The GROMOS Software for (Bio)Molecular Simulation

Volume 6: Technical Details

January 9, 2021

Contents

CHAPTER 1

Outline of the GROMOS Code

1.1. MD++ outline

The code is split into two parts, the first one being an MD library containing basic functions necessary to run an MD simulation, the second one being the actual MD program. This second part is very small. It is therefore easy to write other specialised MD programs that make use of a subset of the functions provided in the library or apply them in a different order. The source code of the library is in turn split up into nine different parts: *math*, *simulation*, *topology*, *configuration*, *algorithm*, *interaction*, *io*, *util* and *check* (represented as C++ *namespaces*).

- *math* contains classes for vectors, matrices and vector arrays, mathematical operations, physical constants and periodic boundary treatment.
- *simulation* contains the simulation parameters supplied to run an MD or SD simulation or an EM.
- *topology* contains the topology of the simulated system, possibly also including a perturbation topology.
- *configuration* contains the state of a system: its coordinates, velocities, forces, restraints data and so on.
- *algorithm* contains classes that use information from *simulation* and *topology* to act upon a *configuration*. All steps during an MD or SD simulation or EM can be carried out using an *algorithm*.
- *interaction* contains the largest algorithm: the energy, forces and virial evaluation. Here, all interaction terms and their parameters are defined. Because of its size, *interaction* is a separate part, though it formally belongs to *algorithm*. The *interaction* part is further split into *bonded*, *nonbonded* and *special* interactions.
- *io* contains classes to read in or write out information. All file access is block oriented and human readable.
- *util* contains a few extra classes that are necessary to set up a simulation but which do not exactly belong to it. Parsing of command line arguments, generation of initial velocities or setting of debug levels are examples of classes found herein.
- *check* contains test routines. Testing includes the automatic calculation of energies under different conditions as well as the calculation of forces, virial tensor and energy λ-derivatives and their comparison to values obtained by finite difference calculations.

One step of an MD or SD simulation or EM consists of several `Algorithm`s (List. 1.1) applied to the `Configuration` in the right order.

```
1  class Algorithm {
2    public:
3    Algorithm(string name) : name(name) {}
4    ~Algorithm() {}
5    virtual int init(Topology & topo,
6                         Configuration & conf,
7                     Simulation & sim) = 0;
8
9    virtual int apply(Topology & topo,
10                     Configuration & conf,
11                     Simulation & sim) = 0;
12
13   string name;
14 };
```

Listing 1.1. Interface of the `Algorithm` class

The `Algorithm_Sequence` class (List. 1.2) is a container for all these algorithms. When a simulation is set up, they are inserted in the correct order into the `Algorithm_Sequence`. Before the start of a simulation, all algorithms will be initialised (by calling the `init()` function). During an MD step (`Algorithm_Sequence::run()`), the algorithms are applied (by calling `Algorithm::apply()`).

```
1  class Algorithm_Sequence : public vector<Algorithm *> {
2    public:
3    Algorithm_Sequence();
4    ~Algorithm();
5
6    int init(Topology & topo,
7             Configuration & conf,
8             Simulation & sim);
9
10   int run(Topology & topo,
11           Configuration & conf,
12           Simulation & sim);
13
14   Algorithm * algorithm(string name);
15 };
```

LISTING 1.2. Interface of the `Algorithm_Sequence` class. It is a container for `Algorithm` objects which provides methods to initialise and run the contained algorithms. It further provides access by name.

The force-field itself is also an `algorithm`, which, when applied, calculates the energies, forces and virial contribution of all force-field terms for the complete system. The force-field terms themselves are `Interaction` classes. The `Forcefield` is therefore a container to store the different `Interaction` objects (in analogy to the `Algorithm_Sequence` and `Algorithm` classes). When the force-field is applied, it calls `calculate_interactions()` on all interaction objects. There are distinct interaction objects for the covalent interactions (bond-length, bond-angle, improper-dihedral and torsional-dihedral interactions), the non-bonded interactions (pairlist construction, long-range interactions and short-range interactions) and the non-physical interactions (atom-position, atom-distance, dihedral-angle, NOE, ^3J-value or S^2 order-parameter restraints). It is very easy to add a custom `Interaction` class to calculate a non-standard interaction. An overview of the (non-bonded) interaction classes is given in Fig. 1.1. The `Nonbonded_Sets` contain independent subsets of the non-bonded interactions. Their `calculate_interactions()` method may be called in parallel (using either *shared* or *distributed* memory parallelisation). The `Nonbonded_Sets` share (through the `Nonbonded_Interaction`) a pairlist construction algorithm, which they call to create the part of the complete pairlist relevant to them. These different parts of the pairlist stay together with the `Nonbonded_Set` and need never be assembled into the complete pairlist. To gain flexibility, the calculation of the individual atom - atom pair interaction is further split up into a `Nonbonded_Outerloop` (loops over the atom - atom pairs), a `Nonbonded_Innerloop` (prepares the parameters necessary to calculate the interaction) and a `Nonbonded_Term` (calculates the atom - atom pair interaction energy, force and virial contribution). The `Storage` class provides directly accessible (local) memory for each `Nonbonded_Set`.

1.1.1. Efficiency. The main goal for writing a new C++ MD engine was to further improve on modularity (using some object-oriented features) and extendability (using clear and common interfaces between the modules). Nevertheless, a simulation code has to be reasonably efficient to be of practical use. The complete code is written in standard C++[1] with no language extensions or machine-specific parts, resulting in a highly portable program. This means that the compiler has to do all machine specific optimisations. We believe that the absence of any machine specific parts of code, which require duplication to be able to run on different machines, facilitates future modification. Furthermore, current compilers are getting ever better at producing fast programs, making use of the specific features available on the machine. In the inner loops of the interaction calculation, templates are used to generate specialised code. There are, for instance, specialised periodicity classes for the different implemented types of periodic boundary conditions (vacuum, rectangular, truncated octahedral and triclinic). The `Innerloop` methods are called with the boundary type as a template argument. Thus the compiler will generate a different specialised version of the inner loops for different boundary conditions automatically. In the same manner, the interaction function term of the non-bonded interaction can also be chosen (*e.g.* with or without switching function for non-bonded interactions) without any *if* statement required in the compiled inner loop. Example code fragments are shown

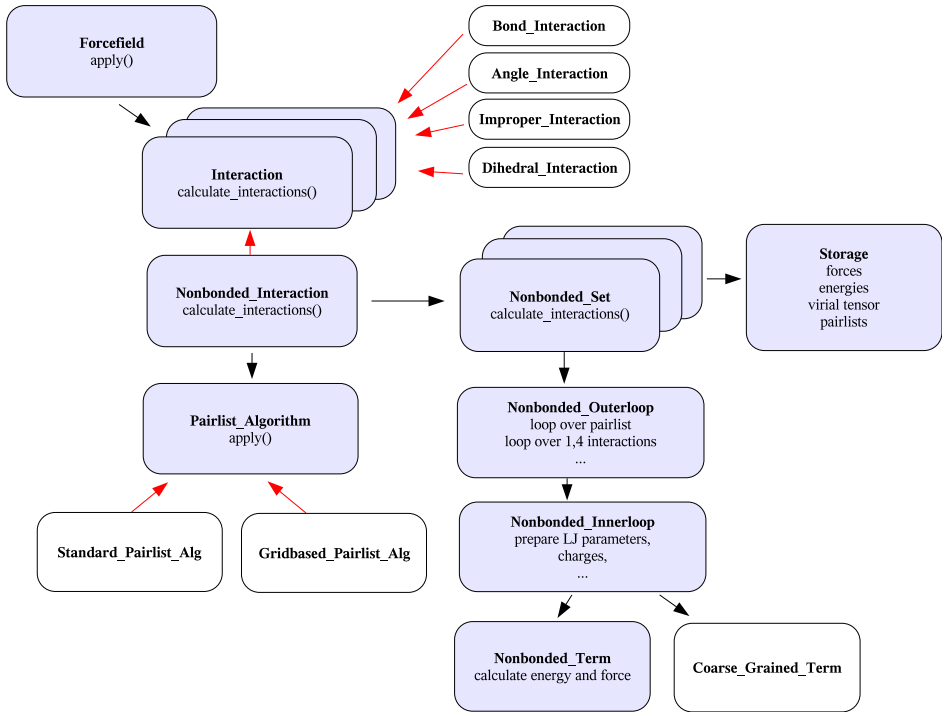

FIGURE 1.1. Illustration of the `Interaction` classes in MD++. The red arrows denote a *is-a* relationship, the black arrows *has-a*. All `Interaction` classes inherit from `Interaction` and, therefore, can be stored in the `Forcefield`, which is a `vector` of `Interaction` classes. The `Nonbonded_Interaction` consists of a `Pairlist_Algorithm` (either a `Standard_Pairlist_Algorithm` or a `Grid_Pairlist_Algorithm`) and (depending on parallelisation) one or more `Nonbonded_Set`s. Those, in turn, consist of `Storage` (to locally store forces, energies, virial tensor and pair lists) and an `Outerloop` (to calculate the interactions). The `Outerloop` relies on the `Innerloop` and on `Term` to calculate the interactions.

in List. 1.3 and List. 1.4. The same technique is used to implement perturbation simulations and different definitions of the virial tensor.

Some algorithms do rely on information from the previous integration step. To help implementing those kinds of algorithms, the complete current and old state (positions, velocities, forces, energies, restraint and constraint data, averages, and so on) of the simulation are stored. During the leap-frog algorithm, the current state becomes the old state and the updated information is stored in the new current state. This transfer is done by a simple and fast pointer exchange. This slightly increases memory usage, but the required space is still small compared to that used to store the pairlists.

1.1.2. Debugging information. It is often difficult to figure out what is going on during an MD or SD simulation or an EM and users tend to use the program as a *black box*. MD++ tries to improve this situation by enabling the user to select a tuneable amount of information to be printed out during the simulation. Every output or debugging message is associated with a debugging level, and the message is printed only if the requested debugging level is high enough. Additionally, every code section belongs to a *module* and a *submodule*. Different debug levels can be specified for all combinations of *modules* and *submodules*. In that way, fine grained control is achieved on how much information from which part of the MD++ code should be printed. For example, running MD++ like this

```
1 ~/> md @f md.args @verb interaction:special:4
```

will print all debug messages in the interaction/special part of the code with a level lower than four. Additional information on debugging can be found in the `doxygen` documentation.

1.1.3. In-code documentation. All classes, structures and enumerations are documented *in-code* using the `doxygen` documentation tool. This documentation contains descriptions of the classes of MD++ and their usage. Inheritance diagrams, function call relationships and interactive links to other classes

```
enum boundary_type {vacuum, rectangular, triclinic};
template<boundary_type boundary>
class Periodicity;

template<>
class Periodicity<vacuum>{
public:
  void nearest_image(Vec const & ri, Vec const & rj, Vec & rij);
};

template<>
class Periodicity<rectangular>{
public:
  void nearest_image(Vec const & ri, Vec const & rj, Vec & rij);
};

template<>
class Periodicity<triclinic>{
public:
  void nearest_image(Vec const & ri, Vec const & rj, Vec & rij);
};

template<boundary_type boundary>
class Interaction{
public:
  virtual int calculate_interactions(Topology const & topology,
                                     Configuration & configuration,
                                     Simulation const & simulation) {

    Vec r;
    Periodicity<boundary> periodicity(configuration.current().box);

    periodicity.nearest_image(
            configuration.current().pos(0),
            configuration.current().pos(1),
            r);
    const double r2 = math::abs2(r);
    // and so on
    return 0;
  }
};
```

LISTING 1.3. Specialized code generation using templates.

```
int main(int argc, char **argv) {
  Interaction<triclinic> interaction;
  interaction.calculate_interactions(
           topology, configuration, simulation);
  return 0;
}
```

LISTING 1.4. The usage of periodic boundary condition specific templates demonstrated on the `Interaction` class.

are automatically generated by the tool. The documentation further contains a brief description of the current input formats used in the given version of MD++. See Sec. 8-3.1 on how to generate the `doxygen` documentation during the compilation procedure of MD++.

1.2. GROMOS++ outline

GROMOS++ is a software package providing the user with tools to prepare all the needed input files for a standard simulation using MD++, e.g. the generation of the molecular topology, initial coordinates of randomly distributed molecules (solvent) or initial coordinates derived from a pdb file (solute), the solvation of a solute in the solvent and the split up of a simulation in multiple jobs with constant or changing simulation parameters over the job sequence. Furthermore, there are multiple programs to analyse the simulations performed. The following is a list of the most important GROMOS++ programs and the corresponding tasks. A complete list is available via the documentation tool, see Sec. 8-3.1 for more information:

- *com_top* combines multiple topology files into one file.

- *dssp* monitors secondary structure elements of a protein, based on the rules defined by Kabsch and Sander[2].
- *ene_ana* analyses (energy) trajectories.
- *frameout* writes out individual configurations or movies from a molecular trajectory file.
- *hbond* monitors the occurrence of hydrogen bonds.
- *ion* replaces water molecules by ions (to get an overall neutral box).
- *make_top* creates molecular topologies from building block and force-field parameter files.
- *mk_script* generates (multiple) script files to run simulations.
- *noe* analyses NOE distances over a trajectory.
- *pdb2g96* converts coordinate files from pdb to the GROMOS file format.
- *ran_box* creates a condensed phase system of any composition (randomly distributed molecules).
- *sim_box* solvates a solute in a box of pre-equilibrated solvent.
- *tser* calculates time series of properties which may be specified flexibly by the user (distances, angles, dihedral angles, intersection angles with planes, ...).

As mentioned before, this list is not complete and a lot of more specific analyses can be done using multiple programs in the right order.

Besides all the programs listed above there is a *contrib* collection of programs, a folder containing some GROMOS++ programs which are not of general use but treat a very specific topic or programs which were replaced by newer versions. A list and short explanation of these programs is available via the documentation tool, see Sec. 8-3.1 for more information.

1.2.1. GROMOS++ source code and in-code documentation. The GROMOS++ source code is divided into two major parts, one containing the programs and *contrib* programs, the other one collecting the tools (classes, structures and enumerations) used within the programs. The second part is in turn split up into eight different parts: *gromos*, *gcore*, *gmath*, *gio*, *bound*, *fit*, *args* and *utils* (represented as C++ *namespaces*):

- *gromos* handles the gromos exceptions (error messages).
- *gcore* contains all the classes that store the information about the molecular system, e.g. angles, bonds, atom properties, Lennard-Jones parameters, information and coordinates of the solvent and many more.
- *gmath* contains the tools of the basic vector and matrix algebra, handles time correlation functions and distributions for a series of values as well. There is also a class to handle a kind of pocket calculations read from a string (useful to mathematically interpret a program input parameter defining some specific properties or calculations).
- *gio* contains the tools to read in data or write them out. The read or written data may for example be a topology, coordinates, building block or input parameter files or any kind of trajectory.
- *bound* contains the classes to handle periodic boundary conditions (rectangular, triclinic, truncated octahedron and vacuum).
- *fit* is the namespace that contains code for translational superpositioning and rotational fitting of configurations.
- *args* contains classes to handle the different command line arguments needed by the programs.
- *utils* is the biggest and most manifold namespace. It contains a class which may perform some basic tests on a molecular topology, classes which provide the tools to observe hydrogen bonds or define secondary structure elements within the backbone of a protein using the rules defined by Kabsch and Sander[2], and many other classes. One of the most used classes within this namespace is probably the class AtomSpecifier: it defines and implements a general form to access atoms in a system. It is used to look over a specific set of atoms, possibly spanning different molecules. An AtomSpecifier is basically a string defining one or a group of atoms, used as an input parameter of a program. More detailed information about the exact format is given in the documentation tool (see Sec. 8-3.1).

All the classes, structures and enumerations of the eight namespaces used in GROMOS++ are documented *in-code* and available via the `doxygen` documentation tool. This also contains a description of all programs together with some example input parameters. Interactive links to other classes are automatically generated and help to understand the specific parts and functions of the code.

CHAPTER 2

Error Messages

Error checking is done in GROMOS with respect to three *types of inconsistencies.*

1. The *array sizes* defined in the header files may not be sufficiently large to cope with the *size of the molecular system* (solute, solvent, restraints, etc.) as specified in the input files. This type of inconsistency is signalled by an error message indicating the subroutine producing the error and that the value of an (input) variable is larger than the array size parameter MAX.... to be found in the header files. So, either the former should be reduced or the latter enlarged.

2. The *files* from which data are read by a program may contain data or data types that are *incompatible with* the *expectations of the program.* This type of inconsistency is signalled by an error message as described under Pt. 1.

3. The *control switches* governing the action of a program may be set such that *incompatible options* or *program actions* are selected. This type of inconsistency is signalled by an error message that specifies the incompatible conditions that have been selected.

The philosophy with respect to error checking in GROMOS is that the *user should be allowed to do silly things*, since what is silly in one case, may be useful in another. This means that only inconsistencies of the first type mentioned above are rigorously checked. GROMOS *error messages state the inconsistency*, so what's wrong, *not what's to be done* to remove the inconsistency. It is up to the user to think of and select the appropriate action to avoid the error message.

With respect to the inconsistencies of the types mentioned above, the error message indicates the line of the file where the error occurs and the name of the program or subroutine. This allows the user to identify and analyse the inconsistency in case the printed error message is not sufficiently informative.

CHAPTER 3

Machine Compatibility

The GROMOS programs, class libraries and subroutines have been written in *standard C++*[1]. This means that GROMOS should compile and run on any machine for which standard C++ compilers are available.

MD++ and GROMOS++ require a set of libraries to carry out numerical calculations. These libraries are written in the C programming language and can be compiled with the same compilers as MD++ and GROMOS++ themselves, See Chap. 8-2. To maximize operating system and compiler compatibility configuration is carried out by a *GNU Autotools* generated configuration script which generates the *Makefile*s and takes care of correct linking of the libraries. All calculations are carried out in 64 bit (double) precision only. Single precision may be available (by some compilers through their options) but is not recommended.

CHAPTER 4

Numerical and Mathematical Functions

The MD++ source code contains a set of mathematical classes and functions to carry out mathematical operations.

4.1. Numerical functions

MD++ contains two *random number generators*.

1. Function *math::RandomGeneratorG96::get()* generates (a series of) *uniformly distributed* random numbers between 0 and 1, using a linear congruential method.

2. Function *math::RandomGeneratorG96::get_gauss()* generates (a series of) normally or *Gaussian distributed* random numbers x with mean $<x>$ and standard deviation σ, using the Box-Müller method. The probability distribution is

$$p(x) \quad = \quad [2\pi\,\sigma^2]^{-1/2}\exp[-(x- < x >)^2/(2\,\sigma^2)]. \tag{4.1}$$

Alternatively, the random number generators as available in the GSL-libraries may be used, which are interfaced via the functions *math::RandomGeneratorGSL::get()* and *math::RandomGeneratorGSL::get_gaussian()*, respectively. The choice of random number generator is determined via the block RANDOMNUMBERS in the MD++ input file.

The Gaussian random number generator is used in program MD++ for the following purposes:

1. Generation of random initial atomic velocities, see Chap. 12.

2. Sampling of the stochastic integrals in a stochastic dynamics simulation, see Chap. 2-13.

4.2. Mathematical functions

4.2.1. MD++. MD++ source code contains a set of mathematical classes and functions to carry out mathematical operations. These classes and functions are grouped into the `math` namespace. Most of the classes are implemented as templates and can be used for either integer and floating point numbers.

1. Class `math::GenericVec<t>` is a generic three-dimensional vector type class. The standard arithmetic operators are overloaded. `math::Vec` is a widely used typedef to `math::GenericVec<double>`.
2. Class `math::GenericMatrix<t>` is a generic 3×3 matrix type class. The standard arithmetic operators are overloaded. `math::Matrix` is a widely used typedef to `math::GenericMatrix<double>`.
3. Class `math::GenericSymmetricMatrix<t>` is a generic 3×3 matrix type class. The only difference to the standard matrix is that it is symmetric.

There are several manipulation functions for vectors and matrices:

1. `product` either calculates the product of a matrix and a vector or the product of a two matrices.
2. `dot` and `cross` are used to calculate the scalar product and the vector product of two vectors.
3. `square` and `pow` are used to calculate the square or power of a matrix.
4. `inverse` calculates the inverse of a matrix.
5. `det` and `trace` compute the determinant and the trace of a matrix.
6. `transpose` gives the transposed of a matrix.

7. `tensor_product` and `dyade` calculate the tensor product of two vectors resulting in a matrix.
8. `symmetric_tensor_product` calculates the tensor product of two vectors assuming the result is a symmetric matrix.
9. `v2s` and `m2s` can be used to convert vectors and matrices to formatted strings.
10. `abs` and `abs2` calculate the length and the squared length of a vector.
11. `norm` normalises a vector to a length of 1.

The standard arithmetic and overloaded functions for vectors `math::GenericVec<t>` and matrices `math::GenericMatrix<t>` are

1. `operator+`/`operator-` either calculates the vector sum/difference or the sum/difference of two matrices/vectors.
2. `operator*`/`operator/` calculates the multiplication/division of a vector with/by a scalar.

