=\; \sigma^A_{1,n} + \sigma^B_{1,m} + \Delta\sigma \tag{5.460}$$

if we define x_i^{AB} as the combined and renumbered set of data points we can write:

$$\sigma^{AB}_{1,n+m} \;=\; \sum_{i=1}^{n+m}\left[x_i^{AB} - \frac{X^{AB}_{1,n+m}}{n+m}\right]^2 \tag{5.461}$$

and thus

$$\sum_{i=1}^{n+m}\left[x_i^{AB} - \frac{X^{AB}_{1,n+m}}{n+m}\right]^2 \;=\; \sum_{i=1}^{n}\left[x_i^{A} - \frac{X^{A}_{1,n}}{n}\right]^2 + \sum_{i=1}^{m}\left[x_i^{B} - \frac{X^{B}_{1,m}}{m}\right]^2 + \Delta\sigma \tag{5.462}$$

or

$$\begin{aligned}
\sum_{i=1}^{n+m}\left[(x_i^{AB})^2 - 2x_i^{AB}\frac{X^{AB}_{1,n+m}}{n+m} + \left(\frac{X^{AB}_{1,n+m}}{n+m}\right)^2\right] - \\
\sum_{i=1}^{n}\left[(x_i^{A})^2 - 2x_i^{A}\frac{X^{A}_{1,n}}{n} + \left(\frac{X^{A}_{1,n}}{n}\right)^2\right] - \\
\sum_{i=1}^{m}\left[(x_i^{B})^2 - 2x_i^{B}\frac{X^{B}_{1,m}}{m} + \left(\frac{X^{B}_{1,m}}{m}\right)^2\right] = \quad \Delta\sigma
\end{aligned}$$

all the x_i^2 terms drop out, and the terms independent of the summation counter i can be simplified:

$$\begin{aligned}
\frac{\left(X^{AB}_{1,n+m}\right)^2}{n+m} - \frac{\left(X^{A}_{1,n}\right)^2}{n} - \frac{\left(X^{B}_{1,m}\right)^2}{m} - \\
2\,\frac{X^{AB}_{1,n+m}}{n+m}\sum_{i=1}^{n+m} x_i^{AB} \;+\; 2\,\frac{X^{A}_{1,n}}{n}\sum_{i=1}^{n} x_i^{A} \;+\; 2\,\frac{X^{B}_{1,m}}{m}\sum_{i=1}^{m} x_i^{B} = \quad \Delta\sigma
\end{aligned}$$

we recognize the three partial sums on the second line and use (5.459) to obtain:

$$\Delta\sigma \;=\; \frac{\left(mX^A_{1,n} - nX^B_{1,m}\right)^2}{nm(n+m)} \tag{5.463}$$

if we check this by inserting $m=1$ we get back (5.452)

Summing energy terms

The *gmx energy* (page 155) 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:

$$X^S_{m,n} \;=\; \sum_{i=m}^{n}\sum_{s=1}^{S} x_i^s \;=\; \sum_{s=1}^{S}\sum_{i=m}^{n} x_i^s \;=\; \sum_{s=1}^{S} X^s_{m,n} \tag{5.464}$$

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:

$$\sigma^S_{m,n} \;=\; \sum_{s=1}^{S} \sigma^s_{m,n} + \Delta\sigma \tag{5.465}$$

if we fill in (5.449):

$$\sum_{i=m}^{n} \left[\left(\sum_{s=1}^{S} x_i^s \right) - \frac{X^S_{m,n}}{m-n+1} \right]^2 \;=\; \sum_{s=1}^{S} \sum_{i=m}^{n} \left[(x_i^s) - \frac{X^s_{m,n}}{m-n+1} \right]^2 + \Delta\sigma \tag{5.466}$$

which we can expand to:

$$\begin{aligned} &\sum_{i=m}^{n} \left[\sum_{s=1}^{S} (x_i^s)^2 + \left(\frac{X^S_{m,n}}{m-n+1} \right)^2 - 2 \left(\frac{X^S_{m,n}}{m-n+1} \sum_{s=1}^{S} x_i^s + \sum_{s=1}^{S} \sum_{s'=s+1}^{S} x_i^s x_i^{s'} \right) \right] \\ - \quad & \sum_{s=1}^{S} \sum_{i=m}^{n} \left[(x_i^s)^2 - 2 \frac{X^s_{m,n}}{m-n+1} x_i^s + \left(\frac{X^s_{m,n}}{m-n+1} \right)^2 \right] \;=\; \Delta\sigma \end{aligned}$$

the terms with $(x_i^s)^2$ cancel, so that we can simplify to:

$$\begin{aligned} &\frac{\left(X^S_{m,n}\right)^2}{m-n+1} - 2 \frac{X^S_{m,n}}{m-n+1} \sum_{i=m}^{n} \sum_{s=1}^{S} x_i^s - 2 \sum_{i=m}^{n} \sum_{s=1}^{S} \sum_{s'=s+1}^{S} x_i^s x_i^{s'} - \\ &\sum_{s=1}^{S} \sum_{i=m}^{n} \left[-2 \frac{X^s_{m,n}}{m-n+1} x_i^s + \left(\frac{X^s_{m,n}}{m-n+1} \right)^2 \right] \;=\; \Delta\sigma \end{aligned}$$

or

$$-\frac{\left(X^S_{m,n}\right)^2}{m-n+1} - 2 \sum_{i=m}^{n} \sum_{s=1}^{S} \sum_{s'=s+1}^{S} x_i^s x_i^{s'} + \sum_{s=1}^{S} \frac{\left(X^s_{m,n}\right)^2}{m-n+1} \;=\; \Delta\sigma \tag{5.467}$$

If we now expand the first term using (5.464) we obtain:

$$-\frac{\left(\sum_{s=1}^{S} X^s_{m,n}\right)^2}{m-n+1} - 2 \sum_{i=m}^{n} \sum_{s=1}^{S} \sum_{s'=s+1}^{S} x_i^s x_i^{s'} + \sum_{s=1}^{S} \frac{\left(X^s_{m,n}\right)^2}{m-n+1} \;=\; \Delta\sigma \tag{5.468}$$

which we can reformulate to:

$$-2 \left[\sum_{s=1}^{S} \sum_{s'=s+1}^{S} X^s_{m,n} X^{s'}_{m,n} + \sum_{i=m}^{n} \sum_{s=1}^{S} \sum_{s'=s+1}^{S} x_i^s x_i^{s'} \right] \;=\; \Delta\sigma \tag{5.469}$$

or

$$-2 \left[\sum_{s=1}^{S} X^s_{m,n} \sum_{s'=s+1}^{S} X^{s'}_{m,n} + \sum_{s=1}^{S} \sum_{i=m}^{n} x_i^s \sum_{s'=s+1}^{S} x_i^{s'} \right] \;=\; \Delta\sigma \tag{5.470}$$

which gives

$$-2 \sum_{s=1}^{S} \left[X^s_{m,n} \sum_{s'=s+1}^{S} \sum_{i=m}^{n} x_i^{s'} + \sum_{i=m}^{n} x_i^s \sum_{s'=s+1}^{S} x_i^{s'} \right] \;=\; \Delta\sigma \tag{5.471}$$

Since we need all data points i to evaluate this, in general this is not possible. We can then make an estimate of $\sigma^S_{m,n}$ using only the data points that are available using the left hand side of (5.466). 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. Phys.*, **123** [11] (2005).

