The GROMOS Software for (Bio)Molecular Simulation

Volume 5: Program Library Manual

January 9, 2021

Contents

CHAPTER 1

Introduction

GROMOS, consisting of MD++ and GROMOS++, is a collection of programs developed to prepare, run and analyse a MD simulation. Most programs belong to GROMOS++ and may be used to set up a simulation or analyse the trajectories of a simulation, while MD++ is used to run the simulation.

This volume gives an overview over all the programs, listed either in Chap. 2 (setup of simulations), Chap. 3 (minimizers and simulators) and Chap. 4 (analysis of trajectories) or Chap. 5 with a program description together with required and optional input arguments as well as standard and additional outputs. The focus is on the use of the programs and not on the source code behind. The reader who wishes to change or add the source code of GROMOS is referred to Chap. 6-1 where an outline of the source code including libraries as well as predefined classes and namespaces is given in more datail.

Most common arguments used and needed by a majority of GROMOS programs are explained more extensively in Sec. 1.2 of this volume. The diverse use of (atom, property and vector) specifiers, a powerful tool to specify a group of atoms, properties as distances, angles and many others, is described in Sec. 1.3, accompanied by multiple examples.

1.1. Nomenclature of GROMOS files

GROMOS is very generous concerning the names and endings of input and output files. It leaves the user absolute freedom. Nevertheless, we strongly recommend a consistent pattern of file name endings which helps keeping the overview over different file types. A possible naming (recommendations) is given in Chap. 4-13.

1.2. Common arguments in GROMOS++

Several arguments appear in many GROMOS++ programs and their explanation is given here.

1. `@topo`
   Molecular topology files are read from the `@topo` argument. The file format of a topology is described in Sec. 4-3.2.
2. `@pbc` `arg1` [`arg2`] [`arg..`]
   Periodic boundary type and gathering parameters are read from `@pbc`. The first argument is the boundary type which may take the following values:
   `v` vacuum, non-periodic boundary conditions
   `r` rectangular periodic boundary conditions
   `c` triclinic periodic boundary conditions
   `t` truncated octahedral periodic boundary conditions
   The second and following arguments determine the gathering method and additional gather options. The available gathering methods (`arg2`) are:

`nog` or `0`	do not gather
`glist` or `1`	(default) gathering, based on a list of pairs of atoms the atom pair should be in the sequence: A B, where A is an atom of the molecule to be gathered, and B is an atom of the reference molecule
`gtime` or `2`	gathering based on previous frame
`gref` or `3`	gathering based on a reference structure
`gltime` or `4`	gather first frame based on a list, next frames based on previous frame
`grtime` or `5`	gather first frame based on a reference structure, next frames based on previous frame
`gbond` or `6`	gathering based on bond connectivity
`cog` or `7`	gathering with respect to the centre of geometry of all atoms of the first molecule in the system
`gfit` or `8`	gather selected molecules based on a reference structure which has been superimposed on the first frame of the trajectory, gather remaining molecules to the cog of selected molecules

Further arguments are necessary or optional for specific gathering methods:

`list <atom pair list>`	e.g. `@pbc` r gref refg coord.cnf
`refg <ReferenceStructure>`	e.g. `@pbc` r gltime list 2:res(15:CA) 1:11 3:134 1:11
`molecules <molecule numbers>`	e.g. `@pbc` r gfit refg coord.cnf molecules 1-5

3. `@outformat`
   Some GROMOS++ programs can write the output coordinates in different formats. The following formats are supported:

`cnf`	Configuration format containing POSITION blocks (extension .cnf).
`trc`	Coordinate trajectory format containing POSITIONRED blocks (extension .trc).
`por`	Position restraints specification format (extension .por).
`pdb`	Protein Data Bank (PDB) format. An additional factor can be given to convert the length unit to Å, default 10.0 (nm to Å).
`vmdam`	VMD's ,,Amber Coordinates" format. An additional factor can be given to convert the length unit to Å, default 10.0 (nm to Å).

1.3. Atom, property and vector specifiers in GROMOS++

Analysis of the trajectories of a MD simulation is, besides the correct setup, of importance to a computational scientist. The more specific the questions, the more flexible the programs which answer those questions have to be. GROMOS++ makes use of three specifiers to keep its flexibility: atom specifiers, property specifiers and vector specifiers. Each of them is used as an input parameter (a string with a well defined format) for some GROMOS++ programs. Note that some shells may modify the brackets or other special characters. Therefore, quotes should be used when using atom specifiers as a command line argument.

This section introduces the reader to each of the three specifiers giving an overview of the different possibilities to use them.

1.3.1. Atom specifiers. Atom specifiers define a general way to access specific atoms of a system. It is even possible to access atoms which are not there (virtual atoms) or common properties (e.g. the center of geometry or center of mass) of multiple atoms. The atom specifier can be defined using four different formats:

- Molecules and Atoms:
  `<mol>[-<mol>]:<atom>[-<atom>]`
- Residues:
  `<mol>[-<mol>]:res(<residue>:<atom>[,<atom>...])`
- Virtual Atoms:
  `va(<type>, <atomspec>)`

- File:
  `file(<filename>)`

`<mol>` is the molecule number and `<atom>` either the atom number or the atom name. Instead of the atom or molecule number one can also specify all solute or solvent molecules or atoms using `a` or `s`, respectively. The solvent is only accessible if there is a topology and a coordinate file given. It is not possible to access the solvent from a topology only, since GROMOS does not know how many solvent molecules the system consists of. The `<type>` argument defines the virtual atom type the specifier accesses. The following types are known in GROMOS++:

0:	explicit/real atom
1:	aliphatic CH_1 group
2:	aromatic CH_1 group
3:	non-stereospecific aliphatic CH_2 group (pseudo atom)
4:	stereospecific aliphatic CH_2 group
5:	single CH_3 group (pseudo atom)
6:	non-stereospecific CH_3 groups (isopropyl; pseudo atom)
7:	non-stereospecific CH_3 groups (tert-butyl; pseudo atom)
-1:	centre of geometry
-2:	centre of mass

`<atomspec>` is a complete atom specifier of one of the four formats above and `<filename>` is the name of the file (output of the `atominfo` program) listing the atoms to consider.

Multiple atom specifiers must be separated by a semicolon while multiple molecules or atoms within the same atom specifier are separated by a comma.

Examples:

atoms 3 an 7 to 12 of molecule 2:

```
2:3,7-12
```

all atoms of molecule 1:

```
1:a
```

all CA, N or C atoms of all molecules

```
a:CA,N,C
```

atom 1 of residue 3 and 5 of molecule 1:

```
1:res(3,5:1)
```

all C, N or CA atoms of residues named SER or THR of molecule 1:

```
1:res(SER,THR:C,N,CA)
```

the centre of mass of molecule 1:

```
va(-2,1:a)
```

the alpha hydrogen of residue 1 in a protein:

```
va(1,1:res(1:CA,N,CB,C))
```

all CA atoms of the first molecule, accessed by a file from atominfo:

```
\$ atominfo @topo ex.topo @atomspec 1:CA > ca.spec
file(ca.spec)
```

In addition, there are the two keywords `not` and `minus` to exclude some atoms from an atom specifier. Note that the atoms specified with the keyword `not` are never included in the resulting atom specifier while the keyword `minus` allows to add the removed atoms within the same specifier later on again.

Examples:

all atoms of the first molecule but without the second residue:

```
1:a minus(1:res(2:a))
```

all atoms of the first molecule but without any C atoms:

```
1:a minus(1:res(2:a)) 1:res(2:C)
```

Finally there are programs that syntactically require an atom specifier although one does not want to specify any atoms for the computation (e.g. to disable rotational fits). For this purpose there is the keyword `no`. It can be used to specify an empty set of atoms.

Example:
no atoms:

```
no
```

1.3.2. Vector specifiers. Vector Specifiers may be used in a property specifier (see Sec. 1.3.3) to calculate some well defined properties using the help of vectors. There are three different formats to specify a vector:

- `cart(<x>,<y>,<z>)`
- `polar(<r>,<α>,<β>)`
- `atom(<atomspec>)`

with `<x>`, `<y>` and `<z>` being the three coordinates of a Cartesian 3D vector, `<r>` the length (norm) of a vector **x**, `<α>` and `<β>` the polar angles in degree,

$$\mathbf{x} = (\mathbf{r}\cos(\alpha)\cos(\beta), \mathbf{r}\sin(\alpha), -\mathbf{r}\cos(\alpha)\sin(\beta)) \quad , \tag{1.1}$$

and `<atomspec>` is an atom specifier (see Sec. 1.3.1).

Examples:

the vector vector $(2, 5, 1)$:

```
cart(2,5,1)
```

the vector $(0, 2.5, 0)$

```
polar(2.5,45.0,90.0)
```

1.3.3. Property specifiers. Like the other two specifiers (see Sec. 1.3.1 and Sec. 1.3.2), property specifiers are used as an argument for some analysis programs of GROMOS++. They are used to specify the property one is interested in. The general format of a property specifier is:

- `<type>%<arg1>[%<arg2>...]`

with `<type>` defining the specific property and `<arg1>` the argument (an atom or vector specifier, compare Sec. 1.3.1 and Sec. 1.3.2, respectively) needed to calculate this property. It is clear that dependent on the property type the number of required arguments may change.

The following is a list of all property types implemented in GROMOS++:

d:	distance
a:	angle
t:	torsion
tp:	periodic torsion
ct:	cross torsion
hb:	hydrogen bond
st:	stacking
o:	order
op:	order parameter
pr:	pseudo rotation
pa:	pucker amplitude
expr:	expression property

Multiple calculations of the same property at different positions of the molecule or system are available via substitutions (see example 4 where it is shown to calculate multiple distances within the same molecule).

Some examples for all properties but the `expr` property follow. The latter is a bit more complex and will be discussed after the examples at the end of Sec. 1.3.1.

Examples:

the distance between atoms 1 and 2 of molecule 1:

```
d%1:1,2
```

the distance between the first atoms of molecule 1 and 2:

```
d%1:1;2:1
```

the distance between the centres of mass of molecules 1 and 2:

```
d%va(com,1:a);va(com,2:a)
```

the distances between the H and N atoms of residues 3 to 5 of molecule 1 (making use substitution):

```
d%1:res((x):H,N)|x=3-5
```

the angle defined by atoms 1, 2 and 3 of molecule 1:

```
a%1:1-3
```

the angle between the virtual alpha hydrogen of residue 1 and the atoms CA and C of residue 1 of molecule 1:

```
a%va(1,1:res(1:CA,N,CB,C));1:res(1:CA,C)
```

the torsion defined by the atoms H, N, CA and CB of residue 2 of molecule 1:

```
t%1:res(2:H,N,CA,CB)
```

the cross torsion defined by the atoms 2, 3, 4 and 5 as well as 3, 4, 5 and 6 of molecule 1:

```
t%1:1,2,3,4%1:3,4,5,6
```

the hyrdogen bond between the H atom of residue 3 and the O atom of residue 5 of the first molecule:

```
hb%1:res(3:N,H);1:res(5:O)
```

the stacking between the HISB ring of residue 44 of molecule 1 and and the pyrimidine ring of residue 2 of of molecule 2:

```
st%1:res(44:CG,CE1,CE2,CD1,CD2,CZ)%2:res(1:N1,C5,N3,C6,C2)
```

the order between the N-H bond of residue 2 and the z-axis:

```
o%atom(1:res(2:H,N))%cart(0,0,1)
```

the order parameter between the x-axis and the vector defined by atoms 1 and 2 of the first molecule:

```
op%cart(1,0,0)%atom(1:1,2)
```

the pseudo rotations of the atoms in the furanose ring of residues 1 to 6 of molecule 1:

```
pr%1:res((x):C1*,C2*,C3*,C4*,O4*)|x=1-6
```

the pucker amplitude of the atoms in the furanose ring of residue 1 of molecule 1:

```
pa%1:res(1:C1*,C2*,C3*,C4*,O4*)
```

The **expression property** allows the evaluation of a multitude of expressions over a trajectory. The general form is:

```
- expr%<f1>(<args1>) <op> <f2>(<args2>)
```

where `<op>` is one of the following arithmethic or logical operators:

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`!`	not
`==`	equals
`!=`	does not equal
`>`	bigger than
`<`	smaller than
`>=`	bigger or equal than
`<=`	smaller or equal than
`&&`	logical and

`||` logical or

`<f1>` and `<f2>` are functions followed by their arguments. Depending on the type of function, the arguments are different. There are the following funcitons:

-	functions where the argument is a scalar:
`sin`	the sine function
`cos`	the cosine function
`tan`	the tangens function
`asin`	the inverse sine function
`acos`	the inverse cosine function
`atan`	the inverse tangens function
`exp`	the exponential function
`ln`	the logarithm
`abs`	the absolute value
`sqrt`	the square root
-	functions where the argument is a vector:
`abs`	the norm of a vector
`abs2`	the squared norm of a vector
-	functions which need two vectors:
`dot`	the dot product of two vectors
`cross`	the cross product of two vectors
`ni`	the nearest image of vector 1 with respect to vector 2

Examples:

calculates the dot product between position of atom(1:1) and the vector (0,0,1); that is the z-component of the position of the first atom of the first molecule:

```
expr%dot(atom(1:1),cart(0,0,1))
```

calculates the distance between the first atom of the first and second molecule. First, the nearest image of atom(2:1) according to atom(1:1) is calculated. Second, this vector is substracted from atom(1:1) and the absolute value is taken:

```
expr%abs(atom(1:1) - ni(atom(2:1), atom(1:1)))
```

returns 1 if the the discussed distance is below 1.0 nm and 0 if not:

```
expr%abs(atom(1:1) - ni(atom(2:1), atom(1:1)))<1.0
```

calculates the order of the vector defined by atoms 1:1,2 and the z-axis. First, the cosine is calculated by the dot product of the vectors and division by their lengths (the second vector has length 1). Then the angle is calculated and converted to degrees:

```
expr%acos(dot(atom(1:1,2),cart(0,0,1)) / abs(atom(1:1,2)))*(180/3.1415)
```

CHAPTER 2

Setup of simulations (preprocessing)

2.1. bin_box (GROMOS++ program)

Program description:

When simulating a molecular liquid, a starting configuration for the solvent molecules has to be generated. To generate a starting configuration for the simulation of a binary mixture, the program `bin_box` can be used. A cubic box is filled with solvent molecules by randomly distributing them on an evenly spaced grid such that the total density of the box and the mole fractions of the solvent components match the specified values. Note that the program `ran_box` (see Sec. 2.25) can be used alternatively, which generates a starting configuration for the simulation of mixtures consisting of an unlimited number of components, in which the molecules are oriented randomly.

Required input arguments

`@topo1`	⟨molecular topology file for a single molecule of the type of mixture component 1⟩
`@pos1`	⟨input coordinate file for a single molecule of the type of mixture component 1⟩
`@topo2`	⟨molecular topology file for a single molecule of the type of mixture component 2⟩
`@pos2`	⟨input coordinate file for a single molecule of the type of mixture component 2⟩
`@nsm`	⟨total number of molecules per dimension (NSM)⟩
`@densit`	⟨density of the binary mixture (kg/m^3)⟩
`@fraction`	⟨mole fraction of mixture component 1⟩

Optional input arguments

none

Standard output

coordinate file with NSM^3 molecules in a cubic box at the specified density.

Additional output

none

2.2. `build_box` (GROMOS++ program)

Program description:

When simulating a molecular liquid, a starting configuration for the solvent molecules has to be generated. Program `build_box` generates a cubic box filled with identical solvent molecules which are put on an evenly spaced grid such that the density of the box matches the specified value. Note that to generate a starting configuration for the simulation of a binary mixture, the program bin_box can be used (see Sec. 2.1). Alternatively, program `ran_box` (see Sec. 2.25) generates a starting configuration for the simulation of mixtures consisting of an unlimited number of components, in which the molecules are oriented randomly.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2) for a single molecule⟩
`@pos`	⟨input coordinate file for a single molecule⟩
`@nsm`	⟨number of molecules per dimension (NSM)⟩
`@dens`	⟨density of the liquid (kg/m^3)⟩

Optional input arguments

none

Standard output

coordinate file with NSM^3 solvent molecules in a cubic box at the specified density

Additional output

none

2.3. `check_box` (GROMOS++ program)

Program description:

To check for the distances between atoms and periodic copies of the other atoms in the system, program `check_box` can be used. `check_box` calculates and writes out the minimum distance between any atom in the central box of the system and any atom in the periodic copies (rectangular box and truncated octahedron are supported).

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2) of the system⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨input coordinate (trajectory) file⟩

Optional input arguments

`@atoms`	⟨atoms to include in calculation (default: all solute)⟩

Standard output

time series of the shortest distance between periodic copies of the selected atoms, followed by the overall minimum over the simulation

Additional output

none

2.4. `check_top` (GROMOS++ program)

Program description:

Making a correct topology is one of the most important requirements for doing a successful simulation. `check_top` helps to remove often made errors from a topology in three ways. First, it runs some standard tests on the molecular topology and warns if something unexpected is observed in the topology. Second, it can calculate all bonded interaction energies for a given set of coordinates to determine the compatibility of the topology with the coordinates. Third, it can check for consistency in the force-field parameters by comparing it to a specified set of building blocks and force-field parameters.

In the first phase, `check_top` tests that:

1. there is maximally one bond defined between any pair of atoms
2. no atom appears twice in the definition of one given bond
3. only bond types are used that are defined in the topology
4. a bond angle is defined for the atoms involved in any two bonds sharing one atom
5. there is maximally one bond angle defined for a given set of three atoms
6. atoms involved in a bond angle definition are bound to the central atom
7. no atom appears twice in the definition of one given bond angle
8. only bond angle types are used that are defined in the topology
9. an improper dihedral angle is defined centered on every atom that is bound to exactly three other atoms
10. there is maximum one improper dihedral angle defined for any set of four atoms
11. atoms involved in an improper dihedral angle definition are bound
12. no atom appears twice in the definition of one given improper dihedral angle
13. only improper dihedral types are used that are defined in the topology
14. atoms involved in a proper dihedral angle are sequentially bound
15. no atom appears twice in the definition of one given dihedral angle
16. only dihedral angle types are used that are defined in the topology
17. only atom types are used that are defined in the topology
18. the sum of partial charges on atoms in one charge group is an integer value
19. excluded atoms are 1,2- or 1,3- or 1,4-neighbours
20. atoms only have atoms with a higher sequence number in their exclusion list
21. 1,2- or 1,3-neighbours are excluded
22. 1,4-exclusions are separated by 3 bonds (1,4-neighbours)
23. atoms only have atoms with a higher sequence number in their 1,4-exclusion list
24. 1,4-neighbours are in the exclusion list or in the 1,4-exclusion list
25. no exclusions or 1,4-exclusions are defined for the last atom in the topology
26. the charge group code of the last atom in the topology is 1

Additionally, for atoms that are 1,4 neighbours but are listed as excluded atoms a warning is printed. This is usually only the case if an aromatic group is involved. Note that a topology that passes all these tests is by no means guaranteed to be error-free. Conversely, some of these tests are merely meant as warnings for the user which may point at errors in the majority of cases. In some cases, the user may very well want to use a topology that does not fulfill all tests.

In the second phase, potential energies of all bonds, bond angles, improper dihedral angles and proper dihedral angles are calculated and written out. Abnormally large energies or deviations from ideal values may indicate an error in the topology, or an inconsistent set of coordinates with the topology. See program `shake_analysis` (see Sec. 5-5.6) for a similar check on the non-bonded interaction energies.

In the third phase `check_top` can compare the topology with other building blocks in a specified molecular topology building block file and the corresponding interaction function parameter file. It checks if in the molecular topology building block file we observe one of the following:

1. other atoms with the same name and the same integer atom code (IAC)
2. other atoms with the specified IAC
3. other atoms with the same IAC and mass
4. other atoms with the same IAC and charge
5. other bonds between atoms of the same IAC with the same bond type
6. other bond angles between atoms of the same IAC with the same bond-angle type

7. other improper dihedral angles between atoms of the same IAC with the same improper dihedral type
8. other dihedral angles between atoms of the same IAC with the same dihedral-angle type

In cases where the parameters specified in the program are not observed anywhere else, or when they are not the most common parameter, the program prints out a list of possible alternatives. Again, we stress that `check_top` only points at possible inconsistencies and does not necessarily indicate errors in your topology.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩

Optional input arguments

`@coord`	⟨coordinate file for energy calculation⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@build`	⟨building block file for consistency check⟩
`@param`	⟨parameter file for consistency check⟩

Standard output

list of inconsistencies

Additional output

none

2.5. `com_top` (GROMOS++ program)

Program description:

To generate molecular topology files for the use in simulations of e.g. (macro) molecular complexes, or mixtures containing several solutes and/or (co)solvents, it is usually convenient to merge existing molecular topology files. Program `com_top` combines multiple topologies into one new topology.

The user has to specify which molecular topologies are to be merged, and from which file the force-field parameters and the solvent have to be taken. The resulting molecular topology file is written out to the standard output.

The program can also be used for topology file format conversion. The argument `@inG96` converts GROMOS96 topologies to the current format. On the other hand `@outG96` converts topologies in the current format to GROMOS96 format.

Required input arguments	
`@topo`	⟨molecular topology files (see Sec. 1.2)⟩
`@param`	⟨index number of molecular topology file to take parameters from⟩
`@solv`	⟨index number of molecular topology file to take solvent from⟩

Optional input arguments	
`@inG96`	⟨reads in a topology file in GROMOS96 format⟩
`@outG96`	⟨the output topology is written in the GROMOS96 format⟩

Standard output

combined topology file

Additional output

none

2.6. `con_top` (GROMOS++ program)

Program description:

A molecular topology file in which a system is described by a specific version of a force-field parameter set can be converted into a molecular topology file with interaction parameters from a different force-field version, using the program `con_top`.

An interaction function parameter file has to be specified that corresponds to the force-field version into which the molecular topology should be converted. `con_top` checks whether the topology is not referring to atom, bond, bond angle or (improper) dihedral types that are not defined in the parameter file. If type numbers of atoms, bonds, bond angles, etc. change with the force-field parameter set, a renumbering file (see Sec. 4-7.2 for its formats) can be given to specify these changes.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2) to be converted⟩
`@param`	⟨interaction function parameter file⟩
`@renum`	⟨renumbering file⟩

Optional input arguments

none

Standard output

converted molecular topology file

Additional output

none

2.7. `copy_box` (GROMOS++ program)

Program description:

Program `copy_box` can be used to duplicate the size of a system in the x, y or z direction (or **a**, **b**, **c** for triclinic boxes). This is especially convenient if one wants to double the size of a system under periodic boundary conditions in which the central box has a rectangular or triclinic shape. If one wants to perform more elaborate transformations, the program `cry` might be of use (see Sec. 2.8). Note that program `com_top` (see Sec. 2.5) can be useful to additionally duplicate the solute block in the topology. The `@pbc` flag is optional. Only if this flag is given, gathering of the molecules will be performed before copying the box.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2) of the system⟩
`@pos`	⟨input coordinate file⟩
`@dir`	⟨direction in which the coordinates are to be duplicated (x,y,z)⟩

Optional input arguments

`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩

Standard output

coordinate file containing the multiplied system

Additional output

none

2.8. `cry` (GROMOS++ program)

Program description:

When using periodic boundary conditions, the computational box containing the molecular system is treated as being translationally invariant. So, periodic boundary conditions can also be used when simulating a crystal as long as the unit cell, or a number of adjacent unit cells is used as computational box. Unless the asymmetric unit is translationally invariant, it cannot be used as computational box. Since crystallographic coordinates of molecular systems are generally only provided for the molecules in one asymmetric unit, the coordinates of the other molecules in the unit cell or cells are to be generated by crystallographic symmetry transformations.

The program `cry` can rotate and translate copies of a system to create a crystal unit cell. Based on a topology and initial (gathered) coordinates, as well as a specification file for the symmetry transformations. A conversion factor can also be given if the translation vector is specified in different units than the coordinates. A coordinate file is generated that contains coordinates for as many systems as there are transformations defined in the specification file. The corresponding topology can easily be generated using the program `com_top` (Sec. 2.5).

