CSCPB - Introduction to GROMOS

Information: Prof. Dr. W.F. van Gunsteren
Laboratory of Physical Chemistry, ETH-Zürich, 8092 Zürich, Switzerland
Tel: +41-44-6325501
Fax: +41-44-6321039
e-mail: wfvgn@igc.chem.ethz.ch

Note: This MD tutorial is still under development. Please report any errors, inconsistencies, obscurities or suggestions for improvements to zrinka@igc.phys.chem.ethz.ch.

Purpose: This molecular dynamics (MD) tutorial serves a number of objectives. These are to obtain experience with:

- computer simulation in general
- executing and understanding GROMOS
- interpreting the results of computer simulations

Version: SVN revision
December 5, 2007
W.F.v.G., Z.G., J.D., A-P.K., C.C., N.S.

Required knowledge: Some knowledge about the following is required:

- Computer programming languages: ability to read simple code (C++).
- Computer operating system language of the machine on which the tutorial will be carried out: UNIX.

Good books on MD are (Frenkel & Smit, 2002) and (Allen & Tildesley, 1987).

Contents

1 INTRODUCTION

1.1 The Lennard-Jones interaction

The Lennard-Jones interaction between two atoms at a distance r is given by (see also Figure 1)

$$V(r) = 4\varepsilon \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right) . \tag{1}$$

For a system of N atoms the (Lennard-Jones) potential energy is then

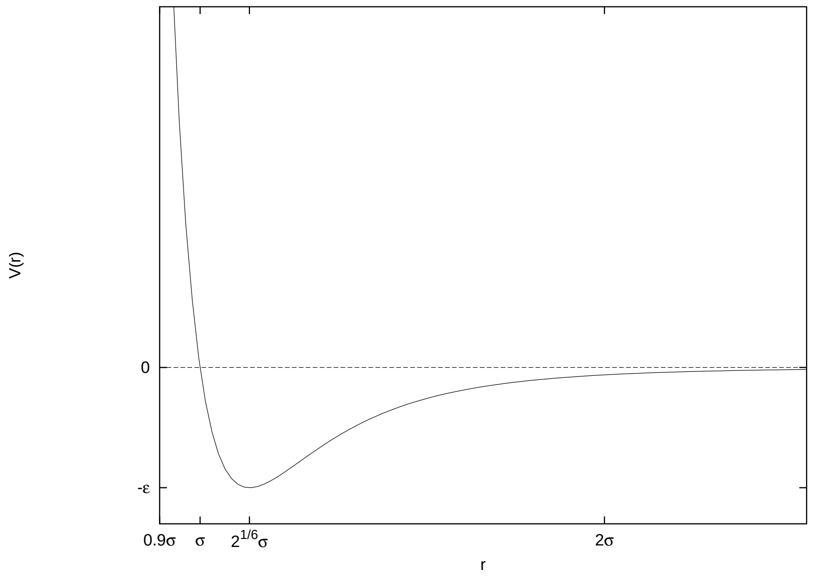


Figure 1: The form of the Lennard-Jones interaction

$$E_{pot} = V(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N) = \sum_{i=1}^{N-1} \sum_{j>i}^{N} V(r_{ij}) \tag{2}$$

where the vector $\mathbf{r}_{ij}$ is defined by the Cartesian position vectors $\mathbf{r}_i$ and $\mathbf{r}_j$ of atoms i and j,

$$\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j . \tag{3}$$

The distance r_{ij} between atoms i and j is

$$r_{ij} = |\mathbf{r}_{ij}| = \left(x_{ij}^2 + y_{ij}^2 + z_{ij}^2 \right)^{\frac{1}{2}} . \tag{4}$$

The partial derivative of $V(r_{ij})$ with respect to $\mathbf{r}_i$ is

$$\frac{\partial V(\mathbf{r}_{ij})}{\partial \mathbf{r}_i} = 4\varepsilon \left(-12 \left(\frac{\sigma}{r_{ij}} \right)^{12} + 6 \left(\frac{\sigma}{r_{ij}} \right)^{6} \right) \left(\frac{\mathbf{r}_{ij}}{r_{ij}^2} \right) . \tag{5}$$

The number of atom pairs in the summation in (2) can be reduced by applying a *cut-off radius* R_c beyond which no interactions $V(r_{ij})$ are computed. Due to the r^{-6} distance dependence of $V(r)$, rather short cut-off radii of $2.5 - 3.3\ \sigma$ can be used (Verlet, 1968).

1.2 Electrostatic interaction

The electrostatic interaction between two atoms i and j at a distance r is given by Coulomb's law

$$V(r) = \frac{q_i q_j}{4\pi\epsilon_0\epsilon_r} \frac{1}{r}, \tag{6}$$

where q_i and q_j are the charges of atoms i and j, respectively. The Coulomb interaction is inversely proportional to distance r. Due to this $\frac{1}{r}$ distance dependence, it is long ranged. The size of the Coulomb interaction at a typical cut-off distance of 0.8–1.0 nm is even for partial charges of size 0.1–0.4 e non-negligible. This would mean that non-bonded Coulomb interactions may only be neglected (cut-off) or approximated using a continuum reaction field beyond long distances, like 2.0–3.0 nm. Since the number of non-bonded neighbors grows proportionally to the third power of the cut-off radius, application of a large non-bonded interaction cut-off radius dramatically increases the required computing time.

1.3 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 that is to be simulated are put into a cubic, or more generally into any periodically space filling shaped box, which is surrounded by 26 translated images of itself in the case of rectangular boxes. When calculating the forces on an atom i in the central box, only the interaction with the nearest image of atom j is taken into account. The vector $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ connecting *nearest images* can be obtained from

$$\begin{aligned} x_{ij} &= x_{ij} - NINT\left(\frac{x_{ij}}{a}\right) a \\ y_{ij} &= y_{ij} - NINT\left(\frac{y_{ij}}{b}\right) b \qquad (7) \end{aligned}$$

$$z_{ij} \;=\; z_{ij} - NINT\left(\frac{z_{ij}}{c}\right) c\,,$$

where the lengths of the edges of the periodic box are denoted by a, b, and c, and the function $NINT(x)$ delivers the integer number that is nearest to x. When an atom leaves the central box at one side, it enters it with identical velocity at the opposite side at the translated image position. The atom i can be kept in the central box that lies in the positive quadrant with respect to an origin at $\mathbf{r}_0$, by applying

$$\begin{aligned} x_i &= x_i - NINT\left(\frac{x_i - x_0 - \frac{a}{2}}{a}\right) a \\ y_i &= y_i - NINT\left(\frac{y_i - y_0 - \frac{b}{2}}{b}\right) b \\ z_i &= z_i - NINT\left(\frac{z_i - z_0 - \frac{c}{2}}{c}\right) c \;\;. \end{aligned} \tag{8}$$

Application of periodic boundary conditions means that in fact a crystal is simulated. For a liquid or solution the periodicity is an artifact of the computation, so the effects of periodicity on the forces on the atoms should be not significant. This means that an atom should not simultaneously interact with another atom *and* its images.

Only interactions between nearest images should be evaluated, that is, the cut-off radius R_c for computation of the interaction should be smaller than half the smallest box edge,

$$R_c < \frac{1}{2} MIN(a, b, c)\,. \tag{9}$$

1.4 Newton's equations of motion

Newton's equations of motion for a system of N atoms with Cartesian position vectors $\mathbf{r}_i$ and masses m_i are

$$\frac{d^2\mathbf{r}_i(t)}{dt^2} = \frac{\mathbf{f}_i(t)}{m_i} \qquad\qquad i = 1, 2, \ldots, N \tag{10}$$

where the force $\mathbf{f}_i(t)$ on atom i is equal to the negative gradient of the potential energy function (2) with respect to the coordinates of atom i

$$\mathbf{f}_i(t) = -\frac{\partial}{\partial \mathbf{r}_i} V(\mathbf{r}_1(t), \mathbf{r}_2(t), \ldots, \mathbf{r}_N(t))\,. \tag{11}$$

These equations can be numerically integrated using small ($\ll$ than the shortest oscillation period in the system) MD time steps Δt producing a *trajectory* (atomic positions as a function of time t) of the system. Equations (10) and (11) conserve the total energy of the system

$$E_{tot} = E_{pot} + E_{kin} \tag{12}$$

and the total translational momentum of the system

$$\mathbf{p}_{tot} = \sum_{i=1}^{N} m_i \mathbf{v}_i \,, \tag{13}$$

where the total kinetic energy of the system is given by

$$E_{kin} = \sum_{i=1}^{N} \frac{1}{2} m_i \mathbf{v}_i^2 \,, \tag{14}$$

Monitoring the conservation of E_{tot} and $\mathbf{p}_{tot}$ is very useful in order to check the integrity of a MD program.

1.5 The leap-frog integration scheme

A simple algorithm for integration of (10) can be obtained by writing a Taylor expansion of the velocity $\mathbf{v}_i(t)$ at a time point t_n for $+\frac{\Delta t}{2}$ and one for $-\frac{\Delta t}{2}$

$$\mathbf{v}_i\left(t + \frac{\Delta t}{2}\right) = \mathbf{v}_i(t_n) + \left.\frac{d\mathbf{v}_i}{dt}\right|_{t_n} \frac{\Delta t}{2} + \left.\frac{d^2\mathbf{v}_i}{dt^2}\right|_{t_n} \frac{\left(\frac{\Delta t}{2}\right)^2}{2!} + \ldots \tag{15}$$

$$\mathbf{v}_i\left(t - \frac{\Delta t}{2}\right) = \mathbf{v}_i(t_n) - \left.\frac{d\mathbf{v}_i}{dt}\right|_{t_n} \frac{\Delta t}{2} + \left.\frac{d^2\mathbf{v}_i}{dt^2}\right|_{t_n} \frac{\left(\frac{\Delta t}{2}\right)^2}{2!} - \ldots \tag{16}$$

Subtracting (16) from (15), rearranging the terms, and using

$$\frac{d\mathbf{v}_i(t)}{dt} = \frac{\mathbf{f}_i(t)}{m_i} \tag{17}$$

one obtains

$$\mathbf{v}_i\left(t_n + \frac{\Delta t}{2}\right) = \mathbf{v}_i\left(t_n - \frac{\Delta t}{2}\right) + \frac{\mathbf{f}_i(t_n)}{m_i}\Delta t + \mathcal{O}\left(\Delta t^3\right) \,. \tag{18}$$

By an analogous procedure using a Taylor expansion for the position $r_i(t)$ at time $t_n + \frac{\Delta t}{2}$ (forward: $\frac{\Delta t}{2}$, backward: $-\frac{\Delta t}{2}$) one obtains

$$\mathbf{r}_i(t_n + \Delta t) = \mathbf{r}_i(t_n) + \mathbf{v}_i \left(t_n + \frac{\Delta t}{2}\right) \Delta t + \mathcal{O}\left(\Delta t^3\right) . \tag{19}$$

Equations (18) and (19) are called the *leap-frog scheme* for integration of the classical equations of motion (10).

1.6 Coupling to a temperature bath

When solving the classical equations of motion (10) and (11) the total energy E_{tot} as well as the volume V is a constant of motion, yielding a microcanonical ensemble. For various reasons this is not very convenient and many approaches have appeared in the literature to yield a type of dynamics in which temperature is the independent variable rather than a derived property. A simple way to control the temperature T of the system is to couple it to a heat or temperature bath of reference temperature T_0. The basic idea is to modify the equations of motion (10) such that the net result on the system is a first-order relaxation of temperature T to a given reference value T_0 (Berendsen et al., 1984),

$$\frac{dT(t)}{dt} = \frac{T_0 - T(t)}{\tau_T} . \tag{20}$$

The modified equations of motion are

$$\frac{d\mathbf{v}_i(t)}{dt} = \frac{\mathbf{f}_i(t)}{m_i} - \frac{c_v^{df}}{k_B \tau_T} \left(\frac{T_0}{T(t)} - 1\right) \mathbf{v}_i \, , \tag{21}$$

where the heat capacity per degree of freedom (number is N_{df}) of the system, c_v^{df}, is defined by

$$\Delta E_{tot} = N_{df} c_v^{df} \Delta T \, . \tag{22}$$

The temperature relaxation time or coupling time τ_T controls the strength of the coupling to the heat bath. This temperature coupling can be easily incorporated into the leap-frog scheme. The temperature can be obtained from the velocities by the relation

$$\frac{3N}{2} k_B T(t) = E_{kin} = \sum_{i=1}^{N} \frac{1}{2} m_i \mathbf{v}_i^2(t) \, , \tag{23}$$

where k_B is Boltzmann's constant. After applying (18) the obtained velocities $\mathbf{v}_i\left(t_n + \frac{\Delta t}{2}\right)$ are to be scaled by a factor

$$\lambda = \left(1 + \frac{2c_v^{df}}{k_B}\frac{\Delta t}{\tau_T}\left(\frac{T_0}{T\left(t - \frac{\Delta t}{2}\right)} - 1\right)\right)^{\frac{1}{2}} , \tag{24}$$

where the temperature at the previous time point $t_n - \frac{\Delta t}{2}$ is used since the temperature $T\left(t_n + \frac{\Delta t}{2}\right)$ is still to be calculated from the velocities $\mathbf{v}_i\left(t_n + \frac{\Delta t}{2}\right)$ obtained after multiplication by λ. The coupling constant τ_T can be arbitrarily chosen, but should exceed the fastest kinetic energy fluctuations, e.g. 10 time steps Δt, to ensure stability of the algorithm. The equations of motion (21) no longer conserve total energy E_{tot} or total momentum $\mathbf{p}_{tot}$ of the system, as when solving (10) and (11).