[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," *JCTC*, **9** [8] 3282–3292 (2013).

[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).

[101] T.T. Pham and M.R. Shirts, "Identifying low variance pathways for free energy calculations of molecular transformations in solution phase," *J. Chem. Phys.*, **135** 034114 (2011).

[102] T.T. Pham and M.R. Shirts, "Optimal pairwise and non-pairwise alchemical pathways for free energy calculations of molecular transformation in solution phase," *J. Chem. Phys.*, **136** 124120 (2012).

[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, "α-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 revelaed 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).

[182] 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).

[183] 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).

[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).

CHAPTER

SIX

GMXAPI PYTHON PACKAGE

This documentation is part of the GROMACS manual and describes the *gmxapi* Python package. `gmxapi` (page 552) allows molecular simulation and analysis work to be staged and run from Python.

From version 0.1, the latest official documentation is at http://manual.gromacs.org/current/gmxapi/. Other releases can also be found at GitHub.

6.1 Python User Guide

6.1.1 Full installation instructions

Installation instructions for the `gmxapi` (page 552) Python package, built on GROMACS.

Command line examples assume the bash shell.

Note: 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 17) configuration script. For the equivalent sort of environment switching with the `gmxapi` (page 552) 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 552) Python package always refers to the same GROMACS installation.

Contents

- *Overview* (page 537)
  - *Install GROMACS* (page 537)
  - *Set up a Python virtual environment* (page 538)
  - *Install the gmxapi Python package* (page 538)
- *Background* (page 538)
  - *GROMACS requirements* (page 538)
  - *Build system requirements* (page 539)
  - *Python environment requirements* (page 539)
  - *Documentation build requirements* (page 540)
  - *Testing requirements* (page 540)
  - *MPI requirements* (page 540)
- *Installing the Python package* (page 541)

- *Recommended installation* (page 541)
- *Install from source* (page 543)
- *Offline install* (page 543)
- *Building a source archive* (page 543)

- *Accessing gmxapi documentation* (page 544)
  - *Build with GROMACS* (page 544)
  - *Docker web server* (page 544)
- *Testing* (page 544)
- *Troubleshooting* (page 545)
  - *AttributeError: module 'enum' has no attribute 'IntFlag'* (page 545)
  - *Couldn't find the `gmxapi` support library?* (page 545)

Note: 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`**

Note: 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).

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 2020 or higher.

See also:

GROMACS installation

The following assumes GROMACS is installed to `/path/to/gromacs`

Set up a Python virtual environment

```
python3 -m venv $HOME/myvenv
. $HOME/myvenv/bin/activate
python -m ensurepip --default-pip
pip install --upgrade pip setuptools
pip install --upgrade cmake scikit-build
```

See also:

Set up a Python virtual environment (page 541)

Install the gmxapi Python package

```
. /path/to/gromacs/bin/GMXRC
pip install gmxapi
```

See also:

Installing the Python package (page 541)

Background

gmxapi comes in three parts:

- GROMACS gmxapi library for C++.
- This Python package, supporting Python 3.7 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.

Then, "source" the `GMXRC` file from the GROMACS installation *as you normally would* (page 17) before using GROMACS, or note its installation location so that you can pass it to the build configuration.

Build system requirements

gmxapi can be built for Python 3.7 and higher.

You will need a C++ 14 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 3.7 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, scikit-build, 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
```

"requirements" files in GROMACS source tree

If you are building from source code in a local copy of the GROMACS source repository, some helpful files allow you to preinstall the Python requirements before installing the *gmxapi* (page 552) package.

pip install -r python_packaging/src/requirements.txt

If building documentation or running tests, **pip install -r python_packaging/requirements-docs.txt** or **pip install -r python_packaging/requirements-test.txt**, respectively, or see below.

Documentation build requirements

See *Accessing gmxapi documentation* (page 544)

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/requirements-test.txt` lists additional requirements for testing. With pip:

```
pip install -r python_packaging/requirements-test.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
```

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 different 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.

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
```

(continues on next page)

(continued from previous page)

If you have a different MPI C compiler wrapper, substitute it for **mpicc** above.

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 set the CMake variable `GMXAPI_USER_INSTALL` (`-DGMXAPI_USER_INSTALL=ON` on the **cmake** command line) and / or 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 543).

Locate or install GROMACS

You need a GROMACS installation that includes the gmxapi headers and library. If GROMACS 2020 or higher is already installed, *and* was configured with `GMXAPI=ON` at build time, you can just source the *GMXRC* (page 17) (so that the Python package knows where to find GROMACS) and skip to the next section.

Otherwise, 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
```

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**.

Create a Python 3 virtual environment:

```
python3 -m venv $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 and setuptools before installing new Python packages:

```
pip install --upgrade pip setuptools
```

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 scikit-build
```

For MPI, we use mpi4py. Make sure it is using the same MPI installation that we are building GROMACS against and building with compatible compilers.

```
python -m pip install --upgrade pip setuptools
MPICC=`which mpicc` pip install --upgrade mpi4py
```

See also:

MPI requirements (page 540)

Install the latest version of gmxapi

Fetch and install the latest official version of gmxapi from the Python Packaging Index:

```
# Get the latest official release.
pip install gmxapi
```

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 --pre gmxapi
```

If **pip** does not find your GROMACS installation, use one of the following environment variables to provide a hint.

gmxapi_DIR

If you have a single GROMACS installation at `/path/to/gromacs`, it is usually sufficient to provide this location to **pip** through the `gmxapi_DIR` environment variable.

Example:

```
gmxapi_DIR=/path/to/gromacs pip install gmxapi
```

GMXTOOLCHAINDIR

If you have multiple builds of GROMACS distinguished by suffixes (e.g. *_d*, *_mpi*, etcetera), or if you need to provide extra hints to **pip** about the software tools that were used to build GROMACS, you can specify a directory in which the installer can find a CMake "tool chain".

In the following example, `${SUFFIX}` is the suffix that distinguishes the particular build of GROMACS you want to target (refer to GROMACS installation instructions for more information.) `${SUFFIX}` may simply be empty, or `''`.

```
GMXTOOLCHAINDIR=/path/to/gromacs/share/cmake/gromacs${SUFFIX} pip␣
↪install gmxapi
```

Install from source

You can also install the *gmxapi* (page 552) 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 552) Python package sources in the `python_packaging/src` directory.

```
cd python_packaging/src
pip install -r requirements.txt
pip install .
```

Offline install

If the required dependencies are already installed, you can do a quick installation without internet access, either from the source directory or from a source archive.

For example, the last line of the previous example could be replaced with:

```
pip install --no-cache-dir --no-deps --no-index --no-build-
↪isolation .
```

Refer to pip documentation for descriptions of these options.

If you have built or downloaded a source distribution archive, you can provide the archive file to **pip** instead of the `.` argument:

```
pip install gmxapi-0.1.0.tar.gz
```

In this example, the archive file name is as was downloaded from PyPI or as built locally, according to the following instructions.