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pos`	⟨input coordinate file for the molecules⟩

Optional input arguments

`@spec`	⟨specification file for the symmetry transformations⟩
`@factor`	⟨conversion factor for distances⟩
`@spacegroup`	⟨space group symbol⟩
`@cell`	⟨cell edge lengths [nm] and angles [degrees]⟩
`@keepbox`	⟨no expansion of the initial box⟩

Standard output

coordinates for a crystal unit cell

Additional output

none

2.9. `duplicate` (GROMOS++ program)

Program description:

Program duplicate searches for duplicated atoms, i.e. atoms having the same coordinates as another atoms. If requested, a coordinate file with the duplicated molecules removed is written out.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pos`	⟨input coordinate file for the molecules⟩

Optional input arguments

`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@write`	⟨write out filtered coordinates⟩

Standard output

A list of molecules containing duplicated atoms. The coordinates filtered for duplicated if requested.

Additional output

none

2.10. `explode` (GROMOS++ program)

Program description:

Program `explode` takes a box with `nsm` molecules and puts them on a grid with distance `dist` between the grid points. This tool is useful in case a vacuum simulation has to be performed by simulating `nsm` molecules at a large intermolecular distance. The input topology and coordinate files must contain at least `nsm` molecules. If there are less molecules than grid positions a message is printed to inform the user. As input coordinates, one can for instance take a liquid box generated by program `build_box` (see Sec. 2.2).

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2) of the system⟩
`@pos`	⟨input coordinate file⟩
`@nsm`	⟨number of solute molecules⟩
`@dist`	⟨distance to put between molecules ⟩

Optional input arguments

none

Standard output

coordinate file containing the coordinates of molecules that were put at larger intermolecular distances

Additional output

none

2.11. `gca` (GROMOS++ program)

Program description:

Sometimes, one may want to modify a specified molecular configuration such as to obtain specified values of bond lengths, bond angles or dihedral angles. Program `gca` allows the user to do this. In addition, series of configurations can be generated in which the molecular properties of choice are modified stepwise. If more than one property to be changed has been specified, configurations for all combinations of values will be generated, allowing for a systematic search of the property space.

In order to fulfill the requested property values, program `gca` will

- for a bond length between atoms i and j, shift all atoms connected to j (not i) and onwards;
- for an bond angle defined by atoms i,j and k rotate all atoms connected to k (not j) and onwards around the axis through atom k and perpendicular to the i,j,k-plane;
- for a dihedral angle defined by atoms i,j,k and l rotate all atoms connected to k and l (not j) around the axis through atoms j and k.

This procedure may lead to distortions elsewhere in the molecule if the atom count is not roughly linear along the molecular structure, or if the specified properties are part of a cyclic structure. The program does not check for steric clashes resulting from the modifications.

The properties to be modified are specified through a property specifier (see Sec. 1.3.3), followed by either one additional argument (single value to be specified) or three additional arguments (to generate a range of values).

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@prop`	⟨property specifier (see Sec. 1.3.3): properties to change⟩
`@traj`	⟨input coordinate file⟩

Optional input arguments	
`@outformat`	⟨output coordinates format, see Sec. 1.2⟩

Standard output

molecular configurations in the requested format

Additional output

none

2.12. gch (GROMOS++ program)

Program description:

In the standard GROMOS force fields, part of the hydrogen atoms (polar, aromatic) are explicitly treated, whereas other hydrogen atoms (aliphatic, some aromatic) are implicitly treated by incorporation into the (carbon)atom to which they are attached. Depending on the presence or absence of hydrogen atom coordinates in a molecular configuration file, hydrogen atom coordinates may have to be recalculated.

Program gch calculates optimal positions for hydrogen atoms for which the connecting bond shows a relative deviation from the zero-energy value larger than a user specified threshold. Coordinates for all hydrogen atoms that are explicitly listed in the topology should already be contained in the coordinate file. Program pdb2g96 (see Sec. 2.19) e.g. will include atoms for which no coordinates were present in the pdb file with coordinates set to zero. If defined, gch uses topological information on bonds, bond angles and dihedral angles to place hydrogen atoms at the optimal location. In cases where the necessary angular parameters are not provided in the topology, gch uses 109.5° for tetrahedral centers and 120° for planar centers.

Eight types of geometries can be handled when generating hydrogen atom coordinates:

1. An atom (a) is bonded to one hydrogen (H) and one other heavy atom (nh). A fourth atom (f) is searched for which is bound to nh and preferably used to define the dihedral around the nh-a bond. The coordinates of H are generated in such a way that the dihedral f-nh-a-H is trans and that the angle nh-a-H and bond length a-H correspond to their minimum energy values.
2. An atom (a) is bonded to one hydrogen (H) and two other heavy atoms (nh1 and nh2). The coordinates of H are generated to be in the plane through nh1, nh2 and a, on the line bisecting the nh1-a-nh2 angle and with an a-H bond length corresponding to the minimum energy value in the topology, such that the nh1-a-H and nh2-a-H angles are larger than 90 degrees.
3. An atom (a) is bonded to two hydrogens (H1 and H2) and one other heavy atom (nh). A fourth atom (f) is searched for which is bound to nh and preferably is used to define the dihedral around the nh-a bond. The coordinates of H1 are generated in such a way that the dihedral f-nh-a-H1 is trans and that the angle nh-a-H1 and bond length a-H1 correspond to their minimum energy values. The coordinates of H2 are generated to have the angles nh-a-H2 and H1-a-H2 as well as the bond length a-H2 at their minimum energy values. If this does not result in a planar configuration around a, the improper dihedral a-nh-H1-H2 will be positive.
4. An atom (a) is bonded to three hydrogens (H1, H2 and H3) and one other heavy atom (nh). A fourth atom (f) is searched for which is bound to nh and preferably is used to define the dihedral around the nh-a bond. The coordinates of H1 are generated in such a way that the dihedral f-nh-a-H1 is trans and that the angle nh-a-H1 and bond length a-H1 correspond to their minimum energy values. The coordinates of H2 are such that the angles nh-a-H2 and H1-a-H2 and the bond length a-H2 are at their minimum energy values, and the improper dihedral a-nh-H1-H2 is positive. The coordinates of H3 are such that the angles nh-a-H3 and H1-a-H3 and the bond length a-H3 are at their minimum energy values and the improper dihedral a-nh-H1-H3 has a negative value.
5. An atom (a) is bonded to one hydrogen atom (H) and three other heavy atoms (nh1, nh2, nh3). The coordinates of H are generated along the line going through atom a and a point corresponding to the average position of nh1, nh2 and nh3, such that the bond length a-H is at its minimum energy value and the angles nh1-a-H, nh2-a-H and nh3-a-H are larger than 90 degrees.
6. An atom (a) is bonded to two hydrogen atoms (H1 and H2) and two other heavy atoms (nh1 and nh2). The coordinates of H1 and H2 are placed above and below the plane going through atoms nh1, nh2 and a, in such a way that the a-H1 and a-H2 bond lengths and the H1-a-H2 bond angle are at their minimum energy values. The improper dihedral angle a-nh1-nh2-H1 will be positive.
7. An atom (a) is bonded to two hydrogen atoms (H1 and H2), but to no heavy atoms. This is likely to be a (crystallographic) water molecule. First a molecule is generated having the a-H1 aligned in the z-direction and the a-H2 in the z-y plane with the angle H1-a-H2 and bond lengths a-H1 and a-H2 according to their minimum energy values. This molecule is then rotated around x, y and z by three random angles.
8. An atom (a) is bonded to four hydrogen atoms (H1, H2, H3 and H4), but to no heavy atoms. A molecule is generated with all bond lengths at their minimum energy values, the a-H1 aligned in the z-direction, H2 in the x-z plane and H3 such that the improper a-H1-H2-H3 is positive and H4 such that the improper a-H1-H2-H4 is negative. The complete molecule is then rotated by three random angles around x, y and z.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pos`	⟨input coordinate file⟩

Optional input arguments

`@tol`	⟨tolerance (default 0.1 %)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩

Standard output

coordinates of the molecular system in which the coordinates of all hydrogen atoms that displayed a relative deviation larger than the specified tolerance have been repositioned

Additional output

none

2.13. `ion` (GROMOS++ program)

Program description:

When simulating a charged solute in solution, one may wish to include counter-ions in the molecular system in order to obtain a neutral system, or a system with a specific ionic strength. The program `ion` can replace solvent molecules by atomic ions by placing the ion at the position of the first atom of a solvent molecule. Substitution of solvent molecules by positive or negative ions can be performed by selecting solvent positions with the lowest or highest Coulomb potential, respectively, or by random selection. In order to prevent two ions being placed too close together, a sphere around each inserted ion can be specified from which no solvent molecules will be substituted by additional ions. In addition, the user can specify specific water molecules that should not be considered for replacement.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@pos`	⟨input coordinate file⟩

Optional input arguments

`@positive`	⟨number⟩	⟨ionname⟩	⟨residue name (optional)⟩
`@negative`	⟨number⟩	⟨ionname⟩	⟨residue name (optional)⟩
`@potential`	⟨cutoff for potential calculation⟩		
`@random`	⟨random seed⟩		
`@exclude`	⟨atom specifier (see Sec. 1.3.1): solvent molecules to be excluded⟩		
`@mindist`	⟨minimum distance between ions⟩		

Standard output

Coordinates of the molecular system in which the specified number of ions replace solvent molecules

Additional output

none

2.14. `link_top` (GROMOS++ program)

Program description:

For branched systems, it may be very cumbersome to create crosslinks in a topology. Program link_top allows the user to apply a pre-defined link to the topology. The link is defined in a special building block file, which contains (after a TITLE block), MTBUILDBLLINK blocks. This block has the following layout:

```
MTBUILDBLLINK
# RNME
XYZ
# number of atoms
7
#ATOM RES ANM IACM MASS CGMICGM MAE MSAE
1 1 CA 14 13.018 0.00000 1 2 2 6
2 1 CB 15 14.027 0.16000 1 2 5 6
3 1 OG 0 0 0.00000 1 0
4 1 HG 0 0 0.00000 1 0
5 2 CB 15 14.027 0.16000 0 1 6
6 2 OG 4 15.994 -0.32000 1 0
7 2 HG 0 0 0.00000 1 0
# bonds
# NB
2
# IB JB MCB
1 2 1
2 6 12
# bond angles
# NBA
2
# IB JB KB MCB
1 2 6 12
2 6 5 12
# improper dihedrals
# NIDA
0
# IB JB KB LB MCB
# dihedrals
# NDA
3
# IB JB KB LB MCB
0 1 2 6 1
1 2 6 5 1
2 6 5 0 1
END
```

The atoms section of the building block contains all atoms that are involved in the link. The second column specifies that these atoms are to be found in the first or second residue of the link. The atoms are identified in the original topology by the residue sequence number indicated in the input (`@linking`) and the name of the atom according to the MTBUILDBLLINK.

In a first step, link_top, removes all atoms for which the IAC is 0. All references to these atoms (exclusions, bonds, angles, etc.) are removed from the topology. Next, the remaining atoms in the link defition get updated: the values in the topology of IAC, mass, charge, and charge group get replaced by whatever is indicated in the MTBUILDBLLINK block. The exclusions of the original topology (without the removed atoms) remain, and the exclusions that are specified in the MTBUILDBLLINK block are added.

Covalent interactions that need to be changed are also specified in the MTBUILDBLLINK block. The program only allows the user to specify bonds, angles and improper dihedral angles that are referring to atoms that are all part of the link specification. Any bonds, angles and improper dihedral angles that were present in the topology for these atoms will be removed and the newly defined interactions are added to the topology. For dihedral angles, the program allows the user to refer to the first and/or last atom to be represented by a number 0. For these atoms, the program will search in the topology for an atom that is bound to the second or third atom, respectively and assign the dihedral angle to this atom. Any dihedral angles that were already defined for this group is replaced. Multiple dihedral angles for the same set of four atoms may be added.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@links`	⟨file containing the MTBUILDBLLINK blocks⟩
`@linking`	⟨residue sequence number⟩ ⟨residue sequence number⟩ ⟨name of link⟩

Optional input arguments

none

Standard output

molecular topology file

Additional output

none

2.15. `make_pt_top` (GROMOS++ program)

Program description:

Program `make_pt_top` takes two or more molecular topologies and writes the differences in the perturbation topology format. Both topologies must contain the same number of atoms. The softness parameters α_{LJ} and α_C can be specified by an input parameter.

Required input arguments

`@topo`	⟨multiple molecular topology files (see Sec. 1.2)⟩
`@softpar`	⟨softness parameters (α_{LJ} and α_C)⟩

Optional input arguments

none

Standard output

perturbation topology, containing the parameter differences.

Additional output

none

2.16. `make_sasa_top` (GROMOS++ program)

Program description:

Program `make_sasa_top` adds the atom-specific information required to use the SASA/VOL implicit solvent model to the molecular topology file. It reads in an existing molecular topology file created using `make_top` (see Sec. 2.17), along with a SASA/VOL specification library file, which contains the atom-specific SASA/VOL parameters. The specification library file must be for the same force field as was used to create the molecular topology file. The inclusion of hydrogen atoms in the calculation of the SASA during the simulation may also be specified.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@sasaspec`	⟨SASA/VOL specification library file⟩

Optional input arguments

`@noH`	⟨do not include hydrogen atoms (default: include)⟩

Standard output

molecular topology file with appended SASA/VOL information

Additional output

none

2.17. `make_top` (GROMOS++ program)

Program description:

Program `make_top` builds a molecular topology from a building block sequence. `make_top` reads in a molecular topology building-block file (e.g. `mtb53a6.dat`) and an interaction function parameter file (e.g. `ifp53a6.dat`), and gathers the specified building blocks to create a topology. Cysteine residues involved in disulfide bridges as well as heme and coordinating residues involved in covalent bonds to the iron atom have to be explicitly specified. Topologies for cyclic sequences of building blocks can be generated using `@cyclic`.

Required input arguments

`@build`	⟨molecular topology building block file⟩
`@param`	⟨interaction function parameter file⟩
`@seq`	⟨sequence of building blocks in the solute⟩
`@solv`	⟨building block for the solvent⟩

Optional input arguments

`@cys`	⟨cys1⟩–⟨cys2⟩ . . . ⟨cys1⟩–⟨cys2⟩
`@heme`	⟨residue sequence number⟩ ⟨heme sequence number⟩

Standard output

molecular topology file

Additional output

none

2.18. `mk_script` (GROMOS++ program)

Program description:

A MD simulation is usually performed by executing a small script that combines all the necessary files and redirects the output to the appropriate places. When simulations are performed on a queue, such scripts become indispensable. Additionally, in many simulation projects the user prepares similar input files and scripts over and over again. Program `mk_script` can either create a series of similar scripts that run a sequential list of simulations (`@script`) or it can create scripts for a more complex set of simulations that perform a specific task (start-up, perturbation; `@joblist`). Scripts for special cases such as `REMD` simulations (`@remd`) can also be written.

GROMOS does not require specific filenames for specific types of files. However, most users find it useful to retain some order in their filenames. `mk_script` has a standard way of constructing filenames that depends on the script number and the system name. The user can specify another set of rules to create filenames through the `mk_script` library file (`@template`). In this file, machine-dependent modifications to the scripts that are to be written can also be specified, such as the job submission command, the MPI command, the stopcommand (to delete all subsequent jobs from the queue in case the current job fails) and which queue to use (`@queue`). A standard location of the `mk_script` library file can be specified through the environment variable `MK_SCRIPT_TEMPLATE`.

Program `mk_script` can write input files for MD++ (`@version`). The MD++ input file (`@files->input`) should also be of the correct format: `mk_script` cannot convert program-specific MD++ input blocks into the analogous blocks for the other version of GROMOS.

In addition to write out scripts and input files, `mk_script` performs a small number of tests on the given input files to prevent the user from submitting a simulation that will fail within the first few seconds. In the messages produced by these tests, a distinction is made between warnings and errors. A warning is given for inconsistencies in the inputs that may lead to an erroneous simulation, but could also be intentional. An error is produced by inconsistencies that will definitely result in the program crashing. Note that passing the tests carried out by `mk_script` does not guarantee that a simulation will work, as these checks are not exhaustive. All performed tests are listed below (Warnings, Errors).

The mentioned tests are done for every script since some variables may change due to a joblist file. If there are no errors, the input file and script will be written to disc. If there are errors, the script and input file will not be written, unless the user forces this (`@force`).

Warnings:

1. the GROMOS binary specified in the `mk_script` input file cannot be found
2. the highest `LAST` atom number in the `MULTIBATH` block in the MD++ input file is not equal to the total number of atoms calculated from the topology file and SYSTEM block
3. `DT` in the `STEP` block is too large in combinations with the geometric constraints in the `CONSTRAINT` block (MD++). Suggested step sizes are:

0.0005 ps:	no constraints on solvent or bonds involving hydrogens
0.001 ps:	no constraints on bonds not involving hydrogens
0.002 ps:	all bonds constrained

4. the `FORCE` for bonds that are `SHAKE`n is calculated
5. the `FORCE` for bonds that are not `SHAKE`n is not calculated
6. smallest box dimension (length) of the periodic box is less than twice the long-range cut-off `RCUTL` in the `PAIRLIST` block of the MD++ input file
7. the reaction field cut-off distance RCRF in the `NONBONDED` block of the MD++ input file is not equal to the long-range cut-off `RCUTL` in the `PAIRLIST` block (MD++)
8. a perturbation topology was specified in the `mk_script` input file but no perturbation was requested in the MD++ input file
9. the combination of RLAM and DLAMT in the `PERTURBATION` block and the number of steps from the `STEP` block in the MD++ input file will lead to a lambda value larger than 1

Errors:

1. one of the essential blocks is missing (MD++): `STEP`, `BOUNDCOND`, `INITIALISE`, `FORCE`, `CONSTRAINT`, `PAIRLIST`, `NONBONDED`

2. there is no `VELOCITY` block in the coordinate file, but `NTIVEL` in the `INITIALISE` block of the MD++ input file specifies that the velocities should be read from file
3. non-zero `NTISHI` in the `INITIALISE` block of the MD++ input file specifies that the lattice shifts should be initialised, but zero `NTB` in the `BOUNDCOND` block specifies a vacuum simulation
4. there is no `LATTICESHIFT` block in the coordinate file, but `NTISHI` in the `INITIALISE` block of the MD++ input file specifies that the lattce shifts should be read from file
5. there is no `GENBOX` block in the coordinate file, but non-zero `NTB` in the `BOUNDCOND` block specifies a non-vacuum simulation
6. the number of the last atom given in the `FORCE` block of the MD++ input file is not equal to the total number of atoms calculated from the topology and `SYSTEM` block
7. the number of atoms calculated from the topology and `SYSTEM` block of the MD++ input file is not equal to the number of atoms in the `POSITION` block of the coordinate file
8. in the `PAIRLIST` block, the short-range cutoff `RCUTP` is larger than the long-range cutoff `RCUTL` (MD++)
9. no position restraints specification file is specified in the `mk_script` input file, but position restraining is switched on in the MD++ input file
10. no perturbation topology file is specified in the `mk_script` input file, but perturbation is switched on in the MD++ input file

Required Input Arguments

`@sys`	⟨system name⟩	
`@bin`	⟨GROMOS binary to use⟩	
`@dir`	⟨where the files should be (all filenames to be given relative to this)⟩	
`@version`	⟨md++ / promd⟩	
`@files`		
	`topo`	⟨molecular topology file (see Sec. 1.2)⟩
	`input`	⟨input file⟩
	`coord`	⟨initial coordinates⟩
	`refpos`	⟨reference positions⟩
	`posresspec`	⟨position restraint specifications⟩
	`disres`	⟨distance restraint specifications⟩
	`dihres`	⟨dihedral restraint specifications⟩
	`jvalue`	⟨j-value restraint specifications⟩
	`order`	⟨order parameter restraint specifications⟩
	`ledih`	⟨local elevation dihedral specifications⟩
	`pttopo`	⟨perturbation topology⟩

Optional input arguments

`@script`	⟨first script⟩ ⟨number of scripts⟩ (default: 1 1)
`@joblist`	⟨joblist file⟩
`@template`	⟨`mk_script` library file or filename template⟩
`@queue`	⟨which queue to use in `mk_script` library file⟩
`@remd`	⟨master / slave hostname port⟩ (replica exchange MD)
`@cmd`	⟨overwrite last command in `mk_script` library file⟩
`@force`	(write script regardless of errors)

Standard output

warnings and errors concerning the consistency checks

Additional output

scripts and input files for the requested MD simulations

2.19. `pdb2g96` (GROMOS++ program)

Program description:

Converts coordinates of a pdb file (Protein Data Bank) to coordinates in GROMOS format. The unit of the coordinates is converted from Å to nm. The order of the atoms in the pdb file does not necessarily correspond to the order of the atoms in the topology, but the residues should come in the proper order. The program identifies atoms and residues based on their names. Alternatives to the atom and residue names in the topology can be specified in a library file (see Sec. 4-7.3). The only requirement on residue numbers in the pdb file is that the residue number should change when going from one residue to the next. Mismatches between the topology and the pdb file are treated as follows:

1. If the expected residue according to the topology is not found, a warning is written out and the next residue in the pdb file is read in until a match with the topology is found.
2. Atoms that are expected according to the topology, but that are not found in the pdb file are written out in the coordinate file with coordinates (0.0, 0.0, 0.0). A warning is written to the standard error.
3. Atoms that are present in the pdb file, but not expected according to the topology are ignored, a warning is written to standard error.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pdb`	⟨pdb coordinates⟩
`@lib`	⟨library for atom and residue names⟩

Optional input arguments	
`@out`	⟨resulting GROMOS coordinates⟩ (optional, defaults to stdout)
`@outbf`	⟨write B factors and occupancies to an additional file⟩

Standard output

GROMOS coordinates for the atoms of the system

Additional output

none

2.20. `pert_top` (GROMOS++ program)

Program description:

Creates a perturbation topology to perturb specified atoms. A perturbation topology is written that defines a perturbation to alter the specified atoms into a specified atom types, charges and masses. Each of the arguments `@types`, `@masses` and `@charges` can be omitted. In this case the values from the topology are taken. If not sufficient values are given, the last given value is taken for all the remaining atoms.

Use program `pt_top` to convert the resulting perturbation topology to a different format or to a regular molecular topology.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1): atoms to be modified⟩
`@types`	⟨IACs of the perturbed atoms⟩
`@charges`	⟨charges of the perturbed atoms⟩
`@masses`	⟨masses of the perturbed atoms⟩

Optional input arguments

none

Standard output

perturbation topology

Additional output

none

2.21. `prep_eds` (GROMOS++ program)

Program description:

The topology file for EDS (dual topology!) is generated from $\mathcal{N}^{(s)}$'normal' topologies, where $\mathcal{N}^{(s)}$ is the number of end states. An end state is in EDS a molecule, e.g. a ligand. In the EDS topology, all states or molecules, respectively, are combined and excluded from another. The resulting molecular topology file is written out to a file called `com_eds.top`.

In the EDS perturbation topology, a molecule is 'visible' in one state and in all other states it consists of dummy atoms. For this the `MPERTATOM` block is used. The resulting perturbation topology file is written out to a file called `pert_eds.ptp`.

The argument `@inG96` converts GROMOS96 topologies to the current formats. On the other hand `outG96` converts topologies in the current format to the GROMOS96 format.