The type `math::Box` is very similar to the matrix and is used to represent the box. There are constructors to convert the matrix and the box into each other. `product` can be used the calculate the product of a box with a vector. There are accessor functions to the column vectors of the box matrix.

The type `configuration::GenericMesh<t>` is a generic mesh class which can be used for gridded quantities. It is cuboid and constructed from the number of grid points. There are accessor functions to get and set the value of a grid point and transformation functions to carry out a fast Fourier transform (FFT) using an underlying FFT library (FFTW).

A more detailed description and a list of all available classes and functions can be found in the MD++ `doxygen` under Modules in math.

4.2.2. GROMOS++. The GROMOS++ source code contains a namespace `gmath` including a set of mathematical classes and functions to carry out mathematical operations for vectors and matrices and the calculation of (weighted) distributions and correlation functions. The namespace consists of the following classes:

1. Class `gmath::correlation` is a class to calculate time correlation functions. It calculates almost any kind of correlation function between two time series of scalars or vectors. The data should be provided by either two (or one) vectors of `double`, statistic classes or vectors `gmath:Vec` for vector correlation functions.
2. Class `gmath::Distribution` calculates a distribution for a series of values. The user has to specify an upper and lower bound and number of grid points. After adding all the values a distribution can be written out.
3. Class template `gmath::Stat<t>` is a class template to perform some basic statistics on a series of numbers (`double`, `float`). This class allows one to store a series of numbers and calculates the average, rmsd and an error estimate.
4. Class template `gmath::StatDisk<t>` is similar to the class template `gmath::Stat<t>` but stores the data in a scratch file and not in the memory.
5. Class `gmath::WDistribution` is similar to the class `gmath::Distribution` but calculates a distribution for a series of values with different weights.
6. Class `gmath::Vec` is a three-dimensional vector type class. The standard arithmetic operators are overloaded.
7. Class `gmath::Matrix` is a 3×3 matrix type class. The standard arithmetic operators are overloaded.

The standard arithmetic and overloaded functions for vectors `gmath::Vec` and matrices `gmath::Matrix` are:

- `operator+`/`operator-` either calculates the vector sum/difference or the sum/difference of two matrices.
- `operator*`/`operator/` calculates the multiplication/division of a vector with/by a scalar. The class `gmath::Matrix` is overloaded for multiplications (with a scalar) only.
- `dot` and `cross` are used to calculate the scalar product and the vector product of two vectors.
- `normalize` stretches a vector to a length of 1.
- `abs` and `abs2` compute the length and the squared length of a vector.
- `luDecomp` performs a single value decomposition of a matrix.
- `diagonaliseSymmetric` is used to diagonalise a symmetric matrix. The corresponding eigenvalues are returned.
- `det` and `fastdet3X3Matrix` calculate the determinant of a matrix.

- `transpose` returns the corresponding transposed matrix of a matrix.

CHAPTER 5

Nomenclature

In Vol. 4 the basic principles of storage and *identification of topological and configurational data* concerning a molecular system were discussed. Data or quantities related to e.g. atoms or atom-atom distance restraints, etc. are identified by their *position in the sequence* of such data or quantities, and *not* by their *names*, e.g. atom names or names of restraints, etc. This choice of the sequence number in a list as the key to identifying data has been made to keep GROMOS independent of naming conventions, e.g. for atoms, which may vary through the various fields of application of computer simulation.

For the convenience of the user, however, atoms can be given *atom names* and (sequential) groups of atoms can be given amino acid *residue* or nucleotide or glucose unit, etc. *names*. In principle, such names are only used for writing to file or printing, not as parameters in an algorithm. However, in GROMOS++ programs, atom names or residue names can be used to define a particular selection, as outlined below.

1. In *program make_top* the argument `@seq` is used to define the sequence of building blocks in order to make a topology. In this case, residue names (e.g. CYS1, CYS2, HIS1, HEME, ...) play a role when building a molecular topology from molecular topology building blocks.

2. In various *analysis programs*, atom names can be used as AtomSpecifiers, see Sec. 5-1.3, to select atoms for which a translational superposition and rotational positional *least-squares fit* is to be performed (e.g. in program rmsd, the argument of type atom specifier `@atomsfit` can be set to 1:CA, which means that all CA atoms of molecule 1 are selected). In various *analysis programs*, atom names and residue names can be or are used to define sets of atoms or quantities over which *averages* are to be calculated.

As long as no ambiguity is introduced, GROMOS data files contain atom names and residue or nucleotide names as defined by the *IUPAC-IUB convention*[3].

We note that for a *bond i-j* or an improper *dihedral angle i-j-k-l* the residue name that is associated with the bond or improper dihedral is the *residue name of the first atom, i, in the definition of the bond or improper dihedral*. For a *bond angle i-j-k* or a *torsional dihedral angle i-j-k-l* the residue name that is associated with the bond angle or torsional dihedral is the *residue name of the second atom, j, in the definition of the bond angle or torsional dihedral*.

CHAPTER 6

Units

Different sets of units are used in molecular simulations. In simulations of model systems, such as Lennard-Jones liquids, it is often advantageous to work with dimensionless quantities (*reduced units*) and apply the appropriate scaling to the required units afterwards. When treating realistic molecular systems the use of *Standard International (SI) units* is recommended. Apart from restrictions when storing or printing data in non-exponential format, the GROMOS programs are independent of the chosen units. The units are defined by the ones used for physical constants and atomic or molecular quantities in the (GROMOS) data files.

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

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

The basic units determine the units for other quantities, e.g. the quoted basic units yield

- *energy*:	E :	kJ mol^{-1}	$= 0.2390$ kcal mol^{-1}
			$= 10^3/\mathrm{N}_{Av}$
			$= 1.6605655\ 10^{-21}$ J
- *force*:	f :	kJ mol^{-1} nm^{-1}	$= 1.6605655\ 10^{-12}$ N
- *pressure*:	P :	kJ mol^{-1} nm^{-3}	$= 10^{30}/N_{\mathrm{Av}}$ Pa
			$= 1.6605655$ MPa
			$= 16.6057$ Bar
			$= 16.3885$ atm
- *velocity*:	v :	nm ps^{-1}	
- *Boltzmann's constant*:	k_B:		$= 8.31441\ 10^{-3}$ kJ mol^{-1} K^{-1}
			(= gas constant)
- *electric field*:	E :	kJ mol^{-1} e^{-1} nm^{-1}	
- *electric dipole*:	μ:	e nm	= 48.032424 D
- *polarisability*:	α:	e^2 nm^2 kJ^{-1} mol	$= 138.9354\ (4\pi\epsilon_0)$ nm^3

We have used the following *physical constants.*

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

Other physical constants required by GROMOS are, again using the basic units quoted above,

- ϵ_0 = permittivity of vacuum
  = 5.727659 10^{-4} kJ^{-1} mol e^2 nm^{-1}
- $(4\pi\epsilon_0)^{-1}$ = 138.9354 kJ mol^{-1} e^{-2} nm
- h = Planck's constant
  = 0.3990313 kJ mol^{-1} ps
- $\hbar$ = $h/\ 2\pi$
  = 0.06350780 kJ mol^{-1} ps
- c = speed of light
  = 2.99792458 10^5 nm ps^{-1}

We note that only a restricted set of units can be chosen independently. For example, if the energy unit is kcal mol^{-1}, the length unit is Angstrom and the mass unit is atomic mass unit, the time unit is a derived unit equal to 0.0488882 ps, which makes the use of these units in simulation a nuisance.

We note that it is possible to use for a quantity in a particular part of a calculation a unit that differs from the unit generally used in GROMOS for the quantity involved. For example, it is general custom to use the unit Hz = 10^{-12} ps^{-1} for ^3J-coupling constants, see Sec. 2-9.7. The recommended SI unit mentioned before would be ps^{-1}. However, one may choose to use the unit Hz for ^3J-coupling constants and the parameters a, b and c in (Eq. 2-9.62) as long as the energy units used for the parameters ${\mathcal{V}^{(Jr)}}_n$ (energy (time)$^{-2}$) are consistent with those of the other interaction terms. The GROMOS data files (Sec. 4-4.11) use kJ mol^{-1} Hz^{-2} for ${\mathcal{V}^{(Jr)}}_n$.

In the *(perturbation) molecular topology file* the atomic *charges* q_i are *stored as such* in the chosen units. In MD++ and GROMOS++, the charges are stored as in the molecular topology file.

A number of *quantities* or interaction function *parameters* in GROMOS are either angles or *dependent on angle units* through their definition.

1. Bond-angle bending interaction Sec. 2-5.2 and Sec. 2-17.2:
   bond angles θ_n (angle)
   parameters θ_n^0 (angle)
   parameters $k_n^{(\theta,h)}$ (energy (angle)$^{-2}$)

2. Improper dihedral-angle bending interaction Sec. 2-5.3 and Sec. 2-17.3:
   improper dihedral angles ξ_n (angle)
   parameters ξ_n^0 (angle)
   parameters $k_n^{(\xi)}$ (energy (angle)$^{-2}$)

3. Trigonometric dihedral-angle torsion interaction Sec. 2-5.4 and Sec. 2-17.4:
   dihedral angles φ_n (angle)

4. Dihedral-angle restraining interaction Sec. 2-9.6:
   dihedral angles φ_n (angle)
   parameters φ_n^0 (angle)
   parameters $k^{(tr)}$ (energy (angle)$^{-2}$)

5. ^3J-coupling constant restraining interaction Sec. 2-9.7:
   dihedral angles η_n (angle)

dihedral angles ζ_n (angle)
parameters δ_n (angle)

6. Local-elevation interaction Sec. 2-9.13.1:

dihedral angles φ_n(angle)
parameters $\varphi_n^{m'}$(angle)
parameters $\Delta\varphi_n^0$(angle)

For convenience of the user these quantities are kept in the *GROMOS data files* using *degrees as angle units*. However, when angles are used in calculations involving mathematical functions such as *sin*, *cos*, etc. they should be expressed in radians. Therefore, upon reading GROMOS data files the values of quantities and parameters that depend on angle units are converted from degrees to radians. So, in the *programs and functions* these quantities and parameters are stored using *radians as angle units.*

Finally, we note that the GROMOS programs can also be used using so-called *reduced units*, which are denoted by *:

- *length*:	r *	=	$r\ /\sigma$
- *mass*:	m *	=	m/M
- *time*:	t *	=	$t/[\sigma\ (M/\epsilon)^{1/2}]$
- *temperature*:	T *	=	$T/[\epsilon/k_B\ ]$
- *charge*:	q *	=	$q/[(4\pi\epsilon_0\sigma\epsilon)^{1/2}]$
- *energy*:	E *	=	$E\ /\epsilon$
- *force*:	f *	=	$f\ /[\epsilon/\sigma\ ]$
- *pressure*:	P *	=	$P\ /[\epsilon/\sigma^{\ 3}]$
- *velocity*:	v *	=	$v\ /[(\epsilon/M)^{1/2}]$

where the parameters ϵ and σ are defined by the Lennard-Jones interaction

$V(r_{ij}) = 4\epsilon\left[(\sigma/r_{ij})^{12} - (\sigma/r_{ij})^{6}\right]$

and M is the mass unit.

CHAPTER 7

Charge Group Codes

The concept of a charge group of atoms has been defined and discussed in Sec. 3-2.6.2. In the GROMOS non-bonded interaction subroutine NONBML the non-bonded energy and forces are only calculated between charge groups: for two (different) charge groups the interaction is, apart from excluded neighbours, either calculated between all or none (if the distance between the charge groups is longer than the cut-off radius) of the atoms forming the charge groups.

The *selection* of atoms belonging to a particular *charge group* is subject to the following *restrictions*.

1. *Atoms belonging to one charge group must have sequential atom sequence numbers.*
2. The solute may contain more than one charge group, but *atoms of different molecules may not belong to one charge group.*
3. *Each solvent molecule consists of one charge group.*

When *defining solute charge groups*, two considerations should be kept in mind.

1. The larger the charge groups, the smaller the total number of charge groups and the faster the charge group pair list can be constructed.
2. The larger the spatial size (radius R^{cg} of a charge group, the larger cut-off radius R_c must be used in order to avoid that the atoms of spatially adjacent charge groups do not interact with each other (see also Sec. 2-4.4):

   $R_c \gg 2 * largest R^{cg}$

The charge group definitions of the GROMOS force-field are given in Tabs. 3-3.12-3-3.16 for the 45A4 and 45B4 GROMOS force fields, Tabs. 3-3.27-3-3.31 for the 54A7 and 54B7 GROMOS force fields and in the figures of Chap. 3-5.

In GROMOS the identification of which atoms of the solute belong to which charge group is done in two different ways.

1. In the GROMOS *data files*, the molecular topology building block file (Sec. 4-5.2) and the molecular topology file (Sec. 4-3.2), *each solute atom i* has a so-called *charge group code* ICGM or ICG: the last atom of a charge group has ICG[i] = 1, whereas the other member atoms of a charge group have ICG[i] = 0.

2. In the *programs* and *functions* a *solute charge group pointer list* INC[1..NCAG] is used to indicate the NCAG charge groups of a solute. The atom sequence number of the last atom of the I -th charge group is stored in INC[I].

Upon reading the GROMOS topology files, the solute charge group codes ICGM[i] or ICG[i] are used to calculate the solute charge group pointer list INC[1..NCAG] (see Sec. 4-3.2 and Sec. 4-5.2).

CHAPTER 8

Pair List Generation

An essential part of the calculation of short-range nonbonded interactions is the construction of a pair list, containing all atom pairs separated by a distance less than the given cutoff distance. Various algorithms for the generation of pair lists are available in MD++.

8.1. Double loop pair list

A pair list is generated by a double loop, looping over $\frac{N_{CG}(N_{CG}-1)}{2}$ possible charge group pairs and checking their respective separation distance against the given cutoff R_c . A double loop pair list algorithm can be applied by setting algorithm=standard(0) in the PAIRLIST block of MD++. The double loop pair list algortihm is slow compared to grid-based pair list algorithms and is not recommended for the simulation of large systems.

8.2. Grid pair list (Heinz and Hünenberger)

A pair list is generated by sorting the charge groups into grid cells with a specified grid index, defined as

$$I_{grid} = N_y N_z i_x + N_z i_y + i_z \ , \tag{8.1}$$

where N gives the number of grid cells in each direction and i_x is defined as $i_x = mod(x, (x_{box}/N_x)$ (similarily for y and z).

For each grid cell, the neighbouring grid cell indices are found by the application of a mask array. This mask array contains the relative grid cell indices to all cells that may possibly be within the cutoff distance of the central cell, and the possibly neighbouring grid cells are found by adding the indices of the mask array to the index of the central cells. Due to the definition of the grid cell index Eq. 8.1, cells are sequential in the z-direction. This allows the mask array to be constructed as stripes, giving the grid cell range of each stripe in the z-direction. The sorted charge groups are stored in an array sorted by the same grid index. The algorithm applies one array of the size $\mathcal{O}(N_{cell}^3)$ pointing to the position in the charge group array of the first charge group of the previous non-empty grid cell. This enables the mask ranges to be converted to ranges in the sorted charge group array. This striping makes the overall algorithm less dependent on the grid size.

The Heinz and Hünenberger grid pair list algorithm can be applied by setting ALGORITHM=2 in the PAIRLIST block of MD++. For further information see[4].

8.3. Grid pair list with expanded coordinates

The original Heinz and Hünenberger pair list algorithm creates the mask such that the periodicity is implicit. This is done by adding the possible box-shifted stripes to the mask. Another approach is to use a non-periodic mask, but expanding the system by duplicating atom coordinates close to the box edge to its periodicly shifted boxes. This requires some overhead in terms of computational speed and memory but prevents the consideration of periodic shifts in the nonbonded calculation routines, and improves data locality.

This modified grid pair list algorithm can be applied by setting ALGORITHM=1 in the PAIRLIST block of MD++.

CHAPTER 9

Boundary Conditions and Periodicity

The concept of periodic boundary conditions has been discussed in Chap. 2-4, where also the formulae for the *three types of periodic boundary conditions* that can be selected in GROMOS were given.

1. *Periodic rectangular box.*

   switch NTB = 1, angle BETA = 90.0
   box lengths BOX[1..3] = a, b, c

2. *Periodic triclinic box.*

   switch NTB = 2, angle BETA[1..3] = α, β, γ
   box lengths BOX [1..3] = a, b, c

3. *Periodic truncated octahedral box.*

   switch NTB = -1, angle BETA = 90.0
   box lengths BOX [1..3] = a, a, a

The use of pressure coupling in a simulation under periodic boundary conditions (Sec. 2-12.5-Sec. 2-12.4) enables variations of the box parameters. The various options for these variations are:

1. no variations of the box parameters (NTP=0);
2. isotropic scaling, *i.e.* identical relative variations of the box-edge lengths only (NTP>0, NPCPL {1,1,1,0,0,0});
3. partially-anisotropic scaling, *i.e.* independent relative variations of the box-edge lengths only (NTP>0, NPCPL {i, j, k, 0, 0, 0} with i,j,k = 0,1,2 or 3 and i≠j or i≠k);
4. fully-anisotropic scaling, *i.e.* independent variations of all box parameters (box-edge lengths, box angles and Euler angles)(NTP>0, NPCPL {1,1,1,1,1,1}).

For a system under vacuum boundary conditions (VBC), only the first option is allowed. For a truncated-octahedral box, only the first two options are allowed. For a rectangular box, only the first three options are allowed. For a triclinic box, all options are allowed.

In GROMOS, the periodic boundary transformations are generally not performed for single atoms, but for all atoms of a charge group, that is, charge groups are (periodically) translated as one entity.

When considering a time series of configurations $\boldsymbol{r}(t_n)$ (n=1,2,3,...$\mathcal{N}_{tot}$), e.g. to obtain averages or time correlation functions, the function $\boldsymbol{r}(t_n)$ should be continuous in the time, that is, the atom positions $\mathbf{r}_i(t_{n-1})$ and $\mathbf{r}_i(t_n)$ at consecutive time points should satisfy the nearest image condition. This is enforced in the analysis programs and will only give correct results if the time difference $t_n - t_{n-1}$ is smaller than the time period that an atom needs to travel over a distance d equal to half the box size (i.e. d should satisfy the relations Eq. 2-4.55 or Eq. 2-4.60 or Eq. 2-4.59 with R_c replaced by d).

When using the GROMOS analysis package GROMOS++, the periodic boundary transformations are controlled by the switch

```
@pbc <boundary type> [<gathermethod>]
     [list <list of atom pairs>]
     [refg <reference structure>]
```

where *boundary type* is the periodic boundary condition used in generating the trajectory:

v - vacuum

r - rectangular

t - truncated octahedral

c - triclinic

The *gathermethod* controls which method is to be used for gathering. During a simulation the molecules in the system can cross the boundaries of the central periodic box. If one would depict the system in such a case, using a molecular visualization program, the molecules would appear as if they were 'broken'. Parts of the molecules that are outside the central box would appear inside of it, but on the opposite edge of the box. The gathering procedure puts the atoms comprising the molecular system into a single central box without 'breaking' the molecules. The available gathering options are:

`nog or 0`	no gathering;
`glist or 1`	gathering of each configuration based on a single list of pairs of atoms that are close to each other; 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; if the list argument is not given by the user, gathering is done based on the first atom of the previous molecule; the list argument should be in the atom specifier format;
`gtime or 2`	gathering based on the previous configuration; the first configuration is not gathered;
`gref or 3`	gathering of each configuration based on a reference structure; refg argument required;
`gltime or 4`	gathering based on the previous configuration; the first configuration is gathered based on a single list file; list argument required;
`grtime or 5`	gathering based on the previous configuration; the first configuration is gathered based on a reference structure; refg argument required;
`gbond or 6`	gathering of each configuration based on bond connectivity.
`cog or 7`	gathering based on the centre of geometry of the first molecule.
`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. Depends on correctly gathered reference ("refg" argument). Molecules to be selecte are specified with the "molecules" argument.

The default option in GROMOS is *glist*. For a single molecule in solution it is often not neccessary to gather the system. If gathering is required, then the methods gbond or gref are the most suitable ones. For systems with more than one molecule and for crystals gathering is usually an essential step in the analysis of trajectories. The correct gathering strongly influences the results of the calculation of atom-positional RMSD, RMSF, and other quantities. Not every method is appriopriate for every case, though. For instance, the *glist* method is not suitable for e.g. lipid bilayers, since the lipids can flip from one layer to the other. The lipid which moved to another layer will be close to completely different lipid molecules than before the flip.

