The GROMOS Software for (Bio)Molecular Simulation

Tutorial with Examples

GRAZ 2023 beginners course
Introduction to Molecular Modeling with GROMOS
13 september 2023
Graz, Austria

September 12, 2023

Contents

CHAPTER 1

Introduction

1.1. Simulation using GROMOS

GROMOS is a software package used for computer simulations of molecular systems like proteins, inorganic and organic chemical compounds, DNA, etc. In order to perform a molecular dynamics simulation using GROMOS one should:

1. create a topology of the system (`*.top` file)
2. have spacial coordinates of the system converted into GROMOS format (`*.cnf` file)
3. prepare an input file (`*.imd`) with parameters for the MD simulation

1.1.1. 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 basic units shown in Tab. 1.1. From this basic set of units you can derive the units for all other quantities used in GROMOS. Units for some important quantities are given in Tab. 1.2. Apart from restrictions when storing or printing data in non-exponential format, the GROMOS programs are independent of the chosen units. The units are defined by the ones used for physical constants (Tab. 1.3) and atomic or molecular quantities in the (GROMOS) data files. Note that a number of *quantities* or interaction function *parameters* in GROMOS are either angles or *dependent on angle units* through their definition (see Chap. 6-6). For convenience of the user these quantities are kept in the *GROMOS data files* using *degrees as angle units*. However, when angles are used in calculations involving mathematical functions such as *sin*, *cos*, etc. they should be expressed in radians. Therefore, upon reading GROMOS data files the values of quantities and parameters that depend on angle units are converted from degrees to radians. So, in the *programs and subroutines* these quantities and parameters are stored using *radians as angle units.* For more information on units see Chap. 6-6.

Quantity		Unit	
length:	r :	nm	10^{-9} m
mass:	m:	u	atomic mass unit
			1/12 of the mass of a ^{12}C atom
			$10^{-3}/N_{\mathrm{Av}}$ 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

TABLE 1.1. Recommended units

1.1.2. File and software organisation. GROMOS knows different types of data and data files. Two types of information concerning a molecular system can be distinguished.

Quantity		Unit	
energy:	$\mathcal{U}$,$\mathcal{K}$:	kJ mol^{-1}	0.2390 kcal mol^{-1}
force:	$\boldsymbol{f}$:	kJ mol^{-1} nm^{-1}	
pressure:	$\mathcal{P}$:	kJ mol^{-1} nm^{-3}	$10^{30}/N_{\mathrm{Av}}$ Pa
			1.6605655 MPa
			16.6057 Bar
			16.3885 atm
velocity:	$\boldsymbol{v}$:	nm ps^{-1}	

TABLE 1.2. Derived units

Constant		Value
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	$R/N_{\mathrm{Av}} = 1.380662 \cdot 10^{-26}$ kJ K^{-1}

TABLE 1.3. Physical constants used

1. *Topological information*: data on the covalent structure, atomic masses, charges, van der Waals parameters, atom-atom distance restraints specification, ^3J-value restraints specification, local-elevation dihedral angles specification, etc.
2. *Configurational information*: atomic coordinates and atomic coordinate dependent or related quantities, such as velocities and forces, atom-atom distances, dihedral angles, ^3J-values, energies, size of the computational box, etc.

These two types of information are generally stored in separate files, since configurations change continuously during a simulation, whereas molecular topological data generally do not change. The naming convention for these files can be found in Chap. 4-**??**. Both types of files, *topological files* and *configurational files,* for a specific molecular system are related through the requirement that in both the sequence of the quantities is the same, e.g.

1. sequence of atoms
2. sequence of atom-atom distance restraints
3. sequence of dihedral angle restraints
4. sequence of ^3J-value restraints

This identity of sequence could be checked e.g. by comparing atom names occurring in topological files with those from the configurational files. However, in order to avoid dependence on naming conventions and to maintain maximum flexibility, this is not done in the GROMOS programs. When molecular information, such as residue numbers and names or atom sequence numbers or names, is present both in a topological file and in a configurational file of a molecular system, the program generally uses the data from the topological file and ignores the corresponding data on the configurational file.

In GROMOS files all the information is contained in *blocks*. For example, the molecular topology building block (`*.mtb`) file and the interaction function parameter (`*.ifp`) file, which are used for creating topologies, consist of blocks describing e.g. the physical constants used in GROMOS (`PHYSICALCONSTANTS` block), names of atom types recognized by GROMOS (`ATOMTYPENAME` block), force constants and lengths of various bonds in molecules (`BONDSTRETCHTYPECODE` block) and other atomic details and force-field parameters.

1.1.3. Summary of the exercise. All topological data mentioned above is used to construct topological building blocks. Many building blocks are chained together to form a final topology. If the system under consideration is charged, *counter ions* may be added, for which parameters are also included in the topological and force-field files (Sec. 2.1).

The second task is to obtain atomic cartesian coordinates of the system in GROMOS format. One can convert e.g. Protein Data Bank (PDB) files into GROMOS `*.cnf` files using programs available in the package (Sec. 2.2).

Further steps of an MD simulation involve minimizing the obtained spacial structure in vacuum, adding solvent molecules (Sec. 2.2.3) and counter ions (Sec. 2.2.4), minimizing the energy of the system and finally equlibration and thermalisation simulations (Sec. 2.3). After performing all these steps the actual MD simulation can be carried out. Input files used for minimisation, equilibration, thermalisation and molecular dynamics simulations also consist of blocks, which contain information about the simulation. For example, one has to specify the system by writing in the `SYSTEM` block how many solvent molecules there are. Simulation time, as well as integration time step, temperature and frequency of writing output files are all defined in the GROMOS `*.imd` files.

1.1.4. Calling the GROMOS programs. GROMOS programs are used with certain arguments, for example argument `@topo` would define the name of the topology file that should be used by the program called. One can write argument (`*.arg`) files which contain each argument and can be read by a program in the following way:

```
~> a_program @f file.arg
```

This saves a lot of time and makes it easier to repeat certain steps of your simulation, if needed.

1.2. Practical information

In this volume practical exercises are presented. You will perform a simulation of a penta-peptide in simple point charge (SPC) water with two chloride counter ions using the molecular dynamics software package GROMOS. Furthermore, you will analyse your results using GROMOS++ programs. Basic knowledge of the *UNIX* operating system is required in order to carry out the exercises. Further, you will want to look at molecular structures and you will want to plot the output of your analyses. On a standalone computer, or when using a virtual machine, you can use e.g. `pymol` or `vmd` to visualise molecules and the graphical data plotting software called `xmgrace`. You can find an online *xmgrace* manual on http://linux.die.net/man/1/xmgrace. See the README file of this tutorial at the `mms-teaching` server for short instructions on how to do these tasks there.

On the `mms-teaching` server, you will find the files that you need in the subfolder `peptide`. Just double click the folder on the left-hand side and start a `terminal` from the launcher. For later reference, you can also download the files from `https://www.gromos.net`.

CHAPTER 2

A practical exercise

2.1. 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 Vol. 3. 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`), 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`). When generating a molecular topology for the system of interest one should also address the protonation state of the molecular groups according to the pH and 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 for the protein and the DNA can be merged using the GROMOS++ program `com_top`.

2.1.1. Creating the topology for the penta-peptide. In this section you should build a molecular topology of a linear charged penta-peptide (Fig. 2.1) with water as a solvent, including two Cl^- counter ions.

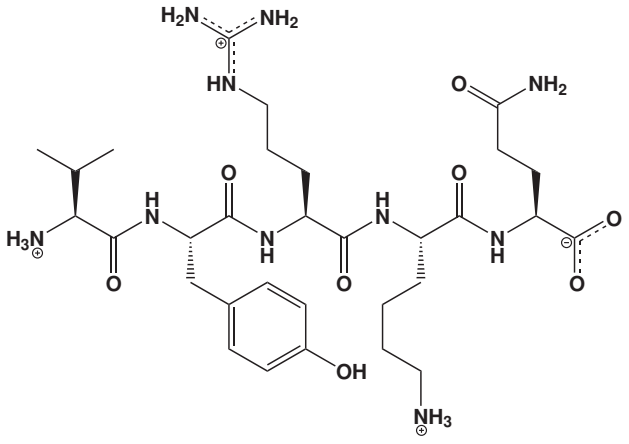

FIGURE 2.1. The topology of the penta-peptide

You need the following programs from the GROMOS simulation package:

GROMOS++: `make_top com_top check_top`

You need the following input files:

`54a7.mtb 54a7.ifp make_top_peptide.arg make_top_Cl.arg com_top_peptide_2Cl.arg`

The procedure will create the following output files:

`peptide_54a7.top Cl_54a7.top peptide_2Cl_54a7.top`

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

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

You will build the molecular topology file of the linear charged penta-peptide Val-Tyr-Arg-Lys-Gln by using the GROMOS++ program `make_top`. The input file `make_top_peptide.arg` is already prepared and contains the following data: under the argument `@build` the molecular topology building block file is specified. The argument `@param` specifies the interaction function parameter file. Under the argument `@seq` the sequence of the building blocks for the amino acid residues, including the amino and carboxy terminus is specified (NH3+ VAL TYR ARG LYSH GLN COO-). Notice that both termini are charged.

Hint: If CA is a CH2 united atom (for natural amimo acids, this is only glycine), the N-terminal patches GH3+ or GH2 should be used instead of NH3+ or NH2. For beta-peptides, the N-terminal patches AH3+ or AH2 should be used to precede a residue where CB is a CH2 united atom, while BH3+ or BH2 should be used where CB is a CH1 united atom.

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. See Chap. 3-4.

The argument `@solv` specifies the solvent. The molecular topology file for the penta-peptide, `peptide_54a7.top`, with water as a solvent can then be generated as

```
~/peptide/topo> make_top @f make_top_peptide.arg > peptide_54a7.top
```

The next step is to build a molecular topology file for a chloride ion using the GROMOS++ program `make_top` and the input file `make_top_Cl.arg`:

```
~/peptide/topo> make_top @f make_top_Cl.arg > Cl_54a7.top
```

Now we will combine the generated molecular topology files (`peptide_54a7.top` and `Cl_54a7.top`) into the molecular topology file `peptide_2Cl_54a7.top` using the GROMOS++ program `com_top`. The input file `com_top_peptide_2Cl.arg` is already prepared and it contains the following data: under the argument `@topo` the molecular topology files that you would like to combine 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_54a7.top`. The arguments `@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 and interaction function files for both topologies, this is not important for us and both numbers are set to 1. To run `com_top` type:

```
~/peptide/topo> com_top @f com_top_peptide_2Cl.arg > peptide_2Cl_54a7.top
```

The file `peptide_2Cl_54a7.top` contains the complete molecular topology of the linear charged penta-peptide including $2Cl^-$ counter ions with water as a solvent. Have a look at it and check whether it makes sense! To be sure that your topology is correct, you should always check your topology using the `check_top` program.

Hint: Use `check_top` with the additional arguments `@build` and `@param` for a more careful check of your topology against the force field. (see the `check_top_peptide.arg` file for an example).

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

2.2. Generating atom 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 naming 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 using X-ray diffraction data) and have to be generated using the GROMOS++ program `gch`. When the structure is determined using NMR data, the PDB structure often contains more hydrogen atoms than are needed for GROMOS, as in the GROMOS force field only polar and aromatic hydrogens are explicitly represented. Aliphatic hydrogens are non-existing due to the use of so-called united atoms. The aliphatic hydrogen and carbon atoms are merged to form united atoms which have their own parameters. If no atomic coordinates for the solute are available from experimental data, the coordinates have to be generated using molecular modeling software. 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 as well. When a simulation of a solute in solution is to be carried out, a (periodic) box (be it rectangular, triclinic or truncated octahedral) is 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.

2.2.1. Generating atomic Cartesian coordinates for the linear charged penta-peptide. You need the following programs from the GROMOS simulation package:

GROMOS++: `pdb2g96 gch frameout`

You need the following input files:

`peptide.pdb pdb2g96_peptide.arg gch_peptide.arg frameout_peptide.arg`

The procedure will create the following output files:

`pdb2g96_peptide.cnf gch_peptide.cnf gch_peptide.pdb`

Go into the subdirectory `coord`. Open the file `peptide.pdb` and check if the atom names match the names in the molecular topology file `peptide_2Cl_54a7.top`.