1.7 Pressure and the virial

The pressure of a system in equilibrium is given by

$$P = \frac{2}{3V}\left(\langle E_{kin}\rangle - \langle \Xi \rangle\right) , \tag{25}$$

where the volume of the system is denoted by V, an ensemble (or trajectory) average is denoted by $\langle \ldots \rangle$ and the virial Ξ is defined by (Hirschfelder et al., 1954)

$$\Xi = -\frac{1}{2}\sum_{i=1}^{N-1}\sum_{j>i}^{N}\mathbf{r}_{ij} \bullet \mathbf{f}_{ij} . \tag{26}$$

1.8 Center of mass motion

It is often useful to monitor the motion of the center of mass of a system since the translational momentum should be a constant of motion when Newton's equations of motion (10) and (11) are integrated. The position of the center of mass of the system is defined as

$$\mathbf{r}_{cm} = m_{tot}^{-1}\sum_{i=1}^{N} m_i \mathbf{r}_i \tag{27}$$

and the velocity as

$$\mathbf{v}_{cm} = m_{tot}^{-1}\sum_{i=1}^{N} m_i \mathbf{v}_i \tag{28}$$

and

$$m_{tot} = \sum_{i=1}^{N} m_i . \tag{29}$$

1.9 Gaussian or Maxwellian distributions

The atomic velocity $\mathbf{v}_i$ of a system in equilibrium will follow a *Maxwellian distribution* (McQuarrie, 1976), that is, the probability that the velocity lies between $\mathbf{v}_i$ and $\mathbf{v}_i + d\mathbf{v}_i$ is

$$p(\mathbf{v}_i)d\mathbf{v}_i = \left(\frac{2\pi k_B T}{m_i}\right)^{-\frac{3}{2}} e^{\frac{-m_i \mathbf{v}_i^2}{2k_B T}} d\mathbf{v}_i\,, \tag{30}$$

where k_B is Boltzmann's constant, T the temperature and m_i the mass of an atom. Mathematically this velocity distribution has the form of a product of three *Gaussian distributions*

$$p(x) = \left(2\pi\sigma^2\right)^{-\frac{1}{2}} e^{-\frac{(x-\langle x\rangle)^2}{2\sigma^2}}\,. \tag{31}$$

When starting a MD simulation the atomic velocities are often sampled from a Maxwellian distribution (30) at the desired temperature T. Technically, sampling from a Gaussian distribution can be performed by taking the sum of a series of random numbers taken from a uniform distribution. This procedure is based on the *central limit theorem* which states that in the limit of an infinite sum of independent stochastic variables obeying an arbitrary (so also uniform) distribution, the sum is a stochastic variable obeying a *Gaussian* distribution.

1.10 The radial distribution function $g(r)$

The *radial distribution function* $g(r)$ is defined such that the quantity $\rho g(r)dr$ is the "probability" of observing a second atom in the spherical shell between r and $r + dr$ given that there is an atom at the origin of $\mathbf{r}$. The quantity

$$\rho = \frac{N}{V} \tag{32}$$

is the number density. This "probability" is not normalized to unity, but we have instead

$$\int_0^\infty \rho g(r) 4\pi r^2 dr = N - 1\,. \tag{33}$$

In fact, (33) shows that $\rho g(r)4\pi r^2 dr$ is really the number of atoms between r and $r + dr$ about a central atom. The function $g(r)$ can be thought of as the factor that multiplies the bulk density ρ to give a local density $\rho(r) = \rho g(r)$ about some fixed atom. Of course $g(r) \to 0$ when $r \to 0$, since atoms can not completely overlap. When r becomes large, the central atom "sees" a uniform distribution, so when $r \to \infty$ then $g(r) \to 1$. The radial distribution function contains structural information on the liquid, and can be determined from X-ray or neutron diffraction experiments.

1.11 Units

Different sets of units are used in MD simulations. In general the use of *Standard International (SI)* units is recommended. In MD simulation of model systems, like Lennard-Jones liquids, it is often advantageous to work with dimensionless quantities *(reduced units)* and apply the appropriate scaling afterwards.

When choosing the SI system it is recommended to use the following basic units:

length:	r:	nm	$= 10^{-9}$ m
mass:	m:	u	= atomic mass unit
			= 1/12 of the mass of a ^{12}C atom
			$= 10^{-3}/N_{Av}$ kg
			$= 1.6605655 \cdot 10^{-27}$ kg
time:	t:	ps	$= 10^{-12}$ s
temperature:	T:	K	
charge:	q:	e	= elementary charge
			$= 1.6021892 \cdot 10^{-19}$ C

Consistent with these units are:

energy:	E:	kJ mol^{-1}	$= 0.2390$ kcal mol^{-1}
force:	f:	kJ mol^{-1} nm^{-1}	
pressure:	P:	kJ mol^{-1} nm^{-3}	$= 10^{30}/N_{Av}$ Pa
			$= 1.6605655$ MPa
			$= 16.6057$ Bar
			$= 16.3885$ atm
velocity:	v:	nm ps^{-1}	

Here we have used:

N_{Av}	= Avogadro's number	$= 6.022045 \cdot 10^{23}$ mol^{-1}
R	= gas constant	$= 8.31441 \cdot 10^{-3}$ kJ mol^{-1} K^{-1}
k_B	= Boltzmann's constant	$= R/N_{Av} = 1.380662 \cdot 10^{-26}$ kJ K^{-1}

2 INSTALLATION OF GROMOS

GROMOS consists of two packages, one for simulation called GROMOSXX and one for simulation analysis and setup called GROMOS++. Both of these packages are obtained as C++ source code and have to be compiled before usage.

2.1 System Requirements

GROMOS can be compiled on almost any operating system compatible with the POSIX standard. Make sure that your system is powerful enough, that you have sufficient disk space, and that all required libraries are installed.

The hardware requirements are rather high, but nowadays most of the analysis can be carried out on personal computers or even laptops.

Architecture Intel or AMD x86, SUN SPARC or IBM PowerPC CPU

Memory This depends on the simulation you are running and the analysis you want to carry out. For most simulations and analyses you need just a few 100 MB of RAM but there are exceptions.

Diskspace This also depends on the simulation. Typical are up to 5 GB for a big simulation. For analysis, up to 1 GB is sufficient

Please have a look at the software requirements because these are usually not installed on an out-of-box operating system.

Operating System POSIX compatible UNIX like Linux

Build-Essentials make, binutils (usually installed by default)

Compiler In principle it is possible to use any ANSI C++ compiler but it is recommended to use the GNU Compiler Collection (GCC) `g++`.

Libraries There are a couple of libraries that you have to install on your system. Make sure you also install the header files (-devel, -dev packages). The libraries you need:

- zlib compression library
- gsl GNU Scientific Library (see Appendix)
- for GROMOSXX: socket, nsl

2.2 Getting and Installing GROMOSXX

GROMOSXX is a batch program for molecular simulation jobs. Because its execution time is critical you have to compile it on the machine you would like to use it to make sure that the maximum performance is obtained. Open a command line shell and find out where your home is. Then create a working and an installation directory.

```
~> pwd
/home/nschmid
~> mkdir temp
~> mkdir gromosxx
```

💡 *Hint:* Change the path `/home/nschmid` to your home (the result of the pwd command).

In your web browser, open the http://www.igc.ethz.ch/education/advanced/cscbp/exe page and download the gromosxx.tar.gz into your working directory. Change into the `temp` directory and unpack GROMOSXX.

```
~> cd temp
~/temp> gunzip gromosXX-0.2.3.tar.gz
~/temp> tar xf gromosXX-0.2.3.tar
~/temp> cd gromosXX-0.2.3
```

In this directory now lies the source code of GROMOSXX which is ready for compilation. The configuration is a bit tricky, because there are many options which are explained below:

- You can specify the installation path using the `--prefix` directive.
- Make sure you disable debugging with `--disable-debug`. GROMOSXX will run *much* faster without debug code.
- If the GNU Scientific Library (gsl) was installed by the superuser `configure` will find it. If it is installed locally in your home you must specify this using the `--with-gsl` directive.
- On clusters where setup of library paths is not guaranteed, it is a good idea to link it statically. `--disable-shared` will disable the shared library and create statically linked executables.

💡 *Hint:* See the Appendix to learn how to compile a parallel version of GROMOS XX which runs on clusters and multicore CPUs.

Now you know what the options mean and you are ready to configure and run the compilation of GROMOSXX.

```
~/temp/gromosXX-0.2.3> ./configure --prefix=/home/nschmid/gromosxx \
                          --with-gsl=/home/nschmid/gsl \
                          --disable-debug \
                          --disable-shared
~/temp/gromosXX-0.2.3> make
~/temp/gromosXX-0.2.3> make install
~/temp/gromosXX-0.2.3> make test
~/temp/gromosXX-0.2.3> touch doc/doxygen.conf.in
~/temp/gromosXX-0.2.3> make doc
~/temp/gromosXX-0.2.3> cp -r doc /home/nschmid/gromosxx/
```

Hint: On multicore CPU machines add the `-j3` flag to `make` to boost the compilation.

GROMOSXX is now installed in your home. You can test it by typing

```
~/temp/gromosxx> /home/nschmid/gromosxx/bin/md
```

If it prints the usage everything is fine.
Clean up the working directory because the static builts need lots of diskspace and are not needed anymore.

```
~/temp/gromosXX-0.2.3> cd ..
~/temp> rm -rf gromosXX-0.2.3*
```

Warning: *Don't* use GROMOSXX version from other people. It is likely that it was modified.

2.3 Getting and Installing GROMOS++

GROMOS++ is a collection of command line programs needed to setup a simulation or to analyze the results of a simulation. In order to use these programs you have to compile and install them on your machine. Open a command line shell and find out where your home is.

```
~> pwd
/home/nschmid
```

Hint: Change the path `/home/nschmid` to your home (the result of the pwd command).

Go the the http://www.igc.ethz.ch/education/advanced/cscbp/exe website and download the gromos++.tar.gz file in home directory. Unpack GROMOS++.

```
~> gunzip gromos++-0.2.4.tar.gz
~> tar xf gromos++-0.2.4.tar
~> cd gromos++-0.2.4
```

You are now in the GROMOS++ source directory and can start to configure and make GROMOS++. Tell configure where it has to look for the GNU Scientific Library (gsl) using the `--with-gsl` directive.

Hint: Usually the GSL is installed in /usr/local and configure will find it without telling, but if you have installed the GSL in your home you have to tell configure using the `--with-gsl` directive.

```
~/gromos++-0.2.4> ./configure --with-gsl=/home/nschmid/gsl
~/gromos++-0.2.4> make
~/gromos++-0.2.4> make install
```

Hint: On multicore CPU machines add the `-j3` flag to `make` to boost the compilation.

Generating the documentation If doxygen is installed on your machine you can generate documentation directly from the source code. Go to the gromos++ directory and type

```
~/gromos++-0.2.4> doxygen doc/gromos++.doxy
```

In the `doc` directory you will find html documentation. Open a web browser and open `file:///home/<username>/<path to gromos++>/gromos++/doc/html/index.html`. Under `available` you find the documentation of the various gromos++ programs. After the successful installation you should clean up the working directory.

```
~/gromos++-0.2.4> make distclean
```

Adding it to the path In order to use the programs without specifying the full path you can add them to your PATH. Add the following two lines to your `~/.bashrc`:

```
export PATH="${PATH}:/home/nschmid/gromos++-0.2.4/bin"
export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:/home/nschmid/gromos++-0.2.4/lib"
```

What is installed

- the program binaries are in `bin/`.
- in the `include/` and `lib/` subdirectories are the files needed for programing with GROMOS++
- `share/` is home of useful files (called libraries) and examples.

General usage Check which arguments a program expects by typing its name without any arguments. For example `rgyr` uses a couple of arguments

```
~> rgyr
# Usage:
#
# rgyr
       @topo          <topology>
       @pbc           <boundary type> [<gathermethod>]
       @time          <time and dt>
       @atoms         <atom specifier for the atoms to consider>
       [@massweighted (use massweighted formula)]
       @traj          <trajectory files>
```

You can pass the arguments to the program via the command line and redirect its output to a file

```
~> rgyr @topo peptide.topo @pbc r @traj peptide.trj.gz > rgyr.out
```

because argument lists can be very long it is more convenient to put them into a file (here `rgyr.inp`).

```
~> rgyr >& rgyr.inp
```

Now you can edit the file with a text editor, specify all the arguments and pass it to the program using the `@f` directive.

```
~> rgyr @f rgyr.inp > rgyr.out
```

The analysis arguments are now read from the file you specified.

Warning: Always use the `@f` directive because otherwise you will forget about the arguments you used.

