GROMACS

Groningen Machine for Chemical Simulations

USER MANUAL

Version 4.5

GROMACS
USER MANUAL

Version 4.5

David van der Spoel, Erik Lindahl, Berk Hess

Carsten Kutzner
Aldert R. van Buuren
Emile Apol
Pieter J. Meulenhoff
D. Peter Tieleman
Alfons L.T.M. Sijbers
K. Anton Feenstra
Rudi van Drunen
Herman J.C. Berendsen

© 1991–2000: Department of Biophysical Chemistry, University of Groningen. Nijenborgh 4, 9747 AG Groningen, The Netherlands.

© 2001–2006: The GROMACS development team.

More information can be found on our website: www.gromacs.org.

Preface & Disclaimer

This manual is not complete and has no pretention 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 are welcome, please send them by e-mail to gromacs@gromacs.org, or to one of the mailing lists (see www.gromacs.org).

We try to 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. Any revision numbers (like 3.1.1) are however independent, to make it possible to implement bug fixes and manual improvements if necessary.

On-line Resources

You can find more documentation and other material at our homepage www.gromacs.org. Among other things there is an on-line reference, several GROMACS mailing lists with archives and contributed topologies/force fields.

Citation information

When citing this document in any scientific publication please refer to it as:

> D. van der Spoel, E. Lindahl, B. Hess, A. R. van Buuren, E. Apol, P. J. Meulenhoff, D. P. Tieleman, A. L. T. M. Sijbers, K. A. Feenstra, R. van Drunen and H. J. C. Berendsen, *Gromacs User Manual version 4.5*, www.gromacs.org (2005)

However, we prefer that you cite (some of) the GROMACS papers [1, 2, 3, 4] when you publish your results. Any future development depends on academic research grants, since the package is distributed as free software!

Current development

Gromacs is a joint effort, with contributions from lots of developers around the world. The core development is currently taking place at

- Department of Cellular and Molecular Biology, Uppsala University, Sweden.
  (David van der Spoel).
- Stockholm Bioinformatics Center, Stockholm University, Sweden
  (Erik Lindahl).
- Stockholm Bioinformatics Center, Stockholm University, Sweden
  (Berk Hess)

GROMACS is *Free Software*

The entire GROMACS package is available under the GNU General Public License. This means it's free as in free speech, not just that you can use it without paying us money. For details, check the COPYING file in the source code or consult www.gnu.org/copyleft/gpl.html.

The GROMACS source code and and selected set of binary packages are available on our homepage, www.gromacs.org. Have fun.

Contents

Chapter 1

Introduction

1.1 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 aim 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 in (a) *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 in a liquid, and (b) *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 [5], to *mesoscopic dynamics* describing densities rather than atoms and fluxes as response to thermodynamic gradients rather than velocities or accelerations as response to forces [6].

For the generation of a representative equilibrium ensemble two methods are available: (a) *Monte Carlo simulations* and (b) *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.

1.2 Molecular Dynamics Simulations

MD simulations solve Newton's equations of motion for a system of N interacting atoms:

$$m_i \frac{\partial^2 \boldsymbol{r}_i}{\partial t^2} = \boldsymbol{F}_i, \; i = 1 \ldots N. \tag{1.1}$$

The forces are the negative derivatives of a potential function $V(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N)$:

$$\boldsymbol{F}_i = - \frac{\partial V}{\partial \boldsymbol{r}_i} \tag{1.2}$$

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

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· · ·O	libration	400– 700
O-H· · ·O	stretch	50– 200

Table 1.1: Typical vibrational frequencies (wavenumbers) in molecules and hydrogen-bonded liquids. Compare $kT/h = 200$ cm^{-1} at 300 K.

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_BT/h. Now at room temperature the wavenumber $\sigma = 1/\lambda = \nu/c$ at which $h\nu = k_BT$ 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 1.1). What can we do?

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: [7]

$$U^{QM} = U^{cl} + kT\left(\frac{1}{2}x - 1 + \frac{x}{e^x - 1}\right) \tag{1.3}$$

$$C_V^{QM} = C_V^{cl} + k\left(\frac{x^2 e^x}{(e^x - 1)^2} - 1\right), \tag{1.4}$$

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 equation of motion. The rational 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 [8]. 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 [9].

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 user-modified 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 bio macro-molecules 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. Due to the minimum-image convention (only one image of each particle in the periodic boundary conditions is considered for a pair interaction), the cut-off range can not 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 [10, 11].

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). This we must avoid if we wish to simulate a bulk system. So we use periodic boundary conditions, to avoid real phase boundaries. But liquids are not crystals, so 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 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.

1.3 Energy Minimization and Search Methods

As mentioned in sec. 1.1, in many cases energy minimization is required. GROMACS provides a number of methods for local energy minimization, as detailed in sec. 3.10.

The potential energy function of a (macro)molecular system is a very complex landscape (or *hyper surface*) 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 [12]]. 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 [13]), 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 [14], 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 building 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.* [15]) 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.

Chapter 2

Definitions and Units

2.1 Notation

The following conventions for mathematical typesetting are used throughout this document:

Item	Notation	Example
Vector	Bold italic	$\boldsymbol{r}_i$
Vector Length	Italic	r_i

We define the *lowercase* subscripts i, j, k and l to denote particles: $\boldsymbol{r}_i$ is the *position vector* of particle i, and using this notation:

$$\boldsymbol{r}_{ij} = \boldsymbol{r}_j - \boldsymbol{r}_i \tag{2.1}$$

$$r_{ij} = |\boldsymbol{r}_{ij}| \tag{2.2}$$

The force on particle i is denoted by $\boldsymbol{F}_i$ and

$$\boldsymbol{F}_{ij} = \text{force on } i \text{ exerted by } j \tag{2.3}$$

Please note that we changed notation as of version 2.0 to $\boldsymbol{r}_{ij} = \boldsymbol{r}_j - \boldsymbol{r}_i$ since this is the notation commonly used. If you encounter an error, let us know.

2.2 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 2.1.

Consistent with these units are a set of derived units, given in Table 2.2.

The **electric conversion factor** $f = \frac{1}{4 \pi \varepsilon_o} = 138.935\,485(9)$ kJ mol^{-1} nm e^{-2}. It relates the mechanical quantities to the electrical quantities as in

$$V = f \frac{q^2}{r} \text{ or } F = f \frac{q^2}{r^2} \tag{2.4}$$

Quantity	Symbol	Unit
length	r	nm $= 10^{-9}$ m
mass	m	u (atomic mass unit) $= 1.6605402(10) \times 10^{-27}$ kg
		(1/12 the mass of a ^{12}C atom)
		$1.6605402(10) \times 10^{-27}$ kg
time	t	ps $= 10^{-12}$ s
charge	q	e = electronic charge $= 1.60217733(49) \times 10^{-19}$ C
temperature	T	K

Table 2.1: Basic units used in GROMACS. Numbers in parentheses give accuracy.

Quantity	Symbol	Unit
energy	E, V	kJ mol^{-1}
Force	$\boldsymbol{F}$	kJ mol^{-1} nm^{-1}
pressure	p	kJ mol^{-1} nm^{-3} $= 10^{30}/N_{AV}$ Pa
		$1.660\,54 \times 10^6$ Pa $= 16.6054$ Bar
velocity	v	nm ps^{-1} $= 1000$ m/s
dipole moment	μ	e nm
electric potential	Φ	kJ mol^{-1} e^{-1} $= 0.010\,364\,272(3)$ Volt
electric field	E	kJ mol^{-1} nm^{-1} e^{-1} $= 1.036\,427\,2(3) \times 10^7$ V/m

Table 2.2: Derived units

Electric potentials Φ and electric fields $\boldsymbol{E}$ 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 recommend strongly to follow the usual practice to include the factor f in expressions that evaluate Φ and $\boldsymbol{E}$:

$$\Phi(\boldsymbol{r}) = f \sum_j \frac{q_j}{|\boldsymbol{r} - \boldsymbol{r}_j|} \tag{2.5}$$

$$\boldsymbol{E}(\boldsymbol{r}) = f \sum_j q_j \frac{(\boldsymbol{r} - \boldsymbol{r}_j)}{|\boldsymbol{r} - \boldsymbol{r}_j|^3} \tag{2.6}$$

With these definitions $q\Phi$ is an energy and $q\boldsymbol{E}$ is a force. The units are those given in Table 2.2: 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: 1.439965 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 the kcal/mol (= 4.184 kJ/mol) is used instead of kJ/mol 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 kJ/mol, 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 2.3. 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 $0.008\,314\,51$ kJ mol^{-1} K^{-1}.

Symbol	Name	Value
N_{AV}	Avogadro's number	$6.022\,136\,7(36) \times 10^{23}$ mol^{-1}
R	gas constant	$8.314\,510(70) \times 10^{-3}$ kJ mol^{-1} K^{-1}
k_B	Boltzmann's constant	idem
h	Planck's constant	$0.399\,031\,32(24)$ kJ mol^{-1} ps
$\hbar$	Dirac's constant	$0.063\,507\,807(38)$ kJ mol^{-1} ps
c	velocity of light	299 792.458 nm/ps

Table 2.3: Some Physical Constants

Quantity	Symbol	Relation to SI
Length	r*	r σ^{-1}
Mass	m*	m M^{-1}
Time	t*	t σ^{-1} $\sqrt{\epsilon/M}$
Temperature	T*	k$_B$T ϵ^{-1}
Energy	E*	E ϵ^{-1}
Force	F*	F σ ϵ^{-1}
Pressure	P*	P $\sigma^3\epsilon^{-1}$
Velocity	v*	v $\sqrt{M/\epsilon}$
Density	ρ*	N σ^3 V^{-1}

Table 2.4: Reduced Lennard-Jones quantities

2.3 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. There is one exception: the *temperature*, which is expressed in $0.008\,314\,51$ reduced units. This is a consequence of the use of 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…units; if a reduced temperature of 1 is required, the GROMACS temperature should be 120.2717.

In Table 2.4 quantities are given for LJ potentials:

$$V_{LJ} = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \qquad (2.7)$$

Chapter 3

Algorithms

3.1 Introduction

In this chapter we first give describe some general concepts used in GROMACS: *periodic boundary conditions* (sec. 3.2) and the *group concept* (sec. 3.3). The MD algorithm is described in sec. 3.4: 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. 3.10. 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 4. 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 8.

3.2 Periodic boundary conditions

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. 3.1). 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* [16] are closer to being a sphere than a cube is, and are therefore

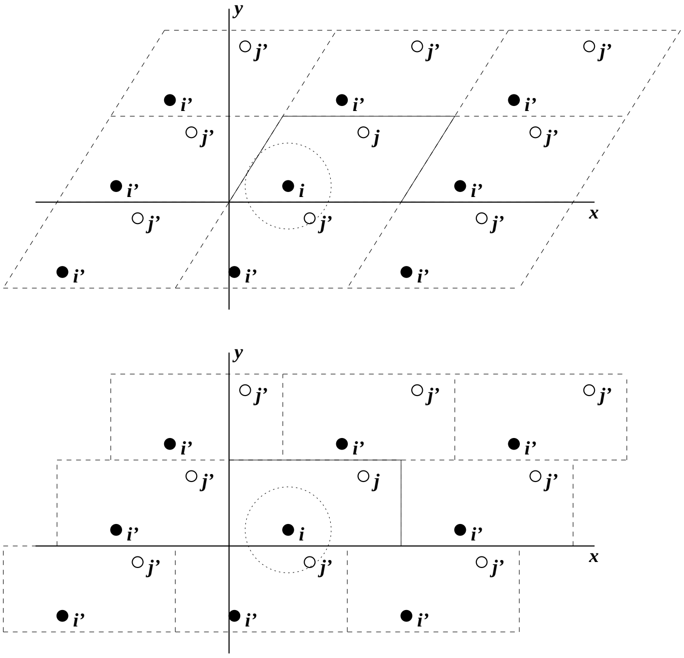


Figure 3.1: Periodic boundary conditions in two dimensions.

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 [17]. 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 **a**,**b** and **c**. The box vectors must satisfy the following conditions:

$$a_y = a_z = b_z = 0 \tag{3.1}$$

$$a_x > 0, \quad b_y > 0, \quad c_z > 0 \tag{3.2}$$

$$|b_x| \leq \frac{1}{2}\,a_x, \quad |c_x| \leq \frac{1}{2}\,a_x, \quad |c_y| \leq \frac{1}{2}\,b_y \tag{3.3}$$

Equations 3.1 can always be satisfied by rotating the box. Inequalities (3.2) and (3.3) 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 reasons, as illustrated in Fig. 3.1 for a 2-dimensional system. From the output trajectory it might therefore seem as if the simulation was done in a rectangular box. The program `trjconv` can be used to convert the trajectory to a different unit-cell representation.

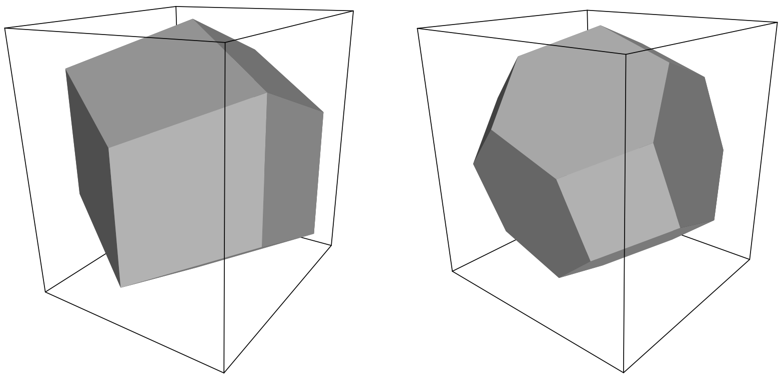


Figure 3.2: A rhombic dodecahedron and truncated octahedron (arbitrary orientations).

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

Table 3.1: The cubic box, the rhombic dodecahedron and the truncated octahedron.

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.

3.2.1 Some useful box types

The three most useful box types for simulations of solvated systems are described in Table 3.1. The rhombic dodecahedron (Fig. 3.2) 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 3.1, 3.2 and 3.3. The program `editconf` 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.

3.2.2 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:

$$R_c < \frac{1}{2} \min(\|\mathbf{a}\|, \|\mathbf{b}\|, \|\mathbf{c}\|), \tag{3.4}$$

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:

$$R_c < \min(a_x, b_y, c_z) \tag{3.5}$$

For simple search the extra restriction is stronger:

$$R_c < \frac{1}{2} \min(a_x, b_y, c_z) \tag{3.6}$$

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 3.4.3). Restriction (3.5) ensures that only 26 images need to be considered.

3.3 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:

T-coupling group The temperature coupling parameters (reference temperature, time constant, number of degrees of freedom, see 3.4.4) 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, in particular when constraints are used as well (in this case you will get very large pressures). Because of this it is recommended to not combine freeze groups with constraints and pressure coupling. For the sake of equilibration it could suffice to start with freezing in a constant volume simulation, and afterward use position restraints in conjunction with constant pressure.

Accelerate group On each atom in an 'accelerate group' an acceleration $\boldsymbol{a}^g$ is imposed. This is equivalent to an external force. This feature makes it possible to drive the system into a non-equilibrium state and enables the performance of non-equilibrium MD and hence to obtain transport properties.

Energy monitor group Mutual interactions between all energy monitor groups are compiled during the simulation. This is done separately for Lennard-Jones and Coulomb terms. In principle up to 256 groups could be defined, but that would lead to 256×256 items! Better use this concept sparingly.

All non-bonded interactions between pairs of energy monitor groups can be excluded (see sec. 7.3). 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.

XTC output group In order to reduce the size of the XTC trajectory file, only a subset of all particles can be stored. All XTC groups that are specified are saved, the rest is not. If no XTC groups are specified, than all atoms are saved to the XTC file.

The use of groups in analysis programs is described in chapter 8.

3.4 Molecular Dynamics

A global flow scheme for MD is given in Fig. 3.3. 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 on-line manual at www.gromacs.org.

THE GLOBAL MD ALGORITHM

---

1. Input initial conditions

Potential interaction V as a function of atom positions
Positions $\boldsymbol{r}$ of all atoms in the system
Velocities $\boldsymbol{v}$ of all atoms in the system
⇓

---

repeat 2,3,4 for the required number of steps:

---

2. Compute forces

The force on any atom

$$\boldsymbol{F}_i = -\frac{\partial V}{\partial \boldsymbol{r}_i}$$

is computed by calculating the force between non-bonded atom pairs:

$$\boldsymbol{F}_i = \sum_j \boldsymbol{F}_{ij}$$

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 are computed.
⇓

3. Update configuration

The movement of the atoms is simulated by numerically solving Newton's equations of motion

$$\frac{\mathrm{d}^2\boldsymbol{r}_i}{\mathrm{d}t^2} = \frac{\boldsymbol{F}_i}{m_i}$$

or

$$\frac{\mathrm{d}\boldsymbol{r}_i}{\mathrm{d}t} = \boldsymbol{v}_i ; \quad \frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = \frac{\boldsymbol{F}_i}{m_i}$$

⇓

4. if required: **Output step**
write positions, velocities, energies, temperature, pressure, etc.

Figure 3.3: The global MD algorithm

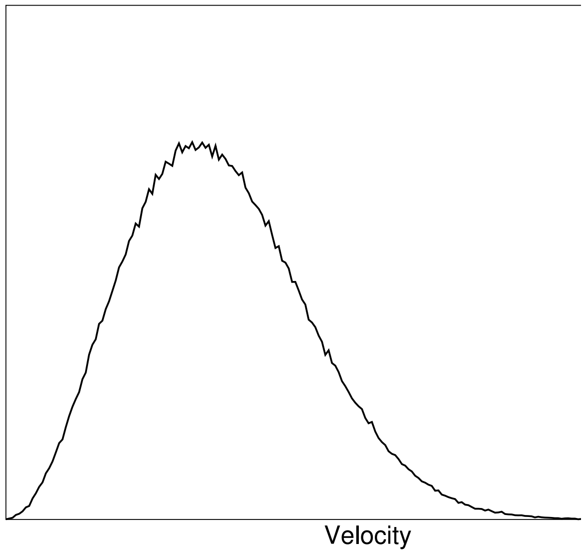


Figure 3.4: A Maxwell-Boltzmann velocity distribution, generated from random numbers.

3.4.1 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 4 and 5, respectively. All this information is static; it is never modified during the run.

Coordinates and velocities

Then, before a run starts, the box size and the coordinates and velocities of all particles are required. The box size and shape is determined by three vectors (nine numbers) $\boldsymbol{b}_1, \boldsymbol{b}_2, \boldsymbol{b}_3$, 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 3.4.4), 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 $v_i, i = 1 \dots 3N$ with a Maxwell-Boltzmann distribution (Fig. 3.4) at a given absolute temperature T:

$$p(v_i) = \sqrt{\frac{m_i}{2\pi kT}} \exp\left(-\frac{m_i v_i^2}{2kT}\right) \tag{3.7}$$

where k is Boltzmann's constant (see chapter 2). 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 eqn. 3.13).

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 cause give such problems.

3.4.2 Neighbor searching

As mentioned in chapter 4, 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. 3.1, 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. Equation 3.1 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 3.3, 3.4 and 3.5 ensure that we can find the 14 nearest triclinic images within a linear combination which 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. GROMACS employs a *pair list* that contains those particle pairs for which non-bonded forces must be calculated. The pair list contains atoms i, a displacement vector for atom i, and all particles j that are within `rshort` of this particular image of atom i. The list is updated every `nstlist` steps, where `nstlist` is typically 10. There is an option to calculate the total non-bonded force on each particle due to all particle in a shell around the list cut-off, *i.e.* at a distance between `rshort` and `rlong`. This force is calculated during the pair list update and retained during `nstlist` steps.

To make the neighbor list all particles that are close (*i.e.* within the neighbor list cut-off) to a

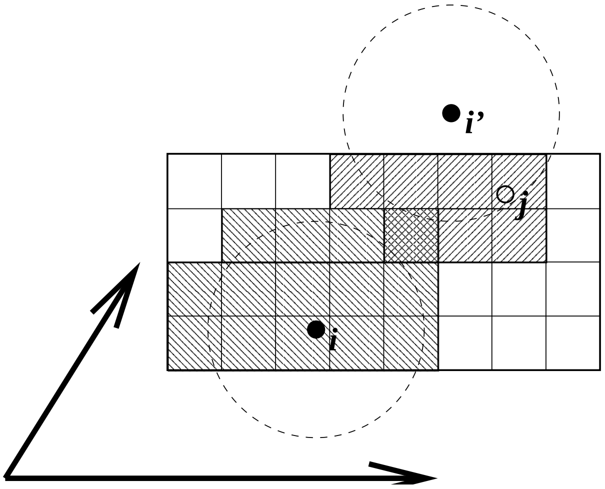


Figure 3.5: Grid search in two dimensions. The arrows are the box vectors.

given particle must be found. This searching, usually called neighbor searching (NS), involves periodic boundary conditions and determining the *image* (see sec. 3.2). Without periodic boundary conditions a simple $O(N^2)$ algorithm must be used. With periodic boundary conditions a grid search can be used, which is $O(N)$.

To completely avoid cut-off artifacts, the non-bonded potentials 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. 4.1.5). One then has a buffer with the size equal to the neighbor list cut-off minus the longest interaction cut-off. In this case one can also choose to let `mdrun` only update the neighbor list when required. That is when one or more particles have moved more than half the buffer size from the center of geometry of the charge group they belong to (see sec. 3.4.2) as determined at the previous neighbor search. This option guarantees that their are no cut-off artifacts. Note that for larger systems this comes at a high computational cost, since the neighbor list update frequency will be determined by just one or two particles moving slightly beyond the half buffer length (which not even necessarily implies that the neighbor list is invalid), while 99.99% of the particles are fine.

Simple search

Due to eqns. 3.1 and 3.6, the vector $\boldsymbol{r}_{ij}$ connecting images within the cut-off R_c can be found by constructing:

$$\begin{aligned}
\boldsymbol{r}''' &= \boldsymbol{r}_j - \boldsymbol{r}_i && (3.8)\\
\boldsymbol{r}'' &= \boldsymbol{r}''' - \mathbf{c} * \mathtt{round}(r'''_z/c_z)) && (3.9)\\
\boldsymbol{r}' &= \boldsymbol{r}'' - \mathbf{b} * \mathtt{round}(r''_y/b_y) && (3.10)\\
\boldsymbol{r}_{ij} &= \boldsymbol{r}' - \mathbf{a} * \mathtt{round}(r'_x/a_x) && (3.11)
\end{aligned}$$

When distances between two particles in a triclinic box are needed that do not obey eqn. 3.1, many shifts of combinations of box vectors need to be considered to find the nearest image.

Grid search

The grid search is schematically depicted in Fig. 3.5. 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. 3.5 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 reduce cut-off artifacts of Coulomb interactions. When a plain cut-off is used, significant jumps in the potential and forces arise when atoms with (partial) charges move in and out of the cut-off radius. When all chemical moieties have a net charge of zero, these jumps can be reduced by moving groups of atoms with net charge zero, called charge groups, in and out of the neighbor list. This reduces the cut-off effects from the charge-charge level to the dipole-dipole level, which decay much faster. With the advent of full range electrostatics methods, such as particle mesh Ewald (sec. 4.9.2), the use of charge groups is no longer required for accuracy. It might even have a slight negative effect on the accuracy or efficiency, depending on how the neighbor list is made and the interactions are calculated.

But there is still an important reason for using "charge groups": efficiency. Where applicable, neighbor searching is carried out on the basis of charge groups are defined in the molecular topology. If the nearest image distance between the *geometrical centers* of the atoms of two charge groups is less than the cut-off radius, all atom pairs between the charge groups are included in the pair list. The neighbor searching for a water system, for instance, is $3^2 = 9$ times faster when each molecule is treated as a charge group. Also the highly optimized water force loops (see sec. B.2.1) only work when all atoms in a water molecule form a single charge group. Currently the name *neighbor-search group* would be more appropriate, but the name charge group is retained for historical reasons. When developing a new force field, the advice is to use charge groups of 3 to 4 atoms for optimal performance. For all-atom force fields this is relatively easy, as one can simply put hydrogen atoms, and in some case oxygen atoms, in the same charge group as the heavy atom they are connected to; for example: CH_3, CH_2, CH, NH_2, NH, OH, CO_2, CO.

3.4.3 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 *groups* of atoms that are separately defined (see sec. 3.3).

Kinetic energy and temperature

The temperature is given by the total kinetic energy of the N-particle system:

$$E_{kin} = \frac{1}{2}\sum_{i=1}^{N} m_i v_i^2 \tag{3.12}$$

From this the absolute temperature T can be computed using:

$$\frac{1}{2} N_{df} kT = E_{kin} \tag{3.13}$$

where k is Boltzmann's constant and N_{df} is the number of degrees of freedom which can be computed from:

$$N_{df} \ = \ 3N - N_c - N_{com} \tag{3.14}$$

Here N_c is the number of *constraints* imposed on the system. When performing molecular dynamics $N_{com} = 3$ 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 $N_{com} = 6$. When more than one temperature coupling group is used, the number of degrees of freedom for group i is:

$$N^i_{df} \ = \ (3N^i - N^i_c)\frac{3N - N_c - N_{com}}{3N - N_c} \tag{3.15}$$

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:

$$\mathbf{E}_{kin} = \frac{1}{2}\sum_{i}^{N} m_i \boldsymbol{v}_i \otimes \boldsymbol{v}_i \tag{3.16}$$

Pressure and virial

The pressure tensor **P** is calculated from the difference between kinetic energy E_{kin} and the virial $\mathbf{\Xi}$

$$\mathbf{P} = \frac{2}{V}(\mathbf{E}_{kin} - \mathbf{\Xi}) \tag{3.17}$$

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:

$$P = \mathrm{trace}(\mathbf{P})/3 \tag{3.18}$$

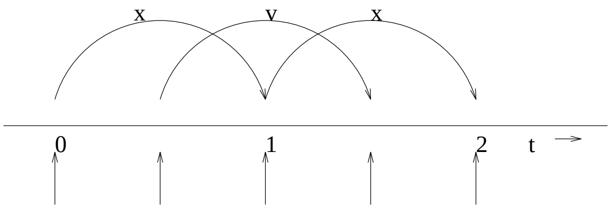


Figure 3.6: The Leap-Frog integration method. The algorithm is called Leap-Frog because r and v are leaping like frogs over each others back.

The virial Ξ tensor is defined as

$$\Xi = -\frac{1}{2}\sum_{i<j} \boldsymbol{r}_{ij} \otimes \boldsymbol{F}_{ij} \tag{3.19}$$

The GROMACS implementation of the virial computation is described in sec. B.1.

3.4.4 The leap frog integrator

The default MD integrator in GROMACS is the so-called *leap-frog* algorithm [18] for the integration of the equations of motion. When extremely accurate integration is temperature and/or pressure coupling velocity Verlet integrators are also present and may be preferable (see 3.4.5). The leap-frog algorithm uses positions $\boldsymbol{r}$ at time t and velocities $\boldsymbol{v}$ at time $t - \frac{1}{2}\Delta t$; it updates positions and velocities using the forces $\boldsymbol{F}(t)$ determined by the positions at time t:

$$\begin{aligned} \boldsymbol{v}(t + \frac{1}{2}\Delta t) &= \boldsymbol{v}(t - \frac{1}{2}\Delta t) + \frac{\Delta t}{m}\boldsymbol{F}(t) && (3.20) \\ \boldsymbol{r}(t + \Delta t) &= \boldsymbol{r}(t) + \Delta t \boldsymbol{v}(t + \frac{1}{2}\Delta t) && (3.21) \end{aligned}$$

The algorithm is visualized in Fig. 3.6. It produces trajectories that are identical to the Verlet [19] algorithm:

$$\boldsymbol{r}(t + \Delta t) = 2\boldsymbol{r}(t) - \boldsymbol{r}(t - \Delta t) + \frac{1}{m}\boldsymbol{F}(t)\Delta t^2 + O(\Delta t^4) \tag{3.22}$$

The algorithm is of third order in $\boldsymbol{r}$ and is time-reversible. See ref. [20] 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.

3.4.5 The velocity Verlet integrator

The velocity Verlet algorithm [21] is also implemented in Gromacs, though it is not yet fully integrated with all sets of options. In velocity Verlet positions $\boldsymbol{r}$ and velocities $\boldsymbol{v}$ at time t are used to integrate the equations of motion; velocities at the previous half step are not required.

$$\begin{aligned} \boldsymbol{v}(t + \frac{1}{2}\Delta t) &= \boldsymbol{v}(t) + \frac{\Delta t}{2m}\boldsymbol{F}(t) && (3.23) \\ \boldsymbol{r}(t + \Delta t) &= \boldsymbol{r}(t) + \Delta t\, \boldsymbol{v}(t + \frac{1}{2}\Delta t) && (3.24) \end{aligned}$$

$$\boldsymbol{v}(t+\Delta t) \quad = \quad \boldsymbol{v}(t+\frac{1}{2}\Delta t)+\frac{\Delta t}{2m}\boldsymbol{F}(t+\Delta t) \tag{3.25}$$

or equivalently:

$$\begin{aligned} \boldsymbol{r}(t+\Delta t) &= \boldsymbol{r}(t)+\Delta t\,\boldsymbol{v}+\frac{\Delta t^2}{2m}\boldsymbol{F}(t) && (3.26)\\ \boldsymbol{v}(t+\Delta t) &= \boldsymbol{v}(t)+\frac{\Delta t}{2m}\left[\boldsymbol{F}(t)+\boldsymbol{F}(t+\Delta t)\right] && (3.27) \end{aligned}$$

With no temperature or pressure coupling, and with corresponding starting points, leapfrog 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 $\boldsymbol{x}(0)$ and $\boldsymbol{v}(0)$, leapfrog and velocity Verlet will *not* give identical trajectories, as leapfrog will interpret the velocities as corresponding to $t=-\frac{1}{2}\Delta t$, while velocity Verlet will interpret them as corresponding to the timepoint $t=0$.

3.4.6 Understanding integrators: The Trotter decomposition

To further understand the relationship between velocity Verlet and leapfrog integration, we introduce the reversible Trotter formulation of dynamics, which is also useful to understanding implementations of thermostats and barostats in Gromacs.

A system of coupled, first order differential equations can be evolved from time $t=0$ to time t by applying the evolution operator

$$\begin{aligned} \Gamma(t) &= \exp(iLt)\Gamma(0)\\ iL &= \dot{\Gamma}\cdot\nabla_\Gamma \end{aligned} \tag{3.28}$$

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:

$$\Gamma(t)=\prod_{i=1}^{P}\exp(iL\Delta t)\Gamma(0) \tag{3.29}$$

For NVE dynamics, the Liouville operator is:

$$iL=\sum_{i=1}^{N}\boldsymbol{v}_i\cdot\nabla_{\boldsymbol{r}_i}+\sum_{i=1}^{N}\frac{1}{m_i}\boldsymbol{F}(r_i)\cdot\nabla_{\boldsymbol{v}_i} \tag{3.30}$$

If this is split into two operators:

$$\begin{aligned} iL_1 &= \sum_{i=1}^{N}\frac{1}{m_i}\boldsymbol{F}(r_i)\cdot\nabla_{\boldsymbol{v}_i}\\ iL_2 &= \sum_{i=1}^{N}\boldsymbol{v}_i\cdot\nabla_{\boldsymbol{r}_i} \end{aligned} \tag{3.31}$$

Then a short time, symmetric, and thus reversible approximation of the true dynamics will be:

$$\exp(iL\Delta t) = \exp(iL_2\frac{1}{2}\Delta t)\exp(iL_1\Delta t)\exp(iL_2\frac{1}{2}\Delta t) + \mathcal{O}(\Delta t^3) \tag{3.32}$$

Which 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:

$$\begin{aligned} \exp(iLn\Delta t) &\approx \Big(\exp(iL_2\frac{1}{2}\Delta t)\exp(iL_1\Delta t)\exp(iL_2\frac{1}{2}\Delta t)\Big)^n \\ &\approx \exp(iL_2\frac{1}{2}\Delta t)\Big(\exp(iL_1\Delta t)\exp(iL_2\Delta t)\Big)^{n-1} \\ &\quad \exp(iL_1\Delta t)\exp(iL_2\frac{1}{2}\Delta t) \end{aligned} \tag{3.33}$$

This formalism allows us to easily see the difference between the different flavors of Verlet integrators. The leapfrog integrator can be seen as starting with Eq. 3.32 with the $\exp(iL_1\Delta t)$ term, instead of the half-step velocity term, yielding:

$$\exp(iLn\Delta t) = \exp(iL_1\Delta t)\exp(iL_2\Delta t) + \mathcal{O}(\Delta t^3) \tag{3.34}$$

Where 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:

$$\exp(iLn\Delta t) \approx \Big(\exp(iL_1\Delta t)\exp(iL_2\Delta t)\Big)^n \tag{3.35}$$

Although this does not at first 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 it 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 leapfrog 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 the sum over all particles of:

$$\begin{aligned} KE_{\text{full}}(t) &= \sum_i \left(\frac{1}{2m_i}(\boldsymbol{v}_i(t)\right)^2 \\ &= \sum_i \frac{1}{2m_i}\left(\frac{1}{2}\boldsymbol{v}_i(t - \frac{1}{2}\Delta t) + \frac{1}{2}\boldsymbol{v}_i(t + \frac{1}{2}\Delta t)\right)^2 \end{aligned} \tag{3.36}$$

with the square on the *outside* of the average. Standard Leapfrog 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 of

$$KE_{\text{average}}(t) = \sum_i \frac{1}{2m_i}\left(\frac{1}{2}\boldsymbol{v}_i(t - \frac{1}{2}\Delta t)^2 + \frac{1}{2}\boldsymbol{v}_i(t + \frac{1}{2}\Delta t)^2\right) \tag{3.37}$$

With the square inside the average.

A nonstandard 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 leapfrog, is also now implemented in Gromacs (as mdp option *md-vv-avek*). Without temperature and pressure coupling, velocity Verlet with half-step-averaged kinetic energies and leapfrog will be identical up to numerical precision. For temperature and pressure control schemes, however, velocity Verlet with half-step-averaged kinetic energies and leapfrog will be different, as will be discussed in the section in thermostats and barostats.

The half-step-averaged kinetic energy temperature are slightly more accurate, in that for a given time step size, the difference in average kinetic energies using the half-step-averaged kinetic energies (*md* and *md-vv-avek*) will be closer to the kinetic energy obtained in the limit of small step size than will the full-step kinetic energy (using *md-vv*). For NVE simulations, this difference is usually not significant, since the trajectories are still identical; it makes a difference in the way the simulations are *interpreted*, not in the trajectories that are actually produced. The only difference is that the effective temperature will be interpreted slightly differently. For NVT simulations, however, there *will* be a difference, as discussed in the section on temperature control, since we measure how to adjust the thermostat based on the estimated temperature kinetic energy. Although the kinetic energy is more accurate with the averaged half step method (in that it it is also more noisy. The noise in the half-step-averaged kinetic energy 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 the trajectories are identical.

In general, the velocity Verlet integrator has been tuned with methods that give the highest degree of thermodynamic accuracy but has not yet been optimized for performance. The integration itself takes negligibly more time than leapfrog, but currently twice as many communication calls are 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, leapfrog will be the best 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.

3.4.7 Twin-range cut-offs

To save computation time, slowly varying forces can be calculated less often than rapidly varying forces. In GROMACS such a multiple time step splitting is possible between short and long range non-bonded interactions. In GROMACS versions up to 4.0 an irreversible integration scheme was used which is also used by the GROMOS simulation package: every n steps the long range forces are determined and these are then also used (without modification) for the next $n-1$ integration steps in eqn. 3.20. Such an irreversible scheme can result in bad energy conservation and, possibly, bad sampling. Since version 4.5, a leap-frog version of the reversible Trotter decomposition scheme[22] is used. In this integrator the long-range forces are determined every n steps and are then integrated into the velocity in eqn. 3.20 using a time step of $\Delta t_{\mathrm{LR}} = n\Delta t$:

$$\boldsymbol{v}(t+\frac{1}{2}\Delta t) = \begin{cases} \boldsymbol{v}(t-\frac{1}{2}\Delta t) + \frac{1}{m}\left[\boldsymbol{v}_{\mathrm{SR}}(t) + n\boldsymbol{F}_{\mathrm{LR}}(t)\right]\Delta t & , \quad \text{step} \ \% \ n = 0 \\ \boldsymbol{v}(t-\frac{1}{2}\Delta t) + \frac{1}{m}\boldsymbol{F}_{\mathrm{SR}}(t)\Delta t & , \quad \text{step} \ \% \ n \neq 0 \end{cases} \tag{3.38}$$

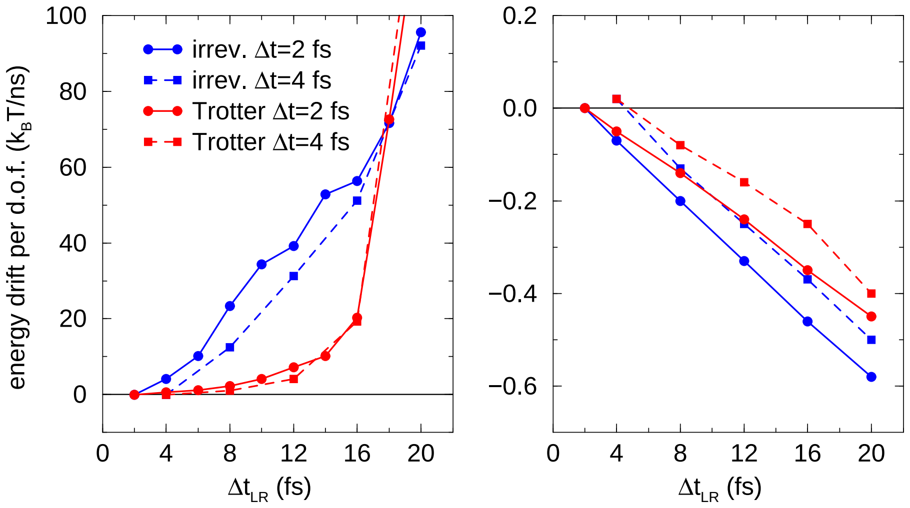


Figure 3.7: Energy drift per degree of freedom in SPC/E water with twin-range cut-offs for reaction field (left) and Lennard Jones interaction (right) as a function of the long-range time step length for the irreversible "GROMOS" scheme and a reversible Trotter scheme.

The parameter n is equal to the neighbor list update frequency. In 4.5, the velocity Verlet version of multiple time-stepping is not yet fully implemented.

Several other simulation packages uses multiple time stepping for bonds and/or the PME mesh forces. In GROMACS we have not implemented this, since we use a different philosophy. Bonds can be constrained (which is also a physically more sound approximation of a 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. 6.5), which brings the shortest time step up to PME mesh update frequency of a multiple time stepping scheme.

As an example we show the energy conservation for integrating the equations of motion for SPC/E water at 300 K. To avoid cut-off effects, reaction field electrostatics with $\epsilon_{RF} = \infty$ and shifted Lennard-Jones interactions are used, both with a buffer region. The long-range interactions were evaluated between 1.0 and 1.4 nm. In Fig. 3.6 one can see that for electrostatics the Trotter scheme does an order of magnitude better up to Δt_{LR} = 16 fs. The electrostatics depends strongly on the orientation of the water molecules, which changes rapidly. For Lennard-Jones interactions the energy drift is linear in Δt_{LR} and roughly two orders of magnitude smaller than for the electrostatics. Lennard-Jones forces are smaller than Coulomb forces and they are mainly affected by translation of water molecules, not rotation.

3.4.8 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. GROMACS can use either the *weak coupling* scheme of

Berendsen [23], the extended ensemble Nosé-Hoover scheme [24, 25], or the velocity rescaling scheme [26] 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 physicality in the dynamics). For larger systems and small drifts, this error is usually negligible, but very few comprehensive comparisons have been carried out, and some caution must be taking in interpreting the results.

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. [27] 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

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{T_0 - T}{\tau} \tag{3.39}$$

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, strictly speaking, one does not generate a proper canonical ensemble, so rigorously, the sampling will indeed be incorrect. This error scales with $1/N$, so for very large systems most ensemble averages properties will not be affected significantly, except for the distribution of the kinetic energy itself. A similar thermostat which does produce a correct ensemble is the velocity rescaling thermostat[26] described below.

The heat flow into or out of the system is effected by scaling the velocities of each particle every step with a time-dependent factor λ, given by

$$\lambda = \left[1 + \frac{\Delta t}{\tau_T}\left\{\frac{T_0}{T(t-\frac{1}{2}\Delta t)} - 1\right\}\right]^{1/2} \tag{3.40}$$

The parameter τ_T is close to, but not exactly equal to the time constant τ of the temperature coupling (eqn. 3.39):

$$\tau = 2C_V\tau_T/N_{df}k \tag{3.41}$$

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 $<=$ λ $<=$ 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.

Velocity rescaling thermostat

The velocity rescaling thermostat[26] is essentially a Berendsen thermostat (see above) with an additional stochastic term which ensures a correct kinetic energy distribution:

$$\mathrm{d}K = (K_0 - K)\frac{\mathrm{d}t}{\tau_T} + 2\sqrt{\frac{KK_0}{N_f}}\frac{\mathrm{d}W}{\sqrt{\tau_T}} \tag{3.42}$$

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. When an NVT ensemble is used, the conserved energy quantity is written to the energy and log file.

Nosé-Hoover temperature coupling

The Berendsen weak coupling algorithm is extremely efficient for relaxing a system to the target temperature, but once your 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, although the difference is usually negligible.

To enable canonical ensemble simulations, GROMACS also supports the extended-ensemble approach first proposed by Nosé [24] and later modified by Hoover[25]. 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 Fig. 3.3 are replaced by

$$\frac{\mathrm{d}^2\boldsymbol{r}_i}{\mathrm{d}t^2} = \frac{\boldsymbol{F}_i}{m_i} - \frac{p_\xi}{Q}\frac{\mathrm{d}\boldsymbol{r}_i}{\mathrm{d}t}, \tag{3.43}$$

where the equation of motion for the heat bath parameter ξ is

$$\frac{\mathrm{d}p_\xi}{\mathrm{d}t} = (T - T_0). \tag{3.44}$$

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]

---

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

The conserved quantity for the Nosè-Hoover equations of motion is not the total energy, but rather

$$H = \sum_{i=1}^{N} \frac{\boldsymbol{p}_i}{2m_i} + U\left(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N\right) + \frac{p_\xi^2}{2Q} + N_f kT\xi \tag{3.45}$$

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

$$Q = \frac{\tau_T^2 T_0}{4\pi^2}. \tag{3.46}$$

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 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 *non-ergodic*, 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 is 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, but in practice, just a few are required. The default is 10, but this can be controlled by user option. In the case of chains, the equations are modified in the following way to include a chain of thermostatting particles [28]:

$$\begin{aligned}
\frac{\mathrm{d}^2\boldsymbol{r}_i}{\mathrm{d}t^2} &= \frac{\boldsymbol{F}_i}{m_i} - \frac{p_{\xi_1}}{Q_1}\frac{\mathrm{d}\boldsymbol{r}_i}{\mathrm{d}t} \\
\frac{\mathrm{d}p_{\xi_1}}{\mathrm{d}t} &= \left(T - T_0\right) - p_{\xi_1}\frac{p_{\xi_2}}{Q_2} \\
\frac{\mathrm{d}p_{\xi_{i=2\ldots N}}}{\mathrm{d}t} &= \left(\frac{p_{\xi_{i-1}}^2}{Q_{i-1}} - kT\right) - p_{\xi_i}\frac{p_{\xi_{i+1}}}{Q_{i+1}} \\
\frac{\mathrm{d}p_{\xi_N}}{\mathrm{d}t} &= \left(\frac{p_{\xi_{N-1}}^2}{Q_{N-1}} - kT\right)
\end{aligned} \tag{3.47}$$

The conserved quantity for Nosè-Hoover chains is:

$$H = \sum_{i=1}^{N} \frac{\boldsymbol{p}_i}{2m_i} + U\left(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N\right) + \sum_{k=1}^{M} \frac{p^2_{\xi_k}}{2Q^{\prime}_k} + N_f kT\xi_1 + kT \sum_{k=2}^{M} \xi_k \tag{3.48}$$

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 this can be overridden by defining the environment variable GMX_NOSEHOOVER_CHAINS, which will print the values of all the positions and velocities of all Nosé-Hoover particles in the chain. Leapfrog simulations currently can only have Nosé-Hoover chain lengths of 1, but this will likely be updated in later version.

For temperature coupling, the reference temperature is calculated differently in velocity Verlet and leapfrog dynamics; velocity Verlet (*md-vv*) uses the full-step kinetic energy, while leapfrog and *md-vv-avek* use the half-step-averaged kinetic energy, discussed earlier in the integrator section.

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 at Ref. [29]), we split the Liouville operator as:

$$iL = iL_1 + iL_2 + iL_{\mathrm{NHC}} \tag{3.49}$$

where:

$$\begin{aligned} iL_1 &= \sum_{i=1}^{N} \left[\frac{\boldsymbol{p}_i}{m_i}\right] \cdot \frac{\partial}{\partial \boldsymbol{r}_i} \\ iL_2 &= \sum_{i=1}^{N} \boldsymbol{F}_i \cdot \frac{\partial}{\partial \boldsymbol{p}_i} \\ iL_{\mathrm{NHC}} &= \sum_{i=1}^{N} -\frac{p_\xi}{Q} \boldsymbol{v}_i \cdot \nabla_{\boldsymbol{v}_i} + \frac{p_\xi}{Q} \frac{\partial}{\partial \xi} + (T - T_0) \frac{\partial}{\partial p_\xi} \end{aligned} \tag{3.50}$$

For standard velocity Verlet with Nosé-Hoover temperature control, this becomes:

$$\begin{aligned} \exp(iL\Delta t) &= \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) \exp\left(iL_2 \Delta t/2\right) \\ &\quad \exp\left(iL_1 \Delta t\right) \exp\left(iL_2 \Delta t/2\right) \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) + \mathcal{O}(\Delta t^3) \end{aligned} \tag{3.51}$$

For half-step-averaged temperature control, 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.

$$\begin{aligned} \exp(iL\Delta t) &= \exp\left(iL_2 \Delta t/2\right) \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) \exp\left(iL_1 \Delta t\right) \\ &\quad \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) \exp\left(iL_2 \Delta t/2\right) + \mathcal{O}(\Delta t^3) \end{aligned} \tag{3.52}$$

This formalism allows us to easily see the difference between the different flavors of velocity Verlet integrator. The leapfrog integrator can be seen as starting with Eq. 3.52 just before the $\exp\left(iL_1 \Delta t\right)$ term, yielding:

$$\begin{aligned} \exp(iL\Delta t) &= \exp\left(iL_1 \Delta t\right) \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) \\ &\quad \exp\left(iL_2 \Delta t\right) \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) + \mathcal{O}(\Delta t^3) \end{aligned} \tag{3.53}$$

and then using some algebra tricks to solve for some quantities are required before they are actually calculated. [30]

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` 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. [31]

One special case should be mentioned: it is possible to T-couple only part of the system, leaving other parts without temperature coupling. This is done by specifying zero for the time constant τ_T for the group of which 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. [31]

3.4.9 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 [23] that scales coordinates and box vectors every step, and the extended ensemble Parrinello-Rahman approach. Both of these can be combined with any of the temperature coupling methods above.

Berendsen pressure coupling

The Berendsen algorithm rescales the coordinates and box vectors every step with a matrix $\boldsymbol{\mu}$, which has the effect of a first-order kinetic relaxation of the pressure towards a given reference pressure $\mathbf{P}_0$:

$$\frac{\mathrm{d}\mathbf{P}}{\mathrm{d}t} = \frac{\mathbf{P}_0 - \mathbf{P}}{\tau_p} \tag{3.54}$$

The scaling matrix $\boldsymbol{\mu}$ is given by

$$\mu_{ij} = \delta_{ij} - \frac{\Delta t}{3\,\tau_p}\beta_{ij}\{P_{0ij} - P_{ij}(t)\} \tag{3.55}$$

Here $\boldsymbol{\beta}$ is the isothermal compressibility of the system. In most cases this will be a diagonal matrix, with equal elements on the diagonal, the value of which is generally not known. It suffices to take a rough estimate because the value of $\boldsymbol{\beta}$ only influences the non-critical time constant of the pressure relaxation without affecting the average pressure itself. For water at 1 atm and 300 K $\beta = 4.6 \times 10^{-10}$ Pa^{-1} $= 4.6 \times 10^{-5}$ Bar^{-1}, which is 7.6×10^{-4} MD units (see chapter 2). Most other liquids have similar values. When scaling completely anisotropically, the system has to be rotated in order to obey eqn. 3.1. This rotation is approximated in first order in the scaling, which is usually less than 10^{-4}. The actual scaling matrix $\boldsymbol{\mu'}$ is:

$$\boldsymbol{\mu'} = \begin{pmatrix} \mu_{xx} & \mu_{xy} + \mu_{yx} & \mu_{xz} + \mu_{zx} \\ 0 & \mu_{yy} & \mu_{yz} + \mu_{zy} \\ 0 & 0 & \mu_{zz} \end{pmatrix} \tag{3.56}$$

The velocities are neither scaled nor rotated.

In GROMACS, the Berendsen scaling can also be done isotropically, which means that instead of $\boldsymbol{P}$ a diagonal matrix with elements of size trace$(\boldsymbol{P})/3$ 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 errors this approximation may yield.

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[32, 33], 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 $\boldsymbol{b}$ obey the matrix equation of motion[2]

$$\frac{\mathrm{d}\boldsymbol{b}^2}{\mathrm{d}t^2} = V\boldsymbol{W}^{-1}\boldsymbol{b}'^{-1}\left(\boldsymbol{P} - \boldsymbol{P}_{ref}\right). \tag{3.57}$$

The volume of the box is denoted V, and $\boldsymbol{W}$ is a matrix parameter that determines the strength of the coupling. The matrices $\boldsymbol{P}$ and $\boldsymbol{P}_{ref}$ 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:

---

[2] The box matrix representation $\boldsymbol{b}$ in GROMACS corresponds to the transpose of the box matrix representation $\boldsymbol{h}$ in the paper by Nosé and Klein. Because of this, some of our equations will look slightly different.

$$\frac{\mathrm{d}^2\boldsymbol{r}_i}{\mathrm{d}t^2} = \frac{\boldsymbol{F}_i}{m_i} - \boldsymbol{M}\frac{\mathrm{d}\boldsymbol{r}_i}{\mathrm{d}t}, \tag{3.58}$$

$$\boldsymbol{M} = \boldsymbol{b}^{-1}\left[\boldsymbol{b}\frac{\mathrm{d}\boldsymbol{b}'}{\mathrm{d}t} + \frac{\mathrm{d}\boldsymbol{b}}{\mathrm{d}t}\boldsymbol{b}'\right]\boldsymbol{b}'^{-1}. \tag{3.59}$$

The (inverse) mass parameter matrix $\boldsymbol{W}^{-1}$ determines the strength of the coupling, and how the box can be deformed. The box restriction (3.1) will be fulfilled automatically if the corresponding elements of $\boldsymbol{W}^{-1}$ 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 $\boldsymbol{\beta}$ and the pressure time constant τ_p in the input file (L is the largest box matrix element):

$$\left(\boldsymbol{W}^{-1}\right)_{ij} = \frac{4\pi^2\beta_{ij}}{3\tau_p^2 L}. \tag{3.60}$$

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 leapfrog 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.

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:

$$\gamma(t) = \frac{1}{n}\int_0^{L_z}\left\{P_{zz}(z,t) - \frac{P_{xx}(z,t)+P_{yy}(z,t)}{2}\right\}\mathrm{d}z \tag{3.61}$$

$$= \frac{L_z}{n}\left\{P_{zz}(t) - \frac{P_{xx}(t)+P_{yy}(t)}{2}\right\} \tag{3.62}$$

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_z:

$$\Delta P_{zz} = \frac{\Delta t}{\tau_p}\{P_{0zz} - P_{zz}(t)\} \tag{3.63}$$

$$\mu_{zz} = 1 + \beta_{zz}\Delta P_{zz} \tag{3.64}$$

This is similar to normal pressure coupling, except that the power of one third 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:

$$\mu_{x/y} = 1 + \frac{\Delta t}{2\,\tau_p}\beta_{x/y}\left(\frac{n\gamma_0}{\mu_{zz}L_z} - \left\{P_{zz}(t) + \Delta P_{zz} - \frac{P_{xx}(t) + P_{yy}(t)}{2}\right\}\right) \tag{3.65}$$

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_z 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 leapfrog 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 leapfrog, 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.* [29] and Tuckerman [34] 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 [34])

The isobaric equations are then:

$$\begin{aligned}
\dot{\boldsymbol{r}}_i &= \frac{\boldsymbol{p}_i}{m_i} + \frac{p_\epsilon}{W}\boldsymbol{r}_i \\
\frac{\dot{\boldsymbol{p}}_i}{m_i} &= \frac{1}{m_i}\boldsymbol{F}_i - \alpha\frac{p_\epsilon}{W}\frac{\boldsymbol{p}_i}{m_i} \\
\dot{\epsilon} &= \frac{p_\epsilon}{W} \\
\frac{\dot{p}_\epsilon}{W} &= \frac{3V}{W}(P_{\text{int}} - P) + (\alpha - 1)\left(\sum_{n=1}^{N}\frac{\boldsymbol{p}_i^2}{m_i}\right) \qquad (3.66) \\
& \qquad (3.67)
\end{aligned}$$

where:

$$P_{\text{int}} = P_{\text{kin}} - P_{\text{vir}} = \frac{1}{3V}\left[\sum_{i=1}^{N}\left(\frac{\boldsymbol{p}_i^2}{2m_i} - \boldsymbol{r}_i\cdot\boldsymbol{F}_i\right)\right] \tag{3.68}$$

The terms including α are required to make phase space incompressible [34]. The ϵ acceleration term can be rewritten as:

$$\frac{\dot{p}_\epsilon}{W} = \frac{3V}{W}(\alpha P_{\text{kin}} - P_{\text{vir}} - P) \tag{3.69}$$

In terms of velocities, these equations become:

$$\begin{aligned}
\dot{\boldsymbol{r}}_i &= \boldsymbol{v}_i + v_\epsilon \boldsymbol{r}_i \\
\dot{\boldsymbol{v}}_i &= \frac{1}{m_i}\boldsymbol{F}_i - \alpha v_\epsilon \boldsymbol{v}_i \\
\dot{\epsilon} &= v_\epsilon \\
\dot{v_\epsilon} &= \frac{3V}{W}(P_{\text{int}} - P) + (\alpha - 1)\left(\sum_{n=1}^{N} \frac{1}{2} m_i \boldsymbol{v}_i^2\right) \\
P_{\text{int}} &= P_{\text{kin}} - P_{\text{vir}} = \frac{1}{3V}\left[\sum_{i=1}^{N}\left(\frac{1}{2} m_i \boldsymbol{v}_i^2 - \boldsymbol{r}_i \cdot \boldsymbol{F}_i\right)\right]
\end{aligned} \tag{3.70}$$

For these equations, the conserved quantity is:

$$H = \sum_{i=1}^{N} \frac{\boldsymbol{p}_i^2}{2m_i} + U(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N) + \frac{p_\epsilon}{2W} + PV \tag{3.71}$$

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:

$$\begin{aligned}
\dot{\boldsymbol{r}}_i &= \frac{\boldsymbol{p}_i}{m_i} + \frac{p_\epsilon}{W}\boldsymbol{r}_i \\
\frac{\dot{\boldsymbol{p}}_i}{m_i} &= \frac{1}{m_i}\boldsymbol{F}_i - \alpha \frac{p_\epsilon}{W}\frac{\boldsymbol{p}_i}{m_i} - \frac{p_{\xi_1}}{Q_1}\frac{\boldsymbol{p}_i}{m_i} \\
\dot{\epsilon} &= \frac{p_\epsilon}{W} \\
\frac{\dot{p}_\epsilon}{W} &= \frac{3V}{W}(\alpha P_{\text{kin}} - P_{\text{vir}} - P) - \frac{p_{\eta_1}}{Q'_1} p_\epsilon \\
\dot{\xi}_k &= \frac{p_{\xi_k}}{Q_k} \\
\dot{\eta}_k &= \frac{p_{\eta_k}}{Q'_k} \\
\dot{p}_{\xi_k} &= G_k - \frac{p_{\xi_{k+1}}}{Q_{k+1}} \qquad k = 1, \ldots, M-1 \\
\dot{p}_{\eta_k} &= G'_k - \frac{p_{\eta_{k+1}}}{Q'_{k+1}} \qquad k = 1, \ldots, M-1 \\
\dot{p}_{\xi_M} &= G_M \\
\dot{p}_{\eta_M} &= G'_M
\end{aligned} \tag{3.72}$$

Where:

$$\begin{aligned}
P_{\text{int}} &= P_{\text{kin}} - P_{\text{vir}} = \frac{1}{3V}\left[\sum_{i=1}^{N}\left(\frac{\boldsymbol{p}_i^2}{2m_i} - \boldsymbol{r}_i \cdot \boldsymbol{F}_i\right)\right] \\
G_1 &= \sum_{i=1}^{N} \frac{\boldsymbol{p}_i^2}{m_i} - N_f kT
\end{aligned}$$

$$
\begin{aligned}
G_k &= \frac{p_{\xi_{k-1}}^2}{2Q_{k-1}} - kT \quad k = 2, \ldots, M \\
G'_1 &= \frac{p_\epsilon^2}{2W} - kT \\
G'_k &= \frac{p_{\eta_{k-1}}^2}{2Q'_{k-1}} - kT \quad k = 2, \ldots, M
\end{aligned}
\tag{3.73}
$$

The conserved quantity is now:

$$
\begin{gathered}
H = \sum_{i=1}^{N} \frac{\boldsymbol{p}_i}{2m_i} + U\left(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_N\right) + \frac{p_\epsilon^2}{2W} + PV + \\
\sum_{k=1}^{M} \frac{p_{\xi_k}^2}{2Q_k} + \sum_{k=1}^{M} \frac{p_{\eta_k}^2}{2Q'_k} + N_f kT \xi_1 + kT \sum_{i=2}^{M} \xi_k + kT \sum_{k=1}^{M} \eta_k
\end{gathered}
\tag{3.74}
$$

Returning to the Trotter decomposition formalism, for pressure control and temperature control we get: [29]

$$
iL = iL_1 + iL_2 + iL_{\epsilon,1} + iL_{\epsilon,2} + iL_{\text{NHC}-\text{baro}} + iL_{\text{NHC}} \tag{3.75}
$$

where NHC-baro correspond to the Nose-Hoover chain of the barostat, and NHC corresponds to the NHC of the particles.

$$
iL_1 = \sum_{i=1}^{N} \left[\frac{\boldsymbol{p}_i}{m_i} + \frac{p_\epsilon}{W} \boldsymbol{r}_i \right] \cdot \frac{\partial}{\partial \boldsymbol{r}_i} \tag{3.76}
$$

$$
iL_2 = \sum_{i=1}^{N} \boldsymbol{F}_i - \alpha \frac{p_\epsilon}{W} \boldsymbol{p}_i \cdot \frac{\partial}{\partial \boldsymbol{p}_i} \tag{3.77}
$$

$$
iL_{\epsilon,1} = \frac{p_\epsilon}{W} \frac{\partial}{\partial \epsilon} \tag{3.78}
$$

$$
iL_{\epsilon,2} = G_\epsilon \frac{\partial}{\partial p_\epsilon} \tag{3.79}
$$

and where

$$
G_\epsilon = 3V \left(\alpha P_{\text{kin}} - P_{\text{vir}} - P \right) \tag{3.80}
$$

Using the Trotter decomposition, we get:

$$
\begin{aligned}
\exp(iL\Delta t) = {} & \exp\left(iL_{\text{NHC}-\text{baro}} \Delta t/2\right) \exp\left(iL_{\text{NHC}} \Delta t/2\right) \\
& \exp\left(iL_{\epsilon,2} \Delta t/2\right) \exp\left(iL_2 \Delta t/2\right) \\
& \exp\left(iL_{\epsilon,1} \Delta t\right) \exp\left(iL_1 \Delta t\right) \\
& \exp\left(iL_2 \Delta t/2\right) \exp\left(iL_{\epsilon,2} \Delta t/2\right) \\
& \exp\left(iL_{\text{NHC}} \Delta t/2\right) \exp\left(iL_{\text{NHC}-\text{baro}} \Delta t/2\right) + \mathcal{O}(\Delta t^3)
\end{aligned}
\tag{3.81}
$$

The action of $\exp\left(iL_1 \Delta t\right)$ comes from the solution of the the differential equation $\dot{\boldsymbol{r}}_i = \boldsymbol{v}_i + v_\epsilon \boldsymbol{r}_i$ with $\boldsymbol{v}_i = \boldsymbol{p}_i / m_i$ and v_ϵ constant with initial condition $\boldsymbol{r}_i(0)$, evaluate at $t = \Delta t$. This yields the evolution:

$$
\boldsymbol{r}_i(\Delta t) = \boldsymbol{r}_i(0) e^{v_\epsilon \Delta t} + \Delta t \boldsymbol{v}_i(0) e^{v_\epsilon \Delta t/2} \frac{\sinh\left(v_\epsilon \Delta t/2\right)}{v_\epsilon \Delta t/2} \tag{3.82}
$$

The action of $\exp((iL_2\Delta t/2)$ comes from the solution of the differential equation $\dot{\boldsymbol{v}}_i = \frac{\boldsymbol{F}_i}{m_i} - \alpha v_\epsilon \boldsymbol{v}_i$, yielding:

$$\boldsymbol{v}_i(\Delta t/2) = \boldsymbol{v}_i(0)e^{-\alpha v_\epsilon \Delta t/2} + \frac{\Delta t}{2m_i}\boldsymbol{F}_i(0)e^{-\alpha v_\epsilon \Delta t/4}\frac{\sinh\left(\alpha v_\epsilon \Delta t/4\right)}{\alpha v_\epsilon \Delta t/4} \tag{3.83}$$

The *md-vv-avek* uses the full step kinetic energies for determining the pressure with the pressure control, but the half-step-averaged kinetic energy for the temperatures, which can be written as a Trotter decomposition as:

$$\begin{aligned}\exp(iL\Delta t) \;=\; & \exp\left(iL_{\mathrm{NHC-baro}}\Delta t/2\right)\exp\left(iL_{\epsilon,2}\Delta t/2\right)\exp\left(iL_2\Delta t/2\right) \\ & \exp\left(iL_{\mathrm{NHC}}\Delta t/2\right)\exp\left(iL_{\epsilon,1}\Delta t\right)\exp\left(iL_1\Delta t\right)\exp\left(iL_{\mathrm{NHC}}\Delta t/2\right) \\ & \exp\left(iL_2\Delta t/2\right)\exp\left(iL_{\epsilon,2}\Delta t/2\right)\exp\left(iL_{\mathrm{NHC-baro}}\Delta t/2\right) + \mathcal{O}(\Delta t^3)\end{aligned} \tag{3.84}$$

With constraints, the equations becomes significantly more complicated, in that each of these equations need to be solved iteratively for the constraint forces. The discussion of the details of the iteration is beyond the scope of this manual; readers are encouraged to see the implementation described in Ref. [35].

3.4.10 The complete update algorithm

The complete algorithm for the update of velocities and coordinates is given using leapfrog in Fig. 3.8. 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* $\boldsymbol{f}_g$, which is a vector, and differs for each *freezegroup* (see sec. 3.3). This vector contains only zero (freeze) or one (don't freeze). When we take this freeze factor and the external acceleration $\boldsymbol{a}_h$ into account the update algorithm for the velocities becomes:

$$\boldsymbol{v}(t+\frac{\Delta t}{2}) \;=\; \boldsymbol{f}_g * \lambda * \left[\boldsymbol{v}(t-\frac{\Delta t}{2}) + \frac{\boldsymbol{F}(t)}{m}\Delta t + \boldsymbol{a}_h\Delta t\right] \tag{3.85}$$

where g and h are group indices which differ per atom.

3.4.11 Output step

The important output of the MD run is the *trajectory file* `name.trj` which contains particle coordinates and -optionally- velocities at regular intervals. 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.

THE UPDATE ALGORITHM

Given:
Positions $\boldsymbol{r}$ of all atoms at time t
Velocities $\boldsymbol{v}$ of all atoms at time $t - \frac{1}{2}\Delta t$
Accelerations $\boldsymbol{F}/m$ 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 eqns. 3.40 and 3.55

$\Downarrow$

2. Update and scale velocities: $\boldsymbol{v}' = \lambda(\boldsymbol{v} + \boldsymbol{a}\Delta t)$

$\Downarrow$

3. Compute new unconstrained coordinates: $\boldsymbol{r}' = \boldsymbol{r} + \boldsymbol{v}'\Delta t$

$\Downarrow$

4. Apply constraint algorithm to coordinates: constrain($\boldsymbol{r}^{'} \rightarrow \boldsymbol{r}''$; $\boldsymbol{r}$)

$\Downarrow$

5. Correct velocities for constraints: $\boldsymbol{v} = (\boldsymbol{r}'' - \boldsymbol{r})/\Delta t$

$\Downarrow$

6. Scale coordinates and box: $\boldsymbol{r} = \mu\boldsymbol{r}''; \boldsymbol{b} = \mu\boldsymbol{b}$

Figure 3.8: The MD update algorithm with the leap frog integrator

3.5 Shell molecular dynamics

GROMACS can simulate polarizability using the shell model of Dick and Overhauser [36]. 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 [37] and water [38].