Required input arguments

`@topo`	⟨molecular topology files of end states⟩
`@numstat`	⟨number of end states $\mathcal{N}^{(s)}$⟩
`@param`	⟨index number of molecular topology file to take parameters from⟩
`@solv`	⟨index number of molecular topology file to take solvent from⟩

Optional input arguments

`@update_tree`	⟨switch for max. spanning tree update (required if `@form`=3)⟩
`@tree`	⟨file with old max. spanning tree (required if `@update_tree` is specified)⟩

Standard output

none

Additional output

combined molecular topology file (written to a file with name `com_eds.top`) and perturbation topology file (written to a file with name `pert_eds.ptp`) for EDS simulation (dual topology)

2.22. `prep_xray` (GROMOS++ program)

Program description:

Program prep_xray converts a crystallographic information file (CIF) containing reflection data into a GROMOS X-ray restraints specification file. Using a mapping file (map) it writes out the element names of the solute and solvent atoms according to their integer atom codes. The atoms' B-factors and occupancies are read from a special file (@bfactor) if requested or defaulted to 0.01nm^2 and 100%. The reflection list can be filtered according to the given resolution range. If no resolution is given, it is determined automatically. A random set of amplitudes is created for the computation of the free R factor.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@cif`	⟨crystallographic information file⟩
`@map`	⟨file with IAC-to-element name mapping⟩
`@bfactor`	⟨file with the B-factors and occupancies⟩
`@spacegroup`	⟨space group in Hermann-Mauguin format⟩
`@cell`	⟨cell descriptor: a, b, c, α, β, γ⟩
`@resolution`	⟨resolution range: minimum, maximum⟩
`@rfree`	⟨percentage taken for the free R factor⟩

Optional input arguments	
`@filter`	⟨filter amplitudes smaller than multiple of RMSD⟩
`@symmetrise`	⟨apply symmetry operations to reflections⟩
`@factor`	⟨factor to convert the length unit to Angstrom⟩

Standard output

crystallographic restraints specification data

Additional output

none

2.23. `prep_xray_le` (GROMOS++ program)

Program description:

Program prep_xray_le creates a X-ray local elevation file. It takes the side chains of the residues contained in the specified atoms. The side chains are defined in a special file (`@library`). It should contain the following block:

```
LESIDECHAIN
# name dim atom names
ARG 4 N CA CB CG CA CB CG CD CB CG CD NE CG CD NE CZ
ASN 2 N CA CB CG CA CB CG OD1
END
```

As the atom names define (dim) dihedral angles they have to be a multiple of four. The local elevation parameters (force constant, the number of bins of the grid, the functional form switch, the width of the potential energy function and its cutoff) are specified using `@leparam`. The X-ray parameters R^0 threshold and cutoff for R_{real} calculation are specified using `@xrayparam`.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@atoms`	⟨atoms to consider⟩
`@library`	⟨library file⟩

Standard output

crystallographic local elevation specification data

Additional output

none

2.24. `pt_top` (GROMOS++ program)

Program description:

Combines topologies with perturbation topologies to produce new topologies or perturbation topologies. Reads a topology and a perturbation topology to produce a new (perturbation) topology. The perturbation topology can contain a `PERTATOMPARAM`, or `MPERTATOM` block (see Sec. 4-3.3). The atom numbers in the perturbation topology do not need to match the numbers in the topology exactly. If the topology and perturbation topology do not match in their atom numbering, a shift can be applied using the `@firstatom` option.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pttopo`	⟨perturbation topology with `PERTATOMPARAM` or `MPERTATOM` block⟩
`@type`	⟨output format: `TOPO`, `PERTTOPO` or `PERTTOPO03`⟩
`@npt`	⟨sequence number of the perturbation in a `MPERTATOM` block to apply (0 = stateA)⟩
`@firstatom`	⟨atom specifier (see Sec. 1.3.1): first atom to which the perturbation will be applied⟩

Optional input arguments

none

Standard output

combined (perturbation) topology

Additional output

none

2.25. `ran_box` (GROMOS++ program)

Program description:

When simulating a molecular liquid, a starting configuration for the solvent molecules has to be generated. Program `ran_box` generates a starting configuration for the simulation of mixtures consisting of an unlimited number of components. The molecules are randomly placed in a cubic or a truncated octahedron box, in a random orientation. Note that for the generation of a starting configuration for the simulation of pure liquids and binary mixtures, the programs `build_box` and `bin_box` can alternatively be used (see Secs. 2.2 and 2.1, respectively).

Required input arguments	
`@topo`	⟨topologies of single molecule for each molecule type: topo1 topo2 ...⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@pos`	⟨coordinates of single molecule for each molecule type: pos1 pos2 ...⟩
`@nsm`	⟨number of molecules for each molecule type: nsm1 nsm2 ...⟩
`@dens`	⟨density of liquid (kg/m^3)⟩

Optional input arguments	
`@thresh`	⟨threshold distance in overlap check (nm) [default: 0.20 nm]⟩
`@layer`	⟨create molecules in layers (along z-axis)⟩
`@boxsize`	⟨boxsize⟩
`@fixfirst`	⟨do not rotate / shift first molecule⟩
`@seed`	⟨random number generator seed⟩

Standard output
coordinate file (cubic or truncated octahedron box, specified density)

Additional output
progress report (written to the standard error)

2.26. `ran_solvation` (GROMOS++ program)

Program description:

When simulating a molecule in solution or in a crystal containing solvent molecules, the atomic coordinates of the solvent molecules are to be generated if they are not available from experiment. Alternatively to `sim_box` (which solvates a solute in a pre-equilibrated box of a molecular liquid, see Sec. 2.28), `ran_solvation` can solvate a solute in a mixture consisting of an unlimited number of components. The program places the solute in the center of a box (rectangular or truncated octahedron) and generates a random distribution of solvent molecules around it, which are placed in a random orientation. The total number of solvent molecules is calculated based on the specified solvent density and the molar fractions. When calculating the solvent density, the excluded volume of the solute is taken into account as specified by the user. `ran_solvation` checks separately for solute-solvent and solvent-solvent overlap after every insertion. If any solute-solvent or solvent-solvent interatomic distance is smaller than the respective threshold distance specified by the user, the insertion trial is rejected. Note that for the optional input flags, either the box-size or the minimum solute-to-wall distance should be specified.

Required input arguments	
`@topo_u`	⟨molecular topology file (see Sec. 1.2) for the solute⟩
`@pos_u`	⟨input coordinate file for the solute to be solvated⟩
`@sev`	⟨solvent-excluded volume of the solute (nm^3) ⟩
`@topo_v`	⟨list of (single molecule) molecular topology files of solvents: topo_solv1 topo_solv2 . . . ⟩
`@pos_v`	⟨list of (single molecule) coordinate files of solvents: pos_solv1 pos_solv2 . . . ⟩
`@molf_v`	⟨mole fraction of each solvent: molf_solv1 molf_solv2 . . . ⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@dens`	⟨mass density of solvent mixture (kg/m^3)⟩

Optional input arguments	
`@minwall`	⟨minimum solute-to-wall distance(s)⟩
`@boxsize`	⟨length of box-edge(s)⟩
`@thresh_u`	⟨threshold interatomic distance in overlap check (solute - solvent); default: 0.40 nm⟩
`@thresh_v`	⟨threshold interatomic distance in overlap check (solvent - solvent); default: 0.20 nm⟩

Standard output

coordinate file for the solvated solute

Additional output

none

2.27. `red_top` (GROMOS++ program)

Program description:

For large molecular complexes, one would sometimes like to consider only a part of the many atoms, thereby reducing the computational effort required by a simulation. Programs `tstrip` and `filter` can filter an atomic coordinate file (see Secs. 4.64 and 4.29, respectively). Accordingly, program `red_top` can cut out parts of a molecular topology.

The user has to list atoms of the molecular topology to be reduced. All atoms, exclusions, bonds, bond angles etc. that involve atoms that are not in this list are removed. Note that to cut out all atoms within a sphere around a part of the system, one could first generate a list of the corresponding atoms using the program `pairlist` (see Sec. 5.5).

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2) to be reduced⟩
`@atoms`	⟨atoms in the system to keep⟩

Optional input arguments

none

Standard output

reduced topology file

Additional output

none

2.28. `sim_box` (GROMOS++ program)

Program description:

When simulating a molecule in solution or in a crystal containing solvent molecules, the atomic coordinates of the solvent molecules are to be generated, if they are not available from experiment. Program `sim_box` can solvate a solute in a pre-equilibrated box of solvent molecules. The file specifying the solvent configuration should contain a `GENBOX` block with the dimensions corresponding to the pre-equilibrated density. The solvent topology is read from the `SOLVENTATOM` block of the specified topology.

To prevent overlap between solute and solvent molecules, only solvent molecules for which the centre of geometry is at a minimum distance from any solute atom (which can be defined via the `@thresh` flag) are put into the box. Before solvating the solute molecules, the solute can be rotated 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, by giving the `@rotate` flag. The resulting box containing solute and solvent molecules can be either rectangular or a truncated octahedron. Its dimensions can be specified via the `@boxsize` flag. If this flag is given, the box dimensions are read in from the `GENBOX` block in the solute coordinate file. Alternatively, when the `@minwall` flag is given, the solute is put into a box filled with solvent molecules with box dimensions guaranteeing a minimum distance between any solute molecule and the box edges. Either one value can be specified for the `@minwall` flag, resulting in a cubic or truncated octahedron box, or three values can be specified, to generate a rectangular box. In the latter case, the solute molecules can be gathered on request (by specifying the `@gather` flag) and the `@rotate` flag must be given. Note that to solvate a solute in a triclinic box, one can use `sim_box` to generate a rectangular box and subsequently apply the appropriate symmetry transformations on the generated box using the program `cry` or use `sim_box` to generate a truncated octahedral box and convert it to a triclinic box using the program `unify_box`.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2) of the solute⟩
`@pbc`	⟨periodic boundary condition (r or t)⟩
`@pos`	⟨input coordinate file for the solute⟩
`@solvent`	⟨input coordinate file for the pre-equilibrated solvent⟩

Optional input arguments	
`@boxsize`	⟨use boxsize specified in input file⟩
`@minwall`	⟨minimum solute to wall distance⟩
`@thresh`	⟨minimum solvent to solute distance (nm) [default: 0.23 nm]⟩
`@gather`	⟨gather solute⟩
`@rotate`	⟨rotate solute: biggest axis along z, second along y⟩

Standard output

coordinate file of the solvated solute.

Additional output

none

CHAPTER 3

Minimizers and simulators

When performing an energy minimization (EM) or a molecular dynamics (MD) or stochastic dynamics (SD) simulation, the minimum requirement is the availability of a starting configuration and a molecular topology file containing the atomic masses and physical force-field data with respect to the molecular system (Chap. 2-5). The non-physical atomic interaction function $\mathcal{V}^{(spec)}$ (Chap. 2-9) needs not be specified, since it only serves special purposes. The different terms in $\mathcal{V}^{(spec)}$ represent different ways in which the motion of the atoms can be restrained or influenced:

- atom position restraining or fixing
- atom-atom distance restraining
- dihedral-angle restraining
- ^3J-coupling constant restraining
- S^2 order-parameter restraining
- X-ray structure factor amplitude, electron density or symmetry restraining
- distancefield distance restraining
- local-elevation interaction

Since application of these special forces is optional, the different types of special force data are kept in different files. When performing a SD simulation, atomic friction coefficients γ_i must be available. These can be given in an atomic friction coefficients file. Finally, when performing a free energy calculation a perturbation molecular topology is required specifying the change in Hamiltonian from state A to state B.

3.1. `md` (MD++ program)

Program description:

Program `md` carries out an EM, MD or SD simulation for a molecular system consisting of solute and solvent (molecules Chaps. 2-11, 2-12 and 2-13). Periodic boundary conditions can be applied Sec. 2-4.1. Bond lengths and in solvent molecules also bond angles can be treated as holonomic constraints (Chap. 2-10). Temperature and pressure can be maintained by weak coupling of different degrees of freedom (solute internal plus rotational, solute translational, solvent) to different temperature baths and of the box dimensions (isotropic or along x-, y- and z-axis) to different pressure baths. Translation of and rotation around the centre of mass of the molecular system can be monitored and halted. Atomic coordinates, velocities, energies, pressure, volume and free energy data can be written to various trajectory files for later analysis.

Required input arguments:

`@topo`	R	⟨molecular topology file (see Sec. 1.2)⟩
`@conf`	R	⟨coordinates and restart data⟩
`@input`	R	⟨input parameters⟩
`@fin`	W	⟨final configuration⟩
`@trc`	W	⟨coordinate trajectory⟩

Optional input arguments:

`@pttopo`	R	⟨perturbation topology file⟩
`@trc`	W	⟨coordinate trajectory⟩
`@trv`	W	⟨velocity trajectory⟩
`@trf`	W	⟨force trajectory⟩
`@trs`	W	⟨special trajectory⟩
`@tramd`	W	⟨RAMD trajectory⟩
`@tre`	W	⟨energy trajectory⟩
`@bae`	W	⟨block averaged energy trajectory⟩
`@trg`	W	⟨free energy trajectory⟩
`@bag`	W	⟨block averaged free energy trajectory⟩
`@posresspec`	R	⟨position restraints specification⟩
`@refpos`	R	⟨position restraints⟩
`@distrest`	R	⟨distance restraints specification⟩
`@dihtrest`	R	⟨dihedral restraints specification⟩
`@jval`	R	⟨J-value restraints specification⟩
`@order`	R	⟨S2-value restraints specification⟩
`@xray`	R	⟨Xray restraints specification⟩
`@lud`	R	⟨local elevation umbrella database⟩
`@led`	R	⟨local elevation coordinate specification⟩
`@friction`	R	⟨atomic friction coefficients⟩
`@print`		⟨print additional information⟩
`@anatraj`	R	⟨re-analyze trajectory⟩
`@verb`		⟨control verbosity⟩
`@version`		⟨print version information⟩

Standard output

general information about the running simulations, printed to the standard output

Additional output

different files (trajectories) depending on the input flags specified in the input file

3.2. `repex_mpi` (MD++ program)

Program description:

This program is used to run replica exchange simulations. See `md` (Sec. 3.1) for the documentation of all command line arguments. Additional command line arguments are reported below.

Required input arguments:

`@repout`	W	⟨output file for replicas⟩
`@repdat`	W	⟨replica data file⟩

Optional input arguments:

none

Standard output

information from master about timings, printed to the standard output

Additional output

general information about the running simulations is printed to the output file specified under `@repout`, different files (trajectories) depending on the input flags specified in the input file

3.3. `eds_2box` (MD++ program)

Program description:

This program is used to run twin-system EDS simulations. The command line arguments are reported below.

Required input arguments:

`@topo1`	R	⟨molecular topology file for box 1 (see Sec. 1.2)⟩
`@topo2`	R	⟨molecular topology file for box 2 (see Sec. 1.2)⟩
`@conf1`	R	⟨coordinates and restart data for box 1⟩
`@conf2`	R	⟨coordinates and restart data for box 2⟩
`@input1`	R	⟨input parameters for box 1⟩
`@input2`	R	⟨input parameters for box 2⟩
`@pttopo1`	R	⟨perturbation topology file for box 1⟩
`@pttopo2`	R	⟨perturbation topology file for box 2⟩
`@tre1`	W	⟨energy trajectory for box 1⟩
`@tre2`	W	⟨energy trajectory for box 2⟩
`@fin1`	W	⟨final configuration for box 1⟩
`@fin2`	W	⟨final configuration for box 2⟩

Optional input arguments:

`@trc1`	W	⟨coordinate trajectory for box 1⟩
`@trc2`	W	⟨coordinate trajectory for box 2⟩
`@trv1`	W	⟨velocity trajectory for box 1⟩
`@trv2`	W	⟨velocity trajectory for box 2⟩
`@trf1`	W	⟨force trajectory for box 1⟩
`@trf2`	W	⟨force trajectory for box 2⟩

Standard output

general information about the running simulations, printed to the standard output

Additional output

different files (trajectories) depending on the input flags specified in the input file

CHAPTER 4

Analysis of trajectories (postprocessing)

4.1. bar (GROMOS++ program)

Program description:

Program bar calculates free energy differences between (at least) two states using Bennett's Acceptance Ratio method.[1] The free energy between two states, i and j, is given by

$$\Delta G(\lambda_i \rightarrow \lambda_j) = k_B T \ln \frac{\langle f(E(\lambda_i) - E(\lambda_j) + C)\rangle_{\lambda_j}}{\langle f(E(\lambda_j) - E(\lambda_i) - C)\rangle_{\lambda_i}} + C \tag{4.1}$$

where $f(x)$ denotes the Fermi function

$$f(x) = \frac{1}{1 + exp(x/k_B T)} \tag{4.2}$$

and

$$C = k_B T ln \frac{N_j}{N_i} + \Delta G(\lambda_i \rightarrow \lambda_j). \tag{4.3}$$

These equations are solved self-consistently, using a numerically stable implementation.[2] The convergence criterion (relative change in free energy between iterations; `@convergence`) and maximum number of iterations (`@maxiterations`) can be specified.

Time series of the energies (with lenght N_i and N_j, respectively) are read in from one file per simulated state, which also contains the energies of the neighbouring states. These files may be generated using program `ext_ti_ana` with option `@bar_data` (see Sec. 4.27).

Error estimates are standardly determined from the ensemble averages in the equations above. Optionally, a bootstrap error can be computed, where the calculation is repeated the indicated number of times (option `@bootstrap`), with random samples of the original time series. The standard deviation of the bootstrap estimates is reported.

The program also computes the overlap integral from distributions of the energy differences $P_i(\Delta E)$ and $P_j(\Delta E)$, using

$$OI = 2 \sum_{\Delta E} \frac{P_i(\Delta E) P_j(\Delta E)}{P_i(\Delta E) + P_j(\Delta E)} \tag{4.4}$$

with the sum running over all bins of the distribution. The distributions may be written out to separate files using option `@printdist`.

Required input arguments	
`@files`	⟨energy files from `ext_ti_ana` (see Sec. 4.27)⟩
`@temp`	⟨absolute temperature⟩

Optional input arguments	
`@maxiterations`	⟨maximum number of iteractions (default: 500)⟩
`@convergence`	⟨relative change in free energy for convergence (default: 1E-5)⟩
`@printdist`	⟨write out distributions⟩
`@bootstrap`	⟨number of bootstrap estimates for error estimates (default: 0)⟩

Standard output

Overview of free energy calculation

Additional output

Distributions of the individual energy differences

4.2. `bilayer_dist` (GROMOS++ program)

Program description:

Distributions along the bilayer normal are useful to characterise membrane systems. This program calculates the distribution of a given set of atoms with respect to the center of mass of another given set of atoms (usually, this set will include all bilayer atoms to give a distribution of atoms with respect to the bilayer center of mass).

By default, all distributions are normalised to unity. However, the user may be interested in density distributions. In this case, the flag `@density` must be included. The user can also obtain mass or electron densities with the help of the `@mult` argument. This will multiply the distribution by a given number. If the user wants to calculate electron density profiles the `@mult` and `@density` arguments must be given. In some cases, the bilayer is not centered in the periodic box and the center of mass will not be in between the two layers but in the bulk of the solvent instead. The argument `@translate` can be used to circumvent this problem.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1): atoms for c.o.m calculation⟩
`@selection`	⟨atom specifier (see Sec. 1.3.1): atoms to consider⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@time`	⟨time and dt⟩
`@grid`	⟨integer (default: 100)⟩
`@translate`	⟨translate box⟩
`@mult`	⟨double (default: 1)⟩
`@density`	⟨calculate density distribution⟩

Standard output

distribution of selected atoms

Additional output

none

4.3. `bilayer_oparam` (GROMOS++ program)

Program description:

Deuterium order parameters (S_{CD}) can be derived from deuterium quadrupole splitting experiments and have used to study biological membranes. The corresponding carbon-hydrogen order parameters (S_{CH}) can be calculated by computing the correlation functions describing the reorientation of the carbon-hydrogen vectors. More precisely, for each methylene group along the chain, an order parameter tensor $\underline{\mathbf{S}}$ can be defined as

$$S_{ij} = \frac{1}{2}\langle 3cos\theta_i \ cos\theta_j - \delta_{ij}\rangle, \tag{4.5}$$

where θ_i is the angle between the i^{th} local molecular axis (x', y' or z') and the bilayer normal (z-axis), δ_{ij} is the Kronecker delta symbol and $\langle ... \rangle$ stands for trajectory averaging. As a convention, for the n^{th} methylene group C_n, the direction of the vector $C_{n-1} - C_{n+1}$ is taken as z', the direction of the vector normal to z' in the plane C_{n-1}, C_n, and C_{n+1} defines y', while x' is the direction of the vector perpendicular both to z' and y'. The quantity $S_{CH} = -(2/3S_{xx} + 1/3S_{yy})$ is the value to be compared with the experimental S_{CD} value.

The order parameters are calculated with respect to a fixed orientational vector (corresponding to the direction of the experimental magnetic field; usually taken along the bilayer normal) given by the flag `@refvec`.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1) for which order parameters will be calculated⟩
`@refvec`	⟨reference orientational vector⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@time`	⟨time and dt⟩

Standard output

order parameter tensor components and carbon-hydrogen order parameter for selected atoms

Additional output

none

4.4. `cluster` (GROMOS++ program)

Program description:

Program `cluster` performs a conformational clustering based on a similarity matrix, such as calculated by the program `rmsdmat` (see Sec. 4.54). The program uses the clustering algorithm of Daura.[3] Structures with RMSD values smaller than a user specified cutoff are considered to be structural neighbours. The structure with the highest number of neighbours is considered to be the central member of the cluster of similar structures forming a conformation. After removing all structures belonging to this first cluster, the procedure is repeated to find the second, third etc. most populated clusters.

One specific structure can be forced to be the central member structure of the first cluster, this can also be the reference structure, by specifying structure number 0. The clustering can be performed on a subset of the matrix, by specifying the maximum number of structures to consider. This allows for an assessment of the development of the number of clusters over time.

Depending on the settings used for program `rmsdmat`, the flags `@human` and `@big` may need to be specified to ensure proper reading of the matrix.

Clusters may be further analysed using program `postcluster` (see Sec. 4.43).

Required input arguments

`@rmsdmat`	⟨RMSD matrix file name⟩
`@cutoff`	⟨cutoff⟩
`@time`	⟨t0⟩ ⟨dt⟩

Optional input arguments

`@maxstruct`	⟨maximum number of structures to consider⟩
`@human`	(use a human readable matrix)
`@force`	⟨structure⟩ (force clustering on the indicated structure, 0 is the reference)
`@big`	(when clustering more than 50'000 structures)

Standard output

listing of the size of every cluster that was found

Additional output

Two additional files are written to disk: `cluster_structures.dat` and `cluster_ts.dat`. `cluster_structures.dat` contains for every cluster its size, the central member and a listing of all structures in the cluster. `cluster_ts.dat` contains a time series of the clusters. At every point in time the structure number and cluster number are given.

4.5. `cog` (GROMOS++ program)

Program description:

Program `cog` calculates the centre of geometry (cog) or centre of mass (com) of the solute molecule(s) in the specified frames of the input trajectory file(s), and writes out a single trajectory file in which the position of the cog or com either replaces the atomic coordinates of the solute molecule(s) or are appended directly after the coordinates of the last atom of the solute molecule(s).

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨input trajectory files⟩

Optional input arguments	
`@nthframe`	⟨write every nth frame (default: 1)⟩
`@cog_com`	⟨calculate center of geometry (cog) or centre of mass (com) of solute molecules (default: cog)⟩
`@add_repl`	⟨add (add) the position of the cog/com or replace (repl) the solute coordinates with the position of the cog/com (default: repl)⟩

Standard output

single trajectory file

Additional output

none

4.6. `cos_dipole` (GROMOS++ program)

Program description:

Program `cos_dipole` calculates the average dipole moments over a selected set of molecules, taking into account also polarizable sites. Standardly it outputs the magnitude of the average total, fixed and induced molecular dipoles, but if needed, the x-, y- and z-components can be written to a file by specifying the `@xyz` flag.

Note that the dipole moment is only well-defined for systems consisting of neutral molecules. If the system carries a net-charge, the dipole moment will depend on the position of the origin. In cases where the overall system is neutral but contains ions, the dipole moment will depend on which periodic copy of the ions is taken. In these cases, the program issues a warning that results will critically depend on the choice of gathering method.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨trajectory files⟩
`@trs`	⟨special trajectories with COS displacements⟩

Optional input arguments	
`@time`	⟨time and dt⟩
`@molecules`	⟨solute molecules to average over, e.g. 1-5,10,17⟩
`@fac`	⟨conversion factor for the unit of the dipole, default: 1; use 48.032045 to convert from e∗nm to Debye⟩
`@xyz`	⟨filename for writing out dipole x-,y-,z-components, if no name given: Mxyz.out⟩
`@solv`	⟨include solvent⟩

Standard output

time series of the magnitude of the average total, fixed and induced molecular dipoles

Additional output

output file `Mxyz.out` contains average x-, y- and z-components of the average total, fixed and induced molecular dipoles

4.7. `cos_epsilon` (GROMOS++ program)

Program description:

Program `cos_epsilon` calculates the dielectric permittivity, $\epsilon(0)$, of a simulation box from a Kirkwood-Fröhlich type of equation as derived by Neumann,**4** taking into account also the polarizable centers.

$$(\epsilon(0)-1)\frac{2\epsilon_{rf}+1}{2\epsilon_{rf}+\epsilon(0)} = \frac{\langle \boldsymbol{M}^2\rangle - \langle \boldsymbol{M}\rangle^2}{3\epsilon_0 \mathcal{V} k_B T} \quad , \tag{4.6}$$

where $\boldsymbol{M}$ is the total dipole moment of the system, ϵ_0 the dielectric permittivity of vacuum, ϵ_{rf} is a reaction-field epsilon value, $\mathcal{V}$ is the volume and k_BT is the absolute temperature multiplied by the Boltzmann constant.

