MagiC: Software package for Multiscale Coarse-Grained Modelling. User guide.

Alexander Mirzoev[1], Alexander Lyubartsev[2]

September 17, 2020

Abstract

MagiC is a software package designed to perform systematic structure-based coarse-graining of a wide range of molecular systems.. The effective pairwise potentials between coarse-grained sites of a low-resolution model are constructed to reproduce structural distribution functions obtained from the modelling of the system in a high resolution (atomistic) description. The software contains tools to read atomistic trajectories generated by different simulation packages, create a coarse-grained trajectory, compute for it radial distribution functions as well as distributions of intramolecular bonds and angles, and then find effective potentials which reproduce these distributions in coarse-grained modeling. The software supports coarse-grained tabulated intramolecular bond and angle interactions, as well as tabulated non-bonded interactions between different site types in the coarse-grained system, with the treatment of long-range electrostatic forces by the Ewald summation. Two methods of effective potentials refinement are implemented: iterative Boltzmann inversion and inverse Monte Carlo, the later accounting for cross-correlations between pair interactions. MagiC uses its own Metropolis Monte Carlo sampling engine, which is efficiently parallelized providing fast convergence of the method and nearly linear scaling at parallel execution. The resulting CG-model can be exported to high performance Molecular Dynamics software (such as LAMMPS, GROMACS, GALAMOST) for subsequent large-scale simulations.

[1] *School of Biological Sciences, Nanyang Technological University, Singapore* `amirzoev@gmail.com`

[2] *Physical Chemistry Division, MMK, Stockholm University, Stockholm, Sweden.* `alexander.lyubartsev@mmk.su.se`

Contents

1 What's new

1.1 Version 3

CGTraj

- Read input all-atom trajectories in XTC/TRR format
- Write output beadmapped trajectories in XTC-format
- Standalone tool for converting XMOL-trajectories to XTC: xmol2xtc
- In addition to single atoms, user can specify ranges of atoms in CG-bead definition

RDF

- Support for XTC trajectories as input
- Parallel multi-core RDF calculation. (Only a shared memory version, no MPI-support)
- Flexible exclusions for non-bonded short-range interactions, based on number of bonds between the pair of atoms.
- Support for SameAsBond-records, allowing to have one bond potential shared between bonds belonging to different molecular types.
- Improved performance and memory consumption for larger systems.

Magic.Core

- Flexible exclusions for non-bonded short-range and electrostatic interactions
- Improved performance
- Many bugs were fixed since previous version
- Starting configurations can be randomly read from a single trajectory file
- Existing external trajectory can be used instead of Monte-Carlo sampling.
- Store and update cross-correlation matrix in external binary file.
- New mode for inverse procedure: Variational Inverse Monte Carlo
- Set probabilities of rotation/translation steps separately for each molecular type. Useful when one have one large molecule surrounded by small molecules with more than 1 atom, as by default all the chances are even for all molecules, except mono-atomic ones.

MagicTools

- Support for LAMMPS and GALAMOST packages as external Coarse Grain Molecular Dynamics engines. MagicTools has unified methods to export system's topology and potentials to internal formats of LAMMPS, GALAMOST and GROMACS.
- Enhanced plotting engine. The plotting interface is reduced to just two methods: OnePlot and MultPlot. They now provide more control over the plot, such as x/y-range, axis labels, title and legend template. User can also directly specify line thickness, color and style for a DFset-object and for every individual function.

1.2 Version 2.2

Code restructuring: Major changes from v.2.0 to v.2.2 consist in code restructuring to make it more clear and more convenient for future development. Also, the algorithm of MC engine was rewritten, particularly a matrix of distances between all pairs of particles is introduced which provided substantial speed-up of the calculations. The support for non uniform Monte Carlo steps was introduced, so individual step size can be defined for every molecule type (for atom displacement, molecule translation and molecule rotation steps). Also added automatic MC step adjustment, which tries to adjust MC steps on runtime, providing desired acceptance ratio (specified by user). This is only performed during equilibration phase.

There were however no principal changes from the user point of view, both concerning new functionality or syntax of input/output files.

1.3 Version 2

New functionality: Possibility to fix/protect certain interaction potentials from correction, i.e. exclude a potential from update within the inverse procedure, is introduced. This gives possibility to use potentials parametrized previously in other simulations, and calculate only those potentials that yet missing in the new system.

New file formats: A new file format (compared to versions 1.*) for RDFs/potentials is introduced and old formats are depreciated (the conversion tools are provided). A new format is also introduced for MagiC-core input file, while keeping limited compatibility with the old format. The new input file format is a free text based, case insensitive and supports comment lines (starting with `!` or `#`) and empty lines. The total number of parameters was reduced and the parameters got more self-explanatory names. The old parameter names are supported as well. The RDF calculation tool rdf.py also got a revised format of the input file, which provides automatic generation of the atom pairs lists used for each specific RDF. The new rdf.py is completely written in Python (the Fortran part was overtaken by numpy), which made it faster and also removed compatibility issues with MacOS.

Improved error reporting: The error reporting was significantly improved and became (hopefully) more tolerant to the user-provided input.

2 Install

2.1 Getting started

First, pull the last stable version of the software from the repository. For this you may need to have mercurial (hg) installed on you computer:

```
hg clone https://bitbucket.org/magic-su/magic-3
```

This will result in a new folder `magic-3`, containing the whole package. Below we will refer to `MAGIC` as full path to this folder.

The whole package consists of 3 parts: `CGTraj` - the tool for Bead Mapping; `magic` - the engine for effective potentials calculation; `MagicTools` - python-based libraries for RDF calculation `rdf.py` and for data manipulation and analysis of all intermediate data `MagicTools.py`;

For the python-based part you need python (ver¿=3.5) and the following modules: numpy, matplotlib, seaborn, pandas, scipy. We recommend to install Jupyter Notebook which provides extremely convenient environment for processing of your data with python-based MagicTools. They all are parts of Anaconda python distribution, which is freely available to download https://www.anaconda.com/download/.

Alternatively, one can use `pip` python package manager to install all the packages:

```
sudo pip install numpy matplotlib scipy seaborn pandas ipython jupyter
```

For the Fortran-based part of MagiC you need to have a Fortran compiler. Currently we support Intel Fortran and GNU Fortran. Unfortunately Oracle Solaris Studio Fortran is deprecated since it is not compatible with Fortran 2008 standard. Hopefully they will catch it up, so we will be able to get it in the future releases.

To get more information about MagiC, please visit the code repository at BitBucket.

Tutorials and examples of the input files can be found here or downloaded using mercurial

```
 hg clone https://bitbucket.org/magic-su/magic-3-tutorials
```

2.2 Fast Installation

Run the `install.sh` script. That is it!

By default, fast installation script uses GNU Fortran compiler and compiles MagiC-core in serial (non-MPI) version. If you want to specify different compiler/options you need to provide them as a command line arguments, for example:

```
> install.sh intel intel-mpi
```

The first argument defines compiler for CGTraj and the second one defines compiler for MagiC.Core

The script will compile all pieces of the package, make links to executable files in folder `$MAGIC/bin` and set environment variables `PATH`, `LD_LIBRARY_PATH` and `PYTHONPATH`.

If you are facing errors during execution of install.sh, you can try to manually compile MagiC step-by-step as described in section 2.3. below.

2.3 Environment variables

To update the environmental variables and get access to the compiled executables and libraries you just need to use the script setvars.sh: `source setvars.sh`

Otherwise you can set the values at startup files of your shell such as `$HOME/.bashrc` or `.profile`.

2.4 Manual Compilation

2.4.1 *CGTraj*

Enter sub-folder `CGTraj` and run `make` specifying the compiler as make option, e.g. `make intel` for Intel Fortran or `make gfortran` for GNU Fortran (default option). If compilation went successfully you will get a binary file called `cgtraj`.

2.4.2 *Magic-Core*

Enter the corresponding sub-folder `magic` and run `make`. To compile this part of the package, you need to have a Fortran compiler and LAPACK linear algebra library. LAPACK is included in Intel MKL, which usually comes with Intel Fortran. Other option is GNU Fortran (gfortran) and a GCC build of LAPACK (liblapack-dev). If you have any of them just use corresponding Makefile for compilation, or use `make` with argument, for example `make intel`. Run `make` to see all options available. As it is the most computationally intensive part of MagiC,it is highly recommended to compile it for parallel execution. To do so you need to have a MPI-library installed. We have tested MagiC with OpenMPI, but it should work with other MPI-implementations. The result of successful compilation is a binary file `magic` which you can copy/link to your default bin folder.

2.4.3 *RDF and MagicTools*

These parts of the package are written in Python and do not require explicit compilation, so you just need to link file `rdf.py` to your default `bin` folder and add `MagicTools` and `MagicTools/lib` folder `PYTHONPATH` environment variable.

To check if the library is added successfully, open terminal, run `ipython` and load the library: `import MagicTools`
If no error message appeared, the library is connected correctly.

2.4.4 *XTC trajectory file format support*

In addition to simple text-based xmol trajectory, MagiC supports XTC, which is a popular binary trajectory file format used in GROMACS and LAMMPS. For this MagiC uses external library xdrfile.

The library is automatically compiled by the `install.sh` script, however it requires access to GNU compiler set (CC and gfortran). You can see if the library was successfully compiled by checking existence of file
`$MAGIC/lib/xdrfile-1.1.4/lib/libxdrfile.a`

For manual compilation enter `$MAGIC/lib/xdrfile-1.1.4` subfolder and run `./configure --enable-shared`
Build the library by `make install`. If you have specified non-default location for the library, add it to the LD_LIBRARY_PATH environment variable. The

last thing is to add python wrapper for the library in PYTHONPATH:

```
export PYTHONPATH=$MAGIC/MagicTools/xdrfile-1.1.4/src/python:$PYTHONPATH
```

or run `source setvars.sh`

3 Using MagiC

3.1 Basic principles of the coarse-graining procedure

In general systematic coarse-graining can be considered as a multi-stage process which leads from a high-resolution model to the low-resolution one (see figure 1).

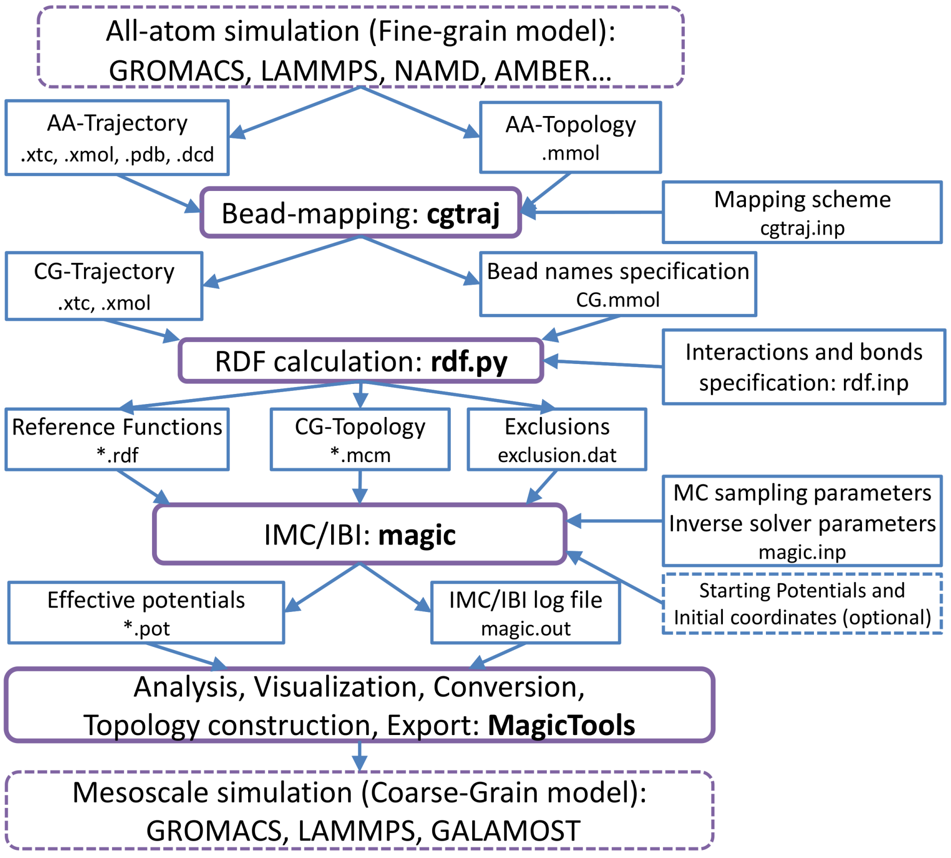


Figure 1: Systematic Coarse-Graining with MagiC: General outline. Blue rectangles denote input/output data; purple rectangles denote data processing procedures. Optional input data and use of external software are marked with dashed frame.

Each step (shown in purple) uses results of the preceding stage output as an input (input/output is shown in blue), and additional input provided by user (rightmost blue blocks).

Six stages can be distinguished:

1. The system of interest is simulated at high resolution, e.g. using Molecular Dynamics with all-atom (AA) force field. Such simulation results in AA trajectory which is supposed to sample the atomistic system well enough. This simulation can be performed by any suitable external molecular dynamics (or Monte Carlo) software.

2. A coarse-grained (CG) trajectory is generated from the atomistic trajectory obtaned during the first stage. This is performed by utility *cgtraj* which is a part of MagiC. It converts AA-trajectory into CG-trajectory, using a user provided mapping scheme which states the correspondence between atomistic and CG representations for every molecular type. This stage results in the coarse-grained trajectory and mass/charge properties of CG-beads stored in molecular description files (`.CG.mmol`).

3. Structural reference distribution functions are calculated by the utility *rdf.py*. Since every distribution function will correspond to an effective potential, at this stage we define all interactions in the CG-model. This includes bead types assignment, definition of pairwise and angle-bending bonds. Based on the bond connectivity, the list of sites excluded from non-bonded interactions is generated. As the result of this stage user gets RDFs containing file (*.rdf), CG molecular topologies (*.mcm) and exclusion definitions (exclusion.dat)

4. The inverse problem is solved by the Inverse Monte Carlo or Iterative Boltzmann Inversion methods. This is the key stage, which is done by a core of the package which is called *magic core*. During this stage, effective potentials between CG sites are iteratively refined to fit the reference RDFs. An extended log-file reports details of every IMC/IBI iteration.

5. Model analysis by the set of post-processing tools *MagicTools*.

   It allows to plot the convergence rate, effective potentials from each iteration, potential corrections at each iteration, intermediate RDFs, etc.