3.5.1 Optimization of the shell positions

The force $\boldsymbol{F}_S$ on a shell particle S can be decomposed into two components:

$$\boldsymbol{F}_S \;=\; \boldsymbol{F}_{bond} + \boldsymbol{F}_{nb} \tag{3.86}$$

where $\boldsymbol{F}_{bond}$ denotes the component representing the polarization energy, usually represented by a harmonic potential and $\boldsymbol{F}_{nb}$ is the sum of Coulomb and van der Waals interactions. If we assume that $\boldsymbol{F}_{nb}$ is almost constant we can analytically derive the optimal position of the shell, i.e. where $\boldsymbol{F}_S$ = 0. If we have the shell S connected to atom A we have

$$\boldsymbol{F}_{bond} \;=\; k_b \left(\boldsymbol{x}_S - \boldsymbol{x}_A\right) \tag{3.87}$$

In an iterative solver, we have positions $\boldsymbol{x}_S(n)$ where n is the iteration count. We now have it iteration n:

$$\boldsymbol{F}_{nb} \;=\; \boldsymbol{F}_S - k_b \left(\boldsymbol{x}_S(n) - \boldsymbol{x}_A\right) \tag{3.88}$$

and the optimal position for the shells $x_S(n+1)$ thus follows from

$$\boldsymbol{F}_S - k_b \left(\boldsymbol{x}_S(n) - \boldsymbol{x}_A\right) + k_b \left(\boldsymbol{x}_S(n+1) - \boldsymbol{x}_A\right) = 0 \tag{3.89}$$

if we write

$$\Delta\boldsymbol{x}_S = \boldsymbol{x}_S(n+1) - \boldsymbol{x}_S(n) \tag{3.90}$$

we finally obtain

$$\Delta\boldsymbol{x}_S = \boldsymbol{F}_S / k_b \tag{3.91}$$

which then yields the algorithm to compute the next trial in the optimization of shell positions:

$$\boldsymbol{x}_S(n+1) \;=\; \boldsymbol{x}_S(n) + \boldsymbol{F}_S / k_b \tag{3.92}$$

3.6 Constraint algorithms

Constraints can be imposed in GROMACS using LINCS (default) or the traditional SHAKE method.

3.6.1 SHAKE

The SHAKE [39] algorithm changes a set of unconstrained coordinates $\boldsymbol{r}^{'}$ to a set of coordinates $\boldsymbol{r}''$ that fulfill a list of distance constraints, using a set $\boldsymbol{r}$ as reference:

$$\text{SHAKE}(\boldsymbol{r}^{'} \rightarrow \boldsymbol{r}''; \, \boldsymbol{r})$$

This action is consistent with solving a set of Lagrange multipliers in the constrained equations of motion. SHAKE needs a *tolerance* `TOL`; it will continue until all constraints are satisfied within a *relative* tolerance `TOL`. 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

$$\sigma_k(\boldsymbol{r}_1 \ldots \boldsymbol{r}_N) = 0; \quad k = 1 \ldots K \tag{3.93}$$

(*e.g.* $(\boldsymbol{r}_1 - \boldsymbol{r}_2)^2 - b^2 = 0$). Then the forces are defined as

$$-\frac{\partial}{\partial \boldsymbol{r}_i}\left(V + \sum_{k=1}^{K} \lambda_k \sigma_k\right) \tag{3.94}$$

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* $\boldsymbol{G}_i$, defined by

$$\boldsymbol{G}_i = -\sum_{k=1}^{K} \lambda_k \frac{\partial \sigma_k}{\partial \boldsymbol{r}_i} \tag{3.95}$$

The displacement due to the constraint forces in the leap frog or Verlet algorithm is equal to $(\boldsymbol{G}_i/m_i)(\Delta t)^2$. 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. 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 [40] (sec. 5.5).

3.6.2 LINCS

The LINCS algorithm

LINCS is an algorithm that resets bonds to their correct lengths after an unconstrained update [41]. 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 3.17.3.

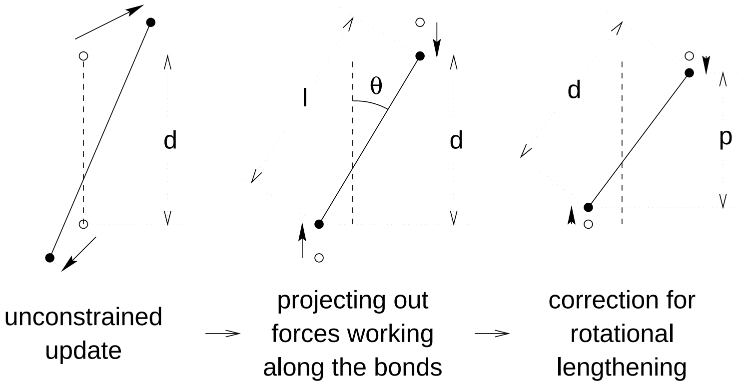


Figure 3.9: The three position updates needed for one time step. The dashed line is the old bond of length d, the solid lines are the new bonds. $l = d\cos\theta$ and $p = (2d^2 - l^2)^{\frac{1}{2}}$.

The LINCS formulas

We consider a system of N particles, with positions given by a $3N$ vector $\boldsymbol{r}(t)$. For molecular dynamics the equations of motion are given by Newton's law

$$\frac{\mathrm{d}^2\boldsymbol{r}}{\mathrm{d}t^2} = \boldsymbol{M}^{-1}\boldsymbol{F} \tag{3.96}$$

where $\boldsymbol{F}$ is the $3N$ force vector and $\boldsymbol{M}$ is a $3N \times 3N$ diagonal matrix, containing the masses of the particles. The system is constrained by K time-independent constraint equations

$$g_i(\boldsymbol{r}) = |\boldsymbol{r}_{i_1} - \boldsymbol{r}_{i_2}| - d_i = 0 \qquad i = 1,\ldots,K \tag{3.97}$$

In a numerical integration scheme LINCS is applied after an unconstrained update, just like SHAKE. The algorithm works in two steps (see figure Fig. 3.9). 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 [41]. Only a short description of the first step is given here.

A new notation is introduced for the gradient matrix of the constraint equations which appears on the right hand side of the equation

$$B_{hi} = \frac{\partial g_h}{\partial r_i} \tag{3.98}$$

Notice that $\boldsymbol{B}$ is a $K \times 3N$ matrix, it contains the directions of the constraints. The following equation shows how the new constrained coordinates $\boldsymbol{r}_{n+1}$ are related to the unconstrained coordinates $\boldsymbol{r}_{n+1}^{unc}$

$$\begin{gathered}\boldsymbol{r}_{n+1} = (\boldsymbol{I} - \boldsymbol{T}_n\boldsymbol{B}_n)\boldsymbol{r}_{n+1}^{unc} + \boldsymbol{T}_n\boldsymbol{d} = \\ \boldsymbol{r}_{n+1}^{unc} - \boldsymbol{M}^{-1}\boldsymbol{B}_n(\boldsymbol{B}_n\boldsymbol{M}^{-1}\boldsymbol{B}_n^T)^{-1}(\boldsymbol{B}_n\boldsymbol{r}_{n+1}^{unc} - \boldsymbol{d})\end{gathered} \tag{3.99}$$

where $\boldsymbol{T} = \boldsymbol{M}^{-1}\boldsymbol{B}^T(\boldsymbol{B}\boldsymbol{M}^{-1}\boldsymbol{B}^T)^{-1}$. The derivation of this equation from eqns. 3.96 and 3.97 can be found in [41].

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 on the old direction is set to

$$p_i = \sqrt{2d_i^2 - l_i^2} \tag{3.100}$$

where l_i is the bond length after the first projection. The corrected positions are

$$\boldsymbol{r}^*_{n+1} = (\boldsymbol{I} - \boldsymbol{T}_n\boldsymbol{B}_n)\boldsymbol{r}_{n+1} + \boldsymbol{T}_n\boldsymbol{p} \tag{3.101}$$

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 $\boldsymbol{B}_n\boldsymbol{M}^{-1}\boldsymbol{B}_n^T$, 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 $\boldsymbol{T}$ is inverted through a power expansion. A $K \times K$ matrix $\boldsymbol{S}$ is introduced which is the inverse square root of the diagonal of $\boldsymbol{B}_n\boldsymbol{M}^{-1}\boldsymbol{B}_n^T$. This matrix is used to convert the diagonal elements of the coupling matrix to one

$$\begin{aligned}(\boldsymbol{B}_n\boldsymbol{M}^{-1}\boldsymbol{B}_n^T)^{-1} &= \boldsymbol{S}\boldsymbol{S}^{-1}(\boldsymbol{B}_n\boldsymbol{M}^{-1}\boldsymbol{B}_n^T)^{-1}\boldsymbol{S}^{-1}\boldsymbol{S}\\ &= \boldsymbol{S}(\boldsymbol{S}\boldsymbol{B}_n\boldsymbol{M}^{-1}\boldsymbol{B}_n^T\boldsymbol{S})^{-1}\boldsymbol{S} = \boldsymbol{S}(\boldsymbol{I} - \boldsymbol{A}_n)^{-1}\boldsymbol{S}\end{aligned} \tag{3.102}$$

The matrix $\boldsymbol{A}_n$ is symmetric and sparse and has zeros on the diagonal. Thus a simple trick can be used to calculate the inverse

$$(\boldsymbol{I} - \boldsymbol{A}_n)^{-1} = \boldsymbol{I} + \boldsymbol{A}_n + \boldsymbol{A}_n^2 + \boldsymbol{A}_n^3 + \ldots \tag{3.103}$$

This inversion method is only valid if the absolute values of all the eigenvalues of $\boldsymbol{A}_n$ 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 eqn. 3.103 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:

$$(\boldsymbol{I} - \boldsymbol{A}_n)^{-1} \approx \boldsymbol{I} + \boldsymbol{A}_n + \ldots + \boldsymbol{A}_n^{N_i} + \left(\boldsymbol{A}^*_n + \ldots + \boldsymbol{A}_n^{*N_i}\right)\boldsymbol{A}_n^{N_i} \tag{3.104}$$

where N_i is the normal order of the expansion and $\boldsymbol{A}^*$ only contains the elements of $\boldsymbol{A}$ 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[42].

The LINCS Parameters

The accuracy of LINCS depends on the number of matrices used in the expansion eqn. 3.103. 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 input file for `mdrun`. 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 input file for `mdrun`.

3.7 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.

3.8 Stochastic Dynamics

Stochastic or velocity Langevin dynamics adds a friction and a noise term to Newton's equations of motion:

$$m_i \frac{\mathrm{d}^2 \boldsymbol{r}_i}{\mathrm{d}t^2} = -m_i \xi_i \frac{\mathrm{d}\boldsymbol{r}_i}{\mathrm{d}t} + \boldsymbol{F}_i(\boldsymbol{r}) + \overset{\circ}{\boldsymbol{r}}_i \tag{3.105}$$

where ξ_i is the friction constant $[1/\mathrm{ps}]$ and $\overset{\circ}{\boldsymbol{r}}_i(t)$ is a noise process with $\langle \overset{\circ}{r}_i(t)\; \overset{\circ}{r}_j(t+s)\rangle = 2m_i\xi_i k_B T\delta(s)\delta_{ij}$. When $1/\xi_i$ is large compared to the time scales present in the system, one could see stochastic dynamics as molecular dynamics with stochastic temperature-coupling. The advantage compared to MD with Berendsen temperature-coupling is that in case of SD the generated ensemble is known. 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 $1/\xi_i$ 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 are two algorithm to integrate equation (3.105). An efficient one, where the relative error in the temperature is $\frac{1}{2}\Delta t\,\xi$. And a more complex leap frog algorithm [43], which has third-order accuracy for any value of $\Delta t\,\xi$. In this complex algorithm four Gaussian random number are required per integration step per degree of freedom and with constraints the 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. Exact continuation of a stochastic dynamics simulation is not possible, because the state of the random number generator is not stored. When using SD as a thermostat, an appropriate value for ξ is 0.5 ps^{-1}, since this results in a friction that is lower than the internal friction of water, while it is high enough to remove excess heat (unless plain cut-off or reaction-field electrostatics is used). With this value of ξ the efficient algorithm will usually be accurate enough.

3.9 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:

$$\frac{\mathrm{d}\boldsymbol{r}_i}{\mathrm{d}t} = \frac{1}{\gamma_i}\boldsymbol{F}_i(\boldsymbol{r}) + \overset{\circ}{\boldsymbol{r}}_i \tag{3.106}$$

where γ_i is the friction coefficient [amu/ps] and $\overset{\circ}{\boldsymbol{r}}_i(t)$ is a noise process with $\langle \overset{\circ}{r}_i(t)\, \overset{\circ}{r}_j(t+s)\rangle = 2\delta(s)\delta_{ij}k_BT/\gamma_i$. In GROMACS the equations are integrated with a simple, explicit scheme:

$$\boldsymbol{r}_i(t+\Delta t) = \boldsymbol{r}_i(t) + \frac{\Delta t}{\gamma_i}\boldsymbol{F}_i(\boldsymbol{r}(t)) + \sqrt{2k_BT\frac{\Delta t}{\gamma_i}}\,\boldsymbol{r}_i^G \tag{3.107}$$

where $\boldsymbol{r}_i^G$ is Gaussian distributed noise with $\mu = 0$, $\sigma = 1$. The friction coefficients γ_i can be chosen the same for all particles or as $\gamma_i = m_i/\xi_i$, where the friction constants ξ_i can be different for different groups of atoms. Because the system is assumed to be over damped, large time-steps 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` program.

3.10 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` program.

3.10.1 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 $\boldsymbol{r}$ as the vector of all $3N$ coordinates. Initially a maximum displacement h_0 (*e.g.* 0.01 nm) must be given.

First the forces $\boldsymbol{F}$ and potential energy are calculated. New positions are calculated by

$$\boldsymbol{r}_{n+1} = \boldsymbol{r}_n + \frac{\boldsymbol{F}_n}{\max(|\boldsymbol{F}_n|)} h_n \tag{3.108}$$

where h_n is the maximum displacement and $\boldsymbol{F}_n$ is the force, or the negative gradient of the potential V. The notation $\max(|\boldsymbol{F}_n|)$ means the largest of the absolute values of the force components. 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

$$f = 2\pi\nu\sqrt{2mkT} \tag{3.109}$$

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.

3.10.2 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 [40], as this has not been implemented. If water is present it must be of a flexible model, which can be specified in the `mdp` 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 can not be performed with constraints. For most other purposes steepest descent is efficient enough.

3.10.3 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 [44, 45], 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 means the potential function at the current coordinates is slightly different from the previous steps used to build the inverse Hessian approximation.

3.11 Normal Mode Analysis

Normal mode analysis [46, 47, 48] can be performed using GROMACS, by diagonalization of the mass-weighted Hessian H:

$$R^T M^{-1/2} H M^{-1/2} R = \mathrm{diag}(\lambda_1, \ldots, \lambda_{3N}) \tag{3.110}$$

$$\lambda_i = (2\pi\omega_i)^2 \tag{3.111}$$

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:

$$H_{ij} = \frac{\partial^2 V}{\partial x_i \partial x_j} \tag{3.112}$$

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:

$$H_{ij} = -\frac{f_i(\mathbf{x} + h\mathbf{e}_j) - f_i(\mathbf{x} - h\mathbf{e}_j)}{2h} \tag{3.113}$$

$$f_i = -\frac{\partial V}{\partial x_i} \tag{3.114}$$

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. What tolerance is required depends on the type of system, but a rough indication is 0.001 kJ mol^{-1}. This 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`. Then `mdrun` 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. `g_nmeig` does the diagonalization and the sorting of the normal modes according to their frequencies. Both `mdrun` and `g_nmeig` should be run in double precision. The normal modes can be analyzed with the program `g_anaeig`. Ensembles of structures at any temperature and for any subset of normal modes can be generated with `g_nmens`. An overview of normal mode analysis and the related principal component analysis (see sec. 8.10) can be found in [49].

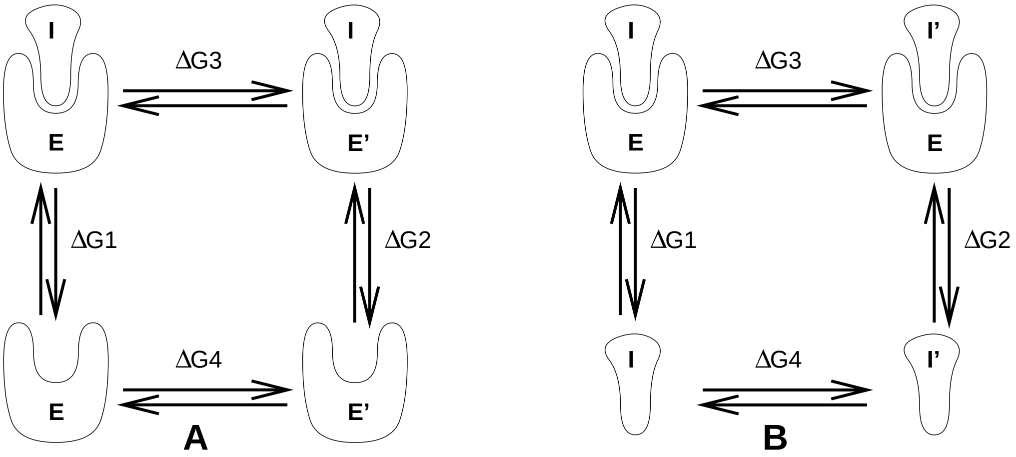


Figure 3.10: Free energy cycles. **A:** to calculate ΔG_{12}, the free energy difference between the binding of inhibitor **I** to enzymes **E** respectively **E'**. **B:** to calculate ΔG_{12}, the free energy difference for binding of inhibitors **I** respectively **I'** to enzyme **E**.

3.12 Free energy calculations

Free energy calculations can be performed in GROMACS using slow-growth methods. An example problem might be: calculate 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. 3.10A) we can write

$$\Delta G_1 - \Delta G_2 = \Delta G_3 - \Delta G_4 \tag{3.115}$$

If we are interested in the left-hand term we can equally well compute the right-hand term.

If we want to compute the difference in free energy of binding of two inhibitors **I** and **I'** to an enzyme **E** (Fig. 3.10B) we can again use eqn. 3.115 to compute the desired property.

Free energy differences between two molecular species can be calculated in GROMACS using the "slow-growth" method. In fact, such free energy differences between different molecular species are physically meaningless, but they can be used to obtain meaningful quantities employing a thermodynamic cycle. The method requires a simulation during which the Hamiltonian of the system changes slowly from that describing one system (A) to that describing the other system (B). The change must be so slow that the system remains in equilibrium during the process; if that requirement is fulfilled, the change is reversible and a slow-growth simulation from B to A will yield the same results (but with a different sign) as a slow-growth simulation from A to B. This is a useful check, but the user should be aware of the danger that equality of forward and backward growth results does not guarantee correctness of the results.

The required modification of the Hamiltonian H is realized by making H a function of a *coupling parameter* $\lambda : H = H(p, q; \lambda)$ in such a way that $\lambda = 0$ describes system A and $\lambda = 1$ describes system B:

$$H(p, q; 0) = H^{\mathrm{A}}(p, q); \quad H(p, q; 1) = H^{\mathrm{B}}(p, q). \tag{3.116}$$

In GROMACS, the functional form of the λ-dependence is different for the various force-field contributions and is described in section sec. 4.5.

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:

$$\begin{aligned}
A(\lambda) &= -k_B T \ln Q && (3.117)\\
Q &= c \iint \exp[-\beta H(p,q;\lambda)]\, dp\, dq && (3.118)\\
G(\lambda) &= -k_B T \ln \Delta && (3.119)\\
\Delta &= c \iiint \exp[-\beta H(p,q;\lambda) - \beta pV]\, dp\, dq\, dV && (3.120)\\
G &= A + pV, && (3.121)
\end{aligned}$$

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:

$$\frac{dA}{d\lambda} = \frac{\iint (\partial H/\partial\lambda) \exp[-\beta H(p,q;\lambda)]\, dp\, dq}{\iint \exp[-\beta H(p,q;\lambda)]\, dp\, dq} = \left\langle \frac{\partial H}{\partial \lambda} \right\rangle_{NVT;\lambda}, \tag{3.122}$$

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:

$$\begin{aligned}
A^{\mathrm{B}}(V,T) - A^{\mathrm{A}}(V,T) &= \int_0^1 \left\langle \frac{\partial H}{\partial \lambda} \right\rangle_{NVT;\lambda} d\lambda && (3.123)\\
G^{\mathrm{B}}(p,T) - G^{\mathrm{A}}(p,T) &= \int_0^1 \left\langle \frac{\partial H}{\partial \lambda} \right\rangle_{NpT;\lambda} d\lambda. && (3.124)
\end{aligned}$$

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:

$$G^{\mathrm{B}}(p) - G^{\mathrm{A}}(p) = A^{\mathrm{B}}(V) - A^{\mathrm{A}}(V) - \int_p^{p^{\mathrm{B}}} [V^{\mathrm{B}}(p') - V]\, dp' \tag{3.125}$$

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.

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.

GROMACS offers the possibility to integrate eq. 3.123 or eq. 3.124 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.

The λ-dependence for the force-field contributions is described in detail in section sec. 4.5.

3.13 Replica exchange

Replica exchange molecular dynamics (REMD) is a method which 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:

$$P(1 \leftrightarrow 2) = \min\left(1, \exp\left[\left(\frac{1}{k_B T_1} - \frac{1}{k_B T_2}\right)(U_1 - U_2)\right]\right) \tag{3.126}$$

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 [50, 51]. 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:

$$U_1 - U_2 = N_{df} \frac{c}{2} k_B (T_1 - T_2) \tag{3.127}$$

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:

$$P(1 \leftrightarrow 2) = \exp\left(-\frac{\epsilon^2 c\, N_{df}}{2(1+\epsilon)}\right) \approx \exp\left(-\epsilon^2 \frac{c}{2} N_{df}\right) \tag{3.128}$$

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[52] and this 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.* [53]. In this work the exchange probability is modified to:

$$P(1 \leftrightarrow 2) = \min\left(1, \exp\left[\left(\frac{1}{k_B T_1} - \frac{1}{k_B T_2}\right)(U_1 - U_2) + \left(\frac{P_1}{k_B T_1} - \frac{P_2}{k_B T_2}\right)(V_1 - V_2)\right]\right) \tag{3.129}$$

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.

Replica exchange is an option of the `mdrun` 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 node. See the manual page of `mdrun` on how to use it.

3.14 Essential Dynamics Sampling

The results from Essential Dynamics (see sec. 8.10) 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 [54, 55]. 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 [56].

The procedure for essential dynamics sampling or flooding is as follows. First the eigenvectors and eigenvalues need to be determined using covariance analysis (`g_covar`) or normal modes analysis (`g_nmeig`). This information is fed into `make_edi` which has many options for selecting vectors and setting parameters, see Appendix D for the manual page of `make_edi`. The generated `edi` input file is then passed to `mdrun`.

3.15 Parallelization

The CPU time required for a simulation can be reduced by running the simulation in parallel over more than one processor or processor core. Ideally one would want to have linear scaling: running on N processors/cores makes the simulation N times faster. In practice this can only be achieved for a small number of processors. The scaling will depend a lot on the algorithms used. Also different algorithms can have different restrictions on the interaction ranges between atoms. In GROMACS we have two types of parallelization: particle decomposition and domain decomposition. Particle decomposition is only useful for a few special cases. Domain decomposition, which is the default algorithm, will always be faster and scale better.

3.16 Particle decomposition

Particle decomposition, also called force decomposition, is the simplest type of decomposition. Here at the start of the simulation particles are assigned to processors. Then forces between particles need to be assigned to processors such that the force load is evenly balanced. This decomposition requires that each processor knows the coordinates of at least half of the particles in the system. Thus for a high number of processors N, about $N \times N/2$ coordinates need to be communicated. Because of this quadratic relation particle decomposition does not scale well.

Particle decomposition was the only method available before version 4 of GROMACS. Now it is only useful in cases where domain decomposition does not work. This is for systems with long-range bonded interactions, especially NMR distance or orientation restraints. With particle decomposition only whole molecules can be assigned to a processor.

3.17 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 processor. Each processor will 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 processors. Most molecular simulation packages use the half-shell method for assigning the forces. But there are two methods which always require less communication: the eighth shell[57] and the midpoint[58] 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[59].

3.17.1 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 which we call domain decomposition cells. Each cell is assigned to a processor. The system is partitioned over the processors at the beginning of each MD step where neighbor searching is performed. Since the neighbor searching is based on charge groups, charge groups are also the units for the domain decomposition. Charge 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 which can cover one or multiple cells. An example of a zone setup is shown in Fig. 3.11.

The coordinates are communicated by moving data along the "negative" direction in x, y or z to the next neighbor. This can be done in one or multiple pulses. In Fig. 3.11 two pulses in x

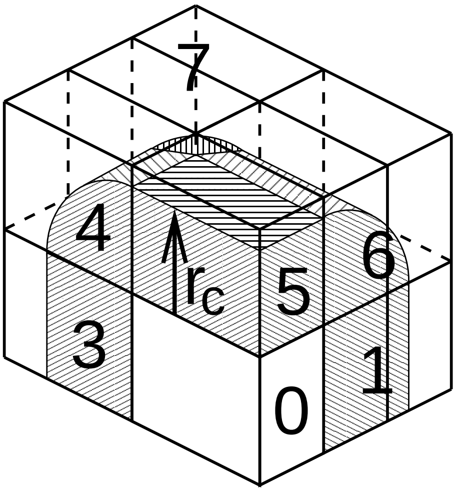


Figure 3.11: A non-staggered domain decomposition grid of 3×2×2 cells. Coordinates in zones 1 to 7 are communicated to the corner cell that has its home particles in zone 0. r_c is the cut-off radius.

are required, then one in y and then one in z. The forces are communicated by reversing this procedure. See the GROMACS 4 paper[59] for details on determining which non-bonded and bonded forces should be calculated on which node.

3.17.2 Dynamic load balancing

When different processors have a different computational load (load imbalance), all processors 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 three 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)

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. 3.12 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.

By default `mdrun` automatically turns on the dynamic load balancing during a simulation when the total performance loss due to the force calculation imbalance is 5% 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.

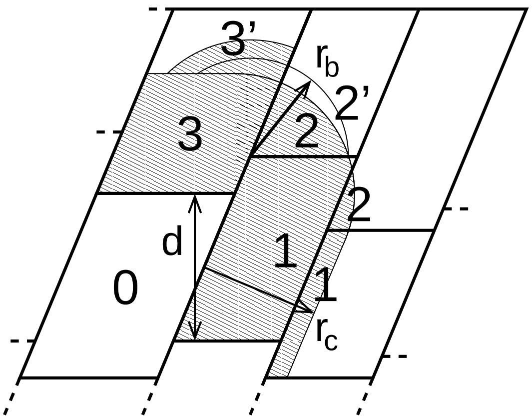


Figure 3.12: The zones to communicate to the processor of zone 0, see the text for details. r_c and r_b are the non-bonded and bonded cut-off radii respectively, d is an example of a distance between following, staggered boundaries of cells.

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 required scaling can be changed with the `-dds` option of `mdrun`.

3.17.3 Constraints in parallel

Since with domain decomposition parts of molecules can reside on different processors, bond constraints can cross cell boundaries. Therefore a parallel constraint algorithm is required. GROMACS uses the P-LINCS algorithm[42], which is the parallel version of the LINCS algorithm[41] (see 3.6.2). The P-LINCS procedure is illustrated in Fig. 3.13. When molecules cross the cell boundaries, atoms in such molecules up to LINCS order plus one 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.

3.17.4 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` 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 3.2.

In most cases the defaults of `mdrun` 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, 10% is added for headroom. For the

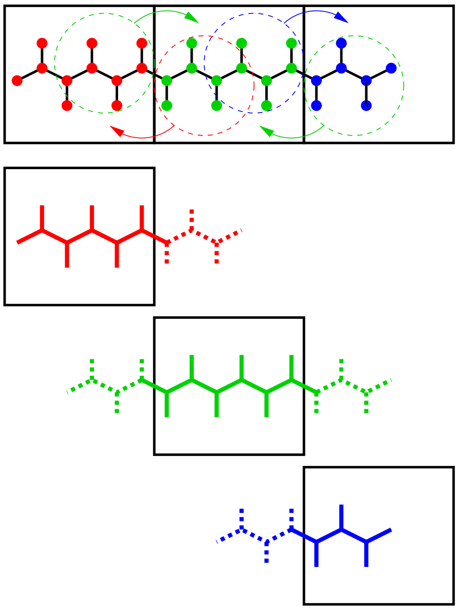


Figure 3.13: Example of the parallel setup of P-LINCS with one molecule split over three domain decomposition cells, using a matrix expansion order of 3. The top part shows which atom coordinates need to be communicated to which cells. The bottom parts show the local constraints (solid) and the non-local constraints (dashed) for each of the three cells.

interaction	range	option	default
non-bonded	r_c = max(r_{list},r_{VdW},r_{Coul})	mdp file	
two-body bonded	max(r_{mb},r_c)	`mdrun -rdd`	starting conf. + 10%
multi-body bonded	r_{mb}	`mdrun -rdd`	starting conf. + 10%
constraints	r_{con}	`mdrun -rcon`	est. from bond lengths
virtual sites	r_{con}	`mdrun -rcon`	0

Table 3.2: The interaction ranges with domain decomposition.

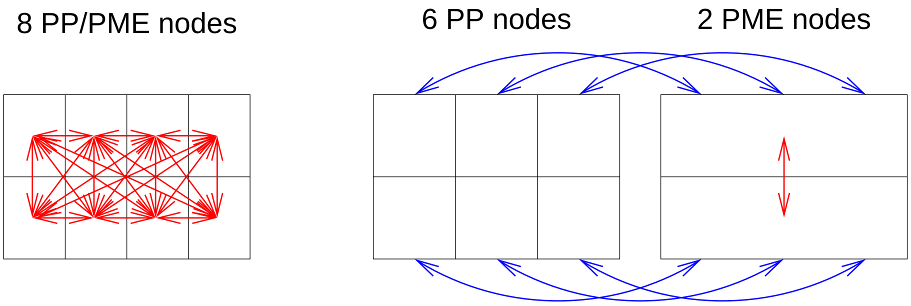


Figure 3.14: Example of 8 nodes without (left) and with (right) MPMD. The PME communication (red arrows) is much higher on the left than on the right. For MPMD additional PP - PME coordinate and force communication (blue arrows) is required, but the total communication complexity is lower.

constraints the r_{con} is determined by taking the maximum distance that LINCS order plus one 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:

$$L_C \geq \max(r_{mb}, r_{con}) \tag{3.130}$$

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, but 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 $r_{mb} > r_c$, `mdrun` 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.

3.17.5 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. 4.9.2). Since with PME all particles interact with each other, global communication is required. This will usually be the limiting factor on the scaling with domain decomposition. To reduce the effect of this problem, we have come up with a Multiple-Program, Multiple-Data approach[59]. Here some processors are selected to do only the PME mesh calculation, while the other processors, called particle-particle (PP) nodes, do all the rest of the work. For rectangular boxes the optimal PP to PME node ratio is usually 3:1, for rhombic dodecahedra usually 2:1. When the number of PME nodes 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 nodes. 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` will by default interleave the PP and PME nodes. If the processors are not number consecutively inside the machines, one might want to use `mdrun -ddorder pp_pme`. For machines with a real 3-D torus and proper communication software that assigns the processors accordingly one should use `mdrun -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` will print an estimate for this load at the end and also `mdrun` calculates the same estimate to determine the optimal number of PME nodes to use. For high parallelization it might be worth to optimize the PME load with the `mdp` settings and/or the number of PME nodes with the `-npme` option of `mdrun`. 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 nodes, since the number of PME nodes is smaller than the number of PP nodes, which leads to less total waiting time.

Currently the PME domain decomposition is 1-D along the x axis. To avoid superfluous communication of coordinates and forces between the PP and PME nodes, the number of DD cells in the x direction should ideally be the same or a multiple of the number of PME nodes. By default `mdrun` takes care of this issue. In the future we will support better parallelizable electrostatics implementations.

3.17.6 Domain decomposition flow chart

In Fig. 3.15 a flow chart is shown for domain decomposition with all possible communication for different algorithms. For simpler simulations the same flow chart applies, but simply without the algorithms and communication for the algorithms which are not used.

3.18 Implicit solvent

Implicit solvent models provide an efficient way of representing the electrostatic effects of solvent molecules, while saving a large piece of the computations involved in an accurate, aqueous description of the surrounding water in molecular dynamics simulations. Implicit solvation models offer several advantages compared with explicit solvation, including eliminating the need for the equilibration of water around the solute, and the absence of viscosity, which allows the protein to more quickly explore conformational space.

Implicit solvent calculations in GROMACS can be done using the generalized Born-formalism, and the Still [60], HCT [61], and OBC [62] models are available for calculating the Born radii.

Here, the free energy G_{solv} of solvation is the sum of three terms, a solvent-solvent cavity term (G_{cav}), a solute-solvent van der Waals term (G_{vdw}), and finally a solvent-solute electrostatics polarization term (G_{pol}).

The sum of G_{cav} and G_{vdw} corresponds to the (non-polar) free energy of solvation for a molecule from which all charges have been removed, and is commonly called G_{np}, and calculated from the total solvent accessible surface area multiplied with a surface tension. The total expression for the solvation free energy then becomes:

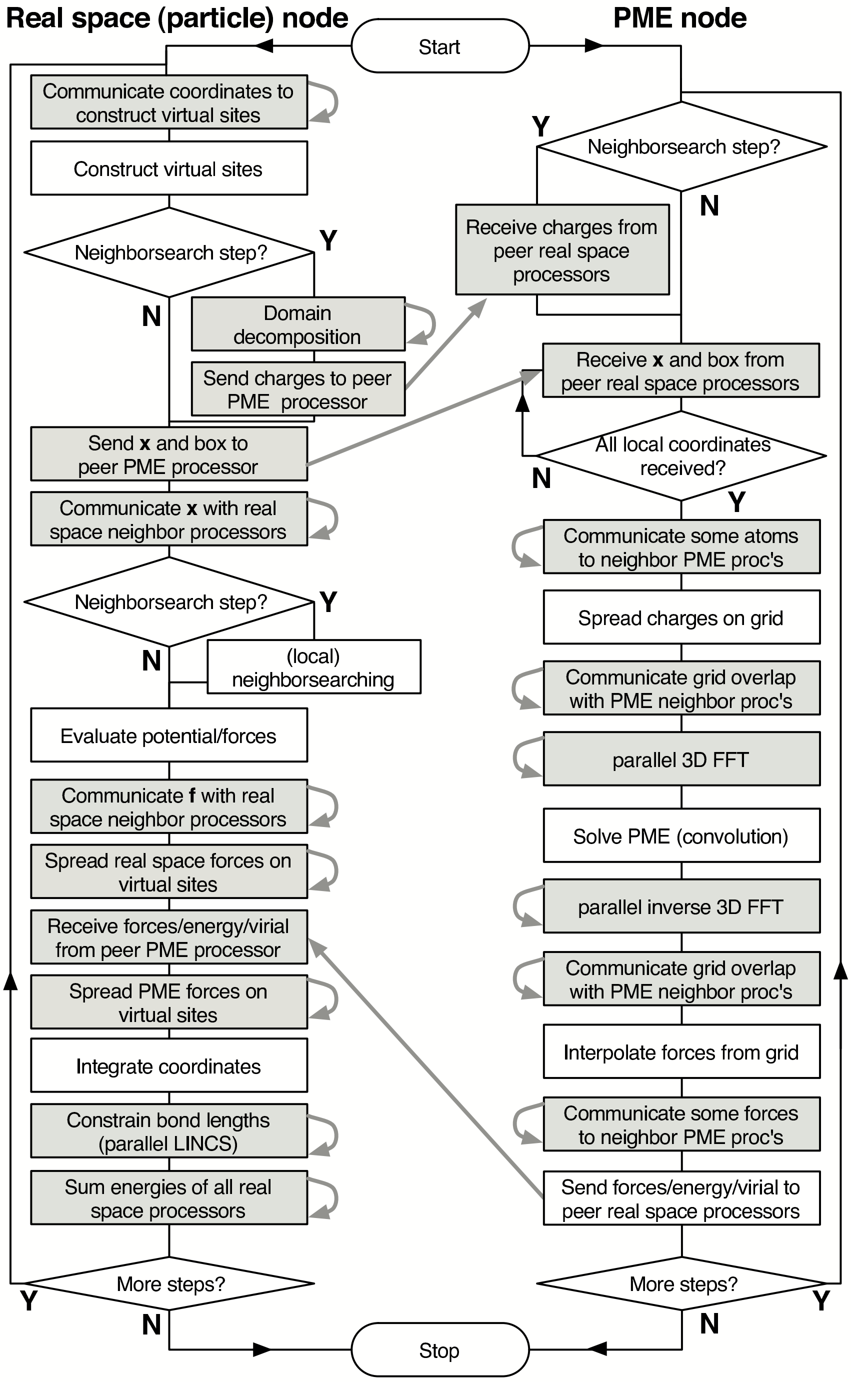


Figure 3.15: Flow chart showing the algorithms and communication (arrows) for a standard MD simulation with virtual sites, constraints and separate PME-mesh nodes.

$$G_{solv} = G_{np} + G_{pol} \tag{3.131}$$

Under the generalized Born model, G_{pol} is calculated from the generalized Born equation [60]

$$G_{pol} = \left(1 - \frac{1}{\epsilon}\right) \sum_{i=1}^{n} \sum_{j>i}^{n} \frac{q_i q_j}{\sqrt{r_{ij}^2 + b_i b_j \exp\left(\frac{-r_{ij}^2}{4 b_i b_j}\right)}} \tag{3.132}$$

In GROMACS we have introduced the substitution [63]

$$c_i = \frac{1}{\sqrt{b_i}} \tag{3.133}$$

which makes it possible to introduce a cheap transformation to a new variable x when evaluating each interaction, such that

$$x = \frac{r_{ij}}{\sqrt{b_i b_j}} = r_{ij} c_i c_j \tag{3.134}$$

In the end, the full re-formulation of 3.132 becomes:

$$G_{pol} = \left(1 - \frac{1}{\epsilon}\right) \sum_{i=1}^{n} \sum_{j>i}^{n} \frac{q_i q_j}{\sqrt{b_i b_j}}\, \xi(x) = \left(1 - \frac{1}{\epsilon}\right) \sum_{i=1}^{n} q_i c_i \sum_{j>i}^{n} q_j c_j\, \xi(x) \tag{3.135}$$

The non-polar part (G_{np}) of Equation 3.131 is calculated directly from the Born radius of each atom using a simple ACE type approximation by Schaefer et al [64], including a simple loop over all atoms. This requires only one extra solvation parameter, independent of atom type, but differing slightly between the three Born radii models.

Chapter 4

Interaction function and force field

To accommodate the potential functions used in some popular force fields (see 4.10), 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.

4.1 Non-bonded interactions

Non-bonded interactions in GROMACS are pair-additive and centro-symmetric:

$$V(\boldsymbol{r}_1, \ldots \boldsymbol{r}_N) = \sum_{i<j} V_{ij}(\boldsymbol{r}_{ij}); \tag{4.1}$$

$$\boldsymbol{F}_i = -\sum_j \frac{dV_{ij}(r_{ij})}{dr_{ij}} \frac{\boldsymbol{r}_{ij}}{r_{ij}} = -\boldsymbol{F}_j \tag{4.2}$$

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.

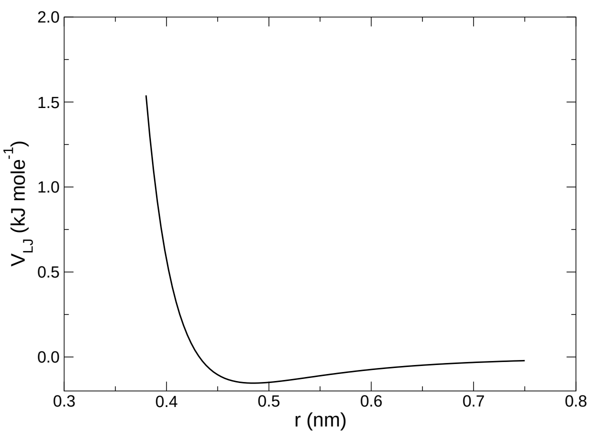


Figure 4.1: The Lennard-Jones interaction.

4.1.1 The Lennard-Jones interaction

The Lennard-Jones potential V_{LJ} between two atoms equals

$$V_{LJ}(r_{ij}) = \frac{C_{ij}^{(12)}}{r_{ij}^{12}} - \frac{C_{ij}^{(6)}}{r_{ij}^{6}} \tag{4.3}$$

see also Fig. 4.1 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.

The force derived from this potential is:

$$\boldsymbol{F}_i(\boldsymbol{r}_{ij}) = \left(12\, \frac{C_{ij}^{(12)}}{r_{ij}^{13}} - 6\, \frac{C_{ij}^{(6)}}{r_{ij}^{7}} \right) \frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{4.4}$$

The LJ potential may also be written in the following form :

$$V_{LJ}(\boldsymbol{r}_{ij}) = 4\epsilon_{ij} \left(\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right) \tag{4.5}$$

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):

$$\begin{array}{rcl} C_{ij}^{(6)} & = & \left(C_{ii}^{(6)} \, C_{jj}^{(6)} \right)^{1/2} \\ C_{ij}^{(12)} & = & \left(C_{ii}^{(12)} \, C_{jj}^{(12)} \right)^{1/2} \end{array} \tag{4.6}$$

or, alternatively the Lorentz-Bertelot 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):

$$\begin{array}{rcl} \sigma_{ij} & = & \frac{1}{2}(\sigma_{ii} + \sigma_{jj}) \\ \epsilon_{ij} & = & (\epsilon_{ii} \, \epsilon_{jj})^{1/2} \end{array} \tag{4.7}$$

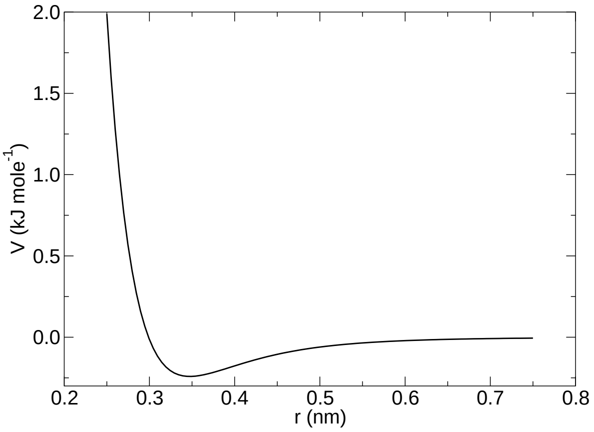


Figure 4.2: The Buckingham interaction.

finally an geometric average for both parameters can be used (type 3):

$$\begin{aligned} \sigma_{ij} &= (\sigma_{ii}\,\sigma_{jj})^{1/2} \\ \epsilon_{ij} &= (\epsilon_{ii}\,\epsilon_{jj})^{1/2} \end{aligned} \tag{4.8}$$

this last rule is used by the OPLS force field.

4.1.2 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:

$$V_{bh}(r_{ij}) = A_{ij}\exp(-B_{ij}r_{ij}) - \frac{C_{ij}}{r_{ij}^6} \tag{4.9}$$

see also Fig. 4.2, the force derived from this is:

$$\boldsymbol{F}_i(r_{ij}) = \left[A_{ij}B_{ij}\exp(-B_{ij}r_{ij}) - 6\frac{C_{ij}}{r_{ij}^7}\right]\frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{4.10}$$

There is only one set of combination rules for Buckingham potentials:

$$\begin{aligned} A_{ij} &= (A_{ii}\,A_{jj})^{1/2} \\ B_{ij} &= \tfrac{1}{2}(B_{ii} + B_{jj}) \\ C_{ij} &= (C_{ii}\,C_{jj})^{1/2} \end{aligned} \tag{4.11}$$

4.1.3 Coulomb interaction

The Coulomb interaction between two charge particles is given by:

$$V_c(r_{ij}) = f\frac{q_i q_j}{\varepsilon_r r_{ij}} \tag{4.12}$$

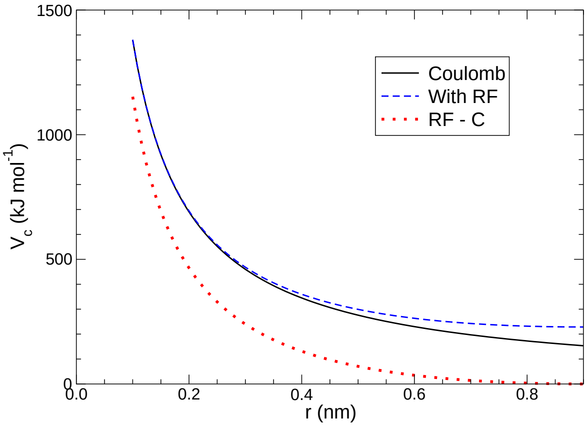


Figure 4.3: The Coulomb interaction (for particles with equal signed charge) with and without reaction field. In the latter case ε_r was 1, ε_{rf} was 78, and r_c was 0.9 nm. The dot-dashed line is the same as the dashed line, except for a constant.

see also Fig. 4.3, where $f = \frac{1}{4\pi\varepsilon_0} = 138.935\,485$ (see chapter 2)

The force derived from this potential is:

$$\boldsymbol{F}_i(\boldsymbol{r}_{ij}) = f \frac{q_i q_j}{\varepsilon_r r_{ij}^2} \frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{4.13}$$

In GROMACS the relative dielectric constant ε_r may be set in the in the input for `grompp`.

4.1.4 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:

$$V_{crf} \;=\; f \frac{q_i q_j}{\varepsilon_r r_{ij}} \left[1 + \frac{\varepsilon_{rf} - \varepsilon_r}{2\varepsilon_{rf} + \varepsilon_r} \frac{r_{ij}^3}{r_c^3}\right] - f \frac{q_i q_j}{\varepsilon_r r_c} \frac{3\varepsilon_{rf}}{2\varepsilon_{rf} + \varepsilon_r} \tag{4.14}$$

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 eqn. 4.14 for simplicity as

$$V_{crf} \;=\; f \frac{q_i q_j}{\varepsilon_r} \left[\frac{1}{r_{ij}} + k_{rf}\, r_{ij}^2 - c_{rf}\right] \tag{4.15}$$

with

$$k_{rf} \;=\; \frac{1}{r_c^3} \frac{\varepsilon_{rf} - \varepsilon_r}{(2\varepsilon_{rf} + \varepsilon_r)} \tag{4.16}$$

$$c_{rf} \;=\; \frac{1}{r_c} + k_{rf}\, r_c^2 \;=\; \frac{1}{r_c} \frac{3\varepsilon_{rf}}{(2\varepsilon_{rf} + \varepsilon_r)} \tag{4.17}$$

For large ε_{rf} the k_{rf} goes to $r_c^{-3}/2$, while for ε_{rf} = ε_r the correction vanishes. In Fig. 4.3 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:

$$\boldsymbol{F}_i(\boldsymbol{r}_{ij}) = f \frac{q_i q_j}{\varepsilon_r} \left[\frac{1}{r_{ij}^2} - 2k_{rf} r_{ij} \right] \frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{4.18}$$

The reaction-field correction should also be applied to all excluded atoms pairs, including self pairs, in which case the normal Coulomb term in eqns. 4.14 and 4.18 is absent.

Tironi *et al.* have introduced a generalized reaction field in which the dielectric continuum beyond the cut-off r_c also has an ionic strength I [65]. In this case we can rewrite the constants k_{rf} and c_{rf} using the inverse Debye screening length κ:

$$\kappa^2 = \frac{2IF^2}{\varepsilon_0 \varepsilon_{rf} RT} = \frac{F^2}{\varepsilon_0 \varepsilon_{rf} RT} \sum_{i=1}^{K} c_i z_i^2 \tag{4.19}$$

$$k_{rf} = \frac{1}{r_c^3} \frac{(\varepsilon_{rf} - \varepsilon_r)(1 + \kappa r_c) + \frac{1}{2}\varepsilon_{rf}(\kappa r_c)^2}{(2\varepsilon_{rf} + \varepsilon_r)(1 + \kappa r_c) + \varepsilon_{rf}(\kappa r_c)^2} \tag{4.20}$$

$$c_{rf} = \frac{1}{r_c} \frac{3\varepsilon_{rf}(1 + \kappa r_c + \frac{1}{2}(\kappa r_c)^2)}{(2\varepsilon_{rf} + \varepsilon_r)(1 + \kappa r_c) + \varepsilon_{rf}(\kappa r_c)^2} \tag{4.21}$$

where F is Faraday's constant, R is the ideal gas constant, T the absolute temperature, c_i the molar concentration for species i and z_i the charge number of species i where we have K different species. In the limit of zero ionic strength ($\kappa = 0$) eqns. 4.20 and 4.21 reduce to the simple forms of eqns. 4.16 and 4.17 respectively.

4.1.5 Modified non-bonded interactions

In the GROMACS force field the non-bonded potentials can be modified by a shift function. 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-step integration produces much smaller errors and there are no such complications as creating charges from dipoles by the truncation procedure. In fact, by using shifted forces there is no need for charge groups in the construction of neighbor lists. However, the shift function produces a considerable modification of the Coulomb potential. Unless the 'missing' long-range potential is properly calculated and added (through the use of PPPM, Ewald, or PME), the effect of such modifications must be carefully evaluated. The modification of the Lennard-Jones dispersion and repulsion is only minor, but it does remove the noise caused by cut-off effects.

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) [66]. 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

$$\boldsymbol{F}_i = cF(r_{ij}) \frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{4.22}$$

For pure Coulomb or Lennard-Jones interactions $F(r) = F_\alpha(r) = r^{-(\alpha+1)}$. The shifted force $F_s(r)$ can generally be written as:

$$\begin{array}{rcll} F_s(r) &=& F_\alpha(r) & r < r_1 \\ F_s(r) &=& F_\alpha(r) + S(r) & r_1 \leq r < r_c \\ F_s(r) &=& 0 & r_c \leq r \end{array} \tag{4.23}$$

When $r_1 = 0$ this is a traditional shift function, otherwise it acts as a switch function. The corresponding shifted coulomb potential then reads:

$$V_s(r_{ij}) = f \Phi_s(r_{ij}) q_i q_j \tag{4.24}$$

where $\Phi(r)$ is the potential function

$$\Phi_s(r) = \int_r^\infty F_s(x)\, dx \tag{4.25}$$

The GROMACS shift function should be smooth at the boundaries, therefore the following boundary conditions are imposed on the shift function:

$$\begin{array}{rcl} S(r_1) &=& 0 \\ S'(r_1) &=& 0 \\ S(r_c) &=& -F_\alpha(r_c) \\ S'(r_c) &=& -F'_\alpha(r_c) \end{array} \tag{4.26}$$

A 3^{rd} degree polynomial of the form

$$S(r) = A(r - r_1)^2 + B(r - r_1)^3 \tag{4.27}$$

fulfills these requirements. The constants A and B are given by the boundary condition at r_c:

$$\begin{array}{rcl} A &=& -\dfrac{(\alpha+4)r_c \; - \; (\alpha+1)r_1}{r_c^{\alpha+2}\,(r_c - r_1)^2} \\ B &=& \dfrac{(\alpha+3)r_c \; - \; (\alpha+1)r_1}{r_c^{\alpha+2}\,(r_c - r_1)^3} \end{array} \tag{4.28}$$

Thus the total force function is

$$F_s(r) = \frac{\alpha}{r^{\alpha+1}} + A(r - r_1)^2 + B(r - r_1)^3 \tag{4.29}$$

and the potential function reads

$$\Phi(r) = \frac{1}{r^\alpha} - \frac{A}{3}(r - r_1)^3 - \frac{B}{4}(r - r_1)^4 - C \tag{4.30}$$

where

$$C = \frac{1}{r_c^\alpha} - \frac{A}{3}(r_c - r_1)^3 - \frac{B}{4}(r_c - r_1)^4 \tag{4.31}$$

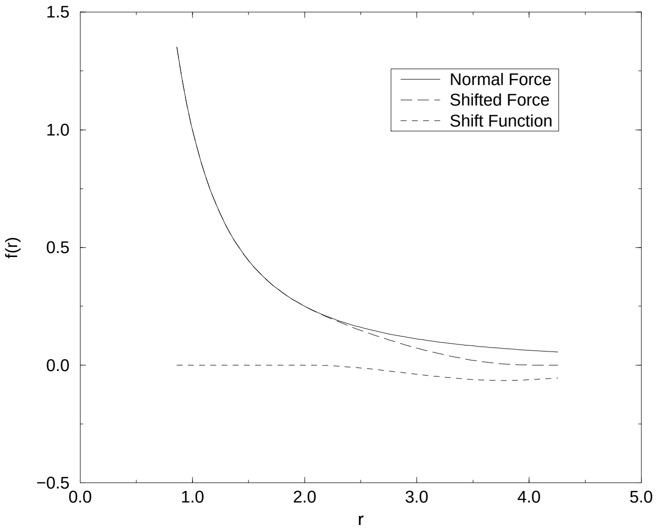


Figure 4.4: The Coulomb Force, Shifted Force and Shift Function $S(r)$, using $r_1 = 2$ and $r_c = 4$.

When $r_1 = 0$, the modified Coulomb force function is

$$F_s(r) = \frac{1}{r^2} - \frac{5r^2}{r_c^4} + \frac{4r^3}{r_c^5} \tag{4.32}$$

identical to the *parabolic force* function recommended to be used as a short-range function in conjunction with a Poisson solver for the long-range part [67]. The modified Coulomb potential function is

$$\Phi(r) = \frac{1}{r} - \frac{5}{3r_c} + \frac{5r^3}{3r_c^4} - \frac{r^4}{r_c^5} \tag{4.33}$$

see also Fig. 4.4.

4.1.6 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, similar to the switch function above. In this case the short range potential is given by

$$V(r) = f\frac{\mathrm{erfc}(\beta r_{ij})}{r_{ij}} q_i q_j, \tag{4.34}$$

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. 4.9.

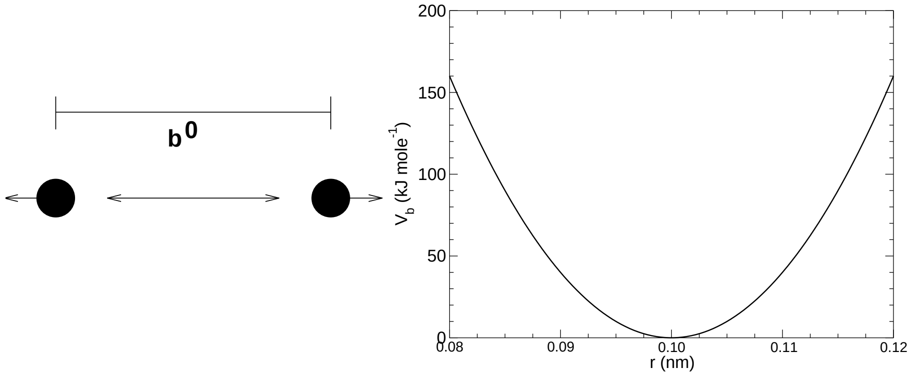


Figure 4.5: Principle of bond stretching (left), and the bond stretching potential (right).

4.2 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).

4.2.1 Bond stretching

Harmonic potential

The bond stretching between two covalently bonded atoms i and j is represented by a harmonic potential

$$V_b\left(r_{ij}\right) = \frac{1}{2}k^b_{ij}(r_{ij}-b_{ij})^2 \tag{4.35}$$

see also Fig. 4.5, with the force

$$\boldsymbol{F}_i(\boldsymbol{r}_{ij}) = k^b_{ij}(r_{ij}-b_{ij})\frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{4.36}$$

Fourth power potential

In the GROMOS-96 force field [68] the covalent bond potential is written for reasons of computational efficiency as:

$$V_b\left(r_{ij}\right) = \frac{1}{4}k^b_{ij}\left(r^2_{ij}-b^2_{ij}\right)^2 \tag{4.37}$$

the corresponding force is:

$$\boldsymbol{F}_i(\boldsymbol{r}_{ij}) = k^b_{ij}(r^2_{ij}-b^2_{ij})\;\boldsymbol{r}_{ij} \tag{4.38}$$

The force constants for this form of the potential is related to the usual harmonic force constant $k^{b,harm}$ (sec. 4.2.1) as

$$2k^b b_{ij}^2 = k^{b,harm} \tag{4.39}$$

The force constants are mostly derived from the harmonic ones used in GROMOS-87 [69]. 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.

4.2.2 Morse potential bond stretching

For some systems that require an anharmonic bond stretching potential, the Morse potential [70] between two atoms *i* and *j* is available in GROMACS. This potential differs from the harmonic potential in having an asymmetric potential well and a zero force at infinite distance. The functional form is:

$$V_{morse}(r_{ij}) = D_{ij}[1 - \exp(-\beta_{ij}(r_{ij} - b_{ij}))]^2, \tag{4.40}$$

see also Fig. 4.6, and the corresponding force is:

$$\begin{aligned}\mathbf{F}_{morse}(\mathbf{r}_{ij}) &= 2D_{ij}\beta_{ij}r_{ij}\exp(-\beta_{ij}(r_{ij} - b_{ij}))* \\ &\quad [1 - \exp(-\beta_{ij}(r_{ij} - b_{ij}))]\frac{\mathbf{r}_{ij}}{r_{ij}},\end{aligned} \tag{4.41}$$

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}:

$$\beta_{ij} = \omega_{ij}\sqrt{\frac{\mu_{ij}}{2D_{ij}}} \tag{4.42}$$

and because $\omega = \sqrt{k/\mu}$, one can rewrite β_{ij} in terms of the harmonic force constant k_{ij}

$$\beta_{ij} = \sqrt{\frac{k_{ij}}{2D_{ij}}} \tag{4.43}$$

For small deviations $(r_{ij} - b_{ij})$, one can approximate the exp-term to first-order using a Taylor expansion:

$$\exp(-x) \approx 1 - x \tag{4.44}$$

and substituting eqn. 4.43 and eqn. 4.44 in the functional from,

$$\begin{aligned}V_{morse}(r_{ij}) &= D_{ij}[1 - \exp(-\beta_{ij}(r_{ij} - b_{ij}))]^2 \\ &= D_{ij}[1 - (1 - \sqrt{\tfrac{k_{ij}}{2D_{ij}}}(r_{ij} - b_{ij}))]^2 \\ &= \tfrac{1}{2}k_{ij}(r_{ij} - b_{ij}))^2,\end{aligned} \tag{4.45}$$

we recover the harmonic bond stretching potential.

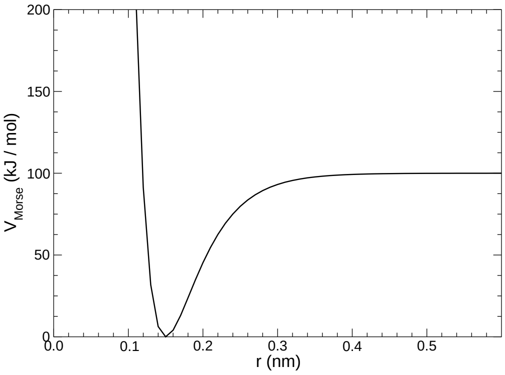


Figure 4.6: The Morse potential well, with bond length 0.15 nm.

4.2.3 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:

$$V_b\,(r_{ij}) = k^b_{ij}(r_{ij}-b_{ij})^2 + k^b_{ij}k^{cub}_{ij}(r_{ij}-b_{ij})^3 \tag{4.46}$$

A flexible water model (based on the SPC water model [71]) including a cubic bond stretching potential for the O-H bond was developed by Ferguson [72]. This model was found to yield a reasonable infrared spectrum. The Ferguson water model is available in the GROMACS library. It should be noted that the potential is asymmetric: overstretching leads to infinitely low energies. The integration timestep is therefore limited to 1 fs.

The force corresponding to this potential is:

$$\boldsymbol{F}_i(\boldsymbol{r}_{ij}) = 2k^b_{ij}(r_{ij}-b_{ij})\,\frac{\boldsymbol{r}_{ij}}{r_{ij}} + 3k^b_{ij}k^{cub}_{ij}(r_{ij}-b_{ij})^2\,\frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{4.47}$$

4.2.4 FENE bond stretching potential

In coarse-grained polymer simulations the beads are often connected by a FENE (finitely extensible nonlinear elastic) potential [73]:

$$V_{\mathrm{FENE}}(r_{ij}) = -\frac{1}{2}k^b_{ij}b^2_{ij}\log\left(1-\frac{r^2_{ij}}{b^2_{ij}}\right) \tag{4.48}$$

The potential looks complicated, but the expression for the force is simpler:

$$F_{\mathrm{FENE}}(\boldsymbol{r}_{ij}) = -k^b_{ij}\left(1-\frac{r^2_{ij}}{b^2_{ij}}\right)^{-1}\boldsymbol{r}_{ij} \tag{4.49}$$

At short distances the potential asymptotically goes to a harmonic potential with force constant k^b, while it diverges at distance b.

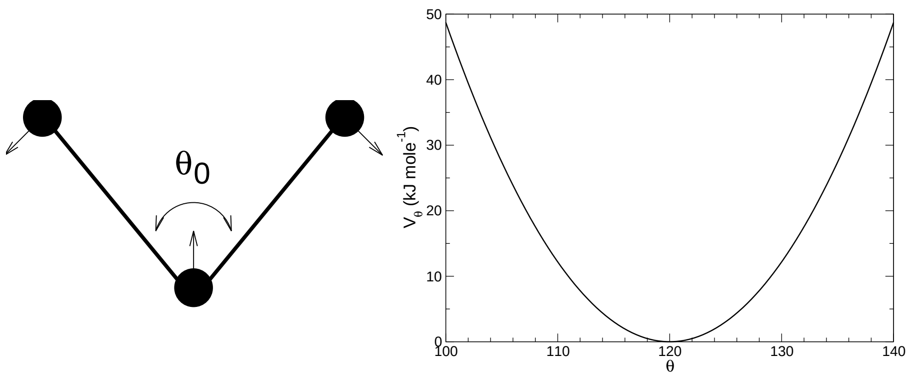


Figure 4.7: Principle of angle vibration (left) and the bond angle potential (right).

4.2.5 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}

$$V_a(\theta_{ijk}) = \frac{1}{2}k^{\theta}_{ijk}(\theta_{ijk} - \theta^0_{ijk})^2 \tag{4.50}$$

As the bond-angle vibration is represented by a harmonic potential, the form is the same as the bond stretching (Fig. 4.5).

The force equations are given by the chain rule:

$$\begin{array}{rcl} \boldsymbol{F}_i & = & -\dfrac{dV_a(\theta_{ijk})}{d\boldsymbol{r}_i} \\ \boldsymbol{F}_k & = & -\dfrac{dV_a(\theta_{ijk})}{d\boldsymbol{r}_k} \\ \boldsymbol{F}_j & = & -\boldsymbol{F}_i - \boldsymbol{F}_k \end{array} \quad \text{where} \quad \theta_{ijk} = \arccos\frac{(\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{kj})}{r_{ij}r_{kj}} \tag{4.51}$$

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. 4.7). **Note** that in the input in topology files, angles are given in degrees and force constants in kJ/mol/rad^2.

4.2.6 Cosine based angle potential

In the GROMOS-96 force field a simplified function is used to represent angle vibrations:

$$V_a(\theta_{ijk}) = \frac{1}{2}k^{\theta}_{ijk}\left(\cos(\theta_{ijk}) - \cos(\theta^0_{ijk})\right)^2 \tag{4.52}$$

where

$$\cos(\theta_{ijk}) = \frac{\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{kj}}{r_{ij}r_{kj}} \tag{4.53}$$

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

$k^{\theta,harm}$ (sec. 4.2.5) by:

$$k^{\theta}\sin^2(\theta^0_{ijk}) = k^{\theta,harm} \tag{4.54}$$

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.

4.2.7 Urey-Bradley potential

The bond Urey-Bradley 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 [74]. The energy is given by:

$$V_a(\theta_{ijk}) = \frac{1}{2}k^{\theta}_{ijk}(\theta_{ijk} - \theta^0_{ijk})^2 + \frac{1}{2}k^{UB}_{ijk}(r_{ik} - r^0_{ik})^2 \tag{4.55}$$

The force equations can be deduced from sections 4.2.1 and 4.2.5.

4.2.8 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 [75]:

$$V_{rr'} \;=\; k_{rr'}\left(|\boldsymbol{r}_i - \boldsymbol{r}_j| - r_{1e}\right)\left(|\boldsymbol{r}_k - \boldsymbol{r}_j| - r_{2e}\right) \tag{4.56}$$

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:

$$\boldsymbol{F}_i = -k_{rr'}\left(|\boldsymbol{r}_k - \boldsymbol{r}_j| - r_{2e}\right)\frac{\boldsymbol{r}_i - \boldsymbol{r}_j}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} \tag{4.57}$$

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: $\boldsymbol{F}_j = -\boldsymbol{F}_i - \boldsymbol{F}_k$.

4.2.9 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 [75]:

$$V_{r\theta} \;=\; k_{r\theta}\left(|\boldsymbol{r}_i - \boldsymbol{r}_k| - r_{3e}\right)\left(|\boldsymbol{r}_i - \boldsymbol{r}_j| - r_{1e} + |\boldsymbol{r}_k - \boldsymbol{r}_j| - r_{2e}\right) \tag{4.58}$$

where $k_{r\theta}$ is the force constant, r_{3e} is the $i - k$ distance, and the other constants are the same as in Equation 4.56. The force associated with the potential on atom i is:

$$\boldsymbol{F}_i \;=\; -\,k_{r\theta}\left[\left(|\boldsymbol{r}_i - \boldsymbol{r}_k| - r_{3e}\right)\frac{\boldsymbol{r}_i - \boldsymbol{r}_j}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} + \left(|\boldsymbol{r}_i - \boldsymbol{r}_j| - r_{1e} + |\boldsymbol{r}_k - \boldsymbol{r}_j| - r_{2e}\right)\frac{\boldsymbol{r}_i - \boldsymbol{r}_k}{|\boldsymbol{r}_i - \boldsymbol{r}_k|}\right] \tag{4.59}$$

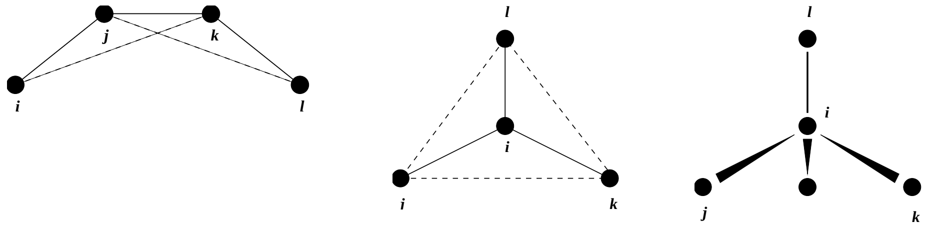


Figure 4.8: Principle of improper dihedral angles. Out of plane bending for rings (left), substituents of rings (middle), out of tetrahedral (right). The improper dihedral angle ξ is defined as the angle between planes (i,j,k) and (j,k,l) in all cases.