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 to the coordinate file according to the topological requirements. The input arguments of the `pdb2g96` program are self explanatory.

```
~/peptide/coord> pdb2g96 @f pdb2g96_peptide.arg > pdb2g96_peptide.cnf
```

Warning: When converting coordinate files from the Protein Data Bank to GROMOS format many difficulties may emerge. If you encounter problems using the `pdb2g96` program, have a look at Sec. 4-7.3. 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 the directory is an example of the PDB library file.

Have a look at the `pdb2g96_peptide.cnf` file. You will notice that the hydrogen atoms have been added to the coordinate file with the Cartesian coordinates being set to zero. In order to generate meaningful 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_54a7.top`, and the coordinate file, `pdb2g96_peptide.cnf`, have to be specified in the `gch` input file. The argument `@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.arg > gch_peptide.cnf
```

Using the GROMOS++ program frameout you can convert the coordinate file gch_peptide.cnf 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.arg
~/peptide/coord> mv FRAME_00001.pdb gch_peptide.pdb
~/peptide/coord> vmd gch_peptide.pdb
```

2.2.2. Energy minimisation of the penta-peptide. You need the following programs from the GROMOS simulation package:

MD++: md

You need the following input files:

```
peptide_54a7.top gch_peptide.cnf em_peptide.imd em_peptide.run
```

The procedure will create the following output files:

```
em_peptide.omd peptide_min.cnf
```

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

The MD++ input file em_peptide.imd contains the following blocks:

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

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

```
ENERGYMIN
#     NTEM    NCYC    DELE    DX0     DXM   NMIN   FLIM
         1       0     0.1   0.01    0.05   2000    0.0
END
```

The existence of the ENERGYMIN block means that the MD++ program will perform an energy minimisation (EM) run. The NTEM switch indicates which minimisation algorithm to be used. With NTEM = 1 we indicate that the steepest-descent algorithm (Sec. 2-11.2) is used. NCYC gives the number of steps before resetting of conjugate-gradient search direction in case we would use the conjugate gradient method (NTEM = 2). Using DELE the energy threshold (the difference in energy between two energy minimisation steps) for stopping the minimisation process (convergence) is specified. The initial step size and maximum step size is given in DX0 and DXM, respectively. Using FLIM the absolute value of the forces can be limited to a maximum value before the algorithm is applied (see also 4-95).

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

In the SYSTEM block you specify the number of solutes (NPM) and solvent (NSM) molecules. You only have one solute NPM = 1 and no solvent molecules NSM = 0 because you still did not add any solvent molecules to the configuration file and the peptide is still in vacuum. Otherwise you would have to tell MD++ how many solvent molecules you are using.

Hint: Note that the solute and solvent topologies contained in the topology file are multiplied by these factors to match the sequence of atoms in the configuration file. If you want to simulate a peptide with two counter ions you still have to specify NPM = 1. The peptide and the counter ions form *one solute*. As NPM > 1 is not supported by MD++ you should use the GROMOS++ program com_top to multiply the topology of a solute if required.

```
INITIALISE
#  NTIVEL  NTISHK  NTINHT  NTINHB  NTISHI  NTIRTC  NTICOM  NTISTI     IG    TEMPI
       0       0       0       0       1       0       0       0  210185  300.0
END
```

This block will be explained in more detail in section Sec. 2.3.

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

In the `STEP` block you specify the maximum number of steps (`NSTLIM`) for the energy minimisation. GROMOS will stop as soon as the energy changes less than `DELE` or this step number is reached. For an MD simulation `T` is the initial time and `DT` is the integration time step used.

```
BOUNDCOND
#      NTB    NDFMIN
         0         1
END
```

In the `BOUNDCOND` block you specify which periodic boundary conditions (PBC) you are going to use in the EM procedure. `NTB` = 0 defines a vacuum simulation: PBC are not applied. To indicate the truncated octahedron (t) PBC, `NTB` is set to -1, for rectangular (r) PBC `NTB` is 1, and for the triclinic (c) PBC `NTB` is 2. `NDFMIN` defines the number of degrees of freedom subtracted from the total number of degrees of freedom for the calculation of the temperature.

```
PRINTOUT
#     NTPR           NTPP
        10              0
END
```

With the `PRINTOUT` block you can specify how often (every `NTPR`th step) you are printing out the energies to the output file.

```
CONSTRAINT
#      NTC    NTCP    NTCP0(1)    NTCS    NTCS0(1)
         3       1      0.0001       1      0.0001
END
```

Bonds vibrate at high frequencies ($h\nu \gg k_B T$). Therefore, these vibrations are of quantum-mechanical nature. So constraining the bond lengths is a better approximation than treating them as classical harmonic oscillators. Constraining all bond lengths of the solute and solvent (`NTC=3`) allows the use of a rather large time step of 2 fs. In this example the constraints are imposed by the `SHAKE` algorithm for both solute (`NTCP=1`) and solvent (`NTCS=1`) with a tolerance of 0.0001. See 4-92 for more information.

```
 FORCE
# NTF(1..6): 0,1 determines terms used in force calculation
#            0: do not include terms
#            1: include terms
# NEGR: ABS(NEGR): number of energy groups
#            > 0: use energy groups
#            < 0: use energy and force groups
# NRE(1..NEGR): >= 1.0 last atom in each energy group
# NTF(1) NTF(2) NTF(3) NTF(4) NTF(5)    NTF(6)
# bonds    angles   improper dihedral electrostatic vdW
  0        1        1        1        1            1
# NEGR   NRE(1)   NRE(2)   ...     NRE(NEGR)
     1      71
END
```

In the `FORCE` block you tell MD++ which terms it should use for the energy and force calculation. For bond angles, improper dihedrals, torsional dihedrals and the non-bonded interactions the standard terms of the GROMOS force field are switched on (1). Because we are using bond-length constraints and the SHAKE algorithm, we have to switch off (0) the bond-stretching terms for the bonds involving hydrogen atoms and not involving hydrogen atoms..

In the last line of this block, the energy groups are defined. In general, we define one or more energy groups for every molecule, and one comprising all the solvent molecules. The first integer is the number of energy groups we want to use (in the present case we only have one energy group). The following numbers are the atom sequence numbers of the last atom of each energy group. By defining these energy groups we

tell MD++ to sum up the energies between the atoms within these groups and calculate the inter-group energies, which can be very useful.

Warning: Think very carefully about the definition of energy groups before running the simulation. Energies of energy groups can not be calculated from the trajectories in an efficient way. So, changing an energy-group definition will result in rerunning the simulation.

```
PAIRLIST
#      algorithm: standard (0) (gromos96 like pairlist)
#                 grid (1) (XX grid pairlist)
#      SIZE:     grid cell size (or auto = 0.5 * RCUTP)
#      TYPE:     chargegoup (0)(chargegroup based cutoff)
#                atomic (1)(atom based cutoff)
#
#   ALGORITHM  NSNB    RCUTP    RCUTL  SIZE  TYPE
          0     5      0.8      1.4   0.4     0
END
```

In the `PAIRLIST` block you specify which algorithm you will use for the pairlist generation. The cut-off used in the short-range pairlist construction is given by `RCUTP` and for GROMOS it is usually 0.8 nm. The cut-off used in the long-range interactions is given by `RCUTL` and for GROMOS it is usually 1.4 nm. The pairlist is generated every 5th (`NSNB`) step. `TYPE` specifies the type of the cut-off, whether it is based on the distance between charge-groups (0) or on the distance between atoms (1).

```
NONBONDED
# NLRELE   APPAK     RCRF    EPSRF    NSLFEXCL
      1     0.0      1.4        1         1
# NSHAPE  ASHAPE   NA2CLC  TOLA2  EPSLS
     -1      1.4       2  1e-10      0
# NKX   NKY   NKZ   KCUT
   10    10   10    100
# NGX  NGY   NGZ  NASORD NFDORD  NALIAS NSPORD
   32   32   32       3      2       3      4
# NQEVAL  FACCUR  NRDGRD  NWRGRD  NLRLJ   SLVDNS
  100000     1.6       0       0      0     33.3
END
```

In the `NONBONDED` block you specify using `NLRELE` which method for the evaluation of long-range electrostatic interactions is used. Since you will use the reaction-field method, the value of `NLRELE` should be equal to 1. The long-range electrostatic interactions are truncated beyond a certain cutoff (`RCUTL` in the `PAIRLIST` block). Beyond the reaction-field cut-off radius (`RCRF`) the electrostatic interactions are replaced by a static reaction field with a dielectric permittivity of `EPSRF`. `RCRF` and `RCUTL` should be identical. Because we are doing the energy minimisation in vacuo `EPSRF` is set to 1. With `NSLFEXCL` equal to 1, you include the contributions of excluded atoms to the electrostatic energy. The ionic strength of the continuum is set to 0 (`APPAK`). All other switches are not used for the reaction-field method. See 4-101 for more information.

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

```
~/peptide/min> ./em_peptide.run
```

Once the energy minimisation is finished, the file with the minimized coordinates, `peptide_min.cnf`, and the general output file, `em_peptide.omd`, that reports the progress of the minimisation, will be written out. Have a look at both files and check if the minimisation has finished successfully. Using the GROMOS++ program `frameout` you can again convert the coordinate file `peptide_min.cnf` into PDB format and view the new configuration using the *VMD* program.

2.2.3. Solvating the penta-peptide in a water box. You need the following programs from the GROMOS simulation package:

GROMOS++:	`sim_box`
MD++:	`md`

You need the following input files:

`peptide_54a7.top spc.cnf em_solvent.imd sim_box_peptide.arg sim_box_peptide.rpr sim_box_peptide.por em_solvent.run`

The procedure will create the following output files:

```
sim_box_peptide.cnf em_solvent.omd peptide_h2o.cnf
```

Now you can put the energy minimized penta-peptide in a box of solvent using the GROMOS++ program `sim_box` which can solvate the solute in a pre-equilibrated box of solvent molecules. Go to the 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 argument `@topo`, the resulting box shape under the argument `@pbc` (r-rectangular, t-truncated octahedron, c-triclinic), the coordinate file of the solute under the argument `@pos`, the coordinate file of the pre-equilibrated box of solvent molecules under the argument `@solvent`, minimum solute-to-wall distance under the argument `@minwall`, and the minimum solute-to-solvent distance under the argument `@thresh`. If you are using a rectangular box (`@pbc` r) it is recommended to use an additional argument, `@rotate`. With this additional argument the solute can be rotated (before solvating) such that the largest distance between any two solute atoms is directed along the z-axis, and the largest atom-atom distance in the xy-plane lies in the y-direction. An input file `sim_box_peptide.arg` is already prepared. To put the solvent box around the penta-peptide type:

```
~/peptide/box> sim_box @f sim_box_peptide.arg > sim_box_peptide.cnf
```

In order to relax the unfavorable atom-atom contacts between the solute and the solvent, energy minimisation of the solvent should be performed while keeping the solute positionally restrained (i.e. connecting the atom to a reference position by a spring). In order to do that two additional files, in which the positionally restrained atoms and the reference coordinates are specified, have to be generated from the coordinate file `sim_box_peptide.cnf`. Copy the coordinate file `sim_box_peptide.cnf` to `sim_box_peptide.por` and to `sim_box_peptide.rpr`. Open the new file `sim_box_peptide.por` with a text editor, write in the `TITLE` block the text "solute atoms to be positionally restrained", and delete all the coordinates of the solvent such that only the coordinates of the solute atoms are left. Then change the keyword `POSITION` at the beginning of the atom coordinate block of `sim_box_peptide.por` to `POSRESSPEC`. You now have a file containing a `POSRESSPEC` block which only contains the atoms of the peptide. With these changes the position restraints specification file `sim_box_peptide.por` should look like this:

```
TITLE
solute atoms to be positionally restrained
END
POSRESSPEC
    1 VAL  H1        1   0.508983407 -0.108220645 -0.056160171
    1 VAL  H2        2   0.478246738 -0.131824700  0.103013982
...
    5 GLN  O1       70  -0.389334812 -0.481578304 -0.248788696
    5 GLN  O2       71  -0.408991389 -0.269160831 -0.187628610
END
```

The preparation of the reference position file is even simpler as it is very similar to the coordinate file and only the `TITLE` block has to be changed and one block has to be renamed. In order to prepare the reference position file, open the new file `sim_box_peptide.rpr` in a text editor, write in the `TITLE` block the text "reference positions for restraining solute atoms", and change the block name `POSITION` to `REFPOSITION`.