6. Once the effective potentials reproducing the reference RDFs with required precision are obtained, they can be exported by *MagicTools* to an external MD software and used for further simulations of the large scale CG system. At present it supports GROMACS, LAMMPS and GALAMOST, however, extensions to other MD simulation software accepting tabulated potentials can be made easily and smoothly.

Since MagiC is implemented as a set of separate programs, it is possible to perform different tasks at different locations, for example run the most time-consuming part of the calculations, inversion of RDFs (stage 4), on a high performance cluster, and perform analysis on a local desktop computer.

3.2 *CGTRAJ*: Bead Mapping

This is a tool to map/convert a high resolution (all-atom) trajectory to a coarse-grained trajectory.
NB! Molecules in the trajectory shall be kept whole, i.e. bonds shall not be broken when crossing periodic box boundaries.

3.2.1 Input/output files:

Input files:

***Main input** providing the mapping scheme, input trajectory and output files parameters.

***Trajectory** The high-resolution trajectory in one of the following formats:

- **XTC** [1] - Compressed binary trajectory format used in GROMACS and LAMMPS. This is the recommended format.
- **TRR** [1] - Binary trajectory format used in GROMACS.
- **XMOL** - Text-based XYZ trajectory format *.xmol, or a set of numerated files as for option MDYN below.
- **PDB** - Text-based trajectory format *.pdb
- **MDYN** - MDynaMix trajectory binary format. It is usually given as a set of numerated files *.001,*.002, etc.
- **DCD** - CHARMM/NAMD binary trajectory file format: *.dcd

It is assumed that in all cases, the atoms and molecules are arranged in the standard way:

`<molecules of type 1><molecules of type 2>...`

in each molecule type:

`<molecule 1><molecules 2>...`

in each molecule:

`<atom1><atom2>...` (the same atom order must be in all molecules of this type)

***Molecular type descriptions** for each type involved: *.mmol files (optional, see detailed description below).

Output files:

CG trajectory in XMOL or XTC[1] format.

CG molecular types descriptions in .mmol format, but without bonds: *.CG.mmol.

Run: `cgtraj cgtraj.inp`

[1]Requires xdrfile-library

3.2.2 cgtraj.inp: main input file

The file consists of two parts. **The first part** describes the input atomistic trajectory, and the second one describes CG-bead mapping scheme.

Trajectory reading subroutine was inherited from *tranal* utility of MDynaMix, and it has the same syntax. The first part of the input file is written in Fortran NAMELIST format which looks like:

```
$TRAJ
parameter=value(s),
...
$END
```

"TRAJ" is the name of this NAMELIST section. The following parameters shall be defined (* denotes mandatory parameters):

`*FNAME = <file_name>` Name of the trajectory file or a base name of the set of files[1]. The trajectory (except XTC) can be written as a sequence of files `<file_name>.001` , `<file_name>.002` and so on, the largest possible number being `<file_name>.9999`.

`*NFORM = <format>`
where `<format>` is one of:

- `XTC` - XTC compressed binary trajectory format (GROMACS, LAMMPS)
- `TRR` - TRR binary trajectory format (GROMACS)
- `XMOL` - XMOL trajectory. It is assumed, that the commentary (second) line of each configuration is written in the format:
  `(char)  <time>  (char-s) BOX:  <box_x> <box_y> <box_z>` where `(char)` is any character word, `<time>` is time in fs, `<box_x> <box_y> <box_z>` (following after keyword `BOX`) are the box sizes in Å.
- `PDBT` - PDB trajectory as generated by "trjconv" utility of GROMACS simulation package.
- `DCDT` - DCD trajectories generated by NAMD package

`*NTYPES = <value>`
Number of molecule types in the trajectory

`*NAMOL = <name1>, <name2>, ... <name_NTYPES>`
List of molecular type names. It is recommended that files `<name1>.mmol`, `<name2>.mmol`,... describing the molecules are present in the directory defined by `PATHDB`. Format of .mmol files is the same as for MDynaMix program. For analyzing trajectories generated by other programs, .mmol files are still used to provide information about atomic masses and charges. It is however enough to have only the first section of .mmol files containing description of atoms.

The program may work without .mmol files, if parameters `NSPEC` and `NSITS` (see below) are given. In this case, the masses of all atoms are set to 1 and the charges to 0, which will result in definition of CG sites as geometric centers of the atomic groups, and zero charges of CG sites (the later can be manually corrected at the next stage).

*`NSPEC = <n1>, <n2>, ..., <n_NTYPES>`
: Number of molecules of each type (`NTYPES` numbers in total).

`NSITS = <n1>, <n2>, ..., <n_NTYPES>`
: Number of atoms in each molecular type (`NTYPES` numbers in total). This parameter is not necessary if .mmol files for each molecular type are provided.

`PATHDB = <value>`
: Directory with molecular description files (.mmol). Default is the current directory (.)

`NFBEG = <value>`
: Number of the first trajectory file (integer between 0 and 9999)

`NFEND = <value>`
: Number of the last trajectory file (integer between 0 and 9999)

`IPRINT = <value>`
: Defines how much you see in the intermediate output. The final output with analysis of results does not depend on it. Default value is 5.

`BOXL = <x-box-size>`

`BOYL = <y-box-size>`

`BOZL = <z-box-size>`

: define the box size (in A) if it is not present in the trajectory (can be used in case of constant-volume simulation) If information of the box sizes is present in the trajectory, box size parameters from the input file are ignored.

`ISTEP = <value>`
: Specifies that only each `ISTEP`-th configuration from the trajectory is taken for the analysis. Default is 1.

The second part of the file describes CG bead mapping scheme. The whole section starts with keyword: `BeadMapping` and ends with `EndBeadMapping`. Every coarse grain molecular type has to be described in a separate subsection, which starts with tag `CGMolecularType: <CGMolecularTypeName>` and ends with `EndCGMolecularType`. Inside such a section, the parental molecular type name and CG beads definition should be given. The parent's name is defined by the tag `ParentType: <ParentMolecularTypeName>`
CG beads are defined in a one-line-per-bead way, where every line has the following structure:
`<Bead name>:<N of atoms in the bead>:<list of atoms atom1,atom2,...>`,
where list of atoms is a comma separated list of atom numbers according to the *mmol*-file describing parental molecular type. User can also put intervals instead of a single number, e.g. `1-3,5,7-9,10` instead of `1,2,3,5,7,8,9,10`. **NB!** The keywords/tags are not case sensitive, and spaces will be automatically removed from the text.

Once cgtraj is executed, it generates a bead-mapped trajectory (in case of xtc-format, user also gets the last frame saved in `last_frame.xmol`, which is handy for VMD-visualization). Also file named `<CGMolecularTypeName>.CG.mmol` will be created for every defined CG-molecular type.

[1]Only supported for XMOL, PDBT, MDYN and DCDT files

3.2.3 Example: cgtraj.inp

This is an example of the input file, which reads a trajectory in binary XTC format. The system presented in the trajectory, consists of 16 DMPC lipid molecules solved in 1600 water molecules.

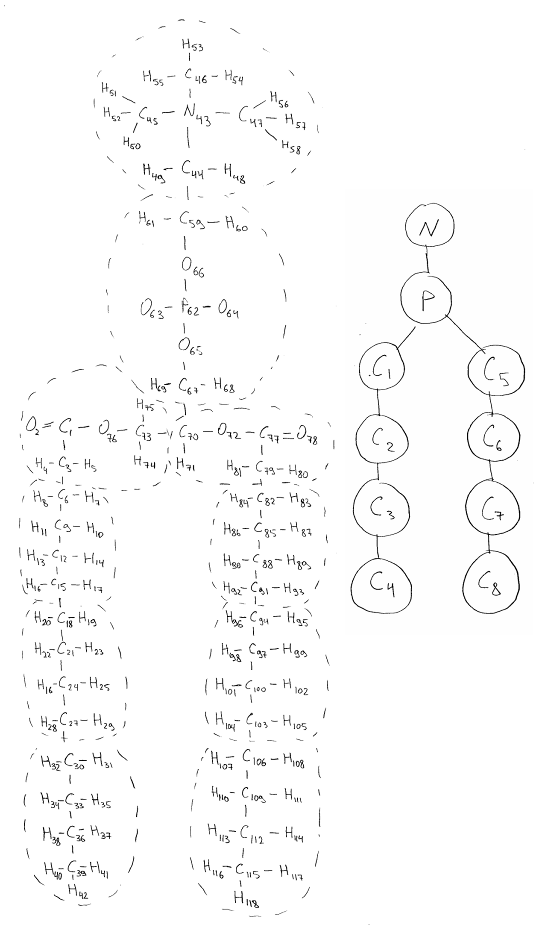


Figure 2: Mapping scheme for DMPC phospholipid, which consists of 118 atoms, into 10-beads CG model. The water is mapped into a single bead.

The bead mapping scheme is shown on figure 2. Note that the resulting CG DMPC molecule has 10 beads, while each water molecule will be represented by a single bead. In order to completely remove the water (implicit solvent) you shall comment out the water-related bead mapping section. The output trajectory format is defined by the extension of the `CGTrajectoryOutputFile` file.

```
&TRAJ
 NFORM='XTC',
 FNAME='dmpc_all_atom_trajectory.xtc',
NTYPES=2,
NAMOL='dmpc_NM','H2O',
 NSPEC=16,1600,
 &END

 BeadMapping
 CGTrajectoryOutputFile:dmpc_cgtraj.xtc
  CGMolecularType:dmpc_NM.CG
    ParentType: dmpc_NM
    N:16:43-58
    P:11:59-69
    C1:9:1-5,73-76
    C2:12:6-17
    C3:12:18-29
    C4:13:30-42
    C5:8:70-81
    C6:12:82-93
    C7:12:94-105
    C8:13:106-118
  EndCGMolecularType
  CGMolecularType:H2O.CG
    ParentType:H2O
    #comment the line below to exclude the water completely
    H2O:3:1, 2, 3
  EndCGMolecularType
EndBeadMapping
```

3.2.4 xmol2xtc Conversion Tool

Simple and handy tool to convert bulky XMOL-trajectories into more compressed XTC-format. It is built on the top of CGTraj. It takes the name of XMOL-file as the only input parameter, and creates corresponding xtc-file. The last frame of the XMOL-file is saved in `last_frame.xmol` to preserve names of the atoms and also to help opening the XTC-file in VMD. The time and box size are read from the second (comment) line of every frame. If only step-number is provided (as in LAMMPS xyz-file), it will be used as time in fs.

Run: `> xmol2xtc trajectory.xmol`

Compile: Enter `$MAGIC/CGTraj` folder and run `make xmol2xtc`

3.3 *rdf.py*: Reference Distribution Functions calculation

This section describes calculation of the structural distribution functions (DF), which will be used as a reference for the effective potentials calculation during the inverse process. Note, that after the previous stage (cgtraj), the generated CG trajectory does not have any information about chemical CG types or bonding, and this information is provided at this stage when CG types and their bonding are defined. Each distribution function defined at this stage results in an individual interaction potential, therefore definition of groups of beads, which belong to a certain DF, is equivalent to definition of specific interactions in the CG system.

Input files:

***Molecular type description** for each type present in the system: **.mmol* or **.mcm*.

***CG trajectory** : **.xmol* or **.xtc*. **NB!** It is assumed that molecules in the trajectory are kept whole against PBC, which shall be the case if you had molecules whole in the original all-atom trajectory.

***Main input** defining RDF calculation parameters, input trajectory, CG-atom/bead types and the list of RDFs to calculate (and beads included in each specific RDF): *rdf.inp*

Output files:

Reference distribution functions **.rdf*. See format details

Coarse-Grain molecular topology **.mcm*. See format details.

Exclusions file *exclusions.dat*. See format details.

Run:

```
rdf.py -i rdf.inp [-np n_cores] [--force]
```

To run in parallel mode, specify the number of cores `n_cores`, and to avoid check of atom names consistency use option `--force`

3.3.1 rdf.inp: main input file

RDF input file consists of several parts: RDF-calculation parameters (`&Parameters`), definition of Coarse-Grain atom types (`&CGTypes`), the list of RDFs to calculate (`&RDFsNB, &RDFsB, &RDFsA`) and optional declaration (`&SameAsBond`) that some bonds in different molecules are equivalent, which means that their distributions are averaged, and they are assigned the same bonded potential in the CG model. The later is a formal way to have the same type of bond in several different molecular types.

RDF calculation parameters: (`&Parameters ... &EndParameters`)
The section describes the input trajectory and defines resolution and cut-off ranges for the reference distribution functions. Note that at this point the reference all-atom trajectory should be already mapped into coarse-grained representation.

The following parameters shall be specified:

`*OutputFile = <filename>` The name of the output file containing a set of calculated RDFs

`*TrajFile = <filename>` The name of the CG-trajectory file. The file format will be detected from the extension, or can be stated explicitly in the parameter `NFORM`

`NFORM = <format>` (Optional) Explicitly specified format of the CG-trajectory. Can be XMOL, TRR or XTC. Detected automatically from the trajectory file extension.

`Step = <value>` (Optional) How often to read frames from the trajectory. Default: 1 (read every frame)

`BeginFile=<value>, EndFile=<value>` (Optional) If the trajectory is split into a number of files enumerated by file name extensions (.001, .002, .003, ...), these parameters specify a range of the files to read.

`*NMType = <value>` Number of molecular types present in the CG-trajectory.

`*NameMType = Type1, Type2, ...  , Type(NMType)` Names of the molecular types. Each type should have a molecular description file (`.CG.mmol`), having the same name as the molecular type.

`*NMolMType = Num1, Num2, ...  ,Num(NMType)` Number of molecules of each molecular type present in the system.

`*RMaxNB=<value>` Cut-off distance (Å) for intermolecular / non-bonded RDFs

`*RMaxB=<value>` Cut-off distance (Å) for intramolecular bond length distributions

`*ResolNB=<value>` Resolution (Å) of the histogram for intermolecular RDF calculation

`NAngleBondsExclude=<value1>,...,<valueNMType>`

`NPairBondsExclude=<value1>,...,<valueNMType>` Atoms having this many pairwise/angle-bending bonds between, will be excluded from non-bonded distributions. User can specify individual value for each molecular type or one value for all of them. Special case: -1 exclude all intramolecular pairs. Default value: 1 (corresponds to exclusion of all intramolecular pairs involved in one of bonded or bending angle interactions).