If the `@autocorr` flag is given, the program also writes out the normalized autocorrelation function $\theta(\tau)$,

$$\theta(\tau) = \frac{< \vec{M}(t)\cdot\vec{M}(t+\tau) >_t}{< M(t)^2 >_t} = exp(-\frac{t}{\tau_\phi}) \tag{4.7}$$

from which the Debye relaxation time τ_D can be calculated:**4**

$$\tau_D = \frac{2\epsilon_{rf}+\epsilon(0)}{2\epsilon_{rf}+1}\tau_\phi \tag{4.8}$$

Using the `@truncate` flag, the autocorrelation output can be truncated when the number of independent contributing frames drops below a given threshold.

Note that the total dipole moment of the system is only well-defined for systems consisting of neutral molecules. If the system carries a net-charge, the dipole moment will depend on the position of the origin. In cases where the overall system is neutral but contains ions, the dipole moment will depend on which periodic copy of the ions is taken. In these cases, cos_epsilon issues a warning that results will critically depend on the choice of gathering method.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@e_rf`	⟨reaction field epsilon⟩
`@temp`	⟨temperature⟩
`@traj`	⟨trajectory files⟩
`@trs`	⟨special trajectories with COS displacements⟩

Optional input arguments	
`@time`	⟨time and dt⟩
`@fac`	⟨conversion factor for the unit of the dipole, default: 1; use 48.032045 to convert from e∗nm to Debye⟩
`@autocorr`	⟨filename for storing time autocorrelation⟩
`@truncate`	⟨minimum number of independent contributing frames after which to truncate the correlation function⟩

Standard output

time series of the box dipole, average molecular dipole and epsilon

Additional output

output file `Mcorr.out` contains the box dipole autocorrelation data

4.8. cry_rms (GROMOS++ program)

Program description:

Program `cry_rms` is used to compute atom positional RMSDs and RMSFs between the asymmetric units within a unit cell of a crystalline system. The symmetry operations are either specified using a special file (`@spec`, `@factor`) or by the space group (`@spacegroup`). In order to identify the individual asymmetric units (ASUs) an @ref AtomSpecifier AtomSpecifier to the first atom of every ASU have to be given (`@asuspec`). If an RMSD is requested (`@atomsrmsd`), the atom positional RMSD between all the asymmetric units is printed in separate columns. The RMSF is calculated for the requested atoms (`@atomsrmsf`) while taking also the fluctuations of the symmetry related copies into account.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨input trajectory files⟩
`@asuspec`	⟨AtomSpecifier to the first atom of every asymetric unit⟩

Optional input arguments	
`@spec`	⟨specification file for the symmetry transformations⟩
`@factor`	⟨conversion factor for distances⟩
`@spacegroup`	⟨space group in Hermann-Mauguin format⟩
`@atomsrmsd`	⟨AtomSpecifier used for RMSD calculation⟩
`@atomsrmsf`	⟨AtomSpecifier used for RMSF calculation⟩

Standard output

Time-series of the atom-positional RMSD between of specified atoms with respect to all asymmetric units. Atom-positional RMSF of the asymmetric unit taking all asymmetric units of the unit cell into account.

Additional output

none

4.9. `dfgrid` (GROMOS++ program)

Program description:

Program `dfgrid` calculates a distancefield grid analogously to the way it is done in md++, where distancefield (DF) distances can be used as restraints and reaction coordinates in path pulling methods[5]. The DF method is based on the mapping of distances from a target point on a grid, where grid points that overlap with the protein are penalized. As a result the shortest path for a ligand to the target point will never go through the protein, making it less likely to get stuck in a dead end.

Program `dfgrid` calculates the grid for any given snapshot, facilitating the visualization and analysis. It writes out a coordinate file which contains both the input coordinates and the distancefield grid, in addition to the target (zero-distance) point which will often be a virtual atom. When choosing pdb as output format, the distances on the grid will be written to the b-factor column for easy visualization.

The use of either `@stride` or `@frames` to reduce the analysis to a few snapshots is recommended.

The following flags are defined in the same way as in the md++ parameter and distance restraints files: `@gridspacing`, `@proteinoffset`, `@proteincutoff` and `@smooth` (see Sec. 2-9.12). `@proteinatoms` has the same function as the corresponding md++ parameter, but here the atom selection is specified in the form of an atomspecifier.

`@max` allows to specify a maximum distance, grid points to which higher distances are mapped will not be written to the coordinate file. `@protect` protects grid points within a certain radius from the target point from being flagged as protein.

With the `@distatoms` flag you can specify (virtual) atoms for which the df distance will be printed in standard output and for which the shortest df path will be added to the output coordinates for visual inspection. If one is only interested in the distances, `@nogrid` will prevent writing of the coordinate file.

Required input arguments	
`@topo`	⟨molecular topology file ⟩
`@pbc`	⟨boundary type [gather method]⟩
`@atom`	⟨(virtual) atom specifier for the target (zero-distance) point⟩
`@gridspacing`	⟨grid spacing ⟩
`@proteinoffset`	⟨penalty for being in the protein ⟩
`@proteincutoff`	⟨cutoff to determine gridpoints within the protein ⟩
`@proteinatoms`	⟨last atom considered as protein⟩
`@traj`	⟨input trajectory files⟩

Optional input arguments	
`@max`	⟨maximum distance: do not write out grid points with higher distances (default: 1)] ⟩
`@smooth`	⟨number of rounds to smoothen the forces at the edge of the protein ⟩
`@protect`	⟨radius around the target atom that will not be flagged as protein ⟩
`@outformat`	⟨output coordinates format ⟩
`@notimeblock`	⟨do not write timestep block ⟩
`@time`	⟨time and dt⟩
`@stride`	⟨write every nth frame (default: 1)⟩
`@frames`	⟨select frames to write out, starts at 0 (default: 0) ⟩
`@distatoms`	⟨(virtual) atom specifier for atoms for which to output the df distance⟩
`@nogrid`	⟨do not write out grid coordinate file⟩

Standard output

none

Additional output

coordinates for the input system, DF grid and target point in the specified coordinate file format

4.10. `dfmult` (GROMOS++ program)

Program description:

Calculates free energy differences between multiple states A and B from an EDS simulation of a reference state R according to

$$\begin{aligned}\Delta\mathcal{F}_{BA} &= \mathcal{F}_B - \mathcal{F}_A = \mathcal{F}_B - \mathcal{F}_R - (\mathcal{F}_A - \mathcal{F}_R) = \Delta\mathcal{F}_{BR} - \mathcal{F}_{AR} \\ &= -\beta^{-1} ln \frac{\langle exp[-\beta(\mathcal{V}_B - \mathcal{V}_R)]\rangle_R}{\langle exp[-\beta(\mathcal{V}_A - \mathcal{V}_R)]\rangle_R}\end{aligned} \tag{4.9}$$

The program reads in energy time series generated by `ene_ana`. It provides an error estimate (err) which is based on calculation of the (co)variances and the statistical inefficiency as described in[6]. The implementation closely follows the Python implementation provided by the authors. When calculating averages and uncertainties special care is taken in order to avoid overflow (see[7]).

Required input arguments	
`@temp`	⟨temperature of the system⟩
`@stateR`	⟨energy time series of the reference state R⟩
`@endstates`	⟨energy time series of the end states⟩

Optional input arguments

none

Standard output

free energy differences between end states and reference state, and between end states

Additional output

none

4.11. `disicl` (GROMOS++ program)

Program description:

Program `disicl` classifies secondary structure elements in proteins and nucleic acids based on dihedral angles.[8,9] Angle, region and class definitions are read from a user-specified library file (see Sec. 4-7.10). The program will warn about overlapping regions and region limits that are outside the chosen periodic range and will abort if two classes have the same definition.

The program writes out classification statistics per residue and averaged over all residues (stat_disicl.out). Timeseries are written for each class (class_XXX.dat) and for the dihedral angles (ts_disicl.dat).

In the output files the determined class will be assigned to the residue 0 (as defined in the DSCLANG block) of the first region contributing to the classification.

The program provides an option (pdbstride) to write out pdb files containing class information in the b-factor column for visualization using the "color by b-factor"-function in your favorite visualization software. An additional pdb is created (colorlegend.pdb), which can be used as a kind of legend for the class color code when loaded into the visualization software together with the output pdbs.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@lib`	⟨library file⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@time`	⟨time and dt⟩
`@atoms`	⟨atoms to include (default: all atoms)⟩
`@skip`	⟨skip n first frames⟩
`@stride`	⟨take every n-th frame⟩
`@periodic`	⟨dihedral angle periodic range (default: -180 180)⟩
`@nots`	(do not write time series)
`@pdbstride`	⟨write out pdb coordinates every n steps (has to be >= stride)⟩

Standard output

timeseries of the dihedrals; timeseries and statistics of the classification

Additional output

pdb files with class information in the b-factor column

4.12. `dg_ener` (GROMOS++ program)

Program description:

Program `dg_ener` applies the perturbation formula to calculate the free energy difference between two states A and B. It reads in the output of program `ener` (section Sec. 4.22), which can be calculated for the same trajectory using two different Hamiltonians. The free energy difference is calculated as

$$\Delta G_{BA} = -k_B T \ln \langle e^{-(\hat{\mathcal{H}}_B - \hat{\mathcal{H}}_A)/k_B T} \rangle_A \tag{4.10}$$

where the average is over all entries of the energy files that are specified and the Hamiltonians are taken from the last column of these files.

Required input arguments

`@temp`	⟨temperature for perturbation ⟩
`@stateA`	⟨energy files for state A ⟩
`@stateB`	⟨energy files for state B ⟩

Optional input arguments

`@col`	⟨numbers of the columns to use from file A and B [default: last] ⟩

Standard output

For every line in the energy files, the program writes out the energy difference and the Boltzmann probability for that particular frame. The last column contains the current estimate of the free-energy difference.

Additional output

none

4.13. `dGslv_pbsolv` (GROMOS++ program)

Program description:

Progam `dGslv_pbsolv` will compute two electrostatic components of the solvation free energy, namely one for Coulombic interactions under non-periodic boundary conditions (CB/NPBC) and one for the user-specified electrostatics scheme (lattice sum, LS or reaction field, RF), under periodic boundary conditions (PBC). The solute will be centered in the computational box, with its center of geometry.

In the following, the abbreviation ΔG_{chg} will be used to denote an electrostatic component of the solvation free energy. ΔG_{chg} will be computed for a user-specified group of atoms. Note that all atoms of the solute topology block will be nonpolarizable, i.e. they will be assigned a relative dielectric permittivity of one.

$\Delta G_{chg}^{CB/NPBC}$ and $\Delta G_{chg}^{LS/PBC}$ will both be computed from two calculations employing finite difference (FD) algorithms.**[10]** One calculation is carried out with a permittivity appropriate for the solvent (ϵ_{sol}), the other calculation is carried out under vacuum conditions (ϵ_0). The differences between both is the solvation free energy. The resulting correction for the LS scheme is:

$$\Delta G_{corr}^{LS/PBC} = \Delta G_{chg}^{CB/NPBC(FD)} - \Delta G_{chg}^{LS/PBC(FD)} \tag{4.11}$$

with

$$\Delta G_{chg}^{env} = \Delta G_{chg;\epsilon_{sol}}^{env} - \Delta G_{chg;\epsilon_0}^{env} \tag{4.12}$$

where *env* can be $CB/NPBC(FD)$ or $LS/PBC(FD)$.

$\Delta G_{chg}^{RF/PBC}$ will be computed from a FFT algorithm.**[11,12]** For the RF scheme, the user should use the corresponding LS calculation to compute a correction to cancel possible grid discretization errors. That is, the resulting correction is:

$$\Delta G_{corr}^{RF/PBC} = \Delta G_{chg}^{CB/NPBC(FD)} - \Delta G_{chg}^{LS/PBC(FD)} + \Delta G_{chg}^{LS/PBC(FFT)} - \Delta G_{chg}^{RF/PBC(FFT)} \tag{4.13}$$

In the LS-scheme, tinfoil boundary conditions are used and a hat charge shaping function will be used. In the RF-scheme, a user-specified relative dielectric permittivity is used. Note that a relative dielectric permittivity of one implies no application of a reaction-field correction.

As an alternative, PQR files can be used instead of GROMOS coordinate files and topologies. PQR files are PDB files with the temperature and occupancy columns replaced by columns containing the per-atom charge and radius.

Required input arguments

if used with a GROMOS coordinate file and topology:

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@coord`	⟨g96 coordinates⟩
`@probeIAC`	⟨integer atom code to take for radius calculation (for water, it would be 4 or 5 depending on the ff)⟩
`@atoms`	⟨atoms to include⟩
`@atomsTOcharge`	⟨atoms to charge⟩
`@rminORsigma`	⟨which radii to use: rmin (0) or sigma (1); default 0⟩

if used with a PQR file:

`@pqr`	⟨pqr file⟩
`@coordinates`	⟨box coordinates in X,Y,Z direction (in nm) that were used in the simulation⟩
`@atoms`	⟨atoms to include; the molecule (as used in gromos-standard format) can be scipped (e.g. simply 'a' is enough to include all atoms)⟩
`@atomsTOcharge`	⟨atoms to charge; the molecule (as used in gromos-standard format) can be scipped (e.g. simply 1-5,7 is enough to include atoms 1 to 5 and 7)⟩

general input:

`@schemeELEC`	⟨electrostatics scheme: LS or RF⟩
`@rcut`	⟨cutoff distance in nm (ONLY USED IF scheme==RF)⟩
`@epsRF`	⟨reaction field relative dielectric permittivity (ONLY USED IF scheme==RF)⟩
`@epsSOLV`	⟨solvent relative dielectric permittivity of the employed solvent model⟩
`@gridspacing`	⟨grid spacing in nm⟩

Optional input arguments

`@epsNPBC`	⟨relative dielectric permittivity for NPBC calculation; default: 78.4⟩
`@maxiter`	⟨maximum number of iteration steps; default: 600⟩
`@cubesFFT`	⟨number of cubes in the fast Fourier transformation for boundary smoothing; default: 4⟩
`@probeRAD`	⟨probe radius in nm; default 0.14 (for water)⟩
`@radH`	⟨your desired hydrogen radius in nm; default 0.05⟩
`@radscal`	⟨scale non-H radii with this factor (in case you want to play with radii); default 1.0⟩
`@verbose`	⟨path to log file to document status and errors⟩

Standard output

Additional output

none

4.14. `diffus` (GROMOS++ program)

Program description:

Program `diffus` calculates the diffusion of the centre-of-geometry of a specified set of atoms. Firstly, the mean square displacements ($\Delta(t)$) are calculated over all considered molecules and over multiple time averages.

$$\Delta(t) = \frac{1}{N_m} \sum_{i=1}^{N_m} \langle [\mathbf{r}_i(t+\tau) - \vec{\mathbf{r}}_i(\tau)]^2 \rangle_{\tau \leq t_{av}-t} \tag{4.14}$$

where N_m is the total number of molecules (or atoms) considered in the analysis, and t_{av} is the duration of the averaging block.

According to the Einstein expression, the function $\Delta(t)$ should be approximately linear and in practice, the diffusion could be obtained from the slope of the $\Delta(t)$ devided by $2N_d t$:

$$D = \lim_{t \to \infty} \frac{\Delta(t)}{2N_d t} \tag{4.15}$$

where N_d is the number of considered dimensions (3 for 3D vectors $\mathbf{r}_i$). The slope of the $\Delta(t)$ is obtained from linear least-square fit (LSF). The diffus program makes an automatic LSF considering the whole time range of $\Delta(t)$, which might not be a reasonable approach due to the poor statistics for bigger values of t. It is recommended that the user analyzes the shape of $\Delta(t)$ and performs the LSF considering only the region of linearity.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@dim`	⟨dimensions to consider⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1): atoms to follow⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@time`	⟨time and dt⟩

Standard output

The value for the diffusion D obtained from LSF and the corresponding R^2 of the LSF are printed to the standart output. The outputfile diffusdp.out contains the time series of the mean square displacement $\Delta(t)$.

Additional output

none

4.15. `dipole` (GROMOS++ program)

Program description:

Program `dipole` will calculate and print the dipole moment of molecules. By default, the program will take all solute atoms into account, but the user can also specify a set of atoms. The dipole moment of a set of atoms carrying a net-charge is ill-defined and depends on the position of the origin. For these cases, the program allows the user to move the centre of geometry of the atoms to the origin.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@time`	⟨time and dt⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1) atoms to include⟩ (default: all solute)
`@cog`	(move molecule to centre of geometry)

Standard output

time series and averages of the magnitude of the dipole moment and its components

Additional output

none

4.16. `ditrans` (GROMOS++ program)

Program description:

Dihedral angle transitions can be monitored during the course of a simulation using MD++. Even though in many cases a molecular trajectory file will not contain every structure of the simulation, dihedral angle transitions can also be determined *a posteriori* from such a trajectory using program `ditrans`. This program can also write the time series of dihedral angles without taking the inherent periodicity of a dihedral angle into account, but rather allow for dihedral angle values below 0° or above 360°.

The program determines the position of maxima and minima in the dihedral angle potential energy function based on the phase shift and multiplicity given in the topology. Energy barriers arising from alternative terms, such as non-bonded interactions, which may in theory shift the position of energy minima and maxima of the dihedral angle are not taken into account.

Two different criteria can be used to count dihedral angle transitions, as described in the manual. A strict criterion only counts a transition once a dihedral angle passes beyond the minimum of an adjacent energy well to prevent counting of short lived transitions of the maximum dividing the two energy wells. Because of a possibly sparse sampling of data in a molecular trajectory, this criterion may be too restrictive. As an alternative a transition can also be counted as soon as a dihedral angle is seen to cross the maximum separating two energy wells.

The (standard) output can be restricted to the number of observed dihedral angle transitions for every dihedral angle that was specified or can be extended to information on every transition encountered.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@prop`	⟨property specifier (see Sec. 1.3.3)⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@time`	⟨time and dt⟩
`@strict`	(use GROMOS96 transition criterion)
`@verbose`	(print out every encountered transition)
`@tser`	⟨file name⟩ (extended time series)

Standard output

number of dihedral angle transitions for every specified dihedral angle

Additional output

time series of dihedral angles without periodicity restrictions (optional)

4.17. `dssp` (GROMOS++ program)

Program description:

Program `dssp` monitors secondary structure elements for protein structures over a molecular trajectory. The definitions are according to the DSSP rules defined by Kabsch and Sander[13]. Within these rules it may occur that one residue is defined as being part of two different secondary-structure elements. In order to avoid duplicates in the output, the following priority rules are applied: Beta Sheet/Bridge > 4-helix > 5-helix > 3-helix > H-bonded turn > Bend. As a consequence, there may be, for instance, helices that are shorter than their minimal length.

The program summarizes the observed occurrences of the secondary structure elements and averages the different properties over the protein. In addition time series for every type of secondary structure element are written to file.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1) for the protein⟩
`@time`	⟨time and dt⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@nthframe`	⟨write every nth frame⟩	(default is 1)

Standard output

summary of the occurrences of all selected residues within different secondary structure elements

Additional output

time series of the occurrences of all structural elements are written to different files: `Bend.out`, `Beta-Bridge.out`, `Beta-Strand.out`, `3-Helix.out`, `4-Helix.out`, `5-Helix.out` and `Turn.out`

4.18. `eds_update_1` (GROMOS++ program)

Program description:

Calculates iteratively the EDS parameters E^R and s from energy time series of a number (`@numstat`) of end states (`@vy`) and the reference state R (`@vr`). The form of the used Hamiltonian (single s = 1, multiple s = 2, maximum spanning tree s = 3) has to be specified (`@form`), as the number of s parameters depends on the functional form. For `@form` = 1, only a single s parameter is calculated (only recommended for $\mathcal{N}^{(s)}$ = 2), for `@form` = 2 $\mathcal{N}^{(s)}$ ($\mathcal{N}^{(s)}$-1)/2 s parameters are calculated and for `@form` = 3 ($\mathcal{N}^{(s)}$-1) s parameters, respectively. There are always $\mathcal{N}^{(s)}$ energy offset parameters E^R, independent of the functional form. The same number of old parameters have to be given (s and E^R).
If a maximum spanning tree is used as functional form (`@form` = 3), an initial tree must be specified (`@tree`) and if this tree shall be updated along with the parameters (`@update_tree` = 1) or not (`update_tree` = 0).

The s parameters are calculated using

$$ln \sum_{j=1,j\neq i}^{M} \left[\left(\left\langle e^{-\beta(|\Delta\mathcal{V}_{ji}|-\Delta {E^R}_{ji})} \right\rangle_i \right)^s \right] = ln(M-1) - 1 \tag{4.16}$$

where M = $\mathcal{N}^{(s)}$ for `@form` = 1, and M = 2 for `@form` = 2,3, respectively.

The energy offset parameters are calculated using

$$ {E^R}_i(new) = -\beta^{-1} \cdot ln \left\langle \left(1 + \sum_{j=1,j\neq i}^{\mathcal{N}^{(s)}} e^{-\beta(\Delta\mathcal{V}_{ji} - \Delta {E^R}_{ji})} \right)^{-1} \right\rangle_{R_{new}} + {E^R}_i(old) \tag{4.17}$$

As the formulae are correlated, they are solved iteratively until both parameters are converged.

At the end of the program, the number of iterations are written out together with the final parameters. For `@form` = 3 and `@update_tree` = 1, the new maximum spanning tree is written to a separate file called `tree.dat`. When calculating averages and distributions special care is taken in order to avoid overflow (see[7]).

Required input arguments	
`@temp`	⟨temperature of the system⟩
`@numstat`	⟨number of end states $\mathcal{N}^{(s)}$⟩
`@form`	⟨functional form of the Hamiltonian⟩
`@vr`	⟨energy time series of the reference state R⟩
`@vy`	⟨energy time series of the end states⟩
`@s`	⟨ list of old s parameters⟩
`@EiR`	⟨ list of old energy offset parameters (${E^R}_i$)⟩

Optional input arguments	
`@update_tree`	⟨switch for max. spanning tree update (required if form=3)⟩
`@tree`	⟨file with old max. spanning tree (required if update_tree is specified)⟩

Standard output

new s and ${E^R}_i$ parameters

Additional output

if `@form`=3, the maximum spanning tree is written to a file with name `tree.dat`

4.19. `eds_update_2` (GROMOS++ program)

Program description:

Calculates the EDS parameters E^R and s from energy time series of two endstates (`@vy`) and the reference state R (`@vr`). Two update schemes are implemented: Scheme 1 calculates the new parameters according to the procedure described in Ref.[14] In that case the parameter `@eunder` corresponds to the energy threshold. Scheme 2 calculates new parameters according to the procedure described in Ref.[15] In that case the parameter `@eunder` corresponds to the energy separating sampling from state A from sampling of state B while the parameters `@etrans` specifies the width of the transition region.

Required input arguments	
`@temp`	⟨temperature of the system⟩
`@vr`	⟨energy time series of the reference state R⟩
`@vy`	⟨energy time series of the end states⟩
`@s`	⟨current s parameter⟩
`@s_old`	⟨old s parameter⟩
`@EiR`	⟨old energy offset parameters ($E^R{}_i$)⟩
`@update`	⟨choice of update scheme⟩
`@eunder`	⟨energy threshold if update=1; separation energy if update=2⟩

Optional input arguments	
`@etrans`	⟨ignored if update=1; size of transition region if update=2 (required)⟩
`@scale`	⟨scaling factor to modify default factors⟩

Standard output

new s and $E^R{}_i$ parameters

Additional output

none

4.20. `edyn` (GROMOS++ program)

Program description:

Program `edyn` performs an essential dynamics analysis over a trajectory. The covariance matrix is calculated for the specified atoms and diagonalised. The eigenvalues and eigenvectors are written to file, as well as information about selected eigenvalues.