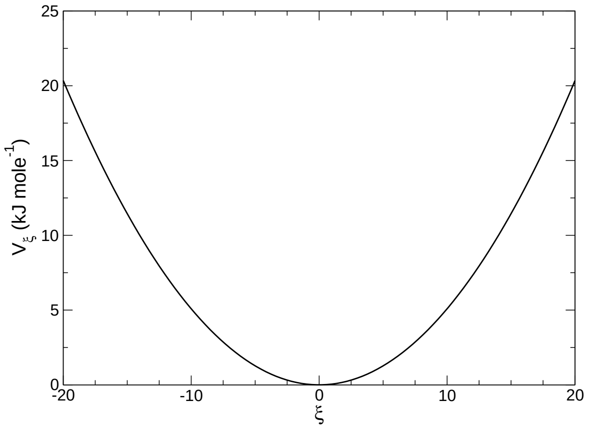


Figure 4.9: Improper dihedral potential.

4.2.10 Quartic angle potential

For special purposes there is an angle potential that uses a fourth order polynomial:

$$V_q(\theta_{ijk}) \;=\; \sum_{n=0}^{5} C_n(\theta_{ijk}-\theta_{ijk}^0)^n \tag{4.60}$$

4.2.11 Improper dihedrals

Improper dihedrals are meant to keep planar groups planar (*e.g.* aromatic rings) or to prevent molecules from flipping over to their mirror images, see Fig. 4.8.

$$V_{id}(\xi_{ijkl}) = \frac{1}{2}k_\xi(\xi_{ijkl}-\xi_0)^2 \tag{4.61}$$

This is also a harmonic potential; it is plotted in Fig. 4.9. 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.

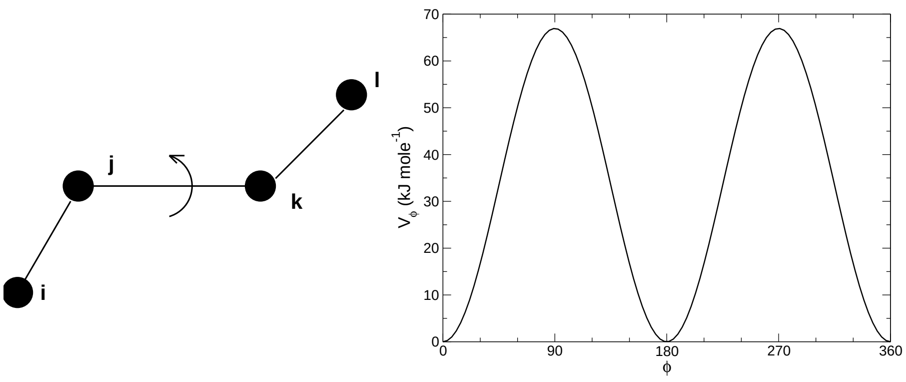


Figure 4.10: Principle of proper dihedral angle (left, in *trans* form) and the dihedral angle potential (right).

4.2.12 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 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).

$$V_d(\phi_{ijkl}) = k_\phi(1 + \cos(n\phi - \phi_s)) \tag{4.62}$$

Proper dihedrals: Ryckaert-Bellemans function

For alkanes, the following proper dihedral potential is often used (see Fig. 4.11)

$$V_{rb}(\phi_{ijkl}) = \sum_{n=0}^{5} C_n(\cos(\psi))^n, \tag{4.63}$$

where $\psi = \phi - 180°$.
Note: A conversion from one convention to another can be achieved by multiplying every coefficient C_n by $(-1)^n$.

C_0	9.28	C_2	-13.12	C_4	26.24
C_1	12.16	C_3	-3.06	C_5	-31.5

Table 4.1: Constants for Ryckaert-Bellemans potential (kJ mol^{-1}).

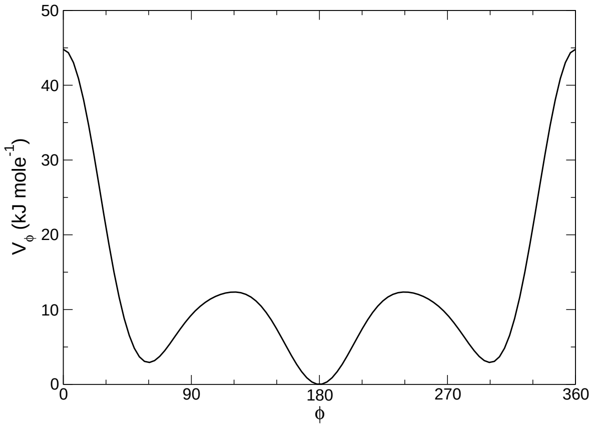


Figure 4.11: Ryckaert-Bellemans dihedral potential.

An example of constants for C is given in Table 4.1.

(**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):

$$V_{rb}(\phi_{ijkl}) \;=\; \frac{1}{2}\left[F_1(1+\cos(\phi)) + F_2(1-\cos(2\phi)) + F_3(1+\cos(3\phi) + F_4(1-\cos(4\phi))\right] \tag{4.64}$$

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:

$$\begin{array}{rcl} C_0 & = & F_2 + \frac{1}{2}(F_1 + F_3) \\ C_1 & = & \frac{1}{2}(-F_1 + 3\,F_3) \\ C_2 & = & -F_2 + 4\,F_4 \\ C_3 & = & -2\,F_3 \\ C_4 & = & -4\,F_4 \\ C_5 & = & 0 \end{array} \tag{4.65}$$

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 or four [76] cosine terms of a Fourier series. In GROMACS the four term function is implemented:

$$V_F(\phi_{ijkl}) \;=\; \frac{1}{2}\left[C_1(1+\cos(\phi)) + C_2(1-\cos(2\phi)) + C_3(1+\cos(3\phi) + C_4(1+\cos(4\phi))\right], \tag{4.66}$$

Internally GROMACS uses the Ryckaert-Bellemans code to compute Fourier dihedrals (see above), because this is more efficient.
Note: Mind the conversion from *kcal mol*$^{-1}$ for literature OPLS parameters to *kJ mol*$^{-1}$ in GROMACS.

4.2.13 Tabulated 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:

$$V_b(r_{ij}) = k\, f^b_n(r_{ij}) \tag{4.67}$$
$$V_a(\theta_{ijk}) = k\, f^a_n(\theta_{ijk}) \tag{4.68}$$
$$V_d(\phi_{ijkl}) = k\, f^d_n(\phi_{ijkl}) \tag{4.69}$$

where k is a force constant in units of energy and f is a cubic spline function, for details see 6.7.1. For each interaction the force constant k and the table number n are specified in the topology. The are two different types of bonds, one that generates exclusions and one that does not. For details see Table 5.5. The table files are supplied to the `mdrun` program. After the table file name an underscore, the letter 'b' for bonds, 'a' for angles or 'd' for dihedrals and the table number are appended. For example, for a bond with $n = 0$ (and using the default table file name) the table is read from the file `table_b0.xvg`. The format for the table files is three columns with x, $f(x)$, $-f'(x)$, where x should be uniformly spaced. The setup of the tables is as follows:
bonds: x is the distance in nanometers, for distances beyond the table length cause `mdrun` to 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

4.3 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.

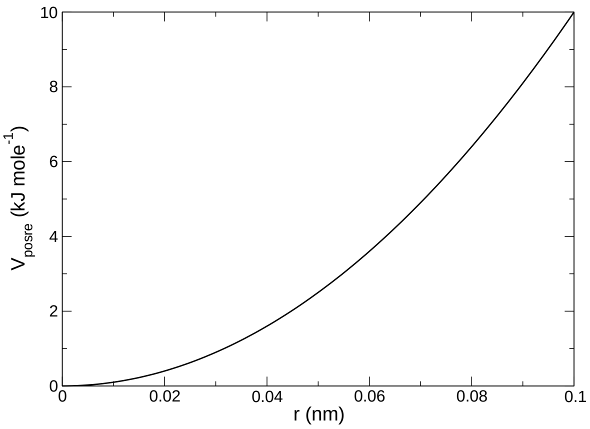


Figure 4.12: Position restraint potential.

4.3.1 Position restraints

These are used to restrain particles to fixed reference positions $\boldsymbol{R}_i$. They can be used during equilibration in order to avoid too 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:

$$V_{pr}(\boldsymbol{r}_i) = \frac{1}{2}k_{pr}|\boldsymbol{r}_i - \boldsymbol{R}_i|^2 \tag{4.70}$$

The potential is plotted in Fig. 4.12.

The potential form can be rewritten without loss of generality as:

$$V_{pr}(\boldsymbol{r}_i) = \frac{1}{2}\left[k_{pr}^x(x_i - X_i)^2\,\hat{\mathbf{x}} + k_{pr}^y(y_i - Y_i)^2\,\hat{\mathbf{y}} + k_{pr}^z(z_i - Z_i)^2\,\hat{\mathbf{z}}\right] \tag{4.71}$$

Now the forces are:

$$\begin{array}{rcl} F_i^x & = & -k_{pr}^x\,(x_i - X_i) \\ F_i^y & = & -k_{pr}^y\,(y_i - Y_i) \\ F_i^z & = & -k_{pr}^z\,(z_i - Z_i) \end{array} \tag{4.72}$$

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 program.

4.3.2 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:

$$V_{ar}(\boldsymbol{r}_i,\boldsymbol{r}_j,\boldsymbol{r}_k,\boldsymbol{r}_l) = k_{ar}(1-\cos(n(\theta-\theta_0))), \quad \text{where } \theta = \arccos\left(\frac{\boldsymbol{r}_j-\boldsymbol{r}_i}{\|\boldsymbol{r}_j-\boldsymbol{r}_i\|}\cdot\frac{\boldsymbol{r}_l-\boldsymbol{r}_k}{\|\boldsymbol{r}_l-\boldsymbol{r}_k\|}\right) \tag{4.73}$$

For one pair of atoms and the Z-axis:

$$V_{ar}(\boldsymbol{r}_i,\boldsymbol{r}_j) = k_{ar}(1-\cos(n(\theta-\theta_0))), \quad \text{where } \theta = \arccos\left(\frac{\boldsymbol{r}_j-\boldsymbol{r}_i}{\|\boldsymbol{r}_j-\boldsymbol{r}_i\|}\cdot\begin{pmatrix}0\\0\\1\end{pmatrix}\right) \tag{4.74}$$

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.

4.3.3 Dihedral restraints

These are used to restrain the dihedral angle ϕ defined by four particles as in an improper dihedral (sec. 4.2.11) but with a slightly modified potential. Using

$$\phi' = (\phi - \phi_0) \ \mathrm{MOD}\ 2\pi \tag{4.75}$$

where ϕ_0 is the reference angle, the potential is defined as:

$$V_{dihr}(\phi') = \begin{cases} \frac{1}{2}k_{dihr}(\phi'-\phi_0-\Delta\phi)^2 & \text{for} \quad \phi' > \Delta\phi \\ 0 & \text{for} \quad \phi' \leq \Delta\phi \end{cases} \tag{4.76}$$

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.

4.3.4 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, as 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. 5.4).
- Piecewise linear/harmonic restraints: `[ bonds ]` type 10.
- Complex NMR distance restraints, optionally with pair, time and/or ensemble averaging.

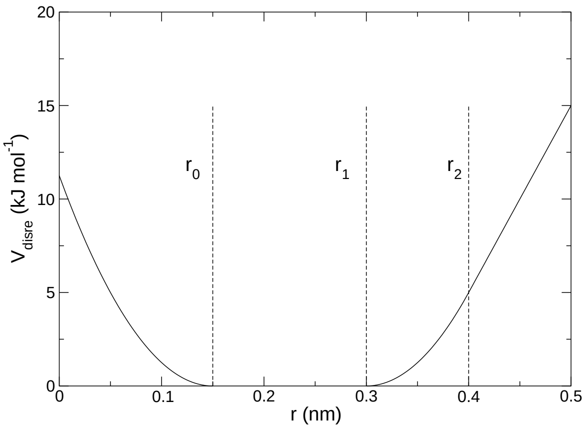


Figure 4.13: Distance Restraint potential.

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. 4.13).

$$V_{dr}(r_{ij}) = \begin{cases} \frac{1}{2}k_{dr}(r_{ij}-r_0)^2 & \text{for} \quad r_{ij} < r_0 \\ 0 & \text{for} \quad r_0 \le r_{ij} < r_1 \\ \frac{1}{2}k_{dr}(r_{ij}-r_1)^2 & \text{for} \quad r_1 \le r_{ij} < r_2 \\ \frac{1}{2}k_{dr}(r_2-r_1)(2r_{ij}-r_2-r_1) & \text{for} \quad r_2 \le r_{ij} \end{cases} \tag{4.77}$$

The forces are

$$\boldsymbol{F}_i = \begin{cases} -k_{dr}(r_{ij}-r_0)\frac{\boldsymbol{r}_{ij}}{r_{ij}} & \text{for} \quad r_{ij} < r_0 \\ 0 & \text{for} \quad r_0 \le r_{ij} < r_1 \\ -k_{dr}(r_{ij}-r_1)\frac{\boldsymbol{r}_{ij}}{r_{ij}} & \text{for} \quad r_1 \le r_{ij} < r_2 \\ -k_{dr}(r_2-r_1)\frac{\boldsymbol{r}_{ij}}{r_{ij}} & \text{for} \quad r_2 \le r_{ij} \end{cases} \tag{4.78}$$

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, atoms, see 5.7.1. 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 [77]. The forces with time averaging are:

$$
\boldsymbol{F}_i = \begin{cases}
-k^a_{dr}(\bar{r}_{ij} - r_0)\frac{\boldsymbol{r}_{ij}}{r_{ij}} & \text{for} \quad \bar{r}_{ij} < r_0 \\
0 & \text{for} \quad r_0 \le \bar{r}_{ij} < r_1 \\
-k^a_{dr}(\bar{r}_{ij} - r_1)\frac{\boldsymbol{r}_{ij}}{r_{ij}} & \text{for} \quad r_1 \le \bar{r}_{ij} < r_2 \\
-k^a_{dr}(r_2 - r_1)\frac{\boldsymbol{r}_{ij}}{r_{ij}} & \text{for} \quad r_2 \le \bar{r}_{ij}
\end{cases} \tag{4.79}
$$

where $\bar{r}_{ij}$ is given by an exponential running average with decay time τ:

$$
\bar{r}_{ij} = \, < r_{ij}^{-3} >^{-1/3} \tag{4.80}
$$

and the force constant k^a_{dr} is switched on slowly to compensate for the lack of history at the beginning of the simulation:

$$
k^a_{dr} = k_{dr}\left(1 - \exp\left(-\frac{t}{\tau}\right)\right) \tag{4.81}
$$

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 which 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` program is done in the following fashion:

$$
\begin{aligned}
\overline{r^{-3}}_{ij}(0) &= r_{ij}(0)^{-3} \\
\overline{r^{-3}}_{ij}(t) &= \overline{r^{-3}}_{ij}(t - \Delta t)\ \exp\left(-\frac{\Delta t}{\tau}\right) + r_{ij}(t)^{-3}\left[1 - \exp\left(-\frac{\Delta t}{\tau}\right)\right]
\end{aligned} \tag{4.82}
$$

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:

$$
\boldsymbol{F}_i = \begin{cases}
k^a_{dr}\sqrt{(r_{ij} - r_0)(\bar{r}_{ij} - r_0)}\frac{\boldsymbol{r}_{ij}}{r_{ij}} & \text{for} \quad r_{ij} < r_0 \quad \text{and} \quad \bar{r}_{ij} < r_0 \\
-k^a_{dr}\min\left(\sqrt{(r_{ij} - r_1)(\bar{r}_{ij} - r_1)}, r_2 - r_1\right)\frac{\boldsymbol{r}_{ij}}{r_{ij}} & \text{for} \quad r_{ij} > r_1 \quad \text{and} \quad \bar{r}_{ij} > r_1 \\
0 & \text{else}
\end{cases} \tag{4.83}
$$

Averaging over multiple pairs

Sometimes it is unclear from experimental data which atom pair gives rise to a single NOE, in other occasions it can be obvious that more than one pair contributes due to the symmetry of the system, *e.g.* a methyl group with three protons. For such a group it is not possible to distinguish between the protons, therefore they should all be taken into account when calculating the distance between this methyl group and another proton (or group of protons). Due to the physical nature of magnetic resonance, the intensity of the NOE signal is inversely proportional to the sixth power of the inter-atomic distance. Thus, when combining atom pairs, a fixed list of N restraints may be taken together, where the apparent "distance" is given by:

$$r_N(t) = \left[\sum_{n=1}^{N} \bar{r}_n(t)^{-6}\right]^{-1/6} \tag{4.84}$$

where we use r_{ij} or eqn. 4.80 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 'too rigid' molecule. 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:

$$\begin{array}{rcl} r_1 & = & r_1 * M^{-1/6} \\ r_2 & = & r_2 * M^{-1/6} \end{array} \tag{4.85}$$

where M is the number of molecules. The GROMACS preprocessor `grompp` can do this automatically when the appropriate option is given. The resulting "distance" is then used to calculate the scalar force according to:

$$\begin{array}{rclr} \boldsymbol{F}_i & = & 0 & r_N < r_1 \\ & = & -\,k_{dr}(r_N - r_1)\frac{\boldsymbol{r}_{ij}}{r_{ij}} & r_1 \le r_N < r_2 \\ & = & -\,k_{dr}(r_2 - r_1)\frac{\boldsymbol{r}_{ij}}{r_{ij}} & r_N \ge r_2 \end{array} \tag{4.86}$$

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 sec. 4.3.4, 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 should 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 which 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 eqn. 4.77. 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.

Some parameters for NMR refinement can be specified in the `grompp.mdp` file:

`disre`**: type of distance restraining.** The `disre` variable sets the type of distance restraint. `no/simple` turns the distance restraints off/on. When multiple proteins or peptides are present in one simulation box, ensemble averaging can be turned on by setting `disre = ensemble`. Normally one would perform ensemble averaging over multiple subsystems, each in a separate box, using `mdrun -multi`; supply `topol0.tpr`, `topol1.tpr`, ... with different coordinates and/or velocities.

`disre_weighting`**: force-weighting in restraints with multiple pairs.** By default, the force due to the distance restraint is distributed equally over all the pairs involved in the restraint. This can also be explicitly selected with `disre_weighting = equal`. If you instead set this option to `disre_weighting = conservative` you get conservative forces when `disre_tau = 0`.

`disre_mixed`**: how to calculate the violations.** `disre_mixed = no` gives normal time-averaged violations. When `disre_mixed = yes` the square root of the product of the time-averaged and the instantaneous violations is used.

`disre_fc`**: force constant k_{dr} for distance restraints.** k_{dr} (eqn. 4.77) can be set as variable `disre_fc = 1000` for a force constant of 1000 kJ mol^{-1} nm^{-2}. This value is multiplied by the value in the `fac` column in the distance restraint entries in the topology file.

`disre_tau`**: time constant for restraints.** τ (eqn. 4.82) can be set as variable `disre_tau = 10` for a time constant of 10 ps. Time averaging can be turned off by setting `disre_tau` to 0.

`nstdisreout`**: pair distance output frequency.** Determines how often the time-averaged and instantaneous distances of all atom pairs involved in distance restraints are written to the energy file.

4.3.5 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 [78].

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 $\boldsymbol{r}_i$ can be written as follows:

$$\delta_i = \frac{2}{3}\mathrm{tr}(\mathbf{S}\mathbf{D}_i) \tag{4.87}$$

where $\mathbf{S}$ is the dimensionless order tensor of the molecule. The tensor $\mathbf{D}_i$ is given by:

$$\mathbf{D}_i = \frac{c_i}{\|\boldsymbol{r}_i\|^\alpha}\begin{pmatrix} 3xx-1 & 3xy & 3xz \\ 3xy & 3yy-1 & 3yz \\ 3xz & 3yz & 3zz-1 \end{pmatrix} \tag{4.88}$$

$$\text{with:} \quad x = \frac{r_{i,x}}{\|\boldsymbol{r}_i\|}, \quad y = \frac{r_{i,y}}{\|\boldsymbol{r}_i\|}, \quad z = \frac{r_{i,z}}{\|\boldsymbol{r}_i\|} \tag{4.89}$$

For a dipolar coupling $\boldsymbol{r}_i$ is the vector connecting the two nuclei, $\alpha = 3$ and the constant c_i is given by:

$$c_i = \frac{\mu_0}{4\pi}\gamma_1^i\gamma_2^i\frac{\hbar}{4\pi} \tag{4.90}$$

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:

$$\mathbf{T}^T\mathbf{S}\mathbf{T} = s\begin{pmatrix} -\frac{1}{2}(1-\eta) & 0 & 0 \\ 0 & -\frac{1}{2}(1+\eta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.91}$$

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:

$$\mathbf{D}_i^c(t) = \mathbf{R}(t)\mathbf{D}_i(t)\mathbf{R}^T(t) \tag{4.92}$$

The calculated orientation for vector i is given by:

$$\delta_i^c(t) = \frac{2}{3}\mathrm{tr}(\mathbf{S}(t)\mathbf{D}_i^c(t)) \tag{4.93}$$

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:

$$MSD(t) = \left(\sum_{i=1}^{N} w_i\right)^{-1} \sum_{i=1}^{N} w_i(\delta_i^c(t) - \delta_i^{exp})^2 \tag{4.94}$$

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 experiment, they should be time averaged in the simulation. However, in a simulation the time as well as the number of copies of a molecule is 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 $\boldsymbol{r}_i$ vectors within the molecule converge much faster than the tumbling time of the molecule, the tensor can be averaged in an axis system which rotates with the molecule, as expressed by equation (4.92). The time averaged tensors are calculated using an exponentially decaying memory function:

$$\mathbf{D}_i^a(t) = \frac{\displaystyle\int_{u=t_0}^{t} \mathbf{D}_i^c(u) \exp\left(-\frac{t-u}{\tau}\right) \mathrm{d}u}{\displaystyle\int_{u=t_0}^{t} \exp\left(-\frac{t-u}{\tau}\right) \mathrm{d}u} \tag{4.95}$$

Assuming that the order tensor $\mathbf{S}$ fluctuates slower than the $\mathbf{D}_i$, the time averaged orientation can be calculated as:

$$\delta_i^a(t) = \frac{2}{3}\mathrm{tr}(\mathbf{S}(t)\mathbf{D}_i^a(t)) \tag{4.96}$$

where the order tensor $\mathbf{S}(t)$ is calculated using expression (4.94) 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:

$$V = \frac{1}{2}k\sum_{i=1}^{N} w_i(\delta_i^c(t) - \delta_i^{exp})^2 \tag{4.97}$$

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 $\boldsymbol{F}_i$ working on vector $\boldsymbol{r}_i$ is:

$$\begin{aligned}
\boldsymbol{F}_i(t) &= -\frac{\mathrm{d}V}{\mathrm{d}\boldsymbol{r}_i} \\
&= -kw_i(\delta_i^c(t) - \delta_i^{exp})\frac{\mathrm{d}\delta_i(t)}{\mathrm{d}\boldsymbol{r}_i} \\
&= -kw_i(\delta_i^c(t) - \delta_i^{exp})\frac{2c_i}{\|\boldsymbol{r}\|^{2+\alpha}}\left(2\mathbf{R}^T\mathbf{S}\mathbf{R}\boldsymbol{r}_i - \frac{2+\alpha}{\|\boldsymbol{r}\|^2}\mathrm{tr}(\mathbf{R}^T\mathbf{S}\mathbf{R}\boldsymbol{r}_i\boldsymbol{r}_i^T)\boldsymbol{r}_i\right)
\end{aligned}$$

Ensemble averaging

Ensemble averaging can be applied by simulating a system of M subsystems which each contain an identical set of orientation restraints. The systems only interact via the orientation restraint potential which is defined as:

$$V = M \frac{1}{2} k \sum_{i=1}^{N} w_i \langle \delta_i^c(t) - \delta_i^{exp} \rangle^2 \tag{4.98}$$

The force on vector $\boldsymbol{r}_{i,m}$ in subsystem m is given by:

$$\boldsymbol{F}_{i,m}(t) = -\frac{\mathrm{d}V}{\mathrm{d}\boldsymbol{r}_{i,m}} = -k w_i \langle \delta_i^c(t) - \delta_i^{exp} \rangle \frac{\mathrm{d}\delta_{i,m}^c(t)}{\mathrm{d}\boldsymbol{r}_{i,m}} \tag{4.99}$$

Time averaging

When using time averaging it is not possible to define a potential. We can still define a quantity which gives a rough idea of the energy stored in the restraints:

$$V = M \frac{1}{2} k^a \sum_{i=1}^{N} w_i \langle \delta_i^a(t) - \delta_i^{exp} \rangle^2 \tag{4.100}$$

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 which has been accumulated:

$$k^a = k \, \frac{1}{\tau} \int_{u=t_0}^{t} \exp\left(-\frac{t-u}{\tau}\right) \mathrm{d}u \tag{4.101}$$

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:

$$\boldsymbol{F}_{i,m}(t) = \begin{cases} 0 & \text{for} \quad a\,b \leq 0 \\ k^a w_i \dfrac{a}{|a|} \sqrt{a\,b} \, \dfrac{\mathrm{d}\delta_{i,m}^c(t)}{\mathrm{d}\boldsymbol{r}_{i,m}} & \text{for} \quad a\,b > 0 \end{cases} \tag{4.102}$$

$$\begin{aligned} a &= \langle \delta_i^a(t) - \delta_i^{exp} \rangle \\ b &= \langle \delta_i^c(t) - \delta_i^{exp} \rangle \end{aligned}$$

Using orientation restraints

Orientation restraints can be added to a molecule definition in the topology 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      1/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, this starts numbering at 1. For each experiment a separate order tensor **S** is optimized. The label should be a unique number larger than zero for each restraint. The `alpha` column contains the power α which is used in equation (4.88) 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.mdp` file, for a study of the effect of different force constants and averaging times and ensemble averaging see [78].

`orire`: **use orientation restraining.** `no/yes` turns the distance restraints off/on. Ensemble averaging can be performed using `mdrun -multi`, which simulates multiple subsystems in separate boxes; supply `topol0.tpr`, `topol1.tpr`, ... with different coordinates and/or velocities.

`orire_fc`: **force constant k for orientation restraints.** The unit of k is kJ mol^{-1}. Note that the force constant for a restraint is this force constant times the weight of the restraint. When set to zero one obtain the calculated orientation without affecting the simulation.

`orire_tau`: **time constant τ for restraints.** Set `orire_tau = 10` for a time constant of 10 ps. Time averaging can be turned off by setting `orire_tau` to 0.

`orire_fitgrp`: **the fit group for the restraints.** 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` should be a reasonable choice.

`nstorireout`: **orientation output frequency.** Determines how often the orientations for all restraints and the order tensor(s) **S** are written to the energy file. When using time and/or ensemble averaging, the time and ensemble averaged orientations as well as the instantaneous non-ensemble averaged orientations are written to the energy file. These can be analyzed using `g_energy`.

4.4 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.

4.4.1 Simple polarization

This is merely a harmonic potential with equilibrium distance 0.

4.4.2 Water polarization

A special potential for water that allows anisotropic polarization of a single shell particle [38].

4.4.3 Thole polarization

Based on early work by Thole [79] Roux and coworkers have implemented potentials for molecules like ethanol [80, 81, 82]. 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:

$$V_{thole} = \frac{q_i q_j}{r_{ij}}\left[1 - \left(1 + \frac{\bar{r}_{ij}}{2}\right)\exp^{-\bar{r}_{ij}}\right] \tag{4.103}$$

(note that there is a sign error in Equation 1 of Noskov *et al.* [82]), where

$$\bar{r}_{ij} = a\frac{r_{ij}}{(\alpha_i \alpha_j)^{1/6}} \tag{4.104}$$

where a is a magic (dimensionless) constant, usually chosen to be 2.6 [82] and α_i, α_j are the polarizabilities of the respective shell particles.

4.5 Free energy interactions

This section describes the λ-dependence of the potentials used for free energy calculations (see sec. 3.12). 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.

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.

$$V_b = \frac{1}{2}\left[(1-\lambda)k_b^A + \lambda k_b^B\right]\left[b - (1-\lambda)b_0^A - \lambda b_0^B\right]^2 \tag{4.105}$$

$$\begin{aligned}
\frac{\partial V_b}{\partial \lambda} &= \frac{1}{2}(k_b^B - k_b^A)\left[b - (1-\lambda)b_0^A + \lambda b_0^B\right]^2 + \\
&\quad (b_0^A - b_0^B)\left[b - (1-\lambda)b_0^A - \lambda b_0^B\right]\left[(1-\lambda)k_b^A + \lambda k_b^B\right] \qquad (4.106)
\end{aligned}$$

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:

$$V_d = \left[(1-\lambda)k_d^A + \lambda k_d^B\right]\left(1 + \cos\left[n_\phi \phi - (1-\lambda)\phi_s^A - \lambda \phi_s^B\right]\right) \qquad (4.107)$$

$$\begin{aligned}
\frac{\partial V_d}{\partial \lambda} &= (k_d^B - k_d^A)\left(1 + \cos\left[n_\phi \phi - (1-\lambda)\phi_s^A - \lambda \phi_s^B\right]\right) + \\
&\quad (\phi_s^B - \phi_s^A)\left[(1-\lambda)k_d^A - \lambda k_d^B\right]\sin\left[n_\phi \phi - (1-\lambda)\phi_s^A - \lambda \phi_s^B\right] \qquad (4.108)
\end{aligned}$$

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:

$$V = ((1-\lambda)k^A + \lambda k^B)\, f \qquad (4.109)$$

$$\frac{\partial V}{\partial \lambda} = (k^B - k^A)\, f \qquad (4.110)$$

Coulomb interaction

The Coulomb interaction between two particles of which the charge varies with λ is:

$$V_c = \frac{f}{\varepsilon_{rf} r_{ij}}\left[(1-\lambda)q_i^A q_j^A + \lambda\, q_i^B q_j^B\right] \qquad (4.111)$$

$$\frac{\partial V_c}{\partial \lambda} = \frac{f}{\varepsilon_{rf} r_{ij}}\left[-q_i^A q_j^A + q_i^B q_j^B\right] \qquad (4.112)$$

where $f = \frac{1}{4\pi\varepsilon_0} = 138.935\,485$ (see chapter 2)

Coulomb interaction with reaction field

The coulomb interaction including a reaction field, between two particles of which the charge varies with λ is:

$$V_c = f\left[\frac{1}{r_{ij}} + k_{rf}\, r_{ij}^2 - c_{rf}\right]\left[(1-\lambda)q_i^A q_j^A + \lambda\, q_i^B q_j^B\right] \qquad (4.113)$$

$$\frac{\partial V_c}{\partial \lambda} = f\left[\frac{1}{r_{ij}} + k_{rf}\,r_{ij}^2 - c_{rf}\right]\left[-q_i^A q_j^A + q_i^B q_j^B\right] \tag{4.114}$$

Note that the constants k_{rf} and c_{rf} are defined using the dielectric constant ε_{rf} of the medium (see sec. 4.1.4).

Lennard-Jones interaction

For the Lennard-Jones interaction between two particles of which the *atom type* varies with λ we can write:

$$V_{LJ} = \frac{(1-\lambda)C_{12}^A + \lambda\,C_{12}^B}{r_{ij}^{12}} - \frac{(1-\lambda)C_6^A + \lambda\,C_6^B}{r_{ij}^6} \tag{4.115}$$

$$\frac{\partial V_{LJ}}{\partial \lambda} = \frac{C_{12}^B - C_{12}^A}{r_{ij}^{12}} - \frac{C_6^B - C_6^A}{r_{ij}^6} \tag{4.116}$$

It should be noted that it is also possible to express a pathway from state A to state B using σ and ϵ (see eqn. 4.5). 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 $\boldsymbol{p}$ as m$\boldsymbol{v}$, since that would result in the sign of $\frac{\partial Ek}{\partial \lambda}$ being incorrect [83]):

$$Ek = \frac{1}{2}\frac{\boldsymbol{p}^2}{(1-\lambda)m^A + \lambda m^B} \tag{4.117}$$

$$\frac{\partial Ek}{\partial \lambda} = -\frac{1}{2}\frac{\boldsymbol{p}^2(m^B - m^A)}{((1-\lambda)m^A + \lambda m^B)^2} \tag{4.118}$$

after taking the derivative, we *can* insert $\boldsymbol{p}$ = m$\boldsymbol{v}$, such that:

$$\frac{\partial Ek}{\partial \lambda} = -\frac{1}{2}\boldsymbol{v}^2(m^B - m^A) \tag{4.119}$$

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 a number of constraint equations g_k:

$$g_k = r_k - d_k \tag{4.120}$$

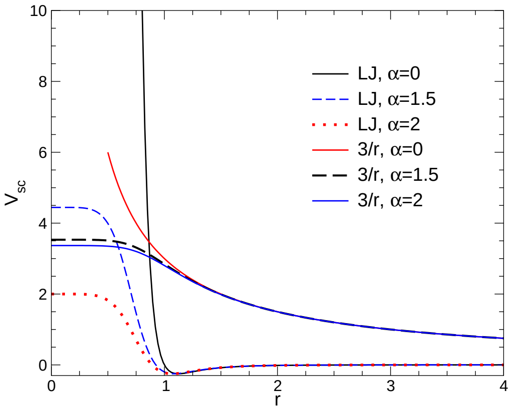


Figure 4.14: Soft-core interactions at $\lambda = 0.5$, with $p = 2$ and $C_6^A = C_{12}^A = C_6^B = C_{12}^B = 1$.

where $\boldsymbol{r}_k$ is the distance vector between two particles and d_k is the constraint distance between the two particles, we can write this using a λ-dependent distance as

$$g_k = r_k - \left((1-\lambda)d_k^A + \lambda d_k^B\right) \tag{4.121}$$

the contribution C_λ to the Hamiltonian using Lagrange multipliers λ:

$$C_\lambda = \sum_k \lambda_k g_k \tag{4.122}$$

$$\frac{\partial C_\lambda}{\partial \lambda} = \sum_k \lambda_k \left(d_k^B - d_k^A\right) \tag{4.123}$$

4.5.1 Soft-core interactions

In a free-energy calculation where particles grow out of nothing, or particles disappear, using the the simple linear interpolation of the Lennard-Jones and Coulomb potentials as described in Equations 4.116 and 4.114 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:

$$V_{sc}(r) = (1-\lambda)V^A(r_A) + \lambda V^B(r_B) \tag{4.124}$$

$$r_A = \left(\alpha\sigma_A^6\lambda^p + r^6\right)^{\frac{1}{6}} \tag{4.125}$$

$$r_B = \left(\alpha\sigma_B^6(1-\lambda)^p + r^6\right)^{\frac{1}{6}} \tag{4.126}$$

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` 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. 4.14). The force is:

$$F_{sc}(r) = -\frac{\partial V_{sc}(r)}{\partial r} = (1-\lambda)F^A(r_A)\left(\frac{r}{r_A}\right)^5 + \lambda F^B(r_B)\left(\frac{r}{r_B}\right)^5 \tag{4.127}$$

where F^A and F^B are the 'hard core' forces. The contribution to the derivative of the free energy is:

$$\begin{aligned}\frac{\partial V_{sc}(r)}{\partial \lambda} &= V^B(r_B) - V^A(r_A) + (1-\lambda)\frac{\partial V^A(r_A)}{\partial r_A}\frac{\partial r_A}{\partial \lambda} + \lambda\frac{\partial V^B(r_B)}{\partial r_B}\frac{\partial r_B}{\partial \lambda} \\ &= V^B(r_B) - V^A(r_A) + \\ &\quad \frac{p\alpha}{6}\left[\lambda F^B(r_B)r_B^{-5}\sigma_B^6(1-\lambda)^{p-1} - (1-\lambda)F^A(r_A)r_A^{-5}\sigma_A^6\lambda^{p-1}\right]\end{aligned} \tag{4.128}$$

The original GROMOS Lennard-Jones soft-core function [84] uses $p = 2$, but $p = 1$ gives a smoother $\partial H/\partial\lambda$ curve. When the changes between the two states involve both the disappearing and appearing of atoms, it is important that the overlapping of atoms happens around $\lambda = 0.5$. This can usually be achieved with $\alpha \approx 0.7$ for $p = 1$ and $\alpha \approx 1.5$ for $p = 2$.

Another issue which 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` 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.

4.6 Methods

4.6.1 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 respectively two atoms are so-called *first neighbors, second neighbors* and *third neighbors*, (see Fig. 4.15). 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**.

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.

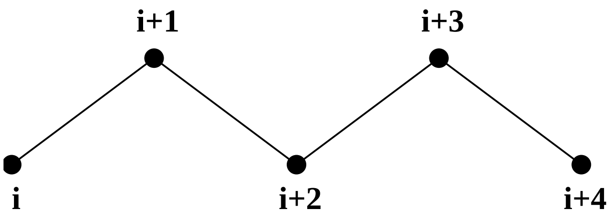


Figure 4.15: Atoms along an alkane chain.

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 [85], 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.

4.6.2 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. When we would apply the 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 so 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*.

4.6.3 Treatment of Cut-offs

GROMACS is quite flexible in treating cut-offs, which implies there can be quite a number of parameters to set. These parameters are set in the input file for grompp. There are two sort of parameters that affect the cut-off interactions; you can select which type of interaction to use in each case, and which cut-offs should be used in the neighbor searching.

For both Coulomb and van der Waals interactions there are interaction type selectors (termed `vdwtype` and `coulombtype`) and two parameters, for a total of six non-bonded interaction parameters. See sec. 7.3 for a complete description of these parameters.

The neighbor searching (NS) can be performed using a single-range, or a twin-range approach. Since the former is merely a special case of the latter we will discuss the more general twin-range. In this case NS is described by two radii `rlist` and max(`rcoulomb`,`rvdw`). Usually one builds the neighbor list every 10 time steps or every 20 fs (parameter `nstlist`). In the neighbor list all interaction pairs that fall within `rlist` are stored. Furthermore, the interactions between pairs that do not fall within `rlist` but do fall within max(`rcoulomb`,`rvdw`) are computed during NS, and the forces and energy are stored separately, and added to short-range forces at every time step between successive NS. If `rlist` = max(`rcoulomb`,`rvdw`), no forces are evaluated during neighbor list generation. The virial is calculated from the sum of the short- and long-range forces. This means that the virial can be slightly asymmetrical at non-NS steps. In single precision the virial is almost always asymmetrical, because the off-diagonal elements are about as large as each element in the sum. In most cases this is not really a problem, since the fluctuations in the virial can be 2 orders of magnitude larger than the average.

Except for the plain cut-off, all of the interaction functions in Table 4.2 require that neighbor searching is done with a larger radius than 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).

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

Table 4.2: Parameters for the different functional forms of the non-bonded interactions.

4.7 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 $\boldsymbol{r}_s$ as a function of the positions of other particles $\boldsymbol{r}_i$: $\boldsymbol{r}_s = f(\boldsymbol{r}_1..\boldsymbol{r}_n)$. The virtual site, which may carry charge, or can 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. [86]. We can write the potential energy as

$$V = V(\boldsymbol{r}_s, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_n) = V^*(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n) \tag{4.129}$$

The force on the particle i is then

$$\boldsymbol{F}_i = -\frac{\partial V^*}{\partial \boldsymbol{r}_i} = -\frac{\partial V}{\partial \boldsymbol{r}_i} - \frac{\partial V}{\partial \boldsymbol{r}_s}\frac{\partial \boldsymbol{r}_s}{\partial \boldsymbol{r}_i} = \boldsymbol{F}_i^{direct} + \boldsymbol{F}'_i \tag{4.130}$$

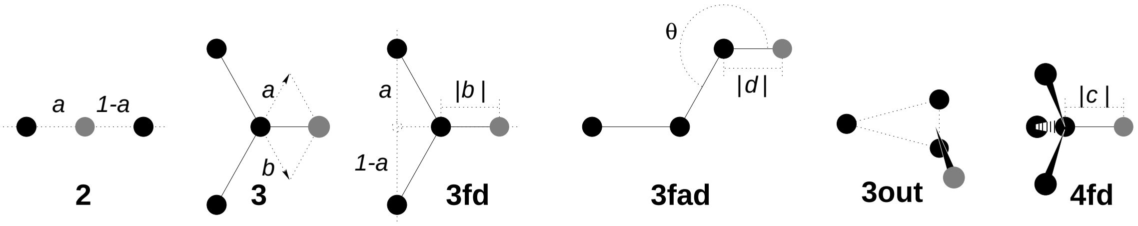


Figure 4.16: The six different types of virtual site construction in GROMACS. The constructing atoms are shown as black circles, the virtual sites in gray.

the first term of which 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:

$$\boldsymbol{F}'_i = \begin{bmatrix} \dfrac{\partial x_s}{\partial x_i} & \dfrac{\partial y_s}{\partial x_i} & \dfrac{\partial z_s}{\partial x_i} \\ \dfrac{\partial x_s}{\partial y_i} & \dfrac{\partial y_s}{\partial y_i} & \dfrac{\partial z_s}{\partial y_i} \\ \dfrac{\partial x_s}{\partial z_i} & \dfrac{\partial y_s}{\partial z_i} & \dfrac{\partial z_s}{\partial z_i} \end{bmatrix} \boldsymbol{F}_s \tag{4.131}$$

where $\boldsymbol{F}_s$ 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 [86].

As a further note, the computation of the virial (eqn. 3.19) is non-trivial when virtual sites are used. Since the virial involves a summation over all the atoms (rather than virtual sites) the forces most be redistributed from the virtual sites to the atoms (using eqn. 4.131) *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.

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. 4.16 depicts 6 of the available virtual site constructions. The conceptually simplest construction types are linear combinations:

$$\boldsymbol{r}_s = \sum_{i=1}^{N} w_i \, \boldsymbol{r}_i \tag{4.132}$$

The force is then redistributed using the same weights:

$$\boldsymbol{F}'_i = w_i \, \boldsymbol{F}_s \tag{4.133}$$

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.

2. As a linear combination of two atoms (Fig. 4.16 2):

$$w_i = 1 - a\ , \quad w_j = a \tag{4.134}$$

In this case the virtual site is on the line through atoms i and j.

3. As a linear combination of three atoms (Fig. 4.16 3):

$$w_i = 1 - a - b\ , \quad w_j = a\ , \quad w_k = b \tag{4.135}$$

In this case the virtual site is in the plane of the other three particles.

3fd. In the plane of three atoms, with a fixed distance (Fig. 4.16 3fd):

$$\boldsymbol{r}_s \; = \; \boldsymbol{r}_i + b\frac{\boldsymbol{r}_{ij} + a\boldsymbol{r}_{jk}}{|\boldsymbol{r}_{ij} + a\boldsymbol{r}_{jk}|} \tag{4.136}$$

In this case the virtual site is in the plane of the other three particles at a distance of $|b|$ from i. The force on particles i, j and k due to the force on the virtual site can be computed as:

$$\begin{array}{rcl} \boldsymbol{F}'_i & = & \boldsymbol{F}_s - \gamma(\boldsymbol{F}_s - \boldsymbol{p}) \\ \boldsymbol{F}'_j & = & (1-a)\gamma(\boldsymbol{F}_s - \boldsymbol{p}) \\ \boldsymbol{F}'_k & = & a\gamma(\boldsymbol{F}_s - \boldsymbol{p}) \end{array} \quad \text{where} \quad \begin{array}{c} \gamma = \dfrac{b}{|\boldsymbol{r}_{ij} + a\boldsymbol{r}_{jk}|} \\ \boldsymbol{p} = \dfrac{\boldsymbol{r}_{is}\cdot\boldsymbol{F}_s}{\boldsymbol{r}_{is}\cdot\boldsymbol{r}_{is}}\boldsymbol{r}_{is} \end{array} \tag{4.137}$$

3fad. In the plane of three atoms, with a fixed angle and distance (Fig. 4.16 3fad):

$$\boldsymbol{r}_s \; = \; \boldsymbol{r}_i + d\cos\theta\frac{\boldsymbol{r}_{ij}}{|\boldsymbol{r}_{ij}|} + d\sin\theta\frac{\boldsymbol{r}_\perp}{|\boldsymbol{r}_\perp|} \quad \text{where} \quad \boldsymbol{r}_\perp \; = \; \boldsymbol{r}_{jk} - \frac{\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{jk}}{\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{ij}}\boldsymbol{r}_{ij} \tag{4.138}$$

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 $\boldsymbol{r}_{ij}$. 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. 5.2.2). The force on particles i, j and k due to the force on the virtual site can be computed as (with $\boldsymbol{r}_\perp$ as defined in eqn. 4.138):

$$\begin{array}{rcrcl} \boldsymbol{F}'_i & = & \boldsymbol{F}_s & - & \dfrac{d\cos\theta}{|\boldsymbol{r}_{ij}|}\boldsymbol{F}_1 + \dfrac{d\sin\theta}{|\boldsymbol{r}_\perp|}\left(\dfrac{\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{jk}}{\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{ij}}\boldsymbol{F}_2 + \boldsymbol{F}_3\right) \\ \boldsymbol{F}'_j & = & & & \dfrac{d\cos\theta}{|\boldsymbol{r}_{ij}|}\boldsymbol{F}_1 - \dfrac{d\sin\theta}{|\boldsymbol{r}_\perp|}\left(\boldsymbol{F}_2 + \dfrac{\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{jk}}{\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{ij}}\boldsymbol{F}_2 + \boldsymbol{F}_3\right) \\ \boldsymbol{F}'_k & = & & & \dfrac{d\sin\theta}{|\boldsymbol{r}_\perp|}\boldsymbol{F}_2 \end{array}$$

$$\text{where} \quad \boldsymbol{F}_1 = \boldsymbol{F}_s - \frac{\boldsymbol{r}_{ij}\cdot\boldsymbol{F}_s}{\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{ij}}\boldsymbol{r}_{ij}\ , \quad \boldsymbol{F}_2 = \boldsymbol{F}_1 - \frac{\boldsymbol{r}_\perp\cdot\boldsymbol{F}_s}{\boldsymbol{r}_\perp\cdot\boldsymbol{r}_\perp}\boldsymbol{r}_\perp \;\text{and}\; \boldsymbol{F}_3 = \frac{\boldsymbol{r}_{ij}\cdot\boldsymbol{F}_s}{\boldsymbol{r}_{ij}\cdot\boldsymbol{r}_{ij}}\boldsymbol{r}_\perp \tag{4.139}$$

3out. As a non-linear combination of three atoms, out of plane (Fig. 4.16 3out):

$$\boldsymbol{r}_s \; = \; \boldsymbol{r}_i + a\boldsymbol{r}_{ij} + b\boldsymbol{r}_{ik} + c(\boldsymbol{r}_{ij} \times \boldsymbol{r}_{ik}) \tag{4.140}$$

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:

$$\begin{aligned}
\boldsymbol{F}'_j &= \begin{bmatrix} a & -c\,z_{ik} & c\,y_{ik} \\ c\,z_{ik} & a & -c\,x_{ik} \\ -c\,y_{ik} & c\,x_{ik} & a \end{bmatrix} \boldsymbol{F}_s \\
\boldsymbol{F}'_k &= \begin{bmatrix} b & c\,z_{ij} & -c\,y_{ij} \\ -c\,z_{ij} & b & c\,x_{ij} \\ c\,y_{ij} & -c\,x_{ij} & b \end{bmatrix} \boldsymbol{F}_s \\
\boldsymbol{F}'_i &= \boldsymbol{F}_s - \boldsymbol{F}'_j - \boldsymbol{F}'_k
\end{aligned} \tag{4.141}$$

4fd. From four atoms, with a fixed distance (Fig. 4.16 4fd):

$$\boldsymbol{r}_s = \boldsymbol{r}_i + c\frac{\boldsymbol{r}_{ij} + a\boldsymbol{r}_{jk} + b\boldsymbol{r}_{jl}}{|\boldsymbol{r}_{ij} + a\boldsymbol{r}_{jk} + b\boldsymbol{r}_{jl}|} \tag{4.142}$$

In this case the virtual site is at a distance of $|c|$ from i. The force on particles i, j, k and l due to the force on the virtual site can be computed as:

$$\begin{aligned}
\boldsymbol{F}'_i &= \boldsymbol{F}_s - \gamma(\boldsymbol{F}_s - \boldsymbol{p}) \\
\boldsymbol{F}'_j &= (1-a-b)\gamma(\boldsymbol{F}_s - \boldsymbol{p}) \\
\boldsymbol{F}'_k &= a\gamma(\boldsymbol{F}_s - \boldsymbol{p}) \\
\boldsymbol{F}'_l &= b\gamma(\boldsymbol{F}_s - \boldsymbol{p})
\end{aligned}
\quad \text{where} \quad
\begin{aligned}
\gamma &= \frac{c}{|\boldsymbol{r}_{ij} + a\boldsymbol{r}_{jk} + b\boldsymbol{r}_{jl}|} \\
\boldsymbol{p} &= \frac{\boldsymbol{r}_{is}\cdot\boldsymbol{F}_s}{\boldsymbol{r}_{is}\cdot\boldsymbol{r}_{is}}\boldsymbol{r}_{is}
\end{aligned} \tag{4.143}$$

N. A linear combination of N atoms with relative weights a_i. The weight for atom i is:

$$w_i = a_i \left(\sum_{j=1}^{N} a_j\right)^{-1} \tag{4.144}$$

There are three options for setting the weights:

COG center of geometry: equal weights

COM 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

COW center of weights: a_i is defined by the user

4.8 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, energy and pressure corrections both are negative. While the energy correction is usually small, it may be important for free energy calculations. The pressure correction in contrast is very large and can not be neglected. Although it is in principle possible to parameterize a force field such that the pressure is close to 1 bar even without correction, such a method makes the parameterization dependent on the cut-off and is therefore undesirable. Please note that it is not consistent to use the long range correction to the dispersion without using either a reaction field method or a proper long range electrostatics method such as Ewald summation or PPPM.

4.8.1 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:

$$V(r_{ij}) \;=\; -\,C_6\,r_{ij}^{-6} \tag{4.145}$$

and the corresponding force is

$$\boldsymbol{F}_{ij} \;=\; -\,6\,C_6\,r_{ij}^{-8}\boldsymbol{r}_{ij} \tag{4.146}$$

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 $\boldsymbol{F}_c$. The long-range contribution to the dispersion energy in a system with N particles and particle density $\rho = N/V$ is:

$$V_{lr} \;=\; \frac{1}{2}N\rho\int_0^{\infty} 4\pi r^2 g(r)\,(V(r)-V_c(r))\,\mathrm{d}r \tag{4.147}$$

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 eqn. 4.147 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:

$$\begin{aligned} V_{lr} &= \frac{1}{2}N\left(\rho\int_0^{r_1} 4\pi r^2 g(r)\,C_6\,S\,\mathrm{d}r + \rho\int_{r_1}^{r_c} 4\pi r^2\,(V(r)-V_c(r))\,\mathrm{d}r + \rho\int_{r_c}^{\infty} 4\pi r^2 V(r)\,\mathrm{d}r\right) \\ &= \frac{1}{2}N\left(\left(\frac{4}{3}\pi\rho r_1^3 - 1\right)C_6\,S + \rho\int_{r_1}^{r_c} 4\pi r^2\,(V(r)-V_c(r))\,\mathrm{d}r - \frac{4}{3}\pi N\rho\,C_6\,r_c^{-3}\right) \end{aligned} \tag{4.148}$$

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 [87]:

$$V_{lr} \;=\; -\frac{2}{3}\pi N\rho\,C_6\,r_c^{-3} \tag{4.149}$$

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*:

$$\langle C_6\rangle = \frac{2}{N(N-1)}\sum_i^N\sum_{j>i}^N C_6(i,j) \tag{4.150}$$

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.

4.8.2 Virial and pressure

The scalar virial of the system due to the dispersion interaction between two particles i and j is given by:

$$\Xi \;=\; -\frac{1}{2}\boldsymbol{r}_{ij} \cdot \boldsymbol{F}_{ij} \;=\; 3\,C_6\,r_{ij}^{-6} \tag{4.151}$$

The pressure is given by:

$$P \;=\; \frac{2}{3\,V}\left(E_{kin} - \Xi\right) \tag{4.152}$$

The long-range correction to the virial is given by:

$$\Xi_{lr} \;=\; \frac{1}{2}N\rho\int_0^{\infty} 4\pi r^2 g(r)(\Xi - \Xi_c)\,\mathrm{d}r \tag{4.153}$$

We can again integrate the long range contribution to the virial assuming $g(r)$ is 1 beyond r_1:

$$\begin{aligned}
\Xi_{lr} \;&=\; \frac{1}{2}N\rho\left(\int_{r_1}^{r_c} 4\pi r^2(\Xi - \Xi_c)\,\mathrm{d}r + \int_{r_c}^{\infty} 4\pi r^2 3\,C_6\,r_{ij}^{-6}\,\mathrm{d}r\right)\\
&=\; \frac{1}{2}N\rho\left(\int_{r_1}^{r_c} 4\pi r^2(\Xi - \Xi_c)\,\mathrm{d}r + 4\pi C_6\,r_c^{-3}\right)
\end{aligned} \tag{4.154}$$

For a plain cut-off the correction to the pressure is [87]:

$$P_{lr} \;=\; -\frac{4}{3}\pi C_6\,\rho^2 r_c^{-3} \tag{4.155}$$

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 $\langle C_6 \rangle$ (eqn. 4.150):

$$P_{lr} \;=\; -\frac{4}{3}\pi\left\langle C_6 \right\rangle \rho^2 r_c^{-3} \tag{4.156}$$

For inhomogeneous systems eqn. 4.156 can be applied under the same restriction as holds for the energy (see sec. 4.8.1).

4.9 Long Range Electrostatics

4.9.1 Ewald summation

The total electrostatic energy of N particles and the periodic images are given by

$$V = \frac{f}{2}\sum_{n_x}\sum_{n_y}\sum_{n_{z*}}\sum_{i}^{N}\sum_{j}^{N}\frac{q_i q_j}{\mathbf{r}_{ij,\mathbf{n}}}. \tag{4.157}$$

$(n_x, n_y, n_z) = \mathbf{n}$ 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 [88]. The idea is to convert the single slowly-converging sum eqn. 4.157 into two quickly-converging terms and a constant term:

$$V = V_{dir} + V_{rec} + V_0 \tag{4.158}$$

$$V_{dir} = \frac{f}{2} \sum_{i,j}^{N} \sum_{n_x} \sum_{n_y} \sum_{n_{z*}} q_i q_j \frac{\mathrm{erfc}(\beta r_{ij,\mathbf{n}})}{r_{ij,\mathbf{n}}} \tag{4.159}$$

$$V_{rec} = \frac{f}{2\pi V} \sum_{i,j}^{N} q_i q_j \sum_{m_x} \sum_{m_y} \sum_{m_{z*}} \frac{\exp\left(-(\pi \mathbf{m}/\beta)^2 + 2\pi i \mathbf{m} \cdot (\mathbf{r}_i - \mathbf{r}_j)\right)}{\mathbf{m}^2} \tag{4.160}$$

$$V_0 = -\frac{f\beta}{\sqrt{\pi}} \sum_{i}^{N} q_i^2, \tag{4.161}$$

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` 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 `fourierspacing` parameter times the box dimensions 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.

4.9.2 PME

Particle-mesh Ewald is a method proposed by Tom Darden [10, 11] to improve the performance of the reciprocal sum. Instead of directly summing wave vectors, the charges are assigned to a grid using cardinal B-spline interpolation. This 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 (3.17.5).

Using PME

To use Particle-mesh Ewald summation in GROMACS, specify the following lines in your `.mdp` 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 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.

4.9.3 PPPM

The Particle-Particle Particle-Mesh methods of Hockney & Eastwood can also be applied in GROMACS for the treatment of long range electrostatic interactions [89, 10, 90]. With this algorithm the charges of all particles are spread over a grid of dimensions (n_x,n_y,n_z) using a weighting function called the triangle-shaped charged distribution:

$$\begin{aligned} W(\boldsymbol{r}) &= W(x)\,W(y)\,W(z) \\ W(\xi) &= \begin{cases} \frac{3}{4} - \left(\frac{\xi}{h}\right)^2 & |\xi| \le \frac{h}{2} \\ \frac{1}{2}\left(\frac{3}{2} - \frac{|\xi|}{h}\right)^2 & \frac{h}{2} < |\xi| < \frac{3h}{2} \\ 0 & \frac{3h}{2} \le |\xi| \end{cases} \end{aligned} \tag{4.162}$$

where ξ (is x, y or z) is the distance to a grid point in the corresponding dimension. Only the 27 closest grid points need to be taken into account for each charge.

Then, this charge distribution is Fourier transformed using a 3D inverse FFT routine. In Fourier space a convolution with function $\hat{G}$ is performed:

$$\hat{G}(k) = \frac{\hat{g}(k)}{\epsilon_0 k^2} \tag{4.163}$$

where $\hat{g}$ is the Fourier transform of the charge spread function g(r). This yield the long range potential $\hat{\phi}(k)$ on the mesh, which can be transformed using a forward FFT routine into the real space potential. Finally the potential and forces are retrieved using interpolation [90]. It is not easy to calculate the full long-range virial tensor with PPPM, but it is possible to obtain the trace. This means that the sum of the pressure components is correct (and therefore the isotropic pressure) but not necessarily the individual pressure components!

Using PPPM

To use the PPPM algorithm in GROMACS, specify the following lines in your `.mdp` file:

```
coulombtype = PPPM
rlist = 1.0
rcoulomb = 0.85
rcoulomb_switch = 0.0
rvdw = 1.0
fourierspacing = 0.075
```

For details on the switch parameters see the section on modified long-range interactions in this manual. When using PPPM we recommend to take at most 0.075 nm per grid point (*e.g.* 20 grid points for 1.5 nm). PPPM does not provide the same accuracy as PME but can be slightly faster in some cases. Due to the problem with the pressure tensor you shouldn't use it with pressure coupling.

We're somewhat ambivalent about PPPM, so if you use it please contact us - otherwise it might be removed from future releases so we can concentrate our efforts on PME.

4.9.4 Optimizing Fourier transforms

To get the best possible performance you should try to avoid large prime numbers for grid dimensions. The FFT code used in GROMACS is optimized for grid sizes of the form $2^a 3^b 5^c 7^d 11^e 13^f$, where $e + f$ is 0 or 1 and the other exponents arbitrary. (See further the documentation of the FFT algorithms at www.fftw.org.

It is also possible to optimize the transforms for the current problem by performing some calculations at the start of the run. This is not done per default since it takes a couple of minutes, but for large runs it will save time. Turn it on by specifying

```
optimize_fft = yes
```

in your `.mdp` file.

When running in parallel the grid must be communicated several times and thus hurting scaling performance. With PME you can improve this by increasing grid spacing while simultaneously increasing the interpolation to *e.g.* sixth order. Since the interpolation is entirely local a this will improve the scaling in most cases.

4.10 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 4.5 includes several force fields, and additional ones are available on the website. If you do not know which one to select we recommend Gromos96 for united-atom setups and OPLS-AA/L for all-atom parameters. That said, we describe the available options in some detail.

4.10.1 GROMOS87

The GROMOS-87 suite of programs and corresponding force field[69] formed the basis for the development of GROMACS in the early 1990s. The original GROMOS87 force field is not available in GROMACS. In previous versions ($< 3.3.2$) there used to be the so-called GROMACS force field which was based on GROMOS-87 [69], with a small modification concerning the interaction between water-oxygens and carbon atoms [91, 92], as well as 10 extra atom types [93, 94, 91, 92, 95]. Whenever using this force field, please cite the above references, and do not call it GROMACS force field, instead name it GROMOS-87 [69] with corrections as detailed in [91, 92].

All-hydrogen force-field

The GROMACS all-hydrogen force-field is almost identical to the normal GROMACS 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.

4.10.2 GROMOS-96

GROMACS supports the GROMOS-96 force fields [68]. All parameters for the 43a1, 43a2 (development, improved alkane dihedrals) and 43b1 (vacuum) force fields are included. All standard building blocks are included and topologies can be build automatically by `pdb2gmx`. The GROMOS-96 force field is a further development of the GROMOS-87 force field on which the GROMACS force field is based. The GROMOS-96 force field has improvements over the GROMACS force field for proteins and small molecules. It is not, however, recommended for use with long alkanes and lipids. The GROMOS-96 force field differs from the GROMACS force field in a few aspects:

- the force field parameters
- the parameters for the bonded interactions are not linked to atom types
- a fourth power bond stretching potential (sec. 4.2.1)
- an angle potential based on the cosine of the angle (sec. 4.2.5)

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 with 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 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`. Such a file 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 [68] for a complete description of the blocks. Note that all GROMACS programs can read compressed (.Z) or gzipped (.gz) files.

4.10.3 OPLS/AA

4.10.4 Amber

4.10.5 CHARMM

As of version 4.5, GROMACS supports the CHARMM27 force field for proteins [96, 97], lipids [98] and nucleic acids [99]. 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) and a fast implicit solvent recently implemented [63] – and the details and results are presented in the paper by Bjelkmar et al. [100]. 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 the default option is to use CMAP (dihedral cross terms for protein backbone), use -nocmap flag otherwise.

4.10.6 Martini

Chapter 5

Topologies

5.1 Introduction

GROMACS must know on which atoms and combinations of atoms the various contributions to the potential functions (see chapter 4) must act. It must also know what parameters must be applied to the various functions. All this is described in the *topology* file `*.top`, 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, but these do not strictly belong to the molecular topology.

This Chapter describes the set up of the topology file, the `*.top` 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 5.8.1 describes the organization of the force-field files.

Note: if you construct your own topologies, we encourage you to upload them to our topology archive at www.gromacs.org! Just imagine how thankful you'd have been if your topology had been available there before you started. The same goes for new force field or modified versions of the standard force fields - contribute them to the force field archive!

5.2 Particle type

In GROMACS there are 5 types of particles, see Table 5.1. Only regular atoms and virtual interaction-sites are used in GROMACS; shells are necessary for polarizable models like the Shell-Water models [38].

Particle	Symbol
atoms	A
shells	S
virtual interaction-sites	V (or D)

Table 5.1: Particle types in GROMACS

5.2.1 Atom types

Depending on the force field GROMACS uses different atom types, a sample from the deprecated "gromacs" force field is listed below, with their corresponding masses (in a.m.u.). This is the same listing as in the file `ff???.atp` (.atp = **a**tom **t**ype **p**arameter file), therefore in this file you can change and/or add an atom type.

O	15.99940	; carbonyl oxygen (C=O)
OM	15.99940	; carboxyl oxygen (CO-)
OA	15.99940	; hydroxyl oxygen (OH)
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)
NR5	14.00670	; aromatic N (5-ring,2 bonds)
NR5*	14.00670	; aromatic N (5-ring,3 bonds)
NP	14.00670	; porphyrin nitrogen
C	12.01100	; bare carbon (peptide,C=O,C-N)
CH1	13.01900	; aliphatic CH-group
CH2	14.02700	; aliphatic CH2-group
CH3	15.03500	; aliphatic CH3-group

Atomic detail is used except for hydrogen atoms bound to (aliphatic) carbon atoms, which are treated as *united atoms*. No special hydrogen-bond term is included. Note that other force field like OPLS/AA and Amber99 use all atoms.

For backward compatibility we retain here some reference to parameters present in the "gromacs" force field. The last 10 atom types were not part of the original GROMOS-87 force field [69] and when you use them you can refer to one or more of the following papers:

- F was taken from ref. [94],
- CP2 and CP3 from ref. [91] and references cited therein,
- CR5, CR6 and HCR from ref. [101]
- OWT3 from ref. [93]
- SD, OD and CD from ref. [95]

Note that we recommend against using these parameters in new projects since they are not well-tested.

Note: GROMACS makes use of the atom types as a name, *not* as a number (as *e.g.* in GROMOS).

5.2.2 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. 4.7.

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 4fd (the different types are explained in sec. 4.7).

Parameters for type 2 should look like this:

```
[ virtual_sites2 ]
; Site from funct a
5 1 2 1 0.7439756
```

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 4fd like this:

```
[ virtual_sites4 ]
; Site from funct a b d
5 1 2 3 4 1 0.33333 0.33333 -0.105
```

This will result in the construction of a virtual site, number 5 (first column '`Site`'), based on the positions of 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

Property	Symbol	Unit
Type	-	-
Mass	m	a.m.u.
Charge	q	electron
epsilon	ϵ	kJ/mol
sigma	σ	nm

Table 5.2: Static atom type properties in GROMACS

specified (last one, two or three columns '`a b ..`').

Note that if any constant bonded interactions defined between virtual sites and/or normal atoms will be removed by `grompp`, this happens after the exclusions have been generated. This way, exclusions will not be affected by an atom being defined as virtual site or not, but by the bonding configuration of the atom.

5.3 Parameter files

5.3.1 Atoms

A number of *static* properties are assigned to the atom types in the GROMACS force field: Type, Mass, Charge, ϵ and σ (see Table 5.2 The mass is listed in `ff???.atp` (see 5.2.1), whereas the charge is listed in `ff???.rtp` (.rtp = **r**esidue **t**opology **p**arameter file, see 5.6.1). This implies that the charges are only defined in the building blocks of amino acids or user defined building blocks. When generating a topology (`*.top`) using the `pdb2gmx` program the information from these files is combined.

The following *dynamic* quantities are associated with an atom

- Position **x**
- Velocity **v**

These quantities are listed in the coordinate file, `*.gro` (see section File format, 5.7.6).

5.3.2 Bonded parameters

The bonded parameters (*i.e.* bonds, bond angles, improper and proper dihedrals) are listed in `ff???bon.itp`. The term `func` is 1 for harmonic and 2 for GROMOS-96 bond and angle potentials. For the dihedral, this is explained after this listing.

```
[ 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
```

Also in this file are the Ryckaert-Bellemans [102] parameters for the CP2-CP2 dihedrals in alkanes or alkane tails with the following constants:

$$\begin{array}{rcrrcrrcr} & & & & \text{(kJ/mol)} & & & & \\ C_0 & = & 9.28 & C_2 & = & -13.12 & C_4 & = & 26.24 \\ C_1 & = & 12.16 & C_3 & = & -\,3.06 & C_5 & = & -31.5 \end{array}$$

(**Note:** The use of this potential implies the exclusion of LJ interactions between the first and the last atom of the dihedral, and ψ is defined according to the 'polymer convention' ($\psi_{trans} = 0$)).

So there are three types of dihedrals in the GROMACS force field:

- proper dihedral : funct = 1, with mult = multiplicity, so the number of possible angles
- improper dihedral : funct = 2
- Ryckaert-Bellemans dihedral : funct = 3

In the file `ff???bon.itp` you can add bonded parameters. If you want to include parameters for new atom types, make sure you define this new atom type in `ff???.atp` as well.