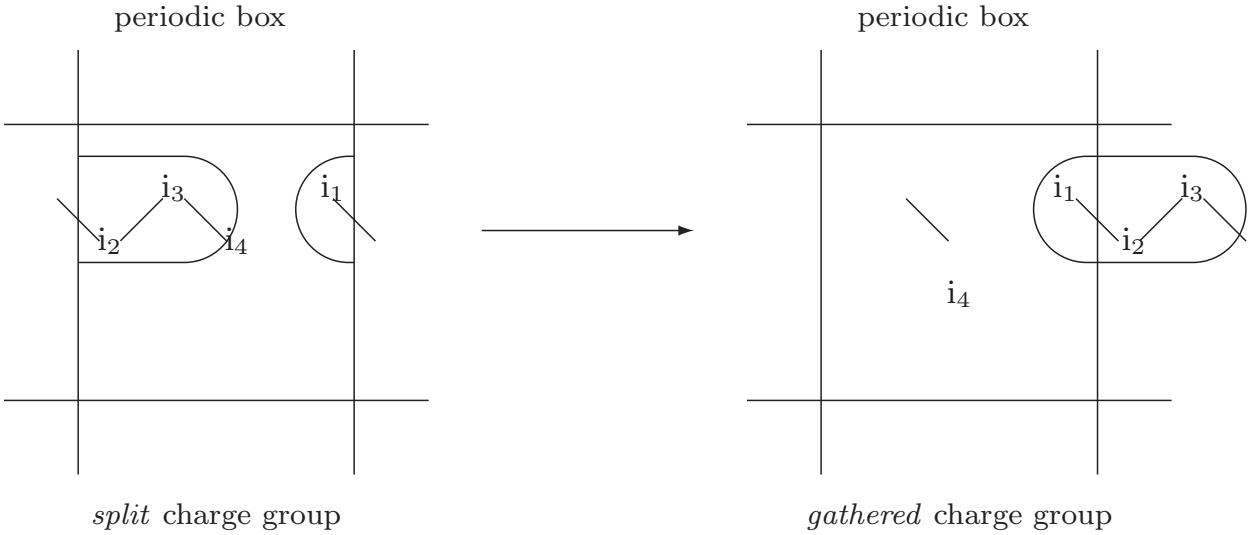

FIGURE 9.1. The atoms with atom sequence numbers i_1, i_2 and i_3 belong to a solute charge group and are gathered such that the spatial closeness of the charge group is not broken by the periodic boundary condition.

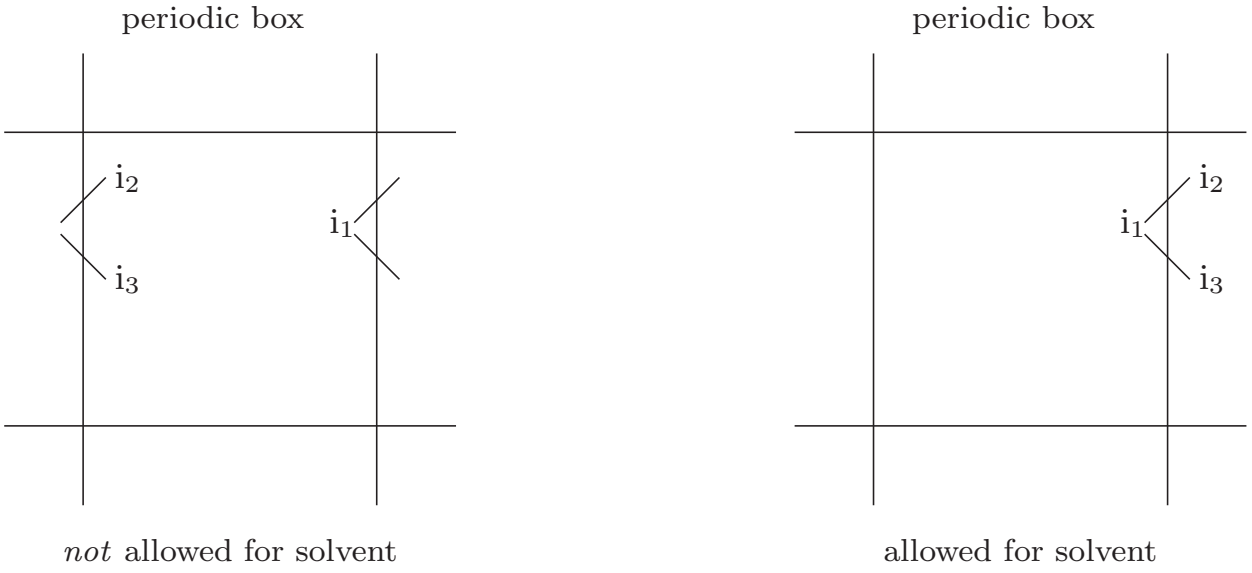

FIGURE 9.2. For the three atoms of a solvent molecule the simulation in the left figure must never occur in GROMOS, that is, the covalent bonds must not be split by the periodic boundary condition. All solvent configurations must be as in the right figure.

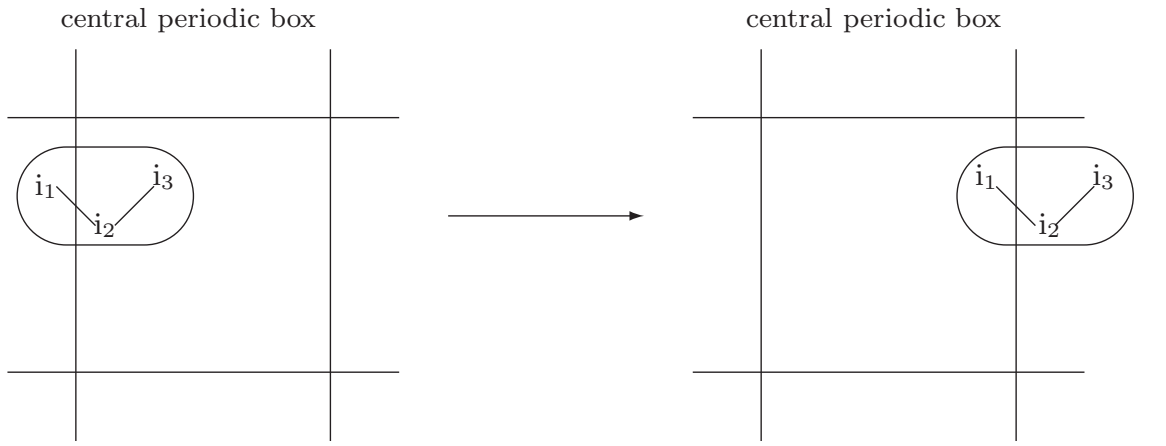

FIGURE 9.3. For the three atoms of a solvent molecule the simulation in the left figure must never occur in GROMOS, that is, the covalent bonds must not be split by the periodic boundary condition. All solvent configurations must be as in the right figure.

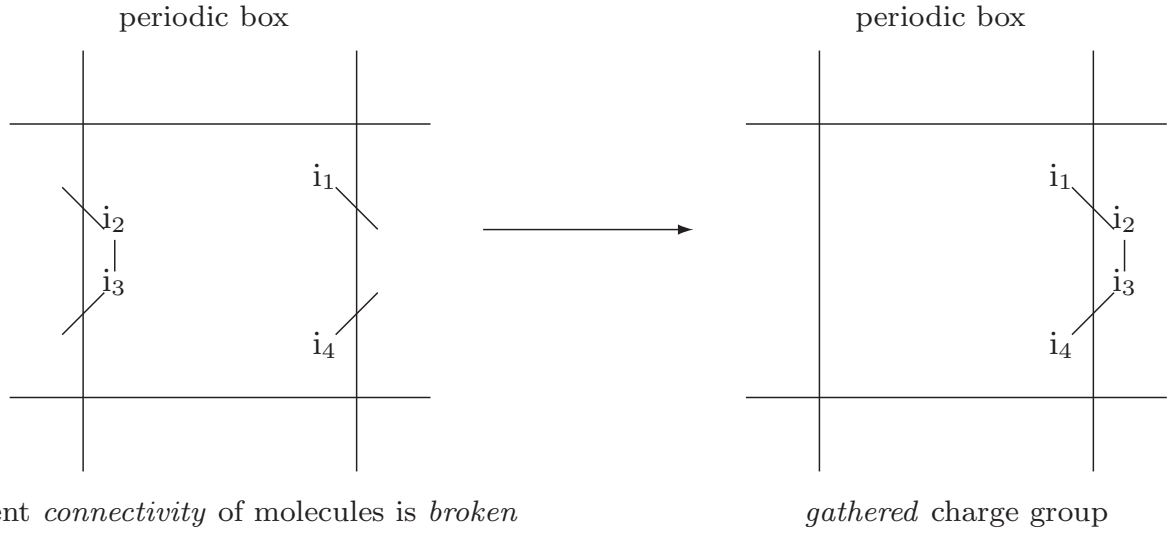

FIGURE 9.4. The covalent connectivity of a molecule with atom sequence numbers i_1, i_2, i_3, i_4 that is broken by the periodic boundary condition, is restored by making the atomic coordinates of atoms for which the sequence numbers differ only by one satisfy the nearest image condition.

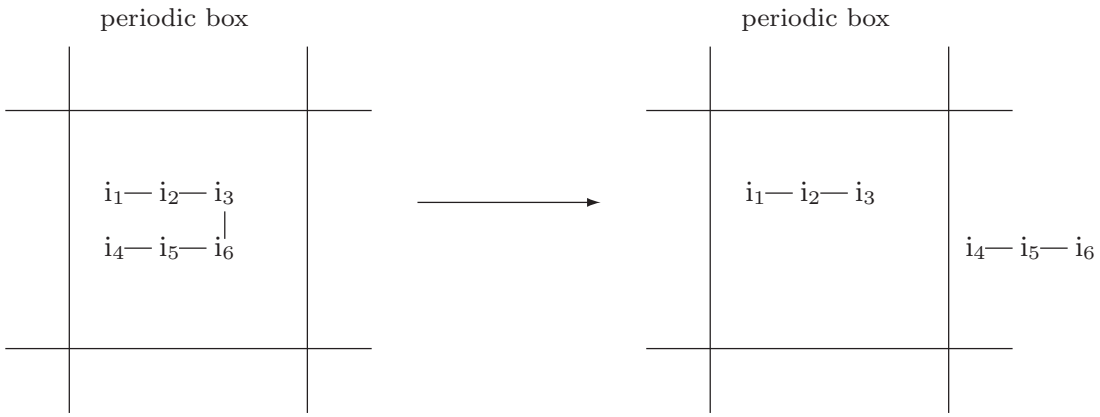

FIGURE 9.5. *Undesired* action of gathering routines.
The atoms are put into the nearest neighbour position with respect to the previous atom in the atom sequence. Since after atom i_3 the numbering of the atoms does not follow more or less (i.e. within half a box length) the covalent connectivity of the chain, the latter is broken between atoms i_3 and i_6 by the gather method. Selecting the proper gather method can be a challenging task.

CHAPTER 10

Generation of Cartesian Coordinates from Internal Coordinates

The configuration of a molecule can be characterized by *different types of coordinates*.

1. *Cartesian coordinates* for all the atoms of the molecule.

2. *Internal coordinates* such as bond lengths, bond angles and dihedral angles, one of each per atom plus the spatial positions of three atoms not lying on a line.

GROMOS++ contains programs and functions performing the transformations from Cartesian to internal coordinates and backwards.

1. Program tser (see Sec. 5-4.63) *calculates bond-length or bond-angle or dihedral-angle values* from sets of Cartesian coordinates.
2. Program gca (see Sec. 5-2.11) *generates Cartesian coordinates* from a set of bond-length, bond-angle and dihedral-angle values.

Using the IUPAC-IUB convention for dihedral angle values the position $\boldsymbol{r}_{l'}$ of atom l after rotation over a dihedral angle $\Delta\varphi$ around the axis defined by the line connecting atoms j and k and starting from a position $\boldsymbol{r}_l$ before the rotation is

$$\mathbf{r}_{l'} = \mathbf{r}_l + [\cos(\Delta\varphi) - 1]\frac{\mathbf{r}_{nk}}{r_{kj}^2} + \sin(\Delta\varphi)\frac{r_{nk}}{r_{kj}^2}\frac{\mathbf{r}_{mk}}{r_{mk}}, \tag{10.1}$$

where

$$\begin{aligned} \mathbf{r}_{kj} &= \mathbf{r}_k - \mathbf{r}_j \\ \mathbf{r}_{lk} &= \mathbf{r}_l - \mathbf{r}_k \\ \mathbf{r}_{mk} &= \mathbf{r}_{kj} \times \mathbf{r}_{lk} \\ \mathbf{r}_{nk} &= \mathbf{r}_{mk} \times \mathbf{r}_{kj} \end{aligned} \tag{10.2}$$

and $\Delta\varphi = \varphi$(i-j-k-l') - φ(i-j-k-l) for example.

Resetting the dihedral angles along a linear covalently bound chain of atoms, as in Fig. 10.1, is straightforward. When rotating atom i_n (n = 4,5,..., N) around bond i_{n-2} - i_{n-1} over $\Delta\varphi$, all atoms i_m with $n < m \leqslant N$ should be rotated over the same angle $\Delta\varphi$ in order to avoid deformation of the molecule. When the chain is branched, as in Fig. 10.2, it must be possible to specify an upper bound M for the atom sequence numbers m so that only the atoms m with $n < m \leqslant M$ of a side chain are rotated with atom n if n denotes a side chain atom.

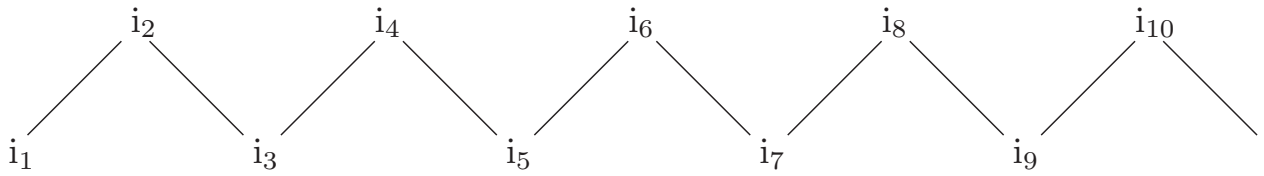

FIGURE 10.1. *Unbranched* covalently bound *chain*. Rotating atom i_4 around $i_2 - i_3$ over an angle $\Delta\varphi$, atoms i_5 to i_{10} should also be rotated over $\Delta\varphi$.

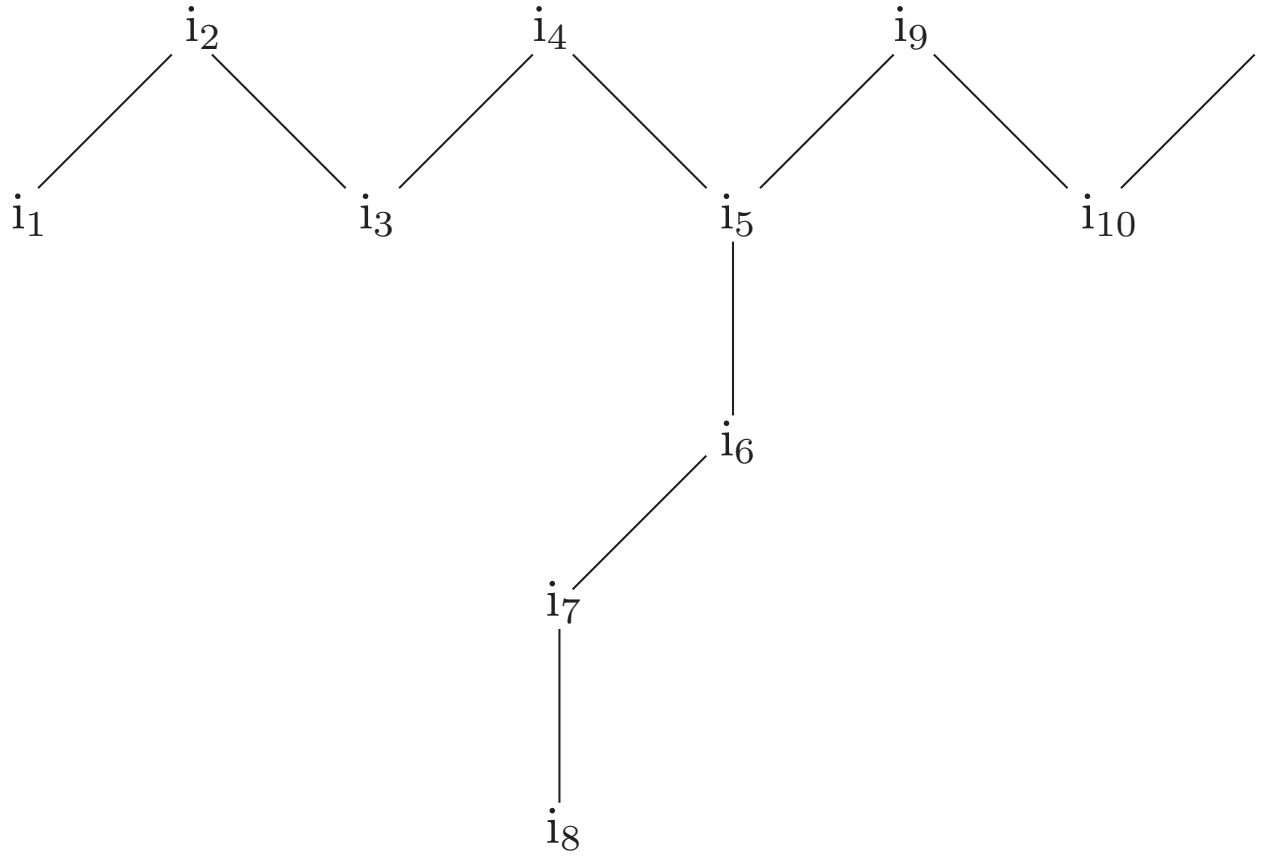

FIGURE 10.2. *Branched* covalently bound *chain*. Rotating atom i_7 around $i_5 - i_6$ over an angle $\Delta\varphi$, atom i_8 should also be rotated over $\Delta\varphi$.

CHAPTER 11

Generation of Hydrogen Atom Coordinates

Generally, molecular configurations as obtained by X-ray diffraction experiments do not contain coordinates for hydrogen atoms. If the molecular model and force field include an explicit representation of hydrogen atoms, their coordinates must be generated using the coordinates of the non-hydrogen atoms of the molecule.

When generating a GROMOS coordinate file from a PDB file, the GROMOS++ program pdb2g96 generates enries for all hydrogen atoms for which no coordinates were present in the PDB file and sets the corresponding Cartesian coordinates to zero (see Sec. 5-2.19).

The GROMOS++ program gch (Sec. 5-2.12) can be used to generate the Cartesian coordinates for the hydrogen atoms. With gch, it is possible to use 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 degrees for tetrahedral centers and 120 degrees for planar centers.