Warning: When reading in a coordinate file with a `POSITION` block the first 24 characters (atom specification) are ignored. However, in a position restraints file each line of the `POSRESSPEC` block must have seven columns as the fourth to seventh column are read in by the program.

The MD++ input files for minimisation of the solvent around the penta-peptide, `em_solvent.imd`, and the script to run the minimisation, `em_solvent.run`, are already prepared. Nevertheless, open the `em_solvent.imd` and compare it with `em_peptide.imd`. You will notice that `em_solvent.imd` contains one input block more, the `POSITIONRES` block.

```
POSITIONRES
#    values for NTPOR
#    0: no position re(con)straining
#    1: use CPOR
#    2: use CPOR/ ATOMIC B-FACTORS
#    3: position constraining
#     NTPOR    NTPORB   NTPORS   CPOR
      1           1       0     25000
END
```

Using this block you are specifying that you want to restrain the positions of your solute molecule. The restraining is achieved by a harmonic force term with a force constant `CPOR`. The `NTPORB` indicates that the reference positions are read from a separate file, which is in our case `sim_box_peptide.rpr`. `NTPORS` equal to 0 prevents the program from changing the reference positions upon pressure scaling.

Hint: Notice and understand the differences in SYSTEM, BOUNDCOND and FORCE block between `em_peptide.imd` and `em_solvent.imd`.

Have a look at `em_solvent.run` and put in the correct paths to your files. Run the energy minimisation of the solvent interactively by typing

```
~/peptide/box> ./em_solvent.run
```

This will take a few minutes. Once the minimisation is finished, the new coordinate file, `peptide_h2o.cnf`, and the general output file `em_solvent.omd` will be written out.

2.2.4. Adding counter ions to the simulation box. You need the following programs from the GROMOS simulation package:

GROMOS++: `ion`

You need the following input files:

`peptide_54a7.top peptide_h2o.cnf ion_peptide.arg`

The procedure will create the following output files:

`peptide_2Cl_h2o.cnf`

In the next step, two chloride ions should be added to the box. Go to the 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/ion> ion @f ion_peptide.arg > peptide_2Cl_h2o.cnf
```

Convert the file `peptide_2Cl_h2o.cnf` into the PDB format using the GROMOS++ program `frameout` and check where the chloride ions have been placed. In Fig. 2.2 you can see how it should look like.

Hint: Have a look at the file `frameout_1.arg` in `~/peptide/ana/frameout/`. Two arguments have been added:

```
@pbc   r
@include  ALL
```

The argument `@pbc` specifies which periodic boundary condition was used (in our case rectangular - `r`) and that we want to gather the frames. The argument `include` specifies whether `frameout` should convert only the solute, which is the default option, only the solvent (`@include SOLVENT`), or all the atoms (`@include ALL`).

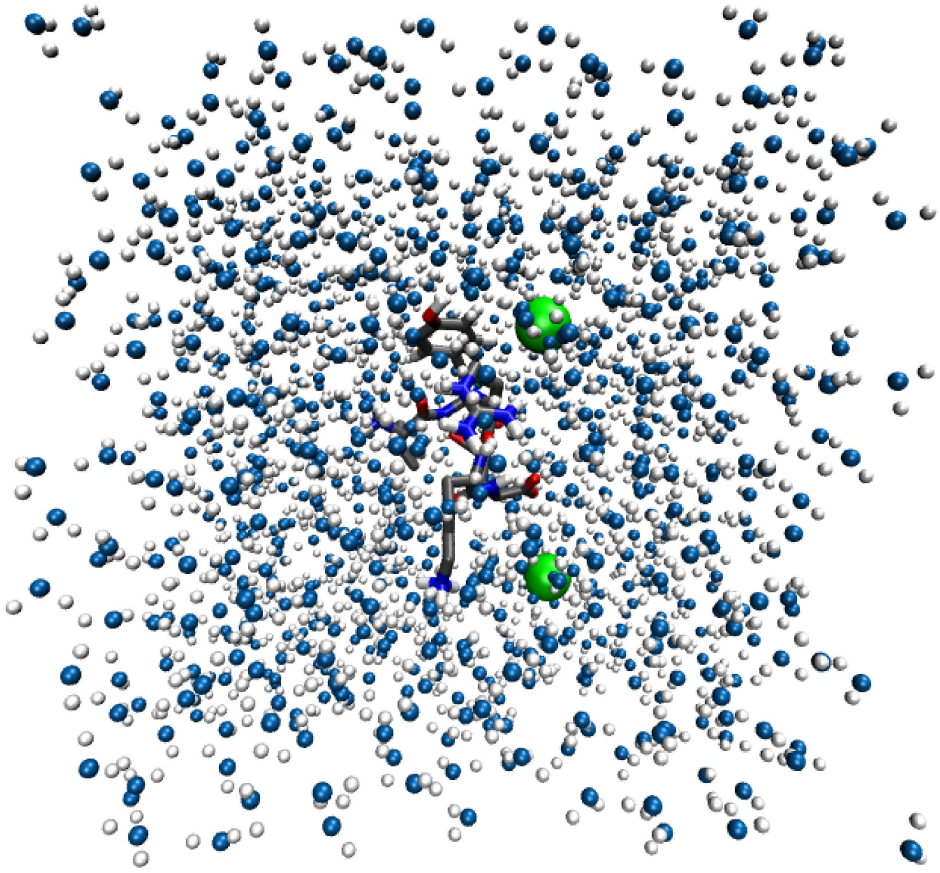

FIGURE 2.2. The penta-peptide and the two counter ions in a box of water.

2.3. Set-up and production simulation of the penta-peptide

2.3.1. Thermalisation and equilibration. You need the following programs from the GROMOS simulation package:

GROMOS++:	`mk_script ene_ana`
MD++:	`md`

You need the following input files:

`eq_mk_script.arg equilibration.imd equilibration.jobs mk_script.lib ene_ana.arg ene_ana.md++.lib`

The procedure will create the following output files:

`eq_peptide_*.run eq_peptide_*.imd eq_peptide_*.omd eq_peptide_*.cnf eq_peptide_*.tre.gz eq_peptide_*.trc.gz totkin.dat`

In the previous steps you have generated a topology and initial coordinates of your system. At this point, you have to generate initial velocities. In the process of thermalisation and equilibration, initial velocities are sampled from a Maxwell-Boltzmann distribution at a low temperature and the system is slowly heated up to the final production simulation 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 too much.

You already know a couple of things about job scripts. Because the set-up 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.

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

```
INITIALISE
#    NTIVEL  NTISHK  NTINHT   NTINHB   NTISHI NTIRTC    NTICOM  NTISTI     IG     TEMPI
         1       0       0        0        1      0         1       0  210185     60.0
END
```

In the INITIALISE block the NTIVEL tells GROMOS whether it should generate the initial velocities or read them from the configuration file. NTISHK is used to restore bond length constraints (SHAKE). NTINHT and NTINHB are only used for Nose-Hoover thermo- and barostats and can be ignored in our case. Every time an atom is leaving the periodic box and entering it from the opposite site this incident is recorded in the so-called lattice shift vectors. Using NTISHI we want to make sure that these vectors are initialised to zero. As you don't want to use roto-translational constraints NTIRTC can be ignored. NTICOM is used for initial removal of centre of mass motion. NTISTI is used to reset the stochastic integrals used in stochastic dynamics (SD) simulations. IG is the random number generator seed and TEMPI the initial temperature used to generate the Maxwell-Boltzmann distribution for generation of initial velocities. See also 4-96 for more information.

Hint: To make sure that you generate a trajectory that is different from the ones of your colleagues, you should set the value of IG to your Matrikel-number

---

In the SYSTEM block you need to replace NSM with the number of solvent molecules in your system (you will find it in peptide_2Cl_h2o.cnf file).

```
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. In this case you want to start at time 0 and you want to carry out a 20 ps simulation, because the time unit happens to be ps.

```
BOUNDCOND
#     NTB   NDFMIN
        1        3
END
```

As previously described, with the BOUNDCOND block you specify which PBC you will use. With NTB=1 you specify rectangular PBC.

```
MULTIBATH
# ALGORITHM:
#     weak-coupling(0):    use weak-coupling scheme
#     nose-hoover(1):      use Nose Hoover scheme
#     nose-hoover-chains(2): use Nose Hoover chains scheme
# NUM: number of chains in Nose Hoover chains scheme
#     !! only specify NUM when needed !!
# NBATHS: number of temperature baths to couple to
#         ALGORITHM
                0
#  NBATHS
        2
# TEMP0(1 ... NBATHS) TAU(1 ... NBATHS)
       60      0.1   60     0.1
#   DOFSET: number of distinguishable sets of d.o.f.
        2
# LAST(1 ... DOFSET) COMBATH(1 ... DOFSET) IRBATH(1 ... DOFSET)
      73        1        1     LSTATM       2        2
END
```

In our case we want to run the simulation at constant temperature. For this purpose, we have to add the MULTIBATH input block (see 4-99). First we specify which algorithm we will use. In this case we will use the weak-coupling scheme (ALGORITHM=0). How many temperature baths we want to couple to the system is specified by NBATHS. You can specify the temperature using the TEMP0 parameter. TAUT is the coupling time used in the weak-coupling method for this bath. DOFSET specifies the number of distiguishable sets of degrees of freedom. LAST is pointing to the last atom for the set of degrees of freedom; thus, you put the number of last atom of your system instead of LSTATM. COMBATH is the temperature bath to which we want to couple the centre of mass motion of this set of degrees of freedom IRBATH is the temperature bath

to which the internal and rotational degrees of freedom of this set of degrees of freedom are coupled. The temperatures in this block are modified by `mk_script`.

```
COMTRANSROT
#   NSCM
    1000
END
```

This block is needed to remove the centre of mass motion (translation and rotation). Without this block it can happen that all the kinetic energy is converted to centre of mass translation (flying ice cube problem). With `NSCM` we specify how often the center-of-mass (COM) motion is removed. If `NSCM` is < 0 translation and rotation motion are removed every $|$`NSCM`$|$th step. If `NSCM` is > 0 only translation motion is removed every `NSCM`th step.

```
COVALENTFORM
# NTBBH: 0,1 controls bond-stretching potential energy term
#       0: quartic form (default)
#       1: harmonic form
# NTBAH: 0,1 controls bond-angle bending potential energy term
#       0: cosine-harmonic (default)
#       1: harmonic
# NTBDN: 0,1 controls torsional dihedral angle potential energy term
#       0: arbitrary phase shifts (default)
#       1: phase shifts limited to 0 and 180 degrees.
#   NTBBH    NTBAH    NTBDN
       0        0        0
END
```

This block is needed to define which functional form we will use for bond-stretching (`NTBBH`), bond-angle bending (`NTBAH`) and for torsional dihedral (`NTBDN`). We just use the default options for all functional forms.

```
WRITETRAJ
# NTWSE = configuration selection parameter
# =0: write normal trajectory
# >0: chose mininimum energy for writing configurations
#     NTWX     NTWSE      NTWV      NTWF     NTWE      NTWG   NTWB
       100         0         0         0      100         0      0
END
```

MD++ produces a massive amount of data and it is impossible to store all the data it produces. The `WRITETRAJ` block meets this demand: Here you specify how often the coordinate trajectory (`NTWX`), the velocity trajectory (`NTWV`), the force trajectory (`NTWF`), the energy trajectory (`NTWE`), the free energy trajectory (`NTWG`) and the block averaged energy trajectory (`NTWB`) are written out. In the present case, we are only interested in the coordinates (`NTWX`) and energies (`NTWE`) and we write them every 100^{th} step. The second switch (`NTWSE`) defines selection criterion for trajectories: if `NTWSE` $= 0$ the normal coordinate trajectory will be written, or if `NTWSE` > 0 a minimum energy trajectory will be written.

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

```
PRINTOUT
#NTPR: print out energies, etc. every NTPR steps
#NTPP: =1 perform dihedral angle transition monitoring
#     NTPR        NTPP
       100           0
END
```

This block is very similar to the `WRITETRAJ` block but the information about the energies (`NTPR`) is printed to the output file. By giving `NTPP`, dihedral angle transitions are written to the special trajectory.