Bead types: `(&CGTypes, ... , &EndCGTypes)`
Here bead types (CG-atom types) are introduced and beads belonging to each type are specified. This is done by a list of lines having a format `<Name of CG-type>:<NameBead1 NameBead2 NameBead3>`, one line per each type, bead names are space separated. Note that the order of bead type lines will define indexes of the bead types in the `.mcm`-files.

Non-Bonded RDFs `(&RDFsNB, ... , &EndRDFsNB)`
List of reference distribution functions for non-bonded interactions, which are radial distribution functions. For each function a list of bead-pairs (CG atom pairs) involved in the specific interaction shall be provided. It is possible to generate the list automatically between all or some of pairs of bead types using the following commands:

`add: all`
This will generate automatically a list of RDFs which includes all possible RDFs based on pair combinations of CG-atom types. For each pair of CG-atom types a RDF will be determined, which includes all pairs of CG atoms of the specified types, and effective potential for this pair of atom types will be calculated on the next stage. With this option, all possible NB-RDFs will be taken into account. This is the most common regime.

`add: <CGType> -- <CGType>`
Create a list of CG-atom pairs having the given CG-atom types, and include it into calculation of RDFs. This will add a single RDF to the list.

`add: <CGType1> -- <CGType2>: AName1 AName2, AName3 AName4`
Explicitly add pairs of atoms AName1-AName2, AName3-AName4 to the RDF for the given pair of CG-atom types. This is the most precise way of setting the atom-pairs list for a given RDF.

`del: <CGType> -- <CGType>`
Remove a specific RDF (interaction) from the set of RDFs generated up to this line.

`del: <CGType> -- <CGType>: AName1 AName2, AName3 AName4`
Exclude a specific pair of atoms from the RDF for given atom types.

RDFs for Pairwise Bonds `(&RDFsB, ... , &EndRDFsB)`

In this section reference distributions for pairwise bonds (e.g. bond length distributions) are specified. Note that this determines bonding in the CG molecule, and thus has to be specified explicitly.

For each independent pairwise bond type one need to specify the molecular type it belongs to, the relative index of the bond, and list of atom pairs connected by the bonds of this type. This is done in a single line record:
`add: <MolType>: <BondIndex>: <AName1> <AName2>, <AName3> <AName4>`
where `<MolType>` the molecular type, `<BondIndex>` the bond type index in the given molecular type, and pairs `<AName1> <AName2>, <AName3> <AName4>,...` determine CG atoms within the molecule connected by bonds of the `<BondIndex>` bond type.

RDFs for Angle-bending bonds `(&RDFsA, ... , &EndRDFsA)`

In this section reference distributions for angle-bending bonds (e.g. bond angle distribution) are determined. It can be done manually, similarly to specifying pairwise bonds, or deduced automatically by setting an A-bond between every two interconnected pairwise bonds (excluding cases when the end atoms of the angle are already connected by a bond). Note that pairwise bonds shall be set in prior, i.e &RDFsA-section shall appear after &RDFsB-section. The following keywords can be used in this section:

`add: all`
Automatically deduce angle-bending bonds for all molecular types of the system

`add: MolType : all`
Automatically deduce angle-bending bonds in the given molecular type

`add: <MolType>: <BondIndex>: <AName1> <AName2> <AName3>, ..., ...`
Explicitly add triplet (triplets) of atoms to the given angle-bending bond of the given molecular type

`del: MolType : all`
Discard all angle-bending bonds in the given molecular type

`del: MolType : <BondIndex>`
Discard the given A-bond

`del: MolType : <BondIndex>: <AName1> <AName2> <AName3>, ...,`
Remove given atoms from the defined previously A-bond

Same Type of the Bond declarations: (`&SameAsBond ... &EndSameAsBond`)
This section has recently appeared in MagiC as a formal way to describe bonds belonging to different molecular types by the same RDF/potential. The MagicCore was written in assumption that different molecular types must have different bonds. To overcome this limitation, we made possibility to link a "secondary" bond to another "original" bond. In all resulting files this two bonds will appear separately, however they will have same distributions. The linking - records have format:
`OriginalBond = LikedBond1, LinkedBond2,...`
Each bond is specified as `MolecularTypeName:BondNumber`, see example below.

```
&SameAsBond
DMPC.CG:1 = DMPC.CG:3, DMPC.CG:4
DMPC.CG:2 = DMPC2.CG:1
&EndSameAsBond
```

3.3.2 Example: rdf.inp

This is an example of rdf.inp file which defines reference distribution function calculation for a CG lipid model with interactions shown on figure 3.

```
&Parameters
 TrajFile = cgtraj.dmpc16-400ns.xmol
 NMType = 1
 NameMType = dmpc_NM.CG.mmol
 NMolMType = 16
 OutputFile = dmpc16-100aa.rdf
 RMaxNB = 20.
 RMaxB =10.0
 ResolNB =0.1
 ResolB=0.02
 ResolA=1.0
&ENDParameters

&CGTypes
N:N
P:P
CH:C2 C3 C4 C6 C7 C8
CO:C1 C5
&EndCGTypes
```

```
&RDFsNB
Add: all
# Add: N--P
# Add: N--P: N P
# Add: N--CO: N C1, N C5
&EndRDFsNB

&RDFsB
add: dmpc_NM.CG: 1: N P
add: dmpc_NM.CG: 2: P C1, P C5
add: dmpc_NM.CG: 3: C2 C3, C3 C4, C6 C7, C7 C8
add: dmpc_NM.CG: 4: C1 C2, C5 C6
add: dmpc_NM.CG: 5: C1 C5
&EndRDFsB

&RDFsA
Add: all
#add: dmpc_NM.CG: All #Other way - generate all A-bonds for the molecule
#add: dmpc_NM.CG: 6: N P C1 #Example - create N-P-CO A-bond
&EndRDFsA

&SameAsBond
# Empty
&EndSameAsBond
```

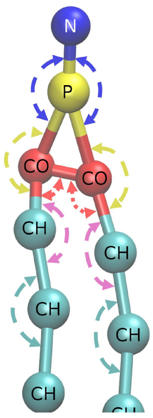


Figure 3: Example: 10-beads CG model of a DMPC lipid. Beads and bonds of the same color have same type; solid lines denote covalent bonds; dashed arrows denote angle bending bonds.

3.4 *MagiC Core*: Inverse Solver IMC/IBI

3.4.1 General description

This is the main part of the whole software package. It performs Metropolis Monte-Carlo sampling of the system described by a trial set of potentials, then compares sampled distribution functions with the reference ones, and introduces a correction to the set of potentials. Then the new iteration starts, now with the corrected set of potentials. The process of inversion of RDFs can be regarded as completed when agreement between sampled and reference distribution functions is reached.

The software automatically analyses the provided input files: molecular descriptions, RDFs and/or potentials files and checks them for consistency. If potentials for some interactions are not provided, they will be deduced from the corresponding RDFs. By default a zero-potential (except the core with zero reference RDF) is used as a starting potential for all non-bonded interactions, and potential of mean force is used for the bonding interactions (pairwise and angular). It is also possible to use potential of mean force as a starting potential for non-bonded interactions (not always suitable for multisite coarse-grained molecules). A user can choose the kind of trial potentials for every interaction in the `.rdf` file using keywords `&InitZero, &InitPMF`. If for some interactions only potential is provided, but the corresponding RDF is missing, this potential will be used for sampling, but excluded from the inverse procedure (i.e. kept fixed). If neither potentials nor RDFs are provided for certain non-bonded interaction, then such interaction will not be used in the MC simulation (which is equivalent that corresponding interaction potential is set to zero). In case if RDF-file is absent, no inverse procedure will be performed and the program will just run a standard MC simulation with the supplied potential. In addition it is also possible to explicitly fix some potentials and do not update them in the inverse procedure using keyword `&Fixed` in the potential file. Other potentials (without keyword `&Fixed`) will be either fit to the corresponding RDFs (standard IMC mode), or the the whole RDF set if variational IMC mode is used.

3.4.2 Input/Output files

Input files:

***Main input** specifying parameters of the Monte-Carlo sampling, inverse solver and input/output files: *magic.inp*

***Topology** for every CG-molecular type: **.mcm*

***Reference distribution functions** : **.rdf*

Starting trial potentials **.pot*. If not provided (or partially provided), the missing trial potential will be deduced from the RDF-file as potential of mean force or zero-potential, depending on the user's choice.

Initial geometry for MC-process in *.xmol or *.xtc (optional). User can provide individual file for every parallel process:
name-of-the-system.p<process-number>.i<iteration-number>.start.xmol
or just a single file with multiple frames, which will be randomly read by all processes.

Exclusion-files: (optional) *.dat Files with non-bonded exclusions specification, one for short-range interactions and another for the electrostatics. If not provided, default exclusions will be used: atoms bonded by pairwise or angle bond will be excluded from both short-range and electrostatic interactions.

External Trajectory: (optional) *.xtc or *.xmol . External trajectory (generated by external MD software) can be used instead of Monte-Carlo sampling, then MagicCore will only update the potentials.

Cross Correlation Matrix (optional) - binary file storing cross-correlation matrix. If it is provided, MagicCore will read the matrix, update it with new values sampled in the current MC-run, and write back the updated matrix to the file. This is useful when you want to split a long simulation into several shorter pieces. NB! The file can become rather large for system with multiple interactions.

Output files:

General output By default it is printed on the screen, but it is recommended to redirect it to a file (see example below)

Log/journal for every parallel process . name-of-the-system.p<process-number>. In serial run it is written to the general output.

Resulting effective potentials (starting potential for current iteration + correction): *name-of-the-system.i<iteration-number>.pot*

Monte-Carlo trajectory of each parallel process: *name-of-the-system.p<process-number>.xmol*

The final snapshot of the system of each parallel process and iteration: *name-of-the-system.i<iteration-number>.p<process-number>.start.xmol* This file can be used as a starting configuration for a next consequent series of MC runs. In case of a parallel MC run, a set of files is produced which are used as starting configurations for each processors at the next iteration.

Cross Correlation Matrix Binary file storing cross-correlation matrix, see above.

Execution:

serial execution: `magic magic.inp > magic.out`

parallel execution: `mpirun -np n_of_proc  magic magic.inp > magic.out`

3.4.3 magic.inp: main input file

In the MagiC-core input file parameters are specified as a set of keywords
`ParameterName = Value`
and generally have self-explanatory names. Warnings or error messages are issued for missing compulsory parameters or inconsistent values. Lists of variables (or vectors) should be comma separated (i.e. BOX = X, Y, Z). Comment lines starting with ! or # or empty lines are ignored. For logical parameters most popular acronyms, such as F, .F., False, FALSE are accepted.

For convenience the parameters are divided into five groups. There is however no need to follow this order, the program accept parameters in any order.

Keywords marked with * are mandatory. New parameters, which have been introduced in version 3, are *emphasized with italic-font.* Old parameter-names inherited from MagiC v1 are denoted by `[ ]` as still supported but not recommended.

System parameters:

NMType* Number of molecular types (species) present in the system.

NameMType* Names of the molecule types present in the system, separated by comma. Every molecule type should have a respective description file(.mcm). For example `NameMolType = H2O, DMPC` defines 2 names: `H2O`-for the first molecule type, and `DMPC` for the second one. The corresponding description files should be named `H2O.mcm` and `DMPC.mcm`.

NMolMType* Number of molecules of each type, written as a comma-separated list, e.g. `NMolMType=392,3` defines system, which consists of 392 molecules of the first type, and 3 molecules of the second type.

LMoveMType Which molecular types are allowed to move in the Monte Carlo simulation. List of comma-separated logical values.
Default `LMOVE = True,.., True` , i.e. all molecules are allowed to move. Frozen molecules coordinates has to be specified in a *.xmol file given in `InputFrozenCoords` parameter.

Epsilon [EPS] Dielectric permittivity constant defining electrostatic interactions in the system. Default: 1.0

TEMP* Temperature of the system, K

Box* [BOXL,BOYL,BOZL] Periodic cell dimensions in Å, separated by comma. The software uses rectangular periodic boundary conditions.

Monte Carlo parameters:

MCSteps* Total number of Monte Carlo steps to be performed on every iteration (including equilibration).

MCStepsEquil* Number of Monte Carlo steps to be performed for the equilibration.

MCStepAtom* Maximum displacement in a Monte Carlo single atom displacement step, Å. To use non-uniform MC stesp (specific to molecule type), one can provide several values: one per each molecule type. Default: 1.0.

MCStepTransMol [MCTRANSSTEP] Maximum displacement in a MC translation of a whole molecule, Å. To use non-uniform MC step (specific to molecule type), one can provide several values: one per each molecule type.
Default: 0.0

MCStepRotMol [MCROTSTEP] Maximum degree of MC rotation of the whole molecule, deg. To use non-uniform MC step (specific to molecule type), one can provide several values: one per each molecule type. Default 0.0

iMCStepTransMol [ITRANS] How often (in terms of MC steps) to perform translation of a randomly chosen molecule. Default: 0, i.e. never

iMCStepRotMol [IROT] How often (in terms of MC step) to perform rotation of a randomly chosen molecule. Default: 0, i.e. never

iCalcEnergy* [IOUT] How often to recalculate the total energy and write energies and pressure to the log-file. If the difference in total energy before and after the recalculation is larger than $0.01k_BT$, a warning message will be given. This procedure is computationally expensive (order N^2), and should not be performed too often.

RCutEl* [RECUT] Real space cutoff for electrostatic energy in real-space part of the Ewald sum. It is recommended to set it equal to cutoff radius of RDFs and short-range interactions, but in some cases other choices can be reasonable. When the periodic box size is big, calculation of reciprocal-space part of the Ewald sum may become performance limiting and then increase of RCutEl may improve the performance. It can be seen on the timing report, printed after every inverse iteration.

AF, FQ Ewald summation parameters: The electrostatic energy in Ewald method can be expressed as

$$U_{el} = \frac{1}{2V} \sum_{k\neq 0}^{k^2<k^2_{cut}} \frac{4\pi}{k^2} |\rho(k)|^2 \exp{(-\frac{k^2}{4\alpha^2})} - \frac{\alpha}{\sqrt{\pi}} \sum_{i=1}^{N} q_i^2 + \frac{1}{2} \sum_{\substack{i\neq j \\ r_{ij}<r_{cut}}}^{N} \frac{q_i q_j \, erfc\,(\alpha r_{ij})}{r_{ij}} \tag{1}$$

where, $\alpha = \frac{AF}{r_{cut}}$, and $k^2_{cut} = 4\alpha^2\, FQ$. In other words, the precision or the first sum is defined by $\exp(FQ)$, while accuracy of the third sum is defined by $erfc(AF)$. Default: `AF=3., FQ=9.0`

RandomSeed [NRS] Initial seed for the random number generator.