---

3 A PRACTICAL EXERCISE

3.1 Building a topology

When a simulation study of a particular system or process is to be carried out, a number of choices have to be made with respect to the set-up of the simulation. The first task is to generate a molecular topology file containing the topological and force field data concerning the molecular system under consideration. Specifying a complete molecular topology for a large molecule however, is a tedious task. Therefore in GROMOS a molecular topology is generated from molecular topology building blocks which carry the topological information of molecules like amino acid residues, nucleotides, etc. (see *The Gromos96 Manual and User Guide, Chapter IV*). The molecular topology building blocks can be linked together into a complete molecular topology file using the GROMOS++ program `make_top`. Many molecular topology building blocks are available in the molecular topology building block files (`mtb*.dat`), which are standard data files that come together with the GROMOS package. In case the needed molecular topology building blocks are not part of the standard distribution, they must be constructed. Constructing a new topology building block may require estimation of additional force-field parameters, which have to be added to the interaction function parameter file (`ifp*.dat`). When generating a molecular topology for the system of interest one should also address the question of the protonation state of the molecular groups according to the pH and the question of the solvent and counter ions that need to be included in the simulation box. In case of a molecular complex, e.g. a DNA-protein complex, the two separately generated molecular topologies can be merged using the GROMOS++ program `com_top`.

3.1.1 Creating the topology for the penta-peptide

In this section you should build a molecular topology of a linear charged penta-peptide (Figure 2) with water as a solvent, including 2 Cl^- counter ions. All input files are in `tutorial_files.tar.gz` and can be downloaded from
http://www.igc.ethz.ch/education/advanced/cscbp/exe .
Download the file (`tutorial_files.tar.gz`) into your `home` and untar and gunzip it (look at page 13).

programs You need the following programs from the GROMOS simulation package

```
GROMOS++: make_top com_top check_topo
```

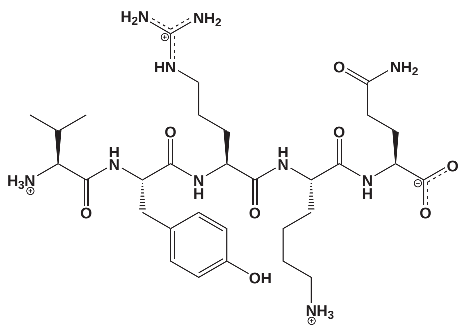


Figure 2: The structure of the penta-peptide.

input files You need the following input files

```
mtb53a6.dat ifp53a6.dat maketop_peptide.inp maketop_Cl.inp comtop_peptide_2Cl.inp
```

output files The procedure will create the following output files

```
peptide_53a6.dat Cl_53a6.dat peptide_2Cl_53a6.dat
```

1 Go into the subdirectory `topo` in directory `peptide` in your home.

```
~> cd ~/peptide/topo
```

You will build the molecular topology file of the linear charged penta-peptide VAL-TYR-ARG-LYSH-GLN by using the GROMOS++ program `make_top`. The input file `maketop_peptide.inp` is already prepared and contains the following data: under the flag `@build` the molecular topology building block file is specified. The flag `@param` specifies the interaction function parameter file. Under the flag `@seq` the sequence of the building blocks for the amino acid residues, including the amino and carboxy terminus is specified. Notice that both termini are charged.

Hint: The ARG and LYSH building blocks correspond to the protonated amino acids. Always check carefully the name of the charged amino acids, to make sure you have the correctly protonated species.

The flag `@solv` specifies the solvent. The molecular topology file for the penta-peptide, `peptide_53a6.dat`, with water as a solvent can then be generated as

```
~/peptide/topo> make_top @f maketop_peptide.inp > peptide_53a6.dat
```

2 As next is to build a molecular topology file for a chloride ion using the GROMOS++ program `make_top` and the input file `maketop_Cl.inp`

```
~/peptide/topo> make_top @f maketop_Cl.inp > Cl_53a6.dat
```

3 Now we will combine the generated molecular topology files (`peptide_53a6.dat` and `Cl_53a6.dat`) into the molecular topology file `peptide_2Cl_53a6.dat` using the GROMOS++ program `com_top`. The input file `comtop_peptide_2Cl.inp` is already prepared and it contains the following data: under the flag `@topo` the molecular topology files that you would like to combine together are specified. Since two chloride ions are needed to neutralize the charge of the penta-peptide, the molecular topology file of the chloride ion is specified as `2:Cl_53a6.dat`. The flags `@param` and `@solv` specify from which molecular topology file the parameters for the solute and solvent should be taken. Since we use the same molecular topology building block file in both cases this is not important for us and both numbers are set to `1`. To run `com_top` type:

```
~/peptide/topo> com_top @f comtop_peptide_2Cl.inp > peptide_2Cl_53a6.dat
```

The file `peptide_2Cl_53a6.dat` contains the complete molecular topology of the linear charged penta-peptide with water as a solvent, including $2Cl^-$ counter ions. Have a look at it and check whether it makes sense!

4 To be sure that your topology is correct, you should always check your topology using the `check_topo` program

```
~/peptide/topo> check_topo @topo peptide_53a6.dat > checktopo.out
```

Hint: Use `check_topo` with the additional arguments `@build` and `@param` for a more careful check of your topology against the force field.

Warning: Be aware that although `check_topo` can find many mistakes in your topology it is always important to carefully check your topology and or building block files manually. This is especially important if you had to modify an existing or create a new building block.

---

p

3.2 Generating atomic Cartesian coordinates for the solute, solvent and counter ions

Coordinates for biomolecules are often available from X-ray or NMR experiments and can be obtained in protein data bank (pdb) format, which can be converted to GROMOS format using the GROMOS++ program `pdb2g96`. However, the conversion is not always straightforward since the names and numbering of the atoms in the pdb format usually do not match the GROMOS format. Moreover, the coordinates for hydrogen atoms are not present in the pdb files (when the structure was determined with X-ray experiments) and have to be generated using the GROMOS++ program `gch`. When the structure is determined with the NMR experiment, the pdb structure contains more hydrogen atoms than it is needed in GROMOS because in GROMOS only polar and aromatic hydrogens are the explicit hydrogens. Aliphatic hydrogens are non-existing due to the fact that we use the United Atoms for aliphatic atoms. If no atomic coordinates for the solute are available from experimental data, the coordinates have to be generated using modeling programs such as the INSIGHTII software package (Accelrys Inc., San Diego, CA). Very often parts of the structure (e.g. flexible loops) are not resolved in the experiment and therefore not available in the pdb and have to be modeled. When a simulation of a solute in solution is to be carried out, a (periodic) box, cubic (c), rectangular (r) or truncated octahedron (t), should be put around the solute and filled with solvent molecules up to the required density. The solvent coordinates can e.g. be generated using the GROMOS++ program `sim_box`. The generated box should be sufficiently large to allow the use of a reasonable non-bonded interaction cut-off radius. Putting the solute in a box of solvent using the `sim_box` program will result in several high-energy atom-atom contacts at the solute-solvent interface and at the box edges. In order to relax the generated configuration the solvent configuration should be energy minimized while positionally restraining the solute. Counter-ion atomic coordinates can then be generated using the GROMOS++ program `ion`, which can replace a number of solvent molecules by ions.

3.2.1 Generating atomic Cartesian coordinates for the linear charged penta-peptide in water including 2 Cl^- counter ions

programs You need the following programs from the GROMOS simulation package

```
GROMOS++: pdb2g96 gch frameout sim_box ion
GROMOSXX: md
```

input files You need the following input files

```
peptide.pdb pdb2g96_peptide.inp gch_peptide.inp frameout_peptide.inp sim_box_peptide.inp
spc.g96 iem_peptide.dat jem_peptide.sh iem_solvent.dat jem_solvent.sh ion_peptide.inp
```

output files The procedure will create the following output files

```
pdb2g96_peptide.g96 gch_peptide.g96 gch_peptide.pdb oempeptide.out peptide.g96
sim_box_peptide.g96 oemsolvent.out peptide_h2o.g96 peptide_2Cl_h2o.g96 peptide_2Cl_h2o.posres
```

1 Go into a subdirectory `coord`. Open the file `peptide.pdb` and check if the atom names match the names in the molecular topology file `peptide_2Cl_45a4.dat`. If not, atom names in the coordinate file `peptide.pdb` should be adapted.

2 In the pdb file `peptide.pdb` the coordinates for hydrogen atoms are not given and have to be generated. Convert the pdb file `peptide.pdb` into the GROMOS format using the GROMOS++ program `pdb2g96`. The hydrogen atoms will be added (or ignored) to the coordinate file according to the topological requirements. The input flags of the pdb2g96 input file are self explanatory.

```
~/peptide/coord> pdb2g96 @f pdb2g96_peptide.inp > pdb2g96_peptide.g96
```

⚠ *Warning:* When converting coordinate files from the pdb database to GROMOS format many difficulties can emerge. If you encounter problems using the `pdb2g96` program, have a look at the `doxygen` documentation (see page 15). There you can find further documentation on the advanced usage of this program especially the use of a library that matches residue and atom names might be useful in many cases. `pdb2g96.lib` which you can find in directory is an example of the pdb library file.

---

3 Have a look at the `pdb2g96_peptide.g96` file. You will notice that the hydrogen atoms have been added to the coordinate file with the Cartesian coordinates being set to `0.00`. In order to generate the correct coordinates for the hydrogen atoms run the GROMOS++ program `gch`. It will generate the coordinates for hydrogen atoms by geometric means using the information from the molecular topology file. Therefore, the molecular topology file, `peptide_2Cl_45a4.dat`, and the coordinate file, `pdb2g96_peptide.g96`, have to be specified in the `gch` input file. The flag `@tol` sets the tolerance that is used for keeping the coordinates of hydrogens that are already present in the coordinate file. To run `gch` type:

```
~/peptide/coord> gch @f gch_peptide.inp > gch_peptide.g96
```

4 Using the GROMOS++ program `frameout` you can convert the coordinate file `gch_peptide.g96` back to the pdb format and look at the structure using the molecular visualization program VMD (Visual Molecular Dynamics).

```
~/peptide/coord> frameout @f frameout_peptide.inp
~/peptide/coord> mv FRAME_00001.pdb gch_peptide.pdb
~/peptide/coord> vmd gch_peptide.pdb
```

5 Before putting the penta-peptide in a box of solvent, its configuration has to be relaxed by energy minimization. Go into a subdirectory called `min`.

The GROMOSXX input file `iem_peptide.dat` contains following blocks.

```
TITLE
steepest descent energy minimization of the peptide in vacuum.
END
```

In the `TITLE` block you specify what is done with following input file.

```
MINIMISE
#    NTEM   NCYC   DELE   DX0    DXM
        1     20     0.1  0.01   0.05
END
```

The existence of `MINIMISE` block means that the GROMOSXX MD program will perform an energy minimisation run. The `NTERM` indicates which minimisation method is in use. With `NTERM` = 1 we indicate that the steepest descent is used. The `NCYC` gives the number of cycles. With `DELE` is given the energy threshold (the difference in energy between two energy minimisation steps) for stopping the minimisation proces. The initial step size and maximum step size is given in `DX0` and `DXM`, respectively. (see also page `II-110` of the *GROMOS Manual and User's Guide*)

```
SYSTEM
#     NPM     NSM
        1       0
END
```

In the `SYSTEM` block you specify the number of solute (`NPM`) and solvent (`NSM`) molecules. You only have one solute molecule, `NPM` = 1 and no solvent molecules `NSM` = 0 because you still did not add any solvent (in vacuo). Otherwise you would have to tell GROMOSXX how many solvent molecules you are using.

```
START
#     NTX    INIT     IG    TEMPI     HEAT  NTX0  BOLTZ
        1       1  210185    300.0  0.00000    1   8.31441E-3
END
```

For the energy minimisation the START block is only needed because of the Boltzmann constant. This block will be explained in more detail in the following chapter (3.3).

```
STEP
#   NSTLIM        T       DT
      2000      0.0    0.002
END
```

In the STEP block you specify how many energy steps (NSTLIM) minimisation you will perform.

```
BOUNDARY
#     NTB   BOX(1)   BOX(2)   BOX(3)     BETA  NRDBOX
        0      0.0      0.0      0.0     90.0      1
END
```

In the BOUNDARY block you specify which periodic boundary conditions (pbc) you are going to use in the energy minimization procedure. NTB $= 0$ defines vacuum pbc. To indicate the truncated octahedron (t) pbc, NTB is set to < 0, and for rectangular (r) pbc NTB is > 0. With the BOX[1..3] you indicate the size of the simulation box. Here you have for BOX[1..3] 0.00 because you did not make the box yet. The BETA indicates the angle between x- and z-axes in degrees (for rectangular and truncated octahedron BETA $= 90$). The NRDBOX flag controls the reading of the BOX dimensions. If NRDBOX $= 0$ the BOX dimensions are read from the BOUNDARY block. In the case of NRDBOX $= 1$ the BOX dimensions is read from the coordinate file (p. II-101).

```
SUBMOLECULES
#    NSPM  NSP(1.. NSPM)
     1     71
END
```

In the SUBMOLECULES block (p. II-129) you tell GROMOSXX the sizes of the different molecules your solute topology consists of. The first integer is the number of molecules. Here, we have only one: the peptide. The following number is the number of the last atom of that molecule, which in this case is atom 71.

```
PRINT
#NTPR: print out energies, etc. every NTPR steps
#NTPL: print out C.O.M motion and total energy fitting every NTPL steps
#NTPP: =1 perform dihedral angle transition monitoring
#    NTPR     NTPL     NTPP
       10        0        0
END
```

With PRINT block you specify how often (for every NTPR step) you are writing out the energies to the output file.

```
SHAKE
#     NTC      TOL
      3    0.00010
END
```

The vibration of bonds happens at very high frequencies ($hv >> k_BT$). Therefore, these vibrations are of quantum-mechanical nature. Constraining the bonds is probably a better approximation than treating them as classical harmonic oscillators. Constraining all bonds (NTC=3) allows us to use a rather large time step of 2 fs. The constraints are imposed by the SHAKE algorithm (p. II-87).

```
FORCE
#     NTF array
# bonds   angles   imp.    dihe    charge nonbonded
# H       H        H       H
  0  0     1  1     1  1    1  1    1  1
# NEGR   NRE(1)   NRE(2)   ...     NRE(NEGR)
    1     71
END
```

In the FORCE block (p. II-143) you tell GROMOSXX which terms it should use for the force calculation. For angles, improper dihedrals, dihedrals and the non-bonded interactions we use the normal terms of the GROMOS force field. Because we are using bond length constraints and the SHAKE algorithm, we have to switch off (0) the bond-stretching term.