In the `FORCE` block you need to replace the `LSTATM` with the number of last atom of your system.

```
PAIRLIST
#       ALGORITHM: standard(0) (gromos96 like pairlist)
#                  grid(1) (MD++ grid pairlist)
#       SIZE:      grid cell size (or auto = 0.5 * RCUTP)
#       TYPE:      chargegroup(0) (chargegroup based cutoff)
#                      atomic(1) (atom based cutoff)
#
#       ALGORITHM      NSNB    RCUTP   RCUTL   SIZE    TYPE
                1         5      0.8     1.4    0.4    0
```

```
END
```

MD++ knows different algorithms for the generation of the pairlist, a list containing the atoms interacting with each other. Here, we use a 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 $\mathcal{O}\left({\mathcal{N}_a}^2\right)$ to $\mathcal{O}\left(\mathcal{N}_a\right)$. The pairlist is generated every 5th (`NSNB`) step. `RCUTP` and `RCUTL` are the cutoffs for the pairlist construction for the short-range and the long-range interactions.

```
POSITIONRES
#     values for NTR
#     0: no position re(con)straining
#     1: use CHO
#     2: use CHO/ ATOMIC B-FACTORS
#     3: position constraining
#      NTPOR    NTPORB    NTPORS    CPOR
       1            1         0   25000
END
```

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

Now you should understand the main blocks of the MD++ input files.

Hint: You can find further information on the GROMOS input file in Chap. 4-8.

Now it is time to have a look at `mk_script`. Open the input file `eq_mk_script.arg`. Here choose a system name `@sys`, describing your simulation. `@bin` points to the MD++ executable `md` 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, MD++ input file, initial coordinate file, position restraints and reference position file). `@template` respectively `mk_script.lib` is a configuration file for `mk_script`. Therein you can adapt `mk_script` to your local system or cluster. Because we are using MD++ you have to give the `@version md++` argument. 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`.

The joblist is already prepared for you. Therein you can change parameters in the MD++ 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 NTIVEL TEMPI TEMP0[1] TEMP0[2] COUPLE NTPOR CPOR subdir run_after
1      1      60.0     60.0    60.0      1     1  2.5E4   .      0
2      0      0.0     120.0   120.0      1     1  2.5E3   .      1
3      0      0.0     180.0   180.0      1     1  2.5E2   .      2
4      0      0.0     240.0   240.0      1     1  2.5E1   .      3
5      0      0.0     300.0   300.0      1     0  0.000   .      4
END
```

All the jobs have a certain ID which you can find in the first column. The following columns specify the parameters of the MD++ 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 `NTIVEL` parameter to one. In the further jobs we will read the velocities from a file, hence we set `NTIVEL` to 0. You can see that we are increasing the temperature for both baths by 60K at every new job (`TEMP0[1..2]`). Simultaneously, the force constant (`CPOR`) for the position restraints is decreased by an order of magnitude at every new job. Finally, in the last columns you specify in which subdirectory and in what order you want to run the jobs. The only thing that is missing to start the equilibration are the files containing the position restraints of the solute. To prepare it, copy the coordinate file `peptide_2Cl_h2o.cnf` from the `ion` directory to the local directory and open it with

a text editor and prepare the files `peptide_2Cl_h2o.por` and `peptide_2Cl_h2o.rpr` as you prepared the files `sim_box_peptide.por` and `sim_box_peptide.rpr` for the energy minimisation of the solvent (Sec. 2.2.3).

Now, run `mk_script`:

```
~/peptide/eq> mk_script @f eq_mk_script.arg
```

This creates five `eq_peptide_*.run` job scripts and the corresponding input files (`eq_peptide_*.imd`).

Warning: `mk_script` will not complain if the refpos file is not found! And the MD++ will crash if the `POSRESSPEC` block contains an empty line (only whitespace) at the end of the file!

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

```
~/peptide/eq> ./eq_peptide_1.run
```

Warning: Depending on your system's speed this will take some time. As we are using the simulation directory as the working directory there will be an error message after every job which can be ignored.

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

After the equilibration has finished, carry out some basic checks in the `eq/ana` directory.

```
~/peptide/eq/ana> ene_ana @f ene_ana.arg
```

This will give you a summary of some properties and create a file called `totkin.dat` which contains the time series of the total kinetic energy. You can plot this file with xmgrace, or directly on the `mms-teaching` server by following the instructions there.

```
~/peptide/eq/ana> xmgrace totkin.dat
```

You should be able to see that the kinetic energy is increasing at every new job (Fig. 2.3).

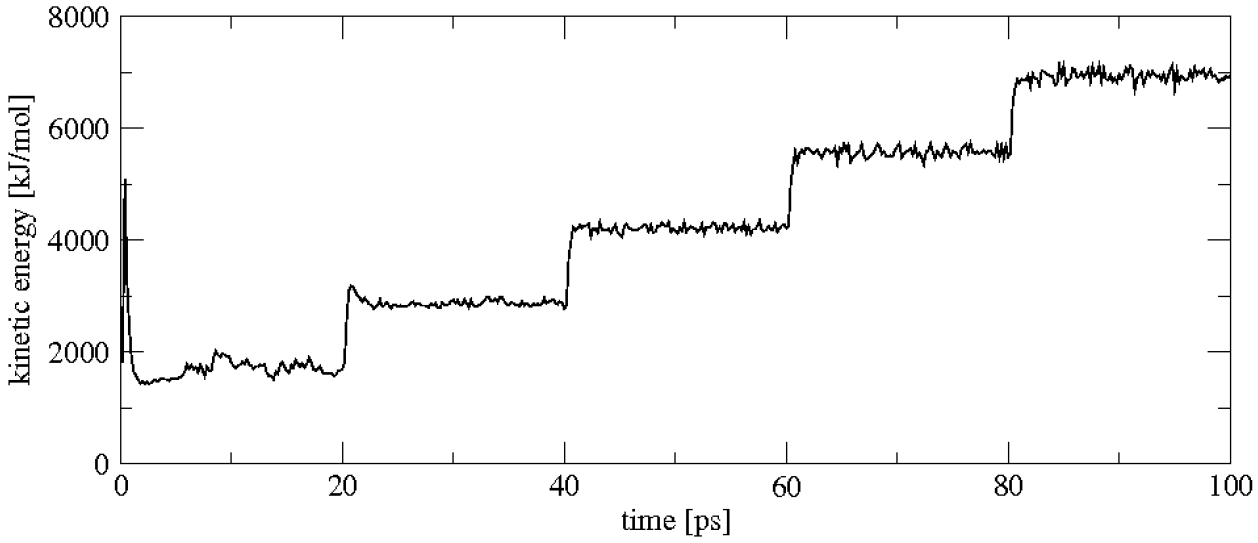

FIGURE 2.3. The kinetic energy during the equilibration.

2.3.2. Molecular dynamics sampling simulation. You need the following programs from the GROMOS simulation package:

GROMOS++:	`mk_script`
MD++:	`md`

You need the following input files:

```
md_mk_script.arg md.imd mk_script.lib
```

The procedure will create the following output files:

```
md_peptide_*.run md_peptide_*.imd md_peptide_*.omd md_peptide_*.cnf md_peptide_*.trc.gz md_peptide_*.tre.gz
```

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

Have a look at the input file `md.imd`. 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 `POSITIONRES` block was removed. We want to simulate under constant pressure rather than constant volume. For this purpose we have to add an additional block:

```
PRESSURESCALE
# COUPLE  SCALE     COMP     TAUP  VIRIAL
      2      1  0.0004575     0.5      2
#   SEMI (semianisotropic couplings: X, Y, Z)
    1 1 1
# PRES0(1...3,1...3)
 0.06102   0    0    0    0.06102  0   0   0   0.06102
END
```

In the `PRESSURESCALE` block we tell MD++ to calculate and scale the pressure by setting `COUPLE` to `2`. As the box should be isotropically scaled we set `SCALE` equal to 1. The weak-coupling method (Sec. 2-12.2.2) uses two additional parameters: `COMP` is the isothermal compressibility and `TAUP` is the coupling time. We are calculating the molecular virial (`VIRIAL` is equal to 2), so intramolecular forces don't contribute to the pressure. The next line is only used for semi-anisotropic pressure coupling and can be ignored in our case. Finally, we have to specify the reference pressure in a 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 (every 250th step only). In the `mk_script` input file (`md_mk_script.arg`) 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 with the first (1). Run `mk_script` to generate the job scripts.

```
~/peptide/md> mk_script @f md_mk_script.arg
```

This command creates 10 `md_peptide_*.run` job scripts and corresponding input files (`md_peptide_*.imd`). Now, you can submit the first script to the job control system (queue) or run it interactively on the command line.

```
~/peptide/md> ./md_peptide_1.run
```

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

2.4. Analysis of the penta-peptide trajectories

2.4.1. Analysis of the energy trajectory. GROMOS can write energies, free-energy λ-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 (`*.tre.gz`) and in the following you will learn how to extract time series and averages from these files.

You need the following programs from the GROMOS simulation package:

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

You need the following input files:

```
ene_ana_peptide.arg
```

The procedure will create the following output files:

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

Go to the direcory `ana`. In the subdirectory `ene_ana` have a look at the input file:

```
~/peptide/ana/ene_ana> cat ene_ana_peptide.arg
@en_files ../../md/md_peptide_1.tre.gz
          ../../md/md_peptide_2.tre.gz
...
@prop    totene totpot totkin solutemp2 solvtemp2 pressu
@topo    ../../topo/peptide_2Cl_54a7.top
@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`. One can specify an `ene_ana` library with the `@library` argument. Now you can run `ene_ana`:

```
~/peptide/ana/ene_ana> ene_ana @f ene_ana_peptide.arg > 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    -33001.249    108.441729    4.49116286
    totpot    -40082.2818   138.689821    4.23048814
    totkin     7081.03283    95.494846    1.17739268
 solutemp2     301.387955    27.0028093   0.357744616
 solvtemp2     303.959409    4.13812131   0.051210902
    pressu      1.9180672    305.759605    7.91642893
```

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 growing sizes extrapolating to infinite block size[1].

Warning: Sometimes the error estimates are `NaN` (not a number), which is due to the fact that we do not have enough values to calculate a meaningful error estimate.

Program `ene_ana` also produced a couple of time series files

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

Exercise: Have a look at these time series with `xmgrace` or on the `mms-teaching` server. Annotate the plots (axes, legends etc.).

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>less ../../md/md_peptide_2.tre.gz
TITLE
   GromosXX
   Automatically generated input file
```

```
   energy trajectory
END
TIMESTEP
            0  100.000000000
END
ENERGY03
# totals
  -3.304678446e+04
   7.179524549e+03
  -4.022630901e+04
   2.125140821e+02
   0.000000000e+00
   1.301350362e+02
   2.765269150e+01
   5.472635430e+01
   0.000000000e+00
  -4.043882310e+04
   6.127461522e+03
  -4.656628462e+04
   0.000000000e+00
   0.000000000e+00
   0.000000000e+00
   0.000000000e+00
```

As you can see there is a number of blocks, but you will not find `totene` anywhere. The `ene_ana.md++.lib` helps `ene_ana` to interpret the (free) energy trajectory files. This library file defines which value `ene_ana` should take from the trajectory if you ask for property `totene`. It can also calculate additional properties by performing simple mathematical operations on the values in the trajectories. [1]

Warning: The `ene_ana.md++.lib` file is very tightly coupled to the exact version of MD++ you use. The program checks if the file you specify matches the version of MD++ with which the energy trajectory was generated. If these do not match, you can find an updated version of `ene_ana.md++.lib` in the `data` directory of your MD++ installation.

The `ene_ana.md++.lib` file will look something like this:

```
~/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 35 1
    size NUM_BATHS
    subblock KINENER NUM_BATHS 3
    size NUM_ENERGY_GROUPS
    subblock BONDED NUM_ENERGY_GROUPS 5
    subblock NONBONDED matrix_NUM_ENERGY_GROUPS 4
    subblock SPECIAL NUM_ENERGY_GROUPS 9
    size NUM_EDS_STATES
```

[1] 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.

