The GROMOS Software for (Bio)Molecular Simulation

Volume 5: Program Library Manual

August 24, 2012

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 se tup 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 Secs. 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] [arg3]`
   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 third arguments determine the gathering method. The available gathering methods (`arg2`) are:

`nog` or `0`	do not gather
`glist` or `1`	(default) gathering, based on a list of atoms
`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 the all atoms of the first molecule in the system

The third argument is used for specific gathering methods. If `glist` or `gltime` is used, then `arg3` should have the form:
`list <AtomsList>`
If `gref` or `grtime` is used, then `arg3` should have the form:
`refg <ReferenceStructure>`

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.
`vmdam`	VMD's „Amber Coordinates" format. An additional factor can be given to convert the length unit to Å, default 10.0.

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:N,CA,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 $\mathbf{x}$, `<`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:N,CA,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 Vol. 5 Sec. 2.23) 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⟩
`@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. 5-2.1). Alternatively, program `ran_box` (see Sec. 5-2.23) 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⟩
`@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