KeepStructure [LCRDPass] Define if the final structure of the previous inverse iteration shall be used as starting for the consequent iteration.
Default: False

NMCAutoAdjust How many auto adjustments of MCstep-size to perform during equilibration phase? Default: 0 (no auto-adjustment)

MCStepAtomAR Desired acceptance ratio for MC atom displacement step (can be either one value, or individual value for every molecule type).
Default: 0.5

MCStepTransMolAR Desired acceptance ratio for MC molecule translation step (can be either one value, or individual value for every molecule type).
Default: 0.5

MCStepRotMolAR Desired acceptance ratio for MC molecule rotation step (can be either one value, or individual value for every molecule type).
Default: 0.5

Inverse procedure parameters:

InverseMode The inverse solver mode: IBI - iterative Boltzmann Inversion, IMC - Inverse Monte Carlo, VIMC (also called NG for Newton-Gauss) - Variational IMC, EVR - Eigen Values Regularization. Default: IMC

UseIMC [LIMC] Outdated. Inverse solver selection. If true, the Inverse Monte Carlo method is used, otherwise iterative Boltzmann inversion is used.
Default: True (IMC)

NIter* [IREPT] Number of inverse iterations to perform. Default: 1

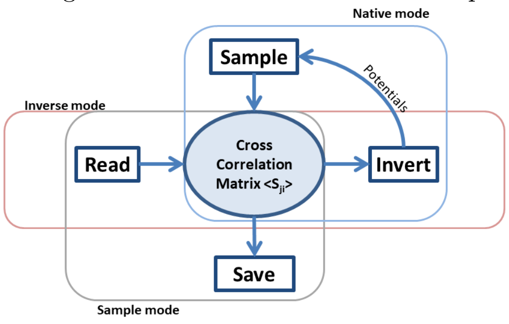
Mode Specify mode of running MagiC Core:

Native (default). This is the standard iterative mode. Perform MC-sampling, gather statistics, write the cross-correlation to file (if provided), invert potentials, then proceed to the next iteration.

Sample. Read cross-correlation matrix from file (if provided), perform MC-sampling, update the matrix and write it to the file.

Inverse. Read cross-correlation matrix from file and invert it providing a new set of the effective potentials.

The Sample and Inverse modes are designed for collecting more sampled date in conditions of limited CPU-time or limited number of cores, allowing user to gain statistics in chunks and then perform the inverse proce-



dure.

RegP Regularization parameter for the potential correction. This parameter defines the relative weight of correction, and has a value between 0 and 1. In case of instability (each next iteration returns larger deviation from the reference RDF), value of REGP should be decreased. Decreasing REGP is also advisable if correction to the potential at each iteration exceeds treshold value (default 2 k_BT). REGP can be increased back closer to 1 if the iteration process is stable and correction to the potential at each iteration is small. Default: 1.0

XREG Parameter of the EVR mode. Values should be between 0 an 1. In the EVR mode the inversion matrix is decomposed over eigen vectors $\vec{J_i}$ with eigen values j^{-1}:

$$\hat{\mathbf{J}}^{-1} = \sum_i |\vec{J_i} > j_i^{-1} < \vec{J_i}|$$

terms of this sum corresponding to eigen values which are less then the one determined by parameter YREG are scaled by factor $(j_i/j_Y)^{1-x}$ where j_Y is determined by paremeter YREG and x is given by XREG. Note that $x = 1$ corresponds to the standard IMC mode.

YREG Determines cut-off eigen value, as the fraction of the total number of eigenvalues counting from the largest eigen value. Contributions to the inversion matrix from eigen vectors having eigen value less than specified by YREG are scaled according to the parameter XREG. YREG=1 is equivalent to the standard IMC mode

iAverage* [IAV] How often to compute averages over the system. Since computation of the averages (RDFs and cross-correlations) involves calculation of distances between all pairs of atom, this procedure is rather expensive, and should not be performed too often. The recommended value is of the order of number of CG atoms in the system The averaging starts after the equilibration, i.e. when first MCStepsEquil steps have passed.

MaxPotCor [DPOTM] Maximal change of potential value at every point during correction procedure, given in k_BT units. Default: 2.0

MaxRelDif [RTM] Parameter limiting maximum relative difference between reference and resulting averages. Default: 10.0

factor_NG Power for the weights in the NG optimization `[0.0-1.0]`. Default: 0.0

PotRcut (used only in variational IMC mode): update potentials only at distances below PorRcut value (in $\AA$), while fitting RDF in the whole range. Is useful if e.g. there are physical arguments that there should be no interaction at distances above PorRcut, but RDFs are still structured at long distances.

iPotCorrCheck How often to perform potential correction check. The program gathers accumulated statistics from all the processes, and then calculates sampled distribution functions and potential corrections. However, this corrections are not applied to the actual interaction potential, but just printed to the log file. This allows user to analyze how well both distribution functions and potential corrections are converged after given number of MC steps of an inverse iteration. The checks are performed after equilibration. Default 0, i.e. no check at all.

ProhibPotLevel [POTCUT] Prohibiting potential level. A high value of potential in kJ/mol to define a core region of the potentials at distances where the corresponding RDFs are zero, to avoid MC steps leading to such distances. Default: 1000.

ExternalTraj File with external CG trajectory, which will be used instead of usual MC sampling. You have to set MCSteps equal to number of frames in the trajectory. NIters will be automatically set to 1. The trajectory shall keep molecules whole against PBC.

Input-Output parameters:

Output* [BASEOUTFILENAME] Prefix name of the system (filename template) to use for writing output files. All names of output files will begin with the given prefix.

VerboseLevel [IPRINT] Verbosity level of the log-file. 1-minimum level, 10 - maximum level. Default: 5.

WriteTraj [ITR] How often (in terms of MC steps) to write current geometry to the trajectory file. Default: 0 - do not write.

InputRDF* [FILRDF] Input file with reference distribution functions. Required for inverse procedure, otherwise only a direct MC simulation will be performed. But why would you run it then?

InputPotential [FILPOT] Input file with a set of trial potentials.

InputStartCoords ***[, Nconf]*** Name of the input file (or prefix for a set of files), followed by an optional integer parameter Nconf. If Nconf is given, MagiC will randomly choose one of the first Nconf, frames from the file.

This is a convenient way to provide reasonably equilibrated starting coordinates, however, being free from over-fitting the potentials to some specific starting configuration.

InputFrozenCoords [FCRD] Name of a single *.xmol file with starting coordinates of all frozen molecules in the system. It will be used to define starting location of the frozen species for every inverse iteration. The file is almost the same as start.xmol, but it does not contains moving molecules/atoms.

DumpLastConf [LXMOL] If true, program dumps the last configuration of MC process in file (or set of files) with ".start.xmol" extension. It is done after every inverse iteration on every parallel process. Default: False - do not dump. In case of parallel execution, output filenames have extensions `<Output>.i<iteration>.p<process>.start.xmol`

CrossCorrFile File to read/write cross-correlation matrix sampled during MC-simulation (in parallel run, the matrix is first gathered from all processes and saved to the file).

exclusionSR, exclusionEL - Files with non-bonded exclusions specification, one for short-range interactions and another for the electrostatics. If not provided, default exclusions will be used: atoms bonded by pairwise or angle bond will be excluded from both short-range and electrostatic interactions.

3.4.4 Example: magic.inp

Here is an example of the input file, representing the system of 2 molecule types (MT1 and MT2), having 10 molecules of each. The program shall read RDF from file MT1MT2.rdf and the IMC will be used for inversion, making 10 iterations in total. As the potentials are not provided, the trial potentials will be deduced from RDFs according to the settings in the RDF file 5 millions MC steps are to be made at every inverse iteration, and half of them are for equilibration.

```
NMType = 2
NameMType = MT1, MT2
NMolMType = 10, 10
LMoveMType = TRUE, T
Box = 15.0, 15.0, 15.0
Epsilon = 1.0
TEMP=303.

MCSteps = 5000000,
MCStepsEquil = 2500000,
MCStepAtom = 0.2, 0.2
MCStepTransMol = 1.0
MCStepRotMol = 0.2
iMCStepTransMol = 50
iMCStepRotMol = 50
iCalcEnergy = 100
RCutEl=0.7
AF = 2.6
FQ = 8.0
```

```
ProhibPotLevel=1000.0
RandomSeed=51
NMCAutoAdjust = 0
MCStepAtomAR = 0.5, 0.5
MCStepRotMolAR =0.5
MCStepRotMolAR =0.5

InvMode = IMC
NPointsNB = 70
NIter=10
IAverage=50
REGP = 0.1,
MaxPotCor=2.0
KeepStructure=False
MaxRelDif=10.0
iPotCorCheck = 250000

VerboseLevel=5
InputRDF= MT1MT2.rdf
InputStartCoords = start, 0
Output = 01.111.MT1MT2
DumpLastConf = .false.
WriteTraj = 100000
```

3.4.5 Conditional compilation: Increase the performance

MagiC core has some additional features, which can be enabled or disabled during compilation. For this edit line with compiler preprocessor directives features: `$(eval FEATURES+=-DPBC_MULT -Dtimer_on -Dpressure)` in `MAGIC/magic/Makefile` The following features are available:

-DPBC_MULT Apply periodic boundary conditions multiple times when calculating distances. Default: enabled.

-Dtimer_on Turn on timing report for the code execution. Disabling this feature sometimes can give about 10% speedup. Default: enabled

-Dpressure Turn on calculation of virial sum and pressure. Disabling this feature can give about 10% speedup. Default: disabled

-Dxdr Turn on support for xtc-trajectory file format. Requires compiled XDR-FILE library. Default: enabled for GNU Fortran, disabled for Intel Fortran.

-Ddebug Turn on trajectory logging: Store every accepted trajectory frame since the most recent total energy recalculation (iCalcEnergy). In case of crash with "Deviation in coordinates" error, the trajectory is written to xmol.file. Helpful for debugging errors in MC-sampling. Default: disabled

3.5 *MagicTools*: Juggle with MagiC's data

This part of the package is an python-based library (set of procedures), which is called in a python interpreter. We recommend to use *jupyter-notebook* as an interpreter, however, the standard python should also work for simple operations. Once you have started the interpretor, you need to import the module. To do this type: `import MagicTools`, if no error message appeared, the import worked correctly.

Analysis usually include the following phases: Reading the data from output files; Saving/Writing the data to files, Plotting the data; Numerical analysis/evaluation of the data; Coarse-grain topology generation, Export of the data to external MD packages.

We assume that user is already familiar with the basic Python syntax. If not yet, there are a lot of free introduction courses to Python available on the web, and we highly recommend to familiarize yourself with this versatile and easy-to-use language.

3.5.1 MagicTools data structures introduction

Before we discuss how these actions can be taken by MagicTools, we shall say few word about the data-structures which we will routinely use. They are representing two types of objects: tabulated functions, such as RDF or potential and molecular system topology representation.

Distribution functions and potentials: The smallest size object here is called **DF** (named after distribution function), it represents a single tabulated function, such as a particular RDF, potential or potential correction.

A set of DF-objects related to the same system is represented by so called **DFset-object**. For example, a single *.rdf-file corresponds to a DFset. It shall not be mixed with **list of DF-objects** (DFs list), such as [DF1, DF2, ... DF_N], as in such list DFs are not necessary related to the same system.

The most complex structure is a **list of DFset-objects**, which is generally used for comparison/plotting of DFsets, such as comparing set of the reference RDFs with set of RDFs sampled in the IMC. Typical example of DFset-list is a list of RDFs sampled at different iterations of IMC.

Topology-related structures Since molecular system coarse-grain topology can be rather complex, we have implemented a number of corresponding object-classes. Here we briefly mention them in a top-down order.

System - Top level object representing the whole molecular system

MolType - Molecular type topology representation

Molecule - Molecule instance representation

BondType - Group of bonds (pairwise or angle-bending) described by the same interactions potential

Bond - Bond instance

AtomType - Atom type representation

Atom - Atom instance

3.5.2 Reading the data

MagicTools can read data from several file types, used in MagiC: RDF and potential files ***.rdf, *.pot** and the MagiC core log-file **magic.out**. This is done by procedures: `ReadRDF`, `ReadPot` and `ReadMagiC`, respectively.

Example:

```
import MagicTools as MT
RDFs_ref=MT.ReadRDF('MT1MT2.rdf')
Pots=MT.ReadPot('01.MT1MT2.i010.pot', quiet=True)
RDFs_smpl=MT.ReadMagiC('01.magic.out', iters=(1,2,10))
RDFs_smpl_mult = MT.ReadMagiC(['01.magic.out','02.magic.out'])
```

The first line imports the library, while the other lines are showing how to call the reading procedures. The procedures put the data into specified variables: `RDFs_ref`, `Pots`, `RDFs_smpl`. The first two variables are instances of DFset, and the third variable, resulting from reading MagiC core output is a list of DFsets, where every element of the list is a set of RDFs sampled while given iteration, as specified in parameter `iters`. The last line shows example of how to read results from several files. `ReadMagiC`, is not limited to reading just RDFs, but it can also extract potentials, corrected potentials, corrections applied at an iteration, reference RDFs. Check the procedure references for more details.

3.5.3 Plotting and Inspecting the data

After the data is imported, it can be visualized. We have reduced the plotting to just two methods: OnePlot, which puts all given DF-objects on a single figure, and MultPlot, which groups DF-objects by the interaction type (or bond type) and then plot each group on an individual figure.

The MultPlot method replaces obsolete `PlotAllDFs` and is designed to plot several DFsets simultaneously, such as if one needs to compare RDFs sampled on different iterations.

Examples:

- Plot the set of RDFs, with one curve per plot:
  `MT.MultPlot(RDFs_ref)`

- Plot a list of sets of RDFs sampled at different iterations:
  `MT.MultPlot(RDFs_smpl)`
  in this case all the RDFs will be automatically grouped by the interaction they refer to.

- Plot together a list of DFsets (e.g. sampled RDFs) and an additional DFset (e.g. reference RDFs)
  `MT.MultPlot(RDFs_smpl+[RDFs_ref])`
  here `[RDFs_ref]` constructs a list with a single DFset in it, and `+` operation concatenates this list with `RDFs_smpl` which is list of DFsets.