Eight types of geometry can be handled by gch:

1. An atom (i) is bonded to one hydrogen (H) and one other heavy atom (j). A fourth atom (k) is searched for which is bonded to j and preferably is used to define the dihedral around the j-i bond. The coordinates of H are generated in such a way that the dihedral k-j-i-H is trans and that the angle j-i-H and bond length i-H correspond to their minimum energy values. Considering that b is the bond length i-H and α is the angle j-i-H, the position $\mathbf{r}_H$ of the hydrogen atom is given by

$$\mathbf{r}_H = \mathbf{r}_i + b \cdot \frac{\mathbf{t}}{t}, \tag{11.1}$$

with

$$\mathbf{t} = \cos\alpha \cdot \frac{\mathbf{r}_{ji}}{r_{ji}} - \sin\alpha \cdot \left[\left(\frac{\mathbf{r}_{ji}}{r_{ji}} \right) \times \left(\frac{\mathbf{t}_a}{t_a} \right) \right], \tag{11.2}$$

in which

$$\mathbf{t}_a = \mathbf{r}_{ji} \times \mathbf{r}_{kj}. \tag{11.3}$$

2. An atom (i) is bonded to one hydrogen (H) and two other heavy atoms (j and k). The coordinates of H are generated to be in the plane through j, k and i, on the line bisecting the j-i-k angle and with an i-H bond length corresponding to the minimum energy value in the topology, such that the j-i-H and k-i-H angles are larger than 90 degrees. Considering that b is the bond length i-H, the position $\mathbf{r}_H$ of the hydrogen atom is given by

$$\mathbf{r}_H = \mathbf{r}_i - b \cdot \frac{\mathbf{t}}{t}, \tag{11.4}$$

with

$$\mathbf{t} = \frac{\mathbf{r}_{ji}}{r_{ji}} + \frac{\mathbf{r}_{ki}}{r_{ki}}. \tag{11.5}$$

3. An atom (i) is bonded to two hydrogens (H1 and H2) and one other heavy atom (j). A fourth atom (k) is searched for which is bonded to nh and preferably is used to define the dihedral around the j-i bond. The coordinates of H1 are generated in such a way that the dihedral k-j-i-H1 is trans and that the angle j-i-H1 (α_1) and bond length i-H1 (b_1) correspond to their minimum energy values. The coordinates of H2 are generated to have the angles j-i-H2 (α_2) and H1-i-H2 (α_3) as well as the bond length i-H2 (b_2) at their minimum energy values. If this does not result in a planar configuration around i, the improper dihedral i-j-H1-H2 will be positive.

$$\mathbf{r}_{H1} = \mathbf{r}_i + \mathbf{t}_1, \tag{11.6}$$

with

$$\mathbf{t}_1 = b_1 \left[\cos\alpha_1 \cdot \frac{\mathbf{r}_{ji}}{r_{ji}} - \sin\alpha_1 \cdot \frac{\mathbf{t}_a}{t_a} \right], \tag{11.7}$$

and with

$$\mathbf{t}_a = \frac{\mathbf{r}_{ji}}{r_{ji}} \times \left[\frac{\mathbf{r}_{kj}}{r_{kj}} \times \frac{\mathbf{r}_{ji}}{r_{ji}} \right]; \tag{11.8}$$

and

$$\mathbf{r}_{H2} = \mathbf{r}_i + \mathbf{t}_2, \tag{11.9}$$

with

$$\mathbf{t}_2 = c_1 \frac{\mathbf{r}_{ji}}{r_{ji}} + c_2 \frac{\mathbf{t}_a}{t_a} + c_3 \left[\frac{\mathbf{r}_{kj}}{r_{kj}} \times \frac{\mathbf{r}_{ji}}{r_{ji}} \right], \tag{11.10}$$

in which the scalar coeficients c_1, c_2 and c_3 are given by

$$c_1 = b_2 \cdot \cos\alpha_2 \tag{11.11}$$

$$c_2 = \frac{b_1 \cdot b_2 \cdot \cos\alpha_3 - c_1 \left(\frac{\mathbf{r}_{ji}}{r_{ji}} \cdot \mathbf{t}_1 \right)}{\frac{\mathbf{t}_a}{t_a} \cdot \mathbf{t}_1} \tag{11.12}$$

$$c_3 = \sqrt{b_2^2 - c_1^2 - c_2^2}. \tag{11.13}$$

4. An atom (i) is bonded to three hydrogens (H1, H2 and H3) and one other heavy atom (j). A fourth atom (k) is searched for which is bonded to j and preferably is used to define the dihedral around the j-i bond. The coordinates of H1 are generated in such a way that the dihedral k-j-i-H1 is trans and that the angle j-i-H1 (α_1) and bond length i-H1 (b_1) correspond to their minimum energy values. The coordinates of H2 are such that the angles j-i-H2 (α_2) and H1-i-H2 (α_4) and the bond length i-H2 (b_2) are at their minimum energy values, and the improper dihedral i-j-H1-H2 is positive. The

coordinates of H3 are such that the angles j-i-H3 (α_3) and H1-i-H3 (α_5) and the bond length i-H3 (b_3) are at their minimum energy values and the improper dihedral i-j-H1-H3 has a negative value.

$$\mathbf{r}_{H1} = \mathbf{r}_i + \mathbf{t}_1, \tag{11.14}$$

with

$$\mathbf{t}_1 = b_1 \left[\cos\alpha_1 \cdot \frac{\mathbf{r}_{ji}}{r_{ji}} - \sin\alpha_1 \cdot \frac{\mathbf{t}_a}{t_a} \right], \tag{11.15}$$

and with

$$\mathbf{t}_a = \frac{\mathbf{r}_{ji}}{r_{ji}} \times \left[\frac{\mathbf{r}_{kj}}{r_{kj}} \times \frac{\mathbf{r}_{ji}}{r_{ji}} \right]; \tag{11.16}$$

now for the second hydrogen atom

$$\mathbf{r}_{H2} = \mathbf{r}_i + \mathbf{t}_2, \tag{11.17}$$

with

$$\mathbf{t}_2 = c_1 \frac{\mathbf{r}_{ji}}{r_{ji}} + c_2 \frac{\mathbf{t}_a}{t_a} + c_3 \left[\frac{\mathbf{r}_{kj}}{r_{kj}} \times \frac{\mathbf{r}_{ji}}{r_{ji}} \right], \tag{11.18}$$

in which the scalar coeficients c_1, c_2 and c_3 are given by

$$c_1 = b_2 \cdot \cos\alpha_2 \tag{11.19}$$

$$c_2 = \frac{b_1 \cdot b_2 \cdot \cos\alpha_4 - c_1 \left(\frac{\mathbf{r}_{ji}}{r_{ji}} \cdot \mathbf{t}_1 \right)}{\frac{\mathbf{t}_a}{t_a} \cdot \mathbf{t}_1} \tag{11.20}$$

$$c_3 = \sqrt{b_2^2 - c_1^2 - c_2^2}. \tag{11.21}$$

finally, for the third hydrogen atom

$$\mathbf{r}_{H3} = \mathbf{r}_i + \mathbf{t}_3, \tag{11.22}$$

with

$$\mathbf{t}_3 = c_4 \frac{\mathbf{r}_{ji}}{r_{ji}} + c_5 \frac{\mathbf{t}_a}{t_a} + c_6 \left[\frac{\mathbf{r}_{kj}}{r_{kj}} \times \frac{\mathbf{r}_{ji}}{r_{ji}} \right], \tag{11.23}$$

in which the scalar coeficients c_4, c_5 and c_6 are given by

$$c_4 = b_3 \cdot \cos\alpha_3 \tag{11.24}$$

$$c_5 = \frac{b_1 \cdot b_3 \cdot \cos\alpha_5 - c_4\left(\frac{\mathbf{r}_{ji}}{r_{ji}} \cdot \mathbf{t}_1\right)}{\frac{\mathbf{t}_a}{t_a} \cdot \mathbf{t}_1} \tag{11.25}$$

$$c_6 = \sqrt{b_3^2 - c_4^2 - c_5^2}. \tag{11.26}$$

5. An atom (i) is bonded to one hydrogen atom (H) and three other heavy atoms (j, k, l). The coordinates of H are generated along the line going through atom i and a point corresponding to the average position of j, k and l, such that the bond length i-H (b) is at its minimum energy value and the angles j-i-H, k-i-H and l-i-H are larger than 90 degrees.

$$\mathbf{r}_H = \mathbf{r}_i - b\frac{\mathbf{t}_a}{t_a}, \tag{11.27}$$

with

$$\mathbf{t}_a = \mathbf{r}_{ji} + \mathbf{r}_{ki} + \mathbf{r}_{li} \tag{11.28}$$

6. An atom (i) is bonded to two hydrogen atoms (H1 and H2) and two other heavy atoms (j and k). The coordinates of H1 and H2 are placed above and below the plane going through atoms j, k and i, in such a way that the i-H1 (b_1) and i-H2 (b_2) bond lengths and the angle H1-i-H2 (α) are at their minimum energy values. The improper dihedral angle i-j-k-H1 will be positive.

$$\mathbf{r}_{H1} = \mathbf{r}_i + b_1\left[\sin\left(\frac{\alpha}{2}\right) \cdot \frac{\mathbf{t}_b}{t_b} + \cos\left(\frac{\alpha}{2}\right) \cdot \frac{\mathbf{t}_a}{t_a}\right], \tag{11.29}$$

and

$$\mathbf{r}_{H2} = \mathbf{r}_i - b_2\left[\sin\left(\frac{\alpha}{2}\right) \cdot \frac{\mathbf{t}_b}{t_b} + \cos\left(\frac{\alpha}{2}\right) \cdot \frac{\mathbf{t}_a}{t_a}\right], \tag{11.30}$$

with

$$\mathbf{t}_a = -(\mathbf{r}_{ji} + \mathbf{r}_{ki}), \tag{11.31}$$

and

$$\mathbf{t}_b = \mathbf{r}_{ji} \times \mathbf{r}_{ki}. \tag{11.32}$$

7. An atom (i) 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 i-H1 aligned in the z-direction and the i-H2 in the z-y plane with the angle H1-i-H2 (α) and bond lengths i-H1 (b_1) and i-H2 (b_2) according to their minimum energy values. This molecule is then rotated around x, y and z by three random angles.

$$\mathbf{r}_{H1} = \mathbf{r}_i + \mathbf{R}\mathbf{t}_1, \tag{11.33}$$

$$\mathbf{r}_{H2} = \mathbf{r}_i + \mathbf{R}\mathbf{t}_2, \tag{11.34}$$

in which $\mathbf{t}_1$ has the coordinates $(0.0, 0.0, b_1)$ and $\mathbf{t}_2$ has the coordinates $(0.0, b_2 \sin\alpha, b_2 \cos\alpha)$ and $\mathbf{R}$ is a matrix that corresponds to rotations around x, y and z by three random angles (ϕ, ψ and θ). $\mathbf{R}$ can be written as:

$$\mathbf{R} = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & \cos\phi & -\sin\phi \\ 0.0 & \sin\phi & \cos\phi \end{pmatrix} \times \begin{pmatrix} \cos\psi & 0.0 & \sin\psi \\ 0.0 & 1.0 & 0.0 \\ -\sin\psi & 0.0 & \cos\psi \end{pmatrix} \times \begin{pmatrix} \cos\theta & -\sin\theta & 0.0 \\ \sin\theta & \cos\theta & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}. \tag{11.35}$$

8. An atom (i) is bonded to four hydrogen atoms (H1, H2, H3 and H4), but to no heavy atoms. A molecule is generated with all bond lengths (b_1, b_2, b_3 and b_4) at their minimum energy value, the i-H1 aligned in the z-direction, H2 in the x-z plane and H3 such that the improper i-H1-H2-H3 is positive and H4 such that the improper i-H1-H2-H4 is negative. In addition, all H_n-i-H_m angles (α) are set to 109.5°. The complete molecule is then rotated by three random angles around x, y and z. Here we also make use of the rotational matrix ($\mathbf{R}$) defined by Eq. 11.35. Similarly to the above case, we have

$$\mathbf{r}_{H1} = \mathbf{r}_i + \mathbf{R}\mathbf{t}_1, \tag{11.36}$$

$$\mathbf{r}_{H2} = \mathbf{r}_i + \mathbf{R}\mathbf{t}_2, \tag{11.37}$$

$$\mathbf{r}_{H3} = \mathbf{r}_i + \mathbf{R}\mathbf{t}_3, \tag{11.38}$$

$$\mathbf{r}_{H4} = \mathbf{r}_i + \mathbf{R}\mathbf{t}_4, \tag{11.39}$$

in which $\mathbf{t}_1$ has the coordinates $(0.0, 0.0, b_1)$, $\mathbf{t}_2$ has the coordinates $(b_2 \sin\alpha,\ 0.0,\ b_2 \cos\alpha)$, $\mathbf{t}_3$ has the coordinates $(b_3 \sin\alpha \cos 120°,\ b_3 \sin\alpha \sin 120°,\ b_3 \cos\alpha)$, and $\mathbf{t}_4$ has the coordinates $(b_4 \sin\alpha \cos 240°,\ b_4 \sin\alpha \sin 240°,\ b_4 \cos\alpha)$.

CHAPTER 12

Generation of Atomic Velocities

The *atomic velocities* $\mathbf{v}_j$ of a molecular system in equilibrium will obey a *Maxwell distribution* at a given temperature T, that is, the probability that the atomic velocity lies between $\mathbf{v}_j$ and $\mathbf{v}_j + \mathbf{dv}_j$ is

$$P(\mathbf{v}_j)\mathbf{dv}_j \quad = \quad [2\pi k_B T/m_j]^{-\frac{3}{2}} \exp[m_j \mathbf{v}_j^2/(2k_B T)]\mathbf{dv}_j \tag{12.1}$$

where k_B is Boltzmann's constant and m_j the mass of atom j. If no constraints are applied, the 3D velocity distribution has the form of a product of three Gaussian distributions Eq. 4.1,

$$\mathbf{p}(x)dx \quad = \quad [2\pi\sigma^2]^{-\frac{1}{2}} \exp[-(x- < x >)^2/(2\sigma^2)]dx \tag{12.2}$$

for the Cartesian velocity components $\mathbf{v}_{jx}$, $\mathbf{v}_{jy}$ and $\mathbf{v}_{jz}$, each with $< x > = 0$ and Eq. 2-12.58

$$\sigma = [k_B\, T/\, m_j\,]^{\frac{1}{2}}\,. \tag{12.3}$$

If constraints are applied, the velocity components that will induce a violation of the constraints, have to be eliminated. For solute or solvent distance constraints this is done using the procedure to obtain so-called shaken or constrained velocities (Sec. 2-10.3.7). For position constrained or fixed atoms the velocities are simply set equal to zero (Sec. 2-10.2). These operations imply a modification of the sampled unconstrained ($\mathbf{v}_j^{uc}$) velocity distribution to a constrained ($\mathbf{v}_j$) velocity distribution, which may involve a change of properties. For example, the solute or solvent temperatures as calculated from the velocities via (Eq. 2-10.49 and Eq. 2-10.50) or (Eq. 2-10.52 and Eq. 2-10.53), before or after shaking may be different.

$$T(\mathbf{v}_j^{uc}) \neq T(\mathbf{v}_j). \tag{12.4}$$

The constrained velocity distribution can be brought to the desired temperature T by coupling to a temperature bath in a simulation (Sec. 2-12.2).

CHAPTER 13

What to Do when SHAKE Fails

When something goes wrong in a simulation that involves constraints handled by the SHAKE method, it often shows up as a SHAKE error. This means that the atomic coordinate resetting, i.e. from the unconstrained atomic positions $\mathbf{r}^{uc}$ to the constrained atomic positions $\mathbf{r}$ in Eq. 2-10.14, cannot be accomplished within the limit of $N_{sh} = 1000$ iterations over the solute or solvent constraints, or it cannot be accomplished since the deviation between ${\mathbf{r}^{uc}}_{k_2}$ and $\mathbf{r}_{k_2}$ is or has become too large, as illustrated in Fig. 2-10.2. This situation can easily occur when something is wrong with

1. the constraint lengths ${d^0}_{k_1 k_2}$ in Eq. 2-10.11,
2. the reference atomic positions $\mathbf{r}_{k_1 k_2}(t)$ in Eq. 2-10.11,
3. the unconstrained atomic positions ${\mathbf{r}^{uc}}_{k_1 k_2}(t + \Delta t)$ in Eq. 2-10.11,
4. the constrained atomic velocities $\mathbf{v}_i(t - \Delta t/2)$ determining ${\mathbf{r}^{uc}}_i(t + \Delta t)$ through Eq. 2-10.6,
5. the unconstrained atomic forces ${\mathbf{f}^{uc}}_i(t)$ determining ${\mathbf{r}^{uc}}_i(t + \Delta t)$ through Eq. 2-10.6.

So, any anomalously large atomic force or arbitrary modification of the quoted quantities may induce SHAKE to fail. However, due to this sensitivity of SHAKE to incorrect forces, velocities or coordinates, a failure of SHAKE often signals an error for which the cause must be sought elsewhere. Here, we list a few *possible causes of errors showing up in SHAKE*, and what could be done *to identify their cause*.

1. The *length unit* in the *molecular topology* does not match that of the *atomic coordinates*, e.g. nm versus Å. This will result in the energies of the bonds being much too large, see Sec. 2-10.3.
2. The *sequence of the atoms* in the *molecular topology* does not match that of the *atomic coordinates*. This error usually shows up in the bond-angle energies.
3. The *chirality* of the *atomic coordinates* does not correspond to the definition of the *improper dihedral angles* in the *molecular topology* (building blocks). This error will show up in the improper dihedral angle energies.
4. The (initial) *molecular configuration* (atomic coordinates) has a *very high energy* in terms of the force-field used. This error will show up in the output of an energy minimization without constraints: the energies in the zero-th EM step will be large.
5. During a simulation, the *forces* may become *too large*, for example, when positively and negatively charged atoms come too close to each other, SHAKE may not be able to maintain the bonds to these atoms. This error will show up in the output of an MD run without constraints: the energy of the bonds will become large.
6. During a simulation, some forces may act largely perpendicular to an *extended*, *constrained planar group of atoms*, for example the side chain atoms of Arg residues. In such a situation SHAKE is not very efficient, since it attempts to compensate the unconstrained step which is *perpendicular* to the plane of the planar group by modifying iteratively the atomic coordinates *within* the plane of the planar group, see Fig. 13.1. This error shows up in the SHAKE error message. The coordinates of the atoms of the planar group or nearby atoms cannot be reset. Use of a smaller time step Δt (so that the positional changes per step are smaller) or switching off (part of) the solute constraints (Sec. 2-10.4) may help to overcome this situation.

We note that the contribution of the various terms in the force field to the (in)stability of a simulation can be analyzed by switching on and off the various force-field terms. The switches NTF[1..10] (forces), NTPOR

(position restraining), NTDIR (distance restraining), NTDLR (dihedral angle restraining), NTJVR (^3J-value restraining) and NTLES (local elevation biasing) can be used to this end (Secs. 2-12.7 and 2-10.3). The solute constraints can be switched on and off using the switch NTC (Sec. 2-10.4), whereas the solvent constraints cannot be switched off (Sec. 2-10.5).

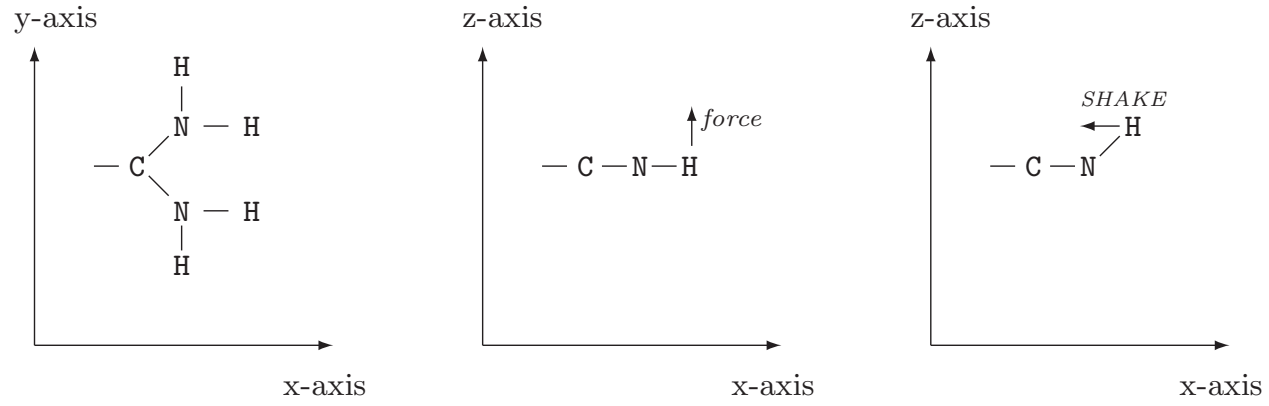

FIGURE 13.1. The inefficiency of SHAKE when forces perpendicular to constrained, extended planar groups of atoms, e.g. in an Arg side chain (left panel), are present. The force (in the z-direction) induces a change of the H atom position out of the (x, y) plane (middle panel), which SHAKE attempts to compensate for by coordinate modification within the (x, y) plane (right panel).

CHAPTER 14

Removal of Centre of Mass Motion

When simulating a system in vacuo, the total translational momentum and the total angular momentum are conserved quantities. When periodic boundary conditions are applied the total angular momentum is not conserved, but the total translational momentum still is. When parts of the system are positionally restrained (e.g. by position restraining, Sec. 2-9.2) neither of these quantities are conserved. In the case of in vacuo or periodic boundary condition simulations, it is common practice to stop the translational motion of the centre of mass and the rotational motion around the centre of mass of the entire molecular system at the start of such a simulation (NTICOM = 1 see Chap. 4-8 block *INITIALISE* for more details) and at regular time intervals afterwards (NSCM $\approx 10^4$) in order to counteract a numerical build-up of centre of mass motion (Sec. 2-4.2, Sec. 2-4.4, and Sec. 2-12.7). Here we present the *algorithm* for*removal of motion of and around the system centre of mass*. It consists of the following steps.