The trajectory is subsequently analysed as projections along the eigenvalues. For all of the selected eigenvalues, the atomic components of the eigenvalues and the time series of the projection along the eigenvalue are written to file. In addition, pdb files are written with coordinates of the specified atoms at the extreme values of the projection along the eigenvalue. With the `@skip` flag, the usually time-consuming projections can be skipped and, in this case, only the covariance matrix, the eigenvalues and the eigenvectors will be printed to file.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1): atoms to be considered⟩
`@ref`	⟨reference coordinates⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@eigenvalues`	⟨list of eigenvalues for which data is written⟩
`@skip`	(skip the (time-consuming) projections)

Standard output

none

Additional output

Most of the output of this program is written to a selection of files:

`AVE.pdb`	contains the average position of the specified atoms
`EIVAL.out`	contains the eigenvalues of the covariance matrix
`EIVEC.out`	contains the eigenvectors of the covariance matrix
`EIFLUC.out`	contains the fluctuation along the eigenvectors
`ESSDYN.out`	contains the averages, fluctuations, minimum and maximum values of the projections along the eigenvectors

In addition, several files are written out for each selected eigenvalue, x:

`EVCOMP_x.out`	contains the atomic contributions to the eigenvector
`EVPRJ_x.out`	contains the time series of the projection of the trajectory along the eigenvector
`PRJMAX_x.pdb`	contains coordinates of the selected atoms, displaced from the average positions along the eigenvector to the maximum value of the observed projection
`PRJMIN_x.pdb`	contains coordinates of the selected atoms, displaced from the average positions along the eigenvector to the minimum value of the observed projection

4.21. `ene_ana` (GROMOS++ program)

Program description:

GROMOS can write energies, free-energy derivatives and block averages of them to separate trajectory files for later analysis. Program `ene_ana` extracts individual values from such files and can perform simple mathematical operations on them. The format for (free) energy trajectory files as written by MD++ is known to the program. In addition, the user can define custom made formats of any trajectory file that comes in a block-format through a library file. `ene_ana` is able to read and interpret series of two types of such files simultaneously, typically referred to as the "energy file" and the "free energy file".

Using the same library file one can define properties to be calculated from the values that are listed in them. For the selected properties, `ene_ana` will calculate the time series, averages, root-mean-square fluctuations and a statistical error estimate. The error estimate is calculated from block averages of different sizes. The time for the time series is taken from the trajectory files, unless a different time interval between blocks is specified through an input parameter. If a topology is supplied, the `ene_ana` uses this to define the total solute mass (`MASS`) and the total number of solute molecules (`NUMMOL`).

Required input arguments	
`@en_files`	⟨energy files⟩
`@fr_files`	⟨free energy files⟩
`@prop`	⟨properties to monitor⟩

Optional input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩ (for `MASS` and `NUMMOL`)
`@time`	⟨t and dt⟩ (overwrites `TIME` in the trajectory files)
`@library`	⟨library for property names⟩ [print]

Standard output

averages, root-mean-square fluctuations and error estimates for the requested properties over the supplied trajectories

Additional output

time series of every property will be written to a separate file with name ⟨property⟩.dat.

4.22. ener (GROMOS++ program)

Program description:

Program `ener` can recalculate interaction energies over molecular trajectory files using the interaction parameters specified in the molecular topology file.

Non-bonded interactions are calculated for all selected atoms with all other atoms in the system. Some atoms can be specified as being soft, indicating that interactions involving any of these atoms have a specified softness parameter, for all other atoms in the system, the softness parameter $\alpha = 0$. Van der Waals interactions between particles i and j are calculated as

$$\mathcal{V}^{(vdw)}{}_{ij} = \left[\frac{C_{12}(i,j)}{(r_{ij}^6 + \alpha_{LJ}\lambda^2 C_{126})} - C_6(i,j)\right] \frac{1}{(r_{ij}^6 + \alpha_{LJ}\lambda^2 C_{126})} \tag{4.18}$$

with $C_{126} = C_{12}/C_6$ for C_{12} and C_6 unequal 0, $C_{126} = 0$ otherwise. C_{12} and C_6 are the interaction parameters taken from the topology, λ and α_{LJ} are specified by the user. Similarly, the electrostatic interaction, including reaction field contribution for a homogeneous medium outside the cutoff sphere is calculated as

$$\mathcal{V}^{(ele)}{}_{ij} = \frac{q_i q_j}{4\pi\epsilon_0}\left[\frac{1}{(r_{ij}^2 + \alpha_C\lambda^2)^{1/2}} - \frac{\frac{1}{2}C_{RF} r_{ij}^2}{(R_{RF}{}^2 + \alpha_C\lambda^2)^{3/2}} - \frac{(1 - \frac{1}{2}C_{RF})}{R_{RF}}\right] \tag{4.19}$$

where ϵ_0 is the dielectric permittivity of vacuum and q_i and q_j are the atomic partial charges. R_{RF} is the reaction field cutoff distance, here assumed to be the same as the interaction cutoff. α_C and λ are again user specified. C_{RF} is calculated from the reaction field dielectric constant ϵ_{RF} and κ_{RF} (user specified) as

$$C_{RF} = \frac{(2 - 2\epsilon_{RF})(1 + \kappa_{RF}R_{RF}) - \epsilon_{RF}(\kappa_{RF}R_{RF})^2}{(1 + 2\epsilon_{RF})(1 + \kappa_{RF}R_{RF}) + \epsilon_{RF}(\kappa_{RF}R_{RF})^2} \tag{4.20}$$

The bonded interactions are calculated for all specified properties using the following interaction functions. For bonds we use the quartic bond stretching interaction form $V^{(b)} = V^{(b,q)}$:

$$V^{(b,q)}(b_n; k_n^{(b,q)}, b_n^0) = {}^1\!/\!_4 k_n^{(b,q)}(b_n{}^2 - b_n^{0\,2})^2 \tag{4.21}$$

with b_n the actual bond length, $k_n^{(b,q)}$ and b_n^0 the force constant and optimal bond length, respectively. For angles we use the cosine-harmonic bond-angle bending interaction form $V^{(\theta)} = V^{(\theta,c)}$:

$$V^{(\theta,c)}(\theta_n; k_n^{(\theta,c)}, \theta_n^0) = {}^1\!/\!_2 k_n^{(\theta,c)}(\cos(\theta_n) - \cos(\theta_n^0))^2 \tag{4.22}$$

with θ_n the actual bond angle, $k_n^{(\theta,c)}$ and θ_n^0 the force constant and optimal bond angle respectively. For proper torsional dihedral angle terms we use:

$$V^{(\varphi)}(\varphi_n; k_n^{(\varphi)}, \varphi_n^0, m_n^{(\varphi)}) = k_n^{(\varphi)}(1 + \cos(\varphi_n^0)\cos(m_n^{(\varphi)}\varphi_n)) \quad \text{with} \quad \varphi_n^0 = 0, \pi\ . \tag{4.23}$$

with φ_n the actual dihedral angle value, $k_n^{(\varphi)}$ the force constant and φ_n^0 and $m_n^{(\varphi)}$ the phase shift and multiplicity, respectively. Improper dihedral energy contributions are calculated from the function $V^{(\xi)}$:

$$V^{(\xi)}(\xi_n; k_n^{(\xi)}, \xi_n^0) = {}^1\!/\!_2 k_n^{(\xi)}(\xi_n - \xi_n^0)^2 \tag{4.24}$$

ξ_n^0 are the force constant and optimal improper dihedral angle value.

The program can print out various energies in separate columns, e.g. bonded energies, non-bonded energies with the solute only, with the solvent only or the total energies.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1) : atoms for non-bonded interaction⟩
`@energies`	⟨energy specifier for the interaction energies to be calculated (1: covalent; 2: elec with solute; 3: elec with solvent; 4: elec total; 5: vdW with solute; 6: vdW with solvent; 7: vdW total; 8: nonbonded total; 9: total)⟩
`@props`	⟨property specifier (see Sec. 1.3.3): bonded properties to be calculated⟩
`@cut`	⟨cut-off distance⟩
`@eps`	⟨epsilon for reaction field contribution⟩
`@kap`	⟨kappa for reaction field contribution⟩
`@RFex`	⟨switch the self term for excluded atoms in the reaction field polarization on or off ⟩
`@soft`	⟨atom specifier (see Sec. 1.3.1) for soft atoms⟩
`@softpar`	⟨lam⟩ ⟨a_lj⟩ ⟨a_c⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@time`	⟨time⟩ ⟨dt⟩

Standard output

time series of calculated energies (averages).

Additional output

none

4.23. `epath` (GROMOS++ program)

Program description:

Program `epath` finds electron-tunneling pathways in proteins. It uses Dijkstra's graph search algorithm[16] to find the pathway with the highest product of the decay factors, corresponding to the "shortest path". The decay factor ϵ_{ij} for the electron transfer between atoms i and j is calculated according to:

$$\epsilon_{ij} = Ae^{B(r_{ij}^2 - R)} \tag{4.25}$$

where r_{ij} is the distance between the atoms and the different parameters A, B and R are specified for jumps through covalent bonds, hydrogen bonds and space as described by Beratan.[17]

For every atom of the system, the neighbouring atoms within a user-specified cutoff are determined. For every neighbouring atom its connectivity is classified as covalent, H-bonded or through space. The decay factor for the neighbouring atoms is calculated using the appropriate parameters and stored if it is higher than the decay factor that was already stored from a previous cycle. Additionally the jump type and the atom from where this jump occured are stored. When all atoms have been visited, the "shortest path" is backtraced from the acceptor to the donor.

The parameters A, B and R are configurable via a parameter file, as well as the parameters for the Hbond detection. All filenames are configurable.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@donor`	⟨atom specifier (see Sec. 1.3.1): electron donor⟩
`@acceptor`	⟨atom specifier (see Sec. 1.3.1): electron acceptor⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@cutoff`	⟨cut-off distance (default 0.6)⟩
`@param`	⟨parameter file⟩
`@outfile`	⟨name of the output file⟩
`@details`	(detailed output)
`@detailsfile`	⟨name of the output file of the details⟩
`@timeseries`	(print time series)
`@timeseriesfile`	⟨filename of the time series file⟩
`@verbose`	(produce a verbose output to stderr giving runtime details)

Standard output

A pdb file containing the coordinates of all atoms that have been part of a path throughout the different frames and how often they have been part of a path as percentage in the B-factor column

Additional output

Detailed output consiting off a summary per frame and a pdb file containing the coordinates of the system as well as the coordinates of the path corresponding to that frame linked together in order to make the path visualisable. Also the program can output to a file a timeseries of the product of the decay factor of the different frames as well as its log, and the avarages of both at the end.

4.24. `eps_field` (GROMOS++ program)

Program description:

Program `eps_field` estimates the relative static dielectric permittivity, $\epsilon(0)$, of a liquid when an external electric field was applied during the simulation. The permittivity for a specific external field is given by

$$\epsilon(0) = 1 + 4\pi \frac{< \mathbf{P} >_t}{\mathbf{E}^{ext}} \tag{4.26}$$

where $\mathbf{E}^{ext}$ is the external electric field, ϵ_0 is the dielectric permittivity of vacuum, and $\mathbf{P}$ is the polarisation of the system defined as

$$\mathbf{P}(t) = \mathcal{V}(t)^{-1}\boldsymbol{M}(t) \tag{4.27}$$

where $\boldsymbol{M}$ is the total dipole moment of the system and $\mathcal{V}$ is the volume.
Note, to get a linear response of the polarisation, the electric field should be small enough to avoid saturation, which is the case if

$$\frac{< \boldsymbol{\mu}_i \mathbf{E}^{ext} >}{3k_B} << T \tag{4.28}$$

with $\boldsymbol{\mu}_i$ the dipole moment of molecule i, k_B the Boltzmann constant and T the temperature, is fulfilled.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@E_ex`	⟨external electric field strength⟩
`@trs`	⟨special trajectory files (with box dipole moment)⟩

Optional input arguments

`@time`	⟨time and dt⟩

Standard output

time series of polarisation in z-direction. Average of polarisation in x-, y-, and z-direction, and $\epsilon(0)$

Additional output

none

4.25. `epsilon` (GROMOS++ program)

Program description:

Program `epsilon` calculates the dielectric properties of the system. For systems containing only neutral molecules, it estimates the relative dielectric permittivity, $\epsilon(0)$, of a simulation box from a Kirkwood-Fröhlich type of equation, as derived by Neumann,[4]

$$(\epsilon(0)-1)\frac{2\epsilon_{rf}+1}{2\epsilon_{rf}+\epsilon(0)} = \frac{\langle \boldsymbol{M}^2\rangle - \langle \boldsymbol{M}\rangle^2}{3\epsilon_0 \mathcal{V} k_B T} \quad , \tag{4.29}$$

where $\boldsymbol{M}$ is the total dipole moment of the system, ϵ_0 the dielectric permittivity of vacuum, ϵ_{rf} is a reaction-field epsilon value, $\mathcal{V}$ is the volume and k_BT is the absolute temperature multiplied by the Boltzmann constant.

For systems containing ionic species, the total dipole moment is split into a rotational part, $\boldsymbol{M_d}$, and a translational part, $\boldsymbol{M_j}$:

$$\boldsymbol{M_d} = \sum_{m=1}^{N_m}\sum_{a=1}^{N_{m,a}} q_{m,a}(\mathbf{r}_{m,a} - \mathbf{r}_{cm,m}) \tag{4.30}$$

$$\boldsymbol{M_j} = \sum_{m=1}^{N_m} q_m \mathbf{r}_{cm,m} \tag{4.31}$$

where N_m is the total number of molecules, $N_{m,a}$ is the number of atoms of the molecule m and cm stands for the center of mass.

The generalized frequency-dependent dielectric constant can be decomposed into the following contributions:

$$\langle \boldsymbol{M_d}^2\rangle \tag{4.32}$$

and the autocorrelation functions

$$\langle \boldsymbol{M_d}(\tau)\boldsymbol{M_d}(\tau+t)\rangle \tag{4.33}$$

$$\langle \boldsymbol{J}(\tau)\boldsymbol{J}(\tau+t)\rangle \tag{4.34}$$

as well as the cross term

$$\langle \boldsymbol{M_d}(\tau)\boldsymbol{J}(\tau+t)\rangle \tag{4.35}$$

where $\boldsymbol{J} = \frac{d\boldsymbol{M_j}}{dt} = \sum_{m=1}^{N_m} q_m \mathbf{v}_{cm,m}$.

Using fit functions one can calculate both the frequency-dependent dielectric response and the static dielectric constant of the system. For more details see Ref.[18].

Practically, to calculate the static dielectric constant the contribution of the cross term can be neglected, while the contribution from Eq. 4.34 can be calculated from $\langle \Delta\boldsymbol{M_j}^2\rangle$ (the mean square displacement of $\boldsymbol{M_j}$) using the Einstein relation. For more details see Ref.[19].

Note that $\langle \Delta\boldsymbol{M_j}^2\rangle$ has to be calculated from an unfolded trajectory. For that reason, the first frame of the trajectory is gathered using the gbond (to avoid broken molecules) and the rest with the gtime method.

The program outputs the relative permittivity (epsilon) calculated exclusively from the $\langle \boldsymbol{M_d}^2\rangle$ contribution. For systems containing ionic species, $\langle \Delta\boldsymbol{M_j}^2\rangle$ is calculated and written in file `Mj2.out`, from which the translational contribution to the relative permittivity can be calculated. See the `fit_Mj2.py` script in the `gromos++/examples/` directory.

Optionally, the program can calculate the $\langle \boldsymbol{M_d}(\tau)\boldsymbol{M_d}(\tau+t)\rangle$ and $\langle \boldsymbol{J}(\tau)\boldsymbol{J}(\tau+t)\rangle$ autocorrelation functions as well as the $\langle \boldsymbol{M_d}(\tau)\boldsymbol{J}(\tau+t)\rangle$ cross term (files `MdMd.out`, `JJ.out` and `MdJ.out`). Note that velocity trajectory files have to be provided to calculate the contributions due to Eq. 4.34 and Eq. 4.35. The translational contribution to the relative permittivity can be calculated from file `JJ.out`. See the `fit_JJ.py` script in the `gromos++/examples/` directory.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@temp`	⟨temperature⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@e_rf`	⟨reaction field epsilon⟩
`@time`	⟨time and dt⟩
`@traj_vel`	⟨velocity trajectory files⟩
`@omega`	⟨enable omega-dependent calculation⟩
`@MdJ`	⟨enable cross-term MdJ calculation (usually neglected)⟩

Standard output

time series of the current estimate of $\epsilon(0)$ calculated exclusively from the $\langle \boldsymbol{M_d}^2 \rangle$ contribution

Additional output

output file `Mj2.out` contains the time series of the mean square displacement of $\boldsymbol{M_j}$
output file `MdMd.out` contains the autocorrelation function of $\boldsymbol{M_d}$
output file `JJ.out` contains the autocorrelation function of $\boldsymbol{J}$
output file `MdJ.out` contains the cross correlation function of $\boldsymbol{M_d}$ and $\boldsymbol{J}$

4.26. espmap (GROMOS++ program)

Program description:

Program espmap calculates the vacuum electrostatic potential around a user specified group of atoms. It uses the atomic partial charges as defined in the topology and calculates the potential on a grid. The results are written to a .pl file that can be converted to a .plt file which can be read in by Gopenmol.

Required input arguments

@topo	⟨molecular topology file (see Sec. 1.2)⟩
@pbc	⟨periodic boundary and gathering (Sec. 1.2)⟩
@atoms	⟨atom specifier (see Sec. 1.3.1): atoms to consider⟩
@grspace	⟨grid spacing (default: 0.2 nm)⟩
@traj	⟨trajectory files⟩

Optional input arguments

none

Standard output

a .pl and .plt file to be read in by Gopenmol

Additional output

none

4.27. `ext_ti_ana` (GROMOS++ program)

Program description:

From a simulation at a single coupling parameter λ program `ext_ti_ana` predicts free energy derivatives $\frac{\partial \mathcal{H}}{\partial \lambda}$ over a range of λ values. To do this it requires that the terms described in ref[20] have been precalculated during the simulation and written to the energy trajectories (using gromos md++ block PRECALCLAM).

The program reconstructs the free energy derivatives at the requested λ_p values and performs a reweighing to obtain the ensemble averages at λ_p, from the simulations at the simulated λ_s, using

$$\left\langle \frac{\partial \mathcal{H}}{\partial \lambda} \right\rangle_p = \frac{\left\langle \left.\frac{\partial \mathcal{H}}{\partial \lambda}\right|_p e^{-\beta[\mathcal{H}(\lambda_p)-\mathcal{H}(\lambda_s)]} \right\rangle_s}{\left\langle e^{-\beta[\mathcal{H}(\lambda_p)-\mathcal{H}(\lambda_s)]} \right\rangle_s} \tag{4.36}$$

The predictions from multiple simulations at λ_s can be merged into a single TI profile using program `ext_ti_merge` (see Sec. 4.28).

In GROMOS the coupling parameter of different interaction types x, Λ_x, can be set individually from a fourth order polynomial of the global coupling parameter λ (`md++` block LAMBDAS, see Sec. 2-14.4):

$$\Lambda_x = a_x\lambda^4 + b_x\lambda^3 + c_x\lambda^2 + d_x\lambda + e_x. \tag{4.37}$$

`ext_ti_ana` can make predictions for other combinations of the coefficients (a_x,b_x,c_x,d_x,and e_x) than the ones used in the simulation. The interaction properties (x) that can be given individual λ dependencies are:
- slj: lennard jones softness
- scrf: coulomb reaction-field softness
- lj: lennard jones
- crf: coulomb reaction-field
- bond: bond
- ang: angle
- impr: improper dihedral angle
- dih: dihedral angle
- kin: kinetic (not implemented in the LAMBDAS block of `md++`)

To facilitate the evaluation of many sets of coefficients for slj and scrf, these can be given in an input file of the following format:

```
TITLE
..
END
SLJ
# uniquelabel a b c d e
1 -0.4 0.4 -0.3 1.3 0.0
2 -0.4 0.4 -0.2 1.2 0.0
3 -0.4 0.4 -0.1 1.1 0.0
END
SCRF
# uniquelabel a b c d e
1 0.0 0.0 0.7 0.3 0.0
2 0.0 0.0 -0.1 0.3 0.0
END
```

Predictions will be made for any combination of every given slj with every given scrf.

The coefficients used in the simulation are specified by the `@lamX_sim` flags or read from the LAMBDAS block in a gromos input parameter file specified by `@imd`. The coefficients we want to predict for are specified by the `@lamX` flags. Also the temperature of the simulation, the simulated λ (`@slam`) and its exponent (`@NLAMs`) and the parameters of the PRECALCLAM block (`@nrlambdas`, `@minlam`, `@maxlam`) can be read from this parameter file. If parameters are specified explicitly as input flags they will always overwrite the corresponding values read from the imd file.

If the flag `@countframes` is set, the number of contributing frames for each predicted λ is appended as an additional column to the output. This number is evaluated as the number of snapshots for which the difference in the predicted energy and the simulated energy is less than the free energy difference between the two states, as calculated using the perturbation formula.

Error estimates can be calculated using bootstrapping. A random set of data points of the size of the original set will be chosen and the predictions made. This is repeated for as many bootstrap replicates as requested. The standard deviation over the bootstrap replicates is reported as a bootstrap error.

The predicted TI curves from several simulations at different λ values can be combined using program `ext_ti_merge` (see Sec. 4.28).

Finally, program `ext_ti_ana` can write out time series of energies at alternative value of λ to be used for free-energy estimates with Bennett's acceptance ratio (BAR). If option `@bar_data` is used without further input parameters, this data is written out for all predicted λ values, or if further input parameters are given it is only written for the selected values of λ. Program `bar` (see Sec. 4.1) can be used to estimate free-energy differences from these files.

Required input arguments

`@en_files`	⟨energy trajectory files⟩
`@fr_files`	⟨free-energy trajectory files⟩
`@library`	⟨library for block information⟩
`@temp`	⟨simulation temperature⟩

Optional input arguments

`@nrlambdas`	⟨number of precalculated lambdas (can also be read from `@imd`)⟩
`@minlam`	⟨minimum precalculated lambda (can also be read from `@imd`)⟩
`@maxlam`	⟨maximum precalculated lambda (can also be read from `@imd`)⟩
`@slam`	⟨lambda value of simulation (can also be read grom `@imd`)⟩
`@NLAMs`	⟨lambda exponent of simulation (can also be read grom `@imd`)⟩
`@lam<X>_sim`	⟨a⟩ ⟨b⟩ ⟨c⟩ ⟨d⟩ ⟨e⟩ coefficients for individual lambda dependence (default: 0 0 0 1 0) where ⟨X⟩ is one of: slj, scrf, lj, crf, bond, ang, impr, dih, kin
`@imd`	⟨gromos input parameter file⟩
`@NLAMp`	⟨lambda exponent value to predict for, default: `@NLAMs`⟩
`@lam<X>`	⟨a⟩ ⟨b⟩ ⟨c⟩ ⟨d⟩ ⟨e⟩ coefficients for individual lambda dependence (default: `@lam<X>_sim`) where ⟨X⟩ is one of: slj, scrf, lj, crf, bond, ang, impr, dih, kin
`@slj_scrf_file`	⟨file with sets of slj and scrf lambda coefficients⟩
`@no_<X>`	exclude ⟨X⟩; free energy derivative contribution, where ⟨X⟩ is one of: slj, scrf, lj, crf, bond, ang, impr, dih, kin
`@countframes`	⟨count nr of contributing frames for each plam⟩
`@pmin`	⟨min index of prediction⟩
`@pmax`	⟨max index of prediction⟩
`@bootstrap`	⟨number of bootstrap cycles⟩
`@outdir`	⟨directory to write output to⟩
`@lam_precision`	⟨lambda value precision in outfiles (default: 2)⟩
`@bar_data`	⟨print energies to be used for BAR (not reweighted)⟩
`@verbose`	⟨print used parameters to file header⟩
`@cpus`	⟨number of omp threads (default: 1)⟩

Standard output

details of the calculation

Additional output

- files with predicted free-energy derivatives using the specified parameters these can be merged with program `ext_ti_merge` (see Sec. 4.28

- files with data to be used by program `bar` (see Sec. 4.1)

4.28. `ext_ti_merge` (GROMOS++ program)

Program description:

Program `ext_ti_merge` combines TI curves predicted by program `ext_ti_ana` (Sec. 4.27) from several simulations at different λ_s points by a linear weighting scheme. Two weights are defined for any value of λ_P, which lies between two simulated points λ_{S1} and λ_{S2}:

$$w_{s1} = \frac{\lambda_p - \lambda_{s2}}{\lambda_{s1} - \lambda_{s2}} \qquad w_{s2} = \frac{\lambda_p - \lambda_{s1}}{\lambda_{s2} - \lambda_{s1}}. \tag{4.38}$$

The appropriate ensembler average of the free energy derivatives $\frac{\partial \mathcal{H}}{\partial \lambda}$ are then computed as,

$$\left\langle \frac{\partial \mathcal{H}(\lambda_p)}{\partial \lambda} \right\rangle_{\lambda_p} = w_{s1} \left\langle \frac{\partial \mathcal{H}(\lambda_p)}{\partial \lambda} \right\rangle_{\lambda_{s1}} + w_{s2} \left\langle \frac{\partial \mathcal{H}(\lambda_p)}{\partial \lambda} \right\rangle_{\lambda_{s2}} \tag{4.39}$$

The integral (using the trapezoidal rule) of the final TI curve (and of its error values) is appended to the output file.