In the last line of this block, the energy groups are defined. In general, we define the energy group(s) for every molecule, and one for all the solvent molecules. The first integer is the number of energy groups we want to use (in present case we only have one energy group). The following numbers point to the last atom of the energy group. By defining these energy groups we tell GROMOSXX to calculate the energies between these groups which can be very useful.

⚠ *Warning:* Always think very carefully about the definition of energy groups. A missing interesting energy-group will result in rerunning the simulation.

```
PLIST
#    NTNB     NSNB    RCUTP    RCUTL
        1        5      0.8      1.4
END
```

In the PLIST block you specify with NTNB = 1, that a pairlist is constructed before the first step. The cut-off used in the pairlist construct is given with RCUTP and for GROMOS is usually 0.8. The cut-off used in the longrange electrostatic interactions is given with RCUTL and for GROMOS is usually 1.4. The pairlist is generated every 5th (NSNB) step (p. II-52).

```
LONGRANGE
# EPSRF    APPAK     RCRF
       1              0.0       1.4
END
```

The longrange electrostatic interactions are truncated after a certain cutoff (RCUTL). Beyond the reaction field cut-off radius (RCRF) the electrostatic interactions are replaced by a static reaction field with a dielectric permittivity of EPSRF (p. II-53). RCRF and RCUTL should be identical. Because we are doing the energy minimisation in vacuo EPSRF is set to 1.

In order to run the GROMOSXX program, a shell script needs to be prepared. Open the the shell script jem_peptide.sh and adapt the paths and the names of the files according to your system. The energy minimization of the solute in vacuo is very fast and you can run it interactively by typing:

```
~/peptide/min> ./jem_peptide.sh
```

Once the energy minimization is finished, the file with the minimized coordinates, peptide.g96, and the general output file, oempeptide.out, that reports the progress of the minimization, will be written out. Have a look at both files using command less and check if the minimization has finished successfully. Using the GROMOS++ program frameout you can again convert the coordinate file peptide.g96 into pdb format and check the new configuration using the VMD program.

6 Put the energy minimized penta-peptide in a box of solvent using the GROMOS++ program sim_box that can solvate the solute in a pre-equilibrated box of solvent molecules. Go to a subdirectory box. In the input file for the program sim_box you have to specify the following input arguments: the molecular topology file under the arguments @topo, the resulting box shape under the arguments @pbc (r-rectangular, t-truncated octahedron), the coordinate file of the solute under the arguments @pos, the coordinate file of the pre-equilibrated box of solvent molecules under the arguments @solvent, minimum solute-to-wall distance under the arguments @minwall, and the minimum solute-to-solvent distance under the arguments @minsol. If you are using a truncated octahedron box, @pbc t, it is recomended to use an extra arguments, @rotate. With this extra arguments the solute can be rotated (before solvating) such that the largest distance between any two solute atoms is directed along the z-axes, and the largest atom-atom distance in the xy-plane lies in the y-direction. The `@rotate` arguments is also used if you have a very elongated molecule to ensure that the minimal distance between your molecule and any wall of your box. An input file `sim_box_peptide.inp` is already prepared. To put the solvent box around the penta-peptide type:

```
~/peptide/box> sim_box @f sim_box_peptide.inp > sim_box_peptide.g96
```

7 In order to relax the unfavorable atom-atom contacts between the solute and the solvent, energy minimization of the solvent should be performed while keeping the solute positionally restrained. In order to do that an additional file, in which the coordinates that need to be restrained are specified, has to be generated from the coordinate file `sim_box_peptide.g96`. Copy the coordinate file `sim_box_peptide.g96` to `sim_box_peptide.posres` open it with a text editor and delete all the coordinates of the solvent such that only the coordinates of the atoms that you want to restrain are left. Then change the keyword `POSITION` at the beginning of this new file to `POSRES`. When all the changes are applied the file should look like this:

```
TITLE
positional restraining of solute atoms.
END
POSRES
    1 VAL   H1        1    2.649927067   2.863070998   0.106762194
    1 VAL   H2        2    2.670227276   2.990499217   0.014952966
...
    5 GLN   O1       72    2.804919660   0.052110834   0.242091806
    5 GLN   O2       73    0.272876445   2.719416563   0.472032607
END
```

Hint: When reading in a coordinate file (with a `POSITION` block) the first 24 characters are ignored. However, in a position restraints file you have to be careful that each line of the `POSRES` block has seven columns as the fourth to seventh column is read in by the program.

The GROMOSXX input files for minimization of the solvent around the penta-peptide, `iem_solvent.dat`, and the script to run the minimization, `jem_solvent.sh`, are already prepared. Open the `iem_solvent.dat` and compare it with `iem_peptide.dat`. You will notice that in `iem_solvent.dat` is one input block more, the `POSREST` block.

```
POSREST
#    values for NTR
#    0: no position re(con)straining
#    1: use CHO
#    2: use CHO/ ATOMIC B-FACTORS
#    3: position constraining
#     NTR      CHO   NRDRX
      1      25000      1
END
```

With this block you are specifying that you want to restrain the positions, in this case the positions of your solute molecule. The restraining is achieved by a harmonic term with a force constant `CHO`. The `NRDRX` indicates that the restained coordinates are read from a separate file, which is in our case `sim_box_peptide.posres`.

Hint: Notice and understand the differences in SYSTEM, SUBMOLECULES, BOUNDARY and FORCE block between `iem_peptide.dat` and `iem_solvent.dat`.

Have a look at `jem_solvent.sh` and put in the correct paths to your files. Run the energy minimization of the solvent interactively by typing

```
~/peptide/min> ./jem_solvent.sh
```

Once the minimization is finished, the new coordinate file, `peptide_h2o.g96`, and the general output file `oemsolvent.out` will be written out.

8 In the next step, the two chloride ions should be added to the simulation box. Go to a subdirectory `ion`. The two chloride ions are added to the simulation box by using the GROMOS++ program `ion` such that they replace the water molecules which have the highest electrostatic potential. You can run `ion` by typing

```
~/peptide/coord> ion @f ion_peptide.inp > peptide_2Cl_h2o.g96
```

Convert the file `peptide_2Cl_h2o.g96` into the pdb format using the GROMOS++ program `frameout` and check where the chloride ions have been placed. In Figure 3 you can see how it should look like.

3.3 Setup and production run of the penta-peptide

programs You need the following programs from the GROMOS simulation package

```
GROMOS++: mk_script ene_ana
GROMOSXX: md
```

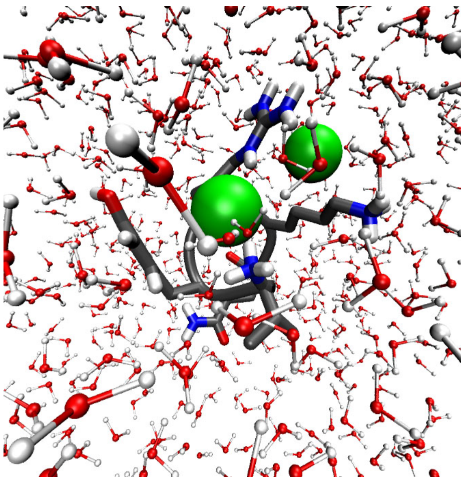


Figure 3: The peptide and the two counter ions in a box of water.

input files You need the following input files

```
eq_mkscript.inp equilibration.inp equilibration.jobs mkscript.lib ene_ana.inp
ene_ana_XX.lib md_mkscript.inp md.inp
```

output files The procedure will create the following output files

```
jeq_peptide_*.sh eq_peptide_*.inp eq_peptide_*.out eq_peptide_*.gsf
eq_peptide_*.ene.gz eq_peptide_*.trj.gz totkin.dat jmd_peptide_*.sh md_peptide_*.inp
md_peptide_*.out md_peptide_*.gsf md_peptide_*.trj.gz md_peptide_*.ene.gz
```

3.3.1 Thermalization and equilibration

In the previous steps you have generated the topology and the initial coordinates of your system. At this point, you have to generate the initial velocities. In the process of thermalization and equilibration, initial velocities are generated from a Maxwell-Boltzmann distribution at a certain temperature and the system is slowly heated up to the final production run temperature. The atoms of the solute are positionally restrained and these restraints are loosened while heating up. With the help of these restraints you make sure, that the random initial velocities do not disrupt the initial conformation.

You already know a couple of things about job scripts. Because the setup of a job script can be a labor-intensive undertaking, there is a little but powerful helper called `mk_script`. This GROMOS++ program is able to automatically generate a job script from a given input file and a series of arguments.

1 Before we have a detailed look at `mk_script`, let's see which GROMOSXX input file we need for the thermalization and equilibration period. Go to a subdirectory `eq`. Open the `equilibration.inp` file. This file contains a series of input blocks some of which you have already seen at the energy minimization step. Here only new or changed input blocks are explained.

```
START
# values for INIT -- starting procedures in RUNMD.f
#  INIT
#    shake X
#           shake V
# centre of mass motion removal if NTCM=1
#    1   yes    yes   yes
#    2    no    yes   yes
#    3    no     no   yes
#    4    no     no    no
#     NTX    INIT     IG    TEMPI     HEAT  NTXO  BOLTZ
        2       4  145117       0   0.00000    1   8.31441E-3
END
```

The `START` block is one of the more complicated input blocks that exist in GROMOS. The `NTX` tells GROMOS whether it should generate the initial velocities or read them from the configuration file. `INIT` is used to restore bond length constraints (SHAKE) and for removal of centre of mass motion. `IG` is the random number seed and `TEMPI` the initial temperature used to generate the Maxwell-Boltzmann distribution for generation of initial velocities. Finally, the Boltzmann constant can be modified using the `BOLTZ` parameter (see also *PROMD Quick Reference Sheet*).

💡 *Hint:* Many of the values in the `START` block are modified by `mk_script`, so don't worry if you don't understand everything now.

```
STEP
#   NSTLIM       T       DT
     10000      0.0    0.002
END
```

In the `STEP` block you specify how many steps you want to simulate (`NSTLIM`), at what time your simulation starts (`T`) and how big the integration time step (`DT`) is.

```
SUBMOLECULES
#    NSPM  NSP(1.. NSPM)
    3   71   72  73
END
```

In the `SUBMOLECULES` block you tell GROMOS the sizes of the different molecules your solute topology consists of. The first integer is the number of molecules. Here, we have three: the peptide and the two chloride ions. The next three numbers point to the last atom of the molecule: 71 is the last atom of the peptide, 72 and 73 of the chloride atoms (which are molecules themselves).

```
BOUNDARY
#     NTB    BOX(1)    BOX(2)    BOX(3)      BETA  NRDBOX
         1       0.0         0.0         0.0       90.0         1
END
```

As previously described, with the `BOUNDARY` block you specify which periodic boundar conditions (pbc) you will use. With `NTB`=1 you specify rectangular (r) pbc (p. `II-101`).

```
TCOUPLE
#     NTT    TEMPO     TAUT
        2      60    0.100
       -2      60    0.100
        1      60    0.100
END
```

In our case we want to run the simulation at constant temperature. For this purpose, we have to add the `TCOUPLE` input block (p. `II-125`). You can specify the temperature using the `TEMPO` parameter. `NTT` are flags to specify what degrees of freedom you which to couple to the temperature bath. `TAUT` is the coupling time, used in the weak-coupling method. The different lines correspond to the different baths: solute internal, solute centre of mass, and solvent. The temperature in this block is modified by `mk_script`.

```
CENTREOFMASS
#   NDFMIN     NTCM      NSCM
       3        0      1000
END
```

This block is needed to remove the centre of mass motion (translation and rotation) (p. `II-141`). Without this block it can happen that all the kinetic energy is converted to centre of mass translation (flying ice cube problem).

```
WRITE
# NTPW = 0 : binary
# NTPW = 1 : formatted
# NTWSE = configuration selection parameter
# =0: write normal trajectory
# >0: chose min energy for writing configurations
#     NTWX    NTWSE     NTWV     NTWE     NTWG  NTWB  NTPW
       100        0        0      100        0     0     1
END
```

GROMOS produces a massive amount of data and it is impossible to store all the data it produces. The `WRITE` block meets this demand: Here you specify how often the coordinate trajectories (`NTWX`), the velocity trajectories (`NTWV`), the energy trajectories (`NTWE`), the free energy trajectories (`NTWG`), the block averaged energies (and free energies if `NTWG` > 0) trajectories (`NTWB`) are written out. Here, we are only interested in the coordinates (`NTWX`) and energies (`NTWE`) and we write them every 100th step. The second switch (`NTWSE`) says what is the selection criteria for trajectories if `NTWSE` $= 0$ the normal coordinate trajectory will be written, or if `NTWSE` > 0 the minimum energy trajectory will be written. The last switch (`NTPW`) controls the format of the trajctory files. If it's set to 0 the trajectories are writen in the binary mode, and if it's 1 they will be formated to the human readable form.