Building a source archive

A source archive for the gmxapi python package can be built from the GROMACS source repository using Python `setuptools` and `scikit-build`.

Example:

```
pip install --upgrade setuptools scikit-build
cd python_packaging/src
python setup.py 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 `setup.py` file, and *<suffix>* is determined by the local environment or by additional arguments to `setup.py`.

See also:

Python documentation for creating a source distribution

Package maintainers may update the online respository by uploading a freshly built `sdist` with `python -m twine upload dist/*`

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.

Sometimes the build environment can choose a different Python interpreter than the one you intended. You can set the `Python3_ROOT` or `CMAKE_PREFIX_PATH` CMake variable to explicitly choose the Python installation or *venv* directory.

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=$(pyenv prefix $(pyenv version-name))
```

Docker web server

Alternatively, build the `docs` Docker image from `python_packaging/docker/docs.dockerfile` or pull a prebuilt image from DockerHub. Refer to the dockerfile or to https://hub.docker.com/r/gmxapi/docs for more information.

Testing

Note *testing requirements* (page 540) above.

After installing the *`gmxapi`* (page 552) 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/src/test/
```

Refer to `python_packaging/README.md` for more detailed information.

Troubleshooting

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.

Couldn't find the `gmxapi` support library?

If you don't want to "source" your *GMXRC* (page 17) file, you can tell the package where to find a gmxapi compatible GROMACS installation with `gmxapi_DIR`. E.g. `gmxapi_DIR=/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_DIR" 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_DIR` (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_DIR=/Users/eric/gromacs python setup.py install --verbose
```

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 552) 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 552) 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 552) 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 `master` 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 552) 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.

6.1.2 Using the Python package

After installing GROMACS, sourcing the "GMXRC" (see GROMACS docs), and installing the gmxapi Python package (see *Full installation instructions* (page 536)), 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 551).

Any Python *exception* raised by gmxapi should be descended from (and catchable as) *gmxapi.exceptions.Error* (page 556). Additional status messages can be acquired through the *Logging* (page 550) 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 541).

Notes on parallelism and MPI

When launching a *gmxapi* script in an MPI environment, such as with **mpiexec** or **srun**, you must help *gmxapi* detect the MPI environment by ensuring that `mpi4py` is loaded. Refer to *MPI requirements* (page 540) for more on installing `mpi4py`.

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
```

gmxapi 0.1 has limited parallelism, but future versions will include seamless acceleration as integration improves with the GROMACS library and computing environment runtime resources. Currently, *gmxapi* and the GROMACS library do not have an effective way to share an MPI environment. Therefore, if you intend to run more than one simulation at a time, in parallel, in a *gmxapi* script, you should build GROMACS with *thread-MPI* instead of a standard MPI library. I.e. configure GROMACS with the CMake flag `-DGMX_THREAD_MPI=ON`. Then, launch your *gmxapi* script with one MPI rank per node, and *gmxapi* will assign each (non-MPI) simulation to its own node, while keeping the full MPI environment available for use via `mpi4py`.

Running simple simulations

Once the `gmxapi` package is installed, running simulations is easy with *`gmxapi.read_tpr()`* (page 554).

```
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.

The *`gmxapi.mdrun()`* (page 554) 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.

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.

For *gmxapi* 0.1, we recommend configuring the GROMACS build with `GMX_THREAD_MPI=ON` and allowing one rank per node in order to allow each simulation ensemble member to run on a separate node.

See also:

Notes on parallelism and MPI (page 547)

Accessing command line tools

In *gmxapi* 0.1, most GROMACS tools are not yet exposed as *gmxapi* Python operations. `gmxapi.commandline_operation` (page 552) 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())
```

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 554).

Simulation input can be modified with the `gmxapi.modify_input()` (page 554) operation before being passed to `gmxapi.mdrun()` (page 554). 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()
```

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 553). 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**.

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 555) 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 555) 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.

Looping

An operation can be executed an arbitrary number of times with a *gmxapi.while_loop()* (page 555) 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 555) 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.
    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
```

Logging

gmxapi uses the Python `logging` module to provide hierarchical logging, organized by submodule. You can access the logger at `gmxapi.logger` or, after importing *gmxapi*, through the Python logging framework:

```
import gmxapi as gmx
import logging

# Get the root gmxapi logger.
gmx_logger = logging.getLogger('gmxapi')
```

(continues on next page)

(continued from previous page)

```
# 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.
ch = logging.StreamHandler()
ch.setFormatter(formatter)
logging.getLogger().addHandler(ch)
```

For more information, refer to the Python logging documentation.

More

Refer to the *gmxapi Python module reference* (page 551) 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)

6.1.3 gmxapi Python module reference

- *gmxapi basic package* (page 552)
- *Status messages and Logging* (page 556)
- *Exceptions module* (page 556)
- *gmx.version module* (page 557)
- *Core API* (page 558)
  - *gmxapi core module* (page 558)
  - *Exceptions* (page 558)
  - *Functions* (page 559)

– *Classes* (page 559)

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 diffently, 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 544)

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.

class `gmxapi.`**`NDArray`**(*data=None*)

N-Dimensional array type.

`gmxapi.`**`commandline_operation`**(*executable=None*, *arguments=()*, *input_files: Optional[dict] = None*, *output_files: Optional[dict] = None*, *stdin: Optional[str] = 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.

Parameters

- **`executable`** – name of an executable on the path
- **`arguments`** – list of positional arguments to insert at `argv[1]`
- **`input_files`** – mapping of command-line flags to input file names
- **`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).

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).

Output: The output node of the resulting operation handle contains

- `file`: the mapping of CLI flags to filename strings resulting from the `output_files` kwarg
- `stdout`: A string mapping from process STDOUT.
- **`stderr`: A string mapping from process STDERR; it will be the** error output (if any) if the process failed.
- `returncode`: return code of the subprocess.

`gmxapi.`**`concatenate_lists`**(*sublists: list = ()*) → gmxapi.typing.Future[*gmxapi.datamodel.NDArray* (page 552)]

Combine data sources into a single list.

A trivial data flow restructuring operation.

`gmxapi.`**`function_wrapper`**(*output: Optional[dict] = None*)

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.`**`join_arrays`**(**, front:* gmxapi.datamodel.NDArray *(page 552) = (), back:* gmxapi.datamodel.NDArray *(page 552) = ()*) → *gmxapi.datamodel.NDArray* (page 552)

Operation that 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. # TODO: elaborate and clarify. # TODO: check type and shape. # TODO: figure out a better annotation.

`gmxapi.`**`logical_not`**(*value: bool*) → gmxapi.typing.Future

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 returns a Future for the logical opposite of the callable's result.

`gmxapi.make_constant`(*value: gmxapi.operation.Scalar*) → gmxapi.typing.Future

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.

`gmxapi.mdrun`(*input*, *label: Optional[str] = None*, *context=None*)

MD simulation operation.

Parameters **input** – valid simulation input

Returns runnable operation to perform the specified simulation

The *output* attribute of the returned operation handle contains dynamically determined outputs from the operation.

`input` may be a TPR file name or a an object providing the SimulationInput interface.

Note: New function names will be appearing to handle tasks that are separate

"simulate" is plausibly a dispatcher or base class for various tasks dispatched by mdrun. Specific work factories are likely "minimize," "test_particle_insertion," "legacy_simulation" (do_md), or "simulation" composition (which may be leap-frog, vv, and other algorithms)

`gmxapi.modify_input`(*input*, *parameters: dict*, *label: Optional[str] = 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*.

`gmxapi.ndarray`(*data=None*, *shape=None*, *dtype=None*)

Create an NDArray object from the provided iterable.

Parameters **data** – object supporting sequence, buffer, or Array Interface protocol

New in version 0.1: *shape* and *dtype* parameters

If `data` is provided, `shape` and `dtype` are optional. If `data` is not provided, both `shape` and `dtype` are required.

If `data` is provided and shape is provided, `data` must be compatible with or convertible to `shape`. See Broadcast Rules in `Data model` documentation.

If `data` is provided and `dtype` is not provided, data type is inferred as the narrowest scalar type necessary to hold any element in `data`. `dtype`, whether inferred or explicit, must be compatible with all elements of `data`.

The returned object implements the gmxapi N-dimensional Array Interface.

`gmxapi.read_tpr`(*filename*, *label: Optional[str] = None*, *context=None*)

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).

`gmxapi.subgraph(variables=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 0.1 API is finalized.

When called, `while_loop` calls `operation` without arguments and captures the return value captured as `_operation`. The object produced by `operation()` must have a `reset`, a `run` method, and an `output` attribute.

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.

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, attach a logging.StreamHandler to the 'gmxapi' logger.

Example:

```
ch = logging.StreamHandler()
# Optional: Set log level.
ch.setLevel(logging.DEBUG)
# Optional: create formatter and add to character stream handler
formatter = logging.Formatter('%(asctime)s - %(name)s -
↪%(levelname)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 552) and its submodules, attach the handler before importing *gmxapi* (page 552)

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.

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 556) 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.**NotImplementedError**
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.

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.

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 557) module functions *api_is_at_least()* (page 557) and *has_feature()* (page 557) 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:

```
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=''*, *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 557), 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 556) 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 556) – If `enable_exception == True` and feature is not found.

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

exception gmxapi._gmxapi.**Exception**

Root exception for the C++ extension module. Derives from `gmxapi.exceptions.Error` (page 556).

exception gmxapi._gmxapi.**NotImplementedError**

Expected feature is not implemented.

exception gmxapi._gmxapi.**ProtocolError**

Behavioral protocol violated.

exception gmxapi._gmxapi.**UnknownException**

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.)

exception gmxapi._gmxapi.**UsageError**

Unacceptable API usage.

Functions

Tools for launching simulations

gmxapi._gmxapi.**from_tpr**(*arg0: str*) → *gmxapi._gmxapi.MDSystem* (page 559)

Return a system container initialized from the given input record.

Tools to manipulate TPR input files

gmxapi._gmxapi.**copy_tprfile**(*source:* gmxapi._gmxapi.TprFile *(page 560)*, *destination: str*) → bool

Copy a TPR file from `source` to `destination`.

gmxapi._gmxapi.**read_tprfile**(*filename: str*) → *gmxapi._gmxapi.TprFile* (page 560)

Get a handle to a TPR file resource for a given file name.

gmxapi._gmxapi.**write_tprfile**(*filename: str*, *parameters:* gmxapi._gmxapi.SimulationParameters *(page 559)*) → 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 37) with `end_time`.

Classes

class gmxapi._gmxapi.**Context**

add_mdmodule(*self:* gmxapi._gmxapi.Context *(page 559)*, *arg0: object*) → None

Add an MD plugin for the simulation.

setMDArgs(*self:* gmxapi._gmxapi.Context *(page 559)*, *arg0:* gmxapi._gmxapi.MDArgs *(page 559)*) → None

Set MD runtime parameters.

class gmxapi._gmxapi.**MDArgs**

set(*self:* gmxapi._gmxapi.MDArgs *(page 559)*, *arg0: dict*) → None

Assign parameters in MDArgs from Python dict.

class gmxapi._gmxapi.**MDSession**

close(*self:* gmxapi._gmxapi.MDSession *(page 559)*) → gmxapi._gmxapi.Status

Shut down the execution environment and close the session.

run(*self:* gmxapi._gmxapi.MDSession *(page 559)*) → gmxapi._gmxapi.Status

Run the simulation workflow

class gmxapi._gmxapi.**MDSystem**

launch(*self:* gmxapi._gmxapi.MDSystem *(page 559)*, *arg0:* gmxapi._gmxapi.Context *(page 559)*) → *gmxapi._gmxapi.MDSession* (page 559)

Launch the configured workflow in the provided context.

class gmxapi._gmxapi.**SimulationParameters**

extract(*self:* gmxapi._gmxapi.SimulationParameters *(page 559)*) → 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 560)*) → *gmxapi._gmxapi.SimulationParameters* (page 559)

After installing GROMACS and the gmxapi Python package, use `pydoc gmxapi` from the command line or `import gmxapi; help(gmxapi)` within Python for package and module documentation.

See also:

gmxapi was first described by

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

6.2 Indices and tables

- genindex
- search

CHAPTER

SEVEN

NBLIB API

This documentation is part of the GROMACS manual and describes the *nblib* 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 },
};
```

(continues on next page)

(continued from previous page)

```
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");
```

(continues on next page)