Required input arguments	
`@files`	⟨data files⟩ generated by `ext_ti_ana`, see Sec. 4.27

Optional input arguments	
`@slam`	⟨lambda values of the simulation⟩ optional if found in the header of the data files after #SLAM
`@noerrors`	⟨do not read and use the error column which might be in the files⟩

Standard output

single merged free energy derivative profile

Additional output

none

4.29. `filter` (GROMOS++ program)

Program description:

Program `filter` reduces coordinate trajectory files and writes out a trajectory file (in GROMOS or pdb format) in which for every frame, the coordinates are only kept for atoms that are within a specific distance of a specified part of the system. To determine if interatomic distances are within the specified cut-off, either an atomic or a charge-group based cut-off scheme can be employed. Additionally, parts of the system can be specified for which in all cases the atomic coordinates should either be kept or rejected.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨input trajectory files⟩

Optional input arguments

`@atoms`	⟨atom specifier (see Sec. 1.3.1): atoms to consider as reference part of the system⟩
`@cutoff`	⟨cut-off distance (nm, default: 0.0)⟩
`@pairlist`	⟨cut-off scheme (ATOMIC (default) or CHARGEGROUP)⟩
`@select`	⟨atom specifier (see Sec. 1.3.1): atoms to keep⟩
`@reject`	⟨atom specifier (see Sec. 1.3.1): atoms not to keep⟩
`@time`	⟨time and dt⟩ (overwrites `TIME` in the trajectory files)
`@outformat`	⟨output coordinates format, see Sec. 1.2⟩

Standard output

filtered trajectory file

Additional output

none

4.30. `follow` (GROMOS++ program)

Program description:

Program `follow` can create a 3D trace of selected atoms through time. The program always takes the nearest image with respect to the previous position of the particle.

Required input arguments

`@topo`	⟨topology⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@dim`	⟨dimensions to consider⟩
`@atoms`	⟨atoms to follow⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@time`	⟨time and dt ⟩

Standard output

none

Additional output

For every atom that is selected, a pdb file is written out (`FOLLOW_x.pdb`) in which the trajectory is indicated in the `CONECT` entries.

4.31. `gathtraj` (GROMOS++ program)

Program description:

Program `gathtraj` applies the periodic boundary conditions to a coordinate trajectory and writes the gathered trajectory.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨input trajectory files⟩

Optional input arguments

none

Standard output

a single trajectory file

Additional output

none

4.32. `hbond` (GROMOS++ program)

Program description:

Program `hbond` monitors the occurrence of hydrogen bonds over a molecular trajectory file. It can monitor conventional hydrogen bonds, as well as three-centered hydrogen bonds through geometric criteria.

A hydrogen bond is considered to be present if the distance between a hydrogen atom, H, connected to a donor atom D, is within a user specified distance (typically 0.25 nm) from an acceptor atom A and the D-H-A angle is larger than another user specified value (typically 135°). Occurrences of three centered hydrogen bonds are defined for a donor atom D, hydrogen atom H and two acceptor atoms A1 and A2 if:

(i) the distances H-A1 and H-A2 are within a user specified value (typically 0.27 nm)

(ii) the angles D-H-A1 and D-H-A2 are larger than a second user specified value (typically 90°)

(iii) the sum of the angles D-H-A1, D-H-A2 and A1-H-A2 is larger than a third user specified value (typically 340°)

(iv) the dihedral angle defined by the planes through the atoms D-A1-A2 and H-A1-A2 is smaller than a fourth user specified value (typically 15°).

The user can specify two groups of atoms (A and B) between which the hydrogen bonds are to be monitored. If hydrogen bond donors and acceptors are not explicitly specified, these can be filtered based on their masses, as can be specified in a so-called "massfile". If a reference structure is given, only hydrogen bonds that are observed in the reference structure will be monitored.

The program calculates average angles, distances and occurrences for all observed hydrogen bonds over the trajectories and prints out a time series of the observed hydrogen bonds.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@DonorAtomsA`	⟨atom specifier (see Sec. 1.3.1)⟩
`@AcceptorAtomsA`	⟨atom specifier (see Sec. 1.3.1)⟩
`@DonorAtomsB`	⟨atom specifier (see Sec. 1.3.1)⟩
`@AcceptorAtomsB`	⟨atom specifier (see Sec. 1.3.1)⟩
`@Hbparas`	⟨distance [nm] and angle [degrees]; default: 0.25, 135⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@threecenter`	⟨distances [nm]⟩ ⟨angles [degrees]⟩ ⟨sum⟩ ⟨dihedral⟩
`@ref`	⟨reference coordinates for native H-bonds⟩
`@massfile`	⟨massfile⟩
`@time`	⟨time and dt⟩

Standard output

Statistics on all monitored hydrogen bonds, consisting of average distances and angles, number of occurrences and percentage of occurrence over the trajectory.

Additional output

Time series of the number of hydrogen bonds and the number of three-centered hydrogen bonds are written to files `Hbnumts.out` and `Hb3cnumts.out`, respectively. Time series for every observed hydrogen bond and three-centered hydrogen bonds are written to files `Hbts.out` and `Hb3cts.out`, respectively.

4.33. `int_ener` (GROMOS++ program)

Program description:

Program `int_ener` recalculates the nonbonded interaction energy between two non-overlapping sets of solute atoms using the interaction parameters specified in the molecular topology file. It can also compute the interaction energy between a specified group of solute atoms and the solvent. If a time series is requested, the total nonbonded interaction is printed at each time point, along with the van der Waals and electrostatic contributions.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@atomsA`	⟨atom specifier (see Sec. 1.3.1) for the first group of atoms⟩
`@traj`	⟨position trajectory file(s)⟩

Optional input arguments	
`@atomsB`	⟨atom specifier (see Sec. 1.3.1) for the second group of atoms⟩
`@solvent`	⟨compute energy between `atomsA` and solvent⟩
`@time`	⟨time and dt⟩
`@timeseries`	⟨print time series⟩
`@timespec`	⟨time points at which to compute the energy: ALL (default), EVERY or SPEC (if time series)⟩
`@timepts`	⟨time points at which to compute the energy (if timesseries and timespec EVERY or SPEC)⟩
`@cut`	⟨cut-off distance (default: 1.4)⟩
`@eps`	⟨epsilon for reaction field contribution (default: 1.0)⟩
`@kap`	⟨kappa for reaction field contribution (default: 0.0)⟩

Standard output

average total nonbonded interaction energy and the van der Waals and electrostatic contributions, preceded by time series if requested

Additional output

none

4.34. `iondens` (GROMOS++ program)

Program description:

Program `iondens` calculates the average density of ions (or other particles) over a trajectory file. A rotational fit of the system onto the solute can be performed, to correct for rotations of the complete simulation box. The density will be calculated on a grid of points. Two sets of densities can be written out, containing 1) occupancies on the grid points, relative to the maximally occupied gridpoint, or 2) occupancies as a percentage of the number of frames. User specified cutoffs determine which gridpoints will be written out.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@grspace`	⟨grid spacing (default: 0.2 nm)⟩
`@ions`	⟨atom specifier (see Sec. 1.3.1): ions to monitor⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1): atoms to use for fit⟩
`@ref`	⟨reference coordinates⟩
`@thresholds`	⟨threshold values for occupancy percentages (default: 20 and 5)⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

none

Standard output

none

Additional output

Four files will be written out, which are all in the same coordinate frame:

1) `ref.pdb`	the reference structure used for fitting
2) `aver.pdb`	the average structure over the trajectory
3) `grid.pdb`	the ion density relative to the most occupied grid point
4) `gridnf.pdb`	the ion density as percentage of the total number of frames

4.35. `jepot` (GROMOS++ program)

Program description:

Program `jepot` computes the ^3J-value local elevation (LE) potential energy term from a LE ^3J-value restrained simulation. The LE potential can be calculated for all values ($0 - 360°$) of all restrained angles at the end of the simulation only (`@fin`) or for selected angles (`@angles`) as a time series throughout the simulation (requires `@topo`, `@pbc`, `@postraj` and `@restraj`). The `@timespec`, `@timepts` and `@restraj` arguments control the time series. The time series can be of the LE potential for all values of the selected angle (`ALL`; default) or for only the current value of the selected angle (`CURR`) at each point in time, giving only the current contribution of the LE potential to the overall potential energy of the selected angle. With `CURR`, the `@jval` file must contain the ^3J-value specifications for the selected angle only.

`@K` is the force constant given in the MD input file. Note that this is multiplied by `WJVR`, the weight factor in the `@jval` file, during the calculation of the LE potential energy to give $k^{(Jr)}$.

Required input arguments

`@jval`	⟨jvalue restraint specifications⟩
`@K`	⟨force constant⟩
`@ngrid`	⟨number of grid points⟩

Optional input arguments

`@angles`	⟨angle⟩ values over which to compute the LE potential energy: `ALL` (default) or `CURR`
`@fin`	⟨file containing final coordinates (if not time series)⟩
`@time`	⟨time dt (optional and only if time series)⟩
`@timespec`	⟨time points at which to compute the LE potential energy: `ALL` (default), `EVERY` or `SPEC` (if time series)⟩
`@timepts`	⟨time points at which to compute the LE potential (if time series and @timespec is `EVERY` or `SPEC`)⟩
`@topo`	⟨molecular topology file (see Sec. 1.2) (if `CURR`)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2), if `CURR`⟩
`@postraj`	⟨position trajectory files (if `CURR`)⟩
`@restraj`	⟨restraint trajectory files (if time series)⟩

Standard output

With `@angles ALL` and `@fin`: ^3J-value LE potential energy over 360° for each of the ^3J-value restraints specified in `@jval` and `@fin` at the end of the simulation.
With `@angles CURR`, `@topo`, `@pbc`, `@postraj` and `@restraj`: the current value of the restrained angle and the ^3J-value LE potential energy for this angle value. A time series will be written unless only one frame is selected (using `@timespec`, `@timepts`). The time values printed can be manipulated using `@time` and the time points for which the LE potential energy is calculated and printed can be controlled using `@time`, `@timespec` and `@timepts`.

Additional output

none

4.36. `jval` (GROMOS++ program)

Program description:

Program `jval` computes the ^3J-values from a single conformation or from a trajectory. It can write out the values of all ^3J-couplings specified in the file specified by `@jval` or the total RMSD over all couplings from the reference values at each point in time. The final part of the output is always a summary of the ^3J-value specification parameters, the averages over the entire trajectory and other statistics. Note that the dihedral angle is computed in a non-periodic manner, possibly going beyond the range [0, 360]. This allows the user to distinguish very dynamic from more restricted dihedral angles through the average and root-mean-square fluctuations of the dihedral angle.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@jval`	⟨^3J-value specification file⟩
`@traj`	⟨position trajectory file(s)⟩

Optional input arguments

`@timeseries`	⟨write time series of ^3J-values⟩
`@rmsd`	⟨write the RMSD over all ^3J-values as a time series⟩
`@time`	⟨time dt (optional and only if time series)⟩
`@timespec`	⟨time points at which to compute the ^3J-values: ALL (default), EVERY or SPEC (if time series)⟩
`@timepts`	⟨time points at which to compute the ^3J-values (if time series and `@timespec` is `EVERY` or `SPEC`)⟩

Standard output

Information required to specify each ^3J-value, averaged ^3J-values and other statistics. If `@timeseries`, the value of each ^3J-coupling is printed at each point in time. If `@rmsd`, the overall RMSD from the reference ^3J-values is printed at each point in time.

Additional output

none

4.37. `m_widom` (GROMOS++ program)

Program description:

Program `m_widom` can calculate the free energy of inserting a test particle into configurations of a molecular system. For every configuration in the given trajectory file, the program places the particle at a user specified number of random positions and evaluates the nonbonded interaction energy, $\mathcal{V}^{(nbd)}$. The free energy is calculated as

$$\Delta G_S = -k_B \mathcal{T} \ln \frac{< \mathcal{V} e^{-\mathcal{V}^{(nbd)}/k_B \mathcal{T}} >}{< \mathcal{V} >} \tag{4.40}$$

with k_B the Boltzmann constant and $\mathcal{T}$ and $\mathcal{V}$ the temperature and volume of the system. The program will also calculate the solute-solvent energies according to

$$\Delta \mathcal{U}_{uv} = \frac{< \mathcal{V}^{(nbd)} \mathcal{V} e^{-\mathcal{V}^{(nbd)}/k_B \mathcal{T}} >}{\mathcal{V} e^{-\mathcal{V}^{(nbd)}/k_B \mathcal{T}}} \tag{4.41}$$

which equals the solute-solvent enthalpy, H_{uv}, as no volume change upon solvation is taking place. The solute-solvent entropy is subsequently calculated from

$$\mathcal{T} \Delta \mathcal{S}_{uv} = \Delta G - \Delta H_{uv} \quad . \tag{4.42}$$

For a more complete description of these free energies, see e.g.[21] In addition to the energetics of the system, the program can also calculate radial distribution functions for all inserted species, with respect to user-specified atoms in the original system. Each group of atoms to include in the rdf calculations is preceded by the keyword `new` in the input string. The radial distribution function is calculated as in the program `rdf` (Sec. 4.48), where all averages are weighted with the Boltzmann probability of every insertion attempt.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@intopo`	⟨topology of the inserted particle⟩
`@inpos`	⟨coordinates of the inserted particle⟩
`@cut`	⟨cut-off distance⟩
`@temp`	⟨temperature⟩
`@ntry`	⟨number of insertion tries per frame⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@time`	⟨time⟩ ⟨dt⟩
`@stride`	⟨take every n-th frame (default: 1)⟩
`@eps`	⟨epsilon for reaction field (default: 1)⟩
`@kap`	⟨kappa for reaction field (default: 0)⟩
`@rdf`	⟨rdf with atom types⟩
`@rdfparam`	⟨rdf-cutoff⟩ ⟨grid⟩

Standard output

time series and summaries of free energies and solute-solvent enthalpies and entropies for all species that are inserted

Additional output

for every species, a file called `rdf_widom_⟨n⟩.out` is written out, where n represents the sequence number of the species in the perturbation topology (the files contain radial distribution functions of the inserted species with user-specified atoms)

4.38. `matrix_overlap` (GROMOS++ program)

Program description:

This program makes use of the GSL library to calculate the overlap between two matrices. Considering the following equation for the difference between matrices $\underline{\mathbf{M}}_1$ and $\underline{\mathbf{M}}_2$

$$\underline{\mathbf{D}} = \sqrt{\mathrm{tr}((\sqrt{\underline{\mathbf{M}}_1} - \sqrt{\underline{\mathbf{M}}_2})^2)} \tag{4.43}$$

where tr is the trace and the square root operator corresponds to the matrix-square root and is calculated according to the following steps. Consider a matrix $\underline{\mathbf{A}}$ that can be diagonalized by

$$\underline{\mathbf{T}} = \underline{\mathbf{V}}^{-1}\underline{\mathbf{A}}\underline{\mathbf{V}} \tag{4.44}$$

The square root of the elements of the diagonal matrix is taken. This procedure corresponds to the application of a normal scalar square root operator to all the elements of the diagonal matrix. Third, the square root of the diagonal matrix is used to calculate the square root of the matrix as

$$\underline{\mathbf{A}}^{1/2} = \underline{\mathbf{V}}\underline{\mathbf{T}}^{1/2}\underline{\mathbf{V}}^{-1} \tag{4.45}$$

The (normalized) overlap $\underline{\mathbf{O}}$ is given by 1 minus the difference $\underline{\mathbf{D}}$ of the matrices divided by the normalization factor as shown below,

$$\underline{\mathbf{O}} = 1 - \frac{\underline{\mathbf{D}}}{\sqrt{tr(\underline{\mathbf{M}}_1) + tr(\underline{\mathbf{M}}_2)}} \quad . \tag{4.46}$$

Required input arguments	
`@m1`	⟨matrix 1⟩
`@m2`	⟨matrix 2⟩
`@dimension`	⟨dimension⟩

Optional input arguments

Standard output

trace, difference and normalised overlap of the two matrices

Additional output

none

4.39. `mdf` (GROMOS++ program)

Program description:

Program `mdf` calculates and lists, for a given set of atoms, the distance to the nearest atom belonging to a second set of atoms. For every selected atom, an output file is written with the minimum distance to, and an atom specifier (see Sec. 1.3.1) for, the nearest atom. This program also works for virtual atoms.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@centre`	⟨atom specifier (see Sec. 1.3.1): central group of atoms⟩
`@with`	⟨atom specifier (see Sec. 1.3.1): group of atoms from which to find the nearest atom⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@time`	⟨time and dt⟩ (overwrites `TIME` in the trajectory files)

Standard output

none

Additional output

for every centre-atom, a file (`MIN_⟨atomspec⟩.dat`) is written out with a time series of the distance to the nearest with-atom and an atom-specification of that atom

4.40. `nhoparam` (GROMOS++ program)

Program description:

Program `nhoparam` calculates order parameters for a given set of nitrogen atoms. In a first step, the program determines the N-H bonds (of which μ is the unit vector) by the atomic masses of nitrogen and hydrogen. For secondary and tertiary amides the different N-H bonds are averaged. Then,

$$S^2 = \frac{1}{2}\left[3\sum_{i=1}^{3}\sum_{j=1}^{3}\langle\mu_i(t)\mu_j(t)\rangle_t^2 - 1\right] \tag{4.47}$$

is applied in order to calculate the order parameter of the N-H bond after performing a least-square rotational fit. Fitting can be controlled using the `@ref` and `@atomsfit` arguments. If `@ref` is absent, the first frame of the trajectory is taken as reference. `@atomsfit` are the atoms used for fitting. If omitted, the nitrogen atoms are used. The fit can be disabled by giving an empty set of atoms.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@winframe`	⟨averarging window (number of frames)⟩
`@atoms`	⟨nitrogen atoms for order parameter calculation⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@time`	⟨time and dt⟩
`@atomsfit`	⟨atoms to consider for fit⟩
`@ref`	⟨reference coordinate (if absent, the first frame of @traj is reference)⟩

Standard output

final results and statistics are written to the standard output

Additional output

running averaged and window averaged (using a window size of `@winframe`) order parameters are written to two seperate time series files (`OPts.out`, `OPwints.out`).

4.41. `noe` (GROMOS++ program)

Program description:

Program `noe` calculates and averages atom-atom restraint distances for specified NOE distances over a molecular trajectory. The NOE distances are to be specified in a NOE specification file that can be prepared with e.g. program `prep_noe` (see Sec. 4.45). Program `noe` will calculate the average distance according to $\langle \boldsymbol{r}^{-p} \rangle^{-1/p}$ for values of $p = 1, 3, 6$. It will also calculate the deviations of these distances from the specified reference distances, $\boldsymbol{r}_0$. These violations can be written to a time series file. The average violation is calculated as the sum of positive violations divided by the total number of NOE distances considered in the analysis.

The output of the program can be further analysed using program `post_noe`.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@noe`	⟨NOE specification file⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@time`	⟨time and dt⟩

Standard output

average distances, violations and average violations of all NOE distances

Additional output

none

4.42. post_noe (GROMOS++ program)

Program description:

Program post_noe allows the user to re-analyse the data that was generated by program noe (see Sec. 4.41). It reads in the NOE specification file, and the filter-file, which were generated by program prep_noe (Sec. 4.45), as well as the output of program noe.

In cases where a stereospecific hydrogen from a CH2-group, without an explicit assignment was given in the library file of program prep_noe (type 4 without subtype), the NOE distance according to both virtual atoms will have been calculated. Program post_noe can be used to select from these distances the proton that shows either the largest or the smallest violation with the experimental data. Additionally, the user can choose to disregard specific NOEs either by specifying a 0 in the filter field of the filter file that was generated by prep_noe, or by giving a cutoff distance.

The user may want to regenerate the filter file using a different kind of pseudo-atom and multiplicity correction by running the prep_noe program a second time. When using post_noe, the reference distances can then be read from this filter file allowing for a quick assessment of the effect of the corrections. Furthermore, the user can tell post_noe to change the exponential in the averaging method.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@noe`	⟨NOE specification file⟩
`@noeoutput`	⟨output of noe-program⟩
`@filter`	⟨NOE filter file⟩
`@averaging`	⟨1/3/6⟩

Optional input arguments	
`@distance`	⟨additional filter distance⟩
`@ref`	⟨noeoutput/filter⟩
`@minmax`	⟨min/max⟩
`@distribution`	⟨binsize⟩

Standard output

overview of NOE violations for the remaining NOE's after the filtering process, average NOE violations and if requested a distribution of the NOE violations

Additional output

none

4.43. postcluster (GROMOS++ program)

Program description:

Program postcluster can do additional analyses on the output of cluster (section Sec. 4.43). Three different kinds of analyses are currently possible on specified clusters:

1. postcluster can perform a lifetime analysis. A lifetime limit can be specified. This is the number of subsequent structures in the time series need to have switched to a different cluster before a true transition to the new conformation is taken into account. This allows the user to disregard single events from being counted as a double transition to and from a new conformation. The program will write out the number of times a certain cluster is observed, its average lifetime. In addition it prints for every cluster the number of transitions to and from the other clusters.
2. postcluster can also be used to analyse combined clusterings, in which the original structures come from different sources (e.g. different trajectories). This can be used to assess the overlap in the sampled conformational space between two simulations. This option is called @rgb. By specifying the number of frames from every individual source, the program will write out a file that can easily be used to produce a bar-plot in which the height of the bar indicates the size of the cluster and individual colors represent the portions of that cluster coming from the different sources.
3. postcluster can be used to write out trajectory files and single structure files containing the central member structures of the clusters. The trajectories can subsequently be used in any other analysis program to monitor properties over all structures belonging to one cluster.

Required input arguments

@topo	⟨molecular topology file (see Sec. 1.2)⟩
@cluster_struct	⟨structures file from cluster⟩
@cluster_ts	⟨time series file from cluster⟩
@clusters	⟨StructureSpecifier⟩

Optional input arguments

@lifetime	⟨lifetime limit⟩
@rgb	⟨red⟩ ⟨green⟩ ⟨blue⟩ ...
@traj	⟨trajectory files⟩

Standard output

lifetime analyses of the specified clusters

Additional output

If requested a file cluster_rgb.dat will be written out, containing the split up of clusters to their origins. If the original trajectory files were specified, central member structures and complete clusters will be written to files cluster_⟨X⟩.cms and cluster_⟨X⟩.trj, respectively, where ⟨X⟩ represents the cluster number.

4.44. `predict_noe` (GROMOS++ program)

Program description:

Program `predict_noe` is used to predict possible NOE pairs from distance averages. The program calculates and averages all possible NOE pairs from a trajectory. The nuclei are taken from the atom specifier provided if those are found in the NOE library file as well.

The averaging is carried out as

$$\overline{r} = \left\langle r^{-p} \right\rangle^{-\frac{1}{p}} \tag{4.48}$$

where p can be either 1, 3 or 6 (`@averaging`). Distances above a threshold level (`@filter`) are discarded in the final output.

Required input arguments

`@topo`	⟨molecular topology file⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1)⟩
`@lib`	⟨NOE specification library⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@dish`	⟨carbon-hydrogen distance (default: 0.1 nm)⟩
`@disc`	⟨carbon-carbon distance (default: 0.153 nm)⟩
`@averaging`	⟨averaging power: 1, 3 or 6 (default 6)⟩
`@filter`	⟨distance above which NOE's should be discareded [nm]⟩

Standard output

list of expected NOEs

Additional output

none

4.45. `prep_noe` (GROMOS++ program)

Program description:

Program `prep_noe` converts NOE data from an X-plor like format to GROMOS format, determining the proper choice of pseudo- or virtual atoms based on the topology and a library file. The output can be used to apply distance restraints during a simulation using program MD++, or to analyse a molecular trajectory using the program `noe` (Sec. 4.41). For a definition of the different types of pseudo- and virtual atoms see Sec. 2-9.4. In cases where the library file specifies a stereospecific CH2 atom (type 4), but does not indicate which of the two protons is specified, NOE upper bounds are created for both protons. Program `post_noe` can process the output of an NOE analysis to determine the best assignment.

The experimentally determined upper bounds are generally listed in a three column format, with distances in Å. `prep_noe` has three types of parsing these three columns, specified by `@parsetype`:

1. take the first value as the upper bound;
2. take the sum of the first and third values as the upper bound (default);
3. take the difference between the first and second values (commonly the lower bound).