1. Determine the *centre of mass coordinates*

$$\mathbf{r}_{cm} \;=\; M^{-1}\sum_{i=1}^{N} m_i\,\mathbf{r}_i \tag{14.1}$$

of the molecular *system* containing N atoms with masses m_i and positions $\mathbf{r}_i$, where

$$M \;=\; \sum_{i=1}^{N} m_i\,. \tag{14.2}$$

2. Determine the *coordinates* of the atoms *relative* to the system *centre of mass*

$$\mathbf{r'}_i \;=\; \mathbf{r}_i - \mathbf{r}_{cm}\,. \tag{14.3}$$

3. Determine the *system centre of mass velocity*

$$\mathbf{v}_{cm} \;=\; M^{-1}\sum_{i=1}^{N} m_i\,\mathbf{v}_i, \tag{14.4}$$

and calculate the system centre of mass translational kinetic energy

$$\mathcal{K}_{cm,tr} \;=\; \tfrac{1}{2} M\,\mathbf{v}_{cm}^2\,. \tag{14.5}$$

4. Determine the *velocities* of the atoms *relative* to the system *centre of mass translation*

$$\mathbf{v'}_i \;=\; \mathbf{v}_i - \mathbf{v}_{cm}\,. \tag{14.6}$$

5. Determine the system *angular momentum around* the system *centre of mass*

$$\mathbf{L}_{cm} \;=\; \sum_{i=1}^{N} m_i\,\mathbf{r'}_i \times \mathbf{v'}_i\,. \tag{14.7}$$

6. Determine the *inertia tensor* of the system with respect to the centre of mass

$$\underline{\mathbf{I}}_{cm} = \sum_{i=1}^{N} m_i \begin{pmatrix} (r'_i)^2 - (x'_i)^2 & -x'_i\,y'_i & -x'_i\,z'_i \\ -y'_i\,x'_i & (r'_i)^2 - (y'_i)^2 & -y'_i\,z'_i \\ -z'_i\,x'_i & -z'_i\,y'_i & (r'_i)^2 - (z'_i)^2 \end{pmatrix}. \tag{14.8}$$

7. Determine the *angular velocity around* the system *centre of mass* using the inverted inertia tensor

$$\mathbf{O}_{cm} \;=\; \underline{\mathbf{I}}_{cm}^{-1}\,\mathbf{L}_{cm}, \tag{14.9}$$

and calculate the rotational kinetic energy around the system centre of mass

$$\mathcal{K}_{cm,rot} \;=\; \tfrac{1}{2}\,\mathbf{O}_{cm}\cdot\mathbf{L}_{cm} \tag{14.10}$$

8. Determine the *velocities* of the atoms *relative* to the system *centre of mass translation* and to its *rotation* around its centre of mass

$$\mathbf{v''}_i \quad = \quad \mathbf{v'}_i - \mathbf{O}_{cm} \times \mathbf{r'}_i \,. \tag{14.11}$$

CHAPTER 15

Saving Trajectories

The molecular information which is generated by a simulation can be stored in different ways in order to enable the user to analyze the various molecular or system properties after completion of the simulation using specialized analysis programs or so-called post-MD programs (see Vol. 5). Saving all information that is generated in a simulation, i.e. atomic coordinates, velocities, forces, pair energies, etc. at each time point $t_{n'}$ would require an excessive amount of disc space. Moreover, the information stored would be redundant, since the values of the mentioned quantities are highly correlated between subsequent simulation time points t_n, $t_{n+1} = t_n + \Delta t$, etc., since Δt is generally small ≈ 0.002 ps. The information may also be redundant due to dependence between different physical quantities. For example, molecular potential energies and atomic forces can be (re)calculated from atomic coordinates (if no velocity dependent forces are present).

The amount and type of molecular or system information that can be stored during a simulation can be controlled in program MD++ within the WRITETRAJ input block (described in Chap. 4-8) in the following manner:

1. In every simulation the final configuration and velocities and other quantities that are needed to continue the simulation are saved (Sec. 5-1.1) in a so-called *single-configuration file*. These configurations, which generally lie far apart in time, typically 10-100 ps, can be used to obtain a low time-resolution picture of the properties of the molecular system.

2. During a simulation different groups of atomic or system quantities can be saved at different time intervals in different *trajectory files*.
   a. *Atomic coordinates* (Sec. 4-4.2) and possibly corresponding energies (Sec. 4-4.17) and time step data (Sec. 4-4.16) in a *coordinates trajectory file*. The switches NTWX and NTWSE control the saving of configurations:
      (i) If $NTWX \neq 0$ and $NTWSE = 0$, atomic coordinates (POSITION block) are saved at *constant time intervals* $|NTWX| \times \Delta t$, where Δt is the simulation time step. If $NTWX > 0$, *solute plus solvent* atomic coordinates are saved, whereas if $NTWX < 0$, *only solute* atomic coordinates are saved.
      (ii) If $NTWX \neq 0$ and $NTWSE > 0$, the *lowest energy* configuration of each sequential block of |NTWX| simulation time points is saved, while the value of $NTWSE$ specify the type of energy used for the selection, $ENER[NTWSE]$ (refer to Sec. 4-4.17 for the description of the energy types). In this case, the *time intervals* between saved configurations are *variable*, minimally Δt and maximally $|NTWX| \times \Delta t$, so the time and step number and the energy are saved together with the configuration (POSITION block, TIMESTEP block, ENERGY03 block). If $NTWX > 0$, *solute plus solvent* atomic coordinates are saved, whereas if $NTWX < 0$, *only solute* atomic coordinates are saved.
   b. *Atomic velocities* (Sec. 4-4.3) in a *velocity trajectory file*. The switch NTWV controls the saving of velocities. If $NTWV \neq 0$, atomic velocities (VELOCITY block) are saved at *constant time intervals* $|NTWV| \times \Delta t$. If $NTWV > 0$, *solute plus solvent* atomic velocities are saved, whereas if $NTWV < 0$, *only solute* atomic velocities are saved.
   c. *Energies, temperature scaling factors, virial, pressure and computational box size* in a so-called *energy trajectory file*. The switch NTWE controls the saving of these data. If $NTWE \neq 0$, the ENERGY03 block and the VOLUMEPRESSURE03 block, as described in Sec. 4-4.17, are saved at *constant time intervals* $|NTWE| \times \Delta t$.
   d. Data to compute relative *free energies* with respect to changing the coupling parameter λ are saved in a so-called *free-energy trajectory file*. The switch NTWG controls the saving of free

energy data. If $NTWG \neq 0$, the FREEENERDERIVS03 block is saved at *constant time intervals* $|NTWG| \times \Delta t$.

e. *Forces* are saved to a *force trajectory file*. If $NTWF \neq 0$, the FORCE block is saved at *constant time intervals* $|NTWF| \times \Delta t$. If $NTWF > 0$, *solute plus solvent* atomic forces are saved, whereas if $NTWF < 0$, *only solute* atomic forces are saved.

f. *Block-averaged energies* are written to a *block average energies file*. If $NTWB \neq 0$, the block averaged energies (and free energies if $NTWG > 0$) are saved at *constant time intervals* $|NTWB| \times \Delta t$.

g. A *special trajectory file* is written for some specific applications (*e.g.* polarisation, NMR data, X-ray data, ...). The writting of this trajectory is not controlled by the WRITETRAJ input block, but within the specific input blocks. For example, the POLARISE block in MD++ contains a $WRITE$ flag which determines the frequency with which the distances Δr between the charges (of the charge on spring model, see Sec. 2-7.5) will be written to the special trajectory (see Chap. 4-8).

When *choosing* the *time interval* between *saved configurations*, velocities, energies or free energy data, the following points should be considered.

1. Sufficient data points (*i.e.* > 1000) should be available to secure sufficient precision of averages, fluctuations.
2. The larger the accessible part of phase space of the molecular system, the more configurations, etc. may be needed for satisfactory averaging.
3. The longer the relaxation time of the property of interest, the more time points should be saved.
4. The wider the time scales that determine a molecular or system property of interest, the more dense the time points for saving configurations or velocities should be chosen.

For example, configurations are typically stored every 0.1-10 ps. In simulations of molecular liquids comprising a few hundred identical molecules, simulation periods of 10-100 ps are sufficient to obtain precise values for quantities, such as the density, heat of vaporization, diffusion coefficient, rotational correlation times, thermal expansion coefficient, isothermal compressibility, specific heat, excess free energy[5,6]. *Dielectric properties* require much longer (> 1 nsec) simulations in which long-range electrostatic interactions are taken into account. Calculation of the shear *viscosity* requires saving the pressure at every time step for more than a nanosecond[5,6].

GROMOS++ analysis programs usually require one or more trajectory files as input. This is usually performed with specifying the flag @traj (although, other types are also possible: @en_files, @fr_files, ...). See Vol. 5 for more details.

If necessary, GROMOS++ program *tstrip* can be used to remove solvent coordinates from the trajectory file (Sec. 5-4.64). In addition, GROMOS++ program *filter* can be used to filter a coordinate trajectory for a limited set of atoms.

CHAPTER 16

Performing a Translational Superposition and a Rotational Least-Squares Fit

When analyzing or averaging quantities that depend on the atomic position vectors $\mathbf{r}_i(t)$ of the atoms of a (solute) molecule generated in a simulation, it is often desired to separate the internal motions or fluctuations from the centre of mass translation of the molecule and the rotation around the centre of mass of the molecule. The translation of and the rotation around the centre of mass of a molecule can be eliminated from the configurations of a trajectory by superimposing the centres of mass of the sequential configurations and subsequently performing a rotational least-squares fit of the positions of corresponding atoms in the configurations of the trajectory and in the first configuration.

Elimination of the translational centre of mass motion from a trajectory of solute configurations $\mathbf{r}_i(t_n)$ with $n = 1, 2, \ldots, \mathcal{N}_a^{solu}$ is achieved by conversion of the atomic coordinates to atomic coordinates relative to the solute centre of mass for each time frame $t_{n'}$,

$$\mathbf{r}'_i(t_n) = \mathbf{r}_i(t_n) - \mathbf{r}_{COM}(t_n) \tag{16.1}$$

with

$$\mathbf{r}_{COM}(t_n) = \frac{1}{m_{solute}} \sum_{i=1}^{\mathcal{N}_a^{solu}} m_i \mathbf{r}_i(t_n) \tag{16.2}$$

and

$$m_{solute} = \sum_{i=1}^{\mathcal{N}_a^{solu}} m_i \quad . \tag{16.3}$$

Eq. 16.2 is calculated in GROMOS++ using the class `fit::PositionUtils`.

Elimination of the rotational motion of the solute around its centre of mass is achieved by performing a least-squares fit of the position vectors $\mathbf{r}'_i(t_n)$ and $\mathbf{r}'_i(t_m)$ of two different configurations at times t_n and t_m. The atoms $i = 1, 2, \ldots, \mathcal{N}$ for which the superposition is to be carried out, should be chosen from the relatively rigid parts of the solute in order to minimize the effect of internal molecular deformations on the rotational fit. For example, one may use all solute atoms, or only the atoms bearing the name CA or the atoms specified in a list for the rotational fit. The problem is to find an orthonormal 3x3 matrix $\underline{Q}$ which represents a solute rotation

$$\mathbf{r}''_i(t_n) = \underline{Q}\mathbf{r}'_i(t_n) \quad , \tag{16.4}$$

and which minimizes the function

$$\begin{aligned} E(\underline{Q}) &= \sum_{i=1}^{\mathcal{N}} w_i \left[\underline{Q}\mathbf{r}'_i(t_n) - \mathbf{r}'_i(t_m)\right]^2 \\ &= \sum_{i=1}^{\mathcal{N}} w_i \left[\mathbf{r}''_i(t_n) - \mathbf{r}'_i(t_m)\right]^2 \end{aligned} \tag{16.5}$$

where the w_i are weights given to the atoms of the solute. All w_i are zero, except for the atoms for which the rotational least-squares fit is to be performed.

Two procedures to obtain $\underline{Q}$ have been implemented in GROMOS++, proposed by McLachlan[7] and Kabsch[8].

1. The *method of McLachlan*, J. Mol.Biol. **128** (1979) 74-77 in subroutine LSQSTR, file `lsqstr.f`.
2. The *method of Kabsch*, Acta Cryst. **A32** (1976) 922-933 in subroutine LSQSTR, file `lsqstrk.f`.

Given the reference coordinates $\mathbf{r}'_i(t_m)$, the coordinates $\mathbf{r}'_i(t_n)$ are rotated around the origin such that $\mathbf{r}''_i(t_n)$ of Eq. 16.4 are obtained and the function Eq. 16.5 is minimal.

Elimination of solute translation and rotation can be selected in a number of analysis programs (frameout, nhoparam, rmsd, rmsdmat, rmsf and solute_entropy, see Vol. 5) and using the input parameter flags `@atomsfit`.

CHAPTER 17

Transformation between Coordinates

17.1. Cartesian and Oblique Contravariant Crystallographic Coordinates

The result of a crystal structure determination of a molecule using X-ray or neutron diffraction is an electron density distribution function $\rho(\mathbf{r}')$, the number of electrons per unit volume at position $\mathbf{r}'$. It is derived from the measured diffracted beam intensities $I_{obs}(\mathbf{h})$, which are proportional to the square of the amplitude of the crystallographic structure factor

$$F(\mathbf{h}) = \int\limits_{V_c} \rho(\mathbf{r}')\, e^{+2\pi i \mathbf{h}\cdot\mathbf{r}'}\, \mathbf{dr}' \quad , \tag{17.1}$$

i.e. the 3-dimensional Fourier transform of $\rho(\mathbf{r}')$ over a crystal unit cell with volume V_c. So, we have

$$I_{obs}(\mathbf{h}) \alpha |F(\mathbf{h})|^2 \quad . \tag{17.2}$$

The electron density $\rho(\mathbf{r}')$ could be derived from the structure factors $F(\mathbf{h})$ through the inverse of Eq. 17.1,

$$\rho(\mathbf{r}') = V_c^{-1} \sum_h \sum_k \sum_l F(\mathbf{h})\, e^{-2\pi i(hx'+ky'+lz')} \quad , \tag{17.3}$$

where $\mathbf{h}$ is the reciprocal space vector[9]. Since only the amplitudes $|F(\mathbf{h})|$ of the complex structure factors $F(\mathbf{h})$ can be obtained from experiment, it is common practice to postulate an analytical form for the function $\rho(\mathbf{r}')$.

In *isotropic crystallographic refinement* it is assumed that the electron density is distributed as an isotropic Gaussian function around the positions $\mathbf{r}'_j$ of the atoms,

$$P_j(\mathbf{r}') = \left[(2\pi)^{3/2} u_j^3\right]^{-1} e^{-(x'^2+y'^2+z'^2)/(2\,u_j^2)} \quad . \tag{17.4}$$

Fourier transforming Eq. 17.4 yields again a Gaussian distribution in $\mathbf{h}$ space

$$\begin{aligned} \hat{P}_j(\mathbf{h}) &= e^{-2\pi^2 u_j^2 (h^2+k^2+l^2)} \\ &= e^{-2\pi^2 u_j^2 (2\sin\theta/\lambda)^2} \\ &= e^{-B_j (\sin\theta/\lambda)^2} \quad , \end{aligned} \tag{17.5}$$

where we have expressed it in terms of the isotropic atomic B-factor or temperature factor

$$B_j = 8\,\pi^2\, u_j^2 \quad , \tag{17.6}$$

the diffracted beam scattering angle θ and wave length λ.

In *anisotropic crystallographic refinement*, the atomic electron density distribution is assumed to be an ellipsoid Gaussian, so

$$\begin{aligned} \hat{P}_j(\mathbf{h}) &= e^{-(b_{11}h^2+b_{22}k^2+b_{33}l^2+b_{12}hk+b_{13}hl+b_{23}kl)} \\ &= e^{-2\pi^2[U'_{11}(ha^*)^2+(U'_{12}+U'_{21})(ha^*kb^*)+\ldots]} \end{aligned} \tag{17.7}$$

in $\mathbf{h}$-space. The anisotropic temperature factors are usually given in the form of the symmetric 3×3 matrix $\underline{b}$ or $\underline{U}'$, where the prime in the latter is used to indicate that a crystallographic, possibly oblique coordinate system has been used in Eq. 17.7. According to crystallographic convention the real space lengths of the edges of the unit cell are indicated by a, b and c and the corresponding lengths of the reciprocal lattice cell by a^*, b^* and c^*.

Since Newton's equations of motion are not valid in oblique coordinates, simulations of molecular crystal unit cells are performed using (orthogonal) Cartesian coordinates. When the crystallographically refined atomic coordinates $\mathbf{r}'_j$ and anisotropic temperature factors $\underline{U}'_j$ are given in an oblique crystallographic coordinate system, indicated by the prime on the symbols, the quantities have to be transformed to an orthogonal (Cartesian) coordinate system. This can be done in the following way:

The *standard orthogonal coordinate system* with coordinates indicated by x, y and z is defined by

1. The x-axis is the projection of the crystallographic x'-axis on the plane orthogonal to the crystallographic y'-axis.
2. The y-axis coincides with the crystallographic y′-axis.
3. The z-axis is orthogonal to the x-axis and y-axis and defined as in a right-handed Cartesian system.

The *transformation* of the *atomic coordinates* (x', y', z') in the *oblique crystallographic* coordinate system to the atomic coordinates (x, y, z) in the *orthogonal (Cartesian)* coordinate system then reads

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} T_{11} & T_{12} & T_{13} \\ T_{21} & T_{22} & T_{23} \\ T_{31} & T_{32} & T_{33} \end{pmatrix} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \quad , \tag{17.8}$$

with

$$\begin{aligned} T_{11} &= \sin\gamma \\ T_{22} &= 1 \\ T_{33} &= [\sin^2\alpha - ((\cos\beta - \cos\alpha\cos\gamma)/\sin\gamma)^2]^{\frac{1}{2}} \\ T_{13} &= (\cos\beta - \cos\alpha\cos\gamma)/\sin\gamma \\ T_{21} &= \cos\gamma \\ T_{23} &= \cos\alpha \\ T_{12} &= T_{31} = T_{32} = 0 \end{aligned} \quad , \tag{17.9}$$

where α is the angle between the positive y'- and z'-axes, β is the angle between the positive z'- and x'-axes and γ is the angle between the positive x'- and y'-axes.

The *transformation* of the matrix $\underline{U}'$ of *atomic temperature factors* in the *oblique crystallographic* coordinate system to the matrix $\underline{U}$ of temperature factors in the *(orthogonal) Cartesian* system reads

$$\underline{U} = (\underline{T}\,\underline{D})\;\underline{U'}\;(\underline{T}\,\underline{D})^{\tau} \tag{17.10}$$

where

$$\underline{D} = \begin{pmatrix} d^{-1}\sin\alpha & 0 & 0 \\ 0 & d^{-1}\sin\beta & 0 \\ 0 & 0 & d^{-1}\sin\gamma \end{pmatrix} \quad , \tag{17.11}$$

with

$$d = [1 - \cos^2\alpha - \cos^2\beta - \cos^2\gamma + 2\cos\alpha\,\cos\beta\,\cos\gamma]^{\frac{1}{2}} \quad , \tag{17.12}$$

and the transpose of a matrix is indicated by the superscript τ. So we find

$$\begin{aligned} (\underline{T}\,\underline{D})_{11} &= d^{-1}\sin\alpha\sin\gamma \\ (\underline{T}\,\underline{D})_{22} &= d^{-1}\sin\beta \\ (\underline{T}\,\underline{D})_{33} &= d^{-1}\sin\gamma[\sin^2\alpha - ((\cos\beta - \cos\alpha\cos\gamma)/\sin\gamma)^2]^{\frac{1}{2}} \\ (\underline{T}\,\underline{D})_{13} &= d^{-1}(\cos\beta - \cos\alpha\cos\gamma) \\ (\underline{T}\,\underline{D})_{21} &= d^{-1}\sin\alpha\cos\gamma \\ (\underline{T}\,\underline{D})_{23} &= d^{-1}\cos\alpha\sin\gamma \\ (\underline{T}\,\underline{D})_{12} &= (\underline{T}\,\underline{D})_{31} = (\underline{T}\,\underline{D})_{32} = 0 \quad . \end{aligned} \tag{17.13}$$

For a *monoclinic crystallographic unit cell* (2^{nd} setting, y'-axis unique) we have $\alpha = \gamma = 90^o$, so $d = \sin\beta$ and

$$\underline{T} = \begin{pmatrix} 1 & 0 & \cos\beta \\ 0 & 1 & 0 \\ 0 & 0 & \sin\beta \end{pmatrix} \tag{17.14}$$

and

$$\underline{T}^{-1} = \begin{pmatrix} 1 & 0 & -\cot\beta \\ 0 & 1 & 0 \\ 0 & 0 & 1/\sin\beta \end{pmatrix} \tag{17.15}$$

and

$$\underline{TD} = \begin{pmatrix} 1/\sin\beta & 0 & \cot\beta \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad . \tag{17.16}$$

CHAPTER 18

Distributions, Averages and Root-Mean-Square Fluctuations

The ensemble or trajectory that is produced in a simulation of a molecular system can be analyzed in a variety of ways.

1. Different *scalar quantities* Q , such as energies, atom-atom distances, ^3J-coupling constants, internal coordinates, etc., or *vector quantities* $\vec{Q}$, such as atomic positions, velocities, molecular dipole moments, etc. can be considered.

2. The static equilibrium properties of quantities Q or $\vec{Q}$ can be analyzed in terms of their *probability distribution* or *frequencies of occurrence* of the various Q or $\vec{Q}$ values, $P(Q)$ or $P(\vec{Q})$, in a trajectory.