(continued from previous page)

```
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`)
`ljCombinationRule`	Enum	Lennard-Jones combination rule (`Geometric`/`LorentzBerthelot`)
`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. Wiki pages at http://www.gromacs.org/Developer_Zone may contain additional information (much of it outdated, though), and can be linked from relevant locations in the developer guide.

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 this page first, and at least post on the developer mailing list. Sometimes we'll 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 mailing list mailing list, 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 638).

  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 621), 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've 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'd like to contribute, so we can't 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 35) you might require and write a *tpr* (page 446),
  - the functionality for *mdrun* (page 183) 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 mailing list, 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's a period when we cannot accept certain kinds of potentially risky changes. (The master 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 591) and *code formatting* (page 591) 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 631).
- *Code documentation*: To ensure proper code documentation, please follow the instructions provided for the use of *doxygen* (page 605). 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 601). This will also facilitate the code review process.

8.1.2 Preparing code for submission

GROMACS uses `git` for *Change Management* (page 581). Instead of accepting "pull requests", GROMACS changes are submitted as individual commits on the tip of the `master` branch hosted at gitlab. Preparing, submitting, and managing patches for a change requires a little bit of set-up. Refer to *Change Management* (page 581) 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 mailing list, so you'll 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 mailing list. 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 290). For users, there are also lists of *anticipated changes* (page **??**) and *deprecated functionality* (page **??**) as of GROMACS 2019.

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.

`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 574) 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 573). 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.

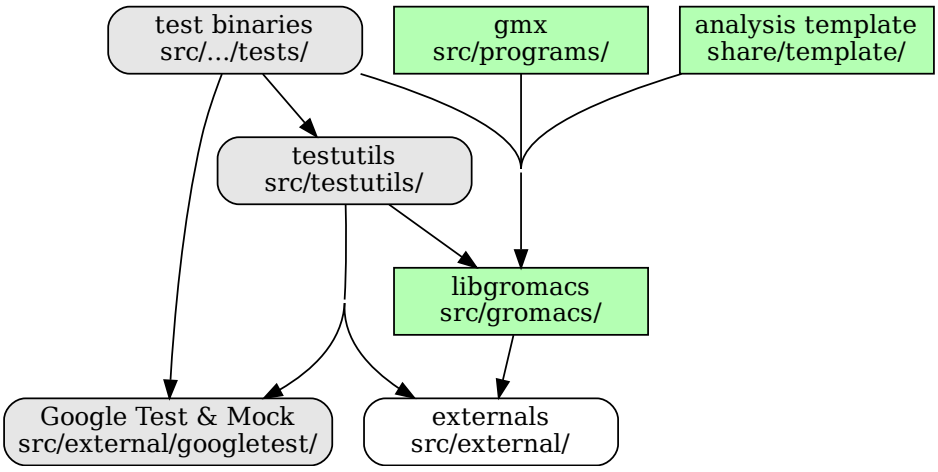
`share/` Contains data files that will be installed under `share/`. These include a template for writing C++ analysis tools, and data files used by GROMACS.

`src/` Contains all source code. See *Source code organization* (page 571).

`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 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 574). 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 636) for details of the testing side.



All the source code (except for the analysis template) is under the `src/` directory. 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`, the whole of the `gmx view` visualization module, 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 636) for details.

`src/testutils/` Contains shared utility code for writing Google Test tests. See *Unit testing* (page 636) 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 574) 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/`. 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, and *public API convenience headers*. These convenience headers provide the main installed headers that other code can use. They do not contain any declarations, but only include a suitable set of headers from the subdirectories. They typically also contain high-level Doxygen documentation for the subdirectory with the same name: `module.h` corresponds to `module/`.

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.

Installed headers are a subset of the headers under `src/gromacs/`. They are installed into a corresponding hierarchy under `include/gromacs/` in the installation directory. 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. 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. Not 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. You can use the public API header if you really require everything declared in it.

See *Naming conventions* (page 594) 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 636) 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 574) 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 568).

Doxygen documentation

`docs/doxygen/` Contains the build rules and some overview content for the Doxygen documentation. See *Using Doxygen* (page 605) 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 where 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 techical 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 3.16.3 or newer is required) to generate the actual build system for the build tool choosen 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 5 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 183), but can be useful for the tools.

TSAN Builds GROMACS for use with ThreadSanitzer in gcc and clang (http://clang.llvm.org/docs/ThreadSanitizer.html) to detect data races. This disables the use of atomics in ThreadMPI, preferring the mutex-based implementation.

ASAN Builds GROMACS for use with AddressSanitzer in gcc and clang (http://clang.llvm.org/docs/AddressSanitizer.html) to detect many kinds of memory mis-use. By default, AddressSanitizer includes LeakSanitizer.

MSAN Builds GROMACS for use with AddressSanitzer in clang (http://clang.llvm.org/docs/MemorySanitizer.html) to detect reads of unitialized 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.

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`) include 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 576). 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

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

GMX_BUILD_SHARED_EXE

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 76) 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 587) 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 reference manual for further information.

GMX_EXTRAE

GMX_EXTERNAL_BLAS

GMX_EXTERNAL_LAPACK

GMX_EXTERNAL_TNG

GMX_FFT_LIBRARY

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

GMX_CLANG_CUDA
Use clang for compiling CUDA GPU code, both host and device.

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 587) for how this influences the build.

GMX_LOAD_PLUGINS

GMX_MPI

GMX_OPENMP

GMX_PREFER_STATIC_LIBS

GMX_SIMD

GMX_SOFTWARE_INVSQRT

GMX_THREAD_MPI

GMX_USE_RDTSCP

GMX_USE_TNG

GMX_VMD_PLUGIN_PATH

GMX_X11

GMX_XML
Currently, this option has no effect on the compilation or linking, since there is no code outside the tests that would use `libxml2`.

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 579) is `ON`). Also, *GMX_COMPILER_WARNINGS* (page 576) is 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 9.0.* with libstdc++<7 or libc++ 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 issue 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 576) does not work with clang-tidy.

Variables affecting special targets

GMXAPI
If set ON, 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 targer, or also as part of the all target, depending on *GMX_DEVELOPER_BUILD* (page 578)). 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 578) 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 579) 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 636).

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 628).

completion Runs the compiled `gmx` executable to generate shell command-line completion definitions. This target is only added if *GMX_BUILD_HELP* (page 578) is not `OFF`, and it is run automatically as part of the default `all` target. See *GMX_BUILD_HELP* (page 578). 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 628).

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 605).

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 578) 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 579).

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 636).

tests Builds all the binaries needed by the tests (but not `gmx`). See *Unit testing* (page 636).

webpage Collection target that runs the other documentation targets to generate the full set of HTML (and linked) documentaion. 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 familiary with using `git` for managing file revisions.

- *Getting started* (page 581)
  - *Setting up login credentials with gitlab* (page 582)
  - *Creating issues* (page 582)
  - *Uploading code for review - creating a merge request* (page 582)
  - *Naming branches* (page 582)
- *Labels* (page 583)
- *Code Review* (page 583)
  - *Reviewing someone else's uploaded code* (page 583)
  - *Guide for reviewing* (page 583)
    - *Update the Status label* (page 584)
  - *Closing Merge Requests* (page 584)
- *More git tips* (page 584)

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 exisiting 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. We follow the workflow described in https://docs.gitlab.com/ee/topics/gitlab_flow.html.

If you do not already have a GROMACS repository set up, user `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 master 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 631) for help meeting and testing the style guidelines.

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, got to gitlab.com, find you user in the right top corner and select settings.

Chose SSH keys in the menu on the left and past 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 if your issue or a similar issue already exists before creating a new one.

Note that all Redmine issues have been transferred to gitlab with the same issue numbers as used in gitlab. 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. The default selection is to mark this a work in progress (WIP) merge-request. We recommend keeping this setting 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 master branch into your branch with `git merge master` while on your branch.

Naming branches

Good names: documentation_UpdateDevelopersDocsTOGitLab, nbnxm_MakeNbnxmGPUIntoClass, pme_FEPPMEGPU. Bad names: branch1234, mybranch, test, etc

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 GROMACS 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 584)

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.
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 change hat 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.

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 *GROMACS coding standards* (page 591);
- 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's 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's 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 601) when a merge request is under review.
- Please update the Status label *for the merge request* (page 584) 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 mailing list, 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 601) 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 master.

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 master, and master has gotten new commits. You can then do

```
git rebase master
```

to move your commits on top of the newest commit in master. This will save each commit you did, and replay them on top of master. 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 master 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 master. Further, assume that master 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 master
```

The first command allows you to edit your local branch without getting conflicts from changes in master. 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 master
```

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 200) and *gmx grompp* (page 166), 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 579).

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 578).

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.

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 605).

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 628).

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 628).

Sphinx Sphinx; at least version 1.6.1 is used for building some parts of the documentation from reStructuredText source files.

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

The following list provides the general formatting/indentation rules for GROMACS code (C/C++):

- 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 or uncrustify, which are both automatic source code formatting tool. The copyright guidelines are enforced by a separate Python script. See *Automatic source code formatting* (page 631) 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.

A number of user provided set-ups are available for the correct settings of your favourite text editor. They are provided for convenience only, and may not exactly conform to the expectations of either formatting tool.

Emacs formatting set-up

Insert the following into your .emacs configuration file:

```
(defun gromacs-c-mode-common-hook ()
;; GROMACS customizations for c-mode

(c-set-offset 'substatement-open 0)
(c-set-offset 'innamespace 0)
;; other customizations can go here

(setq c++-tab-always-indent t)
(setq c-basic-offset 4)                  ;; Default is 2
(setq c-indent-level 4)                  ;; Default is 2
(setq c-file-style "stroustrup")
(setq tab-stop-list '(4 8 12 16 20 24 28 32 36 40 44 48 52 56 60))
(setq tab-width 4)
(setq indent-tabs-mode nil)  ; use tabs if t
)
(add-hook 'c-mode-common-hook 'gromacs-c-mode-common-hook)

(defun gromacs-c++-mode-common-hook ()
;; GROMACS customizations for c++-moe

(c++-set-offset 'substatement-open 0)
(c++-set-offset 'innamespace 0)
;; other customizations can go here

(setq c++-tab-always-indent t)
(setq c++-basic-offset 4)                  ;; Default is 2
(setq c++-indent-level 4)                  ;; Default is 2
(setq c++-file-style "stroustrup")

(setq tab-stop-list '(4 8 12 16 20 24 28 32 36 40 44 48 52 56 60))
(setq tab-width 4)
```

(continues on next page)

(continued from previous page)

```
(setq indent-tabs-mode nil)  ; use tabs if t
)

(add-hook 'c++-mode-common-hook 'gromacs-c++-mode-common-hook)
```

This configuration is based on content from stackoverflow.

Eclipse/cdt formatting set-up

For correct formatting, please use this profile.

8.7.2 Guidelines for #include directives

The following include order is used in GROMACS. 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.

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 628) 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 outside of OpenCL kernels (which compile as C), and for consistency also in CUDA kernels.

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 `libgromacs` should be in a `gmx` namespace. 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.
- 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 `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.

Implementing exceptions for error handling

See *Error handling* (page 602) 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 assocation 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.

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 maintance, 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 neccessary 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 don't submit generic issues resulting from system instabilities and systems *Blowing up* (page 280).

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.

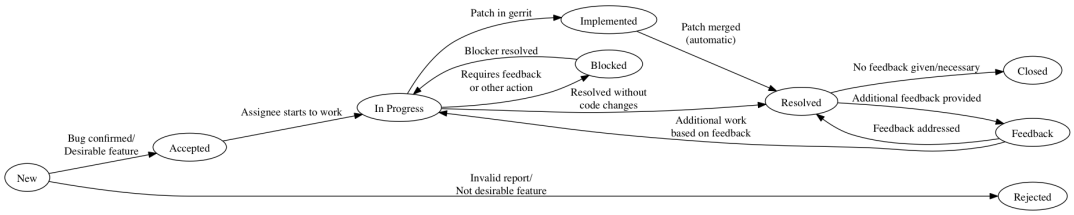
Please refrain from setting things such as target version or deciding on unreasonable priorities. If you decide to fix the issue on your own, please adhere to the other standards mentioned on the related pages *Guidelines for code formatting* (page 591) and *Guidelines for formatting of git commits* (page 601).

See also:

Contribute to GROMACS (page 568)

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 591) 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 can't 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 599).

***Guidelines for code formatting* (page 591)** Guidelines for indentation and other code formatting.

***Guidelines for #include directives* (page 593)** Guidelines for #include style (ordering, paths to use, etc.).

***Naming conventions* (page 594)** Naming conventions for files and various code constructs.

***Allowed language features* (page 597)** Allowed language features.

***Error handling* (page 602)** How to handle errors at run time

***General guidelines for Doxygen markup* (page 606)** Guidelines for using Doxygen to document the source code are currently in a section on the page on general Doxygen usage.

Guidelines for creating meaningful issue reports **(page 600)** Guidelines for preparing and formatting bug reports.

Guidelines for formatting of git commits **(page 601)** Guidelines for formatting git commits when sending in proposed fixes for code review.

8.8 Development-time tools

Several tools have their own individual pages and are listed below.

8.8.1 Using Doxygen

This page 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 573) 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 master 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 628) 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 571), 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 573). 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
{
    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.
```

(continues on next page)

(continued from previous page)

```
         * \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 */
```

(continues on next page)

(continued from previous page)

```
/*! \{ */

/*! \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.
 */
int gmx_function(int value);

/* Any . in the brief description 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
```

(continues on next page)

(continued from previous page)

```
 * 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
 * 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.
 */
```

(continues on next page)

(continued from previous page)

```
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
 * 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
```

(continues on next page)

(continued from previous page)

```
 * 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.
         */
        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
```

(continues on next page)

(continued from previous page)

```
// 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.

Understanding Jenkins builds

This page documents what different Jenkins builds actually run at http://jenkins.gromacs.org/ from the GROMACS source tree. The purpose is two-fold:

- Provide information on how to interpret Jenkins failures and how to run the same tasks locally to diagnose issues (in most cases, referring to the special targets described in *Build system overview* (page 574)).
- Provide information on what changes in the build system (or other parts of the repository) need special care to not break Jenkins builds.

Pre-submit verification

The following builds are triggered for each patch set uploaded to Gerrit.

Compilation and tests

The main build compiles GROMACS with different configurations and runs the tests. The configurations used for Jenkins verification are specified in `admin/builds/pre-submit-matrix.txt`.

The exact build sequence can be found in `admin/builds/gromacs.py`, including the logic that translates the build options in the matrix file to CMake options.

Documentation

This build builds various types of documentation:

- PDF reference manual using LaTeX
- Doxygen documentation extracted from the source code
- Set of HTML pages containing an installation guide, a user guide, and a developer guide, as well as links to the above. This set of HTML pages can be browsed from Jenkins.
- Man pages
- INSTALL text file

The last three require building the `gmx` binary and running it, so compilation failures will also show in this build. All log files that contain warnings are archived as artifacts in the build, and presence of any warnings marks the build unstable. Brief description of which part failed is reported back to Gerrit.

Additionally, the build runs some source code checks that rely on the Doxygen documentation. See the description of the `check-source` target in *Source tree checker scripts* (page 628).

Using Doxygen (page 605) provides general guidelines for Doxygen usage, which can be helpful in understanding and solving Doxygen warnings and some of the `check-source` issues. *Guidelines for #include directives* (page 593) provides guidelines for #include order and style, which is another part of `check-source` checks.

The exact build sequence is in `admin/builds/documentation.py`. See that file for details of what it exactly builds and how. Most changes in the documentation build system will require changes in this script, but Jenkins configuration should be more static.

clang static analysis

The file `admin/builds/clang-analyzer.py` specifies the exact build sequence and the CMake cache variables used for clang static analysis. This file also specifies the clang version used for the analysis, as well as the C++ compiler used (`clang-static-analyzer-<version>`).

To run the analysis outside Jenkins, you should run both `cmake` and `make` under `scan-build` command using the same CMake cache variables as in the build script. When you do the initial CMake configuration with `scan-build`, it sets the C++ compiler to the analyzer. Note that using `scan-build` like this will also analyze C code, but Jenkins ignores C code for analysis. This can result in extra warnings, which can be suppressed by manually setting CMAKE_C_COMPILER to a value other than Clang static analyzer.

uncrustify

This build checks for source code formatting issues with uncrustify, and enforces the copyright style. See *Guidelines for code formatting* (page 591) for the guidelines that are enforced.

The exact build sequence is in `admin/builds/uncrustify.py`, which essentially just runs