The experimentally determined upper bounds can be corrected for pseudo-atom distances (addition of a geometric constant) or multiplicity factors (typically multiplication with $N^{1/p}$, where N is the multiplicity of indistinguishable protons involved and p is the averaging power). Such corrections can either be applied to the distances or can be taken out of a set of distances.

The NOE specification file (`@noe`) should contain the NOESPEC block (see section Sec. 4-7.8), which contains the ambiguous and unambiguous NOEs. For unambiguous NOEs, only the first eight columns of this file are to be specified. For ambiguous restraints, the 9th column repeats the number of the NOE (first column), the 10th column contains the number of NOEs this NOE may be linked to and the remaining columns lists the numbers of the NOEs to which it is linked.

Ambiguous NOEs can be assigned by program `post_noe` (see Sec. 4.42) by removing all but one of the ambiguous NOE distances through the `@minmax` flag. `prep_noe` writes a filter file which can be used to re-evaluate a given analysis over a specific trajectory, without recalculating all distances (also through program `post_noe`).

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@title`	⟨NOE title for output⟩
`@noe`	⟨X-plor like NOE specification file⟩
`@lib`	⟨NOE specification library⟩

Optional input arguments	
`@dish`	⟨carbon-hydrogen distance; default: 0.1 nm⟩
`@disc`	⟨carbon-carbon distance; default: 0.153 nm⟩
`@parsetype`	⟨Upper bound parse type: 1, 2 or 3⟩
	Choices are:
	1: Upper bound = first number
	2: Upper bound = first + third number (most common, default)
	3: Upper bound = first - second number (commonly the lower bound)
`@correction`	⟨correction file⟩ [⟨correction type⟩]
`@action`	⟨add⟩ or ⟨sub⟩ correction from upper bound; default: add
`@filter`	⟨discard NOEs above a certain distance [nm]; default 10000 nm⟩
`@factor`	⟨conversion factor Angstrom to nm; default is 10⟩

Standard output

distance restraints specification file to be used with the analysis tool `noe`

Additional output

noe.filter: NOE filter file, containing upper bounds and information on automatically generated NOE distances in case of unassigned stereospecific protons

noe.dsr: distance restraint file to be used as `@disres` input for MD++.

4.46. r_factor (GROMOS++ program)

Program description:

Program **r_factor** calculates crystallographic structure-factor amplitudes and phases from a given trajectory and compares them to experimental values. Only the atoms given by the AtomSpecifier **@atomssf** are considered for the calculation. The atoms' IAC are mapped to their element names according to the rules given in the **@map** file. The atoms' B-factors and occupancies are read from a special file (**@bfactor**) if requested or defaulted to 0.01nm^2 and 100%. Structure factors are calculated to the given resolution (**@resultion**) while the cell information is calculated from the system's box. Symmetry operations are taken into account by specifying a space group (**@spacegroup**). Make sure you only give asymmetric unit when using spacegroup. The program can write the electron density (a $2 \mid F^0 \mid - \mid F \mid$ map) to special files (FRAME_DENSITY_#.ccp4), if requested (density flag).

A bulk solvent correction can be applied if **@solvent** is given. In a first step a solvent mask is determined. Therefore the parameters r_{vdW}, r_{ion}, r_{shrink} and the IAC of the water oxygen have to be provided. The occupied space is determined by the van-der-Waals radius of the atoms plus a probe radius. The van-der-Waals radius of an atom is calculated as half of the distance where the Lennard Jones potential energy of the atom/water-oxygen interaction reaches its minimum. The probe radius is either taken as r_{vdW} (for neutral atoms) or r_{ion} (for charged atoms). The occupied space is shrinked by r_{shrink}. The structure factor is calculated as

$$F = F_{\mathrm{model}} + \rho \exp(-B \sin(\theta)^2/\lambda^2)\mathcal{F}(\mathbf{M}). \tag{4.49}$$

The parameters ρ and B are determined by least-square fitting. Initial values have to be provided. For numerical stability the reflections are split in a high and low resolution set in the fitting procedure. Therefore a resolution cutoff has to be given. Finally the maximum iterations have to be given.

Required input arguments	
@topo	⟨molecular topology file (see Sec. 1.2)⟩
@pbc	⟨periodic boundary and gathering (Sec. 1.2)⟩
@traj	⟨trajectory files⟩
@time	⟨time and dt⟩
@atomssf	⟨atoms considered in the structure factor calculation⟩
@cif	⟨crystallographic information file⟩
@map	⟨file with IAC-to-element name mapping⟩
@bfactor	⟨file with the B-factors and occupancies⟩
@resolution	⟨resolution range: minimum, maximum⟩

Optional input arguments	
@spacegroup	⟨space group in Hermann-Mauguin format, default: P 1⟩
@density	⟨write electron density maps⟩
@factor	⟨factor to convert length unit to Angstrom⟩
@bins	⟨number of resolution bins for computation of the R-factor⟩
@solvent	⟨solvent parameters: RVDW RION RSHRINK IACW IRHO IB RESCUT MAXIT. RVDW: van-der-Waals radius of the probe, RION: van-der-Waals radius of an atom, RSHRINK: radius for shrinking of the solvent mask, IACW: integer atom code of the solvent van-der-Waals atom (e.g OW for SPC), IRHO: initial value for ρ, IB: initial value for B, RESCUT: resolution cutoff, MAXIT: maximum iterations⟩

Standard output

R-factor for all resolution bins

Additional output

calculated and ,,observed" electron density maps for every frame.

4.47. `r_real_factor` (GROMOS++ program)

Program description:

Program r_real_factor calculates two electron densities. One (ρ) from the atomic positions and a second (ρ^0) from the structure factor amplitudes and calculated phases. Only the atoms given by the AtomSpecifier `@atomssf` are considered for the structure factor calculation.

The real space residual

$$R = \frac{\sum \alpha\rho^0 + \beta - \rho}{\sum \alpha\rho^0 + \beta + \rho} \tag{4.50}$$

is calculated for every residue. Summation is only carried out over the extent of the atoms contained in the AtomSpecifier `@atomsr`

For the documentation of the other arguments see Sec. 4.46.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨trajectory files⟩
`@time`	⟨time and dt⟩
`@atomssf`	⟨atoms considered in the structure factor calculation⟩
`@atomsr`	⟨atoms considered in the R factor calculation⟩
`@cif`	⟨crystallographic information file⟩
`@map`	⟨file with IAC-to-element name mapping⟩
`@bfactor`	⟨file with the B-factors and occupancies⟩
`@resolution`	⟨resolution range: minimum, maximum⟩

Optional input arguments	
`@spacegroup`	⟨space group in Hermann-Mauguin format, default: P 1⟩
`@factor`	⟨factor to convert length unit to Angstrom⟩

Standard output

A time-series of the real-space R-factor.

Additional output

none

4.48. `rdf` (GROMOS++ program)

Program description:

Program `rdf` calculates radial distribution functions over structure files or trajectories. The radial distribution function, $g(r)$, is defined here as the probability of finding a particle of type `J` at distance $\boldsymbol{r}$ from a central particle `I` relative to the same probability for a homogeneous distribution of particles `J` around `I`. Program `rdf` calculates $g(r)$ for a number of discreet distances $r_{r(k)}$, separated by distance dr as

$$g(r) = \frac{\mathcal{N}_{a\,J(k)}}{4\pi {r_{r(k)}}^2 dr \rho_J} \tag{4.51}$$

where $\mathcal{N}_{a\,J(k)}$ is the number of particles of type `J` found at a distance between $r_{r(k)}$ - 1/2 dr and $r_{r(k)}$ + 1/2 dr and ρ_J is the number density of particles `J`. If particles `I` and `J` are of the same type, ρ_J is corrected for that. At long distances, $g(r)$ will generally tend to 1.

Both atoms of type `I` and `J` can be solute atoms, solvent atoms as well as virtual atoms. If more than one particle of type `I` is specified, `rdf` calculates the average radial distribution function for all specified atoms.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@centre`	⟨atom specifier (see Sec. 1.3.1): atoms to take as centre⟩
`@with`	⟨atom specifier (see Sec. 1.3.1): atoms to calculate distances for⟩
`@cut`	⟨maximum distance⟩
`@grid`	⟨number of points⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@nointra`	⟨exclude intramolecular atoms ⟩

Standard output

radial distribution function of particles `J` around `I`

Additional output

none

4.49. `rep_ana` (GROMOS++ program)

Program description:

Program `rep_ana` extracts the information stored in the replica exchange molecular dynamics (REMD) output file `replica.dat`. It produces seven multi column output files:

`temperature.dat`	run number vs. temperature of replica (column 2 corresponds to replica 1, column 3 to replica 2 etc.)
`lambda.dat`	run number vs. lambda value of replica
`epot.dat`	run number vs. potential energy of replica
`probability.dat`	run number vs. switching probability of replica
`switches.dat`	run number vs. switching data of replica (0 = no switch in this run, 1 = switch in this run)
`prob_T.dat`	switching probabilities per temperature
`prob_l.dat`	switching probabilities per lambda

Furthermore it calculates an optimized temperature or lambda set based on the fraction of replicas diffusing from the lowest to the highest temperature (lambda value). This algorithm is based on[22].

Optional input arguments

`@repdata` ⟨REMD output file, `replica.dat`⟩

Optional input arguments

none

Standard output

none

Additional output

`temperature.dat`, `lambda.dat`, `epot.dat`, `probability.dat`, `switches.dat`, `prob_T.dat` and `prob_l.dat`

4.50. `rep_reweight` (GROMOS++ program)

Program description:

Program `rep_rewrite` sorts replica exchange trajectories according to the lambda values or the temperature and writes them to individual files.

Required input arguments

`@input`	⟨input file⟩
`@trj`	⟨cordinate trajectories⟩
`@name`	⟨prefix and postfix of output trajectories⟩

Optional input arguments

none

Standard output

none

Additional output

output file named ⟨`prefix`⟩_⟨`temperature`⟩_⟨`lambda`⟩`.`⟨`postfix`⟩

4.51. `reweight` (GROMOS++ program)

Program description:

Reweights a time series of observed values of X sampled during a simulation at state R (i.e. using the Hamiltonian $\hat{\mathcal{H}}_R = \hat{\mathcal{K}}_R(\vec{p}) + \hat{\mathcal{V}}_R(\vec{r})$) to another state Y (neglecting kinetic contributions for simplicity):

$$\langle X \rangle_Y = \frac{\langle X \exp\left[-\beta\left(\mathcal{V}_Y - \mathcal{V}_R\right)\right]\rangle_R}{\langle \exp\left[-\beta\left(\mathcal{V}_Y - \mathcal{V}_R\right)\right]\rangle_R} = \langle X \exp\left[-\beta\left(\mathcal{V}_Y - \mathcal{V}_R - \Delta\mathcal{F}_{YR}\right)\right]\rangle_R \tag{4.52}$$

with $\Delta\mathcal{F}_{YR} = \mathcal{F}_Y - \mathcal{F}_R$. The observed quantitiy X can be a structural quantity (e.g. the time series of an angle) or an energetic quantity (e.g. the time series of the ligand-protein interaction energy). Note that the reweighting will only give useful results if during the simulation at state R all configurations that are important to Y are sampled. The program reads three time series corresponding to the quantitiy X, the energy of state R, and the energy of state Y. All time series must have been calculated from the same ensemble R. The time series files consist of a time column and a column containing the quantity (i.e. X, $\mathcal{V}_R$, or $\mathcal{V}_Y$). The time series are obtained e.g. by `ene_ana` or `tser`. If the bounds flag is given a normalized distribution of X in the Y ensemble will be written out. When calculating averages and distributions special care is taken in order to avoid overflow (see[7]).

Required input arguments

`@temp`	⟨temperature of the system⟩
`@x`	⟨time series of quantity X⟩
`@vr`	⟨energy time series of the reference state R⟩
`@vy`	⟨energy time series of the end states⟩

Optional input arguments

`@bounds`	⟨lower bound⟩ ⟨upper bound⟩ ⟨grid points⟩

Standard output

average of quantity X in ensemble of state Y

Additional output

if bounds are given, a histogram (distribution) of the reweighted quantity X is given

4.52. `rgyr` (GROMOS++ program)

Program description:

Program `rgyr` calculates the radius of gyration, R_{gyr}, for a selected set of atoms over the trajectory according to

$$R_{gyr} = \sqrt{\frac{1}{N_{i=1}^{N}} \sum (\mathbf{r}_i - \mathbf{r}_{com})^2} \tag{4.53}$$

where N is the number of specified atoms, $\mathbf{r}_i$ is the position of particle i and $\mathbf{r}_{com}$ is the centre-of-mass of the N atoms.

Alternatively, the radius of gyration can be calculated in a mass-weighted manner,

$$R_{gyr} = \sqrt{\frac{1}{M} \sum_{i=1}^{N} m_i (\mathbf{r}_i - \mathbf{r}_{com})^2} \tag{4.54}$$

where M is the total mass of the specified atoms and m_i is the mass of particle i.

Please note that in case atoms from more than one molecule have been chosen, care should be taken in the choice of gathering method to ensure a proper calculation of the centre-of-mass.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@time`	⟨time and dt⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1) for the atoms to consider⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@massweighted`	(use massweighted formula)

Standard output

time series of the radius of gyration for the specified atoms

Additional output

none

4.53. `rmsd` (GROMOS++ program)

Program description:

The structural deformation of a molecule with respect to a reference structure can be expressed in terms of a root-mean-square deviation (RMSD) of the position of selected atoms. Program `rmsd` calculates the RMSD over a molecular trajectory after superimposing the centres of mass and performing a least-squares rotational fit. The fit can be performed using a different set of atoms than the calculation of the RMSD. The fit can be disabled by giving an empty set of atoms (compare Sec. 1.3.1).

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@time`	⟨time and dt⟩
`@atomsrmsd`	⟨atom specifier: atoms to consider for RMSD⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@ref`	⟨reference coordinates (if absent, the first frame of `@traj` is used as reference coordinates)⟩
`@atomsfit`	⟨atom specifier (see Sec. 1.3.1): atoms to consider for fit⟩

Standard output

time series of the root-mean-square deviation from the reference structure

Additional output

none

4.54. `rmsdmat` (GROMOS++ program)

Program description:

Program `rmsdmat` calculates the atom-positional root-mean-square deviation between all pairs of structures in a given trajectory file. This matrix of RMSDs can subsequently be used by program `cluster` to perform a conformational clustering. The matrix can be written out in human readable form, or – to save disk space – in binary format. For efficiency reasons, the RMSD values are written in an integer format. The user can specify the required precision of the RMSD values that are stored, if the precision is less or equal to 4, the values are stored as unsigned short int, otherwise as unsigned int.

Different sets of atoms can be selected to perform a rotational least-squares-fit and to calculate the RMS deviation from. The RMSD matrix can also be calculated from deviations in internal coordinates defined by a set of properties (e.g. torsional angles or hydrogen bonds). A selection of structures in the trajectory file to consider can be made using the options `@skip` and `@stride`. Structure pairs may occur for which the least-squares rotational fit fails for numerical reasons. In these cases both structures are fit to the reference structure. If no user specified reference structure is available, the first structure in the trajectory is taken as such. Specifying a reference structure allows the program `cluster` (section Sec. 4.4) to perform a forced clustering as well, requiring that the first cluster contains the reference structure, regardless of the cluster size.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@atomsfit`	⟨atom specifier (see Sec. 1.3.1): atoms to consider for fit⟩
`@atomsrmsd`	⟨atom specifier (see Sec. 1.3.1): atoms to consider for RMSD⟩
`@prop`	⟨property specifier (see Sec. 1.3.3): properties to consider for RMSD⟩
`@ref`	⟨reference coordinates⟩
`@skip`	⟨skip frames at beginning⟩
`@stride`	⟨use only every step frame⟩
`@human`	(write the matrix in human readable form)
`@precision`	⟨number of digits in the matrix (default 4)⟩
`@big`	(when clustering more than 50'000 structures)

Standard output

some information about the clustering process

Additional output

RMSD matrix in binary form (`RMSDMAT.bin`) or human readable form (`RMSDMAT.dat`) depending on the user specifications

4.55. `rmsf` (GROMOS++ program)

Program description:

Program `rmsf` calculates atom-positional root-mean-square fluctuations (RMSF) around average positions for selected atoms over a trajectory. A superposition of centres of mass and a rotational fit to a reference structure is performed for every structure in the trajectory. Different sets of atoms can be specified for the fitting procedure and for the calculation of the RMSF. The fit can be disabled by giving an empty set of atoms (see Sec. 1.3.1).

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@atomsrmsf`	⟨atom specifier (see Sec. 1.3.1): atoms to consider for RMSF⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@ref`	⟨reference coordinates (if absent, the first frame of `@traj` is reference)⟩
`@atomsfit`	⟨atom specifier: atoms to consider for fit⟩

Standard output

a list containing the atom-positional root-mean-square fluctuation for each of the atoms specified by `atomsrmsf`

Additional output

none

4.56. sasa (GROMOS++ program)

Program description:

Program `sasa` calculates and prints the solvent-accessible surface area (SASA) of all heavy atoms in the solute part of the molecular system. It also calculates the contribution made by a specified set of heavy atoms. The program uses the algorithm of Lee and Richards[23]. A spherical probe of given radius is rolled over the surface of the molecule (the size of the probe is typically 0.14 nm for water). The path traced out by its centre gives the accessible surface. In GROMOS, the radii of the heavy atoms are obtained by calculating the minimum energy distance of the interaction between the heavy atom and the first solvent atom. This value is reduced by the specified probe radius to account for the radius of the solvent atom.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨boundary type⟩ [⟨gather method⟩]
`@atoms`	⟨atom specifier (see Sec. 1.3.1): atoms to consider for sasa⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@zslice`	⟨distance between the Z-slices through the molecule (default: 0.005 nm)⟩
`@probe`	⟨probe IAC and radius (default: 4 0.14 nm)⟩
`@time`	⟨time and dt⟩
`@verbose`	(print summaries)

Standard output

time series of the solvent-accessible surface area for the selected heavy atoms and for all heavy atoms and, if requested, the atomic contributions for the selected atoms are also printed

Additional output

none

4.57. sasa_hasel (GROMOS++ program)

Program description:

Program sasa_hasel computes the solvent-accessible surface area (SASA) of all atoms in the solute part of the molecular system according to the method of Hasel et al.[24]. This is the method implemented in the SASA/VOL implicit solvent model. If a single conformation is given, either the atomic SASA values or the total SASA, along with the hydrophilic and hydrophobic contributions (defined by the sign of the sigma values given in the sasaspec file) may be printed. If multiple conformations are given, the averaged totals, the averaged atomic SASA values, or a time series of the total SASA values may be printed.

Required input arguments	
@topo	⟨molecular topology file (see Sec. 1.2)⟩
@pbc	⟨periodic boundary and gathering (Sec. 1.2)⟩
@sasaspec	⟨sasa specification library file⟩
@probe	⟨IAC of central atom of solvent and radius of solvent molecule⟩
@traj	⟨trajectory file(s)⟩

Optional input arguments	
@time	⟨time and dt (optional and only if time series)⟩
@timeseries	⟨if you want the time series as well as the average⟩
@timespec	⟨time points at which to compute the sasa: ALL (default), EVERY or SPEC (if time series)⟩
@timepts	⟨time points at which to compute the sasa (if time series and timespec EVERY or SPEC)⟩
@atomic	⟨print atomic sasa (only if not time series)⟩
@noH	⟨do not include hydrogen atoms in the sasa calculation (default: include)⟩
@p_12	⟨overlap parameter for bonded atoms (default: 0.8875)⟩
@p_13	⟨overlap parameter for atoms separated by two bonds (default: 0.8875)⟩
@p_1x	⟨overlap parameter for atoms separated by more than one bond (default: 0.3516)⟩

Standard output

single input conformation:

total sasa and contributions made by hydrophilic and hydrophobic atoms

single input conformation and @atomic:

atomic sasa values followed by total sasa and contributions made by hydrophilic and hydrophobic atoms

time series of input conformations:

average total sasa and contributions made by hydrophilic and hydrophobic atoms

time series of input conformations and @atomic:

average atomic sasa values followed by total sasa and contributions made by hydrophilic and hydrophobic atoms

time series of input conformations and @timeseries:

time series of total sasa and contributions made by hydrophilic and hydrophobic atoms, followed by averages

Additional output

none

4.58. `solute_entropy` (GROMOS++ program)

Program description:

Program `solute_entropy` takes a coordinate trajectory and calculates the configurational entropy using the Schlitter and the quasiharmonic analysis methods (see[25]) for a given set of atoms. The entropy can be averaged over a window of a given size. If requested, a superposition of centres of mass and a rotational fit prior to the entropy calculation is carried out.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@atomsentropy`	⟨atoms to consider for entropy calculation⟩
`@temp`	⟨temperature⟩
`@traj`	⟨trajectory files⟩

Optional input arguments	
`@time`	⟨time and dt⟩
`@ref`	⟨reference structure for fit⟩
`@ref_pbc`	⟨reference boundary type⟩
`@atomsfit`	⟨atoms to consider for fit⟩
`@method`	⟨method to use: schlitter or quasiharm⟩
`@n`	⟨entropy is calculated every nth step⟩

Standard output

time series of the calculated configurational entropy

Additional output

none

4.59. `structure_factor` (GROMOS++ program)

Program description:

Program structure_factor calculates crystallographic structure-factor amplitudes and phases from a given trajectory. Only the atoms given by the AtomSpecifier `@atomssf` are considered for the calculation. The atoms' IAC are mapped to their element names according to the rules given in the `@map` file. The atoms' B-factors and occupancies are read from a special file (`@bfactor`) if requested or defaulted to 0.01nm^2 and 100%. Structure-factor amplitudes are calculated to the given resolution (`@resultion`) while the cell information is calculated from the system's box. Symmetry operations are taken into account by specifying a space group (`@spacegroup`). When using `@spacegroup`, make sure only the asymmetric unit is given.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨trajectory files⟩
`@time`	⟨time and dt⟩
`@atomssf`	⟨atoms considered in the structure factor calculation⟩
`@map`	⟨file with IAC-to-element name mapping⟩
`@bfactor`	⟨file with the B-factors and occupancies⟩
`@resolution`	⟨resolution range: minimum, maximum⟩

Optional input arguments	
`@spacegroup`	⟨space group in Hermann-Mauguin format, default: P 1⟩
`@factor`	⟨factor to convert length unit to Angstrom⟩

Standard output

averaged structure factor amplitudes

Additional output

none

4.60. `temperature` (GROMOS++ program)

Program description:

Program `temperature` will calculate the temperature for different sets of atoms, as specified by the atom-specifier(s). Multiple sets of atoms can be specified by white-space separated Atomspecifiers (see Sec. 1.3.1). For each of the sets one dof value is expected.

You can find the number of degree of freedoms for a temperature group in the md++ output file under "DEGREES OF FREEDOM" → "DOF"

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@atoms`	⟨atom specifier⟩
`@dofs`	⟨degrees of freedom⟩
`@traj`	⟨velocity trajectory files⟩

Optional input arguments

`@time`	⟨time and dt⟩

Standard output

timeseries of temperature for each set of atoms

Additional output

none

4.61. `tcf` (GROMOS++ program)

Program description:

Program `tcf` performs simple statistical analyses, calculates distributions and time-correlation functions for any series of data points. It takes files with data listed in any number of columns as input, but no further formatting, e.g. the output files of programs `tser` (see Sec. 4.63) or `ene_ana` (compare Sec. 4.21). Lines starting with the character # are ignored.

For data in the specified columns, the program writes out the number of data points, the average value, root-mean-square fluctuation, a statistical error estimate as well as the minimal and maximal value observed. The error estimate is calculated from block averages of different sizes. In addition, the program can calculate distributions and time-correlation functions. The program does not read time from the data file, but the time interval between data points can be specified by the user. Otherwise it is taken to be 1.

Distributions can be calculated for data in specified columns and can be normalized.

Time correlation functions of the general form

$$C(t) = \langle f(A(\tau), B(\tau + t)) \rangle_\tau \tag{4.55}$$

can be calculated, where $A(\tau)$ and $B(\tau + t))$ represent the data points at different time points and the user can specify any function $f(A, B)$. The program can calculate both auto-correlation functions ($B = A$) and cross correlation functions ($B! = A$) for time series of scalars or vectors. If A and B are represented by scalars and $f(A(\tau), B(\tau+t)) = A(\tau)*B(\tau+t)$, the program makes use of fast Fourier transforms to calculate $C(t)$. In other cases a direct summation algorithm is used, which may be considerably slower.