3. The probability distributions $P(Q)$ or $P(\vec{Q})$ can also be characterized by calculating the *various moments* (p = 1,2,3...)

$$< Q^p > \equiv \frac{1}{\mathcal{N}_{tot}} \sum_{t=1}^{\mathcal{N}_{tot}} [Q(t)]^p \tag{18.1}$$

or

$$< \vec{Q}^p > \equiv \frac{1}{\mathcal{N}_{tot}} \sum_{t=1}^{\mathcal{N}_{tot}} [\vec{Q}(t)]^p \tag{18.2}$$

of the distributions. The $\mathcal{N}_{tot}$ being the total number of data points (time frames) of the trajectory, and the product of two vectors is to be interpreted as their scalar or dot product. From these moments of the distributions the *mean* or *average*

$$< Q > \equiv \frac{1}{\mathcal{N}_{tot}} \sum_{t=1}^{\mathcal{N}_{tot}} Q(t) \tag{18.3}$$

and

$$< \vec{Q} > \equiv \frac{1}{\mathcal{N}_{tot}} \sum_{t=1}^{\mathcal{N}_{tot}} \vec{Q}(t) \tag{18.4}$$

can be determined. If we assume that the instantaneous value of a given property Q is statistically independent of the other values in the time series of this property, then the variance can be simply taken as

$$\sigma^2(Q) = \frac{1}{\mathcal{N}_{tot}} \sum_{t=1}^{\mathcal{N}_{tot}} (Q(t) - < Q >_{\mathcal{N}_{tot}})^2 \tag{18.5}$$

where $< Q >_{\mathcal{N}_{tot}}$ represents an average of $Q(t)$ over the entire run.

If the configurations are correlated, the block averaging method should be used. The *time series* $Q(t)$ is divided into blocks of length t_b, the average of each block being

$$< Q >_b = \frac{1}{t_b} \sum_{t=1}^{t_b} Q(t) \tag{18.6}$$

The variance can then be estimated with the average values for all blocks of this kind as

$$\sigma^2(< Q >_b) = \frac{1}{n_b} \sum_{b=1}^{n_b} (< Q >_b - < Q >_{\mathcal{N}_{tot}})^2 \tag{18.7}$$

where n_b is the number of blocks.

The quantity $\sigma^2(< Q >_b)$ is expected to be inversely proportional to the block size (t_b), for increasing values of t_b. The goal is now to find the proportionality constant that will allow for an estimation of $\sigma^2(< Q >_b)$ for the single large block that characterizes the entire trajectory ($t_b = \mathcal{N}_{tot}$), for which the statistical inefficiency $\mathcal{S}_{inef}$ is defined as

$$\mathcal{S}_{inef} = \lim_{t_b \to \infty} \frac{t_b \sigma^2(< Q >_b)}{\sigma^2(Q)} \tag{18.8}$$

with the error estimation given by

$$\sigma(< Q >_{\mathcal{N}_{tot}}) = \sqrt{\frac{\mathcal{S}_{inef}}{\mathcal{N}_{tot}}} \times \mathrm{RMSD}(Q) \tag{18.9}$$

where RMSD(Q) is the root mean square deviation of Q . This blocking averaging method is used by the GROMOS++ program ene_ana.

The *root-mean-square fluctuations* can be calculated as

$$\begin{aligned} \Delta Q &\equiv \sqrt{< [Q - < Q >]^2 >} \\ &= \sqrt{\frac{1}{\mathcal{N}_{tot}} \sum_{t=1}^{\mathcal{N}_{tot}} [Q(t) - < Q >]^2} \\ &= \sqrt{\frac{1}{\mathcal{N}_{tot}} \sum_{t=1}^{\mathcal{N}_{tot}} [Q(t)]^2 - < Q >^2} \end{aligned} \tag{18.10}$$

or for the vector quantity $\vec{Q}$

$$\begin{aligned} \Delta \vec{Q} &\equiv \sqrt{< [\vec{Q} - < \vec{Q} >]^2 >} \\ &= \sqrt{\frac{1}{\mathcal{N}_{tot}} \sum_{t=1}^{\mathcal{N}_{tot}} [\vec{Q}(t) - < \vec{Q} >]^2} \\ &= \sqrt{\frac{1}{\mathcal{N}_{tot}} \sum_{t=1}^{\mathcal{N}_{tot}} [\vec{Q}(t)]^2 - < \vec{Q} >^2} \end{aligned} \tag{18.11}$$

When the quantity $\vec{Q}$ is an atomic position, one finds the following relation between the 3-dimensional mean square displacement of an atom j,

$$\begin{aligned} (\Delta \mathbf{r}_j)^2 &= < [\mathbf{r}_j - < \mathbf{r}_j >]^2 > \\ &= 3u_j^2 \\ &= 3B_j/(8\pi^2) \end{aligned} \tag{18.12}$$

and the isotropic atomic B-factor or temperature factor B as described in Chap. 17.

4. The *dynamic properties* of the quantities Q or $\vec{Q}$ can be analyzed in terms of *time series* and *time correlation functions* using the GROMOS++ programs tser and tcf, respectively.

CHAPTER 19

Dihedral-Angle Conventions, Names and Transitions

In the literature *different conventions* for the definition of the value and sign of a *dihedral angle* φ(i-j-k-l) defined by the planes through atoms i, j and k and through j, k and l are in use, see Fig. 19.1

1. *IUPAC-IUB convention*[3].
   The dihedral angle φ_I is considered positive or negative according as, when the system is viewed along the central bond j-k in the direction from j to k (or k to j), the bond i-j to the front atom j (or the bond l-k to the front atom k) requires rotation to the right or to the left, respectively, in order that it may eclipse the bond l-k to the rear atom k (or the bond i-j to the rear atom j).
2. *Polymer convention.*
   The *trans* conformation has $\varphi_p = 0°$. When the system is viewed along the central bond j-k in the direction from j to k (or k to j), a counterclockwise rotation of the bond i-j (or l-k) around the central bond j-k is defined to be positive.

In GROMOS the IUPAC-IUB convention is standardly used.

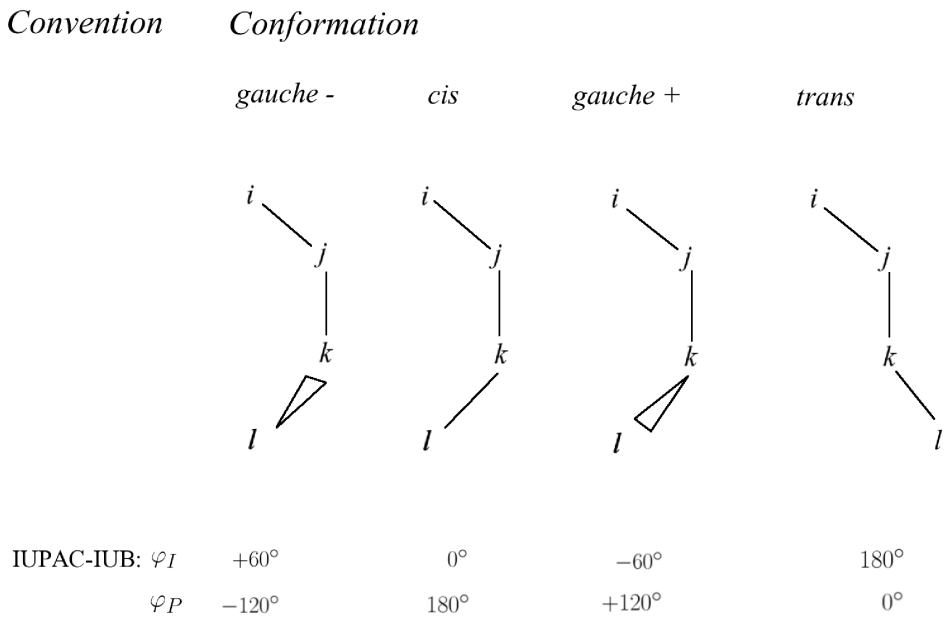

FIGURE 19.1. The relation between a dihedral angle value φ_I according to the IUPAC-IUB convention and the corresponding value φ_p in the polymer convention is $\varphi_p = \varphi_I \pm 180°$. For both conventions the rotation of the angle φ is positive giong from the right to the left through the 4 pictures using the shortest rotation pathway.

The direction of positive rotation is the same in both conventions, only the zero point is shifted by 180°,

$$\varphi_p = \varphi_I \pm 180° \tag{19.1}$$

or

$$\varphi_I = \varphi_p \pm 180° \,. \tag{19.2}$$

The residue name that is associated with a torsional dihedral angle φ(i-j-k-l) is the residue name of the second atom j in the definition of the dihedral angle.

When calculating a dihedral angle value φ using the definition (Eq. 2-5.19) or (Eq. 2-5.14) its value will lie in the range

$$-\pi \leqslant \varphi \leqslant \pi. \tag{19.3}$$

This means that the φ value shows a discontinuity of 2π when passing the point $\varphi = \pm\pi$. When *analyzing trajectories*, that is, dihedral angles as a function of time, the *function $\varphi(t)$ should be made continuous*, i.e. the restriction (Eq. 19.3) should not be applied. This can be achieved by applying the transformation

$$\varphi(t_n) = \varphi(t_n) - NINT((\varphi(t_n) - \varphi(t_{n-1}))/(2\pi)) * 2\pi \tag{19.4}$$

as long as the dihedral angle has not rotated over more than 180° from time point t_{n-1} to time point t_n.

During a simulation the extent of the conformational changes of a solute can be measured by *monitoring the transitions of the torsional dihedral angles* between adjacent minima of the dihedral angle torsional potential energy term (Eq. 2-5.18). The simplest way to define a dihedral angle transition would be to use the passing by the maximum in the torsional dihedral angle energy function $V^{(\varphi)}$, see Fig. 19.2. However, if the dihedral angle φ immediately returns backwards over the barrier, one would not consider such crossing events as two transitions. Therefore, in GROMOS a *dihedral angle transition is only considered to be completed if the dihedral angle passes the bottom of an adjacent well* in the dihedral angle energy function, see Fig. Fig. 19.3.

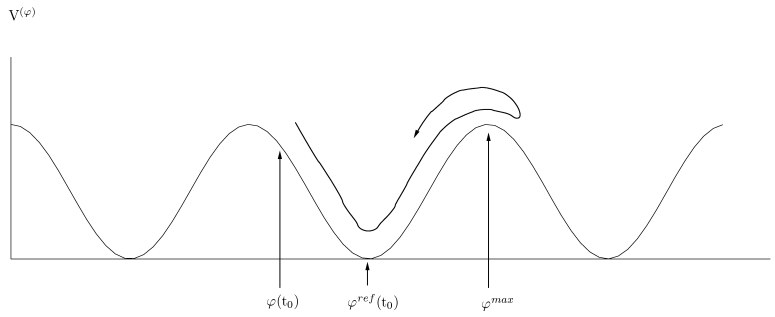

FIGURE 19.2. Monitoring of torsional dihedral angle transitions. A barrier crossing should *not* be counted as a transition.

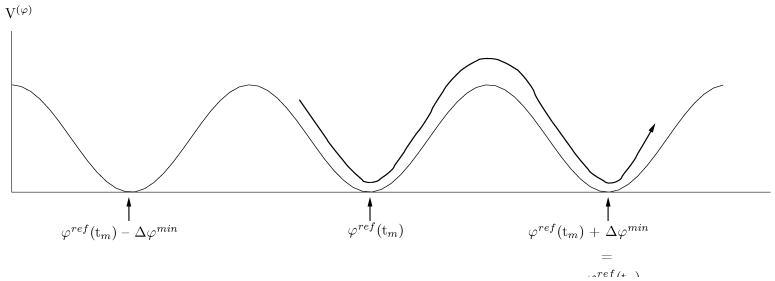

FIGURE 19.3. Monitoring of torsional dihedral angle transitions. A passing by the minimum of an adjacent energy well should be counted as a transition.

The procedure to monitor torsional dihedral angle transitions is the following:

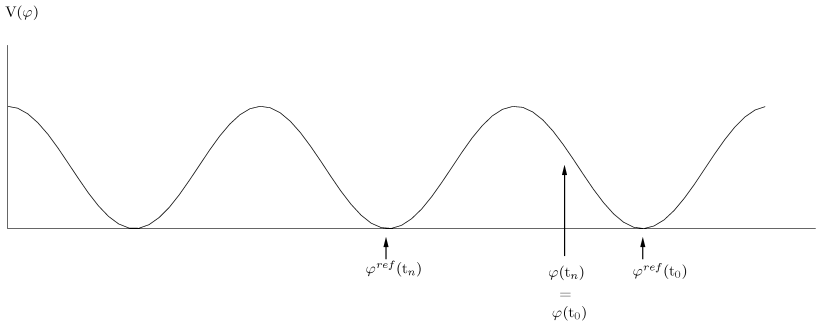

FIGURE 19.4. Monitoring of torsional dihedral angle transitions. Between separate parts of a simulation a transition may be missed.

1. Using the first (t=t_0) molecular configuration, it is determined in which valley of $V^{(\varphi)}$, characterized by the minimum energy dihedral angle $\varphi^{ref}(t_0)$, the dihedral angle $\varphi(t_0)$ is found (Fig. 19.2). The distance between the minima is given by

   $$\Delta\,\varphi^{\min} = 2\pi/m \tag{19.5}$$

   where $m_n^{(\varphi)}$ (=1,2,3,4,5,6) is the multiplicity of the dihedral energy function (Eq. 2-5.18). One of the minima is given by

   $$\varphi^{ref} = \pi[3 - \cos(\varphi_n^0)]/(2m_n^{(\varphi)}) \tag{19.6}$$

   where $\cos(\varphi_n^0) = \pm 1$, see (2.5.5.2). A transformation such as (Eq. 19.7) can be used to bring φ^{ref} within $\pm \quad \pi$ from $\varphi(t_0)$,

   $$\varphi^{ref} = \varphi^{ref} - NINT((\varphi^{ref} - \varphi(t_0))/(2\pi)) * 2\pi, \tag{19.7}$$

   and the minimum $\varphi^{ref}(t_0)$ that is closest to $\varphi(t_0)$ is then found by the transformation

   $$\varphi^{ref}(t_0) = \varphi^{ref} - NINT((\varphi^{ref} - \varphi(t_0))/\Delta\,\varphi^{\min}) * \Delta\,\varphi^{\min}\,. \tag{19.8}$$

   The relation between $\varphi^{ref}(t_0)$ and $\varphi(t_0)$ is illustrated in Fig. Fig. 19.2.
2. At each time point t_n the occurrence of a dihedral angle transition since the last transition at time point t_m is checked. If

   $$|\varphi(t_n) - \varphi^{ref}(t_m)| > \Delta\,\varphi^{\min} \tag{19.9}$$

   a transition is registered and the reference angle is set equal to the minimum of the well that has just been passed by (Fig. Fig. 19.3)

   $$\varphi^{ref}(t_n) = \varphi^{ref}(t_m) + \Delta\,\varphi^{\min}\,\mathrm{sign}(\varphi(t_n) - \varphi^{ref}(t_m)). \tag{19.10}$$

This procedure delivers all transitions that have occurred since the start of a simulation at t=t_0. However, if the monitoring of dihedral transitions is not continuous, but restarted, for example for each separate part (job) of a simulation, a dihedral transition may be missed. This is illustrated in Fig. 19.4. If the first part of a simulation ends with $\varphi(t_n)$ and $\varphi^{ref}(t_n)$ as indicated in Fig. 19.4, the first configuration of the second part, i.e. $\varphi(t_n)$, is used to determine the new $\varphi^{ref}(t_0)$, which will differ (by $\Delta\varphi^{min}$) from the $\varphi^{ref}(t_n)$ at the end of the first part.

CHAPTER 20

Definition of Hydrogen Bonds

Hydrogen bonds may be defined by specifying donor and acceptor atoms and an energetic or geometric criterion. GROMOS++ contains a program called *hbond* (Sec. 5-4.32) which determines the occurrence of hydrogen bonds in a molecular system containing solute and solvent molecules using a geometric criterion.

The *geometry of a hydrogen bond* (Fig. 20.1) is defined by

1. a maximum distance d(H-A) between the hydrogen (H) atom and the acceptor (A) atom, and
2. a minimum angle θ(D-H-A) between the donor (D) atom, the hydrogen (H) and the acceptor (A) atom.

A so-called *three centre hydrogen bond* is defined by the additional specification of

3. a minimum value for the sum of the three angles θ(D-H-A1), θ(D-H-A2) and θ(A1-H-A2), and
4. a maximum value for the (improper) dihedral angle φ(H-A2-A1-D).

FIGURE 20.1. Left: definition of a hydrogen bond using a maximum distance d(H-A) and a minimum angle θ(D-H-A). Right: a three-centre hydrogen bond is defined by the additional specification of a minimum value for the sum of the three angles θ(D-H-A1), θ(D-H-A2) and θ(A1-H-A2), and a maximum value for the (improper) dihedral angle φ(H-A2-A1-D).

In the program, two groups of atoms (A and B) can be specified between which the hydrogen bonds are to be monitored. The hydrogen bond *donor* and *acceptor* atoms are either *identified* using the *atom specifier*, by a *mass file* or a reference structure. In case of the *mass file*, the donor and acceptors are filtered based on their masses which are given in the file (HYDROGENMASS and ACCEPTORMASS block). If a reference structure is provided, only the hydrogen bonds observed in the reference structure are monitored.

In addition, time series of specific hydrogen bonds can be generated using the GROMOS++ program *tser* (Sec. 5-4.63). Thereby is the *property specifier* set to *hb* and the involved atoms (with *atom specifiers*), the distance and angle are to be specified. The default distance is 0.25 nm and the default angle 135 degrees.
General: hb%<atomspec>%<dist_upper>%<angle_lower>
Example: hb%1:res(3:N,H);1:res(5:O)
means the hydrogen bond between the H atom of residue 3 and the O atom of residue 5 of the first molecule.

For *three centre hydrogen bonds*, the default lower angle is 90 degrees, the default sum of angles is 340 degrees and the default angle of the plane is 15 degrees.
General: hb%<atomspec>%<dist_upper>%<angle_lower>%<angle_sum>% <angle_plane>
Example: hb%1:res(3:N,H);1:res(5:O);1:res(6:O)
means the *three centre hydrogen bond* between atom H of residue 3 and atom O of residues 5 and 6 of molecule 1.

CHAPTER 21

Time Correlation Functions and Spectral Densities

As was mentioned in Chap. 18, the dynamic properties of a scalar quantity Q or a vector quantity $\vec{Q}$ can be analyzed in terms of

1. *time series* $Q(t)$ or $\vec{Q}(t)$, and

2. *time correlation functions* $C_Q(t)$ or $C_{\vec{Q}}(t)$.

The time correlation function $C_Q(t)$ of a quantity $Q_i(t)$, or two quantities $Q_i(t)$ and $Q_j(t)$ is defined by

$$\begin{aligned} C_Q(t) &= < Q_i(t') * Q_j(t'+t) >_{t'} \\ &= [t_{\rm MD} - t]^{-1} \int_0^{t_{\rm MD}-t} Q_i(t') Q_j(t'+t) dt' \end{aligned} \tag{21.1}$$

where $t_{\rm MD}$ is the length of the simulation. From a trajectory file the quantities $Q_i(t)$ and $Q_j(t)$ can only be calculated at $\mathcal{N}_t$ discrete, equally spaced time points $n\Delta t$ with n = 0,1,..., $\mathcal{N}_t - 1$. The discrete equivalent of (Eq. 21.1) is then

$$C_Q(n\Delta t) = [\mathcal{N}_t - n]^{-1} \sum_{k=0}^{\mathcal{N}_t - n - 1} Q_i(k\Delta t) Q_j((k+n)\Delta t). \tag{21.2}$$

This formula (Eq. 21.2) requires computation time proportional to ${\mathcal{N}_t}^2$. A much faster method based on the convolution theorem combined with a fast Fourier transform (FFT) algorithm is the following. The discrete Fourier transform of the quantity $Q(t)$ with respect to time t is

$$\hat{Q} = \sum_{k=0}^{\mathcal{N}_t - 1} Q(k\Delta t) e^{+im\Delta\omega k\Delta t} \tag{21.3}$$

where m = 0,1,..., $\mathcal{N}_t - 1$ and

$$\Delta\omega = \frac{2\pi}{\mathcal{N}_t \Delta t}. \tag{21.4}$$

By taking the Fourier transform of (Eq. 21.2) and using the convolution theorem the summation (integral) reduces to a product of the Fourier transformed function $\hat{Q}$ and the complex conjugate $\hat{Q}^*$, which may be subsequently inversely transformed to obtain the time correlation function

$$C_Q(n\Delta t) = [\mathcal{N}_t(\mathcal{N}_t - n)]^{-1} \sum_{m=0}^{N_t - 1} \hat{Q}_i(m\Delta\omega) \hat{Q}_j(m\Delta\omega)^* e^{-im\Delta\omega n\Delta t} \tag{21.5}$$

where $C_Q(t)$ is assumed to be periodic with period $\mathcal{N}_t\Delta t$. This assumption introduces spurious correlations in $C_Q(t)$, which can be avoided by simply adding a series of $\mathcal{N}_t$ zeros to the $n = 0, 1, \ldots, \mathcal{N}_t - 1$ known values $Q_i(n\Delta t)$ and $Q_j(n\Delta t)$. The summation in (Eq. 21.3) and (Eq. 21.5) then contains $2\mathcal{N}_t$ terms and the *FFT expression* for the time correlation function becomes

$$C_Q(n\Delta t) = [2\mathcal{N}_t(\mathcal{N}_t - n)]^{-1} \sum_{m=0}^{2N_t - 1} \hat{Q}_i(m\Delta\omega) \hat{Q}_j(m\Delta\omega)^* e^{-im\Delta\omega n\Delta t} \tag{21.6}$$

with

$$.\Delta\omega = \frac{2\pi}{2\mathcal{N}_t \Delta t} \tag{21.7}$$

This expression for the time correlation function requires computation time proportional to $\mathcal{N}_t lg \mathcal{N}_t$.