⚠ *Warning:* It makes no sense to write the trajectories too often. First, it needs a lot of disk space. Second, the data is highly correlated and so no additional insight is gained from it.

```
PRINT
#NTPR: print out energies, etc. every NTPR steps
#NTPL: print out C.O.M motion and total energy fitting every NTPL steps
#NTPP: =1 perform dihedral angle transition monitoring
#    NTPR     NTPL     NTPP
      100       0        0
END
```

This block is very similar to the `WRITE` block with the difference that the information is written in a human readable format to the output file.

```
PLIST03
#      algorithm: standard (gromos96 like pairlist)
#                 grid (XX grid pairlist)
#      SIZE:     grid cell size (or auto = 0.5 * RCUTP)
#      TYPE:     chargegroup (chargegroup based cutoff)
#                      atomic (atom based cutoff)
#
#      algorithm     NSNB   RCUTP  RCUTL  SIZE   TYPE
       grid          5      0.8    1.4    auto   chargegroup
END
```

GROMOSXX knows different algorithms for the generation of the pairlist, a list containing the atoms interacting with each other. Here, we use the grid based pairlist generation: the space is discretized into grid cells and only the neighboring cells are searched for interacting partners. The use of this algorithm results in a significant speed increase because the scaling of the algorithm is changed from a $O(N^2)$ law to a $O(N)$ law. The pairlist is generated every 5th (`NSNB`) step. `RCUTP` and `RCTL` are the cutoffs for the pairlist construction for the short-range and the long-range interactions (p. `II-53`).

```
POSREST
#    values for NTR
#    0: no position re(con)straining
#    1: use CHO
#    2: use CHO/ ATOMIC B-FACTORS
#    3: position constraining
#     NTR      CHO   NRDRX
      1      25000      1
END
```

Finally, we want to restrain the position of our solute. The restraining is achieved by a harmonic term with a force constant `CHO`. This force constant is also modified by `mk_script`.

Now you should understand the main blocks of the GROMOSXX input files.

Hint: You can find further information in the *PROMD Quick Reference Sheet* (you can find it on the http://www.igc.ethz.ch/education/advanced/cscbp/exe) and in the *GROMOS Manual and User's Guide* (see the page numbers given above).

---

2 Now it is time to have a look at `mk_script`. Open the input file `eq_mkscript.inp`. Here choose a system name `@sys`, describing your simulation. `@bin` points to the GROMOSXX executable and `@dir` points to the directory where the simulation files are. Using the `@files` argument you specify all the files needed by your simulation (topology, GROMOSXX input file, initial coordinate file, position restraints file). `@template` respectively `mkscript.lib` is a configuration file for `mkscript`. Therein you can adapt `mk_script` to your local system. Because we are using GROMOSXX you have to give the `@XX` flag. Finally, you tell `mk_script` what to do, using a joblist file that you specify with the `@joblist` argument.

Hint: All file paths that you give in a `mk_script` input file must be relative to the path `@dir`.

---

3 The joblist is already prepared for you. Therein you can change parameters in the GROMOSXX input file for a certain job. Have a look into the file:

```
~/peptide/eq> cat equilibration.jobs
TITLE
General startup protocol.
heating while loosening the position restraints.
END
JOBSCRIPTS
job_id NTX INIT TEMPI TEMP0[1] TEMP0[2] TEMP0[3] NTP NTR CHO  subdir run_after
1       1    1   60.0     60.0     60.0     60.0    0   1   2.5E4  .       0
2       2    4   0.0     120.0    120.0    120.0    0   1   2.5E3  .       1
3       2    4   0.0     180.0    180.0    180.0    0   1   2.5E2  .       2
4       2    4   0.0     240.0    240.0    240.0    0   1   2.5E1  .       3
5       2    4   0.0     300.0    300.0    300.0    0   1   0.000  .       4
END
```

All the jobs have a certain ID which you can find in the first column. In the next columns you specify the parameters of the GROMOSXX input file that you want to change. In the first job, we have to generate the initial velocities. Thus, we give an initial temperature (`TEMPI`) and set the `NTX` and `INIT` parameters to one. In the further jobs we will read the velocities from a file, hence we set `NTX` to 2 and `INIT` to 4. You can see that we are increasing the temperature by 60K at every new job (`TEMP0[1..3]`). Simultaneously, the force constant (`CHO`) for the position restraints is decreasing by an order of magnitude at every new job. Finally, in the last columns you specify in which directory and in what order you want to run the jobs.

4 The only thing that is missing to start the equilibration is the file containing the position restraints. To prepare it, copy the coordinate file to the local directory and open it with a text editor and prepare the `peptide_2Cl_h2o.posres` as you prepared the `sim_box_peptide.posres` for the energy minimisation of the solvent (page 26).

5 Now, run `mk_script`:

```
~/peptide/eq> mk_script @f eq_mkscript.inp
```

This creates five `jeq_peptide_*.sh` job scripts and the corresponding input files (`eq_peptide_*.inp`).

6 You are now ready to start the thermalization and equilibration. Run the first job script and the others will be automatically executed as soon as the preceding script has finished.

```
~/peptide/eq> ./jeq_peptide_1.sh
```

Warning: Depending on your system's speed this will take up to a couple of hours.

Hint: Have a look at all the output files `eq_peptide_*.out`. If anything goes wrong, a message will be printed to the output file.

7 After the equilibration has finished run some basic checks in the `eq/ana` directory.

```
~/peptide/eq/ana> ene_ana @f ene_ana.inp
~/peptide/eq/ana> xmgrace totkin.dat
```

There you can see that the kinetic energy is increasing at every new job (Figure 4).

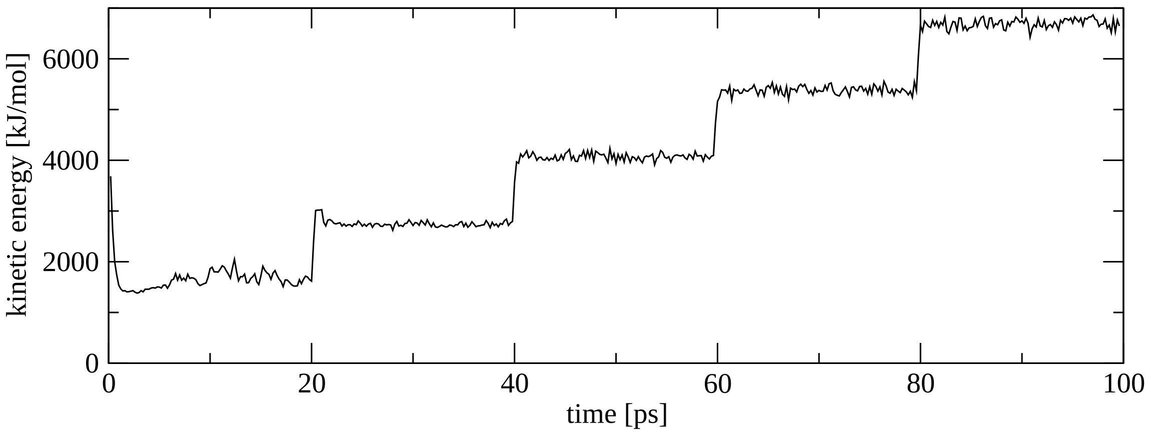


Figure 4: The kinetic energy during the equilibration.

3.3.2 Molecular dynamics sampling run

The equilibration period already produced a short simulation at constant temperature and volume. At this point, we now want to elongate the simulation to a nanosecond under constant temperature and *pressure*. Again, we use the `mk_script` program to create the job scripts and the input files.

1 First, have a look at the input file `md.inp`. Compared to the file used for the equilibration there are only a few differences: First, we don't use position restraining anymore and so, the POSRES block was deleted. Second, we want to simulate under constant pressure rather than constant volume. For this purpose we have to add an additional block:

```
PCOUPLE03
#      COUPLE: off, calc, scale
#      SCALE: off, iso, aniso, full
#      VIRIAL: none, atomic, molecular
#
#   COUPLE SCALE  COMP       TAUP   VIRIAL
    scale  iso    4.575E-4   0.5    molecular
#   reference pressure
    0.06102    0.00000    0.00000
    0.00000    0.06102    0.00000
    0.00000    0.00000    0.06102
END
```

In this PCOUPLE03 we tell GROMOSXX to isotropically scale the box. The weak-coupling method (p. II-131) uses two additional parameters: COMP is the isothermal compressibility and TAUP the coupling time. We are calculating the molecular virial, so intramolecular forces don't contribute to the pressure fluctuations. Finally, we have to specify the reference pressure in tensor form.

In the other blocks only minor things have changed: the temperature was set to 300K and the trajectories are written out less often.

2 In the mk_script input file (md_mkscript.inp) the specifications of the position restraints file and the joblist file are not needed any more. Instead of the joblist we use the @script argument. Here, we tell mk_script to create 10 consecutive scripts, beginning from the first (1).

3 Run mk_script to generate the job scripts.

```
~/peptide/md> mk_script @f md_mkscript.inp
```

This command creates 10 jmd_peptide_*.sh job scripts and corresponding input files (md_peptide_*.inp).

4 Now, you can submit the first script to the job control system (queue).

```
~/peptide/md> ./jmd_peptide_1.sh
```

5 After all the jobs are finished, you should start to analyze the trajectories.

3.4 Analysis of the penta-peptide

3.4.1 Analysis of the energy trajectory

ene_ana GROMOS can write energies, free-energy derivatives and block averages of these to separate trajectory files for later analysis. Program ene_ana extracts individual values from such files and can perform simple mathematical operations on them. In this tutorial we have only written energy trajectories (*.ene.gz) and in the following you will learn how to extract time series and averages from these files.

programs You need the following programs from the GROMOS simulation package:

```
GROMOS++: ene_ana
others:  xmgrace
```

input files You need the following input files

```
ene_ana_peptide.inp
```

output files The procedure will create the following output files

```
ene_ana_peptide.out totene.dat totpot.dat totkin.dat pressu.dat
solutemp2.dat solvtemp2.dat
```

Have a look at the input file:

```
~/peptide/ana/ene_ana> cat ene_ana_peptide.inp
@en_files ../../md/md_peptide_1.ene.gz
          ../../md/md_peptide_2.ene.gz
...
@prop    totene totpot totkin solutemp2 solvtemp2 pressu
@topo    ../../topo/peptide_2Cl_53a6.dat
@time    0 5
@library ene_ana.md++.lib
```

With @en_files you tell ene_ana which energy trajectories should be read in. The @prop argument specifies for which properties the time series should be extracted from the energy trajectory. The topology is specified with @topo. The @time argument gives the starting time of the first energy trajectory and the time difference between energy blocks in the trajectory (i.e. DT*NTWE) in ps. If the @time argument is not given the time is taken from the trajectory file. One can specify an ene_ana library with the @library argument. Note that you have to adapt the input file such that ene_ana finds the library. It is in <path to your gromos++>/gromos++/data/ene_ana.md++.lib.
After adapting the path to the ene_ana.md++.lib library you can run ene_ana

```
~/peptide/ana/ene_ana> ene_ana @f ene_ana_peptide.inp > ene_ana_peptide.out
```

Have a look at the output:

```
~/peptide/ana/ene_ana> cat ene_ana_peptide.out
  property      average        rmsd   error est.
    totene     -30856.6      104.227         NaN
    totpot     -37561.3      136.808         NaN
    totkin      6704.68      96.9994         NaN
 solutemp2      296.306      25.6229         NaN
 solvtemp2      302.299      4.53185         NaN
    pressu     -27.2633      309.218         NaN
```

The program calculates the average of the specified properties as well as the root-mean-square deviations (`rmsd`) and a statistical error estimate (`error est.`). The error estimate is calculated from block averages of different sizes, as described in Allen and Tildesley: "Computer Simulation of Liquids", 1987. As you can see the error estimate is `NaN` (not a number) in this case, which is due to the fact that we do not have enough values to calculate a meaningful error estimate.

`ene_ana` also produced a couple of time series files

```
~/peptide/ana/ene_ana> ls *.dat
pressu.dat   solutemp2.dat solvtemp2.dat totene.dat   totkin.dat
totpot.dat
```

Exercise: Have a look at these time series with `xmgrace`. Annotate the plots (axes, legends etc.). Print the plots and hand them in.