5.3.3 Non-bonded parameters

The non-bonded parameters consist of the van der Waals parameters V (`c6`) and W (`c12`), as listed in the file `ff???nb.itp`, where `ptype` is the particle type (see Table 5.1):

```
[ atomtypes ]
;name mass charge ptype c6 c12
O 15.99940 0.000 A 0.22617E-02 0.74158E-06
OM 15.99940 0.000 A 0.22617E-02 0.74158E-06
.....

[ nonbond_params ]
; i j func c6 c12
O O 1 0.22617E-02 0.74158E-06
O OA 1 0.22617E-02 0.13807E-05
.....

[ 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 parameters V and W can be defined in two different ways, depending on the combination rule that was chosen in the `[ defaults ]` section op the topology file (see 5.7.1):

$$\text{for combination rule 1 :}\quad \begin{array}{lllll} \mathrm{V}_{ii} & = & C_i^{(6)} & = & 4\,\epsilon_i\sigma_i^6 \quad [\,\mathrm{kJ\ mol^{-1}\ nm^6}\,] \\ \mathrm{W}_{ii} & = & C_i^{(12)} & = & 4\,\epsilon_i\sigma_i^{12} \quad [\,\mathrm{kJ\ mol^{-1}\ nm^{12}}\,] \end{array} \tag{5.1}$$

$$\text{for combination rules 2 and 3 :}\quad \begin{array}{lll} \mathrm{V}_{ii} & = & \sigma_i \quad [\,\mathrm{nm}\,] \\ \mathrm{W}_{ii} & = & \epsilon_i \quad [\,\mathrm{kJ\ mol^{-1}}\,] \end{array} \tag{5.2}$$

Some or all combinations for different atom-types can be given in the `[ nonbond_params ]` section. Any combination that is not given will be computed according to the combination rule:

$$\text{for combination rules 1 and 3 :}\quad \begin{array}{lll} C_{ij}^{(6)} & = & \left(C_i^{(6)}\,C_j^{(6)}\right)^{\frac{1}{2}} \\ C_{ij}^{(12)} & = & \left(C_i^{(12)}\,C_j^{(12)}\right)^{\frac{1}{2}} \end{array} \tag{5.3}$$

$$\text{for combination rule 2 :}\quad \begin{array}{lll} \sigma_{ij} & = & \frac{1}{2}(\sigma_i+\sigma_j) \\ \epsilon_{ij} & = & \sqrt{\epsilon_i\,\epsilon_j} \end{array} \tag{5.4}$$

5.3.4 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. 5.4).

The pair interaction parameters for the atom types in `ff???nb.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 which are not present in the `[ pairtypes ]` section are only generated when generate pairs is set to yes in the topology (see 5.7.1). When generate pairs is set to no, `grompp` 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 types 2 and 3 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. 3.12) with all non-bonded parameters, i.e. charges and LJ parameters, of the solute set to zero. But the free-energy difference between the A and B state is not the total hydration free-energy, one has to add the free-energy for reintroducing the internal Coulomb and interactions in the solute. 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. Function type 3 is intended to replace the non-bonded interaction. It uses the unscaled charges and the non-bonded LJ parameters. Type 3 also only uses the A-state parameters. Note that one should add exclusions for all atom pairs participating in pair interactions type 3, otherwise such pairs will also end up in the normal neighbor lists.

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 “14” LJ and Coulomb entries per energy group pair. Energies for type 3 are added to the LJ and Coulomb SR terms.

5.4 Exclusions

The exclusions for non-bonded interactions are generated by `grompp` for neighboring atoms up to a certain number of bonds away, as defined in the `[ moleculetype ]` section in the topology file (see 5.7.1). Particles are considered bonded when they are connected by “chemical” bonds (`[ bonds ]` types 1 to 5, 7 or 8) or constraints (`[ constraints ]` type 1). `[ bonds ]` type 5 can be used to create a connection (chemical bond) between two atoms without creating an interaction. There is a harmonic interaction (`[ bonds ]` type 6) which does not connect the atoms by a chemical bond. There is also a second constraint type (`[ constraints ]` type 2) which fixes the distance, but does not connect the atoms by a chemical bond. For a complete list of all these interactions see Table 5.5.

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. 3.3).

5.5 Constraints

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. 5.4). The distances are constrained using the LINCS or the SHAKE algorithm, which can be selected in the `*.mdp` file. Both types of constraints can be perturbed in free-energy calculations by adding a second constraint distance (see 5.7.5). Several types of bonds and angles (see Table 5.5) can be converted automatically to constraints by `grompp`. There are several options for this in the `*.mdp` file.

We have also implemented the SETTLE algorithm [40] which is an analytical solution of SHAKE specifically for water. SETTLE can be selected in the topology file. Check 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 section `[ settles ]` defines the first atom of the watery molecule. The settle funct is always one, 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 [93]. 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 pdb2gmx input files

The GROMACS program `pdb2gmx` generates topologies from an input coordinate file. Several formats are supported for the coordinate file, but pdb is the most commonly used format (hence the name `pdb2gmx`). `pdb2gmx` searches for force field in the GROMACS `share/top` directory your working directory. Force fields are recognized from the file `forcefield.itp` in a directory with the extension `.ff`. When a file `forcefield.doc` is present, the first line in this file, which should be a short description of the force field, will be printed to help the user in choosing a force field. Two general files are read by `pdb2gmx`, an atom type file (extension `.atp` 5.2.1) from the force field directory and a file called `aminoacids.dat` from the GROMACS `share/top` directory, which determines which residue names are considered aminoacids. `pdb2gmx` 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` file, which contains the topologies of the building blocks, sn mandatory. Information from other files will only be used for building blocks that come from an `.rtp` file with the same base name. The user can add building blocks to a force fields 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` with the `-rtpo` option will allow building blocks in a local file to replace the default ones in the force field.

5.6.1 Residue database

The files holding the residue databases have the extension `.rtp`. 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 file contains information (bonds, charge, 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`, but if changes are needed make them in the `*.top` file (see 5.7.1), or in a `.rtp` file in the working directory as explained in sec. 5.6. But defining topologies of new small molecules is probably easier by writing an include topology file `*.itp` directly. This will be discussed in section 5.7.2. When adding a new protein residue to the database, don't forget to add the residue name to the `aminoacids.dat` file, so that `grompp`, `make_ndx` and analysis tools can recognize the residue as a protein residue (see 8.1.1).

The `.rtp` files are only used by `pdb2gmx`. As mentioned before, the only extra information this program needs from the `.rtp` database is bonds, charges of atoms, charge groups and improper dihedrals, because the rest is read from the coordinate input file (in the case of `pdb2gmx`, a pdb format 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` file. This also holds for molecules in the coordinate file such as phosphate or sulphate ions. 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

[ 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, that indicate 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` for adding missing hydrogens, the atom names defined in the `.rtp` entry should correspond exactly to the naming convention used in the hydrogen database, see 5.6.4. 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. Parameters can be added to bonds, angles, dihedrals and impropers, these parameters override the standard parameters in the `.itp` 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 GROMOS96 `.rtp` files. These strings are copied to the topology file and can be replaced by force field parameters by the C-preprocessor in `grompp` using `#define` statements.

`pdb2gmx` automatically generates all angles. This means that for the GROMACS force field the `[ angles ]` field is only useful for overriding `.itp` parameters. For the GROMOS-96 force field the interaction number off all angles need to be specified.

`pdb2gmx` 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.

`pdb2gmx` 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 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.

5.6.2 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` files is used to convert standard residue names to the force field build block names. If no `.r2b` 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` can contain 2 or 5 columns. The 2-column format has as 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 field. 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` code) through the `.r2b` file and `specbond.dat` files. This convention is only of importance when you are adding residue types to an `.rtp` file. The convention is listed in Table 5.3. For special bonds with, for instance, a heme group, the GROMACS naming convention is introduced through `specbond.dat`, which can subsequently be translated by the `.r2b` file, if required.

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

Table 5.3: Internal GROMACS residue naming convention.

5.6.3 Atom renaming database

Force field often use atom names which do not follow IUPAC or pdb convention. The `.arn` database is used to translate the atom names in the coordinate file to the force field names. Atoms which are not listed keep their names. The file has three columns which contain the building block name, the old atom name and the new atom name respectively. The residue name supports question-mark wildcards, which match a single character.

An additional general atom renaming file called `xlateat.dat` is present in the `share/top` directory, which translated common non-standard atom names in the coordinate file to IUPAC/pdb convention. Thus when writing force fields files, you can assume standard atom names and no further atom name translation is required, except for that from standard atom names to the force field ones.

5.6.4 Hydrogen database

The hydrogen database is stored in `.hdb` files. It contains information for the `pdb2gmx` 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 then 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` file (see 5.6.1) must be the same. The format of the hydrogen database is as follows:

```
; res # additions
# H add type H i j k
```

```
ALA 1
1 1 H N -C CA
ARG 4
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 additions. After that follows one line for each addition, on which we see:

- The number of H atoms added
- The way of adding H atoms, 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. *-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 [103]
    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 degrees
    5. *one tetrahedral hydrogen,* e.g. *C$_3$CH*
       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 i-j-k with angle (n-i-n2)=(n1-i-j)=(n1-i-k)=109.5
    7. *two water hydrogens*
       two hydrogens are generated around atom i according to SPC [71] water geometry. The symmetry axis will alternate between three coordinate axes in both directions
    10. *three water "hydrogens"*
       two hydrogens are generated around atom i according to SPC [71] 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 [93]

11 *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 [104]

- The name of the new H atom
- 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).

5.6.5 Termini database

The termini databases are stored in `???.n.tdb` and `???.c.tdb` for the N- and C-termini respectively. They contain information for the `pdb2gmx` program on how to connect new atoms to existing ones, which atoms should be removed or changed and which bonded interactions should be added. The format of the is as follows (this is an example from `gmx.ff/aminoacids.c.tdb`):

```
[ None ]

[ COO- ]

[ replace ]
C C C 12.011 0.27

[ add ]
2 8 O C CA N
OM 15.9994 -0.635

[ delete ]
O

[ impropers ]
C O1 O2 CA
```

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 `[ None ]` is the first block, corresponding to a terminus that leaves the molecule as it is; `[ COO- ]` is the second terminus type, corresponding to changing the terminal carbon atom into a deprotonated carboxyl group. Block names cannot be any of the following: `replace`, `add`, `delete`, `bonds`, `angles`, `dihedrals`, `impropers`; this would interfere with the parameters of the block, and would probably also be very confusing to human readers.

Per 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` 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 allows 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`. 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 5.6.4), but the possible types of addition are extended by two more, specifically for C-terminal additions:

  8 *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

  9 *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 resp. 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 resp.

  After this line another line follows which 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 5.6.1), when more then 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 `ff???.rtp`, see 5.6.1.

5.7 File formats

5.7.1 Topology file

The topology file is built following the GROMACS specification for a molecular topology. A `*.top` file can be generated by `pdb2gmx`. All possible entries in the topology file are listed in Tables 5.4, 5.5 and 5.6. Also listed are all the units of the parameters, which interactions can be perturbed for free energy calculations, which bonded interactions are used by `grompp` for generating exclusions and which bonded interactions can be converted to constraints by `grompp`.

Description of the file layout:

- semicolon (;) and newline surround comments
- on a line ending with \ the newline character is ignored.
- directives are surrounded by `[` and `]`
- the topology consists of three levels:
  - the parameter level (see Table 5.4)
  - the molecule level, which should contain one or more molecule definitions (see Table 5.5)
  - the system level: `[ system ]`, `[ molecules ]`
- items should be separated by spaces or tabs, not commas
- atoms in molecules should be numbered consecutively starting at 1
- the file is parsed once only 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

Parameters

interaction type	directive	# at.	f. tp	parameters	F. E.
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 5.5, directive `bonds`)			
	`pairtypes`	(see Table 5.5, directive `pairs`)			
	`angletypes`	(see Table 5.5, directive `angles`)			
	`dihedraltypes`$^{(*)}$	(see Table 5.5, directive `dihedrals`)			
	`constrainttypes`	(see Table 5.6, directive `constraints`)			
LJ	`nonbond_params`	2	1	$V^{(a)}$; $W^{(a)}$	
Buckingham	`nonbond_params`	2	2	a (kJ mol^{-1}); b (nm^{-1}); c_6 (kJ mol^{-1}nm^6)	

Molecule definition(s)

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 Tables 5.5 and 5.6					

System

mandatory	`system`			system name	
mandatory	`molecules`			molecule name; number of molecules	

'# at' is the number of atom types
'f. tp' is function type
'F. E.' indicates which parameters can be interpolated during free energy calculations
$^{(cr)}$ the combination rule determines the type of LJ parameters, see 5.3.3
$^{(*)}$ for `dihedraltypes` one can specify 4 atoms or the inner (outer for improper) 2 atoms
$^{(nrexcl)}$ 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 5.4: The topology (`*.top`) file.

Intra-molecular interaction definitions

interaction type	directive	# at.	f. tp	parameters	F. E.
bond	`bonds`$^{(excl,con)}$	2	1	b_0 (nm); k_b (kJ mol^{-1}nm^{-2})	all
G96 bond	`bonds`$^{(excl,con)}$	2	2	b_0 (nm); k_b (kJ mol^{-1}nm^{-4})	all
morse	`bonds`$^{(excl,con)}$	2	3	b_0 (nm); D (kJ mol^{-1}); β (nm^{-1})	
cubic bond	`bonds`$^{(excl,con)}$	2	4	b_0 (nm); $C_{i=2,3}$ (kJ mol^{-1}nm^{-i});	
connection	`bonds`$^{(excl)}$	2	5		
harmonic pot.	`bonds`	2	6	b_0 (nm); k_b (kJ mol^{-1}nm^{-2})	all
FENE bond	`bonds`$^{(excl)}$	2	7	b_m (nm); k_b (kJ mol^{-1}nm^{-2})	
tab. bond	`bonds`$^{(excl)}$	2	8	table number ($\geq$ 0); k (kJ mol^{-1})	k
tab. bond n.c.	`bonds`	2	9	table number ($\geq$ 0); k (kJ mol^{-1})	k
restraint pot.	`bonds`	2	10	low, up$_1$, up$_2$ (nm); k_{dr} (kJ mol^{-1}nm^{-2})	all
LJ/Coul. 1-4	`pairs`	2	1	$V^{(cr)}$; $W^{(cr)}$	all
LJ/Coul. 1-4	`pairs`	2	2	fudge QQ (); q_i, q_j (e), $V^{(cr)}$; $W^{(cr)}$	
LJ/C. pair NB	`pairs_nb`	2	1	q_i, q_j (e); $V^{(cr)}$; $W^{(cr)}$	
angle	`angles`$^{(con)}$	3	1	θ_0 (deg); k_θ (kJ mol^{-1}rad^{-2})	all
G96 angle	`angles`$^{(con)}$	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`$^{(con)}$	3	5	θ_0 (deg); k_θ (kJ mol^{-1}); r_{13} (nm); k_{UB} (kJ mol^{-1})	
quartic angle	`angles`$^{(con)}$	3	6	θ_0 (deg); $C_{i=0,1,2,3,4}$ (kJ mol^{-1}rad^{-i})	
tab. angle	`angles`	3	8	table number ($\geq$ 0); k (kJ mol^{-1})	k
proper dih.	`dihedrals`	4	1	ϕ_s (deg); k_ϕ (kJ mol^{-1}); multiplicity	ϕ, k
improper dih.	`dihedrals`	4	2	ξ_0 (deg); k_ξ (kJ mol^{-1}rad^{-2})	all
RB dihedral	`dihedrals`	4	3	C_0, C_1, C_2, C_3, C_4, C_5 (kJ mol^{-1})	all
Fourier dih.	`dihedrals`	4	5	C_1, C_2, C_3, C_4 (kJ mol^{-1})	all
tab. dihedral	`dihedrals`	4	8	table number ($\geq$ 0); k (kJ mol^{-1})	k
exclusions	`exclusions`	1		one or more atom indices	

'# at' is the number of atom indices

'f. tp' is function type

'F. E.' indicates which parameters can be interpolated during free energy calculations

$^{(cr)}$ the combination rule determines the type of LJ parameters, see 5.3.3

$^{(excl)}$ used by `grompp` for generating exclusions

$^{(con)}$ can be converted to constraints by `grompp`

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.

Table 5.5: Intra-molecular interaction definitions.

Intra-molecular geometry and restraint definitions

interaction type	directive	# at.	f. tp	parameters	F. E.
constraint	`constraints`$^{(excl)}$	2	1	b_0 (nm)	all
constr. n.c.	`constraints`	2	2	b_0 (nm)	all
settle	`settles`	1	1	d_{OH}, d_{HH} (nm)	
vsite2	`virtual_sites2`	3	1	a ()	
vsite3	`virtual_sites3`	4	1	a, b ()	
vsite3fd	`virtual_sites3`	4	2	a (); d (nm)	
vsite3fad	`virtual_sites3`	4	3	θ (deg); d (nm)	
vsite3out	`virtual_sites3`	4	4	a, b (); c (nm^{-1})	
vsite4fd	`virtual_sites4`	5	1	a, b (); d (nm);	
vsite COG	`virtual_sitesn`	1	1	one or more construc. atom ind.	
vsite COM	`virtual_sitesn`	1	2	one or more construc. atom ind.	
vsite COW	`virtual_sitesn`	1	3	one or more pairs consisting of a construc. atom ind. and weight	
position res.	`position_restraints`	1	1	k_x, k_y, k_z (kJ mol^{-1}nm^{-2})	all
restr. pot.	`bonds`	2	10	low, up$_1$, up$_2$ (nm); k_{dr} (kJ mol^{-1}nm^{-2})	all
distance res.	`distance_restraints`	2	1	type; label; low, up$_1$, up$_2$ (nm); weight ()	
orient. res.	`orientation_restraints`	2	1	exp.; label; α; c (U nm$^{\alpha}$); obs. (U); weight (U^{-1})	
angle res.	`angle_restraints`	4	1	θ_0 (deg); k_c (kJ mol^{-1}); multiplicity	θ, k
angle res. z	`angle_restraints_z`	2	1	θ_0 (deg); k_c (kJ mol^{-1}); multiplicity	θ, k

'# at' is the number of atom indices
'f. tp' is function type
'F. E.' indicates which parameters can be interpolated during free energy calculations
$^{(excl)}$ used by `grompp` for generating exclusions
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.

Table 5.6: Intra-molecular geometry and restraint 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` 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 "ffgmx.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 b0 kb
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 c6 c12
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 th0 cth
1 3 4 1 1.200000e+02 2.928800e+02
1 3 5 1 1.200000e+02 2.928800e+02
4 3 5 1 1.200000e+02 3.347200e+02
1 6 7 1 1.200000e+02 2.928800e+02
1 6 8 1 1.200000e+02 2.928800e+02
7 6 8 1 1.200000e+02 3.347200e+02
2 1 3 1 1.215000e+02 5.020800e+02
2 1 6 1 1.215000e+02 5.020800e+02
3 1 6 1 1.170000e+02 5.020800e+02

[ dihedrals ]
; ai aj ak al funct phi cp mult
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 q0 cq
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

[ 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)

; Include SPC water topology
#include "spc.itp"

[ system ]
Urea in Water

[ molecules ]
;molecule name nr.
Urea 1
SOL 1000
```

Here follows the explanatory text.

`[ defaults ]`**:**

- non-bond type = 1 (Lennard-Jones) or 2 (Buckingham)
- combination rule =
    1. For Lennard Jones: supply $C^{(6)}$ and $C^{(N)}$, $C^M_{ij} = \sqrt{C^M_i \, C^M_j}$ (M = 6,N). Default value for N = 12, but it can be overridden using the last parameter on this line. For Buckingham potentials the combination rule is such that you give the A, B and C parameters. $A_{ij} = \sqrt{A_i \, A_j}$ and similar for C_{ij}, $B_{ij} = 2/(1/B_i + 1/B_j)$.
    2. supply σ and ϵ, $\sigma_{ij} = \frac{1}{2}(\sigma_i + \sigma_j)$ and $\epsilon_{ij} = \sqrt{\epsilon_i \, \epsilon_j}$
    3. supply σ and ϵ, $\sigma_{ij} = \sqrt{\sigma_i \, \sigma_j}$, $\epsilon_{ij} = \sqrt{\epsilon_i \, \epsilon_j}$
- generate pairs = no (the default, get 1-4 parameters from the pairtypes list, when parameters are not present in the list stop with a fatal error) or yes (generate 1-4 parameters which are not present in the pair list from normal Lennard-Jones parameters using FudgeLJ)
- FudgeLJ = factor to multiply Lennard-Jones 1-4 interactions with, default 1
- FudgeQQ = factor to multiply electrostatic 1-4 interactions with, default 1
- N = power for the repulsion term in a 6-N potential (with nonbonded-type Lennard Jones only), starting with GROMACS version 4.1 **mdrun** 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: generate pairs, FudgeLJ, FudgeQQ and N are optional, FudgeLJ is only used when generate pairs is set to 'yes', FudgeQQ is always used. However if you want to specify N you need to give a value for the other parameters as well.

`#include "ffgmx.itp"`**:** this includes the bonded and non-bonded GROMACS parameters, the `gmx` in `ffgmx` will be replaced by the name of the force field you are actually using.

`[ moleculetype ]` : defines the name of your molecule in this `*.top` 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 = 2) and no Ryckaert-Bellemans type dihedrals. If you want to include Ryckaert-Bellemans type dihedrals in a topology, do the following (in case of *e.g.* decane): `[ dihedrals ]`

```
; ai aj ak al funct c0 c1 c2
1 2 3 4 3
2 3 4 5 3
```

and do not forget to *erase the 1-4 interaction* in `[ pairs ]`!

`[ position_restraints ]` : harmonically restrain the selected particles to reference positions (sec. 4.3.1). The reference positions are read from a separate coordinate file by grompp.

`#include "spc.itp"` : includes a topology file that was already constructed (see next section, molecule.itp).

`[ 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`. In this example file it stands for 1 urea molecules dissolved in 1000 water molecules. The molecule type SOL is defined in the `spc.itp` file.

5.7.2 Molecule.itp file

If you construct a topology file you will use frequently (like a water molecule, `spc.itp`) it is better to make a `molecule.itp` file, which only lists the information of the molecule:

```
[ moleculetype ]
; name nrexcl
Urea 3

[ atoms ]
; nr type resnr residu atom cgnr charge
1 C 1 UREA C1 1 0.683
.................
.................
8 H 1 UREA H8 3 0.276

[ bonds ]
; ai aj funct c0 c1
3 4 1 1.000000e-01 3.744680e+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
.................
.................
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
.................
.................
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
.................
.................
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
```

This results in a very short `*.top` file as described in the previous section, but this time you only need to include files:

```
; The force field files to be included
#include "ffgmx.itp"

; Include urea topology
#include "urea.itp"

; Include SPC water topology
#include "spc.itp"

[ system ]
Urea in Water
```

```
[ molecules ]
;molecule name number
Urea 1
SOL 1000
```

5.7.3 Ifdef option

A very powerful feature in GROMACS is the use of `#ifdef` statements in your `*.top` file. By making use of this statement, different parameters for one molecule can be used in the same `*.top` 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.* [105] or by Van Buuren and Berendsen [94]. In fact you can use all the options of the C-Preprocessor, `cpp`, because this is used to scan the file. The way to make use of the `#ifdef` option is as follows:

- in `grompp.mdp` (the GROMACS preprocessor input parameters) use the option
  `define = -DDeloof`
  or
  `define =`
- put the `#ifdef` statements in your `*.top`, as shown below:

```
...

[ atoms ]
; 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

...
```

5.7.4 Topologies for free energy calculations

Free energy differences between two systems A and B can be calculated as described in sec. 3.12. The 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 in the `[ atoms ]` field. 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 5.4, 5.5 and 5.6. The λ-dependence of the interactions is described in section sec. 4.5. Which bonded parameters are used, the one 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 5.7. 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 on in the bonded types, `grompp` uses the A-state parameters and issues a warning.

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 "ffG43a1.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
```

B-state atom types all identical to A-state atom types	parameters on line		parameters in bonded types				message
			A atom types		B atom types		
	A	B	A	B	A	B	
yes	+AB	−	x	x			error
	+A	+B	x	x			
	−	−	−	−			
	−	−	+AB	−			
	−	−	+A	+B			
no	+AB	−	x	x	x	x	warning
	+A	+B	x	x	x	x	
	−	−	−	−	x	x	error
	−	−	+AB	−	−	−	warning
	−	−	+A	+B	−	−	warning
	−	−	+A	x	+B	−	
	−	−	+A	x	+	+B	

Table 5.7: 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.

```
2 3 2 gb_17 gb_26
3 4 2 gb_26 gb_26
4 5 2 gb_26

[ pairs ]
; ai aj funct
1 4 1
2 5 1

[ angles ]
; ai aj ak funct par_A par_B
1 2 3 2 ga_11 ga_14
2 3 4 2 ga_14 ga_14
3 4 5 2 ga_14 ga_14

[ dihedrals ]
; ai aj ak al funct par_A par_B
1 2 3 4 1 gd_12 gd_17
2 3 4 5 1 gd_17 gd_17

[ system ]
; Name
Propanol to Pentane
4
```

```
[ molecules ]
; Compound #mols
PropPent 200
```

Atoms that are not perturbed, `PC2` and `PC3`, do not need B parameter specifications, the B parameters will be copied from the A parameters. Bonded interactions between atoms that are not perturbed do not need B parameter specifications, here this is the case for the last bond. 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 interaction 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
```

5.7.5 Constraint force

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 nanometer 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. 6.3. Below is an example for calculating the constraint force at 0.7 nanometer between two methanes in water, by combining the two methanes into one molecule. The added constraint is of function type 2, which means that it is not used for generating exclusions (see sec. 5.4).

```
; Include force field parameters
#include "ffG43a1.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 "spc.itp"

[ system ]
; Name
Methanes in Water

[ molecules ]
; Compound #mols
Methanes 1
SOL 2002
```

5.7.6 Coordinate file

Files with the `.gro` file extension contain a molecular structure in GROMOS87 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:

```
""for (i=0; (i<natoms); i++)
"residuenr,residuename,atomname,atomnr,x,y,z,vx,vy,vz

"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.

5.8 Force-field organization

5.8.1 Force-field files

GROMACS 4.5 includes five force fields. They are listed the file `FF.dat`:

```
5
ffgmx Gromacs Force field (see manual)
ffgmx2 Gromacs Force field with all hydrogens (proteins only)
ffG43a1 GROMOS96 43a1 Force field (official distribution)
ffG43b1 GROMOS96 43b1 Vacuum Force field (official distribution)
ffG43a2 GROMOS96 43a2 Force field (development) (improved ...)
```

All files for each force field have names beginning with the `ff???` string in the `FF.dat` file. A force field is included at the beginning of a topology file with an `#include` statement followed by `ff???.itp`. This statement includes the force-field file, which in turn may include other force field files. A the five force fields are organized in the same way. As an example we show the `ffgmx.itp` force-field file:

```
#define _FF_GROMACS
#define _FF_GROMACS1

[ defaults ]
; nbfunc comb-rule gen-pairs fudgeLJ fudgeQQ
1 1 no 1.0 1.0

#include "ffgmxnb.itp"
#include "ffgmxbon.itp"
```

The first `#define` can be used in topologies to parse data which is specific for all GROMACS force-fields, the second `#define` to parse data which is specific for this force field. The `defaults` section is explained in 5.7.1. The included file `ffgmxnb.itp` contains all atom types and non-bonded parameters. The included file `ffgmxbon.itp` contains all bonded parameters.

For each force field there a five files which are only used by `pdb2gmx`. These are: the residue database (`.rtp`, see 5.6.1) the hydrogen database (`.hdb`, see 5.6.4), two termini databases (`.tdb`, see 5.6.5) and the atom type database (`.atp`) which contains only the masses.

5.8.2 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 in the `[ moleculetype ]` section (see 5.7.1 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 GROMACS and GROMOS force-fields the Lennard-Jones parameters for several combinations of atom types are not according to the standard combination rules. Such combinations (and possibly also combinations that do follow the combination rules) are defined in the `[ nonbonded_params ]` section and changing the Lennard-Jones parameters of an atom type has no effect on these combinations.

5.8.3 Adding atom types

As of GROMACS version 3.1.3, atom types can be added in an extra `[ atomtypes ]` section after the 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.

Chapter 6

Special Topics

6.1 Potential of mean force

A potential of mean force (PMF) is a potential which 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.
- **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. For a system of two non-interacting masses the potential of mean force is:

$$V_{pmf}(r) = -(n_c - 1)k_B T \log(r) \tag{6.1}$$

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}.

6.2 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[106] to obtain the equilibrium free-energy difference ΔG between two distances from many non-equilibrium simulations:

$$\Delta G_{AB} = -k_B T \log \left\langle e^{-\beta W_{AB}} \right\rangle_A \tag{6.2}$$

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 $\beta = 1/k_B T$.

6.3 The pull code

The pull code applies forces or constraints between the centers of mass of one or more pairs of groups of atoms. There is one reference group and one more other pull groups. Instead of a reference group one can also use absolute reference point in space. The most common situation consists of a reference group and one pull group. In this case the two groups are treated equivalently. The distance between a pair of groups can be determined in 1, 2 or 3 dimension, 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 there mass, but an additional weight factor can also be used.

Three different types of calculation are supported, 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.

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

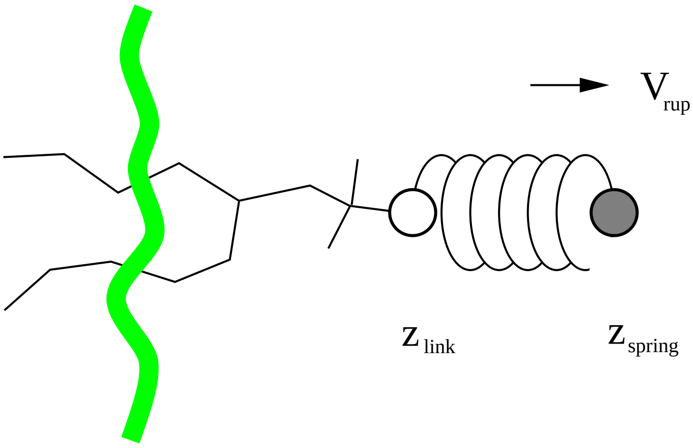


Figure 6.1: Schematic picture of pulling a lipid out of a lipid bilayer with umbrella pulling. V_{rup} is the velocity at which the spring is retracted, Z_{link} is the atom to which the spring is attached and Z_{spring} is the location of the spring.

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 periodic image is used which is closed to the reference atom. 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 this would be closer to center of the group.

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` option `pull_geometry = cylinder` does not use all the atoms of the reference group, but instead dynamically only those within a cylinder with radius `r_1` 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. A second cylinder can be defined with `r_0`, with a linear switch function that weighs the contribution of atoms between `r_0` and `r_1` with distance. This smooths the effects of atoms moving in and out of the cylinder (which causes jumps in the pull forces).

For a group of molecules in a periodic system a plain reference group might not be well defined. An example is a water slab which 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 [107] for details.

When relative weights w_i are used during the calculations, either by supplying weights in the input

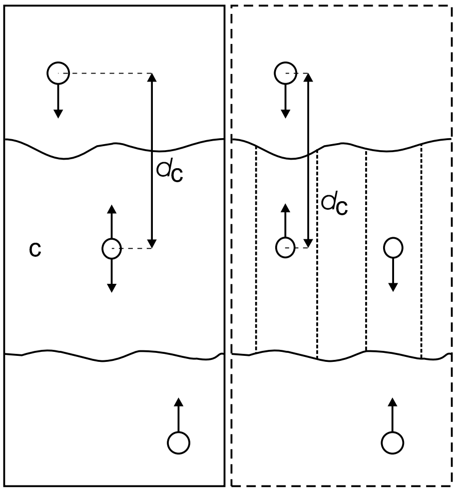


Figure 6.2: Comparison of a plain center of mass reference group versus a cylinder reference group applied to interface systems. C is the reference group. The circles represent the center of mass of two groups plus the reference group, d_c is the reference distance.

or due to cylinder geometry or due to cosine weighting, the weights need to be scaled to conserve momentum:

$$w'_i = w_i \sum_{j=1}^{N} w_j\, m_j \bigg/ \sum_{j=1}^{N} w_j^2\, m_j \tag{6.3}$$

where m_j is the mass of atom j of the group. The mass of the group, required for calculating the constraint force, is:

$$M = \sum_{i=1}^{N} w'_i\, m_i \tag{6.4}$$

The definition of the weighted center of mass is:

$$\mathbf{r}_{com} = \sum_{i=1}^{N} w'_i\, m_i\, \mathbf{r}_i \bigg/ M \tag{6.5}$$

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:

$$\mathbf{F}_i = \frac{w'_i\, m_i}{M}\, \mathbf{F}_{com} \tag{6.6}$$

Limitations

There is one important 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` 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. 6.4. 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 the pull force is small. In such cases the constraint correction displacement of the pull group is small compared to the bond lengths.

6.4 Calculating a PMF using the free-energy code

The free-energy coupling-parameter approach (see sec. 3.12) 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 a special potentials that avoid the generation of extra exclusions, see sec. 5.4). 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` 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 sec. 4.3.1). These positions can be made a function of the coupling parameter λ. The positions for the A and the B state are supplied to `grompp` with the `-r` and `-rb` option, 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 were 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.

6.5 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 than by 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. Where a normal atoms 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. 4.7). 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. 6.5.1 and have previously been described in full detail [108].

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`. Separate options exist to increase hydrogen masses, virtualize all hydrogen atoms or also virtualize all aromatic residues. Note that when all hydrogen atoms are virtualized, also those inside the aromatic residues will be virtualized, *i.e.* hydrogens in the aromatic residues are treated differently depending on the treatment of the aromatic residues.

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` 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` due to the complex nature of the construction needed for a whole aromatic group.

6.5.1 Hydrogen bond-angle vibrations

Construction of virtual sites

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. 6.3).

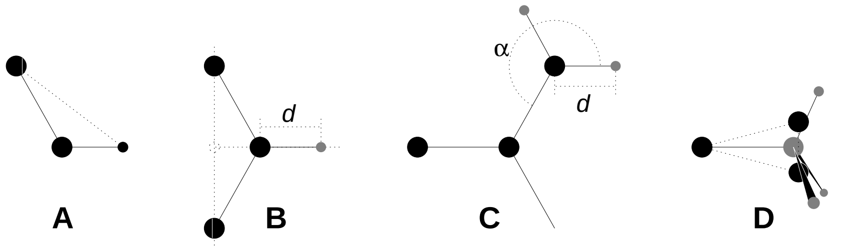


Figure 6.3: The different types of virtual site constructions used for hydrogen atoms. The atoms used in the construction of the virtual site(s) are depicted as black circles, virtual sites as gray ones. Hydrogens are smaller than heavy atoms. A: fixed bond angle, note that here the hydrogen is not a 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.

- *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. 6.3A. This 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. 6.3B.

- *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. 6.3C.

- *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. 6.3D.

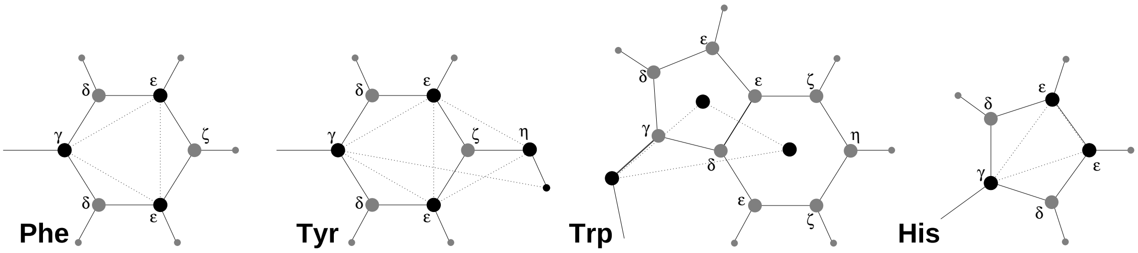


Figure 6.4: The different types of virtual site constructions used for aromatic residues. The atoms used in the construction of the virtual site(s) are depicted as black circles, virtual sites as gray ones. Hydrogens are smaller than heavy atoms. A: phenylalanine; B: tyrosine (note that the hydroxyl hydrogen is *not* a virtual site); C: tryptophan; D: histidine.

6.5.2 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. 4.7) 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. 6.3A.
- *Tyrosine:* The ring is treated identical to the phenylalanine ring. Additionally, constraints are defined between $C_{\epsilon 1}$ and $C_{\epsilon 2}$ 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. 6.3B.
- *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. 6.3C.
- *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. 6.3D.

6.6 Viscosity calculation

The shear viscosity is a property of liquid which can be determined easily by experiment. It is useful for parameterizing the 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:

$$\eta = \frac{1}{2}\frac{V}{k_B T}\lim_{t\to\infty}\frac{\mathrm{d}}{\mathrm{d}t}\left\langle\left(\int_{t_0}^{t_0+t} P_{xz}(t')\mathrm{d}t'\right)^2\right\rangle_{t_0} \tag{6.7}$$

This can be done with `g_energy`. This method converges very slowly [109]. 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 [109]. 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:

$$a_x(z) + \frac{\eta}{\rho}\frac{\partial^2 v_x(z)}{\partial z^2} = 0 \tag{6.8}$$

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:

$$a_x(z) = A\cos\left(\frac{2\pi z}{l_z}\right) \tag{6.9}$$

where l_z is the height of the box. The generated velocity profile is:

$$v_x(z) = V\cos\left(\frac{2\pi z}{l_z}\right) \tag{6.10}$$

$$V = A\frac{\rho}{\eta}\left(\frac{l_z}{2\pi}\right)^2 \tag{6.11}$$

The viscosity can be calculated from A and V:

$$\eta = \frac{A}{V}\rho\left(\frac{l_z}{2\pi}\right)^2 \tag{6.12}$$

In the simulation V is defined as:

$$V = \frac{\displaystyle\sum_{i=1}^{N} m_i v_{i,x} 2\cos\left(\frac{2\pi z}{l_z}\right)}{\displaystyle\sum_{i=1}^{N} m_i} \tag{6.13}$$

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:

$$\mathrm{sh}_{\mathrm{max}} = \max_z\left|\frac{\partial v_x(z)}{\partial z}\right| = A\frac{\rho}{\eta}\frac{l_z}{2\pi} \tag{6.14}$$

For a simulation with: $\eta = 10^{-3}$ [kg m^{-1} s^{-1}], $\rho = 10^3$ [kg m^{-3}] and $l_z = 2\pi$ [nm], $\mathrm{sh}_{\mathrm{max}} = 1$ [ps nm^{-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 picoseconds. In this case A should be smaller than 0.1 [nm ps^{-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 picoseconds 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[27], which can be written in terms of the shear rate as:

$$T_s = \frac{\eta\,\tau}{2\rho\,C_v}\mathrm{sh}_{\mathrm{max}}^2 \tag{6.15}$$

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: $T_s = 25$ [K ps^{-2}] $\mathrm{sh}_{\mathrm{max}}^2$. When we want the shear rate to be smaller than $1/10$ [ps^{-1}], T_s is smaller than 0.25 [K], which is negligible.

Note that the system has to build up the velocity profile when starting from an equilibrium state. This build-up time is of the order of the correlation time of the liquid.

Two quantities are written to the energy file, along with their averages and fluctuations: V and $1/\eta$ as obtained from (6.12).

6.7 Tabulated interaction functions

6.7.1 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:

$$V_s(x) = A_0 + A_1\,\epsilon + A_2\,\epsilon^2 + A_3\,\epsilon^3 \tag{6.16}$$

where the table spacing h and fraction ϵ are given by:

$$h = x_{i+1} - x_i \tag{6.17}$$

$$\epsilon = (x - x_i)/h \tag{6.18}$$

so that $0 \leq \epsilon < 1$. From this we can calculate the derivative in order to determine the forces:

$$-V_s'(x) = -\frac{\mathrm{d}V_s(x)}{\mathrm{d}\epsilon}\frac{\mathrm{d}\epsilon}{\mathrm{d}x} = -(A_1 + 2A_2\,\epsilon + 3A_3\,\epsilon^2)/h \tag{6.19}$$

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:

$$
\begin{aligned}
|V_s - V|_{max} &= V'''' \frac{h^4}{384} + O(h^5) && (6.20)\\
|V_s' - V'|_{max} &= V'''' \frac{h^3}{72\sqrt{3}} + O(h^4) && (6.21)\\
|V_s'' - V''|_{max} &= V'''' \frac{h^2}{12} + O(h^3) && (6.22)
\end{aligned}
$$

V and V' are continuous, while V" is the first discontinuous derivative. The number of points per nanometer is 500 and 2000 for single- and double-precision versions of GROMACS, respectively. This means that the errors in the potential and force will usually be smaller than the single 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 ($\boldsymbol{r}_{ij}$) 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 (ϵ = sr$_{ij} - n_0$) and ϵ^2
4. Do the interpolation to calculate the potential V and the the scalar force f
5. Calculate the vector force $\boldsymbol{F}$ by multiplying f with $\boldsymbol{r}_{ij}$

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.

6.7.2 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

$$
V(r_{ij}) = \frac{q_i q_j}{4\pi\epsilon_0} f(r_{ij}) + C_6\, g(r_{ij}) + C_{12}\, h(r_{ij}) \qquad (6.23)
$$

with f,g,h 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` file:

```
rlist = 1.0
coulombtype = User
rcoulomb = 1.0
```

```
vdwtype = User
rvdw = 1.0
```

the MD program 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 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 table extension can be changed in the `.mdp` 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 single 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, 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 option `energygrp_table` (see sec. 7.3). 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.

6.8 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 is 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[110], GAMESS-UK[111], Gaussian[112] and CPMD[113]).

GROMACS interactions between the two subsystems are either handled as described by Field *et al.*[114] or within the ONIOM approach by Morokuma and coworkers[115, 116].

6.8.1 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:

$$H^{QM/MM} = H_e^{QM} - \sum_i^n \sum_J^M \frac{e^2 Q_J}{4\pi\epsilon_0 r_{iJ}} + \sum_A^N \sum_J^M \frac{e^2 Z_A Q_J}{e\pi\epsilon_0 R_{AJ}}, \tag{6.24}$$

   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 us 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):

$$E_{tot} = E_I^{QM} + E_{I+II}^{MM} - E_I^{MM}, \tag{6.25}$$

   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 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.

6.8.2 Usage

To make use of the QM/MM functionality in GROMACS, one needs to:

1. introduce link atoms at the QM/MM boundary, if needed;
2. specify which atoms are to be treated at a QM level;
3. specify the QM level, basis set, type of QM/MM interface and so on.

Adding link atoms

At the bond that connects the QM and MM subsystems a link atoms is introduced. In GROMACS the link atom has special atomtype, called LA. This atomtype is treated as a hydrogen atom in the QM calculation, and as a dummy atom in the force field calculation. The link atoms, if any, are part of the system, but have no interaction with any other atom, except that the QM force working on it is distributed over the two atoms of the bond. In the topology the link atom (LA), therefore, is defined as a virtual site atom:

```
[ virtual_sites2 ]
LA QMatom MMatom 1 0.65
```

See the dummy atoms section for more details on how dummies are treated. The link atom is replaced at every step of the simulation.

In addition, the bond itself is replaced by a constraint:

```
[ constraints ]
QMatom MMatom 2 0.153
```

Note that, because in our system the QM/MM bond is a carbon-carbon bond (0.153 nm), we use a constraint length of 0.153 nm, and dummy position of 0.65. The latter is the ratio between the ideal C-H bond length and the ideal C-C bond length. With this ratio, the link atom is always 0.1 nm away from the QMatom, consistent with the carbon-hydrogen bond length. If the QM and MM subsystems are connected by a different kind of bond, a different constraint and a different dummy position, appropriate for that bond type, are required.

Specifying the QM atoms

Atoms that should be treated at a QM level of theory, including the link atoms, are added to the index file. In addition, the chemical bonds between the atoms in the QM region are to be defined as connect bonds (bond type 5)in the topology file:

```
[ bonds ]
QMatom1 QMatom2 5
QMatom2 QMatom3 5
```

Specifying the QM/MM simulation parameters

In the mdp file, the following parameters control a QM/MM simulation.

`QMMM = no`

If this is set to `yes`, a QM/MM simulation is requested. Several groups of atoms can be described at different QM levels separately. These are specified in the QMMM-grps field separated by spaces. The level of *ab initio* theory at which the groups are described is specified by `QMmethod` and `QMbasis` Fields. Describing the groups at different levels of theory is only possible with the ONIOM QM/MM scheme, specified by `QMMMscheme`.

`QMMM-grps =`

groups to be described at the QM level

`QMMMscheme = normal`

Options are `normal` and `ONIOM`. This selects the QM/MM interface. `normal` implies that the QM subsystem is electronically embedded in the MM subsystem. There can only be one `QMMM-grps` that is modeled at the `QMmethod` and `QMbasis` level of *ab initio* theory. The rest of the system is described at the MM level. The QM and MM subsystems interact as follows: MM point charges are included in the QM one-electron Hamiltonian and all Lennard-Jones interactions are described at the MM level. If `ONIOM` is selected, the interaction between the subsystem is described using the ONIOM method by Morokuma and co-workers. There can be more than one QMMM-grps each modeled at a different level of QM theory (QMmethod and QMbasis).

`QMmethod =`

Method used to compute the energy and gradients on the QM atoms. Available methods are AM1, PM3, RHF, UHF, DFT, B3LYP, MP2, CASSCF, MMVB and CPMD. For CASSCF, the number of electrons and orbitals included in the active space is specified by `CASelectrons` and `CASorbitals`. For CPMD, the plane-wave cut-off is specified by the `planewavecutoff` keyword.

`QMbasis =`

Gaussian basis set used to expand the electronic wave-function. Only Gaussian basis sets are currently available, i.e. STO-3G, 3-21G, 3-21G*, 3-21+G*, 6-21G, 6-31G, 6-31G*, 6-31+G*, and 6-311G. For CPMD, which uses plane wave expansion rather than atom-centered basis functions, the `planewavecutoff` keyword controls the plane wave expansion.

`QMcharge =`

The total charge in *e* of the `QMMM-grps`. In case there are more than one `QMMM-grps`, the total charge of each ONIOM layer needs to be specified separately.

`QMmult =`

The multiplicity of the `QMMM-grps`. In case there are more than one `QMMM-grps`, the multiplicity of each ONIOM layer needs to be specified separately.

`CASorbitals =`

The number of orbitals to be included in the active space when doing a CASSCF computation.

`CASelectrons =`

The number of electrons to be included in the active space when doing a CASSCF computation.

`SH = no`

If this is set to yes, a QM/MM MD simulation on the excited state-potential energy surface and enforce a diabatic hop to the ground-state when the system hits the conical intersection hyperline in the course the simulation. This option only works in combination with the CASSCF method.

6.8.3 Output

The energies and gradients computed in the QM calculation are added to those computed by GROMACS. In the `.edr` file there is a section for the total QM energy.

6.8.4 Future developments

Several features are currently under development that increase the accuracy of the QM/MM interface. One useful feature is the use of delocalized MM charges in the QM computations. The most important benefit of using such smeared-out charges is that the Coulombic potential has a finite value at inter atomic distances. In the point charge representation, the partially charged MM atoms close to the QM region, tend to ’over-polarize’ the QM system, which leads to artifacts in the calculation.

What is needed as well is a transition state optimizer.

6.9 GROMACS on GPUs

This is an experimental release of GROMACS for accelerated Molecular Dynamics simulations on GPU accelerators. Support is provided by the OpenMM library. This release is targeted at developers and advanced users and care should be taken before production use. The following should be noted before using the GPU accelerated program:

- The current release runs only on modern NVIDIA GPU hardware with CUDA support. Make sure that the necessary CUDA drivers and libraries for your operating system are already installed.
- Multiple GPUs are not supported.
- Only a fairly small subset of the GROMACS features and options are supported on the GPUs. See below for a detailed list.
- Consumer level GPU cards are known to often have problems with faulty memory. It is recommended that a full memory check of the cards is done at least once (for example, using the `memtest=full` option). A partial memory check (for example, `memtest=15`) before and after the simulation run would help spot problems resulting from overheating of the graphics card.
- The maximum size of the simulated systems depends on the available GPU memory, for example, a GTX280 with 1GB memory has been tested with systems of up to about 100,000 atoms.

- In order to take a full advantage of the GPU platform features, many algorithms have been implemented in a very different way than they are on the CPUs. Therefore numerical correspondence between some properties of the system's state should not be expected. Moreover, the values will likely vary when simulations are done on different GPU hardware. However, sufficiently long trajectories should produce comparable statistical averages.

- Frequent retrieval of system state information such as trajectory coordinates and energies can greatly influence the performance of the program due to slow CPU<–>GPU memory transfer speed.

- MD algorithms are complex, and although the GROMACS code is highly tuned for them, they often do not translate very well onto the streaming architectures. Realistic expectations about the achievable speed-up from tests with GTX280: for small protein systems in implicit solvent using all-vs-all kernels the acceleration can be as high as 20 times, but in most other setups involving cutoffs and PME the acceleration is usually only about 5 times relative to a 3GHz CPU.

6.9.1 Supported features

- **Integrators:** `md/md-vv/md-vv-avek`, `sd/sd1` and `bd`.
  OpenMM implements only the velocity-Verlet algorithm for MD simulations. Option `md` is accepted but keep in mind that the actual algorithm is not leap-frog. Thus all three options `md`, `md-vv` and `md-vv-avek` are equivalent. Similarly, options `sd` and `sd1` are also equivalent.

- **Long-range interactions:** `Reaction-Field`, `Ewald`, `PME`.
  `No-cutoff`, i.e. `rcoulomb=0` and `rvdw=0`, is also supported.
  For Ewald summation only 3D geometry is supported, and dipole correction is not.

- **Temperature control:** Supported only with the `sd/sd1`, `bd`, `md/md-vv/md-vv-avek` integrators.
  OpenMM implements only the Andersen thermostat. All values for `tcoupl` are thus accepted and equivalent to `andersen`. Multiple temperature coupling groups are not supported, only `tc-grps=System` will work. Remember that for heterogeneous systems such as membrane proteins, coupling of the whole system will likely lead to different temperatures in the different phases - hot solvent and cold solute.

- **Force fields:** Supported FF are Amber, CHARMM, OPLSAA. GROMOS is not supported.

- **Implicit solvent:** Supported only with `reaction-field` electrostatics. The only supported algorithm for GB is OBC, and the default GROMACS values for the scale factors are hard coded in OpenMM, i.e. obc_alpha=1, obc_beta=0.8 and obc_gamma=4.85.

- **Constraints:** Constraints in OpenMM are done by a combination of SHAKE, SETTLE and CCMA. Accuracy is based on the SHAKE tolerance as set by the `shake_tol` option.

- **Periodic Boundary Conditions:** Only `pbc=xyz` and `pbc=no` in rectangular cells(boxes) are supported.

- **Pressure control:** OpenMM implements the Monte Carlo barostat. All values for `pcoupl` are thus accepted.
- **Simulated annealing:** Not supported.
- **Pulling:** Not supported.
- **Restraints:** Distance, orientation, angle and dihedral restraints are not supported in the current implementation.
- **Free energy calculations:** Not supported in the current implementation.
- **Walls:** Not supported.
- **Non-equilibrium MD:** Option `acc_grps` is not supported.
- **Electric Fields:** Not supported.
- **QMMM:** Not supported.

6.9.2 Installing and running GROMACS-GPU

GROMACS-GPU can be installed either from the officially distributed binary or source packages. We provide pre-compiled binaries built for and tested on the most common Linux, Windows, and Mac OS operating systems (for details see the GROMACS-GPU download page). Using the binary distribution is highly recommended and it should work in most of the cases. Below we summarize how to get the GPU accelerated mdrun-gpu work.

Prerequisites

The current GROMACS-GPU release uses OpenMM acceleration, the necessary libraries and plug-ins are included in the binary packages.

Both the OpenMM library and GROMACS-GPU require version 3.1 of the CUDA libraries and compatible NVIDIA driver (i.e. version ¿ 256).

Last but not least, to run GPU accelerated simulations, a CUDA-enabled graphics card is necessary. Molecular dynamics algorithms are very demanding and unlike in other application areas, only high-end graphics cards are capable of providing performance comparable to or higher then modern CPUs. For this reason, mdrun-gpu is compatible with only a subset of CUDA-enabled GPUs (for detailed list see section 6.9.3) and by default it does not run if it detects non-compatible hardware.

For details about compatibility of NVIDIA drivers with the CUDA library and devices consult the NVIDIA developer page.

Summary of prerequisites:

- OpenMM;
- NVIDIA CUDA libraries;

- NVIDIA driver;
- NVIDIA CUDA-enabled GPU.

Installing

1. Download and unpack the binary package for the respective OS and architecture. Copy the content of the package to your normal GROMACS installation directory (or to a custom location).

   Note that the distributed GROMACS-GPU packages do not contain the entire set of tools and utilities included in a full GROMACS installation. Therefore, it is recommended to have a $\geq$v4.5 standard GROMACS installation along the GPU accelerated one.
2. Add the `openmm/lib` directory to your library path, e.g. in bash:
   `export LD_LIBRARARY_PATH=path_to_gromacs/openmm/lib:$LD_LIBRARY_PATH`.
   If there are other OpenMM versions installed, make sure that the supplied libraries have preference when running mdrun-gpu. Also, make sure that the CUDA libraries installed match the version of CUDA that was used for compilation of GROMACS-GPU.
3. Set the `OPENMM_PLUGIN_DIR` environment variable to contain the path to the `openmm/lib/plugins` directory, e.g. in bash:
   `export OPENMM_PLUGIN_DIR=path_to_gromacs/openmm/lib/plugins`.
4. At this point, running the command `path_to_gromacs/bin/mdrun-gpu -h` should display the standard mdrun help which means that the binary runs and all the necessary libraries are accessible.

Compiling mdrun-gpu

The GPU accelerated mdrun can be compiled on Linux, Mac OS and Windows operating systems, both for 32 and 64 bit. Besides the prerequisites discussed above, in order to compile mdrun-gpu the following additional software is required:

- Cmake version $\leq$2.6.4
- CUDA-compatible compiler:
  - MSVC 8 or 9 on Windows
  - gcc 4.4 on Linux and Mac OS
- OpenMM-2.0 header files

Note, that the current GROMACS-GPU release is compatible with OpenMM version 2.0. While future versions might be compatible, using the officially supported and tested OpenMM version is strongly encouraged. OpenMM binaries as well as source code can be obtained from the project's homepage.

Also note that it is essential that the same version of CUDA is used to compile both mdrun-gpu and the OpenMM libraries.

To compile mdrun-gpu change to the top level directory of the source tree and execute the following commands:

- `export OPENMM_ROOT_DIR=path_to_custom_openmm_installpath`
- `cmake -DGMX_OPENMM=ON [-DCMAKE_INSTALL_PREFIX=desired_install_path]`
- `make mdrun`
- `make install-mdrun`

Testing and troubleshooting

GROMACS-GPU specific mdrun features

Besides the usual command line options, mdrun-gpu also supports a set of "device options", that are meant to give control over acceleration related functionalities. These options can be used in the following form:
`mdrun-gpu -device "ACCELERATION:[DEV_OPTION=VALUE,]... [OPTION]..".`

The option-list prefix `ACCELERATION` specifies which acceleration library should be used. At the moment, the only supported value is `OpenMM`. This is followed by the list of comma-separated `DEV_OPTION=VALUE` option-value pairs which define parameters for the selected acceleration platform. The entire device option string is case insensitive.

Below we summarize the available options (of the OpenMM acceleration library) and their possible values.

Platform Selects the GPGPU platform to be used, currently the only supported value is CUDA (in future OpenCL support will be added).

DeviceID The numeric identifier of the CUDA device on which the simulation will be carried out. The default value is 0, i.e. the first device.

Memtest GPUs, especially consumer-level devices, are prone to memory errors. There might be various reasons for ”soft errors“ to happen including (factory) overclocking, overheating, faulty hardware etc, but the result is always the same: unreliable, possibly incorrect results. Therefore, gromacs-gpu has a built-in mechanism for testing the GPU memory in order to catch the obviously faulty hardware. A set of tests are performed before and after each simulation and if errors are detected, the execution is aborted.

Accepted values for this option are any integer $\leq$15 with an optional "s" prefix representing the approximate amount of time in seconds that should be spent on testing; the default value is `memtest=15s`. It is possible to completely turn off memory testing by setting `memtest=off`, however this is not advisable.

Force-device Option that enables running mdrun-gpu devices that are not supported but CUDA-capable. Using this option might results in very low performance or even crashes and therefor it is not encouraged.

Note, that both the option names and the values are case-insensitive.

6.9.3 Hardware and software compatibility list

Compatible OpenMM versions:

- v2.0

Compatible NVIDIA CUDA versions (also see OpenMM version compatibility above):

- v3.1

Compatible hardware (for details consult the NVIDIA CUDA GPUs list):

- G92/G94:
  - GeForce 9800 GX2/GTX/GTX+/GT
  - GeForce 9800M GT
  - GeForce GTS 150, 250
  - GeForce GTX 280M, 285M
  - Quadro FX 4700
  - Quadro Plex 2100 D4
- GT200:
  - GeForce GTX 260, 270, 280, 285, 295
  - Tesla C1060, S1070, M1060
  - Quadro FX 4800, 5800
  - Quadro CX
  - Quadro Plex 2200 D2, 2200 S4
- GF100 (Fermi)
  - GeForce GTX 460, 465, 470, 480
  - Tesla C2050, C2070, S2050, S2070

Chapter 7

Run parameters and Programs

7.1 On-line and HTML manuals

All the information in this chapter can also be found in HTML format in your GROMACS data directory. The path depends on where your files are installed, but the default location is

`/usr/local/gromacs/share/html/online.html`

Or, if you installed from Linux packages it can be found as

`/usr/local/share/gromacs/html/online.html`

You can also use the online from our web site,

www.gromacs.org/documentation/reference_3.0/online.html

In addition, we install standard UNIX manuals 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`, and you should be able to type e.g. `man grompp`.

The program manual pages can also be found in Appendix D in this manual.

7.2 File types

Table 7.1 lists the file types used by GROMACS along with a short description, and you can find a more detail description for each file in your HTML reference, or in our online version.

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.

7.3 Run Parameters

7.3.1 General

Default values are given in parentheses. 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 is available. This should be appropriate to start a normal simulation. Edit it to suit your specific needs and desires.

Default Name	Ext.	Type	Default Option	Description
atomtp	.atp	Asc		Atomtype file used by pdb2gmx
eiwit	.brk	Asc	-f	Brookhaven data bank file
state	.cpt	xdr		Checkpoint file
nnnice	.dat	Asc		Generic data file
user	.dlg	Asc		Dialog Box data for ngmx
sam	.edi	Asc		ED sampling input
sam	.edo	Asc		ED sampling output
ener	.edr			Generic energy: `edr ene`
ener	.edr	xdr		Energy file in portable xdr format
ener	.ene	Bin		Energy file
eiwit	.ent	Asc	-f	Entry in the protein date bank
plot	.eps	Asc		Encapsulated PostScript (tm) file
conf	.esp	Asc	-c	Coordinate file in ESPResSo format
gtraj	.g87	Asc		Gromos-87 ASCII trajectory format
conf	.g96	Asc	-c	Coordinate file in Gromos-96 format
conf	.gro	Asc	-c	Coordinate file in Gromos-87 format
conf	.gro		-c	Structure: `gro g96 pdb esp tpr tpb tpa`
out	.gro		-o	Structure: `gro g96 pdb esp`
polar	.hdb	Asc		Hydrogen data base
topinc	.itp	Asc		Include file for topology
run	.log	Asc	-l	Log file
ps	.m2p	Asc		Input file for mat2ps
ss	.map	Asc		File that maps matrix data to colors
ss	.mat	Asc		Matrix Data file
grompp	.mdp	Asc	-f	grompp input file with MD parameters
hessian	.mtx	Bin	-m	Hessian matrix
index	.ndx	Asc	-n	Index file
hello	.out	Asc	-o	Generic output file
eiwit	.pdb	Asc	-f	Protein data bank file
residue	.rtp	Asc		Residue Type file used by pdb2gmx
doc	.tex	Asc	-o	LaTeX file
topol	.top	Asc	-p	Topology file
topol	.tpb	Bin	-s	Binary run input file
topol	.tpr		-s	Generic run input: `tpr tpb tpa`
topol	.tpr		-s	Structure+mass(db): `tpr tpb tpa gro g96 pdb`
topol	.tpr	xdr	-s	Portable xdr run input file
traj	.trj	Bin		Trajectory file (architecture specific)
traj	.trr			Full precision trajectory: `trr trj cpt`
traj	.trr	xdr		Trajectory in portable xdr format
root	.xpm	Asc		X PixMap compatible matrix file
traj	.xtc		-f	Trajec., input: `xtc trr trj cpt gro g96 pdb`
traj	.xtc		-f	Trajectory, output: `xtc trr trj gro g96 pdb`
traj	.xtc	xdr		Compressed trajectory (portable xdr format)
graph	.xvg	Asc	-o	xvgr/xmgr file

Table 7.1: The GROMACS file types.

7.3.2 Preprocessing

include:

directories to include in your topology. Format:
`-I/home/john/my_lib -I../more_lib`

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 are already available by default are:

-DFLEXIBLE

Will tell grompp to include flexible water in stead of rigid water into your topology, this can be useful for normal mode analysis.

-DPOSRES

Will tell grompp to include posre.itp into your topology, used for position restraints.

7.3.3 Run control

integrator:

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 **md** leap-frog integrator. The 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 **md** integrator is accurate enough.

md-vv-avek

A velocity Verlet algorithm identical to **md-vv**, except that the kinetic energy is determined as the average of the two half step kinetic energies as in the **md** 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 leap-frog stochastic dynamics integrator. Four Gaussian random number are required per integration step per degree of freedom. 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**) is set with **ref_t** [K], the inverse friction constant for each group is set with **tau_t** [ps]. The parameter **tcoupl** is ignored. The random generator is initialized with **ld_seed**. When used as a thermostat, an appropriate value for **tau_t** 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 (unless **cut-off** or **reaction-field** electrostatics is used). **NOTE:** temperature deviations decay twice as fast as with a Berendsen thermostat with the same **tau_t**.

sd1

An efficient leap-frog stochastic dynamics integrator. This integrator is equivalent to **sd**, except that it requires only one Gaussian random number and one constraint step. This integrator is less accurate. For water the relative error in the temperature with this integrator is 0.5 **delta_t/tau_t**, but for other systems it can be higher. Use with care and check the temperature.

bd

An Euler integrator for Brownian or position Langevin dynamics, the velocity is the force divided by a friction coefficient (**bd_fric** [amu ps^{-1}]) plus random thermal noise (**ref_t**). When **bd_fric**=0, the friction coefficient for each particle is calculated as mass/**tau_t**, as for the integrator `sd`. The random generator is initialized with **ld_seed**.

The following algorithms are not integrators, but selected using
the integrator tag anyway

steep
A steepest descent algorithm for energy minimization. The maximum step size is **emstep** [nm], the tolerance is **emtol** [kJ mol^{-1} nm^{-1}].

cg
A conjugate gradient algorithm for energy minimization, the tolerance is **emtol** [kJ mol^{-1} nm^{-1}]. CG is more efficient when a steepest descent step is done every once in a while, this is determined by **nstcgsteep**. 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` file. GROMACS should be compiled in double precision.

tpi
Test particle insertion. The last molecule in the topology is the test particle. A trajectory should be provided with the `-rerun` option of `mdrun`. This trajectory should not contain the molecule to be inserted. Insertions are performed **nsteps** times in each frame at random locations and with random orientiations of the molecule. When **nstlist** is larger than one, **nstlist** insertions are performed in a sphere with radius **rtpi** around a the same random location using the same neighborlist (and the same long-range energy when **rvdw** or **rcoulomb** $>$ **rlist**, which is only allowed for single-atom molecules). Since neighborlist construction is expensive, one can perform several extra insertions with the same list almost for free. The random seed is set with **ld_seed**. The temperature for the Boltzmann weighting is set with **ref_t**, 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 the `-tpi` option of `mdrun`. The distribution of insertion energies is written to the file specified with the `-tpid` option of `mdrun`. 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 **tpi**, 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** sets the radius for the sphere around this location. Neighbor searching is done only once per frame, **nstlist** is not used. Parallel tpic gives identical results to single node tpic.

tinit: (0) [ps]
starting time for your run (only makes sense for integrators `md`, `sd` and `bd`)

dt: (0.001) [ps]
time step for integration (only makes sense for integrators `md`, `sd` and `bd`)

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

init_step: (0)
The starting step. The time at an step i in a run is calculated as: t = `tinit` + `dt`*(`init_step` + i). The free-energy lambda is calculated as: lambda = `init_lambda` + `delta_lambda`*(`init_step` + 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` to the step number of the restart frame. `tpbconv` does this automatically.

nstcalcenergy: (-1)
The frequency for calculating the energies, 0 is never. This option is only relevant with dynamics. With a twin-range cut-off setup **nstcalcenergy** should be equal to or a multiple of **nstlist**. 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. With global temperature and/or pressure coupling the time step for the coupling algorithm is **nstcalcenergy*dt**. Take this into account when setting **tau_t** and/or **tau_p**. The default value of -1 sets **nstcalcenergy** equal to **nstlist**, unless **nstlist** &le 0, then a value of 10 is used.

comm_mode:

Linear
Remove center of mass translation

Angular
Remove center of mass translation and rotation around the center of mass

No
No restriction on the center of mass motion

nstcomm: (10) [steps]
frequency for center of mass motion removal

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

7.3.4 Langevin dynamics

bd_fric: (0) [amu ps^{-1}]
Brownian dynamics friction coefficient. When **bd_fric**=0, the friction coefficient for each particle is calculated as mass/**tau_t**.

ld_seed: (1993) [integer]
used to initialize random generator for thermal noise for stochastic and Brownian dynamics. When **ld_seed** is set to -1, the seed is calculated as `(time() + getpid()) % 1000000`. When running BD or SD on multiple processors, each processor uses a seed equal to **ld_seed** plus the processor number.

7.3.5 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.

7.3.6 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, or a maximum number of iterations (niter) has been reached

emtol: (10.0) [kJ mol^{-1} nm^{-1}]
the minimization is converged when the maximum force is smaller than this value. For shell MD this value should be 1.0 at most, but since the variable is used for energy minimization as well the default is 10.0.

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/(d^{2}V/d q^2) where mu is the reduced mass of two particles in a flexible constraint and d^{2}V/d q^2 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!