The *spectral density* of the *time correlation function* $C_Q(t)$ is its Fourier transform $\hat{C}_Q(\omega)$, or in discrete form

$$\hat{C}_Q(m\Delta\omega) = \sum_{k=0}^{\mathcal{N}_t-1} C_Q(k\Delta t)e^{+im\Delta\omega k\Delta t}. \tag{21.8}$$

Introduction of spurious components by the Fourier transform (Eq. 21.8) can be avoided by making the function to be transformed periodic (with period $2\mathcal{N}_t$), which can be achieved by adding the inverse sequence of $C_Q(t)$ values after $C_Q((\mathcal{N}_t-1)\Delta t)$. Then we find

$$\hat{C}_Q(m\Delta\omega) = \sum_{k=0}^{\mathcal{N}_t-1} C_Q(k\Delta t)e^{+im\Delta\omega k\Delta t} + \sum_{k=\mathcal{N}_t}^{2\mathcal{N}_t-1} C_Q((2\mathcal{N}_t-k-1)\Delta t)e^{+im\Delta\omega k\Delta t}. \tag{21.9}$$

The GROMOS++ program tcl (see Vol. 5) is able to calculate distributions and time-correlation functions form any series of data points. The time correlation functions of the general form

$$C_Q(t) =< f(Q_i(\tau), Q_j(\tau+t)) >_\tau \tag{21.10}$$

can be calculated, where $Q_i(\tau)$ and $Q_j(\tau+t)$ represent the data points at different time points and the user can specify any function $f(Q_i, Q_j)$ which is then inserted into (Eq. 21.2). The program can calculate both auto-correlation functions ($Q_i = Q_j$) and cross correlation functions ($Q_i \neq Q_j$) for time series of scalars or vectors. If Q_i and Q_j are represented by scalars and $f(Q_i, Q_j) = Q_i(\tau) * Q_j(\tau+t)$, the program makes use of fast Fourier transform to calculate $C_Q(t)$. In other cases the direct summing algorithm is used.

21.1. Use of fast Fourier transform (FFT) routines in GROMOS

Application of the PPPM algorithm in MD++ requires the availability of functions performing fast Fourier transforms (FFT). Such functions are collected in FFT libraries. MD++ is using the FFTW library by default and no other library can be selected. In GROMOS++ either the FFT routines from the Gnu Scientific Library (GSL) or the FFTW library are used depending on the individual program.

CHAPTER 22

Coarse Graining in GROMOS

For coarse graining, two different models are implemented in MD++: the MARTINI model[10] and the GROMOS model[11] (for details of both models see the corresponding literature). To use the MARTINI model ($NTCGRAN = 1$ or 2), the Lennard-Jones parameters have to be specified in the special topology block CGPARAMETERS.
The GROMOS model on the other hand ($NTCGRAN = 3$ or 4) makes use of the normal LJPARAMETERS block for the Lennard-Jones parameters and the BONDSTRETCHTYPE block for the bonds, although the range of particles which are coarse grained has to be specified in the topology block CGSOLUTE. As the coarse grained bonds in the GROMOS model are unconstrained, bonds involving coarse grained particles need to be given in the topology block CGBOND.

CHAPTER 23

Parallelisation in GROMOS

The most time-consuming parts of the GROMOS code are parallelised in order to run on shared- or distributed-memory architectures. The details of the parallelisation employed depend on the part of the GROMOS code.

23.1. Parallelisation in MD++

Computationally, the interaction calculation is by far the most expensive part of an MD or SD simulation or an EM, while the non-bonded interactions constitute the bulk of the effort. Again, MD++ is focused on achieving parallelisation without complicating the code. The non-bonded interaction is split up into `Nonbonded_Set`s, each containing its own storage space for a pairlist, energies, forces, and virials. In this way, the standard code is ready for shared and distributed memory parallelisation without any need for code duplication. If the system is using distributed memory, the (updated) positions and box parameters have to be copied from the master to all other processes before the next interaction calculation. While composing the pairlist in parallel, only a subset of atoms is considered per process, so that each processor creates its own partial and local pairlist. The interactions are calculated from this partial pairlist and stored in local arrays. This ensures synchronisation for shared memory machines and replicated data parallelisation for distributed memory systems. For the PPPM method, additional parallelisation steps are required. The mesh used for the long-range interaction evalulation is split and distributed over the individual compute nodes in a slice manner. Every compute node only evaulates the interactions for the atoms mapped on the slice of the grid. After the charge assignment the bordering cells of the slices are shared with the neighbouring nodes. The longe-range energy and virial are calculated on the individual slices of the mesh and summed in the final reduction step. Before the force calculation, bordering cells of the electrostatic potential are again shared with the neighbouring nodes in order to allow the evalulation of the force. All FFT calculations are carried out in a parallel way using the FFTW MPI library. The computation of the expensive $\tilde{A}_2$ term is also parallelised using MPI. After the partial interaction calculations have finished, the energies, forces, and virials of all non-bonded sets are summed up and stored in the `Configuration` of the master process. The SHAKE algorithm of the solvent molecules is parallelised using MPI. The old and new positions of the atoms are broadcasted to the compute nodes and every node applies the SHAKE algorithm on a subset of the solvent molecules. The resulting positions are then send back to the master node. MD++ can use `OpenMP`[12] for shared memory and `MPI`[13] for distributed memory parallelisation. For best performance the use of MPI is recommended. Reasonable parallelisation (using a small number of parallel processes) can be achieved with only a few lines of code (almost) completely separate from the non-bonded routines.

23.2. Parallelisation in GROMOS++

Some analysis programs in GROMOS++ carry out very intensive computation. These programs can be executed in parallel on shared memory architectures using OpenMP[12]. The programs including some form of parallelisation:

1. `filter`: Parallel filtering using a cutoff criterion.
2. `rdf`: Parallel radial distribution function evaluation.
3. `rot_rel`: Parallel rotational correlation function.
4. `utils::Energy`: Parallel interaction function evaluation in programs using the `utils::Energy` class like the program `ener`.

CHAPTER 24

Fast Solvent Interaction Function Evaluation

In a biomolecular simulation, using an explicit representation of the solvent, the solvent-solvent interaction function evaluation is the most time consuming computational step. Using roughly 75% of the computation time, this step is a good candidate for further optimizations. The following features of solvent molecules in the GROMOS software can be used to speed up the solvent-solvent interaction evaluation:

1. A solvent molecule is also a charge group, all atoms in one solvent molecule interact with all atoms in another solvent molecule.
2. A solvent molecule is rigid (i.e. does only have distance constraints and no harmonic bonds, bond angles or dihedral angles)
3. The atoms within a molecule do not interact with each other (no intra-molecular interaction).
4. Inter-molecular interaction cutoff truncation is applied. The first atom in the solvent topology is used to calculate the molecule-molecule distance.

Solvents which do violate these assumptions cannot be simulated using the GROMOS solvent loops but have to be technically treated as solute molecules. These assumptions, with the additional condition that a charge group is always gathered, i.e. no bonds between atoms of a charge group are broken by the periodic boundary condition, allow us to implement more efficient solvent-solvent loops.

24.1. Solvent innerloops in MD++

MD++ contains four additonal innerloops for solvent-solvent interactions. These loops can be controlled using the `INNERLOOP` block in the input file (see 4-95).

1. The first innerloop (`NTILM`=1) is a generic fast version. It takes advantage of the simplified representation of the solvent in order to speed up the interaction evalulation. The pairlist is evalulated in a molecule against molecule instead of an atom against atom way. Because the molecules are gathered, the nearest image calculation is only applied once. The nonbonded parameters are stored in a small atom against atom matrix which is efficiently cached. Application of this loop does not affect the accuracy and results in a speedup factor of about 1.5 for SPC water.
2. The second innerloop (`NTILM`=2) is a solvent specific version. The solvent has to be specified using the `NTILS` switch. In addition to the first method the molecule against molecule loop is manually unrolled and and the nonbonded parameters are hardcoded. The computation steps are aligned in a special way to help the compiler to use efficient (SSE) optimisation and automatic vectorisation. In the initalisation period, checks are made to ensure that the topological parameters are in agreement with the hardcoded ones. This loop does not affect the accuracy and results in a speedup factor of about 1.6 for SPC water. It is recommended to use this loop. Usage of the first method is only recommended if the solvent of interest is not implemented.
3. The third innerloop (`NTILM`=3) is a solvent specific version. It makes use of tables for the interaction evaluation. For every atom-type pair a hardcoded table holding the Lennard-Jones- and Coulomb-Reaction-Field-energies and forces are used. The energies and forces are tabulated for $\mathbf{r}_{ij}^2$ in order to avoid the expensive square-root and inverse computations. In the initalisation period, checks are made to ensure that the topological and input file parameters are in agreement with the hardcoded ones. Between the individual tabulated data points linear interpolation is used. The tables have to be generated and provided as a header file. The program `tabulate_spc` is used to generate the table for SPC water. By default a table size of 5000 for shortrange- and of 2000 for longrange-interactions is used. The resulting energies and forces are approximate and a speedup by a factor of 2 can be expected. Due to the $\mathbf{r}_{ij}^2$ nature of the table, it contains less grid points for small distances than for long distances. For this reason this method should be used with caution in high temperature or pressure simulations.

4. The fourth innerloop (`NTILM`=4) makes use of graphics processing units (GPUs) as acceleration hardware. The solvent-solvent iteractions are not run on the CPU but are executed on the CUDA enabled devices with the device number `NGPUS`. In a first step the positions and box parameters are transfered to the GPU. The pairlist for the solvent-solvent interaction is generated on the GPU. The pairlist evaluation on the GPU is executed in a parallel way[14]. The resulting energies, forces and virials are transfered to the main memory and summed in double precision. The computation is carried out in mixed precision resulting in numerical differences in comparison to the standard loops. Nevertheless, energetic, structural and dynamic quantities are not affected by this[14]. Depending on the GPUs, CPU and solvent used an overall speedup of a factor of 6 to 9 can be expected. In order to use this acceleration technique MD++ has to be compiled using special compiler options (see Sec. 8-3.1.3).

CHAPTER 25

Replica Exchange Simulation

In MD++, temperature and/or Hamiltonian replica exchange simulation can be performed using the program `repex_mpi`. Note that for this MD++ has to be compiled with `MPI`[13]. The replicas are controlled by the `REPLICA` block in the input file (see 4-103). In case of Hamiltonian replica exchange, the `PERTURBATION` block is additionally required (see 4-100). The number of replicas is given by `NRET*NRELAM`. Each replica with its specific combination of T and λ is assigned to a `MPI` process and remains with this process throughout the simulation. The master process is always replica 1. If a switching occurs, the configuration data is exchanged between two replicas.

Each replica writes into its own trajectory files and output file which are distinguished automatically by `_X` in the file name, where `X` is the number of replica (starting at 1). The output file for the replicas is given with the flag `@repout`. In contrast to normal MD++ simulations, only some information from the master about timings are printed to the standard output.

A continuation run is started by setting the parameter `CONT` in the `REPLICA` block to 1. Thus, a separate input coordinate file, distinguished by `_X` in the file name, where `X` is the number of replica, is read in for each replica. With the flag `@conf` simply the root of the file name has to be given and the program will search automatically for files with this root file name containing `_X`.

Information about the switching behaviour with probabilities and energies is printed to the replica data file specified with the flag `@repdat`. This data can be further analyzed using the GROMOS++ program `rep_ana`.

For optimal performance, it is advised to use the same number of `MPI`[13] processes as replicas. Additional speed-up can be obtained by compiling MD++ as `OpenMP`[12]/`MPI`[13] hybrid where each replica is parallelized further by `OpenMP`[12].

Bibliography

[1] ISO 14882:2003. *Programming languages – C++*. ISO, Geneva, Switzerland, 2003.

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

[3] IUPAC-IUB commission on biochemical nomenclature. Abbreviations and symbols for the description of the conformation of polypeptide chains. Tentative rules (1969). *Biochemistry*, 9:3471–3479, 1970.

[4] T. Heinz and P.H. Hünenberger. A fast pairlist construction algorithm for molecular simulations under periodic boundary conditions. *J. Comput. Chem.*, 25:1474, 2004.

[5] I.G. Tironi and W.F. van Gunsteren. A molecular dynamics simulation study of chloroform. *Mol. Phys.*, 83:381–403, 1994.

[6] H. Liu, F. Müller-Plathe, and W.F. van Gunsteren. A Force Field for Liquid Dimethyl Sulfoxide and Physical Properties of Liquid Dimethyl Sulfoxide Calculated Using Molecular Dynamics Simulation. *J. Am. Chem. Soc.*, 117:4363–4366, 1995.

[7] A.D. McLachlan. Gene duplications in the structural evolution of chymotrypsin. *J. Mol. Biol.*, 128:44–77, 1979.

[8] W Kabsch. Solution for best rotation to relate 2 sets of vectors. *Acta Crystallogr.*, A32:922–923, 1976.

[9] G.H. Stout and L.H. Jensen. *X-ray structure determination*. Wiley, New York, USA, 1989.

[10] S.J. Marrink, A.H. de Vries, and A.E. Mark. Coarse Grained model for Semiquantitative Lipid Simulations. *J. Phys. Chem. B*, 108:750, 2004.

[11] S. Riniker and W.F. van Gunsteren. A simple, efficient polarisable coarse-grained water model for molecular dynamics simulations. *J. Chem. Phys.*, 134:084110, 2011.

[12] OpenMP.

[13] The Message Passing Interface.

[14] N. Schmid, M. Bötschi, and W.F. van Gunsteren. A GPU solvent-solvent interaction calculation accelerator for biomolecular simulations using the GROMOS software. *J. Comput. Chem.*, 31:1636–1643, 2010.

Index

Symbols

Symbol	Meaning
	Common names and abbreviations
GROMOS	The GROMOS software package
MD++	The MD++ simulation engine in C++
GROMOS++	The GROMOS++ analysis package in C++
GROMOS96	The GROMOS96 simulation package (1996)
3D	abbreviation for three dimensions
AA	Atomistic (All Atom) models
BD	Brownian Dynamics simulation
B&S – LEUS	Ball and stick local elevation umbrella sampling
CG	Coarse Grained models
CGEM	Conjugate gradient method for energy minimization
FRCG	Fletcher-Reeves conjugate gradient method for energy minimization
PRCG	Polak-Ribiére conjugate gradient method for energy minimization
COG	Center of geometry
COS	Charge On Spring approach
CP	Car Parrinello approach
DF	Distancefield
DOF	Degrees of freedom (abbreviation)
DPD	Diffusive Particle Dynamics simulation
`doxygen`	Documentation platform
EM	Energy minimisation
EDS	Enveloping distribution sampling
FBC	Fixed boundary conditions
HBC	Hyper-spherical boundary conditions
LE	Local elevation
LEUS	Local elevation umbrella sampling
LS	Lattice-sum method
MC	Monte Carlo sampling
MD	Molecular Dynamics simulation
NOE	Nuclear Overhauser Effect
PBC	Periodic boundary conditions
PPPM	Particle-particle–particle-mesh (P^3M) method
QM	Quantum Mechanical models
QMD	Quantum Molecular Dynamics simulation
RDF	Radial distribution function
RE	Replica Exchange
REMD	Replica Exchange Molecular Dynamics simulation
RF	Reaction-field method
RMSD	Root-mean-square difference
RMSF	Root-mean-square fluctuation
SD	Stochastic Dynamics simulation
SDEM	Steepest descent method for energy minimization
TI	Thermodynamic integration
US	Umbrella sampling

Symbol	Meaning
VBC	Vacuum boundary conditions
Physical constants	
h	Planck's constant [0.3990313 kJ mol^{-1} ps]
$\hbar$	Planck's constant divided by 2π [0.06350780 kJ mol^{-1} ps]
$N_{\rm Av}$	Avogadro's number [6.02214 $\times 10^{23}$]
k_B	Boltzmann's constant [1.380662 $\times 10^{-26}$ kJ K^{-1})]
R	Ideal gas constant ($N_{\rm Av} \times k_B$)
c	Speed of light [2.99792458 $\times 10^5$ nm ps^{-1}]
Degrees of freedom and system configuration	
$\mathcal{N}_d$	Number of degrees of freedom of a system
$\mathcal{N}_a$	Number of particles in a system of particles ($\mathcal{N}_d$ =3$\mathcal{N}_a$)
$\mathcal{N}_a^{solu}$	Number of particles the solute consists of
$\boldsymbol{q}$	$3\mathcal{N}_a$-dimensional generalized coordinate vector of a system of particles
$\boldsymbol{p_q}$	$3\mathcal{N}_a$-dimensional generalized momentum vector of a system of particles
$\boldsymbol{r}$	$3\mathcal{N}_a$-dimensional Cartesian coordinate vector of a system of particles
$\boldsymbol{p}$	$3\mathcal{N}_a$-dimensional Cartesian momentum vector of a system of particles
$\boldsymbol{f}$	$3\mathcal{N}_a$-dimensional Cartesian force vector of a system of particles
$\overline{\boldsymbol{f}}$	$3\mathcal{N}_a$-dimensional Cartesian mean force vector of a system of particles
$\boldsymbol{f}^{st}$	$3\mathcal{N}_a$-dimensional Cartesian stochastic force vector of a system of particles
$\mathbf{f}_i^{st}$	$3\mathcal{N}_a$-dimensional Cartesian stochastic force vector of a system of particles
$\boldsymbol{v}$	$3\mathcal{N}_a$-dimensional Cartesian velocity vector of a system of particles
$\mathbf{r}$	3-dimensional Cartesian coordinate vector of a particle
$\mathbf{p}$	3-dimensional Cartesian momentum vector of a particle
$\mathbf{f}$	3-dimensional Cartesian force vector of a particle
$\mathbf{v}$	3-dimensional Cartesian velocity vector of a particle
Ψ $[\Psi(\boldsymbol{r})]$	Wavefunction (position representation; configuration of a quantum-mechanical system of $\mathcal{N}_a$ particles)
$\{$ $\boldsymbol{r}$,$\boldsymbol{p}$ $\}$	Phase-space point (Cartesian coordinates; configuration of a classical system of $\mathcal{N}_a$ particles)
(Statistical) thermodynamics	
$\mathcal{F}$	Free energy
G	Gibbs free energy
H	Enthalpy
$\mathcal{U}$	Energy of a system
$\mathcal{S}$	Entropy of a system
$\mathcal{Z}$	Partition function
$\mathcal{T}$	Instantaneous temperature
$\mathcal{T}_o$	Reference temperature
$\mathcal{K}$	Instantaneous kinetic energy of a system
$\mathcal{K}_{tr}$	Instantaneous translational kinetic energy
$\mathcal{K}_{ir}$	Instantaneous internal+rotational kinetic energy
$\mathcal{U}$	Instantaneous total potential energy of a system
$\mathcal{W}$	Instantaneous virial of a system
$\mathcal{P}$	Instantaneous pressure of a system
$\mathcal{V}$	Instantaneous volume of a system
ρ_J	Number particle density of particles J
Miscellaneous	