- Plot a single function (RDF or potential) from the set:
  `MT.OnePlot(RDFs_ref[0])`

In addition to plotting of already imported data, one can make a quick inspection of the MagiC core log-file using following procedures: `Deviation` plots total deviation between the set of reference and set of sampled distribution functions at every iteration. `HeatMap` Visualize IMC convergence by drawing a heatmap of interaction-specific RDF deviations. Procedure `AnalyzeIMCOuput` reads and plots reference and resulting DFs obtained in inverse procedure. They both require magic's output file (or list of files) as input.

3.5.4 Numerical analysis and processing of the potentials

MagicTools has several procedures performing simple analysis and processing of the effective potentials resulted from MagiC core:

`TotalPots` returns a set of total potentials, where the electrostatic contributions are added to the short-range intermolecular potentials, which are calculated by the MagiC core.

`GetOptEpsilon` calculates the optimal value of the dielectric permittivity, which provides the fastest decay of short-range intermolecular potentials in the tail region.

`PotsEpsCorrection` creates a new set of potentials, where all non-bonded short range potentials are corrected to correspond to the new value of dielectric permittivity.

`PotsPressCorr` creates a new set of potentials, where all non-bonded short range potentials get added a decaying linear term, which suppose to improve total pressure in the target system.

`PotsExtendTailRange` extend the range of the NB potentials.

`Average` Averages the given list of RDFs/potentials-sets into a single set of RDFs/potentials.

`DFset.Write` Writes the set or RDFs/potentials to file.

`SaveDFsAsText` saves a distribution function from a given list to a separate text-file for subsequent plotting with external software.

3.5.5 Exporting topologies and potentials

Preparing a set of input files for particular MD simulation package is quite a challenge even for an experienced user, especially in case of custom coarse-grain models, having no support of well established force-fields and all-atom structure generating tools. In order to facilitate preparation of input files, MagicTools provides a number of exporting procedures. At present moment we support export to the following packages: LAMMPS, GROMACS, GALAMOST.

The export procedure consists of two semi-independent steps: Export of the potentials and export of the topology. The first step is addressed by the potential exporting procedure PotsExport, please also take a look on the export procedure details in section 4.4.5. The second step is performed by one of three topology exporting procedures: LAMMPSTopology, GromacsTopology and GALAMOSTTopology. These routines have very similar interface. They internally create instances of the system and then call one of the methods WriteLAMMPSData, WriteGromacsTopology, WriteGALAMOSTxml of the System class.

Since each MD package has it's own file format, we suggest the user to check the documentation for the desired package, to understand the structure of topology files produced by MagicTools.

LAMMPS topology consists of two files: The first one is `LAMMPS.data`, which stores most of the topology and can be opened with VMD, using `topo readlammpsdata LAMMPS.data` command. The second one is `LAMMPS.data.run.inc`, which assigns potential table files to all corresponding interactions. The only missing file (beside tabulated potentials, generated above) is the input file with the simulation parameters. We recommend to check out Tutorials to see some working examples.

GROMACS requires five types of files: the topology file (.top), the starting geometry (.gro, .pdb) and the index file (.ndx), the simulation parameters (.mdp) and the tabulated potentials (.xvg). GromacsTopology creates the topology file and the starting structure. Moreover, the corresponding .gro-file has atom types used instead of atom names, in order to simplify generation of the index file. The index file shall have a separate group for each atom type, and the group shall have the same name as the corresponding atom type. Use Gromacs tool `gmx make_ndx -f geometry.gro -o index.ndx` to generate the file. The user has to manually add all atom types and pairs of atom types to the .mdp-file parameters `energygrps` and `energygrp_table`, correspondingly. These values are displayed in the output of the potential export routine, so it only takes a copy-paste to add them to the mdp-file.

It is also possible to use `xmol2gro` to convert `.xmol` structure file into `.gro` format. Another workaround is to use VMD for opening `.xmol` and saving it in `.pdb` format.

GALAMOST requires 5 files: The execution script *.gala, the topology file in HOOMD Blue xml format, the tabulated potential assignment file `tables.inc.py`, and two exclusions files `exclusionsEL.inc.py` and `exclusionsSR.inc.py`, specifying exclusions for electrostatic and short-range interactions.

GALAMOSTTopology creates all of these files, except the execution script, which describes the protocol of the simulation and shall be provided by the user. Note that the files `tables.inc.py, exclusionsEL.inc.py and exclusionsSR.inc.py` have to be included into the execution script (see Tutorials for actual examples).

3.5.6 File conversion utilities

MagicTools provides few file conversion utilities: `tpr2mmol`, `xmol2gro`, `gro2xmol`, `xmol2gro`, `dcd2xtc`

In addition to this, we recommend to use python library MDtraj and it's command-line interface called `mdconvert`, which is versatile file conversion tool for almost any kind of MD trajectory.

Another useful tool, which is related to trajectory files is `GetStartConfs`, which makes a set of starting configurations for MagiC core from existing bead mapped trajectory.

4 *MagicTools* procedures reference:

Reading: `ReadRDF`, `ReadPot`, `ReadMagiC`
Plotting and inspecting: `MultPlot`, `OnePlot`, `Deviation`, `AnalyzeIMCOuput`
Analysis, processing and saving: `TotalPots`, `PotsEpsCorrection`, `GetOptEpsilon`, `PotsPressCorr` `Average` DFset.Write `SaveDFsAsText`,
Exporting: `PotsExport`, `GromacsTopology`, `LAMMPSTopology`, `GALAMOSTTopology`,
Converting: `tpr2mmol`, `xmol2gro`, `gro2xmol`, `xmol2gro`, `dcd2xtc`

4.1 Reading MagiC data files

4.1.1 ReadRDF(ifile)

Read set of RDFs from the given `.rdf` file.

Parameters:

ifile (str) Name of the file

Name (str) Name of the DFset, used for annotations when plotted.

quiet (bool) If to suppress output. Default False

check (bool) If to check that bonded RDF are normalized to 1.0]

Example:

```
rdf_ref = MT.ReadRDF('DMPC-Chol.rdf', Name='Reference RDF', quiet=True)
```

4.1.2 ReadPot(ifile)

Read set of potentials from a `.pot` file

Parameters:

ifile (str) Name of the file

Name (str) Name of the DFset, used for annotations when plotted

quiet (bool) If to suppress output. Default False

Ucut (float) Height of the hard repulsive core at r=0 in kJ/mol

Examples:

```
pot = MT.ReadPot('03.Chol-DMPC.i010.pot', quiet=True)
pot2 = MT.ReadPot('01.MT1MT2.i010.pot', Ucut=5000.0)
```

4.1.3 ReadMagiC()

Reads sets of desired functions from MagiC core output file. Main function for reading data from MagiC.Core log-files. It reads sets of functions as specified by DFType-parameter from all files listed in `ifile` and returns a list of DFset.

Parameters:

ifile The file or list of files to read.

DFType Which function to read, five options are available:

- 'RDF': sampled RDFs,
- 'RDFref': reference RDF,
- 'Pot': potential used at the iteration,
- 'PotNew': potential generated after the iteration,
- 'PotCorr': potential correction applied after the iteration

iters : Which iterations to read. Default None, read all iterations. Ex: "iters=(1,2,3,4)"

test : If True, read results of the intermediate convergence tests

quiet : Suppress output, Default False

These parameters are rarely used:

mcmfile : (optional, autodetected) List of molecular-topology files to read AtomTypes and bond definitions.

PairNamesList : (optional, autodetected) List of pairs of atomic names to search in output file. It consists of three sublists: non-bonded pair interactions, bonded pair interactions, bending angle (1-3) bond interactions.

Returns: List of DFset objects: [DFset1, DFset2, ...]

Examples:
1. Fully automatic: Read RDFs for all pairs and bonds and from all iterations.
Autodetect mcmfiles.
```
RDFs = MT.ReadMagiC(['01.magic.out', '02.magic.out'], DFType='RDF', quiet=True)
```
2. Reading potentials on iteration 1,2,3,
```
Pot=MT.ReadMagiC('03.magic.out', iters=(1,2,3), DFType='Pot',PairNamesList=[['N-N','N-P'],
```
3. Reading corrections to the potentials applied on the iteration 5, and specify
the mcmfile
```
PotCorr=MT.ReadMagiC('03.magic.out', iters=(5), DFType='PotCorr',
    mcmfile='dmpc_NM.CG.mcm')
```

4.2 Visualization and inspection of the data

4.2.1 OnePlot(DFs_list)

Plot all functions from the provided list of DFs (or DFset) on a single figure.

Parameters:

DFs_list - set/list of functions (DF-objects) or even a single DF.

figsize (x,y) - size of the plot in inches

dpi resolution of the plot

hardcopy if true, do not show the plot, just save it to eps.

outfile name of the file to save the plot

title the plot's title. if None, it will be autogenerated using provided title_template string.

title_template template to generate the title. Default is 'Kind.Type.Name.DFsetName'

legend_template template to generate legend for each function. Default is 'Kind.Type.Name.DFsetName'

y_lim tuple(y_min, y_max) specify range for y-axis

x_lim tuple(x_min, x_max) specify range for x-axis

legend_fontsize Font size in legend

title_fontsize Font size in title

legend_transparency Transparency of the legend's background

xlabel label for X axis

ylabel label for Y axis

xylabel_fontsize Font size of axis labels

Examples:

```
MT.OnePlot(singleDF)
MT.OnePlot([DF1, DF2, DF3])
MT.OnePlot(DFset)
MT.OnePlot([DFset1, DFset2])
MT.OnePlot(DFset, figsize=(10, 7), dpi=80,
      hardcopy=True, outfile='plot.eps',
      title='SomeTitle',
      legend_template = 'Name.DFsetName',
      legend_fontsize=14, title_fontsize=18,
      xlabel='Distance', ylabel='RDF', xylabel_fontsize=12))
```

4.2.2 MultPlot(input, coinciding=True, atonce=False, show_sameasbond=False, *args, **kwargs)

Plot distribution functions from the list of DFset grouped by the type of interacting atoms or bond number. Also can be used as universal plotting interface for DF, list(DF), DFset or list(DFset).

input DF-objects (list of DFsets, DFset, list of DFs) to be plotted

coinciding if true - only plot similar DFs that are present in all DFsets of the list, otherwise plot depending on existence of the function

atonce Plot all DFs from the list on a single figure

show_sameasbond Plot bond-related functions which are linked to other functions, by default - false] just plot the original function

***args** Arguments for OnePlot

****kwargs** Optional arguments for OnePlot - figsize=(10, 7), dpi=80, hardcopy=False, outfile=None, title=None, multiplot=False

Examples:

```
MT.MultPlot([DF1, DF2, DF3], atonce=True)
MT.MultPlot([DFset1, DFset2], conciding=True)
MT.MultPlot([DFset1, DFset2], figsize=(10, 7), dpi=80,
      hardcopy=True, outfile='plot.eps',
      title='SomeTitle',
      legend_template = 'Name.DFsetName',
      legend_fontsize=14, title_fontsize=18,
      xlabel='Distance', ylabel='RDF', xylabel_fontsize=12))
```

4.2.3 HeatMap(refDFset, otherDFsets, hardcopy=False, outfile=None, force=False, **kwargs)

Visualize IMC convergence by drawing an interaction-specific RDF deviation HeatMap

Each line represents an interactions and columns represent the iterations. Reported numbers are distances between the reference RDF and corresponding sampled RDF for the particular interaction.

Parameters:

refDFset Reference RDFs

otherDFsets list of RDFs sampled at different IMC iterations

hardcopy If to save the plot to png-file

outfile File to save the plot

force Produce the heatmap even if elements of otherDFset do not match refDFset

****kwargs** Keyword arguments to pass to Seaborn.heatmap()-backend

Examples:

```
rdfs = MT.ReadMagiC('01.magic.out', quiet=True) # Read the list of sampled DFsets
rdf_ref = MT.ReadRDF('1DNA-K.full.rdf', quiet=True) # Read the reference DFset
MT.HeatMap(rdf_ref, rdfs, annot=True, hardcopy=True, outfile="1DNA-K.full.rdf.eps")
```

4.2.4 Deviation(filename, hardcopy=False, returnarrays=False, testpoints=False, outfile='deviation.eps')

Analyze the output file **filename*** (or list of files) produced by the MagiC core and plot deviation between the set of reference distribution functions and sampled distribution function obtained on every iteration of the inverse procedure. Two deviations are calculated:
$\Delta S \sim [\sum_{r_j=0}^{r_j=r_{max}} (S_{iter}(r_j) - S_{ref}(r_j))^2]^{0.5}$ and
$\Delta RDF \sim [\sum_{r_j=0}^{r_j=r_{max}} (g_{iter}(r_j) - g_{ref}(r_j))^2]^{0.5}$
If an intermediate convergence test has been performed during inverse procedure, results of the test are also plotted.

filename* - name of the magic output file or list of such names (mandatory argument).

hardcopy - if the plot should be saved to a .eps file (optional argument). Default - no.

outfile - file to save the plot, if hardcopy=True

testpoints - if the points sampled in intermediate convergence tests shall be also plotted. Default - no.

returnarrays - if true, the procedure returns numpy arrays with iteration number and deviation values.

Examples:

```
MT.Deviation('01.magic.out', testpoints=True)
MT.Deviation(['01.magic.out','02.magic.out'],hardcopy=True)
```

4.2.5 AnalyzeIMCOuput(filename, DFType='RDF', iters=None, hardcopy=True, mcmfile=None, PairNamesList=None)

Analyzes output file produced by MagiC and plot resulting functions of interest:

Parameters:

filename (str) File to read

DFType Which function to read, same as for ReadMagiC

iters Which iteration(s) to read. If nothing mentioned all iterations will be extracted.

test If True, read results of the intermediate convergence tests

hardcopy Do not show plots, save them to files instead.