7.3.7 Test particle insertion

rtpi: (0.05) [nm]

the test particle insertion radius see integrators **tpi** and **tpic**

7.3.8 Output control

nstxout: (100) [steps]

frequency to write coordinates to output trajectory file, the last coordinates are always written

nstvout: (100) [steps]

frequency to write velocities to output trajectory, the last velocities are always written

nstfout: (0) [steps]

frequency to write forces to output trajectory.

nstlog: (100) [steps]

frequency to write energies to log file, the last energies are always written

nstenergy: (100) [steps]

frequency to write energies to energy file, the last energies are always written, should be a multiple of **nstcalcenergy**, note that the exact sums and fluctuations over all MD steps modulo **nstcalcenergy** are stored in the energy file, so `g_energy` can report exact energy averages and fluctuations also when **nstenergy** > 1

nstxtcout: (0) [steps]

frequency to write coordinates to xtc trajectory

xtc_precision: (1000) [real]

precision to write to xtc trajectory

xtc_grps:

group(s) to write to xtc trajectory, default the whole system is written (if **nstxtcout** is larger than zero)

energygrps:

group(s) to write to energy file

7.3.9 Neighbor searching

nstlist: (10) [steps]

$>$ 0

Frequency to update the neighbor list (and the long-range forces, when using twin-range cut-off's). When this is 0, the neighbor list is made only once. With energy minimization the neighborlist will be updated for every energy evaluation when **nstlist** > 0.

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.

-1

Automated update frequency. This can only be used with switched, shifted or user potentials where the cut-off can be smaller than **rlist**. One then has a buffer of size **rlist** minus the longest cut-off. The neighbor list is only updated when one or more particles have moved further than half the buffer size from the center of geometry of their charge group as determined at the previous neighbor search. Coordinate scaling due to pressure coupling or the **deform** option is taken into account. This option guarantees that their are no cut-off artifacts. But for larger systems this can come at a high computational cost, since the neighbor list update frequency will be determined by just one or two particles moving slightly beyond the half buffer length (which not even necessarily implies that the neighbor list is invalid), while 99.99% of the particles are fine.

ns_type:

grid
Make a grid in the box and only check atoms in neighboring grid cells when constructing a new neighbor list every **nstlist** steps. In large systems grid search is much faster than simple search.

simple
Check every atom in the box when constructing a new neighbor list every **nstlist** steps.

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 to 0 and **nstlist**=0. For best performance without cut-offs, use **nstlist**=0, **ns_type**=simple and particle decomposition instead of domain decomposition.

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**. 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

rlist: (1) [nm]
cut-off distance for the short-range neighbor list

rlistlong: (-1) [nm]
Cut-off distance for the long-range neighbor list. This parameter is only relevant for a twin-range cut-off setup with switched potentials. In that case a buffer region is required to account for the size of charge groups. In all other cases this parameter is automatically set to the longest cut-off distance.

7.3.10 Electrostatics

coulombtype:

Cut-off
Twin range cut-off's with neighborlist cut-off **rlist** and Coulomb cut-off **rcoulomb**, where **rcoulomb** $\geq$ **rlist**.

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

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 Particle-Mesh Ewald electrostatics. Direct space is similar to the Ewald sum, while the reciprocal part is performed with FFTs. Grid dimensions are controlled with **fourierspacing** and the interpolation order with **pme_order**. With a grid spacing of 0.1 nm and cubic interpolation the electrostatic forces have an accuracy of 2-3e-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.

PPPM

Particle-Particle Particle-Mesh algorithm for long range electrostatic interactions. Use for example **rlist**=`0.9`, **rcoulomb**=`0.9`. The grid dimensions are controlled by **fourierspacing**. Reasonable grid spacing for PPPM is 0.05-0.1 nm. See `Shift` for the details of the particle-particle potential.

NOTE: PPPM is not functional in the current version, we plan to implement PPPM through a small modification of the PME code.

Reaction-Field

Reaction field with Coulomb cut-off **rcoulomb**, where **rcoulomb** $\geq$ **rlist**. The dielectric constant beyond the cut-off is **epsilon_rf**. The dielectric constant can be set to infinity by setting **epsilon_rf**=0.

Generalized-Reaction-Field

Generalized reaction field with Coulomb cut-off **rcoulomb**, where **rcoulomb** $\geq$ **rlist**. The dielectric constant beyond the cut-off is **epsilon_rf**. The ionic strength is computed from the number of charged (*i.e.* with non zero charge) charge groups. The temperature for the GRF potential is set with **ref_t** [K].

Reaction-Field-zero

In GROMACS normal reaction-field electrostatics leads to bad energy conservation. **Reaction-Field-zero** solves this by making the potential zero beyond the cut-off. It can only be used with an infinite dielectric constant (**epsilon_rf=0**), because only for that value the force vanishes at the cut-off. **rlist** should be 0.1 to 0.3 nm larger than **rcoulomb** to accommodate for the size of charge groups and diffusion between neighbor list updates. This, and the fact that table lookups are used instead of analytical functions make **Reaction-Field-zero** computationally more expensive than normal reaction-field.

Reaction-Field-nec

The same as **Reaction-Field**, but implemented as in GROMACS versions before 3.3. No reaction-field correction is applied to excluded atom pairs and self pairs. The 1-4 interactions are calculated using a reaction-field. The missing correction due to the excluded pairs that do not have a 1-4 interaction is up to a few percent of the total electrostatic energy and causes a minor difference in the forces and the pressure.

Shift

Analogous to **Shift** for **vdwtype**. You might want to use **Reaction-Field-zero** instead, which has a similar potential shape, but has a physical interpretation and has better energies due to the exclusion correction terms.

Encad-Shift

The Coulomb potential is decreased over the whole range, using the definition from the Encad simulation package.

Switch

Analogous to **Switch** for **vdwtype**. Switching the Coulomb potential can lead to serious artifacts, advice: use **Reaction-Field-zero** instead.

User

`mdrun` will now expect to find a file `table.xvg` with user-defined potential functions for repulsion, dispersion and Coulomb. When pair interactions are present, `mdrun` 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** 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** 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 single 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

A combination of PME and a switch function for the direct-space part (see above). **rcoulomb** is allowed to be smaller than **rlist**. This is mainly useful constant energy simulations. For constant temperature simulations the advantage of improved energy conservation is usually outweighed by the small loss in accuracy of the electrostatics.

PME-User

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

PME-User-Switch

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.

rcoulomb_switch: (0) [nm]
where to start switching the Coulomb potential

rcoulomb: (1) [nm]
distance for the Coulomb cut-off

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

epsilon_rf: (1)
The relative dielectric constant of the reaction field. This is only used with reaction-field electrostatics. A value of 0 means infinity.

7.3.11 VdW

vdwtype:

Cut-off
Twin range cut-off's with neighbor list cut-off **rlist** and VdW cut-off **rvdw**, where **rvdw** $\geq$ **rlist**.

Shift
The LJ (not Buckingham) potential is decreased over the whole range and the forces decay smoothly to zero between **rvdw_switch** and **rvdw**. The neighbor search cut-off **rlist** should be 0.1 to 0.3 nm larger than **rvdw** to accommodate for the size of charge groups and diffusion between neighbor list updates.

Switch
The LJ (not Buckingham) potential is normal out to **rvdw_switch**, after which it is switched off to reach zero at **rvdw**. 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). The neighbor search cut-off **rlist** should be 0.1 to 0.3 nm larger than **rvdw** to accommodate for the size of charge groups and diffusion between neighbor list updates.

Encad-Shift
The LJ (not Buckingham) potential is decreased over the whole range, using the definition from the Encad simulation package.

User
See **user** for **coulombtype**. The function value at `x=0` is not important. When you want to use LJ correction, make sure that **rvdw** corresponds to the cut-off in the user-defined function. When **coulombtype** is not set to **User** the values for f and -f' are ignored.

rvdw_switch: (0) [nm]
where to start switching the LJ potential

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

7.3.12 Tables

table-extension: (1) [nm]
Extension of the non-bonded potential lookup tables beyond the largest cut-off distance. The value should be large enough to account for charge group sizes and the diffusion between neighbor-list updates. Without user defined potential the same table length is used for the lookup tables for the 1-4 interactions, which are always tabulated irrespective of the use of tables for the non-bonded interactions.

energygrp_table:
When user tables are used for electrostatics and/or VdW, here one can give pairs of energy groups for which seperate user tables should be used. The two energy groups will be appended to the table file name, in order of their definition in **energygrps**, seperated by underscores. For example, if `energygrps = Na Cl Sol` and `energygrp_table = Na Na Na Cl`, `mdrun` 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.

7.3.13 Ewald

fourierspacing: (0.12) [nm]
The maximum grid spacing for the FFT grid when using PPPM or PME. For ordinary Ewald the spacing times the box dimensions determines the highest magnitude to use in each direction. In all cases each direction can be overridden by entering a non-zero value for **fourier_n***. 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.

fourier_nx (0) ; fourier_ny (0) ; fourier_nz: (0)
Highest magnitude of wave vectors in reciprocal space when using Ewald. Grid size when using PPPM or PME. These values override **fourierspacing** per direction. The best choice is powers of 2, 3, 5 and 7. Avoid large primes.

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 (1e-5)
The relative strength of the Ewald-shifted direct potential at **rcoulomb** is given by **ewald_rtol**. Decreasing this will give a more accurate direct sum, but then you need more wave vectors for the reciprocal sum.

ewald_geometry: (3d)

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.

optimize_fft:

no
Don't calculate the optimal FFT plan for the grid at startup.

yes
Calculate the optimal FFT plan for the grid at startup. This saves a few percent for long simulations, but takes a couple of minutes at start.

7.3.14 Temperature coupling

tcoupl:

no
No temperature coupling.

berendsen
Temperature coupling with a Berendsen-thermostat to a bath with temperature **ref_t** [K], with time constant **tau_t** [ps]. Several groups can be coupled separately, these are specified in the **tc_grps** 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** [ps] 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 energy and log file.

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**, but the stochastic term ensures that a proper canonical ensemble is generated. The random seed is set with **ld_seed**. This thermostat works correctly even for **tau_t**=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** equal to **nstlist**, unless **nstlist** &le 0, then a value of 10 is used. For velocity Verlet integrators **nsttcouple** is set to 1.

nh-chain-length (10)
the number of chained Nose-Hoover thermostats for velocity Verlet integrators, the leap-frog **md** integrator only supports 1. Data for the NH chain variables is not printed to the .edr, but can be using the GMX_NOSEHOOVER_CHAINS environment variable

tc_grps:
groups to couple separately to temperature bath

tau_t: [ps]
time constant for coupling (one for each group in tc_grps), -1 means no temperature coupling

ref_t: [K]
reference temperature for coupling (one for each group in tc_grps)

7.3.15 Pressure coupling

pcoupl:

no

No pressure coupling. This means a fixed box size.

berendsen

Exponential relaxation pressure coupling with time constant **tau_p** [ps]. The box is scaled every timestep. 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.

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** [ps] 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 fluctation 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 **md-vv** or **md-vv-avek**, very similar to Parinello-Raphman. As for Nose-Hoover temperature coupling the time constant **tau_p** [ps] 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 only supports isotropic scaling.

pcoupltype:

isotropic

Isotropic pressure coupling with time constant **tau_p** [ps]. The compressibility and reference pressure are set with **compressibility** [bar^{-1}] and **ref_p** [bar], one value is needed.

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. 2 values are needed for x/y and z directions respectively.

anisotropic

Idem, 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** 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** [bar^{-1}] 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** equal to **nstlist**, unless **nstlist** &le 0, then a value of 10 is used. For velocity Verlet integrators **nstpcouple** is set to 1.

tau_p: (1) [ps]

time constant for coupling

compressibility: [bar^{-1}]

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}].

ref_p: [bar]

reference pressure for coupling

refcoord_scaling:

no

The reference coordinates for position restraints are not modified. Note that with this option the virial and pressure will depend on the absolute positions of the reference coordinates.

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.

7.3.16 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.

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.

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 grompp if you are unsure!

7.3.17 Velocity generation

gen_vel:

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

yes
Generate velocities according to a Maxwell distribution at temperature **gen_temp** [K], with random seed **gen_seed**. This is only meaningful with integrator **md**.

gen_temp: (300) [K]
temperature for Maxwell distribution

gen_seed: (173529) [integer]
used to initialize random generator for random velocities, when **gen_seed** is set to -1, the seed is calculated as `(time() + getpid()) % 1000000`

7.3.18 Bonds

constraints:

none
No constraints except for those defined explicitly in the topology, *i.e.* bonds are represented by a harmonic (or other) potential or a Morse potential (depending on the setting of **morse**) and angles by a harmonic (or other) potential.

hbonds
Convert the bonds with H-atoms to constraints.

all-bonds
Convert all bonds to constraints.

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

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

constraint_algorithm:

LINCS
LINear Constraint Solver. With domain decomposition the parallel version P-LINCS is used. The accuracy in set with **lincs_order**, 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**. The root mean square relative constraint deviation is printed to the log file every **nstlog** steps. If a bond rotates more than **lincs_warnangle** [degrees] 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**, 0.0001 is a good value for ”normal” MD. SHAKE does not support constraints between atoms on different nodes, thus it can not be used with domain decompositon when inter charge-group constraints are present. SHAKE can not be used with energy minimization.

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**+1 constraints. When one wants to scale further than this limit, one can decrease **lincs_order** and increase **lincs_iter**, since the accuracy does not deteriorate when (1+**lincs_iter**)*__lincs_order__ 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) [degrees]
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

7.3.19 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.

7.3.20 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=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). Energy groups `wall0` and `wall1` (for **nwall=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_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

tableuser defined potentials indexed with the z distance from the wall, the tables are read analogously to

the **energygrp_table** 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=table**), a fatal error is generated when atoms are beyond a wall.

wall_atomtype:
the atom type name in the force field for each wall, this allows for independent tuning of the interaction of each atomtype with the walls

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=2**, where one should use **ewald_geometry=3dc**. The empty layer in the box serves to decrease the unphysical Coulomb interaction between periodic images.

7.3.21 COM pulling

pull:

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

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.

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_init** and **pull_rate** are not used.

pull_geometry

distance
Pull along the vector connecting the two groups. Components can be selected with **pull_dim**.

direction
Pull in the direction of **pull_vec**.

direction_periodic
As **direction**, but allows the distance to be larger than half the box size. 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 virial.

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_vec**. From the reference group a cylinder is selected around the axis going through the pull group with direction **pull_vec** using two radii. The radius **pull_r1** gives the radius within which all the relative weights are one, between **pull_r1** and **pull_r0** the weights are switched to zero. Mass weighting is also used. Note that the radii 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.

position
Pull to the position of the reference group plus **pull_init** + time***pull_rate*****pull_vec**.

pull_dim: (Y Y Y)
the distance components to be used with geometry **distance** and **position**, also sets which components are printed int the output files

pull_r1: (1) [nm]
the inner radius of the cylinder for geometry **cylinder**

pull_r0: (1) [nm]
the outer radius of the cylinder for geometry **cylinder**

pull_constr_tol: (1e-6)
the relative constraint tolerance for constraint pulling

pull_start

no
do not modify **pull_init**

yes
add the COM distance of the starting conformation to **pull_init**

pull_nstxout: (10)
frequency for writing out the COMs of all the pull group

pull_nstfout: (1)
frequency for writing out the force of all the pulled group

pull_ngroups: (1)
The number of pull groups, not including the reference group. If there is only one group, there is no difference in treatment of the reference and pulled group (except with the cylinder geometry). Below only the pull options for the reference group (ending on 0) and the first group (ending on 1) are given, further groups work analogously, but with the number 1 replaced by the group number.

pull_group0:
The name of the reference group. When this is empty an absolute reference of (0,0,0) 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_weights0:
see **pull_weights1**

pull_pbcatom0: (0)
see **pull_pbcatom1**

pull_group1:
The name of the pull group.

pull_weights1:
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_pbcatom1: (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_pbcatom1**. A value of 0 means that the middle atom (number wise) is used. This parameter is not used with geometry **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_vec1: (0.0 0.0 0.0)
The pull direction. `grompp` normalizes the vector.

pull_init1: (0.0) / (0.0 0.0 0.0) [nm]
The reference distance at t=0. This is a single value, except for geometry **position** which uses a vector.

pull_rate1: (0) [nm/ps]
The rate of change of the reference position.

pull_k1: (0) [kJ mol^{-1} nm^{-2}] / [kJ mol^{-1} nm^{-1}]
The force constant. For umbrella pulling this is the harmonic force constant in [kJ mol^{-1} nm^{-2}]. For constant force pulling this is the force constant of the linear potential, and thus minus (!) the constant force in [kJ mol^{-1} nm^{-1}].

pull_kB1: (pull_k1) [kJ mol^{-1} nm^{-2}] / [kJ mol^{-1} nm^{-1}]
As **pull_k1**, but for state B. This is only used when **free_energy** is turned on. The force constant is then (1 - lambda)***pull_k1** + lambda***pull_kB1**.

7.3.22 NMR refinement

disre:

no
no distance restraints (ignore distance restraint information in topology file)

simple
simple (per-molecule) distance restraints, ensemble averaging can be performed with `mdrun -multi` where the environment variable GMX_DISRE_ENSEMBLE_SIZE sets the number of systems within each ensemble (usually equal to the mdrun -multi value)

ensemble
distance restraints over an ensemble of molecules in one simulation box, should only be used for special cases, such as dimers (this option is not fuctional in the current version of GROMACS)

disre_weighting:

conservative
the forces are the derivative of the restraint potential, this results in an r^{-7} weighting of the atom pairs

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

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 time averaged violation times 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

disre_tau: (0) [ps]
time constant for distance restraints running average

nstdisreout: (100) [steps]
frequency to write the running time averaged and instantaneous distances of all atom pairs involved in restraints to the energy file (can make the energy file very large)

orire:

no
no orientation restraints (ignore orientation restraint information in topology file)

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

orire_fc: (0) [kJ mol]
force constant for orientation restraints, which is multiplied by a (possibly) different 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

orire_fitgrp:
fit group for orientation restraining, for a protein backbone is a good choice

nstorireout: (100) [steps]
frequency to write the running time averaged and instantaneous orientations for all restraints and the molecular order tensor to the energy file (can make the energy file very large)

7.3.23 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 to the energy file and to `dhdl.xvg`. The potentials, bond-lengths and angles are interpolated linearly as described in the manual. When **sc_alpha** is larger than zero, soft-core potentials are used for the LJ and Coulomb interactions.

init_lambda: (0)
starting value for lambda

delta_lambda: (0)
increment per time step for lambda

foreign_lambda: ()
Zero, one or more lambda values for which Delta G values will be determined and written to dhdl.xvg every **nstdhdl** steps. Free energy differences between different lambda values can then be determined with `g_bar`.

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

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

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** 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** and **couple-lambda1**. 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**, but for lambda=1

couple-intramol:

no
All intra-molecular non-bonded interactions for moleculetype **couple-moltype** 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: (10)
the frequency for writing dH/dlambda and possibly Delta H to dhdl.xvg, 0 means no ouput, should be a multiple of **nstcalcenergy**

7.3.24 Non-equilibrium MD

acc_grps:
groups for constant acceleration (*e.g.*: `Protein Sol`) all atoms in groups Protein and Sol will experience constant acceleration as specified in the **accelerate** line

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

freezegrps:
Groups that are to be frozen (*i.e.* their X, Y, and/or Z position will not be updated; *e.g.* `Lipid SOL`). **freezedim** specifies for which dimension the freezing applies. To avoid spurious contibrutions 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 frozen coordinates are not subject to pressure scaling.

freezedim:
dimensions for which groups in **freezegrps** 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** 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** 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.

7.3.25 Electric fields

E_x ; E_y ; E_z:
If you want to use an electric field in a direction, enter 3 numbers after the appropriate E_*, the first number: the number of cosines, only 1 is implemented (with frequency 0) so enter 1, the second number: the strength of the electric field in V nm^{-1}, the third number: the phase of the cosine, you can enter any number here since a cosine of frequency zero has no phase.

E_xt; E_yt; E_zt:
not implemented yet

7.3.26 Mixed quantum/classical molecular dynamics

QMMM:

no
No QM/MM.

yes
Do a QM/MM simulation. Several groups can be described at different QM levels separately. These are specified in the **QMMM-grps** field separated by spaces. The level of ¡i¿ab initio¡/i¿ theory at which the groups are described is speficied by **QMmethod** and **QMbasis** Fields. Describing the groups at different levels of theory is only possible with the ONIOM QM/MM scheme, specified by **QMMMscheme**.

QMMM-grps:
groups to be descibed at the QM level

QMMMscheme:

normal
normal QM/MM. There can only be one **QMMM-grps** that is modelled at the **QMmethod** and **QMbasis** level of ¡i¿ab initio¡/i¿ theory. The rest of the system is described at the MM level. The QM and MM subsystems interact as follows: MM point charges are included in the QM one-electron hamiltonian and all Lennard-Jones interactions are described at the MM level.

ONIOM
The interaction between the subsystem is described using the ONIOM method by Morokuma and coworkers. There can be more than one **QMMM-grps** each modeled at a different level of QM theory (**QMmethod** and **QMbasis**).

QMmethod: (RHF)
Method used to compute the energy and gradients on the QM atoms. Available methods are AM1, PM3, RHF, UHF, DFT, B3LYP, MP2, CASSCF, and MMVB. For CASSCF, the number of electrons and orbitals included in the active space is specified by **CASelectrons** and **CASorbitals**.

QMbasis: (STO-3G)
Basisset used to expand the electronic wavefuntion. Only gaussian bassisets are currently available, ¡i¿*i.e.*¡/i¿ STO-3G, 3-21G, 3-21G*, 3-21+G*, 6-21G, 6-31G, 6-31G*, 6-31+G*, and 6-311G.

QMcharge: (0) [integer]
The total charge in ¡i¿e¡/i¿ of the **QMMM-grps**. In case there are more than one **QMMM-grps**, the total charge of each ONIOM layer needs to be specified separately.

QMmult: (1) [integer]
The multiplicity of the **QMMM-grps**. In case there are more than one **QMMM-grps**, the multiplicity of each ONIOM layer needs to be specified separately.

CASorbitals: (0) [integer]
The number of orbitals to be included in the active space when doing a CASSCF computation.

CASelectrons: (0) [integer]
The number of electrons to be included in the active space when doing a CASSCF computation.

SH:

no
No surface hopping. The system is always in the electronic ground-state.

yes
Do a QM/MM MD simulation on the excited state-potential energy surface and enforce a ¡i¿diabatic¡/i¿ hop to the ground-state when the system hits the conical intersection hyperline in the course the simulation. This option only works in combination with the CASSCF method.

7.3.27 Implicit solvent

implicit_solvent:

no
No implicit solvent

GBSA
Do a simulation with implicit solvent using the Generalized Born formalism. Three different methods for calculating the Born radii are available, Still, HCT and OBC. These are specified with the **gb_algorithm** field.

gb_algorithm:

Still
Use the Still method to calculate the Born radii

HCT
Use the Hawkins-Cramer-Truhlar method to calculate the Born radii

OBC
Use the Onufriev-Bashford-Case method to calculate the Born radii

nstgbradii: (1) [steps]
Frequency to (re)-calculate the Born radii. For most practial purposes, setting a value larger than 1 violates energy conservation and leads to unstable trajectories.

rgbradii: (1.0) [nm]
Cut-off for the calculation of the Born radii. Currently must be equal to rlist

gb_epsilon_solvent: (80)
Dielectric constant for the implicit solvent

gb_saltconc: (0) [M]
Salt concentration for implicit solvent models, currently not used

gb_obc_alpha (1); gb_obc_beta (0.8); gb_obc_gamma (4.85);
Scale factors for the OBC model. Default values are OBC(II). Values for OBC(I) are 0.8, 0 and 2.91 respectively

gb_dielectric_offset: (0.09) [nm]
Distance for the di-electric offset when calculating the Born radii. This is the offset between the center of each atom the center of the polarization energy for the corresponding atom

sa_algorithm

no
Which algorithm is used the the SA part. Note that currently no specific SA algorithm is implemented. With implicit_solvent=GBSA, a very crude ACE-style algorithm is used by default

sa_surface_tension: (2.092) [kj/mol/nm2]
Default values for surface tension with SA algorithms. The value, 2.092, correponds to 0.005 kcal/mol/Angstrom2

7.3.28 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 to your subroutine. Check the inputrec definition in `src/include/types/inputrec.h`

7.4 Programs by topic

Generating topologies and coordinates

pdb2gmx	converts pdb files to topology and coordinate files
g_x2top	generates a primitive topology from coordinates
editconf	edits the box and writes subgroups
genbox	solvates a system
genion	generates mono atomic ions on energetically favorable positions
genconf	multiplies a conformation in 'random' orientations
genrestr	generates position restraints or distance restraints for index groups
g_protonate	protonates structures

Running a simulation

grompp	makes a run input file
tpbconv	makes a run input file for restarting a crashed run
mdrun	performs a simulation, do a normal mode analysis or an energy minimization

Viewing trajectories

ngmx	displays a trajectory
g_highway	X Window System gadget for highway simulations
g_nmtraj	generate a virtual trajectory from an eigenvector

Processing energies

g_energy	writes energies to xvg files and displays averages
g_enemat	extracts an energy matrix from an energy file
mdrun	with -rerun (re)calculates energies for trajectory frames

Converting files

editconf	converts and manipulates structure files
trjconv	converts and manipulates trajectory files
trjcat	concatenates trajectory files
eneconv	converts energy files
xpm2ps	converts XPM matrices to encapsulated postscript (or XPM)
g_sigeps	convert c6/12 or c6/cn combinations to and from sigma/epsilon

Tools

make_ndx	makes index files
mk_angndx	generates index files for g_angle
gmxcheck	checks and compares files
gmxdump	makes binary files human readable
g_traj	plots x, v and f of selected atoms/groups (and more) from a trajectory
g_analyze	analyzes data sets
trjorder	orders molecules according to their distance to a group
g_filter	frequency filters trajectories, useful for making smooth movies
g_lie	free energy estimate from linear combinations
g_dyndom	interpolate and extrapolate structure rotations
g_morph	linear interpolation of conformations
g_wham	weighted histogram analysis after umbrella sampling
xpm2ps	convert XPM (XPixelMap) file to postscript
g_sham	read/write xmgr and xvgr data sets
g_spatial	calculates the spatial distribution function (more control than g_sdf)
g_sdf	calculates the spatial distribution function (faster than g_spatial)
g_select	selects groups of atoms based on flexible textual selections
g_tune_pme	time mdrun as a function of PME nodes to optimize settings

Distances between structures

g_rms	calculates rmsd's with a reference structure and rmsd matrices
g_confrms	fits two structures and calculates the rmsd
g_cluster	clusters structures
g_rmsf	calculates atomic fluctuations

Distances in structures over time

g_mindist	calculates the minimum distance between two groups
g_dist	calculates the distances between the centers of mass of two groups
g_bond	calculates distances between atoms
g_mdmat	calculates residue contact maps
g_polystat	calculates static properties of polymers
g_rmsdist	calculates atom pair distances averaged with power -2, -3 or -6

Mass distribution properties over time

g_traj	plots x, v, f, box, temperature and rotational energy
g_gyrate	calculates the radius of gyration
g_msd	calculates mean square displacements
g_polystat	calculates static properties of polymers
g_rotacf	calculates the rotational correlation function for molecules
g_rdf	calculates radial distribution functions
g_rotmat	plots the rotation matrix for fitting to a reference structure
g_vanhove	calculates Van Hove displacement functions

Analyzing bonded interactions

g_bond	calculates bond length distributions
mk_angndx	generates index files for g_angle
g_angle	calculates distributions and correlations for angles and dihedrals
g_dih	analyzes dihedral transitions

Structural properties

g_hbond	computes and analyzes hydrogen bonds
g_saltbr	computes salt bridges
g_sas	computes solvent accessible surface area
g_order	computes the order parameter per atom for carbon tails
g_principal	calculates axes of inertia for a group of atoms
g_rdf	calculates radial distribution functions
g_sdf	calculates solvent distribution functions
g_sgangle	computes the angle and distance between two groups
g_sorient	analyzes solvent orientation around solutes
g_spol	analyzes solvent dipole orientation and polarization around solutes
g_bundle	analyzes bundles of axes, *e.g.* helices
g_disre	analyzes distance restraints
g_clustsize	calculate size distributions of atomic clusters
g_anadock	cluster structures from Autodock runs

Kinetic properties

g_traj	plots x, v, f, box, temperature and rotational energy
g_velacc	calculates velocity autocorrelation functions
g_tcaf	calculates viscosities of liquids
g_kinetics	derives information about kinetic processes from your trajectories
g_bar	calculates free energy difference estimates through Bennett's acceptance ratio
g_current	calculate current autocorrelation function of system
g_vanhove	compute Van Hove correlation function
g_principal	calculate principal axes of inertion for a group of atoms

Electrostatic properties

genion	generates mono atomic ions on energetically favorable positions
g_potential	calculates the electrostatic potential across the box
g_dipoles	computes the total dipole plus fluctuations
g_dielectric	calculates frequency dependent dielectric constants
g_current	calculates dielectric constants for charged systems
g_spol	analyze dipoles around a solute

Protein specific analysis

do_dssp	assigns secondary structure and calculates solvent accessible surface area
g_chi	calculates everything you want to know about chi and other dihedrals
g_helix	calculates basic properties of alpha helices
g_helixorient	calculates local pitch/bending/rotation/orientation inside helices
g_rama	computes Ramachandran plots
g_xrama	shows animated Ramachandran plots
g_wheel	plots helical wheels

Interfaces

g_potential	calculates the electrostatic potential across the box
g_density	calculates the density of the system
g_densmap	calculates 2D planar or axial-radial density maps
g_order	computes the order parameter per atom for carbon tails
g_h2order	computes the orientation of water molecules
g_bundle	analyzes bundles of axes, *e.g.* transmembrane helices
g_membed	embeds a protein into a lipid bilayer

Covariance analysis

g_covar	calculates and diagonalizes the covariance matrix
g_anaeig	analyzes the eigenvectors
make_edi	generate input files for essential dynamics sampling

Normal modes

grompp	makes a run input file
mdrun	finds a potential energy minimum
mdrun	calculates the Hessian
g_nmeig	diagonalizes the Hessian
g_nmtraj	generate oscillating trajectory of an eigenmode
g_anaeig	analyzes the normal modes
g_nmens	generates an ensemble of structures from the normal modes

Chapter 8

Analysis

In this chapter different ways of analyzing your trajectory are described. The names of the corresponding analysis programs are given. Specific info on the in- and output of these programs can be found in the on-line manual at www.gromacs.org. The output files are often produced as finished Grace/Xmgr graphs.

First in sec. 8.1 the group concept in analysis is explained. Then the different analysis tools are presented.

8.1 Groups in Analysis.

`make_ndx, mk_angndx`

In chapter 3 it was explained how *groups of atoms* can be used in the MD-program. In most analysis programs groups of atoms are needed to work on. Most programs can generate several default index groups, but groups can always be read from an index file. Let's consider 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

$$4\pi r^2 g_{AB}(r) \;=\; V \sum_{i \in A}^{N_A} \sum_{j \in B}^{N_B} P(r) \tag{8.1}$$

where V is the volume and $P(r)$ is the probability to find a B atom at a 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 files (`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 `make_ndx`. To generate an index file with angles or dihedrals, use `mk_angndx`. 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.

8.1.1 Default Groups

When no index file is supplied to analysis tools or `grompp`, 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 hydrogen and hydrogens on the N-terminus

`SideChain`
: protein side chain atoms; that is all atoms except N, C_α, C, O, backbone amide hydrogen, 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. 5.2.2; this group is only included when it differs from the '`Protein`' group

`Non-Protein`
: all non-protein atoms

`DNA`
: all DNA atoms

`molecule_name`
: for all residues/molecules which are not recognized as protein or DNA, one group per residue/molecule name is generated

`Other`
: all atoms which are neither protein nor DNA.

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 `aminoacids.dat` file.

8.1.2 Selections

`g_select`
GROMACS also includes a `g_select` tool that can be used to select atoms based on more flexible criteria than in `make_ndx`, including selecting atoms based on their coordinates. Currently, the tool is experimental and only supports some basic operations, but in the future the functionality is planned to be included in other analysis tools as well. Description of possible ways of selecting atoms can be read by running `g_select` and typing `help` in the selection prompt that appears. It is also possible to write your own analysis tools to take advantage of the flexibility of these selections: see the `template.c` file in the `share/gromacs/template` directory of your installation for an example.

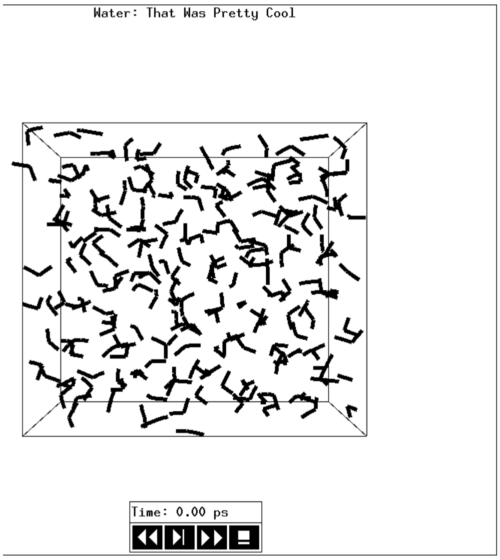


Figure 8.1: The window of `ngmx` showing a box of water.

8.2 Looking at your trajectory

`ngmx`

Before analyzing your trajectory it is often informative to look at your trajectory first. GROMACS comes with a simple trajectory viewer `ngmx`; the advantage with this one is that it does not require OpenGL, which usually isn't present e.g. on supercomputers. It is also possible to generate a hard-copy in Encapsulated Postscript format, see Fig. 8.1. 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 homepage www.gromacs.org for updated links.

8.3 General properties

`g_energy, g_traj`
To analyze some or all *energies* and other properties, such as *total pressure*, *pressure tensor*, *density*, *box-volume* and *box-sizes*, use the program `g_energy`. 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

$$\mathbf{v}_{com} = \frac{1}{M} \sum_{i=1}^{N} m_i \mathbf{v}_i \qquad (8.2)$$

with $M = \sum_{i=1}^{N} m_i$ the total mass of the system, can be monitored in time by the program `g_com`. It is however recommended to remove the center-of-mass velocity every step (see chapter 3)!

8.4 Radial distribution functions

`g_rdf`
The *radial distribution function* (rdf) or pair correlation function $g_{AB}(r)$ between particles of type A and B is defined

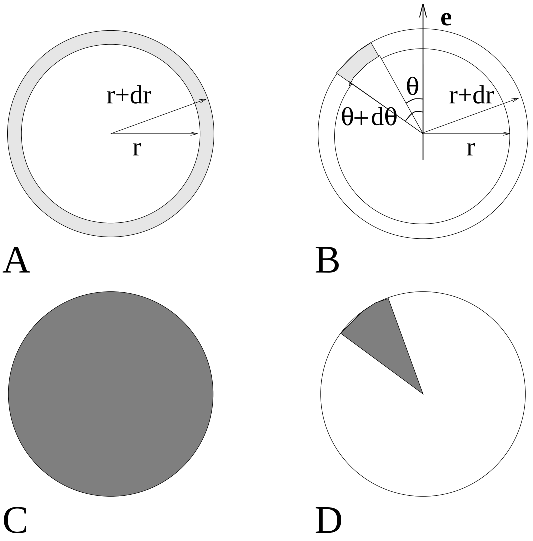


Figure 8.2: Definition of slices in `g_rdf`: A. $g_{AB}(r)$. B. $g_{AB}(r, \theta)$. The slices are colored gray. C. Normalization $\langle \rho_B \rangle_{local}$. D. Normalization $\langle \rho_B \rangle_{local,\, \theta}$. Normalization volumes are colored gray.

in the following way:

$$\begin{aligned} g_{AB}(r) &= \frac{\langle \rho_B(r) \rangle}{\langle \rho_B \rangle_{local}} \\ &= \frac{1}{\langle \rho_B \rangle_{local}} \frac{1}{N_A} \sum_{i \in A}^{N_A} \sum_{j \in B}^{N_B} \frac{\delta(r_{ij} - r)}{4\pi r^2} \end{aligned} \tag{8.3}$$

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. 8.2C).

Usually the value of r_{max} is half of the box length. The averaging is also performed in time. In practice the analysis program `g_rdf` divides the system into spherical slices (from r to $r + dr$, see Fig. 8.2A) and makes a histogram in stead of the δ-function. An example of the rdf of Oxygen-Oxygen in SPC-water [71] is given in Fig. 8.3.

With `g_rdf` 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 $\mathbf{e}$, see Fig. 8.2B.

$$g_{AB}(r,\theta) = \frac{1}{\langle \rho_B \rangle_{local,\, \theta}} \frac{1}{N_A} \sum_{i \in A}^{N_A} \sum_{j \in B}^{N_B} \frac{\delta(r_{ij} - r)\delta(\theta_{ij} - \theta)}{2\pi r^2 sin(\theta)} \tag{8.4}$$

$$cos(\theta_{ij}) = \frac{\mathbf{r}_{ij} \cdot \mathbf{e}}{\|r_{ij}\| \; \|e\|} \tag{8.5}$$

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. 8.2D.

8.5 Correlation functions

8.5.1 Theory of correlation functions

The theory of correlation functions is well established [87]. However we want to 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

$$C_f(t) = \left\langle f(\xi) f(\xi + t) \right\rangle_\xi \tag{8.6}$$

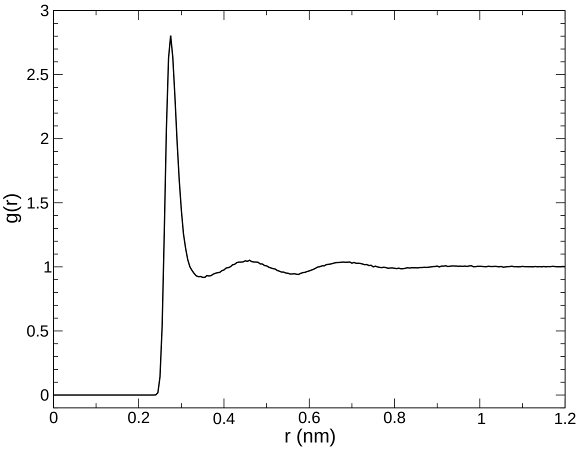


Figure 8.3: $g_{OO}(r)$ for Oxygen-Oxygen of SPC-water.

where the notation on the right hand side means averaging over ξ, *i.e.* over time origins. It is also possible to compute cross-correlation function from two properties $f(t)$ and $g(t)$:

$$C_{fg}(t) \;=\; \langle f(\xi)g(\xi+t)\rangle_\xi \tag{8.7}$$

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:

$$\tau_f \;=\; \int_0^\infty C_f(t)\mathrm{d}t \tag{8.8}$$

In practice correlation functions are calculated based on data points with discrete time intervals Δt, so that the ACF from an MD simulation is:

$$C_f(j\Delta t) \;=\; \frac{1}{N-j}\sum_{i=0}^{N-1-j} f(i\Delta t)f((i+j)\Delta t) \tag{8.9}$$

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 atomic coordinates with short intervals. Another implication of eqn. 8.9 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:

$$C_f(j\Delta t) \;=\; \frac{1}{M}\sum_{i=0}^{N-1-M} f(i\Delta t)f((i+j)\Delta t) \tag{8.10}$$

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 (eqn. 8.6) 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 $kM \leq N$):

$$C_f(j\Delta t) \;=\; \frac{1}{k}\sum_{i=0}^{k-1} f(iM\Delta t)f((iM+j)\Delta t) \tag{8.11}$$

However, one needs very long simulations to get good accuracy this way, because there are many fewer points that contribute to the ACF.

8.5.2 Using FFT for computation of the ACF

The computational cost for calculating an ACF according to eqn. 8.9 is proportional to N^2, which is considerable. However, this can be improved by using fast Fourier transforms to do the convolution [87].

8.5.3 Special forms of the ACF

There are some important varieties on the ACF, *e.g.* the ACF of a vector $\boldsymbol{p}$:

$$C_{\boldsymbol{p}}(t) \;=\; \int_0^{\infty} P_n(\cos\angle\left(\boldsymbol{p}(\xi), \boldsymbol{p}(\xi+t)\right)\mathrm{d}\xi \tag{8.12}$$

where $P_n(x)$ is the n^{th} order Legendre polynomial [1]. Such correlation times can actually be obtained experimentally using *e.g.* NMR or other relaxation experiments. GROMACS can compute correlations using the $1st$ and 2^{nd} order Legendre polynomial (eqn. 8.12). This can a.o. be used for rotational autocorrelation (`g_rotacf`), dipole autocorrelation (`g_dipoles`).

In order to study torsion angle dynamics we define a dihedral autocorrelation function as [117]:

$$C(t) \;=\; \left\langle \cos(\theta(\tau)-\theta(\tau+t))\right\rangle_{\tau} \tag{8.13}$$

Note that this is not a product of two functions as is generally used for correlation functions, but it may be rewritten as the sum of two products:

$$C(t) \;=\; \left\langle \cos(\theta(\tau))\cos(\theta(\tau+t))\,+\,\sin(\theta(\tau))\sin(\theta(\tau+t))\right\rangle_{\tau} \tag{8.14}$$

8.5.4 Some Applications

The program `g_velacc` calculates this *Velocity Auto Correlation Function*.

$$C_{\boldsymbol{v}}(\tau) \;=\; \langle \boldsymbol{v}_i(\tau)\cdot\boldsymbol{v}_i(0)\rangle_{i\in A} \tag{8.15}$$

The self diffusion coefficient can be calculated using the Green-Kubo relation [87]

$$D_A \;=\; \frac{1}{3}\int_0^{\infty}\langle \mathbf{v}_i(t)\cdot\mathbf{v}_i(0)\rangle_{i\in A}\;dt \tag{8.16}$$

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. 8.6), which can also be used for the computation of diffusion constants. However, Allen & Tildesley [87] 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 A is calculated as follows by `g_dipoles`:

$$C_{\mu}(\tau) \;=\; \langle \mu_i(\tau)\cdot\mu_i(0)\rangle_{i\in A} \tag{8.17}$$

with $\mu_i = \sum_{j\in i}\mathbf{r}_j q_j$. The dipole correlation time can be computed using eqn. 8.8. For some applications see [118].

The viscosity of a liquid can be related to the correlation time of the Pressure tensor $\boldsymbol{P}$ [119, 120]. `g_energy` can compute the viscosity, but this is not very accurate [109] (actually the values do not converge...).

8.6 Mean Square Displacement

`g_msd`
To determine the self diffusion coefficient D_A of particles A one can use the Einstein relation [87]

$$\lim_{t\rightarrow\infty}\langle\|\mathbf{r}_i(t)-\mathbf{r}_i(0)\|^2\rangle_{i\in A} \;=\; 6D_A t \tag{8.18}$$

---

[1] $P_0(x)=1$, $P_1(x)=x$, $P_2(x)=(3x^2-1)/2$

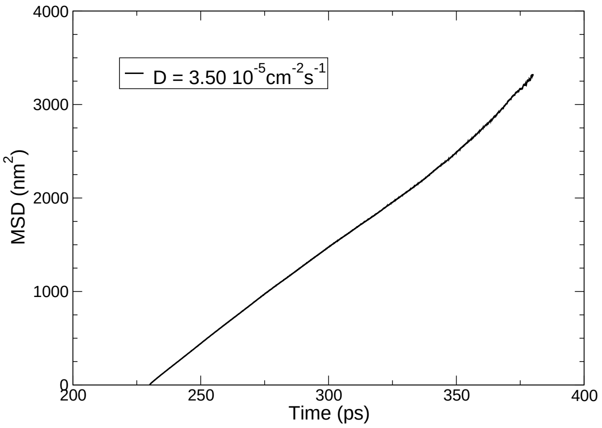


Figure 8.4: Mean Square Displacement of SPC-water.

This *Mean Square Displacement* and D_A are calculated by the program `g_msd`. Normally an index file containing atom numbers is used and the MSD is averaged over 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 `g_msd` 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. 8.4.

8.7 Bonds, angles and dihedrals

`g_bond, g_angle, g_sgangle`
To monitor specific *bonds* in your molecules during time, the program `g_bond` calculates the distribution of the bond length in time. The index file consists of pairs of atom numbers, for example

```
[ bonds_1 ]
1 2
3 4
9 10
[ bonds_2 ]
12 13
```

The program `g_angle` calculates the distribution of *angles* and *dihedrals* in time. It also gives the average angle or dihedral. The index file consists of triplets or quadruples of atom numbers:

```
[ angles ]
1 2 3
2 3 4
3 4 5
[ dihedrals ]
1 2 3 4
2 3 5 5
```

For the dihedral angles you can use either the “biochemical convention” ($\phi = 0 \equiv cis$) or “polymer convention” ($\phi = 0 \equiv trans$), see Fig. 8.5.

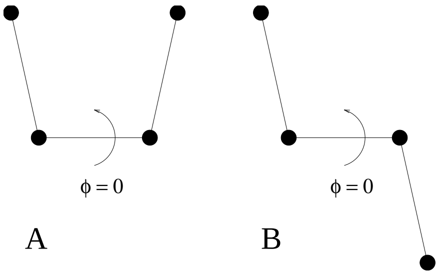


Figure 8.5: Dihedral conventions: A. “Biochemical convention”. B. “Polymer convention”.

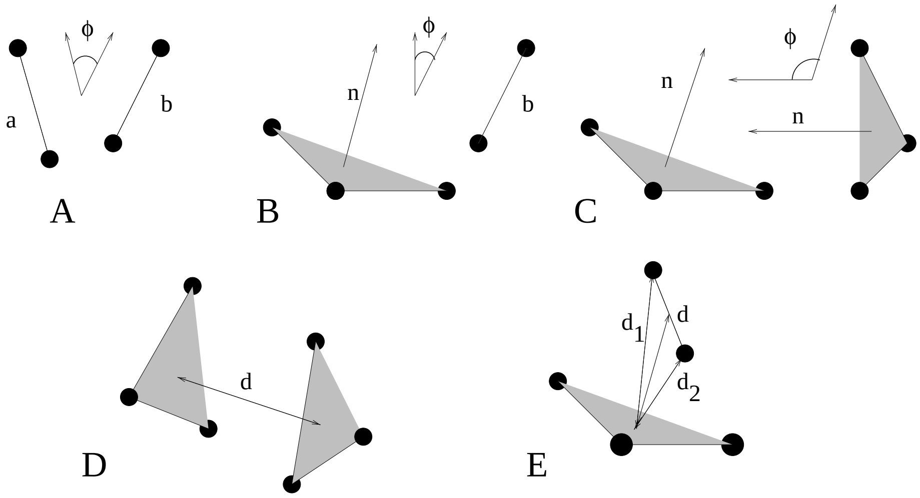


Figure 8.6: Options of `g_sgangle`: A. Angle between 2 vectors. B. Angle between a vector and the normal of a plane. C. Angle between two planes. D. Distance between the geometrical centers of 2 planes. E. Distances between a vector and the center of a plane.

To follow specific *angles* in time between two vectors, a vector and a plane or two planes (defined by 2, resp. 3 atoms inside your molecule, see Fig. 8.6A, B, C), use the program `g_sgangle`.

For planes it uses the normal vector perpendicular to the plane. It can also calculate the *distance* d between the geometrical center of two planes (see Fig. 8.6D), and the distances d_1 and d_2 between 2 atoms (of a vector) and the center of a plane defined by 3 atoms (see Fig. 8.6D). It further calculates the distance d between the center of the plane and the middle of this vector. Depending on the input groups (*i.e.* groups of 2 or 3 atom numbers), the program decides what angles and distances to calculate. For example, the index-file could look like this:

```
[ a_plane ]
1 2 3
[ a_vector ]
3 4 5
```

8.8 Radius of gyration and distances

`g_gyrate, g_sgangle, g_mindist, g_mdmat, xpm2ps`
To have a rough measure for the compactness of a structure, you can calculate the *radius of gyration* with the program

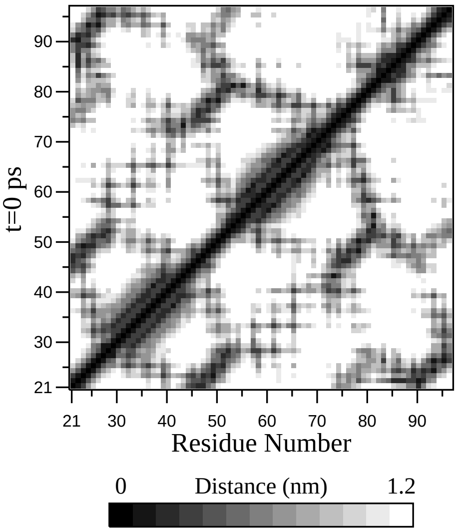


Figure 8.7: A minimum distance matrix for a peptide [121].

`g_gyrate` as follows:

$$R_g = \left(\frac{\sum_i \|\mathbf{r}_i\|^2 m_i}{\sum_i m_i} \right)^{\frac{1}{2}} \tag{8.19}$$

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.

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 `g_sgangle`, as explained in sec. 8.7.
- The *minimum distance* between two groups of atoms during time can be calculated with the program `g_mindist`. It also calculates the *number of contacts* between these groups within a certain radius r_{max}.
- To monitor the *minimum distances between amino-acid residues* within a (protein) molecule, you can use the program `g_mdmat`. This minimum distance between two residues A$_i$ and A$_j$ is defined as the smallest distance between any pair of atoms (i ∈ A$_i$, j ∈ 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` into a Postscript picture, see Fig. 8.7.

  Plotting these matrices for different time-frames, one can analyze changes in the structure, and *e.g.* forming of salt bridges.

8.9 Root mean square deviations in structure

`g_rms, g_rmsdist`

The *root mean square deviation* ($RMSD$) of certain atoms in a molecule with respect to a reference structure can be calculated with the program `g_rms` by least-square fitting the structure to the reference structure ($t_2 = 0$) and subsequently calculating the $RMSD$ (eqn. 8.20).

$$RMSD(t_1, t_2) = \left[\frac{1}{M} \sum_{i=1}^{N} m_i \|\mathbf{r}_i(t_1) - \mathbf{r}_i(t_2)\|^2 \right]^{\frac{1}{2}} \tag{8.20}$$

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 eqn. 8.20 with a structure at time $t_2 = t_1 - \tau$. This gives some insight in the mobility as a function of τ. Also a matrix can be made with the $RMSD$ as a function of t_1 and t_2, this gives a nice graphical impression 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 `g_rmsdist`:

$$RMSD(t) \;=\; \left[\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\|\mathbf{r}_{ij}(t)-\mathbf{r}_{ij}(0)\|^2\right]^{\frac{1}{2}} \tag{8.21}$$

where the *distance* $\mathbf{r}_{ij}$ between atoms at time t is compared with the distance between the same atoms at time 0.

8.10 Covariance analysis

Covariance analysis, also called principal component analysis or essential dynamics [122], can find correlated motions. It uses the covariance matrix C of the atomic coordinates:

$$C_{ij} = \left\langle M_{ii}^{\frac{1}{2}}(x_i - \langle x_i\rangle)M_{jj}^{\frac{1}{2}}(x_j - \langle x_j\rangle)\right\rangle \tag{8.22}$$

where M is a diagonal matrix containing the masses of the atoms (mass-weighted analysis) or the unit matrix (non-mass weighted analysis). C is a symmetric $3N \times 3N$ matrix, which can be diagonalized with an orthonormal transformation matrix R:

$$R^T C R = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{3N}) \quad \text{where} \;\; \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{3N} \tag{8.23}$$

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 $p_i(t)$:

$$\mathbf{p}(t) = R^T M^{\frac{1}{2}}(\mathbf{x}(t) - \langle\mathbf{x}\rangle) \tag{8.24}$$

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:

$$\mathbf{x}^f(t) = \langle\mathbf{x}\rangle + M^{-\frac{1}{2}} R_{*i}\, p_i(t) \tag{8.25}$$

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 [123]. 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:

$$A^{\frac{1}{2}} = R\,\text{diag}(\lambda_1^{\frac{1}{2}}, \lambda_2^{\frac{1}{2}}, \ldots, \lambda_{3N}^{\frac{1}{2}})\, R^T \tag{8.26}$$

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:

$$d(A,B) \;=\; \sqrt{\text{tr}\left(\left(A^{\frac{1}{2}} - B^{\frac{1}{2}}\right)^2\right)} \tag{8.27}$$

$$= \sqrt{\mathrm{tr}\left(A + B - 2A^{\frac{1}{2}}B^{\frac{1}{2}}\right)} \tag{8.28}$$

$$= \left(\sum_{i=1}^{N}\left(\lambda_i^A + \lambda_i^B\right) - 2\sum_{i=1}^{N}\sum_{j=1}^{N}\sqrt{\lambda_i^A\lambda_j^B}\left(R_i^A \cdot R_j^B\right)^2\right)^{\frac{1}{2}} \tag{8.29}$$

where tr is the trace of a matrix. We can now define the overlap s as:

$$s(A,B) = 1 - \frac{d(A,B)}{\sqrt{\mathrm{tr}A + \mathrm{tr}B}} \tag{8.30}$$

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:

$$\mathrm{overlap}(\mathbf{v},\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}(\mathbf{v}_i \cdot \mathbf{w}_j)^2 \tag{8.31}$$

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 proved that the the principal components of random diffusion are cosines with the number of periods equal to half the principal component index[124, 123]. The eigenvalues are proportional to the index to the power -2. The cosine content is defined as:

$$\frac{2}{T}\left(\int_0^T \cos\left(\frac{i\pi t}{T}\right) p_i(t)\mathrm{d}t\right)^2 \left(\int_0^T p_i^2(t)\mathrm{d}t\right)^{-1} \tag{8.32}$$

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 `g_covar`. The principal components and overlap (any many more things) can be plotted and analyzed with `g_anaeig`. The cosine content can be calculated with `g_analyze`.

8.11 Dihedral principal component analysis

`g_angle, g_covar, g_anaeig`
Principal component analysis can be performed in dihedral space [125] using GROMACS. You start by defining the dihedral angles of your interest in an index file, either using `mk_angndx` or otherwise. Then you use the `g_angle` program with the -or flag to produce a new trr file containing the cosine and sine of each dihedral angle in two coordinates respectively. That is, in the trr file you will have a series of numbers corresponding to: cos(ϕ_1), sin(ϕ_1), cos(ϕ_2), sin(ϕ_2), ..., cos(ϕ_n), sin(ϕ_n), the array is padded with zeros if necessary. Then you can use this trr file as input for the `g_covar` program and perform principal component analysis as usual. For this to work you will need to generate a reference file (tpr, gro, pdb etc.) containing the same number of "atoms" as the new trr 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 `g_covar` since the coordinates in the dummy reference file do not correspond in any way to the information in the trr file. Analysis of the results is done using `g_anaeig`.

8.12 Hydrogen bonds

`g_hbond`
The program `g_hbond` analyses the *hydrogen bonds* (H-bonds) between all possible donors D and acceptors A. To

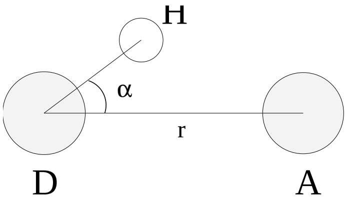


Figure 8.8: Geometrical Hydrogen bond criterion.

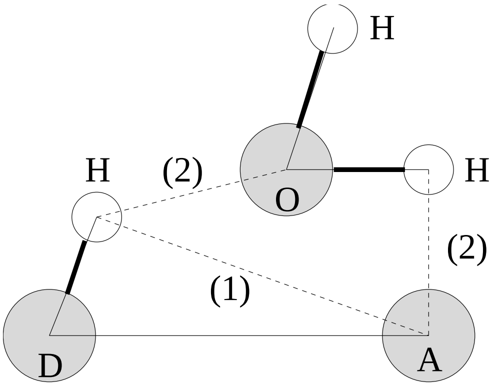


Figure 8.9: Insertion of water into an H-bond. (1) Normal H-bond between two residues. (2) H-bonding bridge via a water molecule.

determine if an H-bond exists, a geometrical criterion is used, see also Fig. 8.8:

$$\begin{array}{rclcl} r & \leq & r_{HB} & = & 0.35\text{nm} \\ \alpha & \leq & \alpha_{HB} & = & 30^o \end{array} \tag{8.33}$$

The value of $r_{HB} = 0.35$ nm corresponds to the first minimum of the rdf of SPC-water (see also Fig. 8.3).

The program `g_hbond` analyses 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.
- 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:

  $$C(\tau) = \langle s_i(t)\, s_i(t+\tau)\rangle \tag{8.34}$$

  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}:

  $$\tau_{HB} = \int_0^{\infty} C(\tau) d\tau \tag{8.35}$$

  Both the integral and the complete auto correlation 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 complicate analysis is given in ref. [126], one of the more fancy option is the Luzar and Chandler analysis of hydrogen bond kinetics [127, 128].

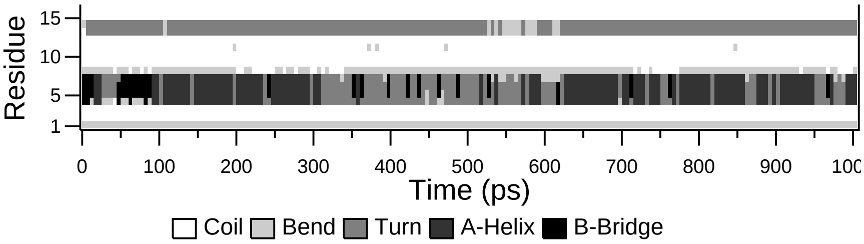


Figure 8.10: Analysis of the secondary structure elements of a peptide in time.

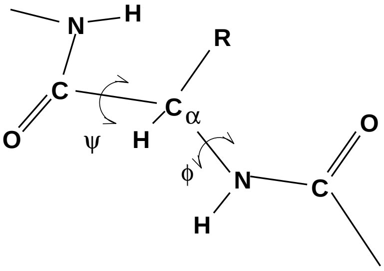


Figure 8.11: Definition of the dihedral angles ϕ and ψ of the protein backbone.

- 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.
- Solvent insertion into H-bonds can be analyzed, see Fig. 8.9. In this case an additional group identifying the solvent must be selected. The occurrence of insertion will be indicated in the existence map. Note that insertion into and existence of a specific H-bond can occur simultaneously and will also be indicated as such in the existence map.

8.13 Protein related items

`do_dssp, g_rama, xrama, wheel`

To analyze structural changes of a protein, you can calculate the radius of gyration or the minimum residue distances during time (see sec. 8.8), or calculate the RMSD (sec. 8.9).

You can also look at the changing of *secondary structure elements* during your run. For this you can use the program `do_dssp`, which is an interface for the commercial program `dssp` [129]. For further information, see the `dssp`-manual. A typical output plot of `do_dssp` is given in Fig. 8.10.

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. 8.11.

To evaluate this Ramachandran plot you can use the program `g_rama`. A typical output is given in Fig. 8.12.

It is also possible to generate an animation of the Ramachandran plot in time. This can be of help for analyzing certain dihedral transitions in your protein. You can use the program `xrama` for this.

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 `wheel` program. Two examples are plotted in Fig. 8.13.

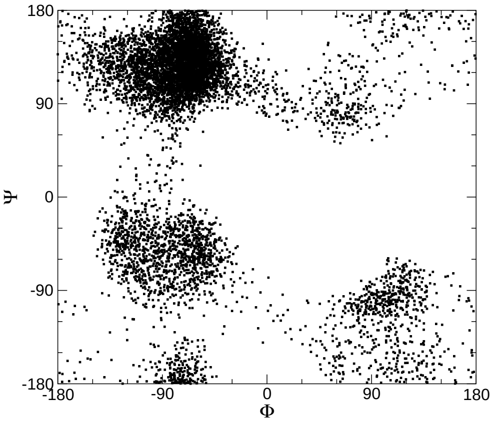


Figure 8.12: Ramachandran plot of a small protein.

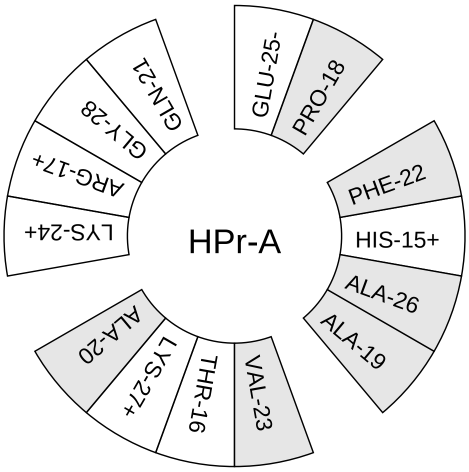


Figure 8.13: Helical wheel projection of the N-terminal helix of HPr.

8.14 Interface related items

`g_order, g_density, g_potential, g_traj`
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 `g_order` can calculate order parameters using the equation

$$S_z = \frac{3}{2}\langle\cos^2\theta_z\rangle - \frac{1}{2} \tag{8.36}$$

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 ($\rho(z)$):

$$\psi(z) - \psi(-\infty) = -\int_{-\infty}^{z} dz' \int_{-\infty}^{z'} \rho(z'')dz''/\epsilon_0 \tag{8.37}$$

where the position $z = -\infty$ is far enough in the bulk phase 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 `g_potential` divides the box in slices and sums all charges of the atoms in each slice. It then integrates this charge density, giving the electric field, and the electric field, giving the potential. Charge density, field and potential are written to `xvgr`-input files.

The program `g_traj` 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 interface systems.

The program `g_pvd` calculates a lot of properties, among which the density of a group in particles per unit of volume, but not a density that takes the mass of the atoms into account. The program `g_density` also calculates the density of a group, but takes the masses into account 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.

8.15 Chemical shifts

`total, do_shift`
You can compute the NMR chemical shifts of protons with the program `do_shift`. This is just an GROMACS interface to the public domain program `total` [130]. For further information, read the article. Although there is limited support for this in GROMACS users are encouraged to use the software provided by the David Case group at Scripps because it seems to be more up-to-date.

Appendix A

Technical Details

A.1 Installation

The entire GROMACS package is Free Software, licensed under the GNU General Public License. The main distribution site is our WWW server at www.gromacs.org.

The package is mainly distributed as source code, but we also provide RPM packages for Linux. On the home page you will find all the information you need to install the package, mailing lists with archives, and several additional on-line resources like contributed topologies, etc. The default installation action is simply to unpack the source code and the issue

```
./configure
make
make install
```

The configuration script should automatically determine the best options for your platform, and it will tell you if anything is missing on your system. You will also find detailed step-by-step installation instructions on the website.

A.2 Single or Double precision

GROMACS can be compiled in either single or double precision. The default choice is single precision, but it is easy to turn on double precision by selecting the `--disable-float` option to the configuration script. Double precision will be 0 to 50% slower than single 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. Assembly loops are available in single and double precision on Pentium 4, Opteron and Itanium processors. On PowerPC processors containing the Altivec unit only single precision is possible. On older Athlon and Pentium 3 processors only the single precision code is available, due to hardware limitations. All other processors use either C or Fortran code for the compute intensive inner loops.

The energies in single 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. B.1). In most cases this is not really a problem, since the fluctuations in the virial can be 2 orders of magnitude larger than the average. In periodic charged systems these errors are often negligible. Especially 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 single precision. Since MD is chaotic, trajectories with very similar starting conditions will diverge rapidly, the divergence is faster in single precision than in double precision.

For most simulations single 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
- calculation of the constraint force between two large groups of atoms
- energy conservation (this can only be done without temperature coupling and without cut-offs)

A.3 Porting GROMACS

The GROMACS system is designed with portability as a major design goal. However there are a number of things we assume to be present on the system GROMACS is being ported on. We assume the following features:

1. A UNIX-like operating system (BSD 4.x or SYSTEM V rev.3 or higher) or UNIX-like libraries running under e.g. Cygwin
2. an ANSI C compiler
3. optionally a Fortran-77 compiler or Fortran-90 compiler for faster (on some computers) inner loop routines
4. optionally the Nasm assembler to use the assembly inner loops on x86 processors.

There are some additional features in the package that require extra stuff to be present, but it is checked for in the configuration script and you will be warned if anything important is missing.

That's the requirements for a single processor system. If you want to compile GROMACS for a multiple processor environment you also need a MPI library (Message-Passing Interface) to perform the parallel communication. This is always shipped with supercomputers, and for workstations you can find links to free MPI implementations through the GROMACS homepage at www.gromacs.org.

A.3.1 Multi-processor Optimization

If you want to, you could write your own optimized communication (perhaps using specific libraries for your hardware) instead of MPI. This should never be necessary for normal use (we haven't heard of a modern computer where it isn't possible to run MPI), but if you absolutely want to do it, here are some clues.

The interface between the communication routines and the rest of the GROMACS system is described in the file `$GMXHOME/src/include/network.h` We will give a short description of the different routines below.

extern void gmx_tx(int pid,void *buf,int bufsize);
This routine, when called with the destination processor number, a pointer to a (byte oriented) transfer buffer, and the size of the buffer will send the buffer to the indicated processor (in our case always the neighboring processor). The routine does **not** wait until the transfer is finished.

extern void gmx_tx_wait(int pid);
This routine waits until the previous, or the ongoing transmission is finished.

extern void gmx_txs(int pid,void *buf,int bufsize);
This routine implements a synchronous send by calling the a-synchronous routine and then the wait. It might come in handy to code this differently.

extern void gmx_rx(int pid,void *buf,int bufsize);

extern void gmx_rx_wait(int pid);

extern void gmx_rxs(int pid,void *buf,int bufsize);
The very same routines for receiving a buffer and waiting until the reception is finished.

extern void gmx_init(int pid,int nprocs);
This routine initializes the different devices needed to do the communication. In general it sets up the communication hardware (if it is accessible) or does an initialize call to the lower level communication subsystem.

extern void gmx_stat(FILE *fp,char *msg);
With this routine we can diagnose the ongoing communication. In the current implementation it prints the various contents of the hardware communication registers of the (Intel *i*860) multiprocessor boards to a file.

A.4 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. Other variables are:

1. `DUMPNL`, dump neighbor list. If set to a positive number the *entire* neighbor list is printed in the log file (may be many megabytes). Mainly for debugging purposes, but may also be handy for porting to other platforms.
2. `GMX_NO_QUOTES`, if this is explicitly set, no cool quotes will be printed at the end of a program
3. `WHERE`, when set print debugging info on line numbers.
4. `LOG_BUFS`, 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.
5. `GMXNPRI`, for SGI systems only. When set, gives the default non-degrading priority (npri) for `mdrun`, `nmrun`, `g_covar` and `g_nmeig`, ***e.g.*** setting `setenv GMXNPRI 250` causes all runs to be performed at near-lowest priority by default.
6. `GMX_VIEW_XPM`, `GMX_VIEW_XVG`, `GMX_VIEW_EPS` and `GMX_VIEW_PDB`, commands used to automatically view resp. `.xvg`, `.xpm`, `.eps` and `.pdb` file types; 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.
7. `GMXTIMEUNIT` the time unit used in output files, can be anything in fs, ps, ns, us, ms, s, m or h.