```
    subblock EDS NUM_EDS_STATES 3
  block VOLUMEPRESSURE03
    subblock MASS 1 1
    size NUM_BATHS
    subblock TEMPERATURE NUM_BATHS 4
    subblock VOLUME 10 1
    subblock PRESSURE 30 1
END
FRENERTRJ
# block definition for the free energy trajectory file.
# which is specified by the input flag fr_files of program ene_ana.
#
# syntax as for the ENERTRJ definition
#
# Following is the definition for a gromosXX free energy trajectory.
#
  block TIMESTEP
    subblock TIME 2 1
  block FREEENERDERIVS03
    subblock RLAM 1 1
    subblock FREEENER 35 1
    size NUM_BATHS
    subblock FREEKINENER NUM_BATHS 3
    size NUM_ENERGY_GROUPS
    subblock FREEBONDED NUM_ENERGY_GROUPS 5
    subblock FREENONBONDED matrix_NUM_ENERGY_GROUPS 4
    subblock FREESPECIAL NUM_ENERGY_GROUPS 9
    size NUM_EDS_STATES
    subblock FREEEDS NUM_EDS_STATES 3
END
VARIABLES
# Here you can define variables to be calculated by the program ene_ana
# In principal the program refers to the blocknames you have defined above,
# accessing individual element using array indices (one- or two-dimensional)
#
# Predefined as well is the Boltzmann constant (as BOLTZ = 0.00831441) and
# the MASS which (if not present in the energy trajectory) will be calculated
# from the topology (if inputflag @topo is given).
#
# Additional properties can be defined here as a direct mapping of a known
# property or as an expression of such properties. Make sure that variables
# and operators are always seperated by spaces. Multi-line expressions are
# allowed.
#
# Examples that work with the standard gromos96 definition are
#         given below and are actually standardly define if no library
#         file is specified.
time   = TIME[2]
dvdl   = FREEENER[3]
totene = ENER[1]
totkin = ENER[2]
totpot = ENER[3]
...
```

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 38 lines (`subblock ENER 38 1`).

Exercise: Use `ene_ana` to calculate the density of your system. You will have to specify a proper variable for the density behind the `@prop` argument. You can find out which variable from the `ene_ana.md++.lib` library file it should be. Plot the resulting density trajectory with `xmgrace` or on the `mms-teaching` server, annotate 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 in the `# nonbonded` block. In section Sec. 2.2.2 we defined four energy groups: the peptide, the first chloride ion, the second chloride ion and the water molecules. The `# nonbonded` matrix contains the following information:

```
# nonbonded
```

```
#   van der Waals      Coulomb  LSR LSK
# nonbonded
  -6.777453864e+01 -3.617661756e+02 0.000000000e+00 0.000000000e+00 # 1 - 1 peptide with peptide
  -2.605382239e-02 -8.326755384e+00 0.000000000e+00 0.000000000e+00 # 1 - 2 peptide with first Cl-
  -4.098207266e-01 -8.853119150e+01 0.000000000e+00 0.000000000e+00 # 1 - 3 peptide with second Cl-
  -3.822977361e+01 -2.549027033e+03 0.000000000e+00 0.000000000e+00 # 1 - 4 peptide with water
   0.000000000e+00 -7.382455923e+01 0.000000000e+00 0.000000000e+00 # 2 - 2 first Cl- with itself
  -2.759122936e-03 1.126804790e+00 0.000000000e+00 0.000000000e+00 # 2 - 3 first Cl- with second Cl-
   3.690831600e+01 -6.603813595e+02 0.000000000e+00 0.000000000e+00 # 2 - 4 first Cl- with water
   0.000000000e+00 -7.382455923e+01 0.000000000e+00 0.000000000e+00 # 3 - 3 second Cl- with itself
   3.924154608e+01 -5.281465864e+02 0.000000000e+00 0.000000000e+00 # 3 - 4 second Cl- with water
   6.258839244e+03 -4.214886612e+04 0.000000000e+00 0.000000000e+00 # 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[2] 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). Now you can specify these newly defined variables as properties (`@prop`) in the `ene_ana_peptide.arg` file. `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` or on the `mms-teaching` server, annotate and save them. The plot should look somewhat like Fig. 2.4.

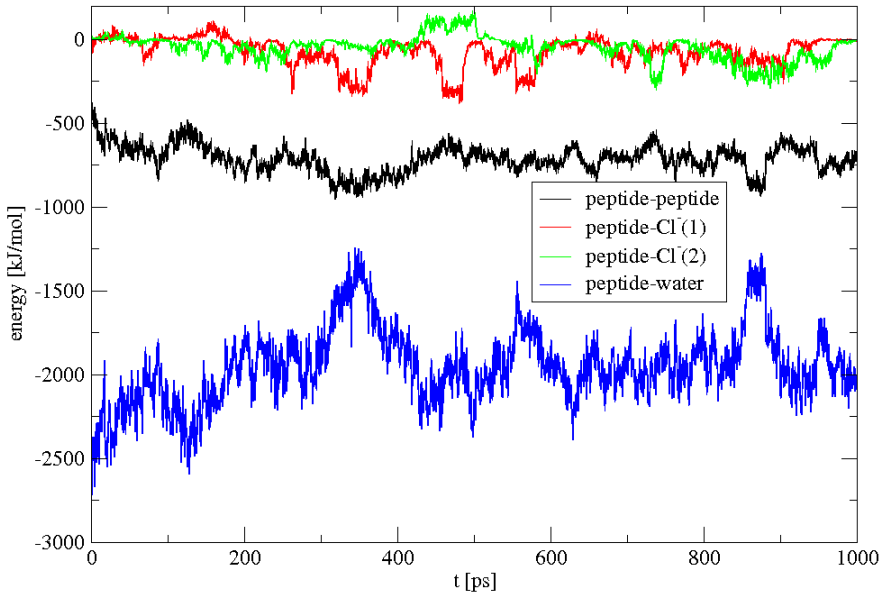

FIGURE 2.4. Time-series of energies.

2.4.2. Analysis of the coordinate trajectory.

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

You need the following programs from the GROMOS simulation package:

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

You need the following input files:

```
frameout_1.arg frameout_2.arg
```

[2] In LS simulations make sure to also add the third and forth column as these contain the real- and reciprocal-space contributions to the electrostatic energy.

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.arg
@topo          ../../topo/peptide_2Cl_54a7.top
@pbc           r
@include       ALL
@outformat     pdb
@traj          ../../ion/peptide_2Cl_h2o.cnf
```

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 is specified with the `@outformat` argument. Here, we use the PDB (Protein Data Bank) format. It is worth mentioning that other formats, including `cnf` the GROMOS coordinate format, are supported.

Hint: You can use frameout to create a gathered GROMOS frame by selecting `cnf` 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.arg
```

This command has generated a `FRAME_00001.pdb` file which can be opened with a molecular visualisation software (PDB viewer) like *VMD* or *pyMOL*.

Warning: Rename the PDB files generated by frameout or you might accidentally overwrite them.

It happens 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.arg
@topo          ../../topo/peptide_2Cl_54a7.top
@pbc           r
@include       SOLUTE
@outformat     pdb
@traj
../../md/md_peptide_1.trc.gz
../../md/md_peptide_2.trc.gz
...
@spec          SPEC
@frames        35 49
```

First, you need to add 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. Visualize these files as well as the previous frame and compare the structures.

Hint: You can use frameout to produce a movie that you can watch using `VMD` or `pymol` or on the `mms-teaching` server. Have a look at the files `frameout_movie.arg`. This will produce a single PDB file that contains all the configurations of the trajectory, after a roto-translational least-square fit based on the C_α atoms. Your favourite visualisation program can play this as a movie.

2.4.2.2. *Radial distribution function.* The radial 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 another type). The theoretical description is given in Sec. 5-4.48. GROMOS++ offers a tool that calculates the radial distribution function called `rdf`.

You need the following programs from the GROMOS simulation package:

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

You need the following input files:

`rdf_peptide_2CL.arg`

The procedure will create the following output files:

`rdf_peptide_2CL.out`

Go to the directory `rdf` and have a look at the input file:

```
~/peptide/ana/rdf> cat rdf_peptide_2CL.arg
@topo  ../../topo/peptide_2Cl_54a7.top
@pbc r
@centre a:CL
@with  s:OW
@cut   1.4
@grid  100
@traj  ../../md/md_peptide_1.trc.gz
       ../../md/md_peptide_2.trc.gz
...
```

The topology file is specified by `@topo`. With the option `@centre` you specify which atoms are supposed to be at the centre of your RDF and by `@with` which atoms should be the surrounding ones.

Hint: **The AtomSpecifier**

`a:CL` looks in all molecules for the atom named `CL` (chlorine ion)

`s:OW` looks in all solvent molecules for the atom named `OW` (water oxygen)

Have a look at the doxygen documentation of the *AtomSpecifier* and the *PropertySpecifier*.

With the parameter `@cut` you specify up to which distance you want to look at[3] and with `@grid` how many bins from 0 to `@cut` you want to have[4]. 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.arg > rdf_peptide_2CL.out
```

Have a look at the output. It should look similar to Fig. 2.5 where the first peak representing the first hydration shell is clearly visible meaning that the water molecules are organised 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.

Exercise: Have a look at the difference between the RDF around the first and the second chlorine atom with `xmgrace` or `matplotlib`. Label the plot (axes, legends etc.).

Hint: All you have to do to solve the exercise is to change the `@centre` in your input file. To find the atom specifier 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_54a7.top @gromosnum 72
```

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

---

[3] do not go over half the box size

[4] bigger number means finer resolution, but less statistics

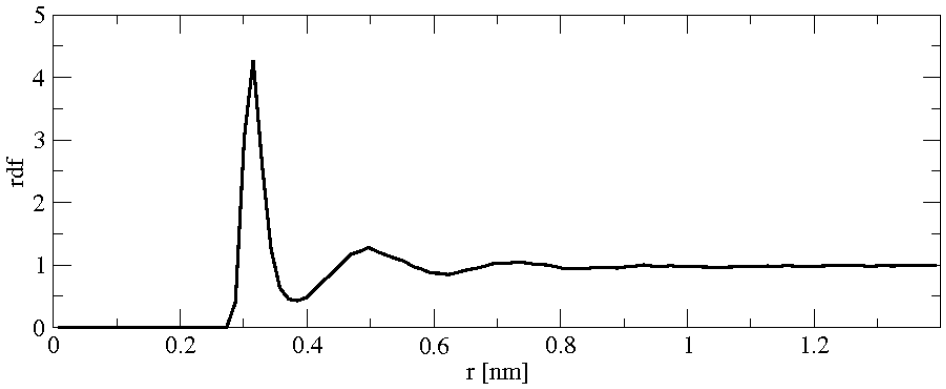

FIGURE 2.5. Cl-OW radial distribution function of chlorine ions in water

2.4.2.3. *Root-mean-square difference.* Atom-positional root-mean-square difference (RMSD) is a measurement of structural difference between two given conformations. For two conformations with the coordinates $\boldsymbol{r}$ and $\boldsymbol{r}^{\mathrm{ref}}$, the RMSD is given by

$$\mathrm{RMSD}(\boldsymbol{r}, \boldsymbol{r}^{\mathrm{ref}}) = \left(\frac{1}{\mathcal{N}_a} \sum_{i=1}^{\mathcal{N}_a} \left(\boldsymbol{r}_i - \boldsymbol{r}_i^{\mathrm{ref}} \right)^2 \right)^{\frac{1}{2}} \tag{2.1}$$

where $\boldsymbol{r}_i$ is the position of the i-th particle in the one configuration and $\boldsymbol{r}_i^{\mathrm{ref}}$ in the other, here called its reference position.

You need the following programs from the GROMOS simulation package:

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

You need the following input files:

`rmsd_peptide.arg`

The procedure will create the following output files:

`rmsd_peptide.out`

Go to the directory `rmsd` and have a look at the input file:

```
~/peptide/ana/rmsd> cat rmsd_peptide.arg
@topo  ../../topo/peptide_2Cl_54a7.top
@pbc   r
@atomsrmsd 1:CA,N,C
@ref   ../../eq/eq_peptide_5.cnf
@traj  ../../md/md_peptide_1.trc.gz
       ../../md/md_peptide_2.trc.gz
...
```

Again the topology is given by `@topo` and `@pbc` defines the periodic boundary condition and gathers the frames. In `@atomsrmsd` one gives the atomspecifier of the atoms of which one wants to calculate the RMSD compared to a reference structure `@ref`. Here we want to analyse the backbone of the peptide (C_α, N, C) as a function of time. With `@traj` the coordinate trajectories are specified.

Now calculate the RMSD using the GROMOS++ program `rmsd`

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

The output should look like Fig. 2.6, where you can see that at the beginning (t=0) the RMSD is zero as the structure in the beginning of the simulation is identical to the reference structure. The next few structures are nearly identical with the reference structure as well. Afterwards the RMSD increases as the structure is evolving away from the reference structure.

Exercise: Compare the RMSD of the backbone with the RMSD of the whole protein by plotting both in the same figure. 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.).

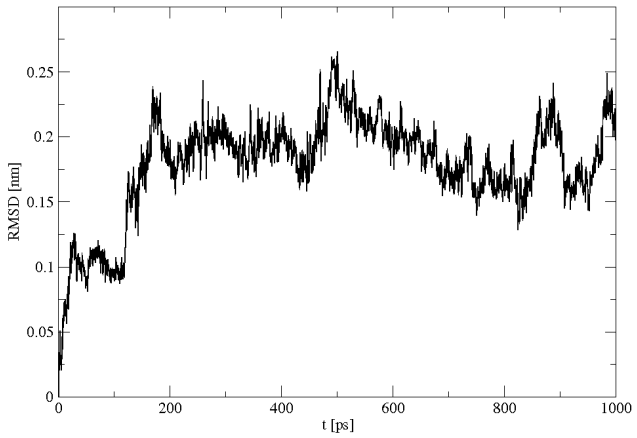

FIGURE 2.6. Atom-positional root-mean-square deviation of the backbone atoms of the penta-peptide from the starting (reference) structure.

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

2.4.2.4. *Root-mean-square fluctuation.* Atom-positional root-mean-square fluctuation RMSF gives fluctuations of an atom coordinate $\boldsymbol{r}_i$ around its mean and is given by :

$$\mathrm{RMSF}(\boldsymbol{r}_i) = \left(\frac{1}{N_t} \sum_{n=1}^{N_t} \left(\boldsymbol{r}_i(t_n) - \langle \boldsymbol{r}_i \rangle \right)^2 \right)^{\frac{1}{2}} \tag{2.2}$$

where $\boldsymbol{r}_i(t_n)$ is the position of atom i at time $t_n = n\Delta t$ and $\langle \boldsymbol{r}_i \rangle$ is its mean position.
You need the following programs from the GROMOS simulation package:

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

You need the following input files:

`rmsf_peptide.arg`

The procedure will create the following output files:

`rmsf_peptide.out`

Go to the directory `rmsf` and have a look at the input file:

```
~/peptide/ana/rmsf> cat rmsf_peptide.arg
@topo  ../../topo/peptide_2Cl_54a7.top
@pbc   r
@atomsfit 1:CA
@atomsrmsf 1:CA,N,C
@ref   ../../eq/eq_peptide_5.cnf
@traj  ../../md/md_peptide_1.trc.gz
       ../../md/md_peptide_2.trc.gz
...
```

Topology and periodic boundary conditions are given by `@topo` and `@pbc`, respectively. With `@atomsfit` one specifies which atoms are used to superimpose the structures (in order to remove translational and rotational changes in $\boldsymbol{r}_i$; just look at the intramolecular structural changes). `@ref` specifies the reference structure. The atoms for which the RMSFs are calculated are given by `@atomsrmsf` (here it is the backbone of the protein), and `@traj` indicates which coordinate trajectories are looked at.
Now you can run `rmsf`:

```
~/peptide/ana/rmsf> rmsf @f rmsf_peptide.arg > 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 peptide to see that the side chains move more than the backbone. Plot your output and label the axes.

2.4.2.5. *Time series of properties.* Often you are interested in the time change of a certain property. You can monitor the properties of your system using time series. In addition, you may want to compare a property of your simulation with an experimental value. In this case a time-average is calculated which can be compared to experimental data. You need the following programs from the GROMOS simulation package:

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

You need the following input files:

`tser_peptide.arg`

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 only its basic function is explained here. Go to the directory `tser` and have a look into the example file:

```
~/peptide/ana/tser> cat tser_peptide.arg
@topo     ../../topo/peptide_2Cl_54a7.top
@pbc      r
@traj
 ../../md/md_peptide_1.trc.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. 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 of the atom specifier after the `%` sign. `d%1:3,69` thus calculates the distance between atoms 3 and 69 in molecule 1 (our penta-peptide). `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 the 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 in the topology the chlorine atom is a molecule on it's own, you have to specify this with a molecule indicator (`2:` for the first chloride ion). Thus `2:1` denotes the first atom of the second molecule which is the first chloride ion.
3. The third property analysed is 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`). You may also use an equivalent form of specifying the angle: `a%1:70-71`.
4. Finally, a torsional angle (`t`) is calculated. The torsional angle is defined by four atoms in the atom specifier after the `%` sign. Torsional angles are an important property of protein backbones (Ramachandran map) and are called ϕ- and ψ-angles. Here we look at the torsional angle between the H-N bond of lysine (atoms `47,46`) and the C_α-C_β (`48,49`) bond of the lysine residue.

Call `tser` and redirect its output:

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

Now you can 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.

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

Hint: If you are simulating an elongated molecule, you should first gather the solute with `frameout` and then analyse the results using `tser` and vacuum boundary conditions (`@pbc v`). This way you avoid flawed results due to the nearest image distance between the specified atoms.

If everything worked out, your arginine-chloride distance should look like shown in Fig. 2.7.

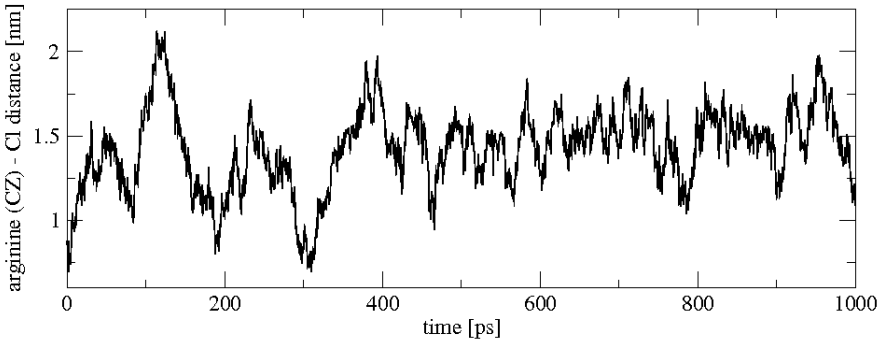

FIGURE 2.7. Time-series of the arginine (CZ) - chloride (1) distance.

Exercise: The distance between the two chloride atoms is an interesting property of your system. Create a time series and plot it with `xmgrace` or on the `mms-teaching` server. Annotate the graph.

2.4.2.6. *Hydrogen-bond analysis.* Hydrogen bonds are important intra- and intermolecular interactions. For example, the conservation of the genetic information of a cell is based on hydrogen bonding between bases in nucleic acids (Watson-Crick base pairing). The secondary structure of proteins is formed by hydrogen bonds between backbone amide hydrogens and oxygens. Thus, it is useful to analyse the hydrogen bonds of the penta-peptide in detail. You need the following programs from the GROMOS simulation package:

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

You need the following input files:

`hbond_peptide.arg`

The procedure will create the following output files:

`hbond_peptide.out Hbond_2c_time_index.out Hbond_2c_time_numHb.out`

The GROMOS++ program `hbond` is a convenient tool for hydrogen bonds analysis. Go to the directory `hbond` and have a look at the input file:

```
~/peptide/ana/hbond> cat hbond_peptide.arg
@topo           ../../topo/peptide_2Cl_54a7.top
@pbc           r
@Hbparas       0.25 135
@massfile      mass.file
@AcceptorAtomsA 1:a
@AcceptorAtomsB 1:a
@DonorAtomsA   1:a
@DonorAtomsB   1:a
@traj
../../md/md_peptide_1.trc.gz
../../md/md_peptide_2.trc.gz
...
```

With the first two arguments you are already familiar from previous analyses. The `hbond` program 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 a hydrogen-bond calculation: a maximum distance between the hydrogen and the acceptor and a minimum angle between donor, hydrogen and acceptor atoms. 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 hydrogen bonds are calculated. In this case we are interested in intramolecular hydrogen bonds, so both of the groups consist of atoms of the peptide only. We thus have to set the `@AcceptorAtomsA` and the `@DonorAtomsB` arguments both to an atom specifier of the whole peptide (`1:a`). The execution of the `hbond` program

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

produces three output files: `hbond_peptide.out` (the redirected standard output), `Hbond_2c_time_index.out` and `Hbond_2c_time_numHb.out`

1. The `hbond_peptide.out` file contains a summary table. In this table all the hydrogen bonds are numbered (first column). In the second column you can find the molecule and residue numbers of the donor and acceptor, followed by the numbers and names of the atom involved in the hydrogen bond. In the next two columns the geometric 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 `Hbond_2c_time_numHb.out` file.
3. The `Hbond_2c_time_index.out` file contains times series of all hydrogen bonds found in the summary table. The first column is the time, and in the second column is the ID number found in the first column of the summary file of the hydrogen bond occurring at this time. `Hbts.out` is usually filtered before plotting. A plot of the unfiltered file is shown in Fig. 2.8

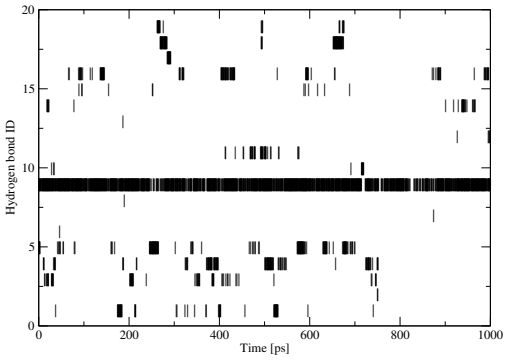

FIGURE 2.8. Time series of hydrogen bonds.

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

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

You need the following programs from the GROMOS simulation package:

GROMOS++: `rmsdmat cluster postcluster`

You need the following input files:

`rmsdmat.arg cluster.arg postcluster.arg`

The procedure will create the following output files:

`RMSDMAT.dat cluster_structures.dat cluster_ts.dat postcluster.out cluster*.trc cluster*.cms cluster*.out`

The first program that we will use is the GROMOS++ program `rmsdmat`. This program calculates the atom-positional root-mean-square deviation between all pairs of structures in a given trajectory file. The RMSD matrix can be written out in a readable form, or in a binary format[5] in order to save disk space.

Go to the directory `cluster` and have a look at the input file `rmsdmat.arg`.

```
~/peptide/ana/cluster> cat rmsdmat.arg
@topo  ../../topo/peptide_2Cl_54a7.top
@pbc   r
@atomsfit 1:CA,N,C
@stride 1
@human
@precision 6
@traj  ../../md/md_peptide_1.trc.gz
       ../../md/md_peptide_2.trc.gz
...
```

Topology and periodic boundary conditions are given by `@topo` and `@pbc`, respectively. With `@atomsfit` one specifies atom names which are to be used for least-squares fitting of the translational and rotational positions for the calculation of the RMSD. Here we only took the backbone atoms (N, C and C_α). With `@stride` one can specify a selection of structures in the trajectory file to be considered in this analysis. As we don't want to skip frames `@stride` is set to one. With `@human` the output file will be written in a readable form, otherwise it will be written in a binary form. With the argument `@precision` we specify the number of digits in the output matrix. With the argument `@traj` we specify the coordinate trajectories which will be analysed.

In the input for the `rmsdmat` program one can optionally specify the reference structure with respect to which the structures are fitted (`@ref ../../eq/eq_peptide_5.cnf`). Furthermore, in case the atoms differ from those on which the fit is performed this can be additionally 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 is taken as the reference structure. By specifying a reference structure, one allows the program `cluster` (see below) to perform a forced clustering, where the first cluster contains the reference structure.

```
~/peptide/ana/cluster> rmsdmat @f rmsdmat.arg
```

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

Now we will use the GROMOS++ program `cluster`. This program performs a conformational clustering based on a similarity matrix, such as calculated in program `rmsdmat`. The clustering algorithm is described in[2]. In this program a cut-off can be specified such that the structure pairs with RMSD values smaller than this cutoff are considered as structural neighbors. The structure with the highest number of neighbors is considered as the central member of the cluster of similar structures.

Have a look at the input file `cluster.arg`:

```
~/peptide/ana/cluster> cat cluster.arg
@rmsdmat RMSDMAT.dat
@human
@precision 6
@cutoff 0.06
@time 0 0.5
#@maxstruct
```

As previously mentioned `RMSDMAT.dat`, the output file of `rmsdmat`, is used here as an input file. With `@human` the program can use the readable matrix file. With the argument `@precision` we specify the number of digits in the input matrix. With the argument `@cutoff` we can specify the similarity criterion for two structures. In this case we are not reading a trajectory and thus we have to tell the program time information using the `@time` argument. The first argument (0) is the starting time and the second argument is the increment in time per element contained in the RMSD matrix. With `@maxstruct` 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 the RMSD matrix with a reference structure, the reference structure used in this program is the first structure. This additional argument should look as follows: `@force 0`.