****kwargs** Arguments for MultPlot/OnePlot

Example:

```
MT.AnalyzeIMCOutput('magic.out', DFType='Pot', iters=(1,2,3), test=False)
```

4.3 Analysis, processing and saving the data

4.3.1 TotalPots(pots, eps, mcmfile)

Creates a set of total potentials by adding electrostatic contribution to short-range intermolecular potentials.

$$U_{tot} = U_{sr} + \frac{q_i * q_j}{4\pi\epsilon\epsilon_0 r_{ij}} \tag{2}$$

Electrostatic part is only applied to the intermolecular potentials, while bond potentials (both pairwise and angular) will be kept the same.

pots* - original set of the short-range potentials

eps* - relative dielectric permittivity ϵ of implicit solvent used in inverse Monte-Carlo

mcmfile* - molecular topology files, required to read the charges of atomic types

Example:

```
pot_DNA_short = MT.ReadPot('DNA.short.sample.pot', Ucut=100000)
pot_DNA_total = MT.TotalPots(pot_DNA_short, 70.0, mcmfile=['DNA.CG.mcm']
```

4.3.2 GetOptEpsilon(pots, eps_old, r1, eps_min=0, eps_max=0, npoints=100, mcmfile=None)

Calculates optimal value of the dielectric permittivity which provides fastest decay of short-range intermolecular potential tales according to the procedure described in: A.A.Mirzoev and A.P.Lyubatsev, Phys.Chem.Chem.Phys., 13, 5722-5727 (2011) DOI: 10.1039/C0CP02397C.

Briefly, the procedure to obtain values of the dielectric permittivity providing fastest decay of short-range potentials set with distance consists in the following. First, we introduce a numerical criteria of a short range potential deviation from zero at large distances:

$$W(U_{sr}^{ij}(r)) = \int_{r_1}^{r_2} \left| r^2 (U_{sr}^{ij}(r)) \right| dr \tag{3}$$

where r^2 factor implies a higher weight of larger distances, r_1 and r_2 are the lower and upper boundaries of the range of distances defining the tail (the r_2 value is taken as the cut-off of RDFs and tabulated effective potentials). The absolute value in the equation is used in order to deal with possible oscillations of the short range part of the potential. From eq. 2, one can write for the short-range part of the potential:

$$W(U_{sr}^{ij}(r)) = \int_{r_1}^{r_2} \left| r^2 (U_{tot}^{ij}(r) - \frac{q_i q_j}{4\pi\varepsilon_0 \varepsilon r}) \right| dr \tag{4}$$

Assume we define the long-range Coulombic potential using another value of permittivity $\varepsilon^\star$. This, according to 2, introduces a new short-range potential as:

$$U_{sr}^{\star\, ij} = U_{sr}^{ij}(r) + \frac{q_i q_j}{4\pi\varepsilon_0 r}(\frac{1}{\varepsilon} - \frac{1}{\varepsilon^\star}) \tag{5}$$

Now we shell find the optimal $\varepsilon^\star$, which produces the fastest decay of all three short range potentials according to criteria defined by eq. 3. We minimize the sum:

$$W(system) = \sum_{i,j} W(U_{sr}^{\star\, ij}(r)) = \sum_{i,j} [U_{sr}^{ij}(r) + \frac{q_i q_j}{4\pi\varepsilon_0 r}(\frac{1}{\varepsilon} - \frac{1}{\varepsilon^\star})] \tag{6}$$

by varying $\varepsilon^\star$. The optimal value of $\varepsilon^\star$ can be considered as effective dielectric permittivity corresponding to the given thermodynamic conditions (temperature, concentration).

pots* - set of potentials to analyze (mandatory argument) NB! The dielectric permittivity value calculation only takes intermolecular potentials into account skipping bonding potentials.

eps_old* - dielectric permittivity used in inverse MC calculation (mandatory argument)

r1* - distance where tail range begins, Å(mandatory argument)

eps_min, eps_max - range of values for the search of ϵ_{opt} (optional argument). By default eps_min=0, eps_max=2*eps_old

npoints - number of points in a mesh to be used for the search, e.g. accuracy of the search is equal to $\frac{\epsilon_{max}-\epsilon_{min}}{npoints}$

mcmfile - molecular description file (or list of files) providing charges for bead/CG-atom types. Required if the potential was read from .pot file rather than from MagiC core log file.

Example:
```
eps_opt = MT.GetOptEpsilon(pots, eps_old=70.0, r1=15, eps_min=50,
                eps_max=100, mcmfile='dmpc.mcm')
```

4.3.3 PotsEpsCorrection(pots, eps_old, eps_new, mcmfile=None)

Creates a new set of potentials, where intermolecular potentials of the given DFset are adjusted to the new value of the dielectric permittivity according to eq.5. Intramolecular (bond) potentials are kept untouched.

pots* - set of effective potentials without electrostatic contributions, as provided by MagiC.Core

eps_old* - original value of the dielectric permittivity used in inverse MC calculation (mandatory argument)

eps_new* - new value of dielectric permittivity.

mcmfile - molecular topology files, required to read charges of atom-types, e.g. `['Na.mcm', 'Cl.mcm']`

Example:
```
newpots=MT.PotsEpsCorrection(Pots, eps_old=70, eps_new=100, mcmfile='dmpc.mcm')
```

4.3.4 PotsPressCorr(pots,Ucorr0)

Creates a new set of short-range potentials by adding a decaying linear term to each intermolecular potential in the set. Such a correction suppose to improve reproduction of a correct pressure in the large scale CG simulation. Intramolecular potentials are kept untouched. Correction term is linear and has value of U_{corr0} at point r=0, and value of 0 at $r = r_{max}$, e.g. $U_{corr}(r) = U_{corr0} \cdot (1 - \frac{r}{r_{max}})$

pots* - Original set of the potentials NB! The correction only affects intermolecular potentials.

Ucorr0* - Magnitude of the correction, kJ/mol

Example:
```
newpots=MT.PotsPressCorr(pots,0.5)
```

4.3.5 PotsExtendTailRange(pots, RcutNB)

Extend the range of NB potentials in the DFset up to RcutNB (Inplace, i.e. no new set is returned)

Parameters:

pots* set of potentials

RcutNB* the upper range to extend the potentials.

Example:

```
pot_short = MT.ReadPot('DNA.short.sample.pot', Ucut=100000)
MT.PotsExtendTailRange(pot_short, RcutNB=10)
```

4.3.6 Average(listofDFset, **kwargs)

Averages the given list of DFset objects into a single DFset, each function of which is an average of corresponding functions from all DFsets of the given list

Parameters:

listofDFset list of DFset-objects to average

force (bool) force the averaging of DFs even if they are not alike (False)

weights list of the weights for the averaging. Must have length of the corresponding DFset

pots* set of potentials

RcutNB* the upper range to extend the potentials.

Example:

```
rdf1 = MT.ReadRDF('file1.rdf')
rdf2 = MT.ReadRDF('file2.rdf')
rdf_average = MT.Average([rdf1, rdf2], weights=[1.0, 1.0])
```

4.3.7 SaveDFsAsText(DFs)

Save function[s] from the given object, which may be single DF, DFset or list of DFs into a separate text-file. Useful when plotting with external software. Each function is saved in a tabulated format: First column - distances in Å, second column - values. The text file has the same name as the according function. The files are ready to be plotted by `gnuplot`, e.g.

```
gnuplot> plot './NB.RDF.NB.N-N.i1.dat' w lines
```

DFs* - set (list of sets) of the functions to save (mandatory argument)

Example:

```
df_set = MT.ReadPot('DMPC.pot', Ucut=1e5)
MT.SaveDFsAsText(pots)
```

4.3.8 DFset.Write(ofile, Split=False)

Write the set of RDFs/potentials to the file `ofile` and optionally split it into a main header file and an additional set of included files.

Parameters:

ofilename (str) : File to write the set

Split : Default False. If True, all functions will be written to a separate include-files. If Split=[True, False, True,....] only those functions DFs[i] where Split[i]=True will be written to include-files, and other will be kept in the main file

Example:

```
df_set = MT.ReadPot('DMPC.pot', Ucut=1e5)
df_set.Write('DMPC.split.pot', Split=True)
df_set.Write('DMPC.split2.pot',
                     Split=[i>10 for i in range(len(df_set.DFs))])
```

4.4 Exporting topologies and potentials

4.4.1 GromacsTopology(inpMagiC=None, system=None, topfile='topol.top', geometry=None, **kwargs)

Creates GROMACS topology file *.top for the system.

The system can be directly provided as a parameter, read from magic.inp file, or initiated from list of molecular types and number of corresponding molecules. For two latter cases *.mcm-files shall be present in the same directory. Number of molecules in the resulting system should be manually checked once the top-file is created.

inpMagiC (str) : Input file for MagiC.Core, used to define system's composition, i.e. list of moleculartypes and number of molecules of each type

system (system) : the system topology to export

geometry (str) : File with the starting geometry of the system in xmol format. If provided a corresponding .gro-file will be generated

topfile (str) : File to write the topology. Default 'topol.top'

dfset (DFset) : Set of potentials/RDFs having SameAsBond-records. If provided, the topology will take into account SameAsBond records from the DFset

These two parameters only required if the system to be directly composed from mcm-files

mcmfile : List of mcm-files defining molecular types

NMolMType : List of number of molecules of each type

Examples:

```
MT.GromacsTopology('magic.inp', geometry='start.xmol')
MT.GromacsTopology(system=system, geometry='start.xmol')
MT.GromacsTopology(system=system, mcmfile=['DMPC.CG', 'Chol.CG'],  NMolMType=[30, 30]))
```

4.4.2 LAMMPSTopology(inpMagiC=None, system=None, outfile="LAMMPS.data", hybrid=False, **kwargs)

Creates LAMMPS's topology file (LAMMPS.data) for the system.

The system can be directly provided as a parameter, read from magic.inp file, or initiated from list of molecular types and number of corresponding molecules. For two latter cases *.mcm-files shall be present in the same directory.

inpMagiC (str) : Input file for MagiC.Core, used to define system's composition, i.e. list of moleculartypes and number of molecules of each type

system (system) : the system topology to export

geometry (str) : File with the starting geometry of the system in xmol format. If provided a corresponding .gro-file will be generated

outfile (str) : file to write LAMMPS topology. Default: LAMMPS.data

dfset (DFset) : Set of potentials/RDFs having SameAsBond-records. If provided, the topology will take into account SameAsBond records from the DFset

hybrid (bool) : Add explicit bond-types to the topology file. Needed if few bond types are used, e.g. *table* and *zero*.

These two parameters only required if the system to be directly composed from mcm-files

mcmfile : List of mcm-files defining molecular types

NMolMType : List of number of molecules of each type

Examples:

```
MT.LAMMPSTopology('01.magic.inp', geometry='start.xmol')
MT.LAMMPSTopology(geometry='start.xmol', outfile='LAMMPS.data',
                   mcmfile=['DMPC.CG', 'Chol.CG'],  NMolMType=[30, 30])
```

4.4.3 GALAMOSTTopology(eps, inpMagiC=None, system=None, outfile="topology.xml", pyfile='tables.inc.py', **kwargs)

Creates GALAMOST topology files (.xml, and .py) for the system

The system can be directly provided as a parameter, read from magic.inp file, or initiated from list of molecular types and number of corresponding molecules. For two latter cases *.mcm-files shall be present in the same directory.

The resulting topology is made of two files: XML-file, with system's geometry, atom types, atoms, bonds, angles. This file has same format as HOOMD-blue topology, so it can be directly opened by VMD. The second file is a python-script, which is assigning tabulated potential files to every particular interactions in the system. These files are shall be independently created using PotsExport.

***eps (float)** : Dielectric permittivity, required for charge conversion into GALAMOST internal units

inpMagiC (str) : Input file for MagiC.Core, used to define system's composition, i.e. list of moleculartypes and number of molecules of each type

system (system) : the system topology to export

geometry (str) : File with the starting geometry of the system in xmol format. If provided a corresponding .gro-file will be generated

outfile (str) : file to write the topology. Default: 'topology.xml'

pyfile (str) : File for writing python records for tabulated potentials. Default 'tables.inc.py'

dfset (DFset) : Set of potentials/RDFs having SameAsBond-records. If provided, the topology will take into account SameAsBond records from the DFset

These two parameters only required if the system to be directly composed from mcm-files

mcmfile : List of mcm-files defining molecular types

NMolMType : List of number of molecules of each type

Example:
```
MT.GALAMOSTTopology(inpMagiC='01.magic.inp', geometry='start.xmol', eps=78.0)
```

4.4.4 PotsExport(pots, MDEngine, ...):

Export the given set of potentials (pots) to one of the available molecular dynamics packages:

Parameters:

pots The set of potentials to export

MDEngine Name of the MD package to export to: 'LAMMPS', 'GROMACS', 'GALAMOST'

npoints Number of points in the resulting tables

Umax Max value of energy (kJ/mol) at the repulsive wall region

Rmaxtable Maximum range of the table in Å(for both non-bonded and pair-bonds). Default 25.0

PHImaxtable Maximum angle value (deg) in the table for angle-bendind bonds. Default 180.0

filename Prefix for the potential files name. Default empty sting.

interpol Perform interpolation (True), or keep original resolution of the table (False). Default True.

method The interpolation method to use `'gauss'` (default) or `'sciint'`, which gives smoother forces, but may produce artifacts near Rcut, so use it with care

zeroforce - do not write forces into .xvg-file, but write zeros instead (optional argument). In such case GROMACS should automatically calculate forces from potentials. Default: False - forces are to be written. NB: Even if `zeroforce=True` force values are plotted.

noplot Do not produce supplementary plots. Default False

hardcopy Save supplementary plot to eps-files. Default False.

figsize Size of the supplementary plots

dpi Resolution of the supplementary plots

Example:
```
MT.PotsExport(pot, MDEngine='GROMACS', Rmaxtable=25.0, PHImaxtable=180.0,
     npoints=2500, Umax=6000.0, interpol=True, method='gauss', sigma=0.5,
     noplot=False, hardcopy=True, figsize=(14,7.5), dpi=120)
```

4.4.5 Potential export procedure details

The export is rather complex process, here we give some details about how it is done.

The molecular dynamics simulation heavily relies on quality of the tabulated potentials, i.e. the potential and force should be smooth and do not have discontinuities at the whole range $[0 - r_{cutoff}]$.

The Monte Carlo approach, however, is much more tolerant to the potential quality, the potential obtained in the inverse process is defined on a relatively sparse grid, may have kinks, and is defined at the limited range $[r_{min} : r_{max}]$.

Thus the short-range potential, which is generated by the MagiC core, have to be extrapolated both at the left-hand side and at the right hand side. Moreover, it is desirable to interpolate the potential to a higher grid density and also make it smooth to avoid numerical instabilities in force calculation.

In MagiC implementation, the left side extension represents a strongly repulsive core, and it is approximated by $U^{left}(r) = ar^2 + br + U_{max}$ on the range $[0 : r_{min}]$, and coefficients a and b are chosen to provide continuity of $\frac{d}{dr}U(r)$ at $r = r_{min}$.