In the following you should learn how to use the `ene_ana` library. In the input file we specified as a first property `totene`. How did `ene_ana` know which numbers should be extracted from the energy trajectory in order to calculate the total energy? Have a look at one of the energy trajectories

```
~/peptide/ana/ene_ana>gunzip ../../md/md_peptide_2.ene.gz
~/peptide/ana/ene_ana>more ../../md/md_peptide_2.ene
TITLE
       GromosXX
       Automatically generated input file
       nschmid Tue Aug 21 13:29:24 2007

       energy trajectory
END
TIMESTEP
          2499  104.998000000
END
ENERGY03
# totals
  -3.066332455e+04
   6.670134364e+03
  -3.733345892e+04
   0.000000000e+00
   1.449533109e+02
   3.461929947e+01
   4.820698711e+01
   6.298362133e+03
```

```
 -4.385960065e+04
  0.000000000e+00
  0.000000000e+00
  0.000000000e+00
  0.000000000e+00
  0.000000000e+00
```

As you can see there are a number of blocks but you will not find `totene` anywhere. To which number `totene` refers is specified in `ene_ana.md++.lib` [*]:

```
~/peptide/ana/ene_ana>cat ene_ana.md++.lib | grep "totene = "
totene = ENER[1]
~/peptide/ana/ene_ana>more ene_ana.md++.lib
TITLE
  XX Library file for ene_ana
END
ENERTRJ
# block definition for the energy trajectory file.
# which is specified by the input flag en_files of program ene_ana.
#
# Use keyword 'block' to specify the blocks
#            'subblock' to specify name and dimensions of a set of data
#            'size' to specify a size that should be read in from the file
#                  this size can be used as dimension specification
#                  in a subblock definition. Using the prefix 'matrix_'
#                  with such a definition will expand the size N to
#                  N*(N+1)/2
#
# Following is the definition for a gromosXX energy trajectory
#
  block TIMESTEP
    subblock TIME 2 1
  block ENERGY03
    subblock ENER 16 1
    size NUM_BATHS
    subblock KINENER NUM_BATHS 3
    size NUM_ENERGY_GROUPS
    subblock BONDED NUM_ENERGY_GROUPS 4
    subblock NONBONDED matrix_NUM_ENERGY_GROUPS 2
    subblock SPECIAL NUM_ENERGY_GROUPS 7
  block VOLUMEPRESSURE03
    subblock MASS 1 1
    size NUM_BATHS
    subblock TEMPERATURE NUM_BATHS 4
    subblock VOLUME 10 1
    subblock PRESSURE 30 1
END
```

As you can see `totene` is defined as the first entry of the `ENER` array. The `ENER` array is defined as a subblock of the `ENERGY03` block. This subblock has 1 column with 16 lines (`subblock ENER 16 1`).

Exercise: Use `ene_ana` to calculate the density of your system. You will have to specify the correct variable for the density behind the `@prop` argument.

[*]some properties can also be calculated without specifying a library file as some part of the library is implemented already in the program itself. However, understanding the library syntax is important as it allows you to calculate any property you wish from the energy trajectory.

Which variable that is you will see in the `ene_ana.md++.lib`. Plot the density with xmgrace, annotate and hand in the plot.

Exercise: This is a rather advanced exercise. We want to extract the time series of the total nonbonded interactions of the peptide with itself, the chloride ions and the water. The energies we need for that are stored in the energy trajectory after `# nonbonded`. Remember we defined four energy groups: the peptide, a first chloride ion, a second chloride ion and the water. The `# nonbonded` matrix contains the following information

```
# nonbonded
#   van der Waals      Coulomb
  -7.834279900e+01 -2.620656341e+02 # 1 - 1  peptide with peptide
  -2.265527287e-01 -7.544253397e+01 # 1 - 2  peptide with first Cl-
  -9.867269334e-02 -3.503961042e+01 # 1 - 3  peptide with second Cl-
  -4.343061015e+00 -1.790762688e+03 # 1 - 4  peptide with water
   0.000000000e+00  0.000000000e+00 # 2 - 2  first Cl- with itself
  -1.374664107e-01  7.029129278e+01 # 2 - 3  first Cl- with second Cl-
   3.750767590e+01 -6.546658248e+02 # 2 - 4  second Cl- with water
   0.000000000e+00  0.000000000e+00 # 3 - 3  second Cl- with itself
   3.680131152e+01 -6.765556824e+02 # 3 - 4  second Cl- with water
   6.307201697e+03 -4.043535997e+04 # 4 - 4  water with water
```

Copy the `ene_ana` library to your current directory and change its name (e.g. add a 'mod'). You will have to define four new variables. Add them at the end of the file (but before the last `END`). The variable for the peptide-peptide interactions could e.g. be called `e_pp`. It would be defined as

```
e_pp = NONBONDED[1][1] + NONBONDED[1][2]
```

i.e. we add up the van der Waals and Coulomb energies of the peptide-peptide interactions. Now define three more variables which you could call e.g. `e_pCl1` (peptide with first chloride), `e_pCl2` (peptide with second chloride), `e_pwater` (peptide with water). This newly defined variables you can now specify as properties (`@prop`). `ene_ana` will then produce the following four files

```
e_pCl1.dat   e_pCl2.dat   e_pp.dat     e_pwater.dat
```

Plot these time series with xmgrace, annotate and hand in the plot. The plot should look like Figure 5.

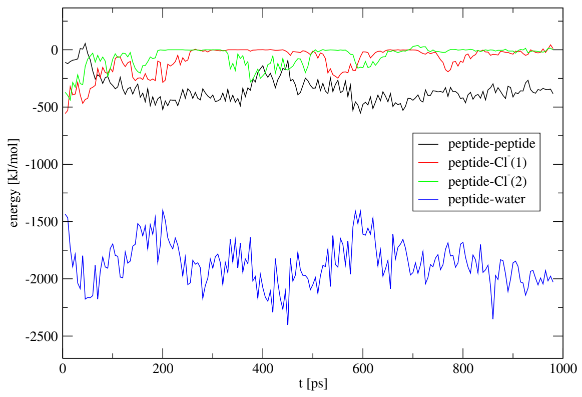


Figure 5: Time-series of energies.

3.4.2 Analysis of the coordinate trajectory

visual analysis You can generate PDB snapshots from your trajectory (or any other) coordinate files using the program `frameout`.

programs You need the following programs from the GROMOS simulation package:

```
GROMOS++: frameout
others:   vmd
```

input files You need the following input files

```
frameout_1.inp frameout_2.inp
```

output files The procedure will create the following output files

```
FRAME_00001.pdb FRAME_00035.pdb FRAME_00049.pdb
```

Like in most GROMOS++ programs you have to specify some basic information about your system using the `@topo` and `@pbc` arguments. Let's look at an example:

```
~/peptide/ana/frameout> cat frameout_1.inp
@topo           ../../topo/peptide_2Cl_53a6.dat
@pbc            r
@include        ALL
@outformat      pdb
@traj           ../../coord/peptide_2Cl_h2o.g96
```

With the `@include` argument you tell frameout which atoms of the topology it should include in the output file. Here, we want to write all the atom coordinates, including the solvent, to the output file. Other options are `SOLVENT` which selects the solvent and `SOLUTE` which selects the solute. The latter is the default and so the `@include` argument can be omitted in this case.

The format of the output file can the specified with the `@outformat` argument. Here, we use the PDB (Protein Data Bank) format. It is worth mentioning that other formats, including `g96`, are supported.

💡 *Hint:* You can use frameout to create a gathered G96 frame by selecting `g96` as output format.

Finally, you have to tell frameout where it can find the coordinate files with the `@traj` argument. Of course you can give multiple files as input.
Now try to run frameout:

```
~/peptide/ana/frameout> frameout @f frameout_1.inp
```

This command has generated a `FRAME_00001.pdb` file which can be opened with a PDB viewer like VMD or pyMOL.

⚠️ *Warning:* Rename the PDB files generated by frameout or you will accidentally overwrite them.

It happens very often that you see a certain effect (like a RMSD increase) at a certain time step and you want to have a look at this frame to see what happened. For this case, you have to add some arguments to the frameout input file:

```
~/peptide/ana/frameout> cat frameout_2.inp
@topo          ../../topo/peptide_2Cl_53a6.dat
@pbc           r
@include       SOLUTE
@outformat     pdb
@traj
../../md/md_peptide_1.trj.gz
../../md/md_peptide_2.trj.gz
...
@spec          SPEC
@frames        35 49
```

First, I added all the md trajectory coordinate files to the `@traj` argument. Because we want to extract a certain frame we tell frameout to be `SPEC`ific using the `@spec` argument. Last, the frames have to be given (`@frames`). This will create two PDB files containing the given frames.

💡 *Hint:* You can use frameout to produce a movie that you can watch in VMD. Have a look at the files `frameout_movie_1.inp` and `frameout_movie_2.inp`.

---

rdf The radial mean distribution function (rdf) gives you the number density of atoms of a specified type around an atom (which can be of the same or of an other type). The theoretical description is given in section 1.10. GROMOS++ offers a tool that calculates the radial mean distribution function called `rdf`.

programs You need the following programs from the GROMOS simulation package:

```
GROMOS++: rdf
others:  xmgrace
```

input files You need the following input files

```
rdf_peptide_2CL.inp
```

output files The procedure will create the following output files

```
rdf_peptide_2CL.out
```

Have a look at the input file:

```
~/peptide/ana/rdf> cat rdf_peptide_2CL.inp
@topo   ../../topo/peptide_2Cl_53a6.dat
@pbc    r
@centre a:CL
@with   s:OW
@cut    1.4
@grid   100
@traj   ../../md/md_peptide_1.trj.gz
        ../../md/md_peptide_2.trj.gz
...
```

The used topology is specified in `@topo`. With `@pbc` you tell `rdf` which periodic boundary condition was used (rectangular r in our case). By `@centre` you specify the which atom(s) you want to be at the center of your rdf and by `@with` which ones should be the surrounding ones.

Hint: atomspecifier
`a:CL` looks in all molecules for CL (chlorine ion)
`s:OW` looks in all solvent molecules for OW (water oxygen)

With the parameter `@cut` you specify to which distance you want to look at (do not go over half the box size) and with `@grid` how many bins from 0 to `@cut` you want to have (bigger number means finer resolution, but less statistics). Finally you specify with `@traj` which trajectories you want to look at. The more trajectories, the better the statistics.
Now you can run `rdf`

```
~/peptide/ana/rdf> rdf @f rdf_peptide_2CL.inp > rdf_peptide_2CL.out
```

Have a look at the output. It should look similar to Figure 6 where the first peak representing the first hydration shell is clearly visible meaning that the water molecules organise around the chlorine ion. Note that there is no water molecule very close to the chloride ion as two atoms cannot be at the same place.

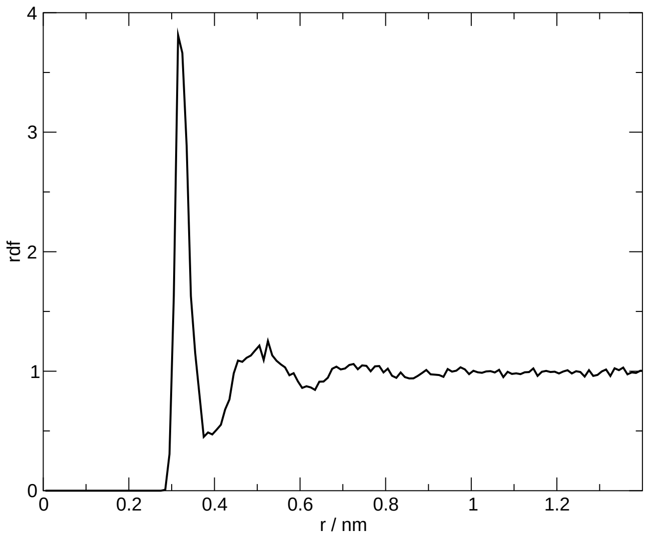


Figure 6: Cl-OW radial distribution function of chlorine ions in water

---

Exercise: Have a look at the difference between the `rdf` around the first and the second chlorine atom with `xmgrace`. Label the plot (axes, legends etc.). Print the plot and hand it in.

Hint: All you have to do to solve the exercise is to change the `@centre` in your input file. To find the atomespecifier for the first chlorine atom, open the topology and see which ATNM (atom number) it has (should be 72). Then use

```
atominfo @topo ../../topo/peptide_2Cl_53a6.dat @gromosnum 72
```

The atomspecifier is then 2:1 (first atom of the second molecule). Proceed in the same way for the other chlorine atom.

---

rmsd Atom-positional root-mean-square difference (RMSD) is a measurement of structural deviation between two given conformations. For two conformations with the coordinates $\mathbf{x}$ and $\mathbf{y}$, the RMSD is given by

$$RMSD(\mathbf{x},\mathbf{y}) = \left(\frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \mathbf{y}_n)^2\right)^{\frac{1}{2}} \tag{34}$$

where x_n is the position of the n-th particle and y_n is its reference position.

programs To calculate the RMSD you need the following programs from the GROMOS simulation package:

```
GROMOS++: rmsd
others:  xmgrace
```

input files You need the following input files

```
rmsd_peptide.inp
```

output files The procedure will create the following output files

```
rmsd_peptide.out
```

Have a look at the input file:

```
~/peptide/ana/rmsd> cat rmsd_peptide.inp
@topo  ../../topo/peptide_2Cl_53a6.dat
@pbc   r
@time  0 0.5
@atomsrmsd 1:CA,N,C
@ref   ../../eq/eq_peptide_5.gsf
@traj  ../../md/md_peptide_1.trj.gz
       ../../md/md_peptide_2.trj.gz
...
```

Again the topology is given by `@topo` and `@pbc` defines the periodic boundary condition. With `@time` you define the starting time (first number) and the increment in time per step (second number). In `@atomsrmsd` one gives the atomspecifier of the atoms of which one wants to calculate the `rmsd` compared to a reference structure `@ref`. By `@traj` the trajectories are specified.
Now calculate the RMSD with `rmsd`

```
~/peptide/ana/rmsd> rmsd @f rmsd_peptide.inp > rmsd_peptide.out
```

The output should look like Figure 7 where you see that first the RMSD is zero as the reference structure and the momentary structure are nearly identical. Then the RMSD increases as the structure is evolving away from the reference.