```
~/peptide/ana/cluster> cluster @f cluster.arg > cluster.out
```

[5] This is crucial for large trajectories.

The output consists of two files: `cluster_structures.dat` and `cluster_ts.dat`.

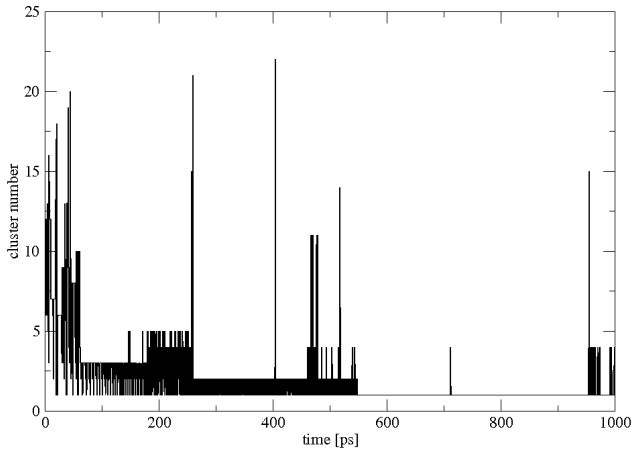

FIGURE 2.9. Time-series of the clusters.

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

In `cluster_ts.dat` occurrences of specific clusters at specific times are listed (the time series of each cluster).

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

Have a look at the input file `postcluster.arg`:

```
~/peptide/ana/cluster> cat postcluster.arg
@topo          ../../topo/peptide_2Cl_54a7.top
@cluster_struct cluster_structures.dat
@cluster_ts    cluster_ts.dat
@clusters      1-8
@lifetime      2
@traj          ../../md/md_peptide_1.trc.gz
               ../../md/md_peptide_2.trc.gz
...
```

Among the analyses 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`. It defines a number of subsequent structures that are significantly different from the structures of the cluster of interest. If this limit is reached, a switch to another cluster will occur. For example, if we specify a lifetime limit equal to 2, this means that at least two subsequent structures of a cluster that differ from the cluster of interest have to occur in order to detect a cluster transition.

```
~/peptide/ana/cluster> postcluster @f postcluster.arg > postcluster.out
```

The output of the `postcluster` program consists of trajectory files `*.trj` as well as `*.cms` files for each cluster. Thus, `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. The resulting trajectories can further be used in any other analysis programs.[6]

2.4.2.8. *NMR analysis.* To compare simulated with experimental NMR data one may calculate NOE (Nuclear Overhauser Effect) distance upper bound violations and ^3J-coupling constant values and compare them with the corresponding experimental values.

You need the following programs from the GROMOS simulation package:

GROMOS++: `prep_noe noe post_noe jval`

You need the following input files:

`prep_noe.arg noe.arg post_noe.arg jval.arg jval.jvr prep.noe noecor.wuthrich noelib.54a7`

[6] You should compress them using `gzip` first.

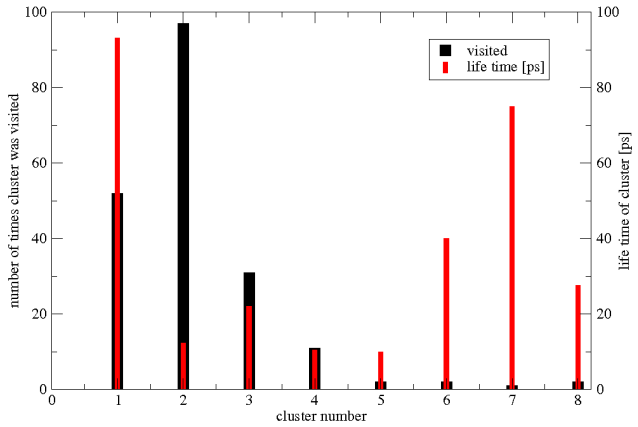

FIGURE 2.10. The number of visits and lifetimes of the clusters.

The procedure will create the following output files:

```
noe.filter prep_noe.out noe.dsr noe_peptide.out noets.out post_noe.out jval.out
```

Go into the subdirectory called `noe`. You will perform three steps in order to analyse the NOE distance upper bound violations. First you need to prepare the bonds derived from experiment such that GROMOS will understand them. Take a look at the argument file `prep_noe.arg`:

```
~/peptide/ana/noe> cat prep_noe.arg
@topo  ../../topo/peptide_2Cl_54a7.top
@title NOE_specification_file
@filter 1000
@factor 10
@noe prep.noe
@lib noelib.54a7
@parsetype 1
@correction noecor.wuthrich
```

The topology file is given by the argument `@topo`. With `@title` one specifies the title in the `prep_noe` output. The `@filter` argument determines the upper limit of the NOE distance bounds which should be considered. By setting `@filter` to large value (1000 nm) we make sure that all distances will be taken into account. Most of the NMR experimental data is reported in Angstrom units [Å], whereas using the GROMOS force field the standard length unit is nanometer [nm]. Thus we have to give the program a `@factor` by which it has to scale all the NOE distance bounds prior to any other calculation. The input file in an *X-PLOR* like format (and units) with all NOE distances between the atoms of interest is specified by the argument `@noe`. Take a look at the `prep.noe` file in a text editor. It contains the atom number and the name of each hydrogen as it is used in structure determination and refinement using *X-PLOR*. In GROMOS the names used for the hydrogens might differ, thus we should specify a library file which converts the *X-PLOR* names into ones that the `noe` program can read. This is done with the argument `@lib`. The last three columns (6, 7 and 8) in the *X-PLOR* file stand for the upper bounds derived from the NOE intensities. There are three ways of parsing these columns. For that purpose program `prep_noe` uses `@parsetype` argument:

`@parsetype=1` : the first of the three columns is taken as the upper bound
`@parsetype=2` : (default value) the upper bound will be the sum of the first and third column
`@parsetype=3` : the upper bound is equal to the difference between the first and second column

If you set `@parsetype` to either `2` or `3`, you have to also use an argument called `@action`, which applies a correction to the upper bound value. The argument `@correction` specifies a file containing the information about types of correction that should be taken into account by `prep_noe`. NOE bound corrections are used in case of non-specific assignment of individual hydrogen atoms. Several methods have been proposed to define pseudo-atoms. Here we use the 'centre average' approach. The correction is two-fold: a pseudo-atom correction and a multiplicity correction. The former aims to solve the dilemma when several hydrogen atoms of group I interact with a second hydrogen atom, labelled S, and the interactions between the various pairs

of hydrogen atoms are practically indistinguishable. The solution to this problem is to define a pseudo atom, labelled Q, at the mean position of the hydrogen atoms of group I.

A sample of the correction file is shown below:

```
~/peptide/ana/noe> cat noecor.wuthrich
TITLE
NOE correction file containing the multiplicity corrections and pseudo
atom corrections in nm as described in
* Wuethrich, K.; Billeter, M. and Braun, W.: J. Mol. Biol. 169:949-961
  (1983)
* Wuethrich, K.: NMR of Protein and Nucleic Acids. John Wiley, New York
  (1986)
END
NOECORGROMOS
# NTPAC: NOE type to which the pseudo-atom correction applies
# NSPAC: NOE suptype to which the pseudo-atom correction applies
#       (set to 0 if no subtype defined)
# FTPAC: Distance of the pseudo-atom correction
#
# Possible combinations of NOE types/subtypes:
# NTPAC NSPAC NOE type
#    -1     1  flipping aromatic ring
#    -1     2  unassigned NH2 group
#     3     0  non-stereospecific aliphatic CH2 group
#     5     0  single CH3 group
#     6     0  non-stereospecific (CH3)2 group (isopropyl)
#     7     0  non-stereospecific (CH3)3 group (tert-butyl)
# NTPAC NSPAC FTPAC
    3    0    0.10000000
    5    0    0.15000000
    6    0    0.29000000
   -1    1    0.20000000
   -1    2    0.10000000
END
MULTIPLICITY
# NTMPC: NOE type to which the multiplicity correction applies
# NSMPC: NOE suptype to which the multiplicity correction applies
# FTMPC: Factor for the multiplicity correction
#
# Possible combinations of NOE types/subtypes:
# NTPAC NSPAC NOE type
#    -1     1  flipping aromatic ring
#    -1     2  unassigned NH2 group
#     3     0  non-stereospecific aliphatic CH2 group
#     5     0  single CH3 group
#     6     0  non-stereospecific (CH3)2 group (isopropyl)
#     7     0  non-stereospecific (CH3)3 group (tert-butyl)
# NTMPC NSMPC NTMPC
    3    0    1.00000000
    5    0    1.00000000
    6    0    1.00000000
   -1    1    1.00000000
   -1    2    1.00000000
END
```

The execution of the `prep_noe` program:

```
~/peptide/ana/noe> prep_noe @f prep_noe.arg > prep_noe.out
```

produces three output files: `prep_noe.out` (the redirected standard output), `noe.filter` and `noe.dsr`. Now look at the argument file `noe.arg` for the `noe` program:

```
~/peptide/ana/noe> cat noe.arg
@topo  ../../topo/peptide_2Cl_54a7.top
@pbc   r
@noe   prep_noe.out
@traj  ../../md/md_peptide_1.trc.gz
       ../../md/md_peptide_2.trc.gz
...
```

Topology and periodic boundary conditions are given by `@topo` and `@pbc`, respectively. Output from the previous program `prep_noe.out` is used here as the input file. Before executing the `noe` program open the `prep_noe.out` file and compare it with the `prep.noe`. With the argument `@traj` we specify the trajectories which will be analysed.

The execution of the `noe` program:

```
~/peptide/ana/noe> noe @f noe.arg > noe.out
```

produces the file `noe.out`. This file already contains the NOE violation information in a computer readable format. To make it more human readable you have to process it using the `post_noe` program. Now open the argument file for the `post_noe` program:

```
~/peptide/ana/noe> cat post_noe.arg
@topo        ../../topo/peptide_2Cl_54a7.top
@noe         prep_noe.out
@noeoutput   noe.out
@filter      noe.filter
@averaging   6
@distribution 0.05
```

The topology is given by `@topo`. The three following arguments `@noe`, `@noeoutput`, and `@filter` are output files of the `prep_noe` and `noe` programs. The `@noe` and `@noeoutput` arguments serve here as input files, whereas the `@filter` filters are the NOE distances which we will not use. With `@averaging 6` one specifies which averaging should be used. Here we are using the r^{-6} averaging. The argument `@distribution` indicates the binsize in which the final data will be separated.

The execution of the `post_noe` program:

```
~/peptide/ana/noe> post_noe @f post_noe.arg > post_noe.out
```

produces one output file: `post_noe.out`

The next analysis is ^3J-coupling constant analysis. In GROMOS, the ^3J-coupling constant can be calculated using program `jval`. It uses the so-called Karplus relation to relate the ^3J-coupling constant to the local molecular structure or torsional angle.

$$^3J(H_N, H_{\alpha/\beta}) = a\cos^2\zeta_n + b\cos\zeta_n + c \tag{2.3}$$

where ζ_n is the dihedral angle between the planes defined by the atoms (H, N, $C_{\alpha/\beta}$) and the atoms (N, $C_{\alpha/\beta}$, $H_{\alpha/\beta}$). In our calculations, we use parameters a, b, c equal to 6.4 Hz, -1.4 Hz and 1.9 Hz, respectively, which were optimized by Wüthrich et al.[3]

We will use the `jval` program to calculate the ^3J-coupling constant. Go into a subdirectory called `jval`.

Before we can calculate the ^3J-coupling constants, we need to define the torsional angle(s) that will be used for the calculation. For this we need a ^3J-coupling constant restraints file. In this example, as there are no experimental ^3J-values reported for this peptide, we will use hypothetical ^3J-coupling constants. The ^3J-coupling constant restraints file `jval.jvr` is already prepared for you and is shown below.