Some other environment variables are specific to one program, such as TOTAL for the `do_shift` program, and DSSP for the `do_dssp` program.

A.5 Running GROMACS in parallel

If you have installed the MPI (Message Passing Interface) on your computer(s) you can compile GROMACS with this library to run simulations in parallel. All supercomputers are shipped with MPI libraries optimized for that particular platform, and if you are using a cluster of workstations there are several good free MPI implementations. You can find updated links to these on the GROMACS homepage www.gromacs.org. Once you have an MPI library installed it's trivial to compile GROMACS with MPI support: Just set the option `--enable-mpi` to the configure script and recompile. (But don't forget to make distclean before running configure if you have previously compiled with a different configuration.) If you are using a supercomputer you might also want to turn of the default nice-ing of the mdrun process with the `--disable-nice` option.

There is usually a program called `mpirun` with which you can fire up the parallel processes. A typical command line looks like:

```
% mpirun -p goofus,doofus,fred 10 mdrun -s topol -v -N 30
```

this runs on each of the machines goofus,doofus,fred with 10 processes on each[1].

If you have a single machine with multiple processors you don't have to use the `mpirun` command, but you can do with an extra option to `mdrun`:

```
% mdrun -np 8 -s topol -v -N 8
```

In this example MPI reads the first option from the command line. Since `mdrun` also wants to know the number of processes you have to type it twice.

Check your local manuals (or on-line manual) for exact details of your MPI implementation.

If you are interested in programming MPI yourself, you can find manuals and reference literature on the internet.

---

[1] Example taken from Silicon Graphics manual

Appendix B

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.

B.1 Single Sum Virial in GROMACS.

The virial Ξ can be written in full tensor form as:

$$\Xi \;=\; -\frac{1}{2}\sum_{i<j}^{N} \boldsymbol{r}_{ij} \otimes \boldsymbol{F}_{ij} \tag{B.1}$$

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 [131].

B.1.1 Virial.

In a system with Periodic Boundary Conditions, the periodicity must be taken into account for the virial:

$$\Xi \;=\; -\frac{1}{2}\sum_{i<j}^{N} \boldsymbol{r}_{ij}^{n} \otimes \boldsymbol{F}_{ij} \tag{B.2}$$

where $\boldsymbol{r}_{ij}^{n}$ 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 $\boldsymbol{r}_i$ of atom i. The difference vector $\boldsymbol{r}_{ij}^{n}$ is thus equal to:

$$\boldsymbol{r}_{ij}^{n} \;=\; \boldsymbol{r}_i + \delta_i - \boldsymbol{r}_j \tag{B.3}$$

or in shorthand:

$$\boldsymbol{r}_{ij}^{n} \;=\; \boldsymbol{r}_i^{n} - \boldsymbol{r}_j \tag{B.4}$$

In a triclinic system there are 27 possible images of i, when truncated octahedron is used there are 15 possible images.

---

[1] $(\mathbf{u} \otimes \mathbf{v})^{\alpha\beta} \;=\; \mathbf{u}_\alpha \mathbf{v}_\beta$

[2] The calculation of Lennard-Jones and Coulomb forces is about 50 floating point operations.

B.1.2 Virial from non-bonded forces.

Here the derivation for the single sum virial in the *non-bonded force* routine is given. $i \neq j$ in all formulae below.

$$\begin{aligned}
\Xi &= -\frac{1}{2}\sum_{i<j}^{N} \boldsymbol{r}_{ij}^{n} \otimes \boldsymbol{F}_{ij} && \text{(B.5)}\\
&= -\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} (\boldsymbol{r}_i + \delta_i - \boldsymbol{r}_j) \otimes \boldsymbol{F}_{ij} && \text{(B.6)}\\
&= -\frac{1}{4}\sum_{i=1}^{N}\sum_{j=1}^{N} (\boldsymbol{r}_i + \delta_i) \otimes \boldsymbol{F}_{ij} - \boldsymbol{r}_j \otimes \boldsymbol{F}_{ij} && \text{(B.7)}\\
&= -\frac{1}{4}\left(\sum_{i=1}^{N}\sum_{j=1}^{N} (\boldsymbol{r}_i + \delta_i) \otimes \boldsymbol{F}_{ij} \;-\; \sum_{i=1}^{N}\sum_{j=1}^{N} \boldsymbol{r}_j \otimes \boldsymbol{F}_{ij}\right) && \text{(B.8)}\\
&= -\frac{1}{4}\left(\sum_{i=1}^{N} (\boldsymbol{r}_i + \delta_i) \otimes \sum_{j=1}^{N} \boldsymbol{F}_{ij} \;-\; \sum_{j=1}^{N} \boldsymbol{r}_j \otimes \sum_{i=1}^{N} \boldsymbol{F}_{ij}\right) && \text{(B.9)}\\
&= -\frac{1}{4}\left(\sum_{i=1}^{N} (\boldsymbol{r}_i + \delta_i) \otimes \boldsymbol{F}_i \;+\; \sum_{j=1}^{N} \boldsymbol{r}_j \otimes \boldsymbol{F}_j\right) && \text{(B.10)}\\
&= -\frac{1}{4}\left(2\sum_{i=1}^{N} \boldsymbol{r}_i \otimes \boldsymbol{F}_i + \sum_{i=1}^{N} \delta_i \otimes \boldsymbol{F}_i\right) && \text{(B.11)}
\end{aligned}$$

In these formulae we introduced

$$\begin{aligned}
\boldsymbol{F}_i &= \sum_{j=1}^{N} \boldsymbol{F}_{ij} && \text{(B.12)}\\
\boldsymbol{F}_j &= \sum_{i=1}^{N} \boldsymbol{F}_{ji} && \text{(B.13)}
\end{aligned}$$

which is the total force on i resp. j. Because we use Newton's third law

$$\boldsymbol{F}_{ij} = -\boldsymbol{F}_{ji} \qquad \text{(B.14)}$$

we must in the implementation double the term containing the shift δ_i.

B.1.3 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 way again

Find a representation of the bonds as a bidirectional graph.

B.1.4 Virial from Covalent Bonds.

The covalent bond force gives a contribution to the virial, we have

$$\begin{aligned} b &= \|\boldsymbol{r}_{ij}^{n}\| && \text{(B.15)}\\ V_b &= \frac{1}{2}k_b(b-b_0)^2 && \text{(B.16)}\\ \boldsymbol{F}_i &= -\nabla V_b && \text{(B.17)}\\ &= k_b(b-b_0)\frac{\boldsymbol{r}_{ij}^{n}}{b} && \text{(B.18)}\\ \boldsymbol{F}_j &= -\boldsymbol{F}_i && \text{(B.19)} \end{aligned}$$

The virial contribution from the bonds then is

$$\begin{aligned} \Xi_b &= -\frac{1}{2}(\boldsymbol{r}_i^{n}\otimes\boldsymbol{F}_i\ +\ \boldsymbol{r}_j\otimes\boldsymbol{F}_j) && \text{(B.20)}\\ &= -\frac{1}{2}\boldsymbol{r}_{ij}^{n}\otimes\boldsymbol{F}_i && \text{(B.21)} \end{aligned}$$

B.1.5 Virial from Shake.

An important contribution to the virial comes from shake. Satisfying the constraints a force **G** 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:

$$\begin{aligned} \Delta\boldsymbol{r}_i &= \boldsymbol{r}_i(t+\Delta t)-[\boldsymbol{r}_i(t)+\mathbf{v}_i(t-\frac{\Delta t}{2})\Delta t+\frac{\boldsymbol{F}_i}{m_i}\Delta t^2] && \text{(B.22)}\\ \mathbf{G}_i &= \frac{m_i\Delta\boldsymbol{r}_i}{\Delta t^2} && \text{(B.23)} \end{aligned}$$

but 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:

$$\Xi = -\frac{1}{2}\sum_{i}^{N_c}\boldsymbol{r}_i\otimes\boldsymbol{F}_i \qquad \text{(B.24)}$$

where N_c is the number of constrained atoms.

B.2 Optimizations

Here we describe some of the algorithmic optimizations used in GROMACS, apart from parallelism. One of these, the implementation of the 1.0/sqrt(x) function is treated separately in sec. B.3. The most important other optimizations are described below.

B.2.1 Inner Loops for Water

GROMACS users special inner loop 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 [71], *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. 4.1.1) and coulomb (sec. 4.1.3) interactions.
4. Atoms two and three have only coulomb interactions, and equal charges.

These loops also works for the SPC/E [132] and TIP3P [93] water models. And for four site water models similar to TIP4P [93]:

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. 4.1.1) interactions.
4. Atoms two and three have only coulomb (sec. 4.1.3) interactions, and equal charges.
5. Atom four has only coulomb interactions.

The gain 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.

B.2.2 Fortran Code

Unfortunately, Fortran compilers are still better than C-compilers, for most machines anyway. For some machines (*e.g.* SGI Power Challenge) the difference may be up to a factor of 3, in the case of vector computers this may be even larger. Therefore, some of the routines that take up a lot of computer time have been translated into Fortran and even assembly code for Intel and AMD x86 processors. In most cases, the Fortran or assembly loops should be selected automatically by the configure script when appropriate, but you can also tweak this by setting options to the configure script.

B.3 Computation of the 1.0/sqrt function.

B.3.1 Introduction.

The GROMACS project started with the development of a $1/\sqrt{x}$ processor which calculates

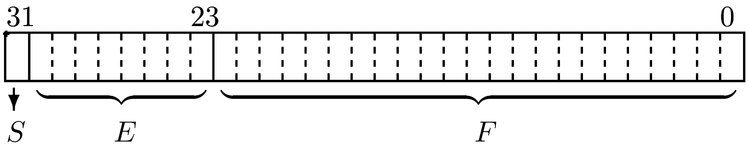
$$Y(x) = \frac{1}{\sqrt{x}} \tag{B.25}$$

As the project continued, the Intel *i*860 processor was used to implement GROMACS, which now turned into almost a full software project. The $1/\sqrt{x}$ processor was implemented using a Newton-Raphson iteration scheme for one step. For this it needed look-up tables to provide the initial approximation. The $1/\sqrt{x}$ function makes it possible to use two almost independent tables for the exponent seed and the fraction seed with the IEEE floating-point representation.

B.3.2 General

According to [133] the $1/\sqrt{x}$ can be calculated using the Newton-Raphson iteration scheme. The inverse function is

$$X(y) = \frac{1}{y^2} \tag{B.26}$$



$$Value = (-1)^S (2^{E-127})(1.F)$$

Figure B.1: IEEE single-precision floating-point format

So instead of calculating

$$Y(a) = q \tag{B.27}$$

the equation

$$X(q) - a = 0 \tag{B.28}$$

can now be solved using Newton-Raphson. An iteration is performed by calculating

$$y_{n+1} = y_n - \frac{f(y_n)}{f'(y_n)} \tag{B.29}$$

The absolute error ε, in this approximation is defined by

$$\varepsilon \equiv y_n - q \tag{B.30}$$

using Taylor series expansion to estimate the error results in

$$\varepsilon_{n+1} = -\frac{\varepsilon_n^2}{2}\frac{f''(y_n)}{f'(y_n)} \tag{B.31}$$

according to [133] equation (3.2). This is an estimation of the absolute error.

B.3.3 Applied to floating-point numbers

floating-point numbers in IEEE 32 bit single-precision format have a nearly constant relative error of $\Delta x/x = 2^{-24}$. As seen earlier in the Taylor series expansion equation (eqn. B.31), the error in every iteration step is absolute and in general dependent of y. If the error is expressed as a relative error ε_r the following holds

$$\varepsilon_{r_{n+1}} \equiv \frac{\varepsilon_{n+1}}{y} \tag{B.32}$$

and so

$$\varepsilon_{r_{n+1}} = -(\frac{\varepsilon_n}{y})^2 y \frac{f''}{2f'} \tag{B.33}$$

for the function $f(y) = y^{-2}$ the term $yf''/2f'$ is constant (equal to $-3/2$) so the relative error ε_{r_n} is independent of y.

$$\varepsilon_{r_{n+1}} = \frac{3}{2}(\varepsilon_{r_n})^2 \tag{B.34}$$

The conclusion of this is that the function $1/\sqrt{x}$ can be calculated with a specified accuracy.

B.3.4 Specification of the look-up table

To calculate the function $1/\sqrt{x}$ using the previously mentioned iteration scheme, it is clear that the first estimation of the solution must be accurate enough to get precise results. The requirements for the calculation are

- Maximum possible accuracy with the used IEEE format
- Use only one iteration step for maximum speed

The first requirement states that the result of $1/\sqrt{x}$ may have a relative error ε_r equal to the ε_r of a IEEE 32 bit single-precision floating-point number. From this the $1/\sqrt{x}$ of the initial approximation can be derived, rewriting the definition of the relative error for succeeding steps, equation (eqn. B.34)

$$\frac{\varepsilon_n}{y} = \sqrt{\varepsilon_{r_{n+1}} \frac{2f'}{yf''}} \tag{B.35}$$

So for the look-up table the needed accuracy is

$$\frac{\Delta Y}{Y} = \sqrt{\frac{2}{3} 2^{-24}} \tag{B.36}$$

which defines the width of the table that must be ≥ 13 bit.

At this point the relative error ε_{r_n} of the look-up table is known. From this the maximum relative error in the argument can be calculated as follows. The absolute error Δx is defined as

$$\Delta x \equiv \frac{\Delta Y}{Y'} \tag{B.37}$$

and thus

$$\frac{\Delta x}{Y} = \frac{\Delta Y}{Y} (Y')^{-1} \tag{B.38}$$

and thus

$$\Delta x = constant \frac{Y}{Y'} \tag{B.39}$$

for the $1/\sqrt{x}$ function $Y/Y' \sim x$ holds, so $\Delta x/x = constant$. This is a property of the used floating-point representation as earlier mentioned. The needed accuracy of the argument of the look-up table follows from

$$\frac{\Delta x}{x} = -2 \frac{\Delta Y}{Y} \tag{B.40}$$

so, using the floating-point accuracy, equation (eqn. B.36)

$$\frac{\Delta x}{x} = -2 \sqrt{\frac{2}{3} 2^{-24}} \tag{B.41}$$

This defines the length of the look-up table which should be ≥ 12 bit.

B.3.5 Separate exponent and fraction computation

The used IEEE 32 bit single-precision floating-point format specifies that a number is represented by a exponent and a fraction. The previous section specifies for every possible floating-point number the look-up table length and width. Only the size of the fraction of a floating-point number defines the accuracy. The conclusion from this can be that the size of the look-up table is length of look-up table, earlier specified, times the size of the exponent $(2^{12}2^{8}, 1Mb)$. The $1/\sqrt{x}$ function has the property that the exponent is independent of the fraction. This becomes clear if the floating-point representation is used. Define

$$x \equiv (-1)^S (2^{E-127})(1.F) \tag{B.42}$$

see Fig. B.1 where $0 \leq S \leq 1$, $0 \leq E \leq 255$, $1 \leq 1.F < 2$ and S, E, F integer (normalization conditions). The sign bit (S) can be omitted because $1/\sqrt{x}$ is only defined for $x > 0$. The $1/\sqrt{x}$ function applied to x results in

$$y(x) = \frac{1}{\sqrt{x}} \tag{B.43}$$

or

$$y(x) = \frac{1}{\sqrt{(2^{E-127})(1.F)}} \tag{B.44}$$

this can be rewritten as

$$y(x) = (2^{E-127})^{-1/2} (1.F)^{-1/2} \tag{B.45}$$

Define

$$(2^{E'-127}) \equiv (2^{E-127})^{-1/2} \tag{B.46}$$

$$1.F' \equiv (1.F)^{-1/2} \tag{B.47}$$

then $\frac{1}{\sqrt{2}} < 1.F' \leq 1$ holds, so the condition $1 \leq 1.F' < 2$ which is essential for normalized real representation is not valid anymore. By introducing an extra term this can be corrected. Rewrite the $1/\sqrt{x}$ function applied to floating-point numbers, equation (eqn. B.45) as

$$y(x) = (2^{\frac{127-E}{2}-1})(2(1.F)^{-1/2}) \tag{B.48}$$

and

$$(2^{E'-127}) \equiv (2^{\frac{127-E}{2}-1}) \tag{B.49}$$

$$1.F' \equiv 2(1.F)^{-1/2} \tag{B.50}$$

then $\sqrt{2} < 1.F \leq 2$ holds. This is not the exact valid range as defined for normalized floating-point numbers in equation (eqn. B.42). The value 2 causes the problem. By mapping this value on the nearest representation < 2 this can be solved. The small error that is introduced by this approximation is within the allowable range.

The integer representation of the exponent is the next problem. Calculating $(2^{\frac{127-E}{2}-1})$ introduces a fractional result if $(127 - E) = odd$. This is again easily accounted for by splitting up the calculation into an odd and an even part. For $(127 - E) = even$ E' in equation (eqn. B.49) can be exactly calculated in integer arithmetic as a function of E.

$$E' = \frac{127 - E}{2} + 126 \tag{B.51}$$

For $(127 - E) = odd$ equation (eqn. B.45) can be rewritten as

$$y(x) = (2^{\frac{127-E-1}{2}})(\frac{1.F}{2})^{-1/2} \tag{B.52}$$

thus

$$E' = \frac{126 - E}{2} + 127 \tag{B.53}$$

which also can be calculated exactly in integer arithmetic. Note that the fraction is automatically corrected for its range earlier mentioned, so the exponent does not need an extra correction.

The conclusions from this are:

- The fraction and exponent look-up table are independent. The fraction look-up table exists of two tables (odd and even exponent) so the odd/even information of the exponent (lsb bit) has to be used to select the right table.
- The exponent table is an 256 x 8 bit table, initialized for odd and $even$.

B.3.6 Implementation

The look-up tables can be generated by a small C program, which uses floating-point numbers and operations with IEEE 32 bit single-precision format. Note that because of the $odd/even$ information that is needed, the fraction table is twice the size earlier specified (13 bit i.s.o. 12 bit).

The function according to equation (eqn. B.29) has to be implemented. Applied to the $1/\sqrt{x}$ function, equation (eqn. B.28) leads to

$$f = a - \frac{1}{y^2} \tag{B.54}$$

and so

$$f' = \frac{2}{y^3} \tag{B.55}$$

so

$$y_{n+1} = y_n - \frac{a - \frac{1}{y_n^2}}{\frac{2}{y_n^3}} \tag{B.56}$$

or

$$y_{n+1} = \frac{y_n}{2}(3 - ay_n^2) \tag{B.57}$$

Where y_0 can be found in the look-up tables, and y_1 gives the result to the maximum accuracy. It is clear that only one iteration extra (in double precision) is needed for a double-precision result.

B.4 Modifying GROMACS

The following files have to be edited in case you want to add a bonded potential of any type.

1. `include/bondf.h`
2. `include/types/idef.h`
3. `include/types/nrnb.h`
4. `include/types/enums.h`
5. `include/grompp.h`
6. `src/kernel/topdirs.c`
7. `src/gmxlib/tpxio.c`
8. `src/gmxlib/bondfree.c`
9. `src/gmxlib/ifunc.c`
10. `src/gmxlib/nrnb.c`
11. `src/kernel/convparm.c`
12. `src/kernel/topdirs.c`
13. `src/kernel/topio.c`

Appendix C

Averages and fluctuations

C.1 Formulae for averaging

Note: this section was taken from ref [134].

When analyzing a MD trajectory averages $\langle x \rangle$ and fluctuations

$$\left\langle (\Delta x)^2 \right\rangle^{\frac{1}{2}} = \left\langle [x - \langle x \rangle]^2 \right\rangle^{\frac{1}{2}} \tag{C.1}$$

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

$$\sigma_x = \sum_{i=1}^{N_x} x_i^2 - \frac{1}{N_x}\left(\sum_{i=1}^{N_x} x_i\right)^2 \tag{C.2}$$

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

$$\sigma_x = \sum_{i=1}^{N_x} [x_i - \langle x \rangle]^2 \tag{C.3}$$

with

$$\langle x \rangle = \frac{1}{N_x} \sum_{i=1}^{N_x} x_i \tag{C.4}$$

Using eqns. C.2 and C.4 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 eqn. C.1 requires only one sequential scan of the series {x$_i$}. However, one may cast eqn. C.2 in another form, containing partial sums, which allows for a sequential update algorithm. Define the partial sum

$$X_{n,m} = \sum_{i=n}^{m} x_i \tag{C.5}$$

and the partial variance

$$\sigma_{n,m} = \sum_{i=n}^{m} \left[x_i - \frac{X_{n,m}}{m-n+1}\right]^2 \tag{C.6}$$

It can be shown that

$$X_{n,m+k} = X_{n,m} + X_{m+1,m+k} \tag{C.7}$$

and

$$\begin{aligned}\sigma_{n,m+k} &= \sigma_{n,m} + \sigma_{m+1,m+k} + \left[\frac{X_{n,m}}{m-n+1} - \frac{X_{n,m+k}}{m+k-n+1}\right]^2 * \\ &\quad \frac{(m-n+1)(m+k-n+1)}{k}\end{aligned} \tag{C.8}$$

For $n = 1$ one finds

$$\sigma_{1,m+k} = \sigma_{1,m} + \sigma_{m+1,m+k} + \left[\frac{X_{1,m}}{m} - \frac{X_{1,m+k}}{m+k}\right]^2 \frac{m(m+k)}{k} \tag{C.9}$$

and for $n = 1$ and $k = 1$ (eqn. C.8) becomes

$$\sigma_{1,m+1} = \sigma_{1,m} + \left[\frac{X_{1,m}}{m} - \frac{X_{1,m+1}}{m+1}\right]^2 m(m+1) \tag{C.10}$$

$$= \sigma_{1,m} + \frac{[\,X_{1,m} - m x_{m+1}\,]^2}{m(m+1)} \tag{C.11}$$

where we have used the relation

$$X_{1,m+1} = X_{1,m} + x_{m+1} \tag{C.12}$$

Using formulae (eqn. C.11) and (eqn. C.12) the average

$$\langle x \rangle = \frac{X_{1,N_x}}{N_x} \tag{C.13}$$

and the fluctuation

$$\left\langle (\Delta x)^2 \right\rangle^{\frac{1}{2}} = \left[\frac{\sigma_{1,N_x}}{N_x}\right]^{\frac{1}{2}} \tag{C.14}$$

can be obtained by one sweep through the data.

C.2 Implementation

In GROMACS the instantaneous energies $E(m)$ are stored in the energy file, 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.

C.2.1 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 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 eqns. C.7 and C.8 the average and standard deviation over part of the trajectory can be computed as:

$$X_{m+1,m+k} = X_{1,m+k} - X_{1,m} \tag{C.15}$$

$$\sigma_{m+1,m+k} = \sigma_{1,m+k} - \sigma_{1,m} - \left[\frac{X_{1,m}}{m} - \frac{X_{1,m+k}}{m+k}\right]^2 \frac{m(m+k)}{k} \tag{C.16}$$

or, more generally (with $p \geq 1$ and $q \geq p$):

$$X_{p,q} = X_{1,q} - X_{1,p-1} \tag{C.17}$$

$$\sigma_{p,q} = \sigma_{1,q} - \sigma_{1,p-1} - \left[\frac{X_{1,p-1}}{p-1} - \frac{X_{1,q}}{q}\right]^2 \frac{(p-1)q}{q-p+1} \tag{C.18}$$

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 eqns. C.11 and C.12:

$$\begin{aligned} X_{1,p-1} &= X_{1,p} - x_p && \text{(C.19)} \\ \sigma_{1,p-1} &= \sigma_{1,p} - \frac{[\,X_{1,p-1} - (p-1)x_p\,]^2}{(p-1)p} && \text{(C.20)} \end{aligned}$$

C.2.2 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:

$$X^{AB}_{1,n+m} = X^A_{1,n} + X^B_{1,m} \tag{C.21}$$

When we want to compute the partial variance from the two components we have to make a correction $\Delta\sigma$:

$$\sigma^{AB}_{1,n+m} = \sigma^A_{1,n} + \sigma^B_{1,m} + \Delta\sigma \tag{C.22}$$

if we define x^{AB}_i as the combined and renumbered set of data points we can write:

$$\sigma^{AB}_{1,n+m} = \sum_{i=1}^{n+m}\left[x^{AB}_i - \frac{X^{AB}_{1,n+m}}{n+m}\right]^2 \tag{C.23}$$

and thus

$$\sum_{i=1}^{n+m}\left[x^{AB}_i - \frac{X^{AB}_{1,n+m}}{n+m}\right]^2 = \sum_{i=1}^{n}\left[x^{A}_i - \frac{X^{A}_{1,n}}{n}\right]^2 + \sum_{i=1}^{m}\left[x^{B}_i - \frac{X^{B}_{1,m}}{m}\right]^2 + \Delta\sigma \tag{C.24}$$

or

$$\begin{aligned} &\sum_{i=1}^{n+m}\left[(x^{AB}_i)^2 - 2x^{AB}_i\frac{X^{AB}_{1,n+m}}{n+m} + \left(\frac{X^{AB}_{1,n+m}}{n+m}\right)^2\right] - \\ &\sum_{i=1}^{n}\left[(x^{A}_i)^2 - 2x^{A}_i\frac{X^{A}_{1,n}}{n} + \left(\frac{X^{A}_{1,n}}{n}\right)^2\right] - \\ &\sum_{i=1}^{m}\left[(x^{B}_i)^2 - 2x^{B}_i\frac{X^{B}_{1,m}}{m} + \left(\frac{X^{B}_{1,m}}{m}\right)^2\right] = \Delta\sigma \end{aligned} \tag{C.25}$$

all the x_i^2 terms drop out, and the terms independent of the summation counter i can be simplified:

$$\begin{aligned} &\frac{\left(X^{AB}_{1,n+m}\right)^2}{n+m} - \frac{\left(X^{A}_{1,n}\right)^2}{n} - \frac{\left(X^{B}_{1,m}\right)^2}{m} - \\ 2\frac{X^{AB}_{1,n+m}}{n+m}&\sum_{i=1}^{n+m}x^{AB}_i + 2\frac{X^{A}_{1,n}}{n}\sum_{i=1}^{n}x^{A}_i + 2\frac{X^{B}_{1,m}}{m}\sum_{i=1}^{m}x^{B}_i = \Delta\sigma \end{aligned} \tag{C.26}$$

we recognize the three partial sums on the second line and use eqn. C.21 to obtain:

$$\Delta\sigma = \frac{\left(mX^A_{1,n} - nX^B_{1,m}\right)^2}{nm(n+m)} \tag{C.27}$$

if we check this by inserting $m = 1$ we get back eqn. C.11

C.2.3 Summing energy terms

The g_energy 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:

$$X^S_{m,n} = \sum_{i=m}^{n} \sum_{s=1}^{S} x_i^s = \sum_{s=1}^{S} \sum_{i=m}^{n} x_i^s = \sum_{s=1}^{S} X^s_{m,n} \tag{C.28}$$

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:

$$\sigma^S_{m,n} = \sum_{s=1}^{S} \sigma^s_{m,n} + \Delta\sigma \tag{C.29}$$

if we fill in eqn. C.6:

$$\sum_{i=m}^{n} \left[\left(\sum_{s=1}^{S} x_i^s \right) - \frac{X^S_{m,n}}{m-n+1} \right]^2 = \sum_{s=1}^{S} \sum_{i=m}^{n} \left[(x_i^s) - \frac{X^s_{m,n}}{m-n+1} \right]^2 + \Delta\sigma \tag{C.30}$$

which we can expand to:

$$\begin{aligned} & \sum_{i=m}^{n} \left[\sum_{s=1}^{S} (x_i^s)^2 + \left(\frac{X^S_{m,n}}{m-n+1} \right)^2 - 2 \left(\frac{X^S_{m,n}}{m-n+1} \sum_{s=1}^{S} x_i^s + \sum_{s=1}^{S} \sum_{s'=s+1}^{S} x_i^s x_i^{s'} \right) \right] \\ - & \sum_{s=1}^{S} \sum_{i=m}^{n} \left[(x_i^s)^2 - 2 \frac{X^s_{m,n}}{m-n+1} x_i^s + \left(\frac{X^s_{m,n}}{m-n+1} \right)^2 \right] = \Delta\sigma \end{aligned} \tag{C.31}$$

the terms with $(x_i^s)^2$ cancel, so that we can simplify to:

$$\begin{aligned} & \frac{\left(X^S_{m,n}\right)^2}{m-n+1} - 2 \frac{X^S_{m,n}}{m-n+1} \sum_{i=m}^{n} \sum_{s=1}^{S} x_i^s - 2 \sum_{i=m}^{n} \sum_{s=1}^{S} \sum_{s'=s+1}^{S} x_i^s x_i^{s'} - \\ & \sum_{s=1}^{S} \sum_{i=m}^{n} \left[-2 \frac{X^s_{m,n}}{m-n+1} x_i^s + \left(\frac{X^s_{m,n}}{m-n+1} \right)^2 \right] = \Delta\sigma \end{aligned} \tag{C.32}$$

or

$$- \frac{\left(X^S_{m,n}\right)^2}{m-n+1} - 2 \sum_{i=m}^{n} \sum_{s=1}^{S} \sum_{s'=s+1}^{S} x_i^s x_i^{s'} + \sum_{s=1}^{S} \frac{\left(X^s_{m,n}\right)^2}{m-n+1} = \Delta\sigma \tag{C.33}$$

If we now expand the first term using eqn. C.28 we obtain:

$$- \frac{\left(\sum_{s=1}^{S} X^s_{m,n}\right)^2}{m-n+1} - 2 \sum_{i=m}^{n} \sum_{s=1}^{S} \sum_{s'=s+1}^{S} x_i^s x_i^{s'} + \sum_{s=1}^{S} \frac{\left(X^s_{m,n}\right)^2}{m-n+1} = \Delta\sigma \tag{C.34}$$

which we can reformulate to:

$$-2 \left[\sum_{s=1}^{S} \sum_{s'=s+1}^{S} X^s_{m,n} X^{s'}_{m,n} + \sum_{i=m}^{n} \sum_{s=1}^{S} \sum_{s'=s+1}^{S} x_i^s x_i^{s'} \right] = \Delta\sigma \tag{C.35}$$

or

$$-2 \left[\sum_{s=1}^{S} X^s_{m,n} \sum_{s'=s+1}^{S} X^{s'}_{m,n} + \sum_{s=1}^{S} \sum_{i=m}^{n} x_i^s \sum_{s'=s+1}^{S} x_i^{s'} \right] = \Delta\sigma \tag{C.36}$$

which gives

$$-2 \sum_{s=1}^{S} \left[X^s_{m,n} \sum_{s'=s+1}^{S} \sum_{i=m}^{n} x_i^{s'} + \sum_{i=m}^{n} x_i^s \sum_{s'=s+1}^{S} x_i^{s'} \right] = \Delta\sigma \tag{C.37}$$

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 eqn. C.30. 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.

Appendix D

Manual Pages

D.1 options

All GROMACS programs have 6 standard options, of which some are hidden by default:

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel

- If the configuration script found Motif or Lesstif on your system, you can use the graphical interface (if not, you will get an error):
  `-X` bool `no` Use dialog box GUI to edit command line options
- When compiled on an SGI-IRIX system, all GROMACS programs have an additional option:
  `-npri` int `0` Set non blocking priority (try 128)
- Optional files are not used unless the option is set, in contrast to non optional files, where the default file name is used when the option is not set.
- All GROMACS programs will accept file options without a file extension or filename being specified. In such cases the default filenames will be used. With multiple input file types, such as generic structure format, the directory will be searched for files of each type with the supplied or default name. When no such file is found, or with output files the first file type will be used.
- All GROMACS programs with the exception of `mdrun`, `nmrun` and `eneconv` check if the command line options are valid. If this is not the case, the program will be halted.
- Enumerated options (enum) should be used with one of the arguments listed in the option description, the argument may be abbreviated. The first match to the shortest argument in the list will be selected.
- Vector options can be used with 1 or 3 parameters. When only one parameter is supplied the two others are also set to this value.
- For many GROMACS programs, the time options can be supplied in different time units, depending on the setting of the `-tu` option.
- All GROMACS programs can read compressed or g-zipped files. There might be a problem with reading compressed `.xtc`, `.trr` and `.trj` files, but these will not compress very well anyway.

- Most GROMACS programs can process a trajectory with less 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.
- Many GROMACS programs will accept the `-tu` option to set the time units to use in output files (e.g. for `xmgr` graphs or `xpm` matrices) and in all time options.

D.2 do_dssp

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. do_dssp assumes that the dssp executable is /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
```

The structure assignment for each residue and time is written to an `.xpm` 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 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 `g_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 `g_chi`. Together these two programs can be used to analyze dihedral properties as a function of secondary structure type.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-ssdump`	`ssdump.dat`	Output, Opt.	Generic data file
`-map`	`ss.map`	Input, Lib.	File that maps matrix data to colors
`-o`	`ss.xpm`	Output	X PixMap compatible matrix file
`-sc`	`scount.xvg`	Output	xvgr/xmgr file
`-a`	`area.xpm`	Output, Opt.	X PixMap compatible matrix file
`-ta`	`totarea.xvg`	Output, Opt.	xvgr/xmgr file
`-aa`	`averarea.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-sss`	string	`HEBT`	Secondary structures for structure count

D.3 editconf

editconf converts generic structure format to `.gro`, `.g96` or `.pdb`.

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.

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. The volume of a dodecahedron is 0.71 and that of a truncated octahedron is 0.77 of that of a cubic box with the same periodic image distance.

Option `-box` requires only one value for a cubic box, dodecahedron and a truncated octahedron.

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 can not 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, 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 file is given as input. A special feature of the scaling option, when the factor -1 is given in one dimension, one obtains a mirror image, mirrored in one of the plains, 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 sizes at the bottom of your input file are correct when the periodicity is to be removed.

When writing `.pdb` 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 an index is larger than the number of 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 (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 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:

```
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.

Files

`-f`	`conf.gro`	Input	Structure file: gro g96 pdb tpr etc.
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`out.gro`	Output, Opt.	Structure file: gro g96 pdb etc.
`-mead`	`mead.pqr`	Output, Opt.	Coordinate file for MEAD
`-bf`	`bfact.dat`	Input, Opt.	Generic data file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-ndef`	bool	`no`	Choose output from default index groups
`-bt`	enum	`triclinic`	Box type for -box and -d: `triclinic`, `cubic`, `dodecahedron` or `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
`-c`	bool	`no`	Center molecule in box (implied by -box and -d)
`-center`	vector	`0 0 0`	Coordinates of geometrical center
`-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
`-princ`	bool	`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
`-pbc`	bool	`no`	Remove the periodicity (make molecule whole again)
`-grasp`	bool	`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
`-sig56`	real	`0`	Use rmin/2 (minimum in the Van der Waals potential) rather than sigma/2
`-vdwread`	bool	`no`	Read the Van der Waals radii from the file vdwradii.dat rather than computing the radii based on the force field
`-atom`	bool	`no`	Force B-factor attachment per atom
`-legend`	bool	`no`	Make B-factor legend
`-label`	string	`A`	Add chain label for all residues
`-conect`	bool	`no`	Add CONECT records to a pdb file when written. Can only be done when a topology is present

- For complex molecules, the periodicity removal routine may break down,
- in that case you can use trjconv.

D.4 eneconv

With *multiple files* specified for the `-f` option:
Concatenates several energy 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 the command `eneconv -o fixed.edr *.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 extentions).

`-settime` is applied first, then `-dt`/`-offset` followed by `-b` and `-e` to select which frames to write.

Files

`-f`	`ener.edr`	Input, Mult.	Energy file
`-o`	`fixed.edr`	Output	Energy file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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
`-settime`	bool	`no`	Change starting time interactively
`-sort`	bool	`yes`	Sort energy files (not frames)
`-scalefac`	real	`1`	Multiply energy component by this factor
`-error`	bool	`yes`	Stop on errors in the file

- 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.

D.5 g_anadock

anadock analyses the results of an Autodock run and clusters the structures together, based on distance or RMSD. The docked energy and free energy estimates are analysed, and for each cluster the energy statistics are printed.

An alternative approach to this is to cluster the structures first (using `g_cluster` and then sort the clusters on either lowest energy or average energy.

Files

`-f`	`eiwit.pdb`	Input	Protein data bank file
`-ox`	`cluster.pdb`	Output	Protein data bank file
`-od`	`edocked.xvg`	Output	xvgr/xmgr file
`-of`	`efree.xvg`	Output	xvgr/xmgr file
`-g`	`anadock.log`	Output	Log file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-free`	bool	`no`	Use Free energy estimate from autodock for sorting the classes
`-rms`	bool	`yes`	Cluster on RMS or distance
`-cutoff`	real	`0.2`	Maximum RMSD/distance for belonging to the same cluster

D.6 g_anaeig

`g_anaeig` analyzes eigenvectors. The eigenvectors can be of a covariance matrix (`g_covar`) or of a Normal Modes analysis (`g_nmeig`).

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 `g_analyze`.

`-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` file with two or three structures (you can use `rasmol -nmrpdb` to view such a pdb 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`, `-eig`, `-v2` and `-eig2` are given, a single number for the overlap between the covariance matrices is generated. 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.

Files

`-v`	`eigenvec.trr`	Input	Full precision trajectory: trr trj cpt
`-v2`	`eigenvec2.trr`	Input, Opt.	Full precision trajectory: trr trj cpt
`-f`	`traj.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-eig`	`eigenval.xvg`	Input, Opt.	xvgr/xmgr file

`-eig2`	`eigenval2.xvg`	Input, Opt.	xvgr/xmgr file
`-comp`	`eigcomp.xvg`	Output, Opt.	xvgr/xmgr file
`-rmsf`	`eigrmsf.xvg`	Output, Opt.	xvgr/xmgr file
`-proj`	`proj.xvg`	Output, Opt.	xvgr/xmgr file
`-2d`	`2dproj.xvg`	Output, Opt.	xvgr/xmgr file
`-3d`	`3dproj.pdb`	Output, Opt.	Structure file: gro g96 pdb etc.
`-filt`	`filtered.xtc`	Output, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-extr`	`extreme.pdb`	Output, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-over`	`overlap.xvg`	Output, Opt.	xvgr/xmgr file
`-inpr`	`inprod.xpm`	Output, Opt.	X PixMap compatible matrix file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-first`	int	`1`	First eigenvector for analysis (-1 is select)
`-last`	int	`8`	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
`-split`	bool	`no`	Split eigenvector projections where time is zero
`-entropy`	bool	`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.

D.7 g_analyze

g_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.

g_analyze always shows the average and standard deviation of each set. For each set it also shows 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).

Option `-cc` plots the resemblance of set i with a cosine of i/2 periods. The formula is:
2 (int0-T y(t) cos(i pi t) dt)^2 / int0-T y(t) y(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 (a (tau1 ((exp(-t/tau1) - 1) tau1/t + 1)) +
(1-a) (tau2 ((exp(-t/tau2) - 1) tau2/t + 1)))),
where T is the total time. a, tau1 and tau2 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 tau1 + (1-a) tau2)). The complete derivation is given in B. Hess, J. Chem. Phys. 116:209-217, 2002.

Option `-bal` finds and subtracts the ultrafast "ballistic" component from a hydrogen bond autocorrelation function by the fitting of a sum of exponentials, as described in e.g. O. Markovitch, J. Chem. Phys. 129:084505, 2008. The fastest term is the one with the most negative coefficient in the exponential, or with `-d`, the one with most negative time derivative at time 0. `-nbalexp` sets the number of exponentials to fit.

Option `-gem` fits bimolecular rate constants ka and kb (and optionally kD) to the hydrogen bond autocorrelation function according to the reversible geminate recombination model. Removal of the ballistic component first is strongly adviced. The model is presented in O. Markovitch, J. Chem. Phys. 129:084505, 2008.

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 tâ, which is accomplished by fitting to a t + b on log-log scale. All points after the first zero or negative value are ignored.

Option `-luzar` performs a Luzar & Chandler kinetics analysis on output from `g_hbond`. The input file can be taken directly from `g_hbond -ac`, and then the same result should be produced.

Files

`-f`	`graph.xvg`	Input	xvgr/xmgr file
`-ac`	`autocorr.xvg`	Output, Opt.	xvgr/xmgr file
`-msd`	`msd.xvg`	Output, Opt.	xvgr/xmgr file
`-cc`	`coscont.xvg`	Output, Opt.	xvgr/xmgr file
`-dist`	`distr.xvg`	Output, Opt.	xvgr/xmgr file
`-av`	`average.xvg`	Output, Opt.	xvgr/xmgr file
`-ee`	`errest.xvg`	Output, Opt.	xvgr/xmgr file
`-bal`	`ballisitc.xvg`	Output, Opt.	xvgr/xmgr file
`-g`	`fitlog.log`	Output, Opt.	Log file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit

`-nice`	int	`0`	Set the nicelevel
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-time`	bool	`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 # sets separated by &
`-d`	bool	`no`	Use the derivative
`-bw`	real	`0.1`	Binwidth for the distribution
`-errbar`	enum	`none`	Error bars for -av: `none`, `stddev`, `error` or `90`
`-integrate`	bool	`no`	Integrate data function(s) numerically using trapezium rule
`-aver_start`	real	`0`	Start averaging the integral from here
`-xydy`	bool	`no`	Interpret second data set as error in the y values for integrating
`-regression`	bool	`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 - A0 x0 - A1 x1 - ... - AN xN)^2 where now Y is the first data set in the input file and xi the others. Do read the information at the option `-time`.
`-luzar`	bool	`no`	Do a Luzar and Chandler analysis on a correlation function and related as produced by g_hbond. 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
`-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
`-smooth`	real	`-1`	If >= 0, the tail of the ACF will be smoothed by fitting it to an exponential function: y = A exp(-x/tau)
`-filter`	real	`0`	Print the high-frequency fluctuation after filtering with a cosine filter of length #
`-power`	bool	`no`	Fit data to: b tâ
`-subav`	bool	`yes`	Subtract the average before autocorrelating
`-oneacf`	bool	`no`	Calculate one ACF over all sets
`-acflen`	int	`-1`	Length of the ACF, default is half the number of frames
`-normalize`	bool	`yes`	Normalize ACF
`-P`	enum	`0`	Order of Legendre polynomial for ACF (0 indicates none): `0`, `1`, `2` or `3`
`-fitfn`	enum	`none`	Fit function: `none`, `exp`, `aexp`, `exp_exp`, `vac`, `exp5`, `exp7` or `exp9`
`-ncskip`	int	`0`	Skip N points in the output file of correlation functions
`-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

D.8 g_angle

g_angle computes the angle distribution for a number of angles or dihedrals. This way you can check whether your simulation is correct. 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, the rest are the individual angles.

With the -of option g_angle also calculates the fraction of trans dihedrals (only for dihedrals) as function of time, but this is probably only fun for a selected few.

With option -oc a dihedral correlation function is calculated.

It should be noted that the indexfile should contain atom-triples 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 PCA analysis using `g_covar`.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`angle.ndx`	Input	Index file
`-od`	`angdist.xvg`	Output	xvgr/xmgr file
`-ov`	`angaver.xvg`	Output, Opt.	xvgr/xmgr file
`-of`	`dihfrac.xvg`	Output, Opt.	xvgr/xmgr file
`-ot`	`dihtrans.xvg`	Output, Opt.	xvgr/xmgr file
`-oh`	`trhisto.xvg`	Output, Opt.	xvgr/xmgr file
`-oc`	`dihcorr.xvg`	Output, Opt.	xvgr/xmgr file
`-or`	`traj.trr`	Output, Opt.	Trajectory in portable xdr format

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-type`	enum	`angle`	Type of angle to analyse: `angle`, `dihedral`, `improper` or `ryckaert-bellemans`
`-all`	bool	`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
`-periodic`	bool	`yes`	Print dihedral angles modulo 360 degrees
`-chandler`	bool	`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.
`-avercorr`	bool	`no`	Average the correlation functions for the individual angles/dihedrals
`-acflen`	int	`-1`	Length of the ACF, default is half the number of frames
`-normalize`	bool	`yes`	Normalize ACF
`-P`	enum	`0`	Order of Legendre polynomial for ACF (0 indicates none): `0`, `1`, `2` or `3`
`-fitfn`	enum	`none`	Fit function: `none`, `exp`, `aexp`, `exp_exp`, `vac`, `exp5`, `exp7` or `exp9`
`-ncskip`	int	`0`	Skip N points in the output file of correlation functions
`-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

- Counting transitions only works for dihedrals with multiplicity 3

D.9 g_bar

g_bar calculates free energy difference estimates through Bennett's acceptance ratio method. Input option `-f` expects multiple dhdl files. Two types of input files are supported:
Files with only one y-value, for such 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.
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's from the legends of Delta H.

The lambda of the simulation is parsed from dhdl.xvg file's legend containing the string 'dH', the foreign lambda's from the legend containing the capitalized letters 'D' and 'H'. The temperature is parsed from the legend line containing 'T ='.

The free energy estimates are determined using BAR with bisection, the precision of the output is 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 blocks numbers for error estimation can be provided with the options `-nbmin` and `-nbmax`.

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_A: 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 (2009) 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 gives an estimate of the quality of sampling (not directly of the actual statistical error, because it assumes independent samples).

Files

`-f`	`dhdl.xvg`	Input, Mult.	xvgr/xmgr file
`-o`	`bar.xvg`	Output, Opt.	xvgr/xmgr file
`-oi`	`barint.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `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

D.10 g_bond

g_bond makes a distribution of bond lengths. If all is well a gaussian distribution should be made when using a harmonic potential. Bonds are read from a single group in the index file in order i1-j1 i2-j2 through in-jn.

`-tol` gives the half-width of the distribution as a fraction of the bondlength (`-blen`). That means, for a bond of 0.2 a tol of 0.1 gives a distribution from 0.18 to 0.22.

Option `-d` plots all the distances as a function of time. This requires a structure file for the atom and residue names in the output. If however the option `-averdist` is given (as well or separately) the average bond length is plotted instead.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input	Index file
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-o`	`bonds.xvg`	Output	xvgr/xmgr file
`-l`	`bonds.log`	Output, Opt.	Log file
`-d`	`distance.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-blen`	real	`-1`	Bond length. By default length of first bond
`-tol`	real	`0.1`	Half width of distribution as fraction of blen
`-aver`	bool	`yes`	Average bond length distributions
`-averdist`	bool	`yes`	Average distances (turns on -d)

- It should be possible to get bond information from the topology.

D.11 g_bundle

g_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 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-ol`	`bun_len.xvg`	Output	xvgr/xmgr file
`-od`	`bun_dist.xvg`	Output	xvgr/xmgr file
`-oz`	`bun_z.xvg`	Output	xvgr/xmgr file
`-ot`	`bun_tilt.xvg`	Output	xvgr/xmgr file
`-otr`	`bun_tiltr.xvg`	Output	xvgr/xmgr file
`-otl`	`bun_tiltl.xvg`	Output	xvgr/xmgr file
`-ok`	`bun_kink.xvg`	Output, Opt.	xvgr/xmgr file
`-okr`	`bun_kinkr.xvg`	Output, Opt.	xvgr/xmgr file
`-okl`	`bun_kinkl.xvg`	Output, Opt.	xvgr/xmgr file
`-oa`	`axes.pdb`	Output, Opt.	Protein data bank file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-na`	int	`0`	Number of axes
`-z`	bool	`no`	Use the Z-axis as reference iso the average axis

D.12 g_chi

g_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
(a) information about the number of residues of each type.
(b) The NMR 3J coupling constants from the Karplus equation.
(c) a table for each residue of the number of transitions between rotamers per nanosecond, and the order parameter S2 of each dihedral.
(d) a table for each residue of the rotamer occupancy.

All rotamers are taken as 3-fold, except for omegas and chi-dihedrals to planar groups (i.e. chi2 of aromatics asp and asn, chi3 of glu and gln, and chi4 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 S2 order parameters are also output to an xvg file (argument `-o`) and optionally as a pdb file with the S2 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 files.

If `-chi_prod` is set (and maxchi > 0), cumulative rotamers, e.g. 1+9(chi1-1)+3(chi2-1)+(chi3-1) (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.

Files

`-s`	`conf.gro`	Input	Structure file: gro g96 pdb tpr etc.
`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-o`	`order.xvg`	Output	xvgr/xmgr file
`-p`	`order.pdb`	Output, Opt.	Protein data bank file
`-ss`	`ssdump.dat`	Input, Opt.	Generic data file
`-jc`	`Jcoupling.xvg`	Output	xvgr/xmgr file
`-corr`	`dihcorr.xvg`	Output, Opt.	xvgr/xmgr file
`-g`	`chi.log`	Output	Log file
`-ot`	`dihtrans.xvg`	Output, Opt.	xvgr/xmgr file
`-oh`	`trhisto.xvg`	Output, Opt.	xvgr/xmgr file
`-rt`	`restrans.xvg`	Output, Opt.	xvgr/xmgr file
`-cp`	`chiprodhisto.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-r0`	int	`1`	starting residue
`-phi`	bool	`no`	Output for Phi dihedral angles
`-psi`	bool	`no`	Output for Psi dihedral angles
`-omega`	bool	`no`	Output for Omega dihedrals (peptide bonds)
`-rama`	bool	`no`	Generate Phi/Psi and Chi1/Chi2 ramachandran plots
`-viol`	bool	`no`	Write a file that gives 0 or 1 for violated Ramachandran angles
`-periodic`	bool	`yes`	Print dihedral angles modulo 360 degrees
`-all`	bool	`no`	Output separate files for every dihedral.
`-rad`	bool	`no`	in angle vs time files, use radians rather than degrees.
`-shift`	bool	`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` or `6`
`-normhisto`	bool	`yes`	Normalize histograms
`-ramomega`	bool	`no`	compute average omega as a function of phi/psi and plot it in an xpm plot
`-bfact`	real	`-1`	B-factor value for pdb file for atoms with no calculated dihedral order parameter
`-chi_prod`	bool	`no`	compute a single cumulative rotamer for each residue
`-HChi`	bool	`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
`-normalize`	bool	`yes`	Normalize ACF
`-P`	enum	`0`	Order of Legendre polynomial for ACF (0 indicates none): `0`, `1`, `2` or `3`
`-fitfn`	enum	`none`	Fit function: `none`, `exp`, `aexp`, `exp_exp`, `vac`, `exp5`, `exp7` or `exp9`
`-ncskip`	int	`0`	Skip N points in the output file of correlation functions
`-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

- 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 g_rama.
- -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

D.13 g_cluster

g_cluster can cluster structures with several different methods. Distances between structures can be determined from a trajectory or read from an XPM 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.

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 eleminate 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.
`-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`).

Files

`-f`	`traj.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-dm`	`rmsd.xpm`	Input, Opt.	X PixMap compatible matrix file
`-o`	`rmsd-clust.xpm`	Output	X PixMap compatible matrix file
`-g`	`cluster.log`	Output	Log file
`-dist`	`rmsd-dist.xvg`	Output, Opt.	xvgr/xmgr file
`-ev`	`rmsd-eig.xvg`	Output, Opt.	xvgr/xmgr file
`-sz`	`clust-size.xvg`	Output, Opt.	xvgr/xmgr file
`-tr`	`clust-trans.xpm`	Output, Opt.	X PixMap compatible matrix file
`-ntr`	`clust-trans.xvg`	Output, Opt.	xvgr/xmgr file
`-clid`	`clust-id.xvg`	Output, Opt.	xvgr/xmgr file
`-cl`	`clusters.pdb`	Output, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-dista`	bool	`no`	Use RMSD of distances instead of RMS deviation
`-nlevels`	int	`40`	Discretize RMSD matrix in # levels
`-cutoff`	real	`0.1`	RMSD cut-off (nm) for two structures to be neighbor
`-fit`	bool	`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
`-av`	bool	`no`	Write average iso middle structure for each cluster
`-wcl`	int	`0`	Write all structures for first # clusters to numbered files
`-nst`	int	`1`	Only write all structures if more than # 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` or `gromos`
`-minstruct`	int	`1`	Minimum number of structures in cluster for coloring in the xpm file
`-binary`	bool	`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	`1993`	Random number seed for Monte Carlo clustering algorithm
`-niter`	int	`10000`	Number of iterations for MC
`-kT`	real	`0.001`	Boltzmann weighting factor for Monte Carlo optimization (zero turns off uphill steps)

D.14 g_clustsize

This program computes the size distributions of molecular/atomic clusters in the gas phase. The output is given in the form of a XPM file. The total number of clusters is written to a XVG 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 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input, Opt.	Portable xdr run input file
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`csize.xpm`	Output	X PixMap compatible matrix file
`-ow`	`csizew.xpm`	Output	X PixMap compatible matrix file
`-nc`	`nclust.xvg`	Output	xvgr/xmgr file
`-mc`	`maxclust.xvg`	Output	xvgr/xmgr file
`-ac`	`avclust.xvg`	Output	xvgr/xmgr file
`-hc`	`histo-clust.xvg`	Output	xvgr/xmgr file
`-temp`	`temp.xvg`	Output, Opt.	xvgr/xmgr file
`-mcn`	`maxclust.ndx`	Output, Opt.	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`

`-cut`	real	`0.35`	Largest distance (nm) to be considered in a cluster
`-mol`	bool	`no`	Cluster molecules rather than atoms (needs tpr file)
`-pbc`	bool	`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 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

D.15 g_confrms

g_confrms computes the root mean square deviation (RMSD) of two structures after LSQ 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` file the two structures will be written as separate models (use `rasmol -nmrpdb`). Also in a `.pdb` file, B-factors calculated from the atomic MSD values can be written with `-bfac`.

Files

`-f1`	`conf1.gro`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-f2`	`conf2.gro`	Input	Structure file: gro g96 pdb tpr etc.
`-o`	`fit.pdb`	Output	Structure file: gro g96 pdb etc.
`-n1`	`fit1.ndx`	Input, Opt.	Index file
`-n2`	`fit2.ndx`	Input, Opt.	Index file
`-no`	`match.ndx`	Output, Opt.	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-one`	bool	`no`	Only write the fitted structure to file
`-mw`	bool	`yes`	Mass-weighted fitting and RMSD
`-pbc`	bool	`no`	Try to make molecules whole again
`-fit`	bool	`yes`	Do least squares superposition of the target structure to the reference
`-name`	bool	`no`	Only compare matching atom names
`-label`	bool	`no`	Added chain labels A for first and B for second structure
`-bfac`	bool	`no`	Output B-factors from atomic MSD values

D.16 g_covar

`g_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 as timestamp.

The eigenvectors can be analyzed with `g_anaeig`.

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 file.

Option `-xpma` writes the atomic covariance matrix to an xpm file, i.e. for each atom pair the sum of the xx, yy and zz covariances is written.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`eigenval.xvg`	Output	xvgr/xmgr file
`-v`	`eigenvec.trr`	Output	Full precision trajectory: trr trj cpt
`-av`	`average.pdb`	Output	Structure file: gro g96 pdb etc.
`-l`	`covar.log`	Output	Log file
`-ascii`	`covar.dat`	Output, Opt.	Generic data file
`-xpm`	`covar.xpm`	Output, Opt.	X PixMap compatible matrix file
`-xpma`	`covara.xpm`	Output, Opt.	X PixMap compatible matrix file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-fit`	bool	`yes`	Fit to a reference structure
`-ref`	bool	`no`	Use the deviation from the conformation in the structure file instead of from the average
`-mwa`	bool	`no`	Mass-weighted covariance analysis
`-last`	int	`-1`	Last eigenvector to write away (-1 is till the last)
`-pbc`	bool	`yes`	Apply corrections for periodic boundary conditions

D.17 g_current

This 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 also allows to get 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 resuls from the fit allow to determine 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 options 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.

Files

`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-o`	`current.xvg`	Output	xvgr/xmgr file
`-caf`	`caf.xvg`	Output, Opt.	xvgr/xmgr file
`-dsp`	`dsp.xvg`	Output	xvgr/xmgr file
`-md`	`md.xvg`	Output	xvgr/xmgr file
`-mj`	`mj.xvg`	Output	xvgr/xmgr file
`-mc`	`mc.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-sh`	int	`1000`	Shift of the frames for averaging the correlation functions and the mean-square displacement.
`-nojump`	bool	`yes`	Removes jumps of atoms across the box.
`-eps`	real	`0`	Dielectric constant of the surrounding medium. eps=0.0 corresponds to eps=infinity (thinfoil 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*tb̂.
`-evit`	real	`5`	End of the fit of the current autocorrelation function to a*tb̂.
`-tr`	real	`0.25`	Fraction of the trajectory taken into account for the integral.
`-temp`	real	`300`	Temperature for calculating epsilon.

D.18 g_density

Compute partial densities across the box, using an index file. Densities in kg/m^3, number densities or electron densities can 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input, Opt.	Index file
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-ei`	`electrons.dat`	Input, Opt.	Generic data file
`-o`	`density.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-d`	string	`Z`	Take the normal on the membrane in direction X, Y or Z.
`-sl`	int	`50`	Divide the box in #nr slices.
`-dens`	enum	`mass`	Density: `mass`, `number`, `charge` or `electron`
`-ng`	int	`1`	Number of groups to compute densities of
`-symm`	bool	`no`	Symmetrize the density along the axis, with respect to the center. Useful for bilayers.
`-center`	bool	`no`	Shift the center of mass along the axis to zero. This means if your axis is Z and your box is bX, bY, bZ, the center of mass will be at bX/2, bY/2, 0.

- When calculating electron densities, atomnames are used instead of types. This is bad.

D.19 g_densmap

g_densmap computes 2D number-density maps. It can make planar and axial-radial density maps. The output `.xpm` 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 file.

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`.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-od`	`densmap.dat`	Output, Opt.	Generic data file
`-o`	`densmap.xpm`	Output	X PixMap compatible matrix file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-bin`	real	`0.02`	Grid size (nm)
`-aver`	enum	`z`	The direction to average over: `z`, `y` or `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
`-mirror`	bool	`no`	Add the mirror image below the axial axis
`-sums`	bool	`no`	Print density sums (1D map) to stdout
`-unit`	enum	`nm-3`	Unit for the output: `nm-3`, `nm-2` or `count`
`-dmin`	real	`0`	Minimum density in output
`-dmax`	real	`0`	Maximum density in output (0 means calculate it)

D.20 g_dielectric

dielectric calculates frequency dependent dielectric constants from the autocorrelation function of the total dipole moment in your simulation. This ACF can be generated by g_dipoles. For an estimate of the error you can run g_statistics on the ACF, and use the output thus generated for this program. The functional forms of the available functions are:

One parameter : y = Exp[-a1 x], Two parameters : y = a2 Exp[-a1 x], Three parameters: y = a2 Exp[-a1 x] + (1 - a2) Exp[-a3 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.

Files

`-f`	`dipcorr.xvg`	Input	xvgr/xmgr file
`-d`	`deriv.xvg`	Output	xvgr/xmgr file
`-o`	`epsw.xvg`	Output	xvgr/xmgr file
`-c`	`cole.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-fft`	bool	`no`	use fast fourier transform for correlation function
`-x1`	bool	`yes`	use first column as X axis rather than first data set
`-eint`	real	`5`	Time were 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`	Epsilon 0 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`, `vac`, `exp5`, `exp7` or `exp9`
`-nsmooth`	int	`3`	Number of points for smoothing

D.21 g_dih

g_dih can do two things. The default is to analyze dihedral transitions by merely computing all the dihedral angles defined in your topology for the whole trajectory. When a dihedral flips over to another minimum an angle/time plot is made.

The opther option is to discretize the dihedral space into a number of bins, and group each conformation in dihedral space in the appropriate bin. The output is then given as a number of dihedral conformations sorted according to occupancy.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-o`	`hello.out`	Output	Generic output file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit

`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-sa`	bool	`no`	Perform cluster analysis in dihedral space instead of analysing dihedral transitions.
`-mult`	int	`-1`	mulitiplicity for dihedral angles (by default read from topology)

D.22 g_dipoles

g_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 $<$ orMuor^2 $>$ and or$<$ Mu $>$or^2 during the simulation. The file dipdist.xvg contains the distribution of dipole moments during the simulation The mu_max 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, JCP 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

g_dipoles -corr mol -P1 -o dip_sqr -mu 2.273 -mumax 5.0 -nofft

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.

Files

`-en`	`ener.edr`	Input, Opt.	Energy file
`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`Mtot.xvg`	Output	xvgr/xmgr file
`-eps`	`epsilon.xvg`	Output	xvgr/xmgr file
`-a`	`aver.xvg`	Output	xvgr/xmgr file
`-d`	`dipdist.xvg`	Output	xvgr/xmgr file
`-c`	`dipcorr.xvg`	Output, Opt.	xvgr/xmgr file
`-g`	`gkr.xvg`	Output, Opt.	xvgr/xmgr file
`-adip`	`adip.xvg`	Output, Opt.	xvgr/xmgr file
`-dip3d`	`dip3d.xvg`	Output, Opt.	xvgr/xmgr file
`-cos`	`cosaver.xvg`	Output, Opt.	xvgr/xmgr file
`-cmap`	`cmap.xpm`	Output, Opt.	X PixMap compatible matrix file

`-q`	`quadrupole.xvg`	Output, Opt.	xvgr/xmgr file
`-slab`	`slab.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-mu`	real	`-1`	dipole of a single molecule (in Debye)
`-mumax`	real	`5`	max dipole in Debye (for histrogram)
`-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` or `total`
`-pairs`	bool	`yes`	Calculate orcos thetaor between all pairs of molecules. May be slow
`-ncos`	int	`1`	Must be 1 or 2. Determines whether the <cos> is computed between all mole cules in one group, or between molecules in two different groups. This turns on the -gkr flag.
`-axis`	string	`Z`	Take the normal on the computational box in direction X, Y or Z.
`-sl`	int	`10`	Divide the box in #nr 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 criterium based on the box length will be used.
`-phi`	bool	`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 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 camp output (for 180 degrees)
`-acflen`	int	`-1`	Length of the ACF, default is half the number of frames
`-normalize`	bool	`yes`	Normalize ACF
`-P`	enum	`0`	Order of Legendre polynomial for ACF (0 indicates none): `0`, `1`, `2` or `3`
`-fitfn`	enum	`none`	Fit function: `none`, `exp`, `aexp`, `exp_exp`, `vac`, `exp5`, `exp7` or `exp9`
`-ncskip`	int	`0`	Skip N points in the output file of correlation functions
`-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

D.23 g_disre

g_disre computes violations of distance restraints. If necessary all protons can be added to a protein molecule using the protonate program.

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 for printing.

When the optional `-q` flag is given a pdb 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.

Files

`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-ds`	`drsum.xvg`	Output	xvgr/xmgr file
`-da`	`draver.xvg`	Output	xvgr/xmgr file
`-dn`	`drnum.xvg`	Output	xvgr/xmgr file
`-dm`	`drmax.xvg`	Output	xvgr/xmgr file
`-dr`	`restr.xvg`	Output	xvgr/xmgr file
`-l`	`disres.log`	Output	Log file
`-n`	`viol.ndx`	Input, Opt.	Index file
`-q`	`viol.pdb`	Output, Opt.	Protein data bank file
`-c`	`clust.ndx`	Input, Opt.	Index file
`-x`	`matrix.xpm`	Output, Opt.	X PixMap compatible matrix file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `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
`-third`	bool	`yes`	Use inverse third power averaging or linear for matrix output

D.24 g_dist

g_dist can calculate the distance between the centers of mass of two groups of atoms as a function of time. The total distance and its x, y and z components are plotted.

Or when `-dist` is set, print all the atoms in group 2 that are closer than a certain distance to the center of mass of group 1.

With options `-lt` and `-dist` the number of contacts of all atoms in group 2 that are closer than a certain distance to the center of mass of group 1 are plotted as a function of the time that the contact was continously present.

Other programs that calculate distances are `g_mindist` and `g_bond`.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`dist.xvg`	Output, Opt.	xvgr/xmgr file
`-lt`	`lifetime.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-dist`	real	`0`	Print all atoms in group 2 closer than dist to the center of mass of group 1

D.25 g_dyndom

g_dyndom reads a pdb file output from DynDom http://www.cmp.uea.ac.uk/dyndom/ It reads the coordinates, and the coordinates of the rotation axis furthermore it reads an index file containing the domains. Furthermore it takes the first and last atom of the arrow file as command line arguments (head and tail) and finally it takes the translation vector (given in DynDom info file) and the angle of rotation (also as command line arguments). If the angle determined by DynDom is given, one should be able to recover the second structure used for generating the DynDom output. Because of limited numerical accuracy this should be verified by computing an all-atom RMSD (using `g_confrms`) rather than by file comparison (using diff).

The purpose of this program is to interpolate and extrapolate the rotation as found by DynDom. As a result unphysical structures with long or short bonds, or overlapping atoms may be produced. Visual inspection, and energy minimization may be necessary to validate the structure.

Files

`-f`	`dyndom.pdb`	Input	Protein data bank file
`-o`	`rotated.xtc`	Output	Trajectory: xtc trr trj gro g96 pdb
`-n`	`domains.ndx`	Input	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-firstangle`	real	`0`	Angle of rotation about rotation vector
`-lastangle`	real	`0`	Angle of rotation about rotation vector
`-nframe`	int	`11`	Number of steps on the pathway
`-maxangle`	real	`0`	DymDom dtermined angle of rotation about rotation vector
`-trans`	real	`0`	Translation (Aangstroem) along rotation vector (see DynDom info file)
`-head`	vector	`0 0 0`	First atom of the arrow vector
`-tail`	vector	`0 0 0`	Last atom of the arrow vector

D.26 genbox

Genbox can do one of 3 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, eg. 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 `editconf` 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 VanderWaals radii of both atoms. A database (`vdwradii.dat`) of VanderWaals radii is read by the program, atoms not in the database are assigned a default distance `-vdwd`. Note that this option will also influence the distances between solvent molecules if they contain atoms that are not in the database.

3) Insert a number (`-nmol`) of extra molecules (`-ci`) at random positions. The program iterates until `nmol` molecules have been inserted in the box. To test whether an insertion is successful the same VanderWaals criterium is used as for removal of solvent molecules. When no appropriately sized holes (holes that can hold an extra molecule) are available the program tries for `-nmol` * `-try` times before giving up. Increase -try if you have several small holes to fill.

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 would have fit into the box.