The right side extension may differ, depending on the type of the potential. Non-bonded potentials should smoothly decay to zero when $r_{max} < r < r_{cutoff}$, this is achieved by using this expression:

$$U^{right}_{short\,range}(r) = U(r_{max}) \cdot \exp[-10\frac{(r - r_{max})}{r_{cutoff} - r_{max}}] \tag{7}$$

Here 10 is some predefined coefficient, r_{cutoff} is the range of the resulting potential table as required by the MD Engine.

Angle-bending potentials should also decay to zero at $\phi = 180°$ which is implemented by:

$$U^{right}_{angle}(\phi) = U(\phi_{max}) \cdot \exp[-100\frac{(\phi - \phi_{max})}{180° - \phi_{max}}] \tag{8}$$

Pairwise bond potentials should have an attractive wall at the right side, which is approximated by harmonic wall in the same way as the repulsive wall at the left side:

$$U^{right}_{pair\,bond}(r) = ar^2 + br + U_{max} \tag{9}$$

coefficients a and b are chosen to provide continuity of $\frac{d}{dr}U(r)$ at $r = r_{max}$.

Once the original potential has been extended, all intermediate points are interpolated to a denser and smoother grid. Number of nodes in such grid is defined by parameter `npoints`.

The interpolation can be made either by SciPy radial basis function interpolation, or Gaussian smoothing, e.g. interpolated values are calculated as a exponential-weight average:

$$U^{new}(r) = \frac{1}{Z(r)} \sum_{r_i=r_{min}}^{r_{max}} U^{orig}(r_i) \cdot \exp\frac{-(r - r_i)^2}{2\sigma^2} \tag{10}$$

$$Z(r) = \sum_{r_i=r_{min}}^{r_{max}} \exp\frac{-(r - r_i)^2}{2\sigma^2} \tag{11}$$

where sigma defines how broad is the averaging. By default σ is half resolution of the original table. The first of the interpolation methods provides smoother force, however may be subject to artifacts at large distances.

If `interpol=False`, the original grid will be kept.

Once the interpolation is done, each resulting potential and force based on it are written to table-files in the format corresponding to the desired MD engine. For LAMMPS it is `*.table`-file as described here and here.

For GROMACS, it is `*.xvg` format described here. Forces can be suppressed by setting `noforce=True`, then zeros will be written to the file. In such case GROMACS should automatically calculate forces from a given potential.

For GALAMOST it is `*.dat` format, which has no description on the web, so we recommend to take one of our Tutorials to see an example there.

Each table file is named by the name and the type of the corresponding potential.

In order to control the results, each original potential, the extrapolated part and the interpolated potential are plotted together, so the user can visually inspect the result and check for artifacts, fluctuations and other numerical issues which may occur. The example of such plot is shown on figure 4

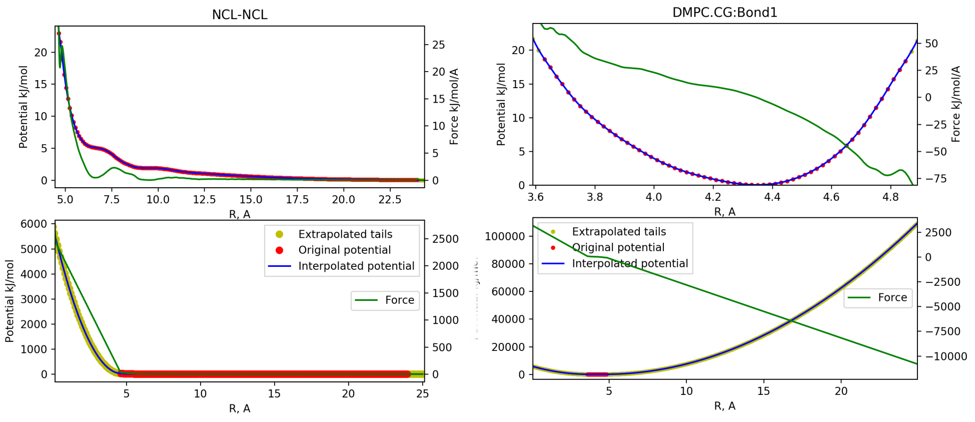


Figure 4: Example of potential exporting control plots. Left side - Non-bonded potential, Right side - pairwise bond potential. Upper plots display the initial range $[r_{min} : r_{max}]$, lower plots display the full range $[0 - r_{cutoff}]$ (after the extension). Red circles denote original potential values, yellow circles show extended values (both defined on sparse grid). Blue line shows interpolated potential defined on dense grid, and green line shows the calculated force.

4.4.6 xmol2gro

Converts `*.xmol` file **ifilename*** to a `*.gro` file **ofilename***.

ifilename - input .xmol file

ofilename - output .gro file

molnames* - list of molecular types that are present in the system

nmols* - list stating how many molecules of the respective type are present in the system

natimol* - list stating how many atoms each molecular type has

nconf - how many configurations to convert, default=1.

Example: `MT.xmol2gro(input.xmol, output.gro, molnames=['DMPC','WAT'], nmols=[98,2700], natimol=[118,3], nconf=1)`
Two component system of 98 molecules of DMPC (118 atoms) and 2700 molecules of water (3 atoms)

4.4.7 gro2xmol(ifilename, ofilename='output.xmol')

Convert `*.gro` file **ifilename*** to a `*.xmol` file **ofilename**. **Example:**
`MT.gro2xmol('input.gro', 'output.xmol')`

4.4.8 tpr2mmol(tprfile)

Convert GROMACS binary input file ***.tpr** to a set of ***.mmol** files. Useful when preparing input data for the bead mapping using existing all-atom GROMACS MD simulation.
Example: `MT.tpr2mmol('mdrun.tpr')`

4.4.9 dcd2xtc(trj, top)

Convert GALAMOST .dcd trajectory to .xtc. It is assumed that GALAMOST files have length unit of nm. Note that timestep is not preserved, as it is not stored in dcd-format

trj (str) : GALAMOST trajectory file in dcd format

top (str) : GALAMOST topology file in xml-format

Example:
```
MT.dcd2xtc("trj.dcd", "topology.xml")
```

4.4.10 GetStartConfs(ifilename, Nconfs, ofilename='start.xmol', Begin=0, End=0, Random=False)

Creates a set of starting configurations to be used in MagiC.Core. I.e. it reduces the full bead-mapped trajectory file to a set of uniformly distributed configurations.

Parameters:

ifilename (str) : Input trajectory in xmol-format

Nconfs (int) : Number of frames to generate

ofilename (str) : Output trajectory filename

Begin, End (int) : Specify range of the configurations to pick from

Random (bool) : Pick frames randomly (with uniform distribution), otherwise use constant step

Example:
```
MT.GetStartConfs('traj.xmol', Nconfs=100, Begin=0, End=10000, Random=False)
```
Uniformly pick 100 configurations from file traj.xmol, starting with configuration 0 and up to configuration 10000. Every 100th configuration will be picked.

5 MagicTools object-classes reference

5.1 Tabulated functions: RDFs, potentials, etc.

5.1.1 DF: Distribution Function

DF is a base class representing a single Distribution Function (e.g. RDF, bond length distribution, angle distribution, intermolecular potential, angle-bending potential, correction to potential, etc.) The class contains properties and methods, which are common for every function, however, some methods are redefined when necessary to keep function specificity:

Properties:

Name Name of the DF

FullName Completely resolved name of the DF, including type and kind of the DF

x,y numpy arrays storing tabulated values of the argument and the function

Min,Max Range of distance/angle values where the function is defined

Type Type of the function: NB, B (pairwise bond), A (angle-bond)

Kind Kind of the function: RDF, Potential

Npoints Number of points in a table defining the function

Resol Resolution of the table defining the function

AtomTypes (NB only) Names of the atom/bead types involved in the interaction represented by the function

BondNumber (B/A only) Number of the bond represented by the function.

MolTypeName (B/A only) Name of the Molecular type the bond function refers to.

AtomGroups (B/A only) List of atom pairs/triplets involved in the bond

Methods:

Write() Write the function into a file-stream

Plot() Plot the function

Save() Save the function in a tabulated for to a text file.

Average() Make a new DF which is average of given list of DFs

ExtendRange() Extend range of the DF

ExtendTail() Extend the tail range of NB distribution function in to RcutNB

CutTail() Cut the tail of NB RDF/potential

IsSimilar() Check if the DF is similar to given one, based on its Type, Kind, AtomTypes, MolecularType and Bond number

ChangeResolution() Change resolution of the potential. The new points will have an average value between closest neighbors

Distance() Calculate the distance between this and the given DF

Examples:
`import DF` - import the class
`RDFref=MagicTools.ReadRDF('dmpc.400ns.v2.rdf')` - read a set of RDFs, which is and object of the class DFset, containing a number of DF-objects.
`rdfNB=RDFref.DFs[0]` - Access the first function of the set. It is an object of class DF representing a non-bonded RDF.
`rdfNB.g` - Access the table of the function
`rdfNB.Plot()` - Plot the function
`rdfNB.AtomTypes` - See what bead/atom types are involved in the function.
`rdfB=RDFref.DFs_B[0]` - Access the first pairwise bond related function of the set (they are indexed from 0)
`rdfB.MolTypeName` - See the name of the molecular type the bond function belongs to.

5.1.2 DFset - set of Distribution Functions

Class representing a set of Distribution Functions (RDF, Potential, Potential correction, etc.)

Properties:

Name - Name of the set (typically name of the file the set was read from)

NTypes - Number of different atom types used in the set

AtomTypes - Names of the atom types involved in the set

Min, Max - Range of distance values for non-bonded interaction functions

Npoints - Number of points in non-bonded interaction functions

DFs - List of functions (all functions in the set)

DFs_NB - List of non-bonded interaction functions

DFs_B - List of pairwise bond interaction functions

DFs_A - List of angle-bending bond interaction functions

NPairBondsExclude, NAngleBondsExclude - Two dictionaries defining exclusions for molecular types involved in the DFset

Methods:

DFset() - Construct the object from provided rdf/pot file (recommended way) or from the provided parameters.

Write() - Write the set of functions to the file (.rdf or pot).

Plot() - Plot the set of functions

Reduce() - Compare the set to the provided one and extract similar functions. Useful to extract functions related to one molecule from larger set of functions.

SetTitle() - Set title for the DFset and for every DF of the set to have nice legends in massive plots

AddCore() - Add repulsive core to the Non-bonded potentials and sets Rmin=0

CutTail(RcutNB) - Shorten the range of NB potentials in the set to RcutNB

ChangeResolution(NewResol) - Changes resolution of the set. NewResol - tuple of 3 values (NB, B, A).

ExtendRange(RcutNB) description - Extend the range of NB potentials in the set to RcutNB

SetPlotProperty(key,value) - Set plot-related keyword property for the DF-set and for every function of the set. Used for fine control of the pictures in massive plots

Examples:

`RDFref=MagicTools.ReadRDF('dmpc.400ns.rdf')` - read a set of RDFs, which is and object of the class DFset.
`RDFref.Plot()` - Plot the functions.
`RDFref.Write('RDFref.rdf')` - Write the set to the file.
`print(RDFref.Name)` - Print the specific property of the set (Name)
`RDFref.DFs[0]` - Access the first function of the set (they are indexed from 0)
`RDFref.DFs_B[0]` - Access the first pairwise bond related function of the set (they are indexed from 0)

5.2 Topology

5.2.1 System: Top level object representing the whole molecular system

Creating the system-object is the first step for all topology-related operations. There are two possibilities for it: Make an empty stub system, and then populate it with underlying structures; or create it using existing MagiC core input file `magic.inp` and corresponding molecular topology files `*.mcm`. The user can also explicitly provide list of molecular type names (mcmfile) and number of molecules of each type (NMolMType) and box size (Box), see examples below.

Properties:

MolTypes - Molecular types of the system

Molecules - Molecules belonging to the system

BondTypes - Bond types (both pairwise and angle-bending) belonging to the system

PairBondTypes - Pairwise bond types belonging to the system

AngleBondTypes - Angle-bending bond types belonging to the system

Bonds - Bonds (both pairwise and angle-bending) belonging to the system

PairBonds - Pairwise bonds belonging to the system

AngleBonds - Angle-bending bonds belonging to the system

AtomTypes - List of atom types defined in the system

Atoms - List of atoms belonging to the system

Sites - List of sites, i.e. atoms belonging to molecular types of the system

Methods:

Construct and populate the system:

AddMolType - Add molecular type to the system

AddAtomType - Add atom type to the system

ReadGeometry - Read system's geometry from XMOL file

SetExclusions - Set exclusion rules for the system

ImputeSameAsBond - Update BondTypes in the system according to the provided set of potentials/RDFs

Write to files:

WriteLAMMPSData - Write the system's topology to LAMMPS data file

WriteGromacsTopology - Write the system's topology to GROMACS-topology file topfile.top

WriteGALAMOSTxml - Write the system's topology to GALAMOST XML format and write records for the tabulated potentials as GALAMOST python script

WriteGALAMOSTExclusions - Create a set of two exclusion files for GALAMOST

WriteMCMs - Save all molecular types of the system to corresponding MCM-files

WriteGeometryGRO - Write system's geometry as .gro file

WriteAsRDFinp - Print the system as lines for RDF.inp file. Useful when writing script generating RDF.inp file

Search and resolve names to objects:

GetBondType - Find bond type by MolTypeName:BondNumber

GetAtomType - Find atom type by it's name

GetMolType - Find molecular type by it's name

IsSystemMatchRDFs - Check if current geometry of the system matches the given set of RDFs

Examples:

```
system0 = System() # empty system
system1 = System(input='magic.inp') # same system as specified in MagiC.Core input file
system2 = System(mcmfile=['MT1.CG', 'MT2.CG'], NMolMType=[10, 10], Box=[10., 10., 10.])
system3 = System(input='magic.inp', dfset='potentials.pot', geometry='start.xmol')
```

5.2.2 MolType: Topology of a single molecular type

Molecular Type is a container for storing molecular topology, i.e. atoms (and their types) and bond-types (and bonds). It must belong to some System. In addition to the topology, MolType stores the list of actual molecules of this type.

The molecular type instance can be read from mcm-file, otherwise an stub molecular type will be created. When the molecular type is read from file, it automatically gets one corresponding molecule assigned.