Symbol	Meaning
t	Time
Δt	discrete time step
$\mathcal{N}_t$	Number of MD steps
P	Probability
m	Mass of a particle
M	Mass of the whole system
$\underline{\mathbf{m}}$	Diagonal mass matrix of a system of $\mathcal{N}_a$ particles
γ	Friction coefficient of a particle
$\underline{\gamma}$	Diagonal friction coefficient matrix of a system of $\mathcal{N}_a$ particles
T	Absolute temperature
β	prefactor: $1/k_BT$
τ_T	relaxation time for the coupling to a temperature bath
$\mathbf{s}$	Vector denoting the collection of all force-field parameters
λ	Coupling parameter Lambda for a lambda dependent Hamiltonian
$\mathcal{N}_\lambda$	Number of λ-values in a TI simulation
H	Heaviside function defined as $H(x) = 0 \; \forall \; x < 0$ and $H(x) = 1 \; \forall \; x > 0$
sign	Sign function: $\text{sign}(x) = 1 \; \forall \; x > 0$ and $\text{sign}(x) = -1 \; \forall \; x < 0$
i	imaginary number, $i^2 = -1$
δ_{ij}	general Kronecker delta
σ	Standard deviation
σ^2	Variance
$\mathcal{N}_{conf}$	Number of configurations in an ensemble
D	Diffusion constant
R_{gyr}	radius of gyration
η	the viscosity of a system
$g(r)$	radial distribution function
s	Smoothness parameter in EDS simulations
E^R	Energy offset parameter in EDS simulations
$\mathcal{N}^{(s)}$	Number of states in EDS simulations
Spatial boundary conditions	
$\underline{\mathcal{B}}$	3×3-matrix of the box-edge vectors (columns) in the reference Cartesian coordinate system (PBC)
$\hat{\mathbf{e}}$	Unit vector
$\mathbf{a}$	First edge vector of a (triclinic) box (in the reference coordinate system)
$\mathbf{b}$	Second edge vector of a (triclinic) box (in the reference coordinate system)
$\mathbf{c}$	Third edge vector of a (triclinic) box (in the reference coordinate system)
a	length of first edge of a (triclinic) box
b	length of second edge of a (triclinic) box
c	length of third edge of a (triclinic) box
$\mathbf{T}$	Position vector of the reference corner of a triclinic box (components in the reference coordinate system and vector relative to the origin of this system)
$\underline{\mathbf{L}}$	Computational box matrix (columns defined by the components of edge vectors $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ in the reference coordinate system)
$\underline{\mathbf{B}}$	Edge length matrix (diagonal, elements a, b and c)
α	First edge angle a triclinic box (between $\mathbf{b}$ and $\mathbf{c}$)
β	Second edge angle a triclinic box (between $\mathbf{a}$ and $\mathbf{c}$)
γ	Third edge angle a triclinic box (between $\mathbf{a}$ and $\mathbf{b}$)
ϕ	First Euler angle of a triclinic box

Symbol	Meaning
θ	Second Euler angle of a triclinic box
ψ	Third Euler angle of a triclinic box
$\check{\mathbf{r}}$	Oblique coordinates of a real-space vector (with reference to the box-edge vectors)
$\check{\mathbf{r}}$	Oblique fractional coordinates of a real-space vector (with reference to the box-edge vectors)
$\check{\mathbf{k}}$	Oblique coordinates of a reciprocal-space vector
$\check{\mathbf{k}}$	Oblique fractional coordinates of a reciprocal-space vector
$\mathbf{l}$	Lattice vector (three-dimensional vector with integer components)
$\mathbf{k}$	Reciprocal-lattice vector ($\mathbf{k} = 2\pi\underline{\mathbf{L}}^{-1}\mathbf{l}$)
$\underline{\mathbf{S}}$	Transformation matrix
$\underline{\mathbf{R}}$	Transformation matrix
$\underline{\mathbf{T}}$	Transformation matrix
Representation of the interaction	
$\hat{\mathcal{H}}$	Hamiltonian operator describing the interaction for quantum-mechanical degrees of freedom
$\hat{\mathcal{K}}$	Kinetic energy operator (kinetic energy contribution to the quantum-mechanical Hamiltonian operator)
$\hat{\mathcal{V}}$	Potential energy operator (potential energy contribution to the quantum-mechanical Hamiltonian operator)
$\mathcal{H}$ $[\mathcal{H}(\boldsymbol{r},\boldsymbol{p})]$	Hamiltonian function describing the interaction for classical degrees of freedom
$\mathcal{K}$ $[\mathcal{K}(\boldsymbol{p})]$	Kinetic energy contribution to the classical Hamiltonian function
$\mathcal{V}$ $[\mathcal{V}(\boldsymbol{r})]$	Potential energy contribution to the classical Hamiltonian function
$\overline{\mathcal{V}}$ $[\overline{\mathcal{V}}(\boldsymbol{r})]$	Potential of mean force contribution to the classical Hamiltonian function
Physical interactions	
φ [Proper dihedral-angle term]	
$\mathcal{V}^{(phys)}$ $[\mathcal{V}^{(phys)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Physical potential energy contribution to $\mathcal{V}$
$\mathcal{V}^{(cov)}$ $[\mathcal{V}^{(cov)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Covalent potential energy contribution to $\mathcal{V}^{(phys)}$
$\mathcal{V}^{(nbd)}$ $[\mathcal{V}^{(nbd)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Non-bonded potential energy contribution to $\mathcal{V}^{(phys)}$
$\mathcal{V}^{(b)}$ $[\mathcal{V}^{(b)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Bond stretching potential energy contribution to $\mathcal{V}^{(cov)}$
$\mathcal{V}^{(\theta)}$ $[\mathcal{V}^{(\theta)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Bond-angle bending potential energy contribution to $\mathcal{V}^{(cov)}$
$\mathcal{V}^{(\xi)}$ $[\mathcal{V}^{(\xi)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Improper dihedral-angle bending potential energy contribution to $\mathcal{V}^{(cov)}$
$\mathcal{V}^{(\varphi)}$ $[\mathcal{V}^{(\varphi)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Proper dihedral-angle torsion potential energy contribution to $\mathcal{V}^{(cov)}$
$\mathcal{V}^{(vdw)}$ $[\mathcal{V}^{(vdw)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Van der Waals potential energy contribution to $\mathcal{V}^{(nbd)}$
$\mathcal{V}^{(ele)}$ $[\mathcal{V}^{(ele)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Electrostatic potential energy contribution to $\mathcal{V}^{(nbd)}$
$\mathcal{V}^{(LJCRF)}$	Sum of the non-bonded potentials $\mathcal{V}^{(vdw)}$ and $\mathcal{V}^{(ele)}$
Physical force-field terms	
$V^{(b)}$ $[V^{(b)}(b;k^{(b)},b^0)]$	Potential energy function associated with the stretching of a single covalent bond (quartic: $V^{(b,q)}$; harmonic: $V^{(b,h)}$; soft harmonic: $V^{(bs,h)}$)
$V_n^{(b)}$ $[V^{(b)}(b_n;k_n^{(b)},b_n^0)]$	Potential energy function associated with the stretching of the nth single covalent bond (quartic: $V_n^{(b,q)}$; harmonic: $V_n^{(b,h)}$; soft harmonic: $V_n^{(bs,h)}$)
$\mathbf{f}^{(b,q)}$	Force due to the bond stretching potential (quartic)
$\mathbf{f}^{(b,h)}$	Force due to the bond stretching potential (harmonic)
$\mathbf{f}^{(bs,h)}$	Force due to the bond stretching potential (soft harmonic)
$N^{(b)}$	Number of covalent bonds in the molecular system
$N^{(bs)}$	Number of soft covalent bonds in the molecular system
$M_n^{(b)}$	Bond type code associated with covalent bond term n

Symbol	Meaning
$b_n \quad [b_n(\boldsymbol{r}, \underline{\mathcal{B}})]$	Length of covalent bond n in the considered configuration
$b_n^0 \quad [b^0(M_n^{(b)}, \mathbf{s})]$	Reference length of covalent bond term n
$k_n^{(b,q)}$	Force constant of stretching for covalent bond term n (quartic potential)
$k_n^{(b,h)}$	Force constant of stretching for covalent bond term n (harmonic potential)
$V^{(\theta)} \quad [V^{(\theta)}(\theta; k^{(\theta)}, \theta^0)]$	Potential energy function associated with the bending of a single covalent bond angle (cosine-harmonic: $V^{(\theta,c)}$; soft cosine-harmonic: $V^{(\theta s,c)}$; angle-harmonic: $V^{(\theta,h)}$)
$V_n^{(\theta)} \quad [V_n^{(\theta)}(\theta_n; k_n^{(\theta)}, \theta_n^0)]$	Potential energy function associated with the bending of the nth covalent bond angle (cosine-harmonic: $V_n^{(\theta,c)}$; soft cosine-harmonic: $V_n^{(\theta s,c)}$; angle-harmonic: $V_n^{(\theta,h)}$)
$\mathbf{f}^{(\theta,c)}$	Force due to the bond angle potential (cosine-harmonic)
$\mathbf{f}^{(\theta s,c)}$	Force due to the bond angle potential (soft cosine-harmonic)
$\mathbf{f}^{(\theta,h)}$	Force due to the bond angle potential (angle-harmonic)
$N^{(\theta)}$	Number of covalent bond angles in the molecular system
$M_n^{(\theta)}$	Bond-angle type code associated with covalent bond-angle term n
$\theta_n \quad [\theta_n(\boldsymbol{r}, \underline{\mathcal{B}})]$	Value of covalent bond angle n in the considered configuration
$\theta_n^0 \quad [\theta^0(M_n^{(\theta)}, \mathbf{s})]$	Reference angle of covalent bond-angle term n
$k_n^{(\theta,c)}$	Force constant of bending for covalent bond-angle term n (cosine-harmonic potential)
$k_n^{(\theta s,c)}$	Force constant of bending for covalent bond-angle term n (soft cosine-harmonic potential)
$k_n^{(\theta,h)}$	Force constant of bending for covalent bond-angle term n (angle-harmonic potential)
$V^{(\xi)} \quad [V^{(\xi)}(\xi; k^{(\xi)}, \xi^0)]$	Potential energy function associated with the bending of a single covalent improper dihedral angle
$V^{(\xi s)} \quad [V^{(\xi s)}(\xi; k^{(\xi)}, \xi^0)]$	Potential energy function associated with the bending of a single covalent improper dihedral angle
$\mathbf{f}^{(\xi)}$	Force due to the improper dihedral-angle potential
$\mathbf{f}^{(\xi s)}$	Force due to the soft improper dihedral-angle potential
$N^{(\xi)}$	Number of covalent improper dihedral angles in the molecular system
$N^{(\xi s)}$	Number of covalent improper dihedral angles in the molecular system
$M_n^{(\xi)}$	Improper dihedral-angle type code associated with covalent improper dihedral-angle term n
$\xi_n \quad [\xi_n(\boldsymbol{r}, \underline{\mathcal{B}})]$	Value of covalent improper dihedral angle n in the considered configuration
$\xi_n^0 \quad [\xi^0(M_n^{(\xi)}, \mathbf{s})]$	Reference angle of covalent improper dihedral-angle term n
$k_n^{(\xi)}$	Force constant of bending for covalent improper dihedral-angle term n
$V^{(\varphi)} \quad [V^{(\varphi)}(\varphi; k^{(\varphi)}, \varphi^0)]$	Potential energy function associated with the torsion of a single covalent proper dihedral angle (symmetric potential: $V^{(\varphi,s)}$; generalized: $V^{(\varphi,g)}$)
$\mathbf{f}^{(\varphi,s)}$	Force due to the symmetric proper dihedral-angle potential
$\mathbf{f}^{(\varphi,g)}$	Force due to the generalized proper dihedral-angle potential
$N^{(\varphi)}$	Number of covalent proper dihedral angles in the molecular system
$M_n^{(\varphi)}$	Proper dihedral-angle type code associated with covalent proper dihedral-angle term n
$\varphi_n \quad [\varphi_n(\boldsymbol{r}, \underline{\mathcal{B}})]$	Value of covalent proper dihedral angle n in the considered configuration
$\varphi_n^0 \quad [\varphi^0(M_n^{(\varphi)}, \mathbf{s})]$	Reference angle (phase shift) of covalent proper dihedral-angle term n
$m_n^{(\varphi)} \quad [m_n^{(\varphi)}(M_n^{(\varphi)}, \mathbf{s})]$	Multiplicity of covalent proper dihedral-angle term n
$k_n^{(\varphi,s)}$	Force constant of torsion for covalent proper dihedral-angle term n (symmetric potential; $\varphi_n^0 = 0, \pi$; $m_n^{(\varphi)} \leq 6$)

Symbol	Meaning
$k_n^{(\varphi,g)}$	Force constant of torsion for covalent proper dihedral-angle term n (generalized potential; $\varphi_n^0 \in [0, 2\pi[$)
q	Partial charge of an atom or site
C_{12}	Van der Waals (Pauli) repulsion coefficient of an atom or site (Lennard-Jones function)
C_6	Van der Waals (London) dispersion coefficient of an atom or site (Lennard-Jones function)
C_{126}	Ratio of Van der Waals coefficients $\frac{C_{12}}{C_6}$ (Lennard-Jones function)
α_{LJ}	Lennard-Jones soft-core switching parameter
α_C	Coulomb soft-core switching parameter
$\mathcal{V}^{(ele,pws)}$ $[\mathcal{V}^{(ele,pws)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Pairwise potential energy contribution to $\mathcal{V}^{(ele)}$
$\mathcal{V}^{(ele,slf)}$ $[\mathcal{V}^{(ele,slf)}(\underline{\mathcal{B}};\mathbf{s})]$	Self potential energy contribution to $\mathcal{V}^{(ele)}$
$\mathcal{V}^{(ele,srf)}$ $[\mathcal{V}^{(ele,srf)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Surface potential energy contribution to $\mathcal{V}^{(ele)}$
$\mathbf{f}^{(nbd)}$	Force due to the non-bonded forces
$\Psi_{ij}^{(ele)}$ $[\Psi_{ij}^{(ele)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Electrostatic influence function associated with the particle pair $i-j$
$\delta_{ij}^{(exc)}$ $[\delta_{ij}^{(exc)}(\mathbf{s})]$	Indicator of non-bonded exclusion for the particle pair $i-j$
$\Psi^{(ele,slf)}$ $[\Psi^{(ele,slf)}(\underline{\mathcal{B}})]$	Electrostatic self influence function
$\psi^{(RF)}$ $[\psi^{(RF)}(\mathrm{x})]$	Influence function at distance x of a particle in RF electrostatics
H $[H(\mathrm{x})]$	Heaviside step function (one if x is positive, zero otherwise)
R_C	Cutoff distance (truncation)
R_{cp}	Short-range cut-off
R_{cl}	Long-range cut-off
R^{cg}	radius of a charge group
N_{cg}	number of atoms belonging to a charge group
R_{RF}	Cutoff distance (onset of the RF continuum; usually set equal to R_C)
ϵ_{RF}	Relative dielectric permittivity of the RF continuum (usually set equal to that of the solvent)
κ_{RF}	Inverse Debye screening length of the RF continuum (usually set to zero)
C_{RF}	Constant characterizing the effect of the RF continuum
$\overline{\mathbf{R}}_{ij}$ $[\overline{\mathbf{R}}_{ij}(\boldsymbol{r})]$	Vector (FBC) or minimum-image vector (PBC) connecting the center of the CG containing particle j to the center of the CG containing particle i (norm $\overline{R}_{ij}$)
$\mathcal{V}^{(ele,pws,RF-CB)}$ $[\mathcal{V}^{(ele,pws,RF-CB)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Coulombic pairwise potential energy contribution to $\mathcal{V}^{(ele,pws)}$ (RF electrostatics)
$\mathcal{V}^{(ele,pws,RF-RF)}$ $[\mathcal{V}^{(ele,pws,RF-RF)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Distance-dependent pairwise potential energy contribution to $\mathcal{V}^{(ele,pws)}$ (RF electrostatics)
$\mathcal{V}^{(ele,pws,RF-RC)}$ $[\mathcal{V}^{(ele,pws,RF-RC)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Distance-independent pairwise potential energy contribution to $\mathcal{V}^{(ele,pws)}$ (RF electrostatics)
$\Psi_{ij}^{(ele,LS-RS)}$ $[\Psi_{ij}^{(ele,LS-RS)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Real-space component of electrostatic influence function $\Psi_{ij}^{(ele)}$ (LS electrostatics)
$\Psi_{ij}^{(ele,LS-KS)}$ $[\Psi_{ij}^{(ele,LS-KS)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Reciprocal-space component of the electrostatic influence function $\Psi_{ij}^{(ele)}$ (LS electrostatics)
$\mathcal{V}^{(ele,pws,LS-RS)}$ $[\mathcal{V}^{(ele,pws,LS-RS)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Real-space pairwise potential energy contribution to $\mathcal{V}^{(ele,pws)}$ (LS electrostatics)
$\mathcal{V}^{(ele,pws,LS-KS)}$ $[\mathcal{V}^{(ele,pws,LS-KS)}(\boldsymbol{r};\underline{\mathcal{B}};\mathbf{s})]$	Reciprocal-space pairwise potential energy contribution to $\mathcal{V}^{(ele,pws)}$ (LS electrostatics)
$\psi^{(LS)}$ $[\psi^{(LS)}(\mathbf{x})]$	Influence function at position $\mathbf{x}$ relative to a particle in LS electrostatics
a	Width of the charge-shaping function
γ $[\gamma(\mathrm{x})]$	Charge-shaping function

Symbol	Meaning
$\hat{\gamma}$ $[\hat{\gamma}$ (x)]	Fourier transformed charge-shaping function
$\mathbf{E}$	Electric field
$\boldsymbol{\mu}$	Dipole
$\boldsymbol{J}$	
α	Electronic polarisability
$\mathbf{P}$	Polarisation
ϵ	Dielectric permittivity
γ^{pol}	γ to calculate position of off site charge
k^{ho}	harmonic force constant in the COS model
ϕ	Electrostatic potential
Unphysical force-field terms	
$\mathcal{V}^{(spec)}$	Unphysical potential energy
$\mathcal{V}^{(res)}$	Restraint energy
$\mathcal{V}^{(pr)}$	Position restraining potential energy contribution to $\mathcal{V}^{(phys)}$
$\mathbf{f}^{(c)}$	Force due to the position constraints
$k^{(pr)}$	Force constant of an unphysical position-restraining term
$\mathcal{N}^{(pr)}$	number of positionally restrained atoms
l	Lagrange multiplier for position constraints
$\mathcal{V}^{(dr)}$	Distance restraining potential energy contribution to $\mathcal{V}^{(phys)}$
$\mathbf{f}^{(dir)}$	Force due to the atom-atom distance restraints
$k^{(dr)}$	Force constant of an unphysical distance-restraining term
$\mathbf{r}^0$	Equilibrium distance of distance restraint
$N^{(dir)}$	Number of atom-atom distance restraints
d_{CH}	carbon-hydrogen distance
d_{CC}	carbon-carbon distance
τ_{dr}	decay time for time-averaged distance restraining
$\mathcal{V}^{(tr)}$	Dihedral-angle restraining potential energy contribution to $\mathcal{V}^{(phys)}$
$k^{(tr)}$	Force constant of an unphysical dihedral-angle restraining term
$\mathcal{N}^{(tr)}$	number of restrained dihedral angles
$\mathcal{V}^{(Jr)}$	^3J-restraining potential energy contribution to $\mathcal{V}^{(phys)}$
$k^{(Jr)}$	Force constant of an unphysical ^3J-value restraining term
^3J	J-value or J-coupling constant
$^3J^0$	experimental J-value
J	general representation of a J-value
J^0	experimental J-value
ΔJ^0	width of flat-bottom for J-value restraining
a	a in Karplus relation
b	b in Karplus relation
c	c in Karplus relation
τ^s_{Jr}	period of scaling in periodically-scaled J-value restraining
Δt_ω	time period for which scaling is suspended in periodically-scaled J-value restraining
N_{le}	number of bins in J-value local elevation biasing
$w_{\zeta ni}$	weight of gaussian in J-value LE
$\mathcal{V}^{(Fxr)}$	$\mid F \mid$-restraining potential energy contribution to $\mathcal{V}^{(phys)}$
$\mathcal{V}^{(exr)}$	ρ-restraining potential energy contribution to $\mathcal{V}^{(phys)}$
$\mathcal{V}^{(sxr)}$	symmetry restraining potential energy contribution to $\mathcal{V}^{(phys)}$

Symbol	Meaning
k^{xr}	(harmonic) force constant for the crystallographic restraining
k^{sym}	harmonic force constant for the crystallographic symmetry restraining
F	Structure factor amplitude
ρ	Electron density
S	space group of a crystal
N_{sym}	Number of symmetry operations of a space group
$\mathbb{S}$	Symmetry operator $\mathbb{S} = \mathbf{R}\mathbf{r} + \mathbf{t}$
$\mathbf{R}$	Rotation matrix of a symmetry operator
$\mathbf{t}$	Translation vector of a symmetry operator
$\mathcal{V}^{(Sr)}$	S^2-restraining potential energy contribution to $\mathcal{V}^{(phys)}$
$k^{(Sr)}$	Force constant of an unphysical S^2-value restraining term
S^2	S^2-order parameter
$S^{2,0}$	experimental S^2-value
S	general representation of a S^2-value
S^0	experimental S^2-value
$\mathcal{V}^{(df)}$	Distancefield restraining potential energy contribution to $\mathcal{V}^{(phys)}$
$\mathbf{f}^{(df)}$	Force due to the atom-atom distance restraints
$k^{(df)}$	Force constant of an unphysical distance-restraining term
l^0	Equilibrium distance of distance restraint
g_s	Distancefield grid distance
$\mathcal{V}^{(le)}$	Local elevation (LE) energy
$\mathcal{V}^{(bias)}$	bias energy
γ	LE basis function
$k^{(le)}$	LE force constant
$\mathbf{r}^{uc}$	unconstrained atomic positions
N_c	Number of constraints
N_{sh}	number of iterations of the SHAKE algorithm
d^0	constraint length
$\mathbf{f}^{uc}$	unconstrained atomic forces