```
admin/uncrustify.sh check --rev=HEAD^
```

If the any changes are required, the build is marked unstable. If the script completely fails (should be rare), the build fails. A file with issues found by the script is archived as an artifact in the build, and a summary is reported back to Gerrit (or the actual issues if there are only a few). See *Automatic source code formatting* (page 631) for more details on code-formatting tools and on scripts to run them.

clang-format

This build checks and enforces code formatting, e.g., indentation. Also, a second part of the build enforces the source code formatting. As above, see *Guidelines for code formatting* (page 591) for the style guidelines.

The build runs according to `admin/builds/clang-format.py`, resulting in running

```
admin/clang-format.sh check --rev=HEAD^
```

The build is marked unstable if the code formatting resulted in any changes to the source code.

On-demand builds

These builds can be triggered on request for certain changes in Gerrit, or manually from Jenkins. See *Triggering builds on GitLab* (page 628) for details on how to trigger these.

Coverage

This build compiles one configuration of GROMACS with instrumentation for coverage, runs the tests, and produces a coverage report using gcovr. The report can be browsed on Jenkins.

The exact build sequence is in `admin/builds/coverage.py`, including specification of the configuration tested.

Source tarball

This build creates the source tarball for distribution. Some of the content that is put into the tarball is generated by executing the **gmx** binary, so this build also compiles the source code (with a minimal set of options).

The build compiles the code and those targets that generate content necessary for the tarball, followed by building the `package_source` target. After that, it just generates a file that is used by other builds.

The exact build sequence is in `admin/builds/source-package.py`.

Release workflow

This build creates source and regressiontest tarballs, builds, installs, and tests a few configuration using those, and builds documentation to be placed on the documentation web site for a new release. The set of configurations tested is specified in `admin/builds/release-matrix.txt`.

The exact build sequence is desribed in *Release engineering with GitLab* (page 628). The build uses the source tarball build as a subbuild, and parts of the build are executed using `admin/builds/gromacs.py` and `admin/builds/documentation.py`.

`admin/builds/get-version-info.py` is used for getting the version information from the source tree as part of this workflow.

`admin/builds/update-regtest-hash.py` has logic to update the regressiontests tarball MD5 sum for the released tarball automatically.

Updating regressiontests data

Sometimes we add new tests to the regressiontests repository. Also, as the source code or data files change, it is sometimes necessary to update regressiontests. This requires a particular CMake build type and both a single and double-precision build of GROMACS to generate all the data. Jenkins can automate much of the tedium here.

- Upload a regressiontests change that lacks the relevant reference data (either because you deleted the outdated data, or because the test is new). Jenkins will do the normal thing, which we ignore. There is now a Gerrit patch number for that change, symbolized here with `MMMM`.
- Go to change `MMMM` on gerrit, select the patch set you want to update with new reference data (usually the latest one), and comment

  ```
  [JENKINS] Update
  ```

  to update against the HEAD of the matching source-code branch, or

  ```
  [JENKINS] Cross-verify NNNN update
  ```

  to update from builds of GROMACS from the latest version of Gerrit source-code patch `NNNN`. You will need to do this when functionality changes in `NNNN` affect either the layout of the files in the reference data, or the results of the simulation, or the results of the subsequent analysis.
- Eventually, Jenkins will upload a new version of the regressiontests patch to Gerrit, which will contain the updated regressiontest data. That upload will again trigger Jenkins to do the normal pre-submit verify, which will now pass (but perhaps will only pass under cross-verify with patch `NNNN`, as above).
- Later, if you later need to verify an updated version of source-code patch `NNNN` against the newly generated reference data, go to the source-code patch `NNNN` and comment

  ```
  [JENKINS] Cross-verify MMMM
  ```

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 on Docker Hub. Images are (re)built manually using details in `admin/containers`.

This documentation is incomplete, pending resolution of Issue 3275.

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 625) 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.) Instead of setting any of these directly in a job definition, try to use one of the pre-defined behaviors (defined as `.rules:<something>` in `admin/gitlab-ci/rules.gitlab-ci.yml`). 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. To reduce errors and unexpected behavior, restrict usage of these controls to regular job definitions (don't 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 Jobs that can only run in the GROMACS GitLab CI Runner infrastructure should require the `k8s-scilifelab` tag. These include jobs that specify Kubernetes configuration variables or require special facilities, such as GPUs or MPI. Note that the *tag* controls which Runners are eligible to take a job. It does not affect whether the job is eligible for addition to a particular pipeline. Additional *rules* logic should be used to make sure that jobs with the `k8s-scilifelab` do not become eligible for pipelines launched outside of the GROMACS project environment. See, for instance, *CI_PROJECT_NAMESPACE*

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 624) 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 `master` and *release* branches in the GROMACS repository.

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 `master` 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.

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 master, 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 REGRESSIONTEST-COMMIT 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 `.variables:default` definition in `admin/gitlab-ci/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 behvior 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.

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 625).

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.

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-master.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`.

EXTRA_INSTALLS List additional OS package requirements. Used in *before_script* for some mix-in job definitions to install additional software dependencies. If using such a job with *extends*, override this variable key with a space-delimited list of packages (default: `""`). Consider proposing a patch to the base Docker images to include the dependency to reduce pipeline execution time.

REGRESSIONTESTBRANCH Use this branch of the regressiontests rather than master 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 master 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`.

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.

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 `master` branch for the set of images currently built.

Within *pipeline jobs* (page 621), jobs specify a Docker image with the *image* property. For image naming convention, see *`utility.image_name()`* (page 626). 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

Utilities

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-20.04-llvm-7-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: argparse.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`

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`

8.8.3 Release engineering with GitLab

We are currently switching our build and testing system to use GitLab and the integrated CI system, with information for the general system found in the official GitLab documentation. The new configuration for the builds and tests can be found in the file `.gitlab-ci.yml`, with the templates for configuring is found in the files in the `admin/ci-templates/` directory. This section is going to be extended with individual build information as it comes available.

See also:

Automation and Infrastructure (page 618)

Triggering builds on GitLab

Pipelines can be triggered through the web interface, with different pipelines available through the use of specified environment variables in the trigger interface.

This section is going to be extended with information for how to trigger different builds and their individual behaviour.

8.8.4 Source tree checker scripts

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
- Checking for consistent style and order of #include directives (see *Guidelines for #include directives* (page 593))
- Actually sorting and reformatting #include directives to adhere to the checked style
- 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 605), 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 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.
  - If the file has a git attribute to identify it as a candidate for include sorting, the include sorter described below should not produce any changes (i.e., the file should follow *Guidelines for #include directives* (page 593)).
- 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 order sorting

The script checks include ordering according to *Guidelines for #include directives* (page 593). If it is not obvious how the includes should be changed to make the script happy, or bulk changes are needed in multiple files, e.g., because of a header rename or making a previously public header private, it is possible to run a Python script that does the sorting:

```
docs/doxygen/includesorter.py -S . -B ../build <files>
```

The script needs to know the location of the source tree (given with `-S`) and the build tree (given with `-B`), and sorts the given files. To sort the whole source tree, one can also use:

```
admin/reformat_all.sh includesort -B=../build
```

For the sorter to work correctly, the build tree should contain up-to-date list of installed files and Doxygen XML documentation. The former is created automatically when `cmake` is run, and the latter can be built using the `doxygen-xml` target.

Note that currently, the sorter script does not change between angle brackets and quotes in include statements.

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.

8.8.5 Automatic source code formatting

The source code can be automatically formatted using clang-format since GROMACS 2020. Both are formatting tools that apply the guidelines in *Guidelines for code formatting* (page 591). Additionally, other Python scripts are used for a few other automatic formatting/checking tasks. The overview tools page contains a list of these tools: *Code formatting and style* (page 642). 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 7.0.1. **`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 version 7.0.1 or 7.1.0. Example:

```
$ clang-format --version
clang-format version 7.1.0 (tags/RELEASE_710/final)
```

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/7.0.1/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 9. **`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 9. Example:

```
$ clang-tidy --version
  LLVM (http://llvm.org/):
    LLVM version 9.0.0
```

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 IN-

STALL.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 http://releases.llvm.org/9.0.0/tools/clang/tools/extra/docs/clang-tidy/index.html

Scripts

`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.

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.

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 uncrustify or 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).

8.8.6 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 571), `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 a separate unit test binary (*`module`*`-test`) under `bin/`, which can execute all the 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.

- 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. It is possible to control which tests are run using command line options. Execute the binary with `-h` 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. Some tests also use Google Mock, which provides a framework for creating mock implementations of C++ classes. Both components are included in the source tree under `src/external/googletest/`, and are 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/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.

8.8.7 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 ran, and some other tests are ran 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.

8.8.8 Change management

GROMACS change management uses git and GitLab for code uploading and testing as well as issues tracking. (For change submission guidelines, refer to *Contribute to GROMACS* (page 568).)

git GROMACS uses git as the version control system. Instructions for setting up git for GROMACS, as well as tips and tricks for its use, can be found in *Change Management* (page 581).

Other basic tutorial material for `git` can be found on the web.

GitLab Bugs and issues, as well as some random features and discussions, are tracked, and all code changes go through a code review system at https://gitlab.com/gromacs/gromacs.

Build testing All changes pushed to GitLab are automatically compiled and otherwise checked on various platforms. *Automation and Infrastructure* (page 618) documents how builds are automated, providing information on how to replicate the builds (e.g., to diagnose issues). *Release engineering with GitLab* (page 628) provides more information on the technical implementation of the builds.

8.8.9 Build system

CMake Main tool used in the build system.

packaging for distribution (CPack)

unit testing (CTest) GROMACS uses a unit testing framework based on Google C++ Testing Framework (gtest) and CTest. All unit tests are automatically run in GitLab CI for each commit. Details can be found on a separate page on *Unit testing* (page 636).

clang static analyzer

coverage

regression tests

floating-point exceptions In debug builds, floating-point exceptions (FPEs) are generated whenever one of the following operations is encountered: division by zero, floating-point overflow, invalid operation (e.g., taking sqrt of a negative number). Such checks are *not* performed in the following configurations:

- release build,
- any build by GCC 7.x or Clang with optimizations,
- build with SYCL support.

In these configurations, FPEs can be enabled by adding `-fpexcept` flag to `gmx` invocation. However, FPEs are not supported on Windows and non-x86 Apple hardware. See `api/legacy/include/gromacs/math/utilities.h` for more details.

8.8.10 Code formatting and style

The tools and scripts listed below are used to automatically check/apply formatting that follows GROMACS style guidelines described on a separate page: *Style guidelines* (page 591).

clang-format We use clang-format to enforce a consistent coding style, with the settings recorded in `.clang-format` in the main tree. See *Setting up clang-format* (page 631) for details.

clang-tidy The source code linter clang-tidy is used to enforce common restrictions to the code, with the checks collected under `.clang-tidy` at the top of the main tree. See *Setting up clang-tidy* (page 632) for details.

`admin/copyright.py` This Python script adds and formats copyright headers in source files. `copyright.sh` (see below) uses the script to check/update copyright years on changed files automatically.

`admin/copyright.sh` This `bash` script runs the `copyright.py` python script to enforce correct copyright information in all files that have local changes and checks that they conform to the prescribed style. Optionally, the script can also apply changes to make the files conform. This script is automatically run by the CI to ensure that all commits adhere to *Guidelines for code formatting* (page 591). If the copyright job does not succeed, it means that this script has something to complain. See *Automatic source code formatting* (page 631) for details.

`admin/clang-format.sh` This script enforces coding style using clang-format. This script is automatically run by our CI to ensure that all commits adhere to *Guidelines for code formatting* (page 591).

`admin/clang-tidy.sh` The clang-tidy code correctness restrictions are enforced by this script. The script is also used by the CI to verify the code, in addition to nightly compilations using clang-tidy on the whole tree.

`admin/git-pre-commit` This sample git pre-commit hook can be used if one wants to apply `clang-tidy.sh`, `copyright.sh` and `clang-format.sh` automatically before every commit to check for formatting issues. See *Automatic source code formatting* (page 631) for details.

`docs/doxygen/includesorter.py` This Python script sorts and reformats #include directives according to the guidelines at *Guidelines for #include directives* (page 593). Details are documented on a separate page (with the whole suite of Python scripts used for source code checks): *Include order sorting* (page 630).

include directive checker In its present form, the above include sorter script cannot be conveniently applied in the formatting script. To check for issues, it is instead integrated into a `check-source` build target. When this target is built, it also checks for include formatting issues. Internally, it uses the sorter script. This check is run in the CI as part of the Documentation job. Details for the checking mechanism are on a separate page (common for several checkers): *Source tree checker scripts* (page 628).

`admin/reformat_all.sh` This `bash` script runs clang-format/`copyright.py`/include sorter on all relevant files in the source tree (or in a particular directory). The script can also produce the list of files where these scripts are applied, for use with other scripts. See *Automatic source code formatting* (page 631) for details.

git attributes git attributes (specified in `.gitattributes` files) are used to annotate which files are subject to automatic formatting checks (and for automatic reformatting by the above scripts). See `man gitattributes` for an overview of the mechanism. We use the `filter` attribute to specify the type of automatic checking/formatting to apply. Custom attributes are used for specifying some build system dependencies for easier processing in CMake.

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 183). 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 MdrunIO tests fail on Power9

The mentioned tests fail with small differences in the reported energies that are just outside of the set tolerances when using exactly 2 ranks.

Issue 3747

CHAPTER

NINE

DOXYGEN DOCUMENTATION

The doxygen code documentation is available on the GROMACS webpage.

PYTHON MODULE INDEX

g

gmxapi, 552
gmxapi._gmxapi, 558
gmxapi._logging, 556
gmxapi.exceptions, 556
gmxapi.version, 557

s

scripted_gmx_docker_builds, 627

u

utility, 626