```
TITLE
penta-peptide, Val-Tyr-Arg-Lys-Gln, linear,
J-coupling constant restraints, hypothetical
END
JVALRESSPEC
# IPJV, JPJV, KPJV, LPJV:
#      atom sequence numbers of the real atoms defining the dihedral
#      angle that is related to the restrained J-value
# WJVR: individual 3J-value restraint weight factor by which
#      the restraining term for each 3J-value may be multiplied
# PJR0: the experimental or reference 3J-value. In case of a full-harmonic
#      3J-value restraint (NHJV = 0), it is the minimum-energy 3J-value;
#      in the case of an attractive or repulsive half-harmonic
#      3J-value restraint (NHJV = +- 1), it is the upper or lower bound,
#      respectively, beyond which the restraining force becomes non-zero.
# PSJR: phase shift or difference between the dihedral angle formed
#      by the possibly non-existing H-atoms defining the J-coupling
#      and the dihedral angle i-j-k-l formed by the real atoms
#      present in the simulation (in degrees)
# AJV, BJV, CJV:
#      Karplus parameters a, b and c for the J-coupling constant
#      expressed as a function of the dihedral angle
# NHJV: type of J-value restraint
#       0: full harmonic [recommended]
#      -1: half-harmonic repulsive
#      +1: half-harmonic attractive
#
#IPJV JPJV KPJV LPJV  WJVR  PJR0   PSJR    AJV    BJV    CJV   NHJV
    9   11   13   27    1.0   7.3  -60.0   6.4   -1.4   1.9    0.0
   27   29   31   44    1.0   8.6  -60.0   6.4   -1.4   1.9    0.0
```

```
   29   31   32   33    1.0   7.4 -120.0  9.5   -1.6  1.8    0.0
   29   31   32   33    1.0   5.7    0.0  9.5   -1.6  1.8    0.0
   44   46   48   57    1.0   7.4  -60.0  6.4   -1.4  1.9    0.0
END
```

The first four columns describe the dihedral angle η_n from which the ^3J-value is derived using the Karplus relation. The weight factor in the fifth column is only used for restraining and is ignored in this case. The sixth column is used as reference ^3J-value in order to calculate the deviation from it. As the measured ^3J-coupling constant may arise from a torsional angle ζ_n whose atoms, such as aliphatic hydrogens, are not explicilty represented in the GROMOS force field, the torsional angle ζ_n has to be related to the torsional angle η_n by a phase shift $\delta_n = \zeta_n - \eta_n$. This phase shift is read from the seventh column. In the following three columns you can specify different parameters for the Karplus relation for every individual ^3J-coupling constant. Finally in the last column the type of the ^3J-value restraining is specified. For more detailed information you can refer to Sec. 4-3.7.

Have a look at the input file `jval.arg`:

```
~/peptide/ana/jval> cat jval.arg
@topo          ../../topo/peptide_2Cl_54a7.top
@pbc           r
@jval          jval.jvr
@traj
../../md/md_peptide_1.trc.gz
../../md/md_peptide_2.trc.gz
...
```

You should be already familiar with the first three arguments from previous analyses. The argument `@jval` determines the file which contains the torsional angle specification corresponding to the specific ^3J-coupling constant.

The execution of the `jval` program:

```
~/peptide/ana/jval> jval @f jval.arg > jval.out
```

produces the output file `jval.out`. In the first columns it contains the information that we gave in the `jval.jvr` file, and the last five columns give the information about our calculated ^3J-coupling constants.

Hint: You can plot the observed vs. the calculate ^3J-values by `xmgrace -block jval.out -settype xydy -bxy 17:20:21`.

CHAPTER 3

Constructing a new building block

3.1. Using program prep_bb

As mentioned in Chap. 7-1 there might be cases where the needed molecular topology building block is not available in the standard distribution. The following task is to construct a new building block. If you do not need to create a new building block for your project, you do not need to do this part of the tutorial.

You need the following programs from the GROMOS simulation package:

GROMOS++: `prep_bb`

You need the following input files:

`54a7.mtb 54a7.ifp prep_bb_DTT.arg spec_DTT.dat`

The procedure will create the following output files:

`BUILDING.out`

When creating a new building block it is highly recommended to first prepare pictures containing all the topological information. Draw your molecule on paper taking the united atom approach into account. Number the atoms sequentially, thinking about how the charge groups will be defined. By analogy to the existing GROMOS building blocks in Chap. 3-4 define the integer atom codes, mass codes and charges of all the atoms as they are described in Chap. 3-3. Define the charge groups making sure they are neutral unless the molecule itself is charged. By analogy to the existing GROMOS building blocks define also bond types, bond angle types, dihedral angle types and improper dihedral angle types. Once you have the picture ready use the GROMOS++ program `prep_bb`, which will help you create the building block.

In this exercise you will generate a building block for 2,3-dihydroxy-1,4-dithiobutane (DTT), a small detergent molecule. The pictures with the parameters for DTT are already prepared (Fig. 3.1).

Go into the subdirectory `DTT_bb` of the directory `peptide` in your home.

```
~> cd ~/peptide/DTT_bb
```

Have a look at the input file `prep_bb_DTT.arg`. Next to the standard force field files specified under the flags `@build` and `@param` a special file which lists sequentially all the atoms in the molecule and all the bonds between them has to be specified. For the case of DTT this file is already prepared and is called `spec_DTT.dat`. Since we will run the program `prep_bb` interactively the flag `@interact` is also needed. Try to run the program `prep_bb`

```
~/peptide/DTT_bb> prep_bb @f prep_bb_DTT.arg
```

The program will first check whether there are any aromatc rings present in the molecule. Then it will go through the list of atoms specified in the file `spec_DTT.dat` and for each atom will ask you to input the respective integer atom type (IAC), mass code, charge and charge group specifier. It will also give you suggestions for all these values as you go along. You can compare these suggestions with the parameters in Fig. 3.1. All the parameters are given in Chap. 3-3. Let's do this for the atom (CA in Fig. 3.1). First you have to give IAC (integer atom code). As CA is a united atom consiting of a carbon with two hydrogens, its IAC is 15, in agreement with Fig. 3.1 A. This means its mass code is 4. Next the program asks for the charge. We give a 0.150. The next thing the program wants to know is the charge group code. GROMOS topologies usually have charge groups with an integer charge, preferably zero. As the CA is charged, we should give a 0 and add the next atoms until we have an uncharged or integer charge charge group. Then we would give a charge group code 1 to the last atom. Otherwise if CA is uncharged, the charge group can be closed with a 1. In the GROMOS building block figures charge groups are specified by color. Now do the same for the other atoms using Fig. 3.1 A for support.

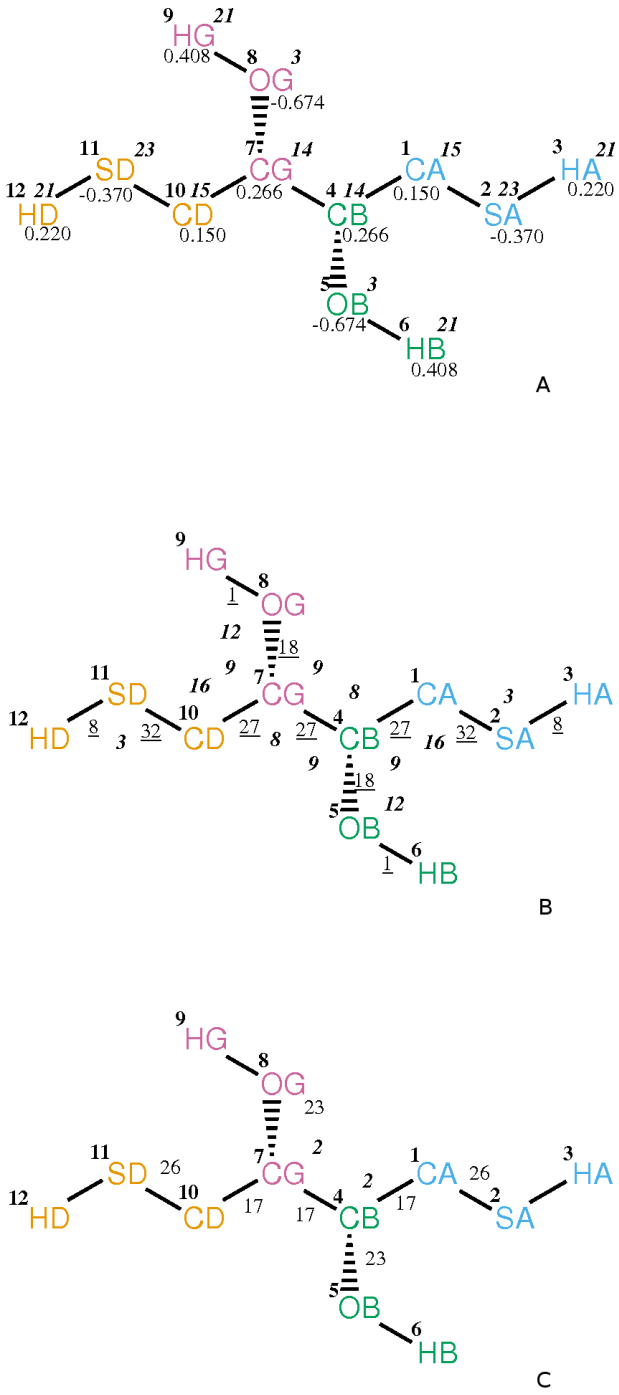

FIGURE 3.1. Force-field parameters for DTT. A) atom number (plain bold), integer atom code (italic bold), IAC charge (plain); B) atom number (plain bold), bond type (plain underlined), bond angle type (italic bold); C) atom number (plain bold), dihedral angle type (plain), improper dihedral angle type (italic bold). The charge groups are depicted in yellow, blue, green and red color.

The same procedure will follow for the bonds, bond angles, dihedral angles and improper dihedral angles, respectively. The program will first ask you for the bonds. The first bond is between atom 1 (CA) and 1 (SA). In Fig. 3.1 B you see that we want this bond to be of type 32. Continue for the other bonds. The bond angles follow, with the first angle being between atoms 2 (SA) - 1 (CA) -4 (CB) being of type 16. Again, continue for all the bond angles. Then the improper dihedrals are set. For the first one 4 (CB) - 1 (CA) -5 (OB) - 7 (CG) the type is given as 2 (Fig. 3.1 C). Note that DTT has two chiral centers (CB and CG). Therefore there are four different combinations of the two correspoding improper dihedrals. In this exercise you decide for one of them. Fig. 3.1 shows (2R,3R)-dihydroxy-1,4-dithiobutan. Finally the program asks you about the proper dihedrals. The first proper dihedral is 4 (CB) - 1 (CA) - 2 (SA) - 3(HA) and has type 26. Define the rest of them yourself.

At the end the building block for DTT will be written to a file called BUILDING.out. Open it and verify if it is correct.

CHAPTER 4

Final words

You have now worked your way through a reduced version of the GROMOS tutorial. It was intended to get you going with GROMOS. The full version of the tutorial contains some more exercises about enhanced sampling techniques and free-energy calculations, but this is probably too much in the context of this advanced GRAZ course. If you are interested, do not hesitate to ask for the full version. There is an even more elaborate tutorial available if you want to do more.[4]

Hint: Check out `https://www.gromos.net` if you are interested in obtaining the GROMOS code, the force-field files and the complete manual. The full tutorial is described as volume 7 of the manual.

Bibliography

[1] M.P. Allen and D.J. Tildesley. *Computer simulation of liquids.* Oxford University Press, New York, USA, 1987.

[2] X. Daura, W.F. van Gunsteren, and A.E. Mark. Folding-Unfolding Thermodynamics of a beta-Heptapeptide From Equilibrium Simulations. *Proteins*, 34:269–280, 1999.

[3] K Wuethrich, M Billeter, and W. Braun. Pseudo-structures for the 20 common amino acids for use in studies of protein conformations by measurements of intramolecular proton-proton distance constraints with nuclear magnetic resonance. *J. Mol. Biol.*, 169:949–961, 1983.

[4] B. Lier, C. Öhlknecht, A. de Ruiter, J. Gebhardt, W.F. van Gunsteren, C. Oostenbrink, and N. Hansen. A suite of advanced tutorials for the gromos biomolecular simulation software [article v1.0]. *Living Journal of Computational Molecular Science*, 2:18552, 2020.

Index