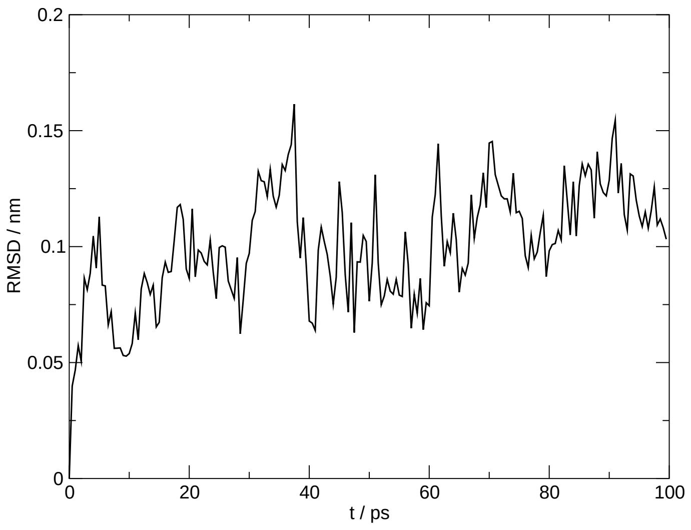


Figure 7: Atom-positional root-mean-square deviation of the backbone atoms of the penta-peptide from the starting (reference) structure.

Exercise: Compare the RMSD of the backbone with the RMSD of the whole protein using `xmgrace`. Do you see more deviation from the original structure in the RMSD from the backbone or from the whole protein? Is that what you expected? Annotate the plots (axes, legends etc.). Print the plot, answer the questions and hand them in.

Hint: The atom specifier for the whole protein is `1:a` (all atoms of the first molecule).

---

rmsf Atom-positional root-mean-square fluctuation RMSF gives fluctuations of an atom coordinate $\mathbf{r}_i$ around its mean and is given by

$$RMSF(\mathbf{r}_i) = \left(\frac{1}{N_t} \sum_{n=1}^{N_t} \left(\mathbf{x}_i(t_n) - \langle \mathbf{x}_i \rangle \right)^2 \right)^{\frac{1}{2}} \tag{35}$$

where $x_i(t_n)$ is the postion of atom i at time $t_n = n\Delta t$ and $< x_i >$ is its mean position.

programs To calculate the RMSF you need the following programs from the GROMOS simulation package:

```
GROMOS++: rmsf
others:  xmgrace
```

input files You need the following input files

```
rmsf_peptide.inp
```

output files The procedure will create the following output files

```
rmsf_peptide.out
```

Have a look at the input file:

```
~/peptide/ana/rmsf> cat rmsf_peptide.inp
@topo  ../../topo/peptide_2Cl_53a6.dat
@pbc   r
@atomsfit 1:CA
@atomsrmsf 1:CA,N,C
@ref   ../../eq/eq_peptide_5.gsf
@traj  ../../md/md_peptide_1.trj.gz
       ../../md/md_peptide_2.trj.gz
...
```

Topology and periodic boundary conditions are given by `@topo` and `@pbc`, respectively. With `@atomsfit` one specifies which atoms are used to superpose the structures (in order to remove translational and rotational changes in x_i; just look at the structural changes) and `@ref` specifies the reference structure. The atoms for which the RMSF's are calculated are specified by `@atomsrmsf` (here it is the backbone of the protein). And `@traj` specifies which trajectories are looked at.
Now you can run `rmsf`

```
~/peptide/ana/rmsf> rmsf @f rmsf_peptide.inp > rmsf_peptide.out
```

Exercise: Look at the RMSF of the backbone to see that the ends of the peptide are more flexible than the middle. Look at the RMSF of the whole protein to see that the sidechains move more than the backbone. Plot your output with `xmgrace`, label the graph and hand it in.

---

time series of properties Very often you are interested in a certain property which changes during time. The monitoring of properties is done using a time series. In addition, you may want to compare a property of your simulation with an experimental value. In this case an average is calculated which can be compared to experimental data.

programs You need the following programs from the GROMOS simulation package:

```
GROMOS++: tser
others:  xmgrace
```

input files You need the following input files

```
tser_peptide.inp
```

output files The procedure will create the following output files

```
tser_peptide.out
```

Such kind of analysis is carried out with the `tser` GROMOS++ program. `tser` is a very powerful program and its basic function is explained now. Have a look into the example file:

```
~/peptide/ana/tser> cat tser_peptide.inp
@topo     ../../topo/peptide_2Cl_53a6.dat
@pbc      r
@time     0 5
@traj
 ../../md/md_peptide_1.trj.gz
...
@prop
d%1:3,69
d%1:37,2:1
a%1:70,69,71
t%1:47,46,48,49
```

First, you have to tell `tser` were the topology (`@topo`) resides and which boundary conditions (`@pbc`) you are using. Using the `@time` argument you indicate at what time your trajectory starts (0) and at what time interval your frames are printed out (e.g. every 5 ps, i.e. `NTWE*DT`). With the `@traj` argument, tell `tser` where it can find the trajectory coordinates files.

Second, tell `@tser` using `@prop` which properties it should calculate and print out.

1 In our penta-peptide system an interesting property is the head to tail distance. Its fluctuations over time give you an indication on the stiffness of the secondary structure: a stable α-helix, for example, has a rather constant head to tail distance. With `tser`, the calculation of distances is very easy: the `d` defines a distance between the two atoms in the atom specifier after the `%` sign. `d%1:3,69` thus calculates the distance between atoms one and two in the atomspecifier `1:3,69`. `1:3` is the nitrogen atom at the N-terminus, and `69` is the carbon atom at the C-terminus. Because atom `69` also belongs to the first molecule, `1:` can be omitted.

2 In this second example (`d%1:37,2:1`) the distance between the arginine residue (`1:37` is the `CZ` atom) and the first chloride ion is calculated. Because the chlorine atom is a molecule on it's own, you have to specify this with the `2:` molecule indicator. `2:1` so is the first atom of the second molecule which is the chloride ion.

3 Here we look at an angle (`a`). The angle is defined by the 3 atoms in the atom specifier after the `%` sign. In this case we monitor the atoms of the C-terminal carboxy group (`O1`, `C` and `O2`).

4 Finally, in this last example a torsional angle (`t`) is calculated. The torsional angle is defined by four atoms in the atom specifier after the `%` sign. Torsional angles of protein backbones are of interest (Ramachandran map) and are called ϕ- and ψ-angles. Here we look at the angle between the H-N bond (atoms `47,46` and the CA-CB (`48,49`) bond of the lysine residue.

Call `tser` and redirect its output:

```
~/peptide/ana/tser> tser @f tser_peptide.inp > tser_peptide.out
```

Now you can open the file in `xmgrace` and plot the time series. E.g. plot the time (1, first column in output) versus the arginine-chloride distance (3, third column in output)

```
~/peptide/ana/tser> xmgrace -block tser_peptide.out -bxy 1:3
```

Hint: The last line of the output file contains the averages of the properties which can be compared to experimental data.

Hint: Have a look at the doxygen documentation of the `atom specifier` and the `property specifier`. There you will find that you can specify many more properties (`@prop`) than shown in this simple example.

If everything worked out, your arginine-chloride distance should look like shown in Figure 8.

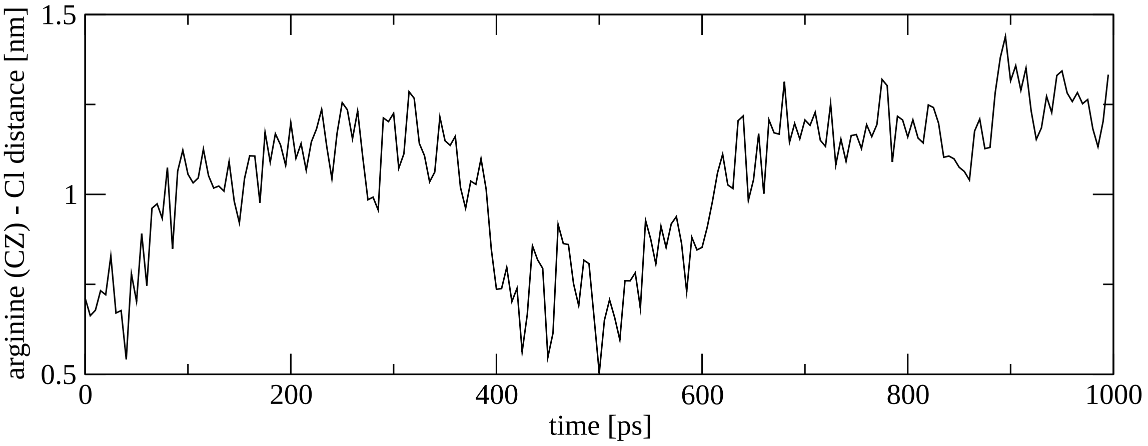
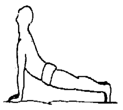


Figure 8: Time-series of the arginine (CZ) - chloride (1) distance.

 *Exercise:* The distance between the two chloride atoms is an interesting property of your system. Create a time series and plot it with xmgrace. Annotate and hand in the graph.

---

hydrogen bonds belong to important intra- and intermolecular interactions. For example, the conservation of the genetic information of a cell is based on hydrogen bonding between nucleic acids (Watson-Crick base pairing). The secondary structure of proteins is formed by hydrogen bonds between backbone amide hydrogens and oxygens. It is thus obvious that you want to analyze the hydrogen bonds of the penta-peptide in detail.

programs You need the following programs from the GROMOS simulation package:

```
GROMOS++: hbond
others:  xmgrace
```

input files You need the following input files

```
hbond_peptide.inp
```

output files The procedure will create the following output files

```
hbond_peptide.out Hbts.out Hbnumts.out
```

The GROMOS++ program `hbond` meets the demand for hydrogen bond analysis. Have a look at the input file:

```
~/peptide/ana/hbond> cat hbond_peptide.inp
@topo           ../../topo/peptide_2Cl_53a6.dat
@pbc           r
@time          0 5
@Hbparas       0.25 135
@massfile      mass.file
@AcceptorAtomsA 1:a
@DonorAtomsA   1:a
@AcceptorAtomsB 1:a
@DonorAtomsB   1:a
@traj
../../md/md_peptide_1.trj.gz
../../md/md_peptide_2.trj.gz
...
```

With the first three arguments you are already familiar from previous analyses. `hbond` determines the hydrogen bonds between donors (the atoms carrying a hydrogen atom) and acceptors (the atoms to which the bonds are formed) by geometrical criteria. `@Hbparas` takes two parameters for H-bond calculation: the maximum distance between the hydrogen and the acceptor and the minimum angle between donor, hydrogen and acceptor. The parameters in this example are used very often and can be considered as standard. `hbond` has to know which atoms it may consider as acceptors or donors. In the `@massfile` the acceptors and donors (and the hydrogen itself) are identified by their masses. You can define two groups (A and B) between which the H-bonds are calculated. In this case we are interested in intramolecular hydrogen bonds, so both of the groups consist only of atoms of the peptide itself. We thus have to set the `@AcceptorAtomsA`, `@DonorAtomsA` and the `@AcceptorAtomsB`, `@DonorAtomsB` arguments all to an atom specifier selecting the whole peptide (`1:a`). The execution of the `hbond` program

```
~/peptide/ana/hbond> hbond @f hbond_peptide.inp > hbond_peptide.out
```

produces three output files: `hbond_peptide.out` (the redirected standard output), `Hbnumts.out` and `Hbts.out`

1 The `hbond_peptide.out` file contains a summary table. In this table all the hydrogen bonds are numbered. In the first column you can find the molecule and residue of the donor and acceptor, followed by the atom numbers and names. In the next two columns the geometrical properties are listed (average hydrogen-acceptor distance, average donor-hydrogen-acceptor angle). Finally, the last two columns show the occurrence (absolute and relative) of the hydrogen bonds.

2 The total number of hydrogen bonds at a certain time is listed in the `Hbnumts.out` file.

3 Finally, the `Hbts.out` file contains times series of all hydrogen bonds found in the summary table. In the first column the time, in the second the hydrogen bond number is listed. This file is usually filtered before plotting. A plot of the unfiltered file can be seen in Figure 9.

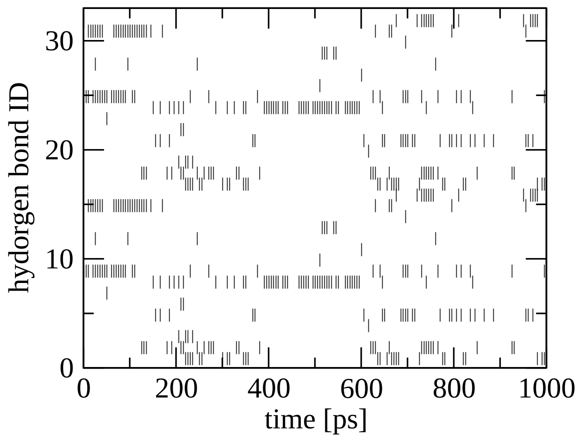


Figure 9: Time series of hydrogen bonds.