In cases where one is interested in the correlation function of the fluctuations around the average, this average value can be subtracted from the data points before calculating the correlation function. A power spectrum can also be calculated. Because the correlation function is usually very noisy at larger times, the noise level can be specified as the fraction of the correlation function that should be considered. This part of the correlation function is then smoothened by multiplication by a cosine to make sure that it is zero at the end. It is then mirrored: all data points are repeated in reverse order at the end. From this the Fourier transform is taken, which is the resulting spectrum.

Required input arguments

`@time`	⟨time⟩ ⟨time step⟩
`@files`	⟨data files⟩

Optional input arguments

`@distribution`	⟨data columns to consider⟩
`@bounds`	⟨lower bound⟩ ⟨upper bound⟩ ⟨grid points⟩
`@normalize`	(normalize the distributions)
`@tcf`	⟨data columns to consider⟩
`@expression`	⟨expression for correlation function⟩
`@spectrum`	⟨noise level⟩
`@subtract_average`	(take difference with respect to average value for `tcf`)

Standard output

statistical analyses for all specified data columns, distributions and / or time-correlation functions

Additional output

none

4.62. `trs_ana` (GROMOS++ program)

Program description:

Program `trs_ana` extracts individual values from gromos trajectory files and can perform simple mathematical operations on them.

The program is based on `ene_ana` (see Sec. 4.21). It uses the same library format to define blocks which can be read from any trajectory file that comes in the Gromos block-format. In contrast to `ene_ana` it does not require that all the blocks defined in the library are present in the trajectory file or in the specified order. It can handle trajectories where not all timesteps contain the same number of blocks, e.g. when different properties were written to the trajectory at different intervals. The time in the output timeseries will always correspond to the time in the previous TIMESTEP block if no time is given by the user, else the time will be increased by the given timestep at every occurrence of a TIMESTEP block. If multiple blocks of the same name occur between two TIMESTEPs, only the last one will be used.

In the library file one can also define properties to be calculated from the defined entries. For the selected properties, trs_ana will calculate the time series, averages, root-mean-square fluctuations and a statistical error estimate. The error estimate is calculated from block averages of different sizes, as described in Allen and Tildesley: "Computer Simulation of Liquids", 1987. If a topology is supplied, the trs_ana uses this to define the total solute mass (MASS) and the total number of solute molecules (NUMMOL).

Required input arguments

`@trs`	⟨trajectory files⟩
`@prop`	⟨properties to monitor⟩
`@library`	⟨library for property names⟩

Optional input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩ (for `MASS` and `NUMMOL`)
`@time`	⟨t and dt⟩ (overwrites `TIME` in the trajectory files)

Standard output

averages, root-mean-square fluctuations and error estimates for the requested properties over the supplied trajectories

Additional output

time series of every property will be written to a separate file with name ⟨property⟩.dat.

4.63. `tser` (GROMOS++ program)

Program description:

Program `tser` can calculate structural quantities from a trajectory file and print the time series and/or a distribution of the value associated with the requested property. The quantity to be calculated is specified through a property specifier (compare Sec. 1.3.3) and can be any of the structural properties described in Sec. 1.3, which can be calculated from atomic positions in the trajectory file. Time series can later be analysed further with e.g. the program `tcf`.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@time`	⟨time and dt⟩
`@prop`	⟨property specifier (see Sec. 1.3.3)⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@nots`	(do not write time series)
`@dist`	⟨steps [min max]⟩
`@norm`	(normalise distribution)
`@solv`	(read in solvent)
`@skip`	⟨skip n first frames⟩
`@stride`	⟨take every n-th frame⟩

Standard output

time series and/or distributions of the specified properties

Additional output

none

4.64. `tstrip` (GROMOS++ program)

Program description:

Program `tstrip` removes all solvent coordinates from a (list of) trajectory files for ease of later analysis. Note that program `filter` (see Sec. 4.29) captures the functionality of `tstrip` as well.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@traj`	⟨input trajectory file(s)⟩

Optional input arguments	
`@nthframe`	⟨write every nth frame (default: 1)⟩
`@time`	⟨time and dt⟩ (overwrites `TIME` in the trajectory files)
`@notimeblock`	(suppresses reading and writing of TIMESTEP block)

Standard output

trajectory file for the solute atoms only

Additional output

none

4.65. `visco` (GROMOS++ program)

Program description:

Program `visco` calculates the bulk and shear viscosities from the elements of the pressure tensor that are written to MD++ energy trajectory files. In order to access this data, `visco` makes use of the `ene_ana` library. To obtain more accurate results from the simulation, the Einstein relation is used instead of the direct evaluation of the autocorrelation function (Green-Kubo formulation). Consider $\mathcal{P}_{\alpha\beta}$ as being an element of the pressure tensor. Consider that $\mathcal{G}_{\alpha\beta}(t)$ is the integral of $\mathcal{P}_{\alpha\beta}dt$:

$$\mathcal{G}_{\alpha\beta}(t) = \int_0^t \mathcal{P}_{\alpha\beta}(t)dt \tag{4.56}$$

We define $\eta_{\alpha\beta}$ as a viscosity term calculated in terms of the integral of the pressure component ($\mathcal{G}_{\alpha\beta}$). It will be proportional to the mean-square "displacements" of $\mathcal{G}_{\alpha\beta}(t)$ in the limit of infinit time.

$$\eta_{\alpha\beta} = \frac{\mathcal{V}}{2k_BT} \lim_{t\to\infty} \langle \frac{[\mathcal{G}_{\alpha\beta}(t+\tau) - \mathcal{G}_{\alpha\beta}(t)]^2}{\tau} \rangle \tag{4.57}$$

where $\mathcal{V}$ is the volume of the periodic box, k_B is the Boltzmann constant and T is the absolute temperature of the system. For isotropic systems, the estimation of the bulk viscosities can be obtained from the average of the viscosity terms obtained from the diagonal components of the pressure tensor:

$$\eta_{bulk} = (\eta_{xx} + \eta_{yy} + \eta_{zz})/3 \tag{4.58}$$

The shear viscosities of an isotropic system can be estimated by averaging the viscosity terms obtained from the off-diagonal elements of the pressure tensor:

$$\eta_{shear} = (\eta_{xy} + \eta_{xz} + \eta_{yz})/3 \quad . \tag{4.59}$$

The time series of the mean square "displacements" of $\mathcal{G}_{\alpha\beta}(t)$ are printed to separate files (`Gxx_msd.dat`, `Gyy_msd.dat`, `Gzz_msd.dat`, `Gxy_msd.dat`, `Gxz_msd.dat`, `Gyz_msd.dat`). In view of the poor statistics for long times, it is up to the user to decide the interval for which the least-squares-fitting should be performed. For convenience, program `visco` also prints the constant $\frac{\mathcal{V}}{2k_BT}$ and the conversion factors with respect to the commonly used units.

Required input arguments

`@en_files`	⟨energy files⟩
`@temp`	⟨temperature⟩
`@library`	⟨library file (same as for `ene_ana`)⟩

Optional input arguments

`@time`	⟨time and dt⟩

Standard output

none

Standard output

`Gxx_msd.dat`, `Gyy_msd.dat`, `Gzz_msd.dat`, `Gxy_msd.dat`, `Gxz_msd.dat`, `Gyz_msd.dat`

4.66. `xrayts` (GROMOS++ program)

Program description:

Program `xrayts` extracts the crystallographic restraints information form a special trajectory. In addition it calculated the minimal normal and free R factors.

Required input arguments	
`@restraj`	⟨special trajectory files⟩

Optional input arguments	
`@time`	⟨time and dt⟩

Standard output

Time-series of R factors.

Standard output

none

CHAPTER 5

Miscellaneous

5.1. `atominfo` (GROMOS++ program)

Program description:

Internally the GROMOS preparation and analysis tools determine which atoms belong to one molecule based on bonds specified in the topology. These programs can make use of the convenient atom specifier to select atoms, molecular properties etc. For efficiency reasons, the MD engine `md` numbers all atoms in the molecular system sequentially. Program `atominfo` can read both atom specifiers and sequential numbers (GROMOS-numbers) and will list the properties of the selected atoms.

The atom list can be sorted, according to the following priority: solute atom < virtual atom < solvent molecule. All programs that make use of atom specifiers (see Sec. 1.3.1) can also read in a file containing the output of atominfo, by specifying a file (⟨`atominfo` output file⟩). This allows the user to store complicated selections in a file for future use.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@gromosnum`	⟨GROMOS atom number⟩
`@atomspec`	⟨atom specifier (see Sec. 1.3.1)⟩

Optional input arguments	
`@sort`	(sort the atoms)
`@redun`	⟨1 for redundancy check (default), 0 for not (important when generating an atom list for gathering with redundant presence of one or more atoms)⟩

Standard output

list with atomic information of all specified atoms

Additional output

none

5.2. `close_pair` (GROMOS++ program)

Program description:

Program close_pair is to find the closest atom pairs between two molecules in a system with multiple molecules. It is mainly used to propose an atom list for the gathering of a complicated system, and can also be used to analyze the close contacts of two or more molecules in a system.

Periodicity and time series are supported in the program close_pair.

Note: the atom list proposed by the program close_pair represents the atom pairs that are closest to each other in two (specified) molecules, but not necessarily the best choice for gathering since in order to obtain an ideal picture of the unit cell, one should also consider the assembly of the molecules. For this purpose, the closest pairs beween one molecule and all the other molecules are also calculated, and if the closest molecule to the molecule that is to be gathered is not a good reference, one may choose other molecules as the reference, depending on the symmetry of the system.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨boundary type⟩
`@groupA`	⟨atoms to be analyzed that are from the molecule group to be gathered⟩
`@groupB`	⟨atoms to be analyzed that are from the reference molecule group for the gathering of group A⟩
`@traj`	⟨coordinate file⟩

Optional input arguments

`@dist`	⟨lower limit of distance for searching the closest pair. Default: 0.3 nm⟩
`@time`	⟨t0 and dt⟩

Standard output

The standard output contains two parts:

1. the close atom pairs between each molecule in group A and all molecules that are specified in group B;

2. the Summary part: contains the close atom pairs between two molecules that are closest to each other.

Additional output

A file called atominfo.atomspec is generated with an atom list which is from the Summary part of the standard output. The atom list is formatted and can be directly used by the program atominfo to generate an atominfo atom list file.

5.3. `frameout` (GROMOS++ program)

Program description:

Program frameout can be used to extract individual configurations from a molecular trajectory file. Three different formats are supported: the GROMOS96 format, the PDB format and an VMD-Amber format which can be read by program VMD. The user determines which frames should be written out and if solvent should be included or not. Atom positions can be corrected for periodicity by taking the nearest image to connected atoms, or to the corresponding atom in a reference structure. A centres of mass superposition and least-squares rotational fit to a reference structure can be performed based on selected atoms.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@spec`	⟨specification for writing out frames: `ALL` (default), `EVERY` or `SPEC`⟩
`@frames`	⟨frames to be written out⟩
`@outformat`	⟨output coordinates format, see Sec. 1.2⟩
`@include`	⟨`SOLUTE` (default), `SOLVENT` or `ALL`⟩
`@ref`	⟨reference structure to fit to or to gather with respect to⟩
`@atomsfit`	⟨atom specifier (see Sec. 1.3.1): atoms to fit to⟩
`@single`	⟨write to a single file⟩
`@time`	⟨time and dt⟩ (overwrites `TIME` in the trajectory files)
`@notimeblock`	(suppresses reading and writing of TIMESTEP block)

Standard output

none

Additional output

selected frames are written to files `FRAME_xxxxx.ext`, where `xxxxx` is the frame number of the individual frame and `ext` is determined by the file format

5.4. `inbox` (GROMOS++ program)

Program description:

Even though all GROMOS programs correct for periodic boundary conditions whenever necessary, it can sometimes be quite cumbersome to create a simulation box for display that contains all molecules. For simulations containing one or a few solute molecules, program `frameout` in combination with the proper gathering method will be sufficient, but for molecular systems consisting of many solute molecules, it may be that none of the gather settings works correctly.

Program `inbox` puts the atoms into the positive quadrant of the computational box according to the periodic boundary conditions. It can be used to visualize the computational box in a crystal simulation. The connectivity and gathering of charge groups is ignored, thus the charge groups (and solvent molecules) will not be gathered after application of this program.

One can specify the atoms which are put into the box. All other atoms are not affected by the program. By default all atoms are put into the box.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@atoms`	⟨atom specifier (see Sec. 1.3.1): atoms to put in the box⟩

Standard output

shifted coordinates, with all atoms forced to be within the simulation box, in pdb format

Additional output

none

5.5. `pairlist` (GROMOS++ program)

Program description:

Program `pairlist` determines all particles within user specified cutoffs from a given reference point. The reference point can either be an atom specifier (see Sec. 1.3.1) to a single atom or a set of three Cartesian coordinates. The output can be written in the same style as the output of the program `atominfo` to allow usage as an atom specifier (see Sec. 1.3.1) itself.

The program can produce two pairlists at the time, one short-range and one long-range. It will also print out a list of particles that occur in the long-range pairlist only. The pairlist determination can be done on an atomic basis or based on charge groups.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@coord`	⟨coordinates to base the list on⟩
`@refpos`	⟨atom specifier (see Sec. 1.3.1)⟩ or ⟨vector⟩

Optional input arguments

`@cutp`	⟨small cutoff⟩
`@cutl`	⟨large cutoff⟩
`@type`	⟨ATOMIC (default) or CHARGEGROUP⟩
`@atominfo`	(write in atominfo style)

Standard output

lists of particles within the short-range and long-range cutoff, or in the shell between short- and long-range cutoffs

Additional output

none

5.6. `shake_analysis` (GROMOS++ program)

Program description:

A SHAKE failure in one of the MD engines is one of the few indications that something is going wrong in your simulation. Most often, there is a mismatch between the topology and the set of coordinates, or an extremely strong force between particles is built up otherwise. Program `shake_analysis` is a diagnostic tool that can be used to evaluate all interaction energies for selected atoms, on a coordinate file right before or after a SHAKE failure. The output can be limited by specifying the number of interactions that should be displayed, or by giving an energy cutoff above which interactions should be listed.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@atoms`	⟨atom specifier (see Sec. 1.3.1): atoms for which shake fails⟩
`@cut`	⟨cut-off distance⟩
`@coord`	⟨coordinate file⟩

Optional input arguments

`@eps`	⟨epsilon for reaction field correction⟩
`@kap`	⟨kappa for reaction field correction⟩
`@top`	⟨number of non-bonded interactions per atom to print⟩
`@higher`	⟨print energies higher than specified value⟩
`@nocov`	(do not print covalent interactions)

Standard output

tables with interaction energies involving the specified atoms

Additional output

none

5.7. `unify_box` (GROMOS++ program)

Program description:

Program `unify_box` can convert different box shapes. All periodic boxes can be described as a triclinic box, which is defined by vectors **a**, **b** and **c**. The program is mostly used to convert a truncated octahedral box into a triclinic box or vice versa, according to[26]. The user can also specify a rotation matrix and **a**, **b** and **c** vectors directly.

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@to_pbc`	⟨target boundary condition⟩
`@pos`	⟨coordinate file⟩

Optional input arguments

`@from_pbc`	⟨original boundary condition⟩
`@rot`	⟨rotation matrix⟩
`@KLM`	⟨**a**, **b** and **c**⟩

Standard output

coordinates in which the molecular system has been rotated to fit the target box shape

Additional output

none

5.8. `rot_rel` (GROMOS++ program)

Program description:

The rotational relaxation time of molecules can be estimated from the autocorrelation function of the Legendre polynomials of molecular axes $\mathbf{r}_i, \mathbf{r}_j$ and $\mathbf{r}_k$.

$$C_1(t) = \langle \mathbf{r}_i(\tau) \cdot \mathbf{r}_i(\tau + t) \rangle_\tau \quad (5.1)$$

$$C_2(t) = \frac{1}{2}(3\langle \mathbf{r}_i(\tau) \cdot \mathbf{r}_i(\tau + t) \rangle_\tau^2 - 1) \quad (5.2)$$

Program `rot_rel` calculates the first and second order Legendre polynomials and calculates the time correlation functions. The user specifies two of the molecular axes, the third is defined as the cross product of the first two. The program can average the correlation functions over multiple molecules in the system. Note that the output of this program can also be produced by a combination of programs `tser` and `tcf` (see Secs. 4.63 and 4.61, respectively).

Required input arguments

`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@ax1`	⟨specify molecular axis 1⟩
`@ax2`	⟨specify molecular axis 2⟩
`@traj`	⟨trajectory files⟩

Optional input arguments

`@average`	⟨average over all molecules⟩
`@time`	⟨time and dt⟩

Standard output

autocorrelation functions of the first and second Legendre polynomials of the molecular axes

Additional output

none

5.9. `VMD plugin` (GROMOS++ program)

Program description:

This program is not an individual program but a plugin library which runs in the VMD (Visual Molecular Dynamics) program. It is used to open GROMOS configuration files directly in VMD.

Once a file is opened using one of the GROMOS plugins VMD will prompt for the arguments in the VMD console. Because the topological data in GROMOS is separated from the configurational data the plugin can only roughly guess the data from a configuration file. For this reason it is recommended to give information about the topology and periodic boundary conditions using the arguments. The arguments can also be read from a file (`@f`). Gathering and superpositioning and rotational fitting (to a reference structure or the first structure in the trajectory) can be carried out directly in the plugin.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩

Optional input arguments	
`@pbc`	⟨periodic boundary and gathering (Sec. 1.2)⟩
`@include`	⟨`SOLUTE` (default), `SOLVENT` or `ALL`⟩
`@ref`	⟨reference structure to fit to⟩
`@atomsfit`	⟨atom specifier (see Sec. 1.3.1): atoms to fit to⟩
`@time`	⟨time and dt⟩
`@factor`	⟨factor to convert length unit to Angstrom, 10.0⟩

5.10. `xray_map` (GROMOS++ program)

Program description:

Program `xray_map` is used to transform and/or filter crystallographic maps. It reads given CCP4 map files (`@map`) and atomic coordinates (`@pos`) and writes the final result to a CCP4 map file (`@out`) and/or prints some statistics (`@stat`) to the standard output.

If requested by `@expression` an expression is evaluated to calculate every grid points value from the maps, which are available in the expression via the symbols `rho1`, `rho2`, etc. By default the expression `rho1` is evaluated which corresponds of the value of the first map provided. A difference map, for example, can be calculated by giving `rho1 - rho2`.

The final map can be filtered by a simple cutoff criterion. All grid points closer than a given distance (`@cutoff`) to given atom centres (`@centre`) are included in the final map. All other grid points are set to zero.

If `@symmetrise` is given, the symmetry operations of the space group are applied in oder to create a P 1 map of the while unit cell.

Required input arguments	
`@topo`	⟨molecular topology file (see Sec. 1.2)⟩
`@pos`	⟨coordinate file for filtering and expressions⟩
`@map`	⟨map files⟩
`@out`	⟨output filename⟩

Optional input arguments	
`@stat`	⟨print map statistics⟩
`@expression`	⟨expression to evaluate at every grid point⟩
`@centre`	⟨AtomSpecifier of centre atoms⟩
`@cutoff`	⟨grid cell cutoff⟩
`@symmetrise`	⟨apply symmetry operations to create a P 1 map⟩
`@factor`	⟨factor to convert length unit to Angstrom⟩

Standard output

Map statistics

Additional output

A CCP4 crystallographic map

Bibliography

[1] C. H. Bennett. Efficient estimation of free-energy differences from Monte-Carlo data. *J. Comput. Phys.*, 22(2):245–268, 1976.

[2] M.R. Shirts, E. Blair, G. Hooker, and V.S. Pande. Equilibrium free energies from nonequilibrium measurements using maximum-likelihood methods. *Phys. Rev. Lett.*, 91:140601, 2003.

[3] 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.

[4] M. Neumann. Dipole-Moment Fluctiation Formulas in Computer-Simulations of Polar Systems. *Mol. Phys.*, 50(4):841–858, 1983.

[5] A. de Ruiter and C. Oostenbrink. Protein-ligand binding from distancefield distances and Hamiltonian replica exchange simulations. *J. Chem. Theor. Comput.*, 9:883 – 892, 2012.

[6] J. D. Chodera, W. C. Swope, J. W. Pitera, C. Seok, and K. A. Dill. Use of the weighted histogram analysis method for the analysis of simulated and parallel tempering simulations. *J. Chem. Theory Comput.*, 3(1):26–41, 2007.

[7] B.A. Berg. Multicanonical simulations step by step. *Comput. Phys. Commun.*, 153(3):397–406, 2003.

[8] G. Nagy and C. Oostenbrink. Dihedral-based segment identification and classification of biopolymers I: Proteins. *J. Chem. Inf. Model.*, 54:266 – 277, 2014.

[9] G. Nagy and C. Oostenbrink. Dihedral-based segment identification and classification of biopolymers II: Polynucleotides. *J. Chem. Inf. Model.*, 54:278 – 288, 2014.

[10] M.E. Davis, J.D. Madura, B.A. Luty, and J.A. McCammon. Electrostatics and diffusion of molecules in solution: simulations with the university of houston brownian dynamics program. *Comput. Phys. Commun.*, 62:187 – 197, 1991.

[11] C. Peter, W.F. van Gunsteren, and P.H. Hünenberger. Solving the Poisson equation for solute-solvent systems using fast Fourier transforms. *J. Chem. Phys.*, 116:7434–7451, 2002.

[12] C. Peter, W.F. van Gunsteren, and P.H. Hünenberger. A fast-Fourier-transform method to solve continuum-electrostatics problems with truncated electrostatic interactions: algorithm and application to ionic solvation and ion-ion interaction. *J. Chem. Phys.*, 119:12205–12223, 2003.

[13] W. Kabsch and C. Sander. Dictionary of protein secondary structure - Pattern-recognition of hydrigen-bonden and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983.

[14] S. Riniker, C.D. Christ, N. Hansen, A.E. Mark, P.C. Nair, and W.F. van Gunsteren. Comparison of enveloping distribution sampling and thermodynamic integration to calculate binding free energies of phenylethanolamine N-methyltransferase inhibitors. *J. Chem. Phys.*, 135:024105, 2011.

[15] N. Hansen, J. Dolenc, M. Knecht, S. Riniker, and W.F. van Gunsteren. Assessment of enveloping distribution sampling to calculate relative free enthalpies of binding for eight netropsin-DNA duplex complexes in aqueous solution. *J. Comput. Chem.*, 33:640–651, 2012.

[16] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1:269–271, 1959.

[17] D. N. Beratan. Electron-Tunneling Pathways in Ruthenated Proteins. *J. Am. Chem. Soc.*, 112:7915–7921, 1990.

[18] C. Schroeder and O. Steinhauser. Using fit functions in computational dielectric spectroscopy. *J. Chem. Phys.*, 132:244109, 2010.

[19] C. Schroeder. Collective translational motions and cage relaxations in molecular ionic liquids. *J. Chem. Phys.*, 135:024502, 2011.

[20] A. de Ruiter and C. Oostenbrink. Extended thermodynamic integration: efficient prediction of lambda derivatives at non-simulated points. *J. Chem. Theory Comput.*, 12:4476 – 4486, 2016.

[21] H. Yu and W.F. van Gunsteren. Charge-on-spring polarizable water models revisited: From water clusters to liquid water to ice. *J. Chem. Phys.*, 121:9549–9564, 2004.

[22] H. G. Katzgraber, S. Trebst, D. A. Huse, and M. Troyer. *Feedback-Optimized Parallel Tempering Monte Carlo Journal of Statistical Mechanics-Theory and Experiment*. Iop Publishing Ltd, 2006.

[23] B. Lee and F. M. Richards. The interpretation of protein structures: estimation of static accesibility. *J. Mol. Biol.*, 55:379–400, 1971.

[24] W. Hasel, T. F. Hendrickson, and W. C. Still. A rapid approximation to the solvent accessible surface areas of atoms. *Tetra. Comput. Method.*, 1:103–116, 1988.

[25] R. Baron, W.F. van Gunsteren, and P.H. Hünenberger. Estimating the configurational entropy from molecular dynamics simulations: anharmonicity and correlation corrections to the quasi-harmonic approximation. *Trends Phys. Chem.*, 11:87–122, 2006.

[26] H. Bekker. Unification of box shapes in molecular simulations. *J. Comput. Chem.*, 18:1930–1942, 1997.

Index