The program can optionally rotate the solute molecule to align the longest molecule axis along a box edge. This way the amount of solvent molecules necessary is reduced. It should be kept in mind that this only works for short simulations, as eg. an alpha-helical peptide in solution can rotate over 90 degrees, within 500 ps. In general it is therefore better to make a more or less cubic box.

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 editconf).

Finally, genbox 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.

Files

`-cp`	`protein.gro`	Input, Opt.	Structure file: gro g96 pdb tpr etc.
`-cs`	`spc216.gro`	Input, Opt., Lib.	Structure file: gro g96 pdb tpr etc.
`-ci`	`insert.gro`	Input, Opt.	Structure file: gro g96 pdb tpr etc.
`-o`	`out.gro`	Output	Structure file: gro g96 pdb etc.
`-p`	`topol.top`	In/Out, Opt.	Topology file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-box`	vector	`0 0 0`	box size
`-nmol`	int	`0`	no of extra molecules to insert
`-try`	int	`10`	try inserting -nmol*-try times
`-seed`	int	`1997`	random generator seed
`-vdwd`	real	`0.105`	default vdwaals distance
`-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
`-vel`	bool	`no`	keep velocities from input solute and solvent

- Molecules must be whole in the initial configurations.

D.27 genconf

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.

Files

`-f`	`conf.gro`	Input	Structure file: gro g96 pdb tpr etc.
`-o`	`out.gro`	Output	Structure file: gro g96 pdb etc.
`-trj`	`traj.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-nbox`	vector	`1 1 1`	Number of boxes
`-dist`	vector	`0 0 0`	Distance between boxes
`-seed`	int	`0`	Random generator seed, if 0 generated from the time
`-rot`	bool	`no`	Randomly rotate conformations
`-shuffle`	bool	`no`	Random shuffling of molecules
`-sort`	bool	`no`	Sort molecules on X coord
`-block`	int	`1`	Divide the box in blocks on this number of cpus
`-nmolat`	int	`3`	Number of atoms per molecule, assumed to start from 0. If you set this wrong, it will screw up your system!
`-maxrot`	vector	`180 180 180`	Maximum random rotation
`-renumber`	bool	`yes`	Renumber residues

- The program should allow for random displacement of lattice points.

D.28 g_enemat

g_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 `g_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) = E0 + kT log(<exp((E-E0)/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.

Files

`-f`	`ener.edr`	Input, Opt.	Energy file
`-groups`	`groups.dat`	Input	Generic data file
`-eref`	`eref.dat`	Input, Opt.	Generic data file
`-emat`	`emat.xpm`	Output	X PixMap compatible matrix file
`-etot`	`energy.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-sum`	bool	`no`	Sum the energy terms selected rather than display them all
`-skip`	int	`0`	Skip number of frames between data points
`-mean`	bool	`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
`-coul`	bool	`yes`	extract Coulomb SR energies
`-coulr`	bool	`no`	extract Coulomb LR energies
`-coul14`	bool	`no`	extract Coulomb 1-4 energies
`-lj`	bool	`yes`	extract Lennard-Jones SR energies
`-lj`	bool	`no`	extract Lennard-Jones LR energies
`-lj14`	bool	`no`	extract Lennard-Jones 1-4 energies
`-bhamsr`	bool	`no`	extract Buckingham SR energies
`-bhamlr`	bool	`no`	extract Buckingham LR energies
`-free`	bool	`yes`	calculate free energy
`-temp`	real	`300`	reference temperature for free energy calculation

D.29 g_energy

g_energy extracts energy components or distance restraint data from an energy file. The user is prompted to interactively select the energy terms she wants.

Average, RMSD and drift are calculated with full precision from the simulation (see printed manual). Drift is calculated by performing a LSQ 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 g_energy statistics output more accurate than the xvg output. When exact averages are not present in the energy file the statistics mentioned above is simply over the single, per-frame energy values.

The term fluctuation gives the RMSD around the LSQ fit.

Some fluctuation-dependent properties can be calculated provided the correct energy terms are selected. The following properties will be computed:
Property Energy terms needed
————————————————————

Heat capacity Cp (NPT sims) Enthalpy, Temp
Heat capacity Cv (NVT sims) Etot, Temp
Thermal expansion coeff. (NPT) Enthalpy, Vol, Temp
Isothermal compressibility Vol, Temp
Adiabatic bulk modulus Vol, Temp [PBR] ————————————————————
You always need to set the number of molecules `-nmol`, and, if you used constraints in your simulations you will need to give the number of constraints per molecule `-nconstr` in order to correct for this: (nconstr/2) kB is subtracted from the heat capacity in this case. For instance in the case of rigid water you need to give the value 3 to this option.

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.

With `-fee` an estimate is calculated for the free-energy difference with an ideal gas state:
Delta A = A(N,V,T) - A_idgas(N,V,T) = kT ln < e(̂Upot/kT) >
Delta G = G(N,p,T) - G_idgas(N,p,T) = kT ln < e(̂Upot/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_idgas(N,V,T) = (<Upot> - Delta A)/T
Delta S(N,p,T) = S(N,p,T) - S_idgas(N,p,T) = (<Upot> + pV - Delta G)/T

When a second energy file is specified (`-f2`), a free energy difference is calculated dF = -kT ln < e ^(EB-EA)/kT >A , where EA and EB are the energies from the first and second energy files, and the average is over the ensemble A. **NOTE** that the energies must both be calculated from the same trajectory.

Files

`-f`	`ener.edr`	Input	Energy file
`-f2`	`ener.edr`	Input, Opt.	Energy file
`-s`	`topol.tpr`	Input, Opt.	Run input file: tpr tpb tpa

`-o`	`energy.xvg`	Output	xvgr/xmgr file
`-viol`	`violaver.xvg`	Output, Opt.	xvgr/xmgr file
`-pairs`	`pairs.xvg`	Output, Opt.	xvgr/xmgr file
`-ora`	`orienta.xvg`	Output, Opt.	xvgr/xmgr file
`-ort`	`orientt.xvg`	Output, Opt.	xvgr/xmgr file
`-oda`	`orideva.xvg`	Output, Opt.	xvgr/xmgr file
`-odr`	`oridevr.xvg`	Output, Opt.	xvgr/xmgr file
`-odt`	`oridevt.xvg`	Output, Opt.	xvgr/xmgr file
`-oten`	`oriten.xvg`	Output, Opt.	xvgr/xmgr file
`-corr`	`enecorr.xvg`	Output, Opt.	xvgr/xmgr file
`-vis`	`visco.xvg`	Output, Opt.	xvgr/xmgr file
`-ravg`	`runavgdf.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-b`	time	`0`	First frame (ps) to read from trajectory
`-e`	time	`0`	Last frame (ps) to read from trajectory
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-fee`	bool	`no`	Do a free energy estimate
`-fetemp`	real	`300`	Reference temperature for free energy calculation
`-zero`	real	`0`	Subtract a zero-point energy
`-sum`	bool	`no`	Sum the energy terms selected rather than display them all
`-dp`	bool	`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
`-mutot`	bool	`no`	Compute the total dipole moment from the components
`-skip`	int	`0`	Skip number of frames between data points
`-aver`	bool	`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
`-nconstr`	int	`0`	Number of constraints per molecule. Necessary for calculating the heat capacity
`-fluc`	bool	`no`	Calculate autocorrelation of energy fluctuations rather than energy itself
`-orinst`	bool	`no`	Analyse instantaneous orientation data
`-ovec`	bool	`no`	Also plot the eigenvectors with -oten
`-acflen`	int	`-1`	Length of the ACF, default is half the number of frames
`-normalize`	bool	`yes`	Normalize ACF
`-P`	enum	`0`	Order of Legendre polynomial for ACF (0 indicates none): `0`, `1`, `2` or `3`
`-fitfn`	enum	`none`	Fit function: `none`, `exp`, `aexp`, `exp_exp`, `vac`, `exp5`, `exp7` or `exp9`
`-ncskip`	int	`0`	Skip N points in the output file of correlation functions
`-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

D.30 genion

genion replaces solvent molecules by monoatomic ions at the position of the first atoms with the most favorable electrostatic potential or at random. The potential is calculated on all atoms, using normal GROMACS particle based methods (in contrast to other methods based on solving the Poisson-Boltzmann equation). The potential is recalculated after every ion insertion. If specified in the run input file, a reaction field, shift function or user function can be used. For the user function a table file can be specified with the option `-table`. 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. Ions which can have multiple charge states get the multiplicilty added, without sign, for the uncommon states only.

With the option `-pot` the potential can be written as B-factors in a pdb file (for visualisation using e.g. rasmol). The unit of the potential is 1000 kJ/(mol e), the scaling be changed with the `-scale` option.

For larger ions, e.g. sulfate we recommended to use genbox.

Files

`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-table`	`table.xvg`	Input, Opt.	xvgr/xmgr file
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`out.gro`	Output	Structure file: gro g96 pdb etc.
`-g`	`genion.log`	Output	Log file
`-pot`	`pot.pdb`	Output, Opt.	Protein data bank file
`-p`	`topol.top`	In/Out, Opt.	Topology file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-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
`-random`	bool	`yes`	Use random placement of ions instead of based on potential. The rmin option should still work
`-seed`	int	`1993`	Seed for random number generator
`-scale`	real	`0.001`	Scaling factor for the potential for -pot
`-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 file. Overrides the -np and nn options.
`-neutral`	bool	`no`	This option will add enough ions to neutralize the system. In combination with the concentration option a neutral system at a given salt concentration will be generated.

- Calculation of the potential is not reliable, therefore the `-random` option is now turned on by default.

- 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.

D.31 genrestr

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. A single isotropic force constant may be given on the command line instead of three components.

WARNING: position restraints only work for the one molecule at a time. Position restraints are interactions within molecules, therefore they should be included within the correct `[ moleculetype ]` block in the topology. Since the atom numbers in every moleculetype in the topology start at 1 and the numbers in the input file for genpr number consecutively from 1, genpr will only produce a useful file for the first molecule.

The -of option produces an index file that can be used for freezing atoms. In this case the input file must be a pdb 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 C-alpha atoms in a protein, one can maintain the overall conformation of a protein without tieing it to a specific position (as with position restraints).

Files

`-f`	`conf.gro`	Input	Structure file: gro g96 pdb tpr etc.
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`posre.itp`	Output	Include file for topology
`-of`	`freeze.ndx`	Output, Opt.	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-fc`	vector	`1000 1000 1000`	force constants (kJ mol-1 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
`-disre`	bool	`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)
`-constr`	bool	`no`	Generate a constraint matrix rather than distance restraints

D.32 g_filter

g_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 which has been fitted on the coordinates in the structure file.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-ol`	`lowpass.xtc`	Output, Opt.	Trajectory: xtc trr trj gro g96 pdb
`-oh`	`highpass.xtc`	Output, Opt.	Trajectory: xtc trr trj gro g96 pdb

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-nf`	int	`10`	Sets the filter length as well as the output interval for low-pass filtering
`-all`	bool	`no`	Write all low-pass filtered frames
`-nojump`	bool	`yes`	Remove jumps of atoms across the box
`-fit`	bool	`no`	Fit all frames to a reference structure

D.33 g_gyrate

g_gyrate computes the radius of gyration of a group of atoms and the radii of gyration about the x, y and z axes, as a function of time. The atoms are explicitly mass weighted.

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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`gyrate.xvg`	Output	xvgr/xmgr file
`-acf`	`moi-acf.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-nmol`	int	`1`	The number of molecules to analyze

`-q`	bool	`no`	Use absolute value of the charge of an atom as weighting factor instead of mass
`-p`	bool	`no`	Calculate the radii of gyration about the principal axes.
`-moi`	bool	`no`	Calculate the moments of inertia (defined by the principal axes).
`-nz`	int	`0`	Calculate the 2D radii of gyration of # slices along the z-axis
`-acflen`	int	`-1`	Length of the ACF, default is half the number of frames
`-normalize`	bool	`yes`	Normalize ACF
`-P`	enum	`0`	Order of Legendre polynomial for ACF (0 indicates none): `0`, `1`, `2` or `3`
`-fitfn`	enum	`none`	Fit function: `none`, `exp`, `aexp`, `exp_exp`, `vac`, `exp5`, `exp7` or `exp9`
`-ncskip`	int	`0`	Skip N points in the output file of correlation functions
`-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

D.34 g_h2order

Compute the orientation of water molecules with respect to the normal of the box. The program determines the average cosine of the angle between de 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input	Index file
`-nm`	`index.ndx`	Input, Opt.	Index file
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-o`	`order.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-d`	string	`Z`	Take the normal on the membrane in direction X, Y or Z.
`-sl`	int	`0`	Calculate order parameter as function of boxlength, dividing the box in #nr slices.

- The program assigns whole water molecules to a slice, based on the firstatom 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.

D.35 g_hbond

g_hbond computes and analyzes hydrogen bonds. Hydrogen bonds are determined based on cutoffs for the angle Acceptor - Donor - Hydrogen (zero is extended) and the distance Hydrogen - Acceptor. 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.

```
[ selected ]
20 21 24
25 26 29
1 3 6
```

Note that the triplets need not be on separate lines. Each atom triplet specifies a hydrogen bond to be analyzed, note also that no check is made for the types of atoms.

`-ins` turns on computing solvent insertion into hydrogen bonds. In this case an additional group must be selected, specifying the solvent molecules.

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).

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-n`	`index.ndx`	Input, Opt.	Index file
`-num`	`hbnum.xvg`	Output	xvgr/xmgr file
`-g`	`hbond.log`	Output, Opt.	Log file
`-ac`	`hbac.xvg`	Output, Opt.	xvgr/xmgr file
`-dist`	`hbdist.xvg`	Output, Opt.	xvgr/xmgr file
`-ang`	`hbang.xvg`	Output, Opt.	xvgr/xmgr file
`-hx`	`hbhelix.xvg`	Output, Opt.	xvgr/xmgr file
`-hbn`	`hbond.ndx`	Output, Opt.	Index file
`-hbm`	`hbmap.xpm`	Output, Opt.	X PixMap compatible matrix file
`-don`	`donor.xvg`	Output, Opt.	xvgr/xmgr file
`-dan`	`danum.xvg`	Output, Opt.	xvgr/xmgr file
`-life`	`hblife.xvg`	Output, Opt.	xvgr/xmgr file

`-nhbdist`	`nhbdist.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-ins`	bool	`no`	Analyze solvent insertion
`-a`	real	`30`	Cutoff angle (degrees, Acceptor - Donor - Hydrogen)
`-r`	real	`0.35`	Cutoff radius (nm, X - Acceptor, see next option)
`-da`	bool	`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)
`-nitacc`	bool	`yes`	Regard nitrogen atoms as acceptors
`-contact`	bool	`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
`-fitstart`	real	`1`	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
`-smooth`	real	`-1`	If >= 0, the tail of the ACF will be smoothed by fitting it to an exponential function: y = A exp(-x/tau)
`-dump`	int	`0`	Dump the first N hydrogen bond ACFs in a single xvg 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
`-merge`	bool	`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.
`-geminate`	enum	`none`	Use reversible geminate recombination for the kinetics/thermodynamics calclations. See Markovitch et al., J. Chem. Phys 129, 084505 (2008) for details.: `none`, `dd`, `ad`, `aa` or `a4`
`-diff`	real	`-1`	Dffusion coefficient to use in the rev. gem. recomb. kinetic model. If non-positive, then it will be fitted to the ACF along with ka and kd.
`-acflen`	int	`-1`	Length of the ACF, default is half the number of frames
`-normalize`	bool	`yes`	Normalize ACF
`-P`	enum	`0`	Order of Legendre polynomial for ACF (0 indicates none): `0`, `1`, `2` or `3`
`-fitfn`	enum	`none`	Fit function: `none`, `exp`, `aexp`, `exp_exp`, `vac`, `exp5`, `exp7` or `exp9`
`-ncskip`	int	`0`	Skip N points in the output file of correlation functions
`-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

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

D.36 g_helix

g_helix computes all kind of helix properties. First, the peptide is checked to find the longest helical part. This is 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:

1. Helix radius (file radius.xvg). This is merely the RMS deviation in two dimensions for all Calpha atoms. it is calced 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
2. Twist (file twist.xvg). The average helical angle per residue is calculated. For alpha helix it is 100 degrees, for 3-10 helices it will be smaller, for 5-helices it will be larger.
3. Rise per residue (file rise.xvg). The helical rise per residue is plotted as the difference in Z-coordinate between Ca atoms. For an ideal helix this is 0.15 nm
4. 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).
5. Number of helical residues (file n-ahx.xvg). The title says it all.
6. Helix Dipole, backbone only (file dip-ahx.xvg).
7. RMS deviation from ideal helix, calculated for the Calpha atoms only (file rms-ahx.xvg).
8. Average Calpha-Calpha dihedral angle (file phi-ahx.xvg).
9. Average Phi and Psi angles (file phipsi.xvg).
10. Ellipticity at 222 nm according to *Hirst and Brooks*

Files

`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-n`	`index.ndx`	Input	Index file
`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-to`	`gtraj.g87`	Output, Opt.	Gromos-87 ASCII trajectory format
`-cz`	`zconf.gro`	Output	Structure file: gro g96 pdb etc.
`-co`	`waver.gro`	Output	Structure file: gro g96 pdb etc.

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-r0`	int	`1`	The first residue number in the sequence
`-q`	bool	`no`	Check at every step which part of the sequence is helical
`-F`	bool	`yes`	Toggle fit to a perfect helix
`-db`	bool	`no`	Print debug info
`-prop`	enum	`RAD`	Select property to weight eigenvectors with. WARNING experimental stuff: `RAD`, `TWIST`, `RISE`, `LEN`, `NHX`, `DIP`, `RMS`, `CPHI`, `RMSA`, `PHI`, `PSI`, `HB3`, `HB4`, `HB5` or `CD222`
`-ev`	bool	`no`	Write a new 'trajectory' file for ED
`-ahxstart`	int	`0`	First residue in helix
`-ahxend`	int	`0`	Last residue in helix

D.37 g_helixorient

g_helixorient calculates the coordinates and direction of the average axis inside an alpha helix, and the direction/vectors of both the alpha carbon and (optionally) a sidechain atom relative to the axis.

As input, you need to specify an index group with alpha carbon 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/CA-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 theta1-3.xvg

Files

`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input, Opt.	Index file
`-oaxis`	`helixaxis.dat`	Output	Generic data file
`-ocenter`	`center.dat`	Output	Generic data file
`-orise`	`rise.xvg`	Output	xvgr/xmgr file
`-oradius`	`radius.xvg`	Output	xvgr/xmgr file
`-otwist`	`twist.xvg`	Output	xvgr/xmgr file
`-obending`	`bending.xvg`	Output	xvgr/xmgr file
`-otilt`	`tilt.xvg`	Output	xvgr/xmgr file
`-orot`	`rotation.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-sidechain`	bool	`no`	Calculate sidechain directions relative to helix axis too.
`-incremental`	bool	`no`	Calculate incremental rather than total rotation/tilt.

D.38 g_highway

highway is the gromacs highway simulator. It is an X-windows gadget that shows a (periodic) Autobahn with a user defined number of cars. Fog can be turned on or off to increase the number of crashes. Nice for a background CPU-eater. A sample input file is in $GMXDATA/top/highway.dat

Files

`-f`	`highway.dat`	Input	Generic data file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel

D.39 g_lie

g_lie computes a free energy estimate based on an energy analysis from. One needs an energy file with the following components: Coul (A-B) LJ-SR (A-B) etc.

Files

`-f`	`ener.edr`	Input	Energy file
`-o`	`lie.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `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

D.40 g_mdmat

g_mdmat makes distance matrices consisting of the smallest distance between residue pairs. With -frames these distance matrices can be stored as a function of time, to be able to see differences in tertiary structure as a funcion of time. If you choose your options unwise, this may generate a large output file. 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 xpm2ps to make a PostScript (tm) plot.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-mean`	`dm.xpm`	Output	X PixMap compatible matrix file
`-frames`	`dmf.xpm`	Output, Opt.	X PixMap compatible matrix file
`-no`	`num.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-t`	real	`1.5`	trunc distance
`-nlevels`	int	`40`	Discretize distance in # levels

D.41 g_membed

g_membed embeds a membrane protein into an equilibrated lipid bilayer at the position and orientation specified by the user.

SHORT MANUAL ————

The user should merge the structure files of the protein and membrane (+solvent), creating a single structure file with the protein overlapping the membrane at the desired position and orientation. Box size should be taken from the membrane structure file. The corresponding topology files should also be merged. Consecutively, create a tpr file (input for g_membed) from these files,with the following options included in the mdp file.

- integrator = md

- energygrp = Protein (or other group that you want to insert)

- freezegrps = Protein

- freezedim = Y Y Y

- energygrp_excl = Protein Protein

The output is a structure file containing the protein embedded in the membrane. If a topology file is provided, the number of lipid and solvent molecules will be updated to match the new structure file.

For a more extensive manual see Wolf et al, J Comp Chem 31 (2010) 2169-2174, Appendix.

SHORT METHOD DESCRIPTION

---

1. The protein is resized around its center of mass by a factor -xy in the xy-plane (the membrane plane) and a factor -z in the z-direction (if the size of the protein in the z-direction is the same or smaller than the width of the membrane, a -z value larger than 1 can prevent that the protein will be enveloped by the lipids).

2. All lipid and solvent molecules overlapping with the resized protein are removed. All intraprotein interactions are turned off to prevent numerical issues for small values of -xy or -z

3. One md step is performed.

4. The resize factor (-xy or -z) is incremented by a small amount ((1-xy)/nxy or (1-z)/nz) and the protein is resized again around its center of mass. The resize factor for the xy-plane is incremented first. The resize factor for the z-direction is not changed until the -xy factor is 1 (thus after -nxy iteration).

5. Repeat step 3 and 4 until the protein reaches its original size (-nxy + -nz iterations).

For a more extensive method descrition see Wolf et al, J Comp Chem, 31 (2010) 2169-2174.

NOTE —-

- Protein can be any molecule you want to insert in the membrane.

- It is recommended to perform a short equilibration run after the embedding (see Wolf et al, J Comp Chem 31 (2010) 2169-2174, to re-equilibrate the membrane. Clearly protein equilibration might require longer.

Files

`-f`	`into_mem.tpr`	Input	Run input file: tpr tpb tpa
`-n`	`index.ndx`	Input, Opt.	Index file
`-p`	`topol.top`	In/Out, Opt.	Topology file
`-o`	`traj.trr`	Output	Full precision trajectory: trr trj cpt
`-x`	`traj.xtc`	Output, Opt.	Compressed trajectory (portable xdr format)
`-cpi`	`state.cpt`	Input, Opt.	Checkpoint file
`-cpo`	`state.cpt`	Output, Opt.	Checkpoint file

`-c`	`membedded.gro`	Output	Structure file: gro g96 pdb etc.
`-e`	`ener.edr`	Output	Energy file
`-g`	`md.log`	Output	Log file
`-ei`	`sam.edi`	Input, Opt.	ED sampling input
`-rerun`	`rerun.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-table`	`table.xvg`	Input, Opt.	xvgr/xmgr file
`-tablep`	`tablep.xvg`	Input, Opt.	xvgr/xmgr file
`-tableb`	`table.xvg`	Input, Opt.	xvgr/xmgr file
`-dhdl`	`dhdl.xvg`	Output, Opt.	xvgr/xmgr file
`-field`	`field.xvg`	Output, Opt.	xvgr/xmgr file
`-table`	`table.xvg`	Input, Opt.	xvgr/xmgr file
`-tablep`	`tablep.xvg`	Input, Opt.	xvgr/xmgr file
`-tableb`	`table.xvg`	Input, Opt.	xvgr/xmgr file
`-rerun`	`rerun.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-tpi`	`tpi.xvg`	Output, Opt.	xvgr/xmgr file
`-tpid`	`tpidist.xvg`	Output, Opt.	xvgr/xmgr file
`-ei`	`sam.edi`	Input, Opt.	ED sampling input
`-eo`	`sam.edo`	Output, Opt.	ED sampling output
`-j`	`wham.gct`	Input, Opt.	General coupling stuff
`-jo`	`bam.gct`	Output, Opt.	General coupling stuff
`-ffout`	`gct.xvg`	Output, Opt.	xvgr/xmgr file
`-devout`	`deviatie.xvg`	Output, Opt.	xvgr/xmgr file
`-runav`	`runaver.xvg`	Output, Opt.	xvgr/xmgr file
`-px`	`pullx.xvg`	Output, Opt.	xvgr/xmgr file
`-pf`	`pullf.xvg`	Output, Opt.	xvgr/xmgr file
`-mtx`	`nm.mtx`	Output, Opt.	Hessian matrix
`-dn`	`dipole.ndx`	Output, Opt.	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-deffnm`	string		Set the default filename for all file options
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-xyinit`	real	`0.5`	Resize factor for the protein in the xy dimension before starting embedding
`-xyend`	real	`1`	Final resize factor in the xy dimension
`-zinit`	real	`1`	Resize factor for the protein in the z dimension before starting embedding
`-zend`	real	`1`	Final resize faction in the z dimension
`-nxy`	int	`1000`	Number of iteration for the xy dimension
`-nz`	int	`0`	Number of iterations for the z dimension
`-rad`	real	`0.22`	Probe radius to check for overlap between the group to embed and the membrane
`-pieces`	int	`1`	Perform piecewise resize. Select parts of the group to insert and resize these with respect to their own geometrical center.
`-asymmetry`	bool	`no`	Allow asymmetric insertion, i.e. the number of lipids removed from the upper and lower leaflet will not be checked.
`-ndiff`	int	`0`	Number of lipids that will additionally be removed from the lower (negative number) or upper (positive number) membrane leaflet.
`-maxwarn`	int	`0`	Maximum number of warning allowed
`-compact`	bool	`yes`	Write a compact log file
`-v`	bool	`no`	Be loud and noisy

D.42 g_mindist

g_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 an other 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. It also plots the maximum distance within the group and the lengths of the three box vectors.

Other programs that calculate distances are `g_dist` and `g_bond`.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-od`	`mindist.xvg`	Output	xvgr/xmgr file
`-on`	`numcont.xvg`	Output, Opt.	xvgr/xmgr file
`-o`	`atm-pair.out`	Output, Opt.	Generic output file
`-ox`	`mindist.xtc`	Output, Opt.	Trajectory: xtc trr trj gro g96 pdb
`-or`	`mindistres.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-matrix`	bool	`no`	Calculate half a matrix of group-group distances
`-max`	bool	`no`	Calculate *maximum* distance instead of minimum
`-d`	real	`0.6`	Distance for contacts
`-group`	bool	`no`	Count contacts with multiple atoms in the first group as one
`-pi`	bool	`no`	Calculate minimum distance with periodic images
`-split`	bool	`no`	Split graph where time is zero
`-ng`	int	`1`	Number of secondary groups to compute distance to a central group
`-pbc`	bool	`yes`	Take periodic boundary conditions into account
`-respertime`	bool	`no`	When writing per-residue distances, write distance for each time point
`-printresname`	bool	`no`	Write residue names

D.43 g_morph

g_morph does a linear interpolation of conformations in order to create intermediates. Of course these are completely unphysical, but that you may try to justify yourself. Output is in the form of a generic trajectory. The number of intermediates can be controlled with the -ninterm flag. The first and last flag correspond to the way of interpolating: 0 corresponds to input structure 1 while 1 corresponds to input structure 2. If you specify first < 0 or last > 1 extrapolation will be on the path from input structure x1 to x2. In general the coordinates of the intermediate x(i) out of N total intermidates correspond to:

x(i) = x1 + (first+(i/(N-1))*(last-first))*(x2-x1)

Finally the RMSD with respect to both input structures can be computed if explicitly selected (-or option). In that case an index file may be read to select what group RMS is computed from.

Files

`-f1`	`conf1.gro`	Input	Structure file: gro g96 pdb tpr etc.
`-f2`	`conf2.gro`	Input	Structure file: gro g96 pdb tpr etc.
`-o`	`interm.xtc`	Output	Trajectory: xtc trr trj gro g96 pdb cpt
`-or`	`rms-interm.xvg`	Output, Opt.	xvgr/xmgr file
`-n`	`index.ndx`	Input, Opt.	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-ninterm`	int	`11`	Number of intermediates
`-first`	real	`0`	Corresponds to first generated structure (0 is input x0, see above)
`-last`	real	`1`	Corresponds to last generated structure (1 is input x1, see above)
`-fit`	bool	`yes`	Do a least squares fit of the second to the first structure before interpolating

D.44 g_msd

g_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, g_msd plots the MSD for individual molecules: for each individual molecule a diffusion constant is computed for its center of mass. The chosen index group will be split into molecules.

The default way to calculate a MSD is by using mass-weighted averages. This can be turned off with `-nomw`.

With the option `-rmcomm`, the center of mass motion of a specific group can be removed. For trajectories produced with GROMACS this is usually not necessary, as mdrun usually already removes the center of mass motion. When you use this option be sure that the whole system is stored in the trajectory file.

The diffusion coefficient is determined by linear regression of the MSD, where, unlike for the normal output of D, the times are weighted according to the number of reference points, i.e. short times have a higher weight. Also when `-beginfit`=-1,fitting starts at 10% and when `-endfit`=-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`.

Option `-pdb` writes a pdb file with the coordinates of the frame at time `-tpdb` with in the B-factor field the square root of the diffusion coefficient of the molecule. This option implies option `-mol`.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`msd.xvg`	Output	xvgr/xmgr file
`-mol`	`diff_mol.xvg`	Output, Opt.	xvgr/xmgr file
`-pdb`	`diff_mol.pdb`	Output, Opt.	Protein data bank file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-b`	time	`0`	First frame (ps) to read from trajectory
`-e`	time	`0`	Last frame (ps) to read from trajectory
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-type`	enum	`no`	Compute diffusion coefficient in one direction: `no`, `x`, `y` or `z`
`-lateral`	enum	`no`	Calculate the lateral diffusion in a plane perpendicular to: `no`, `x`, `y` or `z`
`-ten`	bool	`no`	Calculate the full tensor
`-ngroup`	int	`1`	Number of groups to calculate MSD for
`-mw`	bool	`yes`	Mass weighted MSD
`-rmcomm`	bool	`no`	Remove center of mass motion
`-tpdb`	time	`0`	The frame to use for option -pdb (ps)
`-trestart`	time	`10`	Time between restarting points in trajectory (ps)
`-beginfit`	time	`-1`	Start time for fitting the MSD (ps), -1 is 10%
`-endfit`	time	`-1`	End time for fitting the MSD (ps), -1 is 90%

D.45 gmxcheck

gmxcheck reads a trajectory (`.trj`, `.trr` or `.xtc`), an energy file (`.ene` or `.edr`) or an index file (`.ndx`) 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 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, gmxcheck provides a quick check for such problems.

The program can compare two run input (`.tpr`, `.tpb` or `.tpa`) files when both `-s1` and `-s2` are supplied. 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`.

In case the `-m` flag is given a LaTeX file will be written consisting of a rough outline for a methods section for a paper.

Files

`-f`	`traj.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-f2`	`traj.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-s1`	`top1.tpr`	Input, Opt.	Run input file: tpr tpb tpa
`-s2`	`top2.tpr`	Input, Opt.	Run input file: tpr tpb tpa
`-c`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-e`	`ener.edr`	Input, Opt.	Energy file
`-e2`	`ener2.edr`	Input, Opt.	Energy file
`-n`	`index.ndx`	Input, Opt.	Index file
`-m`	`doc.tex`	Output, Opt.	LaTeX file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-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
`-rmsd`	bool	`no`	Print RMSD for x, v and f
`-tol`	real	`0.001`	Relative tolerance for comparing real values defined as 2*(a-b)/(oraor+orbor)
`-abstol`	real	`0.001`	Absolute tolerance, useful when sums are close to zero.
`-ab`	bool	`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.

D.46 gmxdump

gmxdump reads a run input file (`.tpa`/`.tpr`/`.tpb`), a trajectory (`.trj`/`.trr`/`.xtc`), an energy file (`.ene`/`.edr`), or a checkpoint file (`.cpt`) and prints that to standard output in a readable format. This program is essential for checking your run input file in case of problems.

The program can also preprocess a topology to help finding problems. Note that currently setting GMXLIB is the only way to customize directories used for searching include files.

Files

`-s`	`topol.tpr`	Input, Opt.	Run input file: tpr tpb tpa
`-f`	`traj.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-e`	`ener.edr`	Input, Opt.	Energy file
`-cp`	`state.cpt`	Input, Opt.	Checkpoint file
`-p`	`topol.top`	Input, Opt.	Topology file
`-mtx`	`hessian.mtx`	Input, Opt.	Hessian matrix
`-om`	`grompp.mdp`	Output, Opt.	grompp input file with MD parameters

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit

`-nice`	int	`0`	Set the nicelevel
`-nr`	bool	`yes`	Show index numbers in output (leaving them out makes comparison easier, but creates a useless topology)
`-sys`	bool	`no`	List the atoms and bonded interactions for the whole system instead of for each molecule type

D.47 g_nmeig

g_nmeig calculates the eigenvectors/values of a (Hessian) matrix, which can be calculated with `mdrun`. 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 as timestamp. The eigenvectors can be analyzed with `g_anaeig`. An ensemble of structures can be generated from the eigenvectors with `g_nmens`. 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).

Files

`-f`	`hessian.mtx`	Input	Hessian matrix
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-of`	`eigenfreq.xvg`	Output	xvgr/xmgr file
`-ol`	`eigenval.xvg`	Output	xvgr/xmgr file
`-v`	`eigenvec.trr`	Output	Full precision trajectory: trr trj cpt

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-m`	bool	`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

D.48 g_nmens

`g_nmens` generates an ensemble around an average structure in a subspace which 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.

Files

`-v`	`eigenvec.trr`	Input	Full precision trajectory: trr trj cpt
`-e`	`eigenval.xvg`	Input	xvgr/xmgr file
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`ensemble.xtc`	Output	Trajectory: xtc trr trj gro g96 pdb

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-temp`	real	`300`	Temperature in Kelvin
`-seed`	int	`-1`	Random seed, -1 generates a seed from time and pid
`-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)

D.49 g_nmtraj

`g_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 unrealistic 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.

Files

`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-v`	`eigenvec.trr`	Input	Full precision trajectory: trr trj cpt
`-o`	`nmtraj.xtc`	Output	Trajectory: xtc trr trj gro g96 pdb

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 in Kelvin
`-amplitude`	real	`0.25`	Amplitude for modes with eigenvalue<=0
`-nframes`	int	`30`	Number of frames to generate

D.50 g_order

Compute 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).

The program 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input	Index file
`-nr`	`index.ndx`	Input	Index file
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-o`	`order.xvg`	Output	xvgr/xmgr file
`-od`	`deuter.xvg`	Output	xvgr/xmgr file
`-ob`	`eiwit.pdb`	Output	Protein data bank file
`-os`	`sliced.xvg`	Output	xvgr/xmgr file
`-Sg`	`sg-ang.xvg`	Output, Opt.	xvgr/xmgr file
`-Sk`	`sk-dist.xvg`	Output, Opt.	xvgr/xmgr file
`-Sgsl`	`sg-ang-slice.xvg`	Output, Opt.	xvgr/xmgr file
`-Sksl`	`sk-dist-slice.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-d`	enum	`z`	Direction of the normal on the membrane: `z`, `x` or `y`
`-sl`	int	`1`	Calculate order parameter as function of boxlength, dividing the box in #nr slices.
`-szonly`	bool	`no`	Only give Sz element of order tensor. (axis can be specified with -d)
`-unsat`	bool	`no`	Calculate order parameters for unsaturated carbons. Note that this cannot be mixed with normal order parameters.
`-permolecule`	bool	`no`	Compute per-molecule Scd order parameters
`-radial`	bool	`no`	Compute a radial membrane normal
`-calcdist`	bool	`no`	Compute distance from a reference (currently defined only for radial and permolecule)

D.51 g_polystat

g_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>).

Files

`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`polystat.xvg`	Output	xvgr/xmgr file
`-v`	`polyvec.xvg`	Output, Opt.	xvgr/xmgr file
`-p`	`persist.xvg`	Output, Opt.	xvgr/xmgr file
`-i`	`intdist.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-mw`	bool	`yes`	Use the mass weighting for radii of gyration
`-pc`	bool	`no`	Plot average eigenvalues

D.52 g_potential

Compute 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's 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, 2 is more appropriate in many cases.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input	Index file
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-o`	`potential.xvg`	Output	xvgr/xmgr file
`-oc`	`charge.xvg`	Output	xvgr/xmgr file
`-of`	`field.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)

`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `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 #nr slices.
`-cb`	int	`0`	Discard first #nr slices of box for integration
`-ce`	int	`0`	Discard last #nr slices of box for integration
`-tz`	real	`0`	Translate all coordinates <distance> in the direction of the box
`-spherical`	bool	`no`	Calculate spherical thingie
`-ng`	int	`1`	Number of groups to consider
`-correct`	bool	`no`	Assume net zero charge of groups to improve accuracy

- Discarding slices for integration should not be necessary.

D.53 g_principal

g_principal calculates the three principal axes of inertia for a group of atoms.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-a1`	`axis1.dat`	Output	Generic data file
`-a2`	`axis2.dat`	Output	Generic data file
`-a3`	`axis3.dat`	Output	Generic data file
`-om`	`moi.dat`	Output	Generic data file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-foo`	bool	`no`	Dummy option to avoid empty array

D.54 g_protonate

`protonate` reads (a) conformation(s) and adds all missing hydrogens as defined in `ffgmx2.hdb`. If only `-s` is specified, this conformation will be protonated, if also `-f` is specified, the conformation(s) will be read from this file which can be either a single conformation or a trajectory.

If a pdb file is supplied, residue names might not correspond to to the GROMACS naming conventions, in which case these residues will probably not be properly protonated.

If an index file is specified, please note that the atom numbers should correspond to the **protonated** state.

Files

`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb

`-f`	`traj.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`protonated.xtc`	Output	Trajectory: xtc trr trj gro g96 pdb

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-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 when t MOD dt = first time (ps)

D.55 g_rama

g_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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-o`	`rama.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`

D.56 g_rdf

The structure of liquids can be studied by either neutron or X-ray scattering. The most common way to describe liquid structure is by a radial distribution function. However, this is not easy to obtain from a scattering experiment.

g_rdf calculates radial distribution functions in different ways. The normal method is around a (set of) particle(s), the other methods are around the center of mass of a set of particles (`-com`) or to the closest particle in a set (`-surf`). With all methods rdf's can also be calculated around axes parallel to the z-axis with option `-xy`. With option `-surf` normalization can not be used.

The option `-rdf` sets the type of rdf to be computed. Default is for atoms or particles, but one can also select center of mass or geometry of molecules or residues. In all cases only the atoms in the index groups are taken into account. For molecules and/or the center of mass option a run input file is required. Other weighting than COM or COG can currently only be achieved by providing a run input file with different masses. Options `-com` and `-surf` also work in conjunction with `-rdf`.

If a run input file is supplied (`-s`) and `-rdf` is set to `atom`, exclusions defined in that file are taken into account when calculating the rdf. The option `-cut` is meant as an alternative way to avoid intramolecular peaks in the rdf plot. It is however better to supply a run input file with a higher number of exclusions. For eg. benzene a topology with nrexcl set to 5 would eliminate all intramolecular contributions to the rdf. Note that all atoms in the selected groups are used, also the ones that don't have Lennard-Jones interactions.

Option `-cn` produces the cumulative number rdf, i.e. the average number of particles within a distance r.

To bridge the gap between theory and experiment structure factors can be computed (option `-sq`). The algorithm uses FFT, the grid spacing of which is determined by option `-grid`.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-d`	`sfactor.dat`	Input, Opt.	Generic data file
`-o`	`rdf.xvg`	Output, Opt.	xvgr/xmgr file
`-sq`	`sq.xvg`	Output, Opt.	xvgr/xmgr file
`-cn`	`rdf_cn.xvg`	Output, Opt.	xvgr/xmgr file
`-hq`	`hq.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-bin`	real	`0.002`	Binwidth (nm)
`-com`	bool	`no`	RDF with respect to the center of mass of first group
`-surf`	enum	`no`	RDF with respect to the surface of the first group: `no`, `mol` or `res`
`-rdf`	enum	`atom`	RDF type: `atom`, `mol_com`, `mol_cog`, `res_com` or `res_cog`
`-pbc`	bool	`yes`	Use periodic boundary conditions for computing distances. Without PBC the maximum range will be three times the largest box edge.
`-norm`	bool	`yes`	Normalize for volume and density
`-xy`	bool	`no`	Use only the x and y components of the distance
`-cut`	real	`0`	Shortest distance (nm) to be considered
`-ng`	int	`1`	Number of secondary groups to compute RDFs around a central group
`-fade`	real	`0`	From this distance onwards the RDF is tranformed by g'(r) = 1 + [g(r)-1] exp(-(r/fade-1)^2 to make it go to 1 smoothly. If fade is 0.0 nothing is done.
`-nlevel`	int	`20`	Number of different colors in the diffraction image
`-startq`	real	`0`	Starting q (1/nm)
`-endq`	real	`60`	Ending q (1/nm)
`-energy`	real	`12`	Energy of the incoming X-ray (keV)

D.57 g_rms

g_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` 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 `xpm2ps`.

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 file masses will be taken from there, otherwise the masses will be deduced from the atommass.dat file in the GROMACS library directory. 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 happend 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.

Files

`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-f2`	`traj.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`rmsd.xvg`	Output	xvgr/xmgr file
`-mir`	`rmsdmir.xvg`	Output, Opt.	xvgr/xmgr file
`-a`	`avgrp.xvg`	Output, Opt.	xvgr/xmgr file
`-dist`	`rmsd-dist.xvg`	Output, Opt.	xvgr/xmgr file
`-m`	`rmsd.xpm`	Output, Opt.	X PixMap compatible matrix file
`-bin`	`rmsd.dat`	Output, Opt.	Generic data file
`-bm`	`bond.xpm`	Output, Opt.	X PixMap compatible matrix file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-what`	enum	`rmsd`	Structural difference measure: `rmsd`, `rho` or `rhosc`
`-pbc`	bool	`yes`	PBC check
`-fit`	enum	`rot+trans`	Fit to reference structure: `rot+trans`, `translation` or `none`
`-prev`	int	`0`	Compare with previous frame
`-split`	bool	`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
`-mw`	bool	`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

D.58 g_rmsdist

g_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 g_rms. 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.

g_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.:

```
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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-equiv`	`equiv.dat`	Input, Opt.	Generic data file
`-o`	`distrmsd.xvg`	Output	xvgr/xmgr file
`-rms`	`rmsdist.xpm`	Output, Opt.	X PixMap compatible matrix file
`-scl`	`rmsscale.xpm`	Output, Opt.	X PixMap compatible matrix file
`-mean`	`rmsmean.xpm`	Output, Opt.	X PixMap compatible matrix file
`-nmr3`	`nmr3.xpm`	Output, Opt.	X PixMap compatible matrix file
`-nmr6`	`nmr6.xpm`	Output, Opt.	X PixMap compatible matrix file
`-noe`	`noe.dat`	Output, Opt.	Generic data file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-nlevels`	int	`40`	Discretize rms in # levels
`-max`	real	`-1`	Maximum level in matrices
`-sumh`	bool	`yes`	average distance over equivalent hydrogens
`-pbc`	bool	`yes`	Use periodic boundary conditions when computing distances

D.59 g_rmsf

g_rmsf computes the root mean square fluctuation (RMSF, i.e. standard deviation) of atomic positions after (optionally) fitting to a reference frame.

With option `-oq` the RMSF values are converted to B-factor values, which are written to a pdb file with the coordinates, of the structure file, or of a pdb file when `-q` is specified. Option `-ox` writes the B-factors to a file with the average coordinates.

With the option `-od` the root mean square deviation with respect to the reference structure is calculated.

With the option `aniso` g_rmsf will compute anisotropic temperature factors and then it will also output average coordinates and a pdb file with ANISOU records (corresonding 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 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 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-q`	`eiwit.pdb`	Input, Opt.	Protein data bank file
`-oq`	`bfac.pdb`	Output, Opt.	Protein data bank file
`-ox`	`xaver.pdb`	Output, Opt.	Protein data bank file
`-o`	`rmsf.xvg`	Output	xvgr/xmgr file
`-od`	`rmsdev.xvg`	Output, Opt.	xvgr/xmgr file
`-oc`	`correl.xvg`	Output, Opt.	xvgr/xmgr file
`-dir`	`rmsf.log`	Output, Opt.	Log file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-res`	bool	`no`	Calculate averages for each residue
`-aniso`	bool	`no`	Compute anisotropic termperature factors
`-fit`	bool	`yes`	Do a least squares superposition before computing RMSF. Without this you must make sure that the reference structure and the trajectory match.

D.60 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. grompp also reads parameters for the mdrun (eg. number of MD steps, time step, cut-off), and others such as NEMD parameters, which are corrected so that the net acceleration is zero. Eventually a binary file is produced that can serve as the sole input file for the MD program.

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.

grompp uses a built-in preprocessor to resolve includes, macros etcetera. 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` file:
define = -DVARIABLE1 -DVARIABLE2
include = /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.

If your system does not have a c-preprocessor, you can still use grompp, but you do not have access to the features from the cpp. Command line options to the c-preprocessor can be given in the `.mdp` file. See your local manual (man cpp).

When using position restraints a file with restraint coordinates can be supplied with `-r`, otherwise restraining will be done with respect to the conformation from the `-c` option. For free energy calculation the the 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. Note that these velocities will not be used when `gen_vel = yes` in your `.mdp` file. An energy file can be supplied with `-e` to read Nose-Hoover and/or Parrinello-Rahman coupling variables. Note that for continuation it is better and easier to supply a checkpoint file directly to mdrun, since that always contains the complete state of the system and you don't need to generate a new run input file. Note that if you only want to change the number of run steps tpbconv is more convenient than grompp.

Using the `-morse` option grompp can convert the harmonic bonds in your topology to morse potentials. This makes it possible to break bonds. For this option to work you need an extra file in your $GMXLIB with dissociation energy. Use the -debug option to get more information on the workings of this option (look for MORSE in the grompp.log file using less or something like that).

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 make notice 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 `grompp` has read. If in doubt you can start grompp with the `-debug` option which will give you more information in a file called grompp.log (along with real debug info). Finally, you can see the contents of the run input file with the `gmxdump` program.

Files

`-f`	`grompp.mdp`	Input, Opt.	grompp input file with MD parameters
`-po`	`mdout.mdp`	Output	grompp input file with MD parameters
`-c`	`conf.gro`	Input	Structure file: gro g96 pdb tpr etc.
`-r`	`conf.gro`	Input, Opt.	Structure file: gro g96 pdb tpr etc.
`-rb`	`conf.gro`	Input, Opt.	Structure file: gro g96 pdb tpr etc.
`-n`	`index.ndx`	Input, Opt.	Index file
`-p`	`topol.top`	Input	Topology file
`-pp`	`processed.top`	Output, Opt.	Topology file
`-o`	`topol.tpr`	Output	Run input file: tpr tpb tpa
`-t`	`traj.trr`	Input, Opt.	Full precision trajectory: trr trj cpt
`-e`	`ener.edr`	Input, Opt.	Energy file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-v`	bool	`no`	Be loud and noisy
`-time`	real	`-1`	Take frame at or first after this time.
`-rmvsbds`	bool	`yes`	Remove constant bonded interactions with virtual sites
`-maxwarn`	int	`0`	Number of allowed warnings during input processing
`-zero`	bool	`no`	Set parameters for bonded interactions without defaults to zero instead of generating an error
`-renum`	bool	`yes`	Renumber atomtypes and minimize number of atomtypes

D.61 g_rotacf

g_rotacf calculates the rotational correlation function for molecules. Three atoms (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, controlled by the -d switch, you can calculate the rotational correlation function for linear molecules by specifying two atoms (i,j) in the index file.

EXAMPLES

g_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 till 20.0 ps to a two parameter exponential.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-n`	`index.ndx`	Input	Index file
`-o`	`rotacf.xvg`	Output	xvgr/xmgr file

Other options

-h	bool	no	Print help info and quit
-version	bool	no	Print version info and quit
-nice	int	19	Set the nicelevel
-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 when t MOD dt = first time (ps)
-w	bool	no	View output xvg, xpm, eps and pdb files
-xvg	enum	xmgrace	xvg plot formatting: `xmgrace`, `xmgr` or `none`
-d	bool	no	Use index doublets (vectors) for correlation function instead of triplets (planes)
-aver	bool	yes	Average over molecules
-acflen	int	-1	Length of the ACF, default is half the number of frames
-normalize	bool	yes	Normalize ACF
-P	enum	0	Order of Legendre polynomial for ACF (0 indicates none): `0`, `1`, `2` or `3`
-fitfn	enum	none	Fit function: `none`, `exp`, `aexp`, `exp_exp`, `vac`, `exp5`, `exp7` or `exp9`
-ncskip	int	0	Skip N points in the output file of correlation functions
-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

D.62 g_rotmat

g_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 `trjconv -fit rotxy+transxy` to remove the rotation in the xy-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.

Files

-f	traj.xtc	Input	Trajectory: xtc trr trj gro g96 pdb cpt
-s	topol.tpr	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
-n	index.ndx	Input, Opt.	Index file
-o	rotmat.xvg	Output	xvgr/xmgr file

Other options

-h	bool	no	Print help info and quit
-version	bool	no	Print version info and quit
-nice	int	19	Set the nicelevel
-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 when t MOD dt = first time (ps)
-w	bool	no	View output xvg, xpm, eps and pdb files
-xvg	enum	xmgrace	xvg plot formatting: `xmgrace`, `xmgr` or `none`

`-ref`	enum	`none`	Determine the optimal reference structure: `none`, `xyz` or `xy`
`-skip`	int	`1`	Use every nr-th frame for -ref
`-fitxy`	bool	`no`	Fit the x/y rotation before determining the rotation
`-mw`	bool	`yes`	Use mass weighted fitting

D.63 g_saltbr

g_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, (eg. the cut-off), then 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-ResnameResnr-Atomnr`. There may be many such files.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-t`	real	`1000`	trunc distance
`-sep`	bool	`no`	Use separate files for each interaction (may be MANY)

D.64 g_sas

g_sas computes hydrophobic, hydrophilic and total solvent accessible surface area. As a side effect the Connolly surface can be generated as well in a pdb file where the nodes are represented as atoms and the vertices connecting the nearest nodes as CONECT records. The program will ask for a group for the surface calculation and a group for the output. The calculation group should always consists of all the non-solvent atoms in the system. The output group can be the whole or part of the calculation group. The area can be plotted per residue and atom as well (options `-or` and `-oa`). In combination with the latter option an `itp` file can be generated (option `-i`) which can be used to restrain surface atoms.

By default, periodic boundary conditions are taken into account, this can be turned off using the `-nopbc` option.

With the `-tv` option the total volume and density of the molecule can be computed. Please 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, in e.g. ice Ih one can easily fit water molecules in the pores which would yield too low volume, too high surface area and too high density.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-o`	`area.xvg`	Output	xvgr/xmgr file
`-or`	`resarea.xvg`	Output, Opt.	xvgr/xmgr file

`-oa`	`atomarea.xvg`	Output, Opt.	xvgr/xmgr file
`-tv`	`volume.xvg`	Output, Opt.	xvgr/xmgr file
`-q`	`connelly.pdb`	Output, Opt.	Protein data bank file
`-n`	`index.ndx`	Input, Opt.	Index file
`-i`	`surfat.itp`	Output, Opt.	Include file for topology

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-probe`	real	`0.14`	Radius of the solvent probe (nm)
`-ndots`	int	`24`	Number of dots per sphere, more dots means more accuracy
`-qmax`	real	`0.2`	The maximum charge (e, absolute value) of a hydrophobic atom
`-f_index`	bool	`no`	Determine from a group in the index file what are the hydrophobic atoms rather than from the charge
`-minarea`	real	`0.5`	The minimum area (nm^2) to count an atom as a surface atom when writing a position restraint file (see help)
`-pbc`	bool	`yes`	Take periodicity into account
`-prot`	bool	`yes`	Output the protein to the connelly pdb file too
`-dgs`	real	`0`	default value for solvation free energy per area ($kJ/mol/nm^2$)

D.65 g_sdf

g_sdf calculates the spatial distribution function (SDF) of a set of atoms within a coordinate system defined by three atoms. There is single body, two body and three body SDF implemented (select with option -mode). In the single body case the local coordinate system is defined by using a triple of atoms from one single molecule, for the two and three body case the configurations are dynamically searched complexes of two or three molecules (or residues) meeting certain distance consitions (see below).

The program needs a trajectory, a GROMACS run input file and an index file to work. You have to setup 4 groups in the index file before using g_sdf:

The first three groups are used to define the SDF coordinate system. The program will dynamically generate the atom triples according to the selected -mode: In -mode 1 the triples will be just the 1st, 2nd, 3rd, ... atoms from groups 1, 2 and 3. Hence the nth entries in groups 1, 2 and 3 must be from the same residue. In -mode 2 the triples will be 1st, 2nd, 3rd, ... atoms from groups 1 and 2 (with the nth entries in groups 1 and 2 having the same res-id). For each pair from groups 1 and 2 group 3 is searched for an atom meeting the distance conditions set with -triangle and -dtri relative to atoms 1 and 2. In -mode 3 for each atom in group 1 group 2 is searched for an atom meeting the distance condition and if a pair is found group 3 is searched for an atom meeting the further conditions. The triple will only be used if all three atoms have different res-id's.

The local coordinate system is always defined using the following scheme: Atom 1 will be used as the point of origin for the SDF. Atom 1 and 2 will define the principle axis (Z) of the coordinate system. The other two axis will be defined inplane (Y) and normal (X) to the plane through Atoms 1, 2 and 3. The fourth group contains the atoms for which the SDF will be evaluated.

For -mode 2 and 3 you have to define the distance conditions for the 2 resp. 3 molecule complexes to be searched for using -triangle and -dtri.

The SDF will be sampled in cartesian coordinates. Use '-grid x y z' to define the size of the SDF grid around the reference molecule. The Volume of the SDF grid will be V=x*y*z (nm^3). Use -bin to set the binwidth for grid.

The output will be a binary 3D-grid file (gom_plt.dat) in the .plt format that can be be read directly by gOpenMol. The option -r will generate a .gro file with the reference molecule(s) transferred to the SDF coordinate system. Load this file into gOpenMol and display the SDF as a contour plot (see http://www.csc.fi/gopenmol/index.phtm for further documentation).

For further information about SDF's have a look at: A. Vishnyakov, JPC A, 105, 2001, 1702 and the references cited within.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input	Index file
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-o`	`gom_plt.dat`	Output	Generic data file
`-r`	`refmol.gro`	Output, Opt.	Structure file: gro g96 pdb etc.

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-mode`	int	`1`	SDF in [1,2,3] particle mode
`-triangle`	vector	`0 0 0`	r(1,3), r(2,3), r(1,2)
`-dtri`	vector	`0.03 0.03 0.03`	dr(1,3), dr(2,3), dr(1,2)
`-bin`	real	`0.05`	Binwidth for the 3D-grid (nm)
`-grid`	vector	`1 1 1`	Size of the 3D-grid (nm,nm,nm)

D.66 g_select

g_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. Any combination of the output options is possible, but note that `-om` only operates on the first selection.

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 `-dump`, the frame time and the number of positions is omitted from the output. In this case, only one selection can be given.

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 `-dump`, the frame time is omitted from the output.

Files

`-f`	`traj.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-sf`	`selection.dat`	Input, Opt.	Generic data file
`-n`	`index.ndx`	Input, Opt.	Index file
`-os`	`size.xvg`	Output, Opt.	xvgr/xmgr file
`-oc`	`cfrac.xvg`	Output, Opt.	xvgr/xmgr file
`-oi`	`index.dat`	Output, Opt.	Generic data file
`-om`	`mask.dat`	Output, Opt.	Generic data file
`-on`	`index.ndx`	Output, Opt.	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-rmpbc`	bool	`yes`	Make molecules whole for each frame
`-pbc`	bool	`yes`	Use periodic boundary conditions for distance calculation
`-select`	string		Selection string (use 'help' for help)
`-selrpos`	enum	`atom`	Selection reference position: `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` or `dyn_mol_cog`
`-seltype`	enum	`atom`	Default analysis 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` or `dyn_mol_cog`
`-dump`	bool	`no`	Do not print the frame time (-om, -oi) or the index size (-oi)
`-norm`	bool	`no`	Normalize by total number of positions with -os
`-cfnorm`	bool	`no`	Normalize by covered fraction with -os
`-resnr`	enum	`number`	Residue number output type: `number` or `index`

D.67 g_sgangle

Compute the angle and distance between two groups. The groups are defined by a number of atoms given in an index file and may be two or three atoms in size. If -one is set, only one group should be specified in the index file and the angle between this group at time 0 and t will be computed. The angles calculated depend on the order in which the atoms are given. Giving for instance 5 6 will rotate the vector 5-6 with 180 degrees compared to giving 6 5.

If three atoms are given, the normal on the plane spanned by those three atoms will be calculated, using the formula P1P2 x P1P3. The cos of the angle is calculated, using the inproduct of the two normalized vectors.

Here is what some of the file options do:
-oa: Angle between the two groups specified in the index file. If a group contains three atoms the normal to the plane defined by those three atoms will be used. If a group contains two atoms, the vector defined by those two atoms will be used.
-od: Distance between two groups. Distance is taken from the center of one group to the center of the other group.
-od1: If one plane and one vector is given, the distances for each of the atoms from the center of the plane is given separately.
-od2: For two planes this option has no meaning.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input	Index file
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-oa`	`sg_angle.xvg`	Output	xvgr/xmgr file
`-od`	`sg_dist.xvg`	Output, Opt.	xvgr/xmgr file
`-od1`	`sg_dist1.xvg`	Output, Opt.	xvgr/xmgr file
`-od2`	`sg_dist2.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-one`	bool	`no`	Only one group compute angle between vector at time zero and time t
`-z`	bool	`no`	Use the Z-axis as reference

D.68 g_sham

g_sham makes multi-dimensional free-energy, enthalpy and entropy plots. g_sham reads one or more xvg files and analyzes data sets. g_sham basic purpose is plotting 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 due to Kumar et. al. When also 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 a dimension 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.

Files

`-f`	`graph.xvg`	Input	xvgr/xmgr file
`-ge`	`gibbs.xvg`	Input, Opt.	xvgr/xmgr file
`-ene`	`esham.xvg`	Input, Opt.	xvgr/xmgr file
`-dist`	`ener.xvg`	Output, Opt.	xvgr/xmgr file
`-histo`	`edist.xvg`	Output, Opt.	xvgr/xmgr file
`-bin`	`bindex.ndx`	Output, Opt.	Index file
`-lp`	`prob.xpm`	Output, Opt.	X PixMap compatible matrix file
`-ls`	`gibbs.xpm`	Output, Opt.	X PixMap compatible matrix file
`-lsh`	`enthalpy.xpm`	Output, Opt.	X PixMap compatible matrix file
`-lss`	`entropy.xpm`	Output, Opt.	X PixMap compatible matrix file
`-map`	`map.xpm`	Output, Opt.	X PixMap compatible matrix file
`-ls3`	`gibbs3.pdb`	Output, Opt.	Protein data bank file
`-mdata`	`mapdata.xvg`	Output, Opt.	xvgr/xmgr file
`-g`	`shamlog.log`	Output, Opt.	Log file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-time`	bool	`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 # sets separated by &
`-d`	bool	`no`	Use the derivative
`-bw`	real	`0.1`	Binwidth for the distribution
`-sham`	bool	`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
`-mname`	string		Legend label for the custom landscape

D.69 g_sigeps

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.

Files

`-o`	`potje.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `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`	sig
`-eps`	real	`1`	eps
`-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

D.70 g_sorient

g_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:
theta1: the angle with the vector from the first atom of the solvent molecule to the midpoint between atoms 2 and 3.
theta2: 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`: distribtion of cos(theta1) for rmin<=r<=rmax.

`-no`: distribution of cos(theta2) for rmin<=r<=rmax.

`-ro`: <cos(theta1)> and <3cos^2(theta2)-1> as a function of the distance.

`-co`: the sum over all solvent molecules within distance r of cos(theta1) and 3cos^2(theta2)-1 as a function of r.

`-rc`: the distribution of the solvent molecules as a function of r

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`sori.xvg`	Output	xvgr/xmgr file
`-no`	`snor.xvg`	Output	xvgr/xmgr file
`-ro`	`sord.xvg`	Output	xvgr/xmgr file
`-co`	`scum.xvg`	Output	xvgr/xmgr file
`-rc`	`scount.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-com`	bool	`no`	Use the center of mass as the reference postion
`-v23`	bool	`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)
`-pbc`	bool	`no`	Check PBC for the center of mass calculation. Only necessary when your reference group consists of several molecules.

D.71 g_spatial

g_spatial calculates the spatial distribution function and outputs it in a form that can be read by VMD as Gaussian98 cube format. This was developed from template.c (gromacs-3.3). For a system of 32K atoms and a 50ns 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 xtc 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 arbitrarty Cartesian coordinate. To do that, simply omit the preliminary trjconv steps.

USAGE:

1. Use make_ndx to create a group containing the atoms around which you want the SDF

2. trjconv -s a.tpr -f a.xtc -o b.xtc -center tric -ur compact -pbc none

3. trjconv -s a.tpr -f b.xtc -o c.xtc -fit rot+trans

4. run g_spatial on the xtc output of step #3.

5. Load grid.cube into VMD and view as an isosurface.

*** Systems such as micelles will require 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 trjconv 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.

BUGS:

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 option. However, the program does not detect all such events. If you encounter a segmentation fault, run it again with an increased -nab value.

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 trjconv. However, be sure to set the -nab option to a sufficiently high value since memory is allocated for cube bins based on the initial coords and the -nab (Number of Additional Bins) option value.

Files

`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-n`	`index.ndx`	Input, Opt.	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-pbc`	bool	`no`	Use periodic boundary conditions for computing distances
`-div`	bool	`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 in nm
`-nab`	int	`4`	Number of additional bins to ensure proper memory allocation

D.72 g_spol

g_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 at 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`scdist.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-com`	bool	`no`	Use the center of mass as the reference postion
`-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

D.73 g_tcaf

g_tcaf computes tranverse current autocorrelations. These are used to estimate the shear viscosity eta. For details see: Palmer, JCP 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 tcaf's. Each of these tcaf's 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 tau's and eta's. The fit weights decay with time as exp(-t/wt), the tcaf and fit are calculated up to time 5*wt. The eta's 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 tcaf's over all k-vectors with the same length. This results in more accurate tcaf's. Both the cubic tcaf's 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) = eta0 (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.

Files

`-f`	`traj.trr`	Input	Full precision trajectory: trr trj cpt

`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-ot`	`transcur.xvg`	Output, Opt.	xvgr/xmgr file
`-oa`	`tcaf_all.xvg`	Output	xvgr/xmgr file
`-o`	`tcaf.xvg`	Output	xvgr/xmgr file
`-of`	`tcaf_fit.xvg`	Output	xvgr/xmgr file
`-oc`	`tcaf_cub.xvg`	Output, Opt.	xvgr/xmgr file
`-ov`	`visc_k.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-mol`	bool	`no`	Calculate tcaf of molecules
`-k34`	bool	`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
`-normalize`	bool	`yes`	Normalize ACF
`-P`	enum	`0`	Order of Legendre polynomial for ACF (0 indicates none): `0`, `1`, `2` or `3`
`-fitfn`	enum	`none`	Fit function: `none`, `exp`, `aexp`, `exp_exp`, `vac`, `exp5`, `exp7` or `exp9`
`-ncskip`	int	`0`	Skip N points in the output file of correlation functions
`-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

D.74 g_traj

g_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 file with the average coordinates. 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 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt

`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-ox`	`coord.xvg`	Output, Opt.	xvgr/xmgr file
`-oxt`	`coord.xtc`	Output, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-ov`	`veloc.xvg`	Output, Opt.	xvgr/xmgr file
`-of`	`force.xvg`	Output, Opt.	xvgr/xmgr file
`-ob`	`box.xvg`	Output, Opt.	xvgr/xmgr file
`-ot`	`temp.xvg`	Output, Opt.	xvgr/xmgr file
`-ekt`	`ektrans.xvg`	Output, Opt.	xvgr/xmgr file
`-ekr`	`ekrot.xvg`	Output, Opt.	xvgr/xmgr file
`-vd`	`veldist.xvg`	Output, Opt.	xvgr/xmgr file
`-cv`	`veloc.pdb`	Output, Opt.	Protein data bank file
`-cf`	`force.pdb`	Output, Opt.	Protein data bank file
`-av`	`all_veloc.xvg`	Output, Opt.	xvgr/xmgr file
`-af`	`all_force.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-com`	bool	`no`	Plot data for the com of each group
`-mol`	bool	`no`	Index contains molecule numbers iso atom numbers
`-nojump`	bool	`no`	Remove jumps of atoms across the box
`-x`	bool	`yes`	Plot X-component
`-y`	bool	`yes`	Plot Y-component
`-z`	bool	`yes`	Plot Z-component
`-ng`	int	`1`	Number of groups to consider
`-len`	bool	`no`	Plot vector length
`-bin`	real	`1`	Binwidth for velocity histogram (nm/ps)
`-scale`	real	`0`	Scale factor for pdb output, 0 is autoscale

D.75 g_tune_pme

For a given number `-np` or `-nt` of processors/threads, this program systematically times mdrun with various numbers of PME-only nodes 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` file to g_tune_pme together with other options for mdrun as needed.

Which executables are used can be set in the environment variables MPIRUN and MDRUN. If these are not present, 'mpirun' and 'mdrun' will be used as defaults. 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"'

Please call g_tune_pme with the normal options you would pass to mdrun and add `-np` for the number of processors to perform the tests on, or `-nt` for the number of threads. You can also add `-r` to repeat each test several times to get better statistics.

g_tune_pme can test various real space / reciprocal space workloads for you. With `-ntpr` you control how many extra `.tpr` files will be written with enlarged cutoffs and smaller fourier grids respectively. Typically, the first test (no. 0) will be with the settings from the input `.tpr` file; the last test (no. `ntpr`) will have cutoffs multiplied by (and at the same time fourier grid dimensions divided by) the scaling factor `-fac` (default 1.2). The remaining `.tpr` files will have equally spaced values inbetween these extremes. Note that you can set `-ntpr` to 1 if you just want to find the optimal number of PME-only nodes; in that case your input `.tpr` 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) you may have to 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: `g_tune_pme -np 64 -s protein.tpr -launch`

After calling mdrun 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`.

Files

`-p`	`perf.out`	Output	Generic output file
`-err`	`errors.log`	Output	Log file
`-so`	`tuned.tpr`	Output	Run input file: tpr tpb tpa
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-o`	`traj.trr`	Output	Full precision trajectory: trr trj cpt
`-x`	`traj.xtc`	Output, Opt.	Compressed trajectory (portable xdr format)
`-cpi`	`state.cpt`	Input, Opt.	Checkpoint file
`-cpo`	`state.cpt`	Output, Opt.	Checkpoint file
`-c`	`confout.gro`	Output	Structure file: gro g96 pdb etc.
`-e`	`ener.edr`	Output	Energy file
`-g`	`md.log`	Output	Log file
`-dhdl`	`dhdl.xvg`	Output, Opt.	xvgr/xmgr file
`-field`	`field.xvg`	Output, Opt.	xvgr/xmgr file
`-table`	`table.xvg`	Input, Opt.	xvgr/xmgr file
`-tablep`	`tablep.xvg`	Input, Opt.	xvgr/xmgr file
`-tableb`	`table.xvg`	Input, Opt.	xvgr/xmgr file
`-rerun`	`rerun.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-tpi`	`tpi.xvg`	Output, Opt.	xvgr/xmgr file
`-tpid`	`tpidist.xvg`	Output, Opt.	xvgr/xmgr file
`-ei`	`sam.edi`	Input, Opt.	ED sampling input
`-eo`	`sam.edo`	Output, Opt.	ED sampling output
`-j`	`wham.gct`	Input, Opt.	General coupling stuff
`-jo`	`bam.gct`	Output, Opt.	General coupling stuff
`-ffout`	`gct.xvg`	Output, Opt.	xvgr/xmgr file
`-devout`	`deviatie.xvg`	Output, Opt.	xvgr/xmgr file
`-runav`	`runaver.xvg`	Output, Opt.	xvgr/xmgr file
`-px`	`pullx.xvg`	Output, Opt.	xvgr/xmgr file
`-pf`	`pullf.xvg`	Output, Opt.	xvgr/xmgr file
`-mtx`	`nm.mtx`	Output, Opt.	Hessian matrix
`-dn`	`dipole.ndx`	Output, Opt.	Index file
`-bo`	`bench.trr`	Output	Full precision trajectory: trr trj cpt

`-bx`	`bench.xtc`	Output	Compressed trajectory (portable xdr format)
`-bcpo`	`bench.cpt`	Output	Checkpoint file
`-bc`	`bench.gro`	Output	Structure file: gro g96 pdb etc.
`-be`	`bench.edr`	Output	Energy file
`-bg`	`bench.log`	Output	Log file
`-beo`	`bench.edo`	Output, Opt.	ED sampling output
`-bdhdl`	`benchdhdl.xvg`	Output, Opt.	xvgr/xmgr file
`-bfield`	`benchfld.xvg`	Output, Opt.	xvgr/xmgr file
`-btpi`	`benchtpi.xvg`	Output, Opt.	xvgr/xmgr file
`-btpid`	`benchtpid.xvg`	Output, Opt.	xvgr/xmgr file
`-bjo`	`bench.gct`	Output, Opt.	General coupling stuff
`-bffout`	`benchgct.xvg`	Output, Opt.	xvgr/xmgr file
`-bdevout`	`benchdev.xvg`	Output, Opt.	xvgr/xmgr file
`-brunav`	`benchrnav.xvg`	Output, Opt.	xvgr/xmgr file
`-bpx`	`benchpx.xvg`	Output, Opt.	xvgr/xmgr file
`-bpf`	`benchpf.xvg`	Output, Opt.	xvgr/xmgr file
`-bmtx`	`benchn.mtx`	Output, Opt.	Hessian matrix
`-bdn`	`bench.ndx`	Output, Opt.	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-np`	int	`1`	Number of nodes to run the tests on (must be >2 for separate PME nodes)
`-npstring`	enum	`-np`	Specify the number of processors to $MPIRUN using this string: `-np`, `-n` or `none`
`-nt`	int	`1`	Number of 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 nodes to test with
`-min`	real	`0.25`	Min fraction of PME nodes to test with
`-npme`	enum	`auto`	Benchmark all possible values for -npme or just the subset that is expected to perform well: `auto`, `all` or `subset`
`-upfac`	real	`1.2`	Upper limit for rcoulomb scaling factor (Note that rcoulomb upscaling results in fourier grid downscaling)
`-downfac`	real	`1`	Lower limit for rcoulomb scaling factor
`-ntpr`	int	`0`	Number of tpr files to benchmark. Create these many files with scaling factors ranging from 1.0 to fac. If <1, automatically choose the number of tpr files to test
`-four`	real	`0`	Use this fourierspacing value instead of the grid found in the tpr input file. (Spacing applies to a scaling factor of 1.0 if multiple tpr files are written)
`-steps`	step	`1000`	Take timings for these many steps in the benchmark runs
`-resetstep`	int	`100`	Let dlb equilibrate these many steps before timings are taken (reset cycle counters after these many steps)
`-simsteps`	step	`-1`	If non-negative, perform these many steps in the real run (overwrite nsteps from tpr, add cpt steps)
`-launch`	bool	`no`	Lauch the real simulation after optimization
`-deffnm`	string		Set the default filename for all file options at launch time
`-ddorder`	enum	`interleave`	DD node order: `interleave`, `pp_pme` or `cartesian`
`-ddcheck`	bool	`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` or `yes`
`-dds`	real	`0.8`	Minimum allowed dlb scaling of the DD cell size
`-gcom`	int	`-1`	Global communication frequency
`-v`	bool	`no`	Be loud and noisy
`-compact`	bool	`yes`	Write a compact log file
`-seppot`	bool	`no`	Write separate V and dVdl terms for each interaction type and node to the log file(s)
`-pforce`	real	`-1`	Print all forces larger than this (kJ/mol nm)
`-reprod`	bool	`no`	Try to avoid optimizations that affect binary reproducibility
`-cpt`	real	`15`	Checkpoint interval (minutes)
`-cpnum`	bool	`no`	Keep and number checkpoint files
`-append`	bool	`yes`	Append to previous output files when continuing from checkpoint instead of adding the simulation part number to all file names (for launch only)
`-maxh`	real	`-1`	Terminate after 0.99 times this time (hours)
`-multi`	int	`0`	Do multiple simulations in parallel
`-replex`	int	`0`	Attempt replica exchange every # steps
`-reseed`	int	`-1`	Seed for replica exchange, -1 is generate a seed
`-ionize`	bool	`no`	Do a simulation including the effect of an X-Ray bombardment on your system

D.76 g_vanhove

g_vanhove computes the Van Hove correlation function. The Van Hove G(r,t) is the probability that a particle that is at r0 at time zero can be found at position r0+r at time t. g_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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-om`	`vanhove.xpm`	Output, Opt.	X PixMap compatible matrix file
`-or`	`vanhove_r.xvg`	Output, Opt.	xvgr/xmgr file
`-ot`	`vanhove_t.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `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

D.77 g_velacc

g_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.

Files

`-f`	`traj.trr`	Input	Full precision trajectory: trr trj cpt
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-o`	`vac.xvg`	Output	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-m`	bool	`no`	Calculate the momentum autocorrelation function
`-mol`	bool	`no`	Calculate the velocity acf of molecules
`-acflen`	int	`-1`	Length of the ACF, default is half the number of frames
`-normalize`	bool	`yes`	Normalize ACF
`-P`	enum	`0`	Order of Legendre polynomial for ACF (0 indicates none): `0`, `1`, `2` or `3`
`-fitfn`	enum	`none`	Fit function: `none`, `exp`, `aexp`, `exp_exp`, `vac`, `exp5`, `exp7` or `exp9`
`-ncskip`	int	`0`	Skip N points in the output file of correlation functions
`-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

D.78 g_wham

This 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).

At present, three input modes are supported:
⋆ With option `-it`, the user provides a file which contains the filenames of the umbrella simulation run-input files (tpr files), AND, with option -ix, a file which contains filenames of the pullx mdrun output files. The tpr and pullx files must be in corresponding order, i.e. the first tpr created the first pullx, etc.
⋆ Same as the previous input mode, except that the the user provides the pull force ouput file names (pullf.xvg) with option -if. From the pull force the position in the ubrella potential is computed. This does not work with tabulated umbrella potentials. ⋆ With option `-ip`, the user provides filenames of (gzipped) pdo files, i.e. the gromacs 3.3 umbrella output files. If you have some unusual reaction coordinate you may also generate your own pdo files and feed them with the -ip option into to g_wham. The pdo file header must be similar to the folowing:

```
# UMBRELLA 3.0
# Component selection:  0 0 1
# nSkip 1
# Ref.  Group 'TestAtom'
# Nr.  of pull groups 2
# Group 1 'GR1' Umb.  Pos.  5.0 Umb.  Cons.  1000.0
# Group 2 'GR2' Umb.  Pos.  2.0 Umb.  Cons.  500.0
#####
```

Nr of pull groups, umbrella positions, force constants, and names may (of course) differ. Following the header, a time column and a data columns for each pull group follow (i.e. the displacement with respect to the umbrella center). Up to four pull groups are possible at present.

By default, the output files are
`-o` PMF output file
`-hist` histograms output file

The umbrella potential is assumed to be harmonic and the force constants are read from the tpr or pdo files. If a non-harmonic umbrella force was applied a tabulated potential can be provied with -tab.

WHAM OPTIONS

`-bins` Nr of bins used in analysis
`-temp` Temperature in the simulations
`-tol` Stop iteration if profile (probability) changed less than tolerance
`-auto` Automatic determination of boudndaries
`-min,-max` Boundaries of the profile
The data points which are used to compute the profile can be restricted with options -b, -e, and -dt. Play particularly with -b to ensure sufficient equilibration in each umbrella window!

With -log (default) the profile is written in energy units, otherwise (-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, choose with -zprof0 (useful with bootstrapping, see below).

For cyclic (or periodic) reaction coordinates (dihedral angle, channel PMF without osmotic gradient), -cycl is useful.
`-cycl yes` min and max are assumed to be neighboring points and histogram points outside min and max are mapped into the interval [min,max] (compare histogram output).
`-cycl weighted` First, a non-cyclic profile is computed. Subsequently, periodicity is enforced by adding corrections dG(i) between neighboring bins i and i+1. The correction is chosen proportional to 1/[n(i)*n(i+1)]âlpha, where n(i) denotes the total nr of data points in bin i as collected from all histograms.

alpha is defined with -alpha. The corrections are written to the file defined by -wcorr. (Compare Hub and de Groot, PNAS 105:1198 (2008))

ERROR ANALYSIS
Statistical errors may be estimated with bootstrap analysis. Use it with care, otherwise the statistical error may be substantially undererstimated !!
`-nBootstrap` defines the nr of bootstraps. Two bootstrapping modes are supported.
`-histbs` Complete histograms are considered as independent data points (default). For each bootstrap, N histograms are randomly chosen from the N given histograms (allowing duplication). 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 undererstimated!
`-nohistbs` The given histograms are used to generate new random histograms, such that the generated data points are distributed according the given histograms. The number of points generated for each bootstrap histogram can be controlled with -bs-dt. Note that one data point should be generated for each *independent* point in the given histograms. With the long autocorrelations in MD simulations, this procedure may easily understimate the error!
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 `-nohistBS`, the cummulants of the histogram.

Files

`-ix`	`pullx-files.dat`	Input, Opt.	Generic data file
`-if`	`pullf-files.dat`	Input, Opt.	Generic data file
`-it`	`tpr-files.dat`	Input, Opt.	Generic data file
`-ip`	`pdo-files.dat`	Input, Opt.	Generic data file
`-o`	`profile.xvg`	Output	xvgr/xmgr file
`-hist`	`histo.xvg`	Output	xvgr/xmgr file
`-bsres`	`bsResult.xvg`	Output, Opt.	xvgr/xmgr file
`-bsprof`	`bsProfs.xvg`	Output, Opt.	xvgr/xmgr file
`-tab`	`umb-pot.dat`	Input, Opt.	Generic data file
`-wcorr`	`cycl-corr.xvg`	Input, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-min`	real	`0`	Minimum coordinate in profile
`-max`	real	`0`	Maximum coordinate in profile
`-auto`	bool	`yes`	determine min and max automatically
`-bins`	int	`200`	Number of bins in profile
`-temp`	real	`298`	Temperature
`-tol`	real	`1e-06`	Tolerance
`-v`	bool	`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
`-histonly`	bool	`no`	Write histograms and exit
`-boundsonly`	bool	`no`	Determine min and max and exit (with -auto)
`-log`	bool	`yes`	Calculate the log of the profile before printing

`-unit`	enum	`kJ`	energy unit in case of log output: `kJ`, `kCal` or `kT`
`-zprof0`	real	`0`	Define profile to 0.0 at this position (with -log)
`-cycl`	enum	`no`	Create cyclic/periodic profile. Assumes min and max are the same point.: `no`, `yes` or `weighted`
`-alpha`	real	`2`	for '-cycl weighted', set parameter alpha
`-flip`	bool	`no`	Combine halves of profile (not supported)
`-hist-eq`	bool	`no`	Enforce equal weight for all histograms. (Non-Weighed-HAM)
`-nBootstrap`	int	`0`	nr of bootstraps to estimate statistical uncertainty
`-bs-dt`	real	`0`	timestep for synthetic bootstrap histograms (ps). Ensure independent data points!
`-bs-seed`	int	`-1`	seed for bootstrapping. (-1 = use time)
`-histbs`	bool	`yes`	In bootstrapping, consider complete histograms as one data point. Accounts better for long autocorrelations.
`-histbs-block`	int	`8`	when mixin histograms only mix within blocks of -histBS_block.
`-vbs`	bool	`no`	verbose bootstrapping. Print the cummulants and a histogram file for each bootstrap.

D.79 g_wheel

wheel plots a helical wheel representation of your sequence. The input sequence is in the .dat file where the first line contains the number of residues and each consecutive line contains a residuename.

Files

`-f`	`nnnice.dat`	Input	Generic data file
`-o`	`plot.eps`	Output	Encapsulated PostScript (tm) file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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)
`-nn`	bool	`yes`	Toggle numbers

D.80 g_x2top

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 entry, which you can then add to the rtp 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 forcefield 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 pdb2gmx just looks for the corresponding file.

Files

`-f`	`conf.gro`	Input	Structure file: gro g96 pdb tpr etc.
`-o`	`out.top`	Output, Opt.	Topology file
`-r`	`out.rtp`	Output, Opt.	Residue Type file used by pdb2gmx

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-ff`	string	`oplsaa`	Force field for your simulation. Type ”select” for interactive selection.
`-cwd`	bool	`no`	Also read force field files from the current working directory
`-v`	bool	`no`	Generate verbose output in the top file.
`-nexcl`	int	`3`	Number of exclusions
`-H14`	bool	`yes`	Use 3rd neighbour interactions for hydrogen atoms
`-alldih`	bool	`no`	Generate all proper dihedrals
`-remdih`	bool	`no`	Remove dihedrals on the same bond as an improper
`-pairs`	bool	`yes`	Output 1-4 interactions (pairs) in topology file
`-name`	string	`ICE`	Name of your molecule
`-pbc`	bool	`yes`	Use periodic boundary conditions.
`-pdbq`	bool	`no`	Use the B-factor supplied in a pdb file for the atomic charges
`-param`	bool	`yes`	Print parameters in the output
`-round`	bool	`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)

- 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 files in the data directory). Please extend it and send the results back to the GROMACS crew.

D.81 g_xrama

xrama shows a Ramachandran movie, that is, it shows the Phi/Psi angles as a function of time in an X-Window.

Static Phi/Psi plots for printing can be made with g_rama.

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

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-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 when t MOD dt = first time (ps)

D.82 make_edi

`make_edi` generates an essential dynamics (ED) sampling input file to be used with mdrun based on eigenvectors of a covariance matrix (`g_covar`) or from a normal modes anaysis (`g_nmeig`). 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 .edo 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 implentation point of view. Because parallel ED requires much extra communication, expect the performance to be lower as in a free MD simulation, especially on a large number of nodes.

All output of mdrun (specify with -eo) is written to a .edo file. In the output file, per OUTFRQ step the following information is present:

* the step number
the number of the ED dataset. (Note that you can impose multiple ED constraints in a single simulation - on different molecules e.g. - if several .edi files were concatenated first. The constraints are applied in the

order they appear in the .edi file.)
RMSD (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 configurational 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 file under DELTA_F0 and EFL_NULL.

Files

`-f`	`eigenvec.trr`	Input	Full precision trajectory: trr trj cpt
`-eig`	`eigenval.xvg`	Input, Opt.	xvgr/xmgr file
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-tar`	`target.gro`	Input, Opt.	Structure file: gro g96 pdb tpr etc.
`-ori`	`origin.gro`	Input, Opt.	Structure file: gro g96 pdb tpr etc.
`-o`	`sam.edi`	Output	ED sampling input

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `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
`-flood`	string		Indices of eigenvectors for flooding
`-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
`-outfrq`	int	`100`	Freqency (in steps) of writing output in .edo file
`-slope`	real	`0`	Minimal slope in acceptance radius expansion
`-maxedsteps`	int	`0`	Max nr of steps per cycle
`-deltaF0`	real	`150`	Target destabilization energy - used for flooding
`-deltaF`	real	`0`	Start deltaF with this parameter - default 0, i.e. 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
`-eqsteps`	int	`0`	Number of steps to run without any perturbations

`-Eflnull`	real	`0`	This is the starting value of the flooding strength. The flooding strength is updated according to the adaptive flooding scheme. To use 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
`-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
`-restrain`	bool	`no`	Use the flooding potential with inverted sign -> effects as quasiharmonic restraining potential
`-hessian`	bool	`no`	The eigenvectors and eigenvalues are from a Hessian matrix
`-harmonic`	bool	`no`	The eigenvalues are interpreted as spring constant

D.83 make_ndx

Index groups are necessary for almost every gromacs program. All these programs can generate default index groups. You ONLY have to use make_ndx when you need SPECIAL index groups. There is a default index group for the whole system, 9 default index groups are generated for proteins, a default index group is generated for every other residue name.

When no index file is supplied, also 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 NOT, AND and OR, you can split groups into chains, residues or atoms. You can delete and rename groups.

The atom numbering in the editor and the index file starts at 1.

Files

`-f`	`conf.gro`	Input, Opt.	Structure file: gro g96 pdb tpr etc.
`-n`	`index.ndx`	Input, Opt., Mult.	Index file
`-o`	`index.ndx`	Output	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-natoms`	int	`0`	set number of atoms (default: read from coordinate or index file)

D.84 mdrun

The mdrun program 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 g_nmeig.

The mdrun program reads the run input file (`-s`) and distributes the topology over nodes if needed. mdrun produces at least four output files. A single log file (`-g`) is written, unless the option `-seppot` is used, in which case each node writes a log file. 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.

When mdrun is started using MPI with more than 1 node, parallelization is used. By default domain decomposition is used, unless the `-pd` option is set, which selects particle decomposition.

With domain decomposition, the spatial decomposition can be set with option `-dd`. By default mdrun selects a good decomposition. The user only needs to change this when the system is very inhomogeneous. Dynamic load balancing is set with the option `-dlb`, which can give a significant performance improvement, especially for inhomogeneous systems. The only disadvantage of dynamic load balancing is that runs are no longer binary reproducible, but in most cases this is not important. By default the dynamic load balancing is automatically turned on when the measured performance loss due to load imbalance is 5% or more. At low parallelization these are the only important options for domain decomposition. At high parallelization the options in the next two sections could be important for increasing the performace.

When PME is used with domain decomposition, separate nodes can be assigned to do only the PME mesh calculation; this is computationally more efficient starting at about 12 nodes. The number of PME nodes is set with option `-npme`, this can not be more than half of the nodes. By default mdrun makes a guess for the number of PME nodes when the number of nodes is larger than 11 or performance wise not compatible with the PME grid x dimension. But the user should optimize npme. Performance statistics on this issue are written at the end of the log file. For good load balancing at high parallelization, the PME grid x and y dimensions should be divisible by the number of PME nodes (the simulation will run correctly also when this is not the case).

This section lists all options that affect the domain decomposition.
Option `-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. Atoms 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 indepedent 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 mdrun based on the initial coordinates. The chosen value will be a balance between interaction range and communication cost.
When inter charge-group bonded interactions are beyond the bonded cut-off distance, mdrun 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`.
When constraints are present, option `-rcon` influences the cell size limit as well. Atoms 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 mdrun 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 the option `-rcon`.
The `-dds` option sets the minimum allowed x, y and/or z scaling of the cells with dynamic load balancing. mdrun 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.

The option `-gcom` can be used to only do global communication every n steps. This can improve performance for highly parallel simulations where this global communication step becomes the bottleneck. For a global thermostat and/or barostat the temperature and/or pressure will also only be updated every -gcom steps. By default it is set to the minimum of nstcalcenergy and nstlist.

With `-rerun` an input trajectory can be given for which forces and energies will be (re)calculated. Neighbor searching will be performed for every frame, unless `nstlist` is zero (see the `.mdp` file).

ED (essential dynamics) sampling is switched on by using the `-ei` flag followed by an `.edi` file. The `.edi` file can be produced using options in the essdyn menu of the WHAT IF program. mdrun produces a `.edo` file that contains projections of positions, velocities and forces onto selected eigenvectors.

When user-defined potential functions have been selected in the `.mdp` 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 the table file name is modified in a different way: before the file extension an underscore is appended, then a b for bonds, an a for angles or a d for dihedrals and finally the table number of the interaction type.

The options `-px` and `-pf` are used for writing pull COM coordinates and forces when pulling is selected in the `.mdp` file.

With `-multi` multiple systems are simulated in parallel. As many input files are required as the number of systems. The system number is appended to the run input and each output filename, for instance topol.tpr becomes topol0.tpr, topol1.tpr etc. The number of nodes per system is the total number of nodes divided by the number of systems. One use of this option is for NMR refinement: when distance or orientation restraints are present these can be ensemble averaged over all the systems.

With `-replex` replica exchange is attempted every given number of steps. The number of replicas is set with the `-multi` option, see above. All run input files should use a different coupling temperature, the order of the files is not important. The random seed is set with `-reseed`. The velocities are scaled and neighbor searching is performed after every exchange.

Finally some experimental algorithms can be tested when the appropriate options have been given. Currently under investigation are: polarizability, and X-Ray bombardments.

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.

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 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 nodes 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.

When mdrun receives a TERM signal, it will set nsteps to the current step plus one. When mdrun receives an INT signal (e.g. when ctrl+C is pressed), it will stop after the next neighbor search step (with nstlist=0 at the next step). In both cases all the usual output will be written to file. When running with MPI, a signal to one of the mdrun processes is sufficient, this signal should not be sent to mpirun or the mdrun process that is the parent of the others.

When mdrun is started with MPI, it does not run niced by default.

Files

`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-o`	`traj.trr`	Output	Full precision trajectory: trr trj cpt
`-x`	`traj.xtc`	Output, Opt.	Compressed trajectory (portable xdr format)
`-cpi`	`state.cpt`	Input, Opt.	Checkpoint file
`-cpo`	`state.cpt`	Output, Opt.	Checkpoint file
`-c`	`confout.gro`	Output	Structure file: gro g96 pdb etc.
`-e`	`ener.edr`	Output	Energy file
`-g`	`md.log`	Output	Log file
`-dhdl`	`dhdl.xvg`	Output, Opt.	xvgr/xmgr file
`-field`	`field.xvg`	Output, Opt.	xvgr/xmgr file
`-table`	`table.xvg`	Input, Opt.	xvgr/xmgr file
`-tablep`	`tablep.xvg`	Input, Opt.	xvgr/xmgr file
`-tableb`	`table.xvg`	Input, Opt.	xvgr/xmgr file
`-rerun`	`rerun.xtc`	Input, Opt.	Trajectory: xtc trr trj gro g96 pdb cpt
`-tpi`	`tpi.xvg`	Output, Opt.	xvgr/xmgr file
`-tpid`	`tpidist.xvg`	Output, Opt.	xvgr/xmgr file
`-ei`	`sam.edi`	Input, Opt.	ED sampling input
`-eo`	`sam.edo`	Output, Opt.	ED sampling output
`-j`	`wham.gct`	Input, Opt.	General coupling stuff
`-jo`	`bam.gct`	Output, Opt.	General coupling stuff
`-ffout`	`gct.xvg`	Output, Opt.	xvgr/xmgr file
`-devout`	`deviatie.xvg`	Output, Opt.	xvgr/xmgr file
`-runav`	`runaver.xvg`	Output, Opt.	xvgr/xmgr file
`-px`	`pullx.xvg`	Output, Opt.	xvgr/xmgr file
`-pf`	`pullf.xvg`	Output, Opt.	xvgr/xmgr file
`-mtx`	`nm.mtx`	Output, Opt.	Hessian matrix
`-dn`	`dipole.ndx`	Output, Opt.	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-deffnm`	string		Set the default filename for all file options
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-pd`	bool	`no`	Use particle decompostion
`-dd`	vector	`0 0 0`	Domain decomposition grid, 0 is optimize
`-nt`	int	`0`	Number of threads to start (0 is guess)

`-npme`	int	`-1`	Number of separate nodes to be used for PME, -1 is guess
`-ddorder`	enum	`interleave`	DD node order: `interleave`, `pp_pme` or `cartesian`
`-ddcheck`	bool	`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` or `yes`
`-dds`	real	`0.8`	Minimum allowed dlb scaling of the DD cell size
`-gcom`	int	`-1`	Global communication frequency
`-v`	bool	`no`	Be loud and noisy
`-compact`	bool	`yes`	Write a compact log file
`-seppot`	bool	`no`	Write separate V and dVdl terms for each interaction type and node to the log file(s)
`-pforce`	real	`-1`	Print all forces larger than this (kJ/mol nm)
`-reprod`	bool	`no`	Try to avoid optimizations that affect binary reproducibility
`-cpt`	real	`15`	Checkpoint interval (minutes)
`-cpnum`	bool	`no`	Keep and number checkpoint files
`-append`	bool	`yes`	Append to previous output files when continuing from checkpoint instead of adding the simulation part number to all file names
`-maxh`	real	`-1`	Terminate after 0.99 times this time (hours)
`-multi`	int	`0`	Do multiple simulations in parallel
`-replex`	int	`0`	Attempt replica exchange every # steps
`-reseed`	int	`-1`	Seed for replica exchange, -1 is generate a seed
`-ionize`	bool	`no`	Do a simulation including the effect of an X-Ray bombardment on your system

D.85 mk_angndx

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.

Files

`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-n`	`angle.ndx`	Output	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-type`	enum	`angle`	Type of angle: `angle`, `dihedral`, `improper` or `ryckaert-bellemans`
`-hyd`	bool	`yes`	Include angles with atoms with mass < 1.5
`-hq`	real	`-1`	Ignore angles with atoms with mass < 1.5 and orqor < hq

D.86 ngmx

ngmx 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Run input file: tpr tpb tpa
`-n`	`index.ndx`	Input, Opt.	Index file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-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 when t MOD dt = first time (ps)

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

D.87 pdb2gmx

This program reads a pdb file, reads some database files, adds hydrogens to the molecules and generates coordinates in Gromacs (Gromos) format and a topology in Gromacs format. These files can subsequently be processed to generate a run input file.

The force fields in the distribution are currently:

oplsaa OPLS-AA/L all-atom force field (2001 aminoacid dihedrals)
gromos43a1 GROMOS96 43a1 Forcefield
gromos43a2 GROMOS96 43a2 Forcefield (improved alkane dihedrals)
gromos45a3 GROMOS96 45a3 Forcefield
gromos53a5 GROMOS96 53a5 Forcefield
gromos53a6 GROMOS96 53a6 Forcefield
gmx Gromacs Forcefield (a modified GROMOS87, see manual)
encads Encad all-atom force field, using scaled-down vacuum charges
encadv Encad all-atom force field, using full solvent charges

The corresponding data files can be found in the library directory in the subdirectory <forcefield>.ff. Note that pdb2gmx will also look for a `forcefield.itp` file in such subdirectories in the current working directory. After choosing a force field, all files will be read only from the corresponding directory, unless the `-cwd` option is used. Check chapter 5 of the manual for more information about file formats. By default the forcefield 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 pdb2gmx just looks for the corresponding file.

Note that a pdb 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 she wants. 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 criterium, specified by the maximum hydrogen-donor-acceptor angle and donor-acceptor distance, which are set by `-angle` and `-dist` respectively.

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 have a HEME group bound to a protein. To handle this, pdb2gmx now has a new option `-chainsep` so you can choose whether a new chain should start when we find a TER record, when the chain id changes or combinations of either or both of these. There is als an option `-merge` to interactively ask if you want to merge consecutive chains into one molecule - this can be useful for connecting chains with a disulfide brigde or intermolecular distance restraints.

pdb2gmx will also check the occupancy field of the pdb 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 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` and `.g96` file formats do not support chain identifiers. Therefore it is useful to enter a pdb file name at the `-o` option when you want to convert a multichain pdb 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, elminating the fast improper dihedral fluctuations in these rings. 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.

Files

`-f`	`eiwit.pdb`	Input	Structure file: gro g96 pdb tpr etc.
`-o`	`conf.gro`	Output	Structure file: gro g96 pdb etc.
`-p`	`topol.top`	Output	Topology file
`-i`	`posre.itp`	Output	Include file for topology
`-n`	`clean.ndx`	Output, Opt.	Index file
`-q`	`clean.pdb`	Output, Opt.	Structure file: gro g96 pdb etc.

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-cwd`	bool	`no`	Also read force field files from the current working directory
`-rtpo`	bool	`no`	Allow an entry in a local rtp file to override a library rtp entry
`-chainsep`	enum		
		`id_or_ter`	Condition in PDB files when a new chain should be started: `id_or_ter`, `id_and_ter`, `ter`, `id` or `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 or tip5p
-inter	bool	no	Set the next 8 options to interactive
-ss	bool	no	Interactive SS bridge selection
-ter	bool	no	Interactive termini selection, iso charged
-lys	bool	no	Interactive Lysine selection, iso charged
-arg	bool	no	Interactive Arganine selection, iso charged
-asp	bool	no	Interactive Aspartic Acid selection, iso charged
-glu	bool	no	Interactive Glutamic Acid selection, iso charged
-gln	bool	no	Interactive Glutamine selection, iso neutral
-his	bool	no	Interactive Histidine selection, iso 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)
-una	bool	no	Select aromatic rings with united CH atoms on Phenylalanine, Tryptophane and Tyrosine
-ignh	bool	no	Ignore hydrogen atoms that are in the pdb file
-missing	bool	no	Continue when atoms are missing, dangerous
-v	bool	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 or aromatics
-heavyh	bool	no	Make hydrogen atoms heavy
-deuterate	bool	no	Change the mass of hydrogens to 2 amu
-chargegrp	bool	yes	Use charge groups in the rtp file
-cmap	bool	yes	Use cmap torsions (if enabled in the rtp file)
-renum	bool	no	Renumber the residues consecutively in the output
-rtpres	bool	no	Use rtp entry names as residue names

D.88 tpbconv

tpbconv can edit run input files in four ways.

1st. by modifying the number of steps in a run input file with options -extend, -until or -nsteps (nsteps=-1 means unlimited number of steps)

2nd. (OBSOLETE) by creating a run input file for a continuation run when your simulation has crashed due to e.g. a full disk, or by making a continuation run input file. This option is obsolete, since mdrun now writes and reads checkpoint files. Note that a frame with coordinates and velocities is needed. When pressure and/or Nose-Hoover temperature coupling is used an energy file can be supplied to get an exact continuation of the original run.

3rd. 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 Ca tpx file. **WARNING: this tpx file is not fully functional**. **4th.** 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.

Files

-s	topol.tpr	Input	Run input file: tpr tpb tpa
-f	traj.trr	Input, Opt.	Full precision trajectory: trr trj cpt
-e	ener.edr	Input, Opt.	Energy file
-n	index.ndx	Input, Opt.	Index file
-o	tpxout.tpr	Output	Run input file: tpr tpb tpa

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-extend`	real	`0`	Extend runtime by this amount (ps)
`-until`	real	`0`	Extend runtime until this ending time (ps)
`-nsteps`	int	`0`	Change the number of steps
`-time`	real	`-1`	Continue from frame at this time (ps) instead of the last frame
`-zeroq`	bool	`no`	Set the charges of a group (from the index) to zero
`-vel`	bool	`yes`	Require velocities from trajectory
`-cont`	bool	`yes`	For exact continuation, the constraints should not be applied before the first step

D.89 trjcat

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 `trjcat -o fixed.trr *.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 file. The xvg file should contain something like:

0 0 1 2 3 4 5
2 1 0 2 3 5 4
Where 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 file then the program tries to be smart. Beware.

Files

`-f`	`traj.xtc`	Input, Mult.	Trajectory: xtc trr trj gro g96 pdb cpt
`-o`	`trajout.xtc`	Output, Mult.	Trajectory: xtc trr trj gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-demux`	`remd.xvg`	Input, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `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)
`-prec`	int	`3`	Precision for .xtc and .gro writing in number of decimal places
`-vel`	bool	`yes`	Read and write velocities if possible
`-settime`	bool	`no`	Change starting time interactively
`-sort`	bool	`yes`	Sort trajectory files (not frames)

`-keeplast`	bool	`no`	keep overlapping frames at end of trajectory
`-overwrite`	bool	`no`	overwrite overlapping frames during appending
`-cat`	bool	`no`	do not discard double time frames

D.90 trjconv

trjconv can convert trajectory files in many ways:
1. from one format to another
2. select a subset of atoms
3. change the periodicity representation
4. keep multimeric molecules together
5. center atoms in the box
6. fit atoms to reference structure
7. reduce the number of frames
8. change the timestamps of the frames (`-t0` and `-timestep`)
9. cut the trajectory in small subtrajectories according to information in an index file. This allows subsequent analysis of the subtrajectories that could, for example be the result of a cluster analysis. Use option `-sub`. This assumes that the entries in the index file are frame numbers and dumps each group in the index file to a separate trajectory file.
10. select frames within a certain range of a quantity given in an `.xvg` file.

The program `trjcat` can concatenate multiple trajectory files.

Currently seven formats are supported for input and output: `.xtc`, `.trr`, `.trj`, `.gro`, `.g96`, `.pdb` and `.g87`. The file formats are detected from the file extension. The precision of `.xtc` and `.gro` output is taken from the input file for `.xtc`, `.gro` and `.pdb`, 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` and `.trj` output can be single or double precision, depending on the precision of the trjconv binary. Note that velocities are only supported in `.trr`, `.trj`, `.gro` and `.g96` files.

Option `-app` can be used to append output to an existing trajectory file. No checks are performed to ensure integrity of the resulting combined trajectory file.

Option `-sep` can be used to write every frame to a separate .gro, .g96 or .pdb file, default all frames all written to one file. `.pdb` 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` 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 or for whatever. 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.
`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 octahedrons. 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.

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 `g_filter` can reduce the number of frames while using low-pass frequency filtering, this reduces aliasing of high frequency motions.

Using `-trunc` trjconv can truncate `.trj` in place, i.e. without copying the file. This is useful when a run has crashed during disk I/O (one more disk full), or when two contiguous trajectories must be concatenated without have double frames.

`trjcat` is more suitable for concatenating trajectory files.

Option `-dump` can be used to extract a frame at or near one specific time from your trajectory.

Option `-drop` reads an `.xvg` 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-o`	`trajout.xtc`	Output	Trajectory: xtc trr trj gro g96 pdb
`-s`	`topol.tpr`	Input, Opt.	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file
`-fr`	`frames.ndx`	Input, Opt.	Index file
`-sub`	`cluster.ndx`	Input, Opt.	Index file
`-drop`	`drop.xvg`	Input, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-b`	time	`0`	First frame (ps) to read from trajectory
`-e`	time	`0`	Last frame (ps) to read from trajectory
`-tu`	enum	`ps`	Time unit: `fs`, `ps`, `ns`, `us`, `ms` or `s`
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-skip`	int	`1`	Only write every nr-th frame
`-dt`	time	`0`	Only write frame when t MOD dt = first time (ps)
`-round`	bool	`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` or `whole`
`-ur`	enum	`rect`	Unit-cell representation: `rect`, `tric` or `compact`
`-center`	bool	`no`	Center atoms in box
`-boxcenter`	enum	`tric`	Center for -pbc and -center: `tric`, `rect` or `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` or `progressive`
`-ndec`	int	`3`	Precision for .xtc and .gro writing in number of decimal places
`-vel`	bool	`yes`	Read and write velocities if possible
`-force`	bool	`no`	Read and write forces if possible
`-trunc`	time	`-1`	Truncate input trj file after this time (ps)
`-exec`	string		Execute command for every output frame with the frame number as argument
`-app`	bool	`no`	Append output
`-split`	time	`0`	Start writing new file when t MOD split = first time (ps)
`-sep`	bool	`no`	Write each frame to a separate .gro, .g96 or .pdb file
`-nzero`	int	`0`	Prepend file number in case you use the -sep flag with this number of zeroes
`-dropunder`	real	`0`	Drop all frames below this value
`-dropover`	real	`0`	Drop all frames above this value
`-conect`	bool	`no`	Add conect records when writing pdb files. Useful for visualization of non-standard molecules, e.g. coarse grained ones

D.91 trjorder

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.

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 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.

Files

`-f`	`traj.xtc`	Input	Trajectory: xtc trr trj gro g96 pdb cpt
`-s`	`topol.tpr`	Input	Structure+mass(db): tpr tpb tpa gro g96 pdb
`-n`	`index.ndx`	Input, Opt.	Index file

`-o`	`ordered.xtc`	Output, Opt.	Trajectory: xtc trr trj gro g96 pdb
`-nshell`	`nshell.xvg`	Output, Opt.	xvgr/xmgr file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`19`	Set the nicelevel
`-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 when t MOD dt = first time (ps)
`-xvg`	enum	`xmgrace`	xvg plot formatting: `xmgrace`, `xmgr` or `none`
`-na`	int	`3`	Number of atoms in a molecule
`-da`	int	`1`	Atom used for the distance calculation, 0 is COM
`-com`	bool	`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
`-z`	bool	`no`	Order molecules on z-coordinate

D.92 xpm2ps

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 do_dssp, g_rms or g_mdmat.

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 setting are set by 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 2nd 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 grey-scale matrices can be turned into attractive color pictures.

Merged or rainbowed matrices can be written to an XPixelMap file with the `-xpm` option.

Files

`-f`	`root.xpm`	Input	X PixMap compatible matrix file
`-f2`	`root2.xpm`	Input, Opt.	X PixMap compatible matrix file
`-di`	`ps.m2p`	Input, Opt., Lib.	Input file for mat2ps

`-do`	`out.m2p`	Output, Opt.	Input file for mat2ps
`-o`	`plot.eps`	Output, Opt.	Encapsulated PostScript (tm) file
`-xpm`	`root.xpm`	Output, Opt.	X PixMap compatible matrix file

Other options

`-h`	bool	`no`	Print help info and quit
`-version`	bool	`no`	Print version info and quit
`-nice`	int	`0`	Set the nicelevel
`-w`	bool	`no`	View output xvg, xpm, eps and pdb files
`-frame`	bool	`yes`	Display frame, ticks, labels, title and legend
`-title`	enum	`top`	Show title at: `top`, `once`, `ylabel` or `none`
`-yonce`	bool	`no`	Show y-label only once
`-legend`	enum	`both`	Show legend: `both`, `first`, `second` or `none`
`-diag`	enum	`first`	Diagonal: `first`, `second` or `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` or `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
`-zeroline`	bool	`no`	insert line in xpm matrix where axis label is zero
`-legoffset`	int	`0`	Skip first N colors from xpm file for the legend
`-combine`	enum	`halves`	Combine two matrices: `halves`, `add`, `sub`, `mult` or `div`
`-cmin`	real	`0`	Minimum for combination output
`-cmax`	real	`0`	Maximum for combination output

Bibliography

[1] Bekker, H., Berendsen, H. J. C., Dijkstra, E. J., Achterop, S., van Drunen, R., van der Spoel, D., Sijbers, A., Keegstra, H., Reitsma, B., Renardus, M. K. R. Gromacs: A parallel computer for molecular dynamics simulations. In *Physics Computing 92* (Singapore, 1993). de Groot, R. A., Nadrchal, J., eds. . World Scientific.

[2] Berendsen, H. J. C., van der Spoel, D., van Drunen, R. GROMACS: A message-passing parallel molecular dynamics implementation. Comp. Phys. Comm. 91:43–56, 1995.

[3] Lindahl, E., Hess, B., van der Spoel, D. Gromacs 3.0: A package for molecular simulation and trajectory analysis. J. Mol. Mod. 7:306–317, 2001.

[4] van der Spoel, D., Lindahl, E., Hess, B., Groenhof, G., Mark, A. E., Berendsen, H. J. C. GROMACS: Fast, Flexible and Free. J. Comp. Chem. 26:1701–1718, 2005.

[5] van Gunsteren, W. F., Berendsen, H. J. C. Computer simulation of molecular dynamics: Methodology, applications, and perspectives in chemistry. Angew. Chem. Int. Ed. Engl. 29:992–1023, 1990.

[6] Fraaije, J. G. E. M. Dynamic density functional theory for microphase separation kinetics of block copolymer melts. J. Chem. Phys. 99:9202–9212, 1993.

[7] McQuarrie, D. A. Statistical Mechanics. New York: Harper & Row. 1976.

[8] van Gunsteren, W. F., Berendsen, H. J. C. Algorithms for macromolecular dynamics and constraint dynamics. Mol. Phys. 34:1311–1327, 1977.

[9] van Gunsteren, W. F., Karplus, M. Effect of constraints on the dynamics of macromolecules. Macromolecules 15:1528–1544, 1982.

[10] Darden, T., York, D., Pedersen, L. Particle mesh Ewald: An N-log(N) method for Ewald sums in large systems. J. Chem. Phys. 98:10089–10092, 1993.

[11] Essmann, U., Perera, L., Berkowitz, M. L., Darden, T., Lee, H., Pedersen, L. G. A smooth particle mesh ewald potential. J. Chem. Phys. 103:8577–8592, 1995.

[12] Geman, S., Geman, D. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. IEEE Trans. Patt. Anal. Mach. Int. 6:721, 1984.

[13] Nilges, M., Clore, G. M., Gronenborn, A. M. 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.

[14] van Schaik, R. C., Berendsen, H. J. C., Torda, A. E., van Gunsteren, W. F. A structure refinement method based on molecular dynamics in 4 spatial dimensions. J. Mol. Biol. 234:751–762, 1993.

[15] Zimmerman, K. All purpose molecular mechanics simulator and energy minimizer. J. Comp. Chem. 12:310–319, 1991.

[16] Adams, D. J., Adams, E. M., Hills, G. J. The computer simulation of polar liquids. Mol. Phys. 38:387–400, 1979.

[17] Bekker, H., Dijkstra, E. J., Renardus, M. K. R., Berendsen, H. J. C. An efficient, box shape independent nonbonded force and virial algorithm for molecular dynamics. Mol. Sim. 14:137–152, 1995.

[18] Hockney, R. W., Goel, S. P., Eastwood, J. Quiet highresolution computer models of a plasma. J. Comp. Phys. 14:148–158, 1974.

[19] Verlet., L. Computer experiments on classical fluids. i. thermodynamical properties of lennard-jones molecules. Phys. Rev. 159:98–103, 1967.

[20] Berendsen, H. J. C., van Gunsteren, W. F. Practical algorithms for dynamics simulations.

[21] Swope, W. C., Andersen, H. C., Berens, P. H., Wilson, K. R. 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.

[22] Tuckerman, M., Berne, B. J., Martyna, G. J. Reversible multiple time scale molecular dynamics. J. Chem. Phys. 97(3):1990–2001, 1992.

[23] Berendsen, H. J. C., Postma, J. P. M., DiNola, A., Haak, J. R. Molecular dynamics with coupling to an external bath. J. Chem. Phys. 81:3684–3690, 1984.

[24] Nosé, S. A molecular dynamics method for simulations in the canonical ensemble. Mol. Phys. 52:255–268, 1984.

[25] Hoover, W. G. Canonical dynamics: equilibrium phase-space distributions. Phys. Rev. **A** 31:1695–1697, 1985.

[26] Bussi, G., Donadio, D., Parrinello, M. Canonical sampling through velocity rescaling. J. Chem. Phys. 126:014101, 2007.

[27] Berendsen, H. J. C. Transport properties computed by linear response through weak coupling to a bath. In: Computer Simulations in Material Science. Meyer, M., Pontikis, V. eds. . Kluwer 1991 139–155.

[28] Martyna, G. J., Klein, M. L., Tuckerman, M. E. Nosé hoover chains: The canonical ensemble via continuous dynamics. J. Chem. Phys. 97:2635–2643, 1992.

[29] Martyna, G. J., Tuckerman, M. E., Tobias, D. J., Klein, M. L. Explicit reversible integrators for extended systems dynamics. Mol. Phys. 87:1117–1157, 1996.

[30] Holian, B. L., Voter, A. F., Ravelo, R. Thermostatted molecular dynamics: How to avoid the toda demon hidden in nosé-hoover dynamics. Phys. Rev. E 52(3):2338–2347, 1995.

[31] Eastwood, M. P., Stafford, K. A., Lippert, R. A., Jensen, M. O., Maragakis, P., Predescu, C., Dror, R. O., Shaw, D. E. Equipartition and the calculation of temperature in biomolecular simulations. J. Chem. Theory Comp. ASAP:DOI: 10.1021/ct9002916, 2010.

[32] Parrinello, M., Rahman, A. Polymorphic transitions in single crystals: A new molecular dynamics method. J. Appl. Phys. 52:7182–7190, 1981.

[33] Nosé, S., Klein, M. L. Constant pressure molecular dynamics for molecular systems. Mol. Phys. 50:1055–1076, 1983.

[34] Tuckerman, M. E., Alejandre, J., López-Rendón, R., Jochim, A. L., Martyna, G. J. A liouville-operator derived measure-preserving integrator for molecular dynamics simulations in the isothermal-isobaric ensemble. J. Phys. A. 59:5629–5651, 2006.

[35] Yu, T.-Q., Alejandre, J., Lopez-Rendon, R., Martyna, G. J., Tuckerman, M. E. Measure-preserving integrators for molecular dynamics in the isothermal-isobaric ensemble derived from the liouville operator. Chem. Phys. 370:294–305, 2010.

[36] Dick, B. G., Overhauser, A. W. Theory of the dielectric constants of alkali halide crystals. Phys. Rev. 112:90–103, 1958.

[37] Jordan, P. C., van Maaren, P. J., Mavri, J., van der Spoel, D., Berendsen, H. J. C. Towards phase transferable potential functions: Methodology and application to nitrogen. J. Chem. Phys. 103:2272–2285, 1995.

[38] van Maaren, P. J., van der Spoel, D. Molecular dynamics simulations of a water with a novel shell-model potential. J. Phys. Chem. B. 105:2618–2626, 2001.

[39] Ryckaert, J. P., Ciccotti, G., Berendsen, H. J. C. 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.

[40] Miyamoto, S., Kollman, P. A. SETTLE: An analytical version of the SHAKE and RATTLE algorithms for rigid water models. J. Comp. Chem. 13:952–962, 1992.

[41] Hess, B., Bekker, H., Berendsen, H. J. C., Fraaije, J. G. E. M. LINCS: A linear constraint solver for molecular simulations. J. Comp. Chem. 18:1463–1472, 1997.

[42] Hess, B. P-lincs: A parallel linear constraint solver for molecular simulation. J. Chem. Theory Comp. 4:116–122, 2007.

[43] van Gunsteren, W. F., Berendsen, H. J. C. A leap-frog algorithm for stochastic dynamics. Mol. Sim. 1:173–185, 1988.

[44] Byrd, R. H., Lu, P., Nocedal, J. A limited memory algorithm for bound constrained optimization. SIAM J. Scientif. Statistic. Comput. 16:1190–1208, 1995.

[45] Zhu, C., Byrd, R. H., Nocedal, J. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization. ACM Trans. Math. Softw. 23:550–560, 1997.

[46] Levitt, M., Sander, C., Stern, P. S. The normal modes of a protein: Native bovine pancreatic trypsin inhibitor. Int. J. Quant. Chem: Quant. Biol. Symp. 10:181–199, 1983.

[47] Gō, N., Noguti, T., Nishikawa, T. Dynamics of a small globular protein in terms of low-frequency vibrational modes. Proc. Natl. Acad. Sci. USA 80:3696–3700, 1983.

[48] Brooks, B., Karplus, M. Harmonic dynamics of proteins: Normal modes and fluctuations in bovine pancreatic trypsin inhibitor. Proc. Natl. Acad. Sci. USA 80:6571–6575, 1983.

[49] Hayward, S., Gō, N. Collective variable description of native protein dynamics. Annu. Rev. Phys. Chem. 46:223–250, 1995.

[50] Hukushima, K., Nemoto, K. Exchange monte carlo method and application to spin glass simulations. J. Phys. Soc. Jpn. 65:1604–1608, 1996.

[51] Sugita, Y., Okamoto, Y. Replica-exchange molecular dynamics method for proteint folding. Chem. Phys. Lett. 314:141–151, 1999.

[52] Seibert, M., Patriksson, A., Hess, B., van der Spoel, D. Reproducible polypeptide folding and structure prediction using molecular dynamics simulations. J. Mol. Biol. 354:173–183, 2005.

[53] Okabe, T., Kawata, M., Okamoto, Y., Mikami, M. Replica-exchange Monte Carlo method for the isobaric-isothermal ensemble. Chem. Phys. Lett. 335:435–439, 2001.

[54] de Groot, B. L., Amadei, A., van Aalten, D. M. F., Berendsen, H. J. C. 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.

[55] de Groot, B. L., Amadei, A., Scheek, R. M., van Nuland, N. A. J., Berendsen, H. J. C. An extended sampling of the configurational space of hpr from *e. coli*. PROTEINS: Struct. Funct. Gen. 26:314–322, 1996.

[56] Lange, O. E., Schafer, L. V., Grubmuller, H. Flooding in gromacs: Accelerated barrier crossings in molecular dynamics. J. Comp. Chem. 27:1693–1702, 2006.

[57] Liem, S. Y., Brown, D., Clarke, J. H. R. Molecular dynamics simulations on distributed memory machines. Comput. Phys. Commun. 67(2):261–267, 1991.

[58] Bowers, K. J., Dror, R. O., Shaw, D. E. The midpoint method for parallelization of particle simulations. J. Chem. Phys. 124(18):184109–184109, 2006.

[59] Hess, B., Kutzner, C., van der Spoel, D., Lindahl, E. Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. J. Chem. Theory Comp. 435, 2008.

[60] Qui, D., Shenkin, P., Hollinger, F., Still, W. The GB/SA continuum model for solvation. A fast analytical method for the calculation of approximate born radii. J. Phys. Chem. A. 101:3005–3014, 1997.

[61] Hawkins, D., Cramer, C., Truhlar, D. Parametrized models of aqueous free energies of solvation based on pairwise descreening of solute atomic charges from a dielectric medium. J. Phys. Chem. A. 100:19824–19839, 1996.

[62] Onufriev, A., Bashford, D., Case, D. Exploring protein native states and large-scale conformational changes with a modified generalized born model. 55(2).

[63] Larsson, P., Lindahl, E. A high-performance parallel-generalized born implementation enabled by tabulated interaction rescaling. J. Comp. Chem.

[64] Schaefer, M., Bartels, C., Karplus, M. Solution conformations and thermodynamics of structured peptides: molecula r dynamics simulation with an implicit solvation model. J. Mol. Biol. 284(3):835–848, 1998.

[65] Tironi, I. G., Sperb, R., Smith, P. E., van Gunsteren, W. F. A generalized reaction field method for molecular dynamics simulations. J. Chem. Phys. 102:5451–5459, 1995.

[66] van der Spoel, D., van Maaren, P. J. The origin of layer structure artifacts in simulations of liquid water. J. Chem. Theory Comp. 2:1–11, 2006.

[67] Berendsen, H. J. C. Electrostatic interactions. In: Computer Simulation of Biomolecular Systems. van Gunsteren, W. F., Weiner, P. K., Wilkinson, A. J. eds. . ESCOM Leiden 1993 161–181.

[68] van Gunsteren, W. F., Billeter, S. R., Eising, A. A., Hünenberger, P. H., Krüger, P., Mark, A. E., Scott, W. R. P., Tironi, I. G. Biomolecular Simulation: The GROMOS96 manual and user guide. Zürich, Switzerland: Hochschulverlag AG an der ETH Zürich. 1996.

[69] van Gunsteren, W. F., Berendsen, H. J. C. Gromos-87 manual. Biomos BV Nijenborgh 4, 9747 AG Groningen, The Netherlands 1987.

[70] Morse, P. M. Diatomic molecules according to the wave mechanics. II. vibrational levels. Phys. Rev. 34:57–64, 1929.

[71] Berendsen, H. J. C., Postma, J. P. M., van Gunsteren, W. F., Hermans, J. Interaction models for water in relation to protein hydration. In: Intermolecular Forces. Pullman, B. ed. . D. Reidel Publishing Company Dordrecht 1981 331–342.

[72] Ferguson, D. M. Parametrization and evaluation of a flexible water model. J. Comp. Chem. 16:501–511, 1995.

[73] Warner Jr., H. R. Kinetic theory and rheology of dilute suspensions of finitely extendible dumbbells. Ind. Eng. Chem. Fundam. 11(3):379–387, 1972.

[74] Brooks, B. R., Bruccoleri, R. E., Olafson, B. D., States, D. J., Swaminathan, S., Karplus, M. CHARMM: a program for macromolecular energy, minimization, and dynamics calculation. J. Comp. Chem. 4:187–217, 1983.

[75] Lawrence, C. P., Skinner, J. L. Flexible tip4p model for molecular dynamics simulation of liquid water. Chem. Phys. Lett. 372:842–847, 2003.

[76] Jorgensen, W. L., Tirado-Rives, J. Potential energy functions for atomic-level simulations of water and organic and biomolecular systems. Proc. Natl. Acad. Sci. USA 102:6665–6670, 2005.

[77] Torda, A. E., Scheek, R. M., van Gunsteren, W. F. Time-dependent distance restraints in molecular dynamics simulations. Chem. Phys. Lett. 157:289–294, 1989.

[78] Hess, B., Scheek, R. M. Orientation restraints in molecular dynamics simulations using time and ensemble averaging. J. Magn. Reson. 164:19–27, 2003.

[79] Thole, B. T. Molecular polarizabilities with a modified dipole interaction. Chem. Phys. 59:341–345, 1981.

[80] Lamoureux, G., Roux, B. Modeling induced polarization with classical drude oscillators: Theory and molecular dynamics simulation algorithm. J. Phys. Chem. A. 119:3025–3039, 2004.

[81] Lamoureux, G., MacKerell, A. D., Roux, B. A simple polarizable model of water based on classical drude oscillators. J. Phys. Chem. A. 119:5185–5197, 2004.

[82] Noskov, S. Y., Lamoureux, G., Roux, B. Molecular dynamics study of hydration in ethanol-water mixtures using a polarizable force field. J. Phys. Chem. B. 109:6705–6713, 2005.

[83] van Gunsteren, W. F., Mark, A. E. Validation of molecular dynamics simulations. J. Chem. Phys. 108:6109–6116, 1998.

[84] Beutler, T. C., Mark, A. E., van Schaik, R. C., Greber, P. R., van Gunsteren, W. F. Avoiding singularities and numerical instabilities in free energy calculations based on molecular simulations. Chem. Phys. Lett. 222:529–539, 1994.

[85] Jorgensen, W. L., Tirado-Rives, J. The OPLS potential functions for proteins. energy minimizations for crystals of cyclic peptides and crambin. J. Am. Chem. Soc. 110:1657–1666, 1988.

[86] Berendsen, H. J. C., van Gunsteren, W. F. Molecular dynamics simulations: Techniques and approaches. In: Molecular Liquids-Dynamics and Interactions. et al., A. J. B. ed. NATO ASI C 135. Reidel Dordrecht, The Netherlands 1984 475–500.

[87] Allen, M. P., Tildesley, D. J. Computer Simulations of Liquids. Oxford: Oxford Science Publications. 1987.

[88] Ewald, P. P. Die Berechnung optischer und elektrostatischer Gitterpotentiale. Ann. Phys. 64:253–287, 1921.

[89] Hockney, R. W., Eastwood, J. W. Computer simulation using particles. New York: McGraw-Hill. 1981.

[90] Luty, B. A., Tironi, I. G., van Gunsteren, W. F. Lattice-sum methods for calculating electrostatic interactions in molecular simulations. J. Chem. Phys. 103:3014–3021, 1995.

[91] van Buuren, A. R., Marrink, S. J., Berendsen, H. J. C. A molecular dynamics study of the decane/water interface. J. Phys. Chem. 97:9206–9212, 1993.

[92] Mark, A. E., van Helden, S. P., Smith, P. E., Janssen, L. H. M., van Gunsteren, W. F. Convergence properties of free energy calculations: α-cyclodextrin complexes as a case study. J. Am. Chem. Soc. 116:6293–6302, 1994.

[93] Jorgensen, W. L., Chandrasekhar, J., Madura, J. D., Impey, R. W., Klein, M. L. Comparison of simple potential functions for simulating liquid water. J. Chem. Phys. 79:926–935, 1983.

[94] van Buuren, A. R., Berendsen, H. J. C. Molecular dynamics simulation of the stability of a 22 residue alpha-helix in water and 30 % trifluoroethanol. Biopolymers 33:1159–1166, 1993.

[95] Liu, H., Müller-Plathe, F., van Gunsteren, W. F. A force field for liquid dimethyl sulfoxide and liquid properties of liquid dimethyl sulfoxide calculated using molecular dynamics simulation. J. Am. Chem. Soc. 117:4363–4366, 1995.

[96] MacKerell, J. A. D., Feig, M., Brooks, C. L. 3rd. 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.

[97] MacKerell, A. D., Bashford, D., Bellott, Dunbrack, R. L., Evanseck, J. D., Field, M. J., Fischer, S., Gao, J., Guo, H., Ha, S., Joseph-McCarthy, D., Kuchnir, L., Kuczera, K., Lau, F. T. K., Mattos, C., Michnick, S., Ngo, T., Nguyen, D. T., Prodhom, B., Reiher, W. E., Roux, B., Schlenkrich, M., Smith, J. C., Stote, R., Straub, J., Watanabe, M., Wiorkiewicz-Kuczera, J., Yin, D., Karplus, M. All-atom empirical potential for molecular modeling and dynamics studies of proteins. J. Phys. Chem. B. 102(18):3586–3616, 1998.

[98] Feller, S. E., MacKerell, A. D. An improved empirical potential energy function for molecular simulations of phospholipids. J. Phys. Chem. B. 104(31):7510–7515, 2000.

[99] Foloppe, N., MacKerell, A. D. 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.

[100] Bjelkmar, P., Larsson, P., Cuendet, M. A., Hess, B., Lindahl, E. 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 Comp. 6:459–466, 2010.

[101] van der Spoel, D., van Buuren, A. R., Tieleman, D. P., Berendsen, H. J. C. Molecular dynamics simulations of peptides from BPTI: A closer look at amide-aromatic interactions. J. Biomol. NMR 8:229–238, 1996.

[102] Ryckaert, J. P., Bellemans, A. Molecular dynamics of liquid alkanes. Far. Disc. Chem. Soc. 66:95–106, 1978.

[103] on Biochemical Nomenclature, I.-I. C. Abrreviations and symbols for the description of the conformation of polypeptide chains. tentative rules (1969). Biochemistry 9:3471–3478, 1970.

[104] Mahoney, M. W., Jorgensen, W. L. 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.

[105] de Loof, H., Nilsson, L., Rigler, R. Molecular dynamics simulations of galanin in aqueous and nonaqueous solution. J. Am. Chem. Soc. 114:4028–4035, 1992.

[106] Jarzynski, C. Nonequilibrium equality for free energy differences. Phys. Rev. Lett. 78(14):2690 – 2693, 1997.

[107] O. Engin, M. S. A. Villa, Hess, B. Driving forces for adsorption of amphiphilic peptides to air-water interface. J. Phys. Chem. B.

[108] Feenstra, K. A., Hess, B., Berendsen, H. J. C. Improving efficiency of large time-scale molecular dynamics simulations of hydrogen-rich systems. J. Comp. Chem. 20:786–798, 1999.

[109] Hess, B. Determining the shear viscosity of model liquids from molecular dynamics. J. Chem. Phys. 116:209–217, 2002.

[110] Dewar, M. J. S. Development and status of mindo/3 and mndo. J. Mol. Struct. 100:41, 1983.

[111] Guest, M. F., Harrison, R. J., van Lenthe, J. H., van Corler, L. C. H. Computational chemistry on the fps-x64 scientific computers - experience on single-processor and multiprocessor systems. Theor. Chim. Act. 71:117, 1987.

[112] Frisch, M. J., Trucks, G. W., Schlegel, H. B., Scuseria, G. E., Robb, M. A., Cheeseman, J. R., Montgomery, J. A. Jr., Vreven, T., Kudin, K. N., Burant, J. C., Millam, J. M., Iyengar, S. S., Tomasi, J., Barone, V., Mennucci, B., Cossi, M., Scalmani, G., Rega, N., Petersson, G. A., Nakatsuji, H., Hada, M., Ehara, M., Toyota, K., Fukuda, R., Hasegawa, J., Ishida, M., Nakajima, T., Honda, Y., Kitao, O., Nakai, H., Klene, M., Li, X., Knox, J. E., Hratchian, H. P., Cross, J. B., Bakken, V., Adamo, C., Jaramillo, J., Gomperts, R., Stratmann, R. E., Yazyev, O., Austin, A. J., Cammi, R., Pomelli, C., Ochterski, J. W., Ayala, P. Y., Morokuma, K., Voth, G. A., Salvador, P., Dannenberg, J. J., Zakrzewski, V. G., Dapprich, S., Daniels, A. D., Strain, M. C., Farkas, O., Malick, D. K., Rabuck, A. D., Raghavachari, K., Foresman, J. B., Ortiz, J. V., Cui, Q., Baboul, A. G., Clifford, S., Cioslowski, J., Stefanov, B. B., Liu, G., Liashenko, A., Piskorz, P., Komaromi, I., Martin, R. L., Fox, D. J., Keith, T., Al-Laham, M. A., Peng, C. Y., Nanayakkara, A., Challacombe, M., Gill, P. M. W., Johnson, B., Chen, W., Wong, M. W., Gonzalez, C., Pople, J. A. Gaussian 03, Revision C.02. Gaussian, Inc., Wallingford, CT, 2004.

[113] Car, R., Parrinello, M. Unified approach for molecular dynamics and density-functional theory. Phys. Rev. Lett. 55:2471–2474, 1985.

[114] Field, M., Bash, P. A., Karplus, M. A combined quantum mechanical and molecular mechanical potential for molecular dynamics simulation. J. Comp. Chem. 11:700, 1990.

[115] Maseras, F., Morokuma, K. Imomm: A new ab initio + molecular mechanics geometry optimization scheme of equilibrium structures and transition states. J. Comp. Chem. 16:1170–1179, 1995.

[116] Svensson, M., Humbel, S., Froes, R. D. J., Matsubara, T., Sieber, S., Morokuma, K. 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.

[117] van der Spoel, D., Berendsen, H. J. C. Molecular dynamics simulations of Leu-enkephalin in water and DMSO. Biophys. J. 72:2032–2041, 1997.

[118] van der Spoel, D., van Maaren, P. J., Berendsen, H. J. C. A systematic study of water models for molecular simulation. J. Chem. Phys. 108:10220–10230, 1998.

[119] Smith, P. E., van Gunsteren, W. F. The viscosity of spc and spc/e water. Comp. Phys. Comm. 215:315–318, 1993.

[120] Balasubramanian, S., Mundy, C. J., Klein, M. L. Shear viscosity of polar fluids: Miolecular dynamics calculations of water. J. Chem. Phys. 105:11190–11195, 1996.

[121] van der Spoel, D., Vogel, H. J., Berendsen, H. J. C. Molecular dynamics simulations of N-terminal peptides from a nucleotide binding protein. PROTEINS: Struct. Funct. Gen. 24:450–466, 1996.

[122] Amadei, A., Linssen, A. B. M., Berendsen, H. J. C. Essential dynamics of proteins. PROTEINS: Struct. Funct. Gen. 17:412–425, 1993.

[123] Hess, B. Convergence of sampling in protein simulations. Phys. Rev. **E** 65:031910, 2002.

[124] Hess, B. Similarities between principal components of protein dynamics and random diffusion. Phys. Rev. **E** 62:8438–8448, 2000.

[125] Mu, Y., Nguyen, P. H., Stock, G. Energy landscape of a small peptide revelaed by dihedral angle principal component analysis. PROTEINS: Struct. Funct. Gen. 58:45–52, 2005.

[126] van der Spoel, D., van Maaren, P. J., Larsson, P., Timneanu, N. Thermodynamics of hydrogen bonding in hydrophilic and hydrophobic media. J. Phys. Chem. B. 110:4393–4398, 2006.

[127] Luzar, A., Chandler, D. Hydrogen-bond kinetics in liquid water. Nature 379:55–57, 1996.

[128] Luzar, A. Resolving the hydrogen bond dynamics conundrum. J. Chem. Phys. 113:10663–10675, 2000.

[129] Kabsch, W., Sander, C. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. Biopolymers 22:2577–2637, 1983.

[130] Williamson, M. P., Asakura, T. Empirical comparisons of models for chemical-shift calculation in proteins. J. Magn. Reson. Ser. B 101:63–71, 1993.

[131] Bekker, H., Berendsen, H. J. C., Dijkstra, E. J., Achterop, S., v. Drunen, R., v. d. Spoel, D., Sijbers, A., Keegstra, H., Reitsma, B., Renardus, M. K. R. Gromacs method of virial calculation using a single sum. In *Physics Computing 92* (Singapore, 1993). de Groot, R. A., Nadrchal, J., eds. . World Scientific.

[132] Berendsen, H. J. C., Grigera, J. R., Straatsma, T. P. The missing term in effective pair potentials. J. Phys. Chem. 91:6269–6271, 1987.

[133] Bekker, H. Ontwerp van een special-purpose computer voor moleculaire dynamica simulaties. Master's thesis. RuG. 1987.

[134] van Gunsteren, W. F., Berendsen, H. J. C. Molecular dynamics of simple systems. Practicum Handleiding voor MD Practicum Nijenborgh 4, 9747 AG, Groningen, The Netherlands 1994.

Index