Exercise: Backbone hydrogen bonds are crucial for secondary structure. Can you tell which of them are present at $t = 640$ ps?

---

clustering To map the structures sampled during the simulations onto a set of generic conformations, we can perform a clustering analysis on the MD trajectories.

programs You need the following programs from the GROMOS simulation package:

```
GROMOS++: rmsdmat, cluster, postcluster
```

input files You need the following input files

```
rmsdmat.inp, cluster.inp, postcluster.inp
```

output files The procedure will create the following output files

```
RMSDMAT.bin, cluster_structures.dat, cluster_ts.dat, postcluster.out
```

First program that we will use is GROMOS++ program `rmsdmat`. This program calculates the atom-positional root-mean-square deviation between all pairs of structure in a given trajectory file. The RMSD matrix can be written out in human readable form, but for saving the disk space we will write it out in binary format.

Have a look at the input file rmsdmat.inp.

```
~/peptide/ana/cluster> cat rmsdmat.inp
@topo  ../../topo/peptide_2Cl_53a6.dat
@pbc   r
@atomsfit 1:CA,N,C
@stride 1
@traj  ../../md/md_peptide_1.trj.gz
         ../../md/md_peptide_2.trj.gz
...
```

Topology and periodic boundary conditions are given by `@topo` and `@pbc`, respectively. With `@atomsfit` one specifies with atom names which atoms are used for least-squares translational and rotational fitting of atom in coordinates for the calculation of RMSD. Here we took only the backbone atoms (N, C and C_α). With `@stride` one can give a selection of structures in the trajectory file to be considered in this analysis. Here we are using all frames. With argument `@traj` we specify the trajectories which will be analyzed.

In the input for `rmsdmat` program one can optionally specify the reference structure with respect to which the RMSDs will be calculated (`@ref ../../eq/eq_peptide_5.gsf`). Furthermore, in case the atoms differ from those on which the fit is performed this can be extra specified with the argument `@atomsrmsd` 1:CA,N,C . If the reference structure is not provided in the input file, the first structure from the trajectory file in taken as the reference structure. By specifying a reference structure, one allows the program `cluster` (see further in text) to perform a forced clustering, where the first cluster contains the reference structure.

As an output file you will get RMSDMAT.bin. This output file is used as an input file for the cluster program.

Now we will use GROMOS++ program `cluster`. This program performs a conformational clustering based on a similarity matrix, such as calculated in program `rmsdmat`. The clustering algorithm is the one described in Daura, 1999. In this program a cutoff can be specified such that the structures with rmsd values smaller than this cutoff are considered as structural neighbors. The structure with the highest number of neighbors is considered as central member of the cluster of similar structures.

Have a look at the input file cluster.inp

```
~/peptide/ana/cluster> cat cluster.inp
@rmsdmat RMSDMAT.bin
@cutoff 0.1
@time 0 0.5
@maxstruct
```

As previously mentioned RMSDMAT.bin, the output file of `rmsdmat` is used here as an input file. With the argument `@cutoff` we can specify the similarity criteria for two structures. With `@time` you define the starting time (first number) and the increment in time per step (second number). With `@maxstructure` we can specify how many structures are to be considered. Here we take all of them. If we have to perform a forced clustering to a specific structure, an extra argument has to be specified in the input file together with the number of the structure to which the clustering will be forced. For example, if one performs the calculation of RMSDMAT with an reference structure, the reference structure used in this program is the zeroth structure in the RMSDMAT.bin. This extra argument should look than as following: `@force 0`.

As output files we will get two files: cluster_structures.dat and cluster_ts.dat.

Cluster_structures.dat contain information about the clustering procedure, information about each cluster such as the size of the cluster (how many structures are in each cluster), averaged lifetime of a structures in a given cluster and the number of the structure which is the central member structure of a specific cluster.

In cluster_ts.dat the accurance of a specific cluster in specific time is specified (the time series of each cluster).

Next we will use GROMOS++ program `postcluster`. This program can do additional analysis on the output of the `cluster` program.

Have a look at the input file, postcluster.inp

```
~/peptide/ana/cluster> cat postcluster.inp
@topo  ../../topo/peptide_2Cl_53a6.dat
@cluster_struct       cluster_structures.dat
@cluster_ts    cluster_ts.dat
@clusters      1-27
@lifetime             2
@traj           ../../md/md_peptide_1.trj.gz
                              ../../md/md_peptide_2.trj.gz
...
```

Among important analysis which one can do with the `postcluster` program is the lifetime-analysis. To perform it one has to specify the lifetime limit of a certain cluster with the argument `@lifetime` . This limit is specified as a number of subsequent structures different from the structures of the cluster of interest, which are needed for a switch to another cluster to occur. For example, if we specify lifetime limit equal to 2, this means that at least 2 subsequent structures of cluster that differs from the cluster of interest have to occur in order to detect a cluster transition.

`postcluster` can also be used to write out the trajectory files and single structure files containing the central member structure of the cluster. With the argument `@clusters` we can specify for how many clusters we want to write out the trajectories and the corresponding central member structures. Resulting trajectories can further be used in any other analysis program.

4 REFERENCES

Copies of references for which the code MD*.* is given below, can be obtained from the group for computer-aided chemistry of the Laboratory of Physical Chemistry, ETH, Hönggerberg, CH-8093 Zürich, Switzerland, phone +41-44-6325502

M.P. Allen and D.J. Tildesly, Computer Simulation of Liquids (1987), Clarendon, Oxford

H.C. Andersen, J.Chem.Phys., 72 (1980) 2384-2393

D. Beeman, J.Compt.Phys., 20 (1976) 130-139

H.J.C. Berendsen and W.F. van Gunsteren, In: Molecular Liquids - Dynamics and Interactions, A.J. Barnes et al. eds., NATO ASI Series C135 (1984), Reidel, Dordrecht, pp 475-500 (MD84.6)

H.J.C. Berendsen, J.P.M. Postma, W.F. van Gunsteren, A. DiNola and J.R. Haak, J.Chem.Phys., 81 (1984) 3684-3690 (MD84.9)

H.J.C. Berendsen and W.F. van Gunsteren, In: Molecular-Dynamics Simulation of Statistical-Mechanical Systems, Proceedings of the International School of Physics "Enrico Fermi", course 97, G. Ciccotti and W.G. Hoover eds. (1986), North-Holland, Amsterdam, pp 43-65 (MD86.7)

D. Brown and J.H.R. Clarke, Mol.Phys., 51 (1984) 1243

D. Frenkel and B. Smit, Understanding Molecular Simulation: From Algorithms to Applications, (2002), Academic Press, Elsevier (USA)

C.W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations, (1971), Prentice-Hall, Englewood Cliffs, N.J.

H. Goldstein, Classical Mechanics, (1950), Addison-Wesley, Reading, Mass.

W.F. van Gunsteren, H.J.C. Berendsen and J.A.C. Rullmann, Mol.Phys., 44 (1981) 69-95 (MD81.2)

W.F. van Gunsteren and H.J.C. Berendsen, Mol.Phys., 45 (1982) 637-647 (MD82.3)

W.F. van Gunsteren, H.J.C. Berendsen, F. Colonna, D. Perahia, J.P. Hollenberg and D. Lellouch, J.Comput.Chem., 5 (1984) 272-279 (MD84.3)

W.F. van Gunsteren, S.R. Billeter, A.A. Eising, P.H. Hünenberger, P. Krüger, A.E. Mark, W.R.P. Scott, I.G. Tironi, Biomolecular Simulation: The GROMOS96 Manual and User Guide Vdf Hochschulverlag AG an der ETH Zürich, Zürich, Switzerland (1996) (MD96.40)

W.F. van Gunsteren and A.E. Mark, J. Chem. Phys., 108 (1998) 6109-6116 (MD98.3)

W.R.P. Scott, P.H. Hünenberger, I.G. Tironi, A.E. Mark, S.R. Billeter, J. Fennen, A.E. Torda, T. Huber, P. Krüger and W.F. van Gunsteren, The GROMOS Biomolecular Simulation Program Package, J. Phys. Chem. A, 103 (1999) 3596-3607 (MD99.11)

M. Christen, P.H. Hünenberger, D. Bakowies, R. Baron, R. Bürgi, D.P. Geerke, T.N. Heinz, M.A. Kastenholz, V. Kräutler, C. Oostenbrink, C. Peter, D. Trzesniak, W.F. van Gunsteren, The GROMOS software for biomolecular simulation: GROMOS05, J. Comput. Chem., 26 (2005) 1719-1751 (MD05.32)

J.M. Haile and H.W. Graben, J.Chem.Phys., 73 (1980) 2421

J.P. Hansen and J.-J. Weis, Phys.Rev., 188 (1969) 314

D.M. Heynes, Chem.Phys., 82 (1983) 285-301

J.O. Hirschfelder, C.F. Curtiss and R.B. Bird, Molecular theory of gases and liquids, (1954), Wiley, New York

R.W. Hockney, S.P. Goel and J.W. Eastwood, J.Comput.Phys., 14 (1974) 148

E. Kestemont and J. van Craen, J.Comput.Phys., 22 (1976) 451-458

D.A. McQuarrie, Statistical Mechanics, (1976), Harper & Row, New York

G.C. Maitland, M. Rigby, E.B. Smith and W.A. Wakeham, Intermolecular forces: their origin and determination, (1981), Clarendon Press, Oxford

N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller, J.Chem.Phys., 21 (1953) 1087

K. Nakanishi and K. Toukubo, J.Chem.Phys., 70 (1979) 5848-5850

B. Quentrec and C. Brot, J.Comput.Phys., 13 (1973) 430-432

A. Rahman, Phys.Rev., 136 (1964) 405

M. Rao and D. Levesque. J.Chem.Phys., 65 (1976) 3233-3236

T. Schneider and E. Stoll, Phys.Rev., B17 (1978) 1302; B18 (1978) 6468

L. Verlet, Phys.Rev., 159 (1967) 98-103

L. Verlet, Phys.Rev., 165 (1968) 201

B. Widom, J.Chem.Phys., 39 (1963) 2808-2812

X. Daura, Proteins, 34 (1999) 269-280

A INSTALLATION

A.1 GNU Scientific Library

The GNU Scientific Library is a C library used to carry out scientific calculations like complex number arithmetic, fast Fourier transformations, integration of functions etc. It is needed by both GROMOSXX and GROMOS++. Usually it can be installed via your operating systems package manager

---

💡 *Hint:* On Debian linux (or Ubuntu) you can install it by typing `sudo apt-get install libgsl0 libgsl0-dev`. On Windows, install it via the CYGWIN setup.

If it is not distributed with your operating system or you are not super user you have to compile it from the source code. Fortunately this is rather easy and straight forward.

Installation from source In your home create a directory where you want to install the library `/home/nathan/gsl` and a working directory `/home/nathan/tmp`. Go the GSL web page ftp://ftp.gnu.org/gnu/gsl/ and download the latest version into your working directory. In a command line shell `cd /home/nathan/tmp` to your working directory and untar the package:

```
~/tmp> gunzip gsl-1.9.tar.gz
~/tmp> tar xf gsl-1.9.tar
~/tmp> cd gsl-1.9
```

Now you need to `./configure` for the compilation and installation. As a prefix give the directory where you want to install the library. After successful configuration make and install the GSL.

```
~/tmp/gsl-1.9> ./configure --prefix=/home/nathan/gsl
~/tmp/gsl-1.9> make
~/tmp/gsl-1.9> make install
```

💡 *Hint:* On multi-core CPU machines you can add the `-j3` flag to `make` to boost the compilation.

After the installation was successful you can delete the working directory

```
~/tmp/gsl-1.9> cd ..
~/tmp> rm -rf gsl-1.9*
```

The GSL is now successfully installed in your home.

A.2 Parallel version of GROMOSXX

In order to enable GROMOSXX to use the full power of your computer's hardware you have to compile a parallel version. GROMOSXX know two kinds of parallelization:

OpenMP This parallelization is straight forward and enables GROMOSXX to run on multiple core CPUs (like all recent x84 CPUs are). No additional software is required. You can enable this by adding `--enable-openmp` to configure.

MPI MPI is used for parallelization when no shared memory is available: the different CPUs you want to use for the calculations are located in different machines. This is the case in computer clusters[†].

To compile the MPI version you have to use a special set of compilers (which knows which MPI version and implementation you have on the cluster). In general these compilers are called `mpicc` for the C compiler and `mpiCC` for the C++ compiler. You have to tell `configure` to use the compilers and to `--enable-mpi`:

```
~/temp/gromosXX-0.2.3> ./configure CC=mpicc CXX=mpiCC \
                       --enable-mpi \
                       --disable-shared --disable-debug \
                       --with-gsl=/home/nschmid/gsl \
                       --prefix=/home/nschmid/gromosxx
```

After successful configuration just `make` and `make install` it as usual. If the test call

```
~/temp/gromosXX-0.2.3> /home/nschmid/gromosxx/md/md_mpi
```

does not tell you to enable MPI, everything is fine.

[†]huge networks of many rather small computers