Properties:

Name - The molecular type name

System - The system which the Molecular Type belongs to

Molecules - Molecules of the Molecular Type

BondTypes (also PairBondTypes and AngleBondTypes) - List of BondTypes belonging to the Molecular Type

Bonds (also PairBonds and AngleBond) - List of Bonds belonging to the Molecular Type

Atoms - List of atoms belonging to molecules of the molecular type

Methods:

AddMolecule - Add molecule to the MolType

Write2MCM - Write the molecular file to a mcm-file

Examples:

```
moltype_DNA = MagicTools.MolType('DNA.CG', system) # Read molecular type from file
moltype_stub = MagicTools.MolType('stub', system) # Create a stub
```

5.2.3 Molecule

Class representing a single molecule.

Properties:

MolType - Molecular Type of the molecule

Name - Name of the molecule, either user-provided or generated from MolecularType name and molecule number

Number - Serial number of the molecule within all molecules of the corresponding molecular type

ID - Serial number of the molecule within all molecules of the system

Atoms - List of atoms belonging to the molecule

Bonds (also PairBonds and AngleBond) - List of Bonds belonging to the Molecule

Methods:

AddAtom(atom) - Add the atom to the molecule

AddBond(bond) - Add the bond to the molecule

5.2.4 BondType:

The BondType is a group of bonds (pairwise or angle-bending), which are belonging to the same molecular type and described by the same interactions potential.

Properties:

MolType - Molecular Type the BondType belongs to

Name - Bond Type name for text representation

ID - Serial number of the Bond Type within all BondTypes of the System

Number - Serial number of the BondType within all BondTypes of the MolecularType

Bonds - List of Bonds belonging to this BondType

AtomGroups - List of atom groups (duplets/triplets), each group represents one bond of the BondType

Methods:

AddBond(bond) - Add bond to the BondType

Write2MCM(stream) - Write the bond type to the output-stream in mcm-file format.

WriteAsRDFinp() - Print the BondType as line for RDF.inp file. Useful when writing script generating RDF.inp

5.2.5 Bond:

Class representing a single Bond connecting two or three atoms, depending on kind of the bond

Properties:

BondType - Bond Type of the Bond

Molecule - Molecule the bond belongs to

Name - Bond name for text representation

ID - Serial number of the Bond Type within all BondTypes of the System

Number - Serial number of the bond within all bonds in the system having the same kind (pairwise or angle-bending)

Atoms - Pair or Triplet of atoms bonded by the bond

Value - Distance or angle of the bond

5.2.6 AtomType

Class representing an atom type.

Properties:

System - The system which the Atom Type belongs to

Number - Serial number of the atom type

Atoms - List of Atoms belonging to this AtomType

Charge - Charge of the atom-type as average over charges of all atoms of the type in a.u.

Mass - Mass of the atom-type as average over masses of all atoms of the type in a.u.

Methods:

WriteAsRDFinp() - Return a string representation of AtomType for RDF.inp

AddAtom(atom) - Add atom to the AtomType

5.2.7 Atom

Class representing a single atom

Properties:

AtomType(AtomType) - Type of the atom

Molecule (Molecule) - The molecule where the atoms belongs to

Name(str) - Name of the atom

Number(int) - Atom's serial number within all atoms of the molecule

ID(int) - Atom's serial number within all atoms of the system

Charge (float) - Charge of the atom in electron charges

Mass (float) - Mass of the atom in a.u.

R (np.ndarray) - Coordinates of the atom (A)

Bonds - List of bonds (Pairwise and angle) where the atom is involved

BondedAtoms (also PairBondedAtoms, AngleBondedAtoms) - List of atoms bonded to this one. Both angle- and pairwise bonds are taken into account

Methods:

IsBonded(that_atom) - Check if that atom is bonded to this one

Distance(that_atom) - Calculate Euclidian Distance from that atom to this one

PBC() - Apply periodic boundary conditions to the atom's coordinates (in-place!)

Write2MCM(stream) - Write atom to the mcmfile-stream

6 File formats

6.1 .xmol

This is plain text trajectory format, which can be produced by many molecular modeling packages, including MagiC. It consists of a number of consequent frames, with each frame having the following structure:

line 1: Number of atoms in the frame (N)

line 2: A commentary line

line 3: Name(atom1) X(atom1) Y(atom1) Z(atom1)

line 4: Name(atom2) X(atom2) Y(atom2) Z(atom2)

lines 5,6...,N+2: Names and coordinates of atoms 3 - N.

In case of trajectory, configuration files from each time frame are written consequently one after another.

There is no common requirements for the commentary (second) line. For CGtraj module of MagiC it is assumed that the second line of each configuration follows this format (accepted in MDynaMix):

```
(char) <time> (char-s) BOX: <box_x> <box_y> <box_z>
```

where `(char)` is any character word, `<time>` is time in fs, `<box_x> <box_y> <box_z>` (following after keyword `BOX`). The length unit is Ångströms and time unit is femtoseconds. The time step information is not needed for CGTraj but the box size information is essential. If no box size information is present in the trajectory, the box size information can be supplied in the input file, but the later has a sense only in constant-volume simulations.

In xmol-files produced by MagiC the second line may have no time-stamp and box sizes.

6.2 .mmol

MMOL is a molecular topology file format, which is inherited from MDynaMix MD software. It consists of two parts: first part describes atomic composition, geometry, charges, masses and non-bonded interactions; the second part defines bonds, angles and torsions in the molecule. MMOL-topolygy files are used by cgtraj (subsection 3.2) for converting high-resolution trajectory to a coarse grained one and for calculation of the reference distribution functions rdf (subsection 3.3). In both cases only information from the first section of .mmol file (information about atomic composition) is used, and the bonding part may be omitted.

The first part of the file, which is part of interest has the following structure: The first non-commentary line of a .mmol file is the number of atoms in the molecule. After it the corresponding number of lines follows, one line per atom. Each line contains 8 compulsory parameters. They are: 1) atom name in the program; 2),3) and 4) are the initial X,Y,Z coordinates of the atom in the molecular coordinate system, 5) mass in atom units, 6) charge, 7) Lennard-Jones parameter σ in Å, 8) Lennard-Jones parameter ε in kJ/M. Two optional columns may present.

For the correct work of CGTraj utility, the only important information is the number of atoms in the molecule and masses of atoms. CGTraj utility may work without supplying .mmol files, in such case the masses of atoms are set to 1 and charges to 0.

6.2.1 MMOL Example: H2O.mmol

```
#==================================================I
#       Molecular Dynamics Data Base               I
#       Configuration and interaction potential    I
#==================================================I
#           SPC H2O model                          I
#==================================================I
#       Number of sites
 3
#       X        Y         Z         M          Q         sigma     epsilon
#               (A)                                                  (kJ/M)
O       0.       0.    -0.064609  15.9994      -0.82       3.1656    0.6502
H1      0.    -0.81649  0.51275    1.008        0.41       0.        0.
H2      0.     0.81649  0.51275    1.008        0.41       0.        0.
# We care only up to this line! The rest of this file can be skipped.
#    Num. of strings for the reference
4
       SPC water model
       Parameters from:
       K TOUKAN AND A.RAHMAN,
       PHYS. REV. B Vol. 31(2) 2643 (1985)
#   Num. of bonds
 3
#ID(typ)     N1   N2         Reqv      Force        D        RHO (A**-1)
 1           1    2          1.        2811.      420.      2.566
 1           1    3          1.        2811.      420.      2.566
 0           2    3          1.633      687.       0.        0.
#   Num. of angles
 0
#   Num of dihedrals
 0
#   Additional options
#   flexible SPC water
fSPC
```

6.3 .mcm

Coarse-grain topology file, which is similar to CG.mmol, but includes information about bead/CG-atom types and intramolecular bonds. In general it consists of three parts: the first part describes atoms involved in the molecule, the second part contains list of covalent-like bonds and the third part lists angle-bending bonds. An individual .mcm file should be provided for each molecular type present in the system. These files are automatically generated by rdf.py utility during computation of the reference distribution functions, but they can be manually edited if necessary. For example, partial charges of the CG sites can be corrected.

Format of .mcm file:
NB! Lines beginning with '#' or '!' are commentaries, they are dropped while parsing the file.

First the atom description block is specified:
1 line: Number of bead/atoms in the molecule (Natoms)
Natoms lines: Atom records. One record per line.
Each record contains 8 parameters. Atom name; X,Y,Z coordinates (Å) of the atom in a local coordinate system, mass (au.); charge (el.); index of the atom type; name of the bead/atom type. The short-range non-bonded interaction between a pair of beads/atoms will be defined by the bead/atom types defined in this file. Atoms of the same atom type interact by the same non-bonded potential.
Then the pairwise bond block is specified:
1 line: The total number of pairwise bond types present in the molecule (Nbonds).
Then for every individual bond type (of Nbonds) one need to specify:
1 line: Number of atom pairs which are involved in the bond of this type (NPairs);
NPairs lines: List of such atom pairs, one pair per line.
Then the angle bending bond block is specified:
1 line: Total number of angle bending bond types present in the molecule. For every individual bond type (NAngles) one need to specify:
1 line: Number of atom triplets which are involved in the bond of this type (NAngles);
NAngles lines: List of such atom triplets, one triplet per line.

NB! In the first version of MagiC the triplet used to had unusual order: central atom stands last in the triple, e.g. triplet 1 3 2, defines angle between 1-2 and 2-3. Now it is changed to the regular 1-2-3 order, and to avoid misunderstanding `rdf.py` utility automatically writes `Order=1-2-3` after the number of total angle bending bonds.

6.3.1 .mcm file example: DMPC.CG.mcm

The mcm-file listed below defines 10-beads model of DMPC-lipid, as shown on figure 5.

```
#Number of atoms
10
# Name   X      Y       Z      Mass   Q  NumofType NameofType
N    -6.1804 -17.2679 17.2781 73.139 0.76 1 N
P    -9.6995 -17.2121 14.915 123.0256 -0.89 2 P
C2   -18.1023 -13.2828 10.2371 56.108 -0.0 3 CH
C3   -21.9375 -11.5562 7.3382 56.108 -0.0 3 CH
C4   -24.5552 -9.7683 3.2938 57.116 -0.0 3 CH
C6   -15.3122 -17.1232 8.1441 56.108 -0.0 3 CH
C7   -18.7644 -15.0198 5.1392 56.108 -0.0 3 CH
C8   -21.6201 -13.2988 1.0155 57.116 -0.0 3 CH
C1   -14.7182 -15.6667 12.7078 72.0638 -0.09 4 CO
C5   -13.3132 -18.7418 11.2877 71.0558 0.22 4 CO
#
# Here we define covalent like bonds
# Total number of covalent bond types: 5
5
# Covalent bond N-P
# One atom pair belongs to covalent bond type 1
```

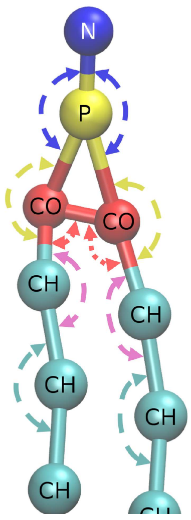


Figure 5: Example: 10-beads CG model of DMPC-lipid. Beads and bonds of same color have same type; solid lines denote covalent bonds; dashed arrows denote angle bending bonds.

```
1
# Define pair of atoms by their numbers in the list above: 1 (N) 2 (P),
# the third number should always be 1 for covalent bond.
1  2
# Covalent bond P-CO
# Two atom pairs belong to covalent bond type 2
2
# Define 2 pairs of atoms by their numbers in the list above: 2 (P) and 9,10 (C1,C5)
2  9
2  10
# Covalent bond CH-CH
# Four atom pairs belong to covalent bond type 3
4
# Define 4 pairs of atoms by their numbers in the list above: 3-4, 4-5, 6-7, 7-8
3  4
4  5
6  7
7  8
# Covalent bond type 4
2
9  3
10  6
# Covalent bond type 5
1
9  10
#
# Here we define angle bending bonds
# Total number of angle bending bonds:5
```

```
5
# Two atom triplets belong to angle bond type 1:
2
1  2  9
1  2  10
# Two atom triplets belong to angle bond type 2:
2
2  9   3
2  10  6
# angle bond type 3:
2
9  3  4
10  6  7
# angle bond type 4:
2
3  4  5
6  7  8
# angle bond type 3:
2
3  9  10
6  10  9
```

6.4 .rdf and .pot file formats

The described here file formats for .rdf and .pot files are valid for MagiC v. 2.0 and higher. The format of files for .rdf and for potentials is similar and we give here a common description of it refering as "RDF/potential" file format.

The RDF/potential file consists of a header section, marked by tags `&General` and `&EndGeneral` describing general properties of the RDFs/Potentials set for a specific system, and independent RDF/potential records, marked `&Potential` ... `&EndPotential` or `&RDF` ... `&EndRDF` respectively, which specify RDF/potentials for each individual interaction. Each individual RDF/Potential can be included to the RDF/potential file from a separate file by `include` statement.

Non-Bonded (NB) RDF/Potentials are identified by atom types involved in the RDF/Potential. Pairwise (B) and Angle-bending (A) bonds are identified by the molecular type they belongs to and the relative bond number in the molecular type. Each Potential can be protected from correction (Fixed) by specifying the flag Fixed=True in the corresponding section of the potential file. Such potentials do not change during the inverse procedure.

6.4.1 Header

`&General - &EndGeneral`

The header defines common properties of the RDFs/potentials provided in the file. Since individual RDF/potential records can be included from external files, the header section provides important information which helps to provide consistency between all records.

NTypes - Number of bead/atom types present in the system.

N_NB, N_B, N_A - Number of Non-Bonded, pairwise Bonded, and Angle-bending records, respectively

NPoints - Number of points in each NB-record.

Min, Max - Range of distance (in Å) where Non-Bonded RDF/Potentials are defined

6.4.2 RDF/potential record:

`&RDF` ... `&EndRDF`
or
`&Potential` ... `&EndPotential`

Each individual record specifies one RDF/potential, providing information the CG atom types, range, number of points and the data-table with actual values. A record can be also included from a separate file. Every record consists of specifications, data table (`&Table...&EndTable`) and optional include section (`&IncludePotential...&EndIncludePotential`).

Name - Name of the RDF/potential record. Is used as a comment line

Type - Type of the record. Can be NB, B, A, i.e. non-bonded, pairwise bond, angle-bending bond, respectively

Min, Max - Range (in Å) where the record is defined

NPoints - Number of points in the record. Note, that for the core region of RDF, which has zero values and often omitted, the Min value is typically not zero

AtomTypes - For NB-record, specifies the pair of bead/atom types which are involved in the interaction.

MolType - For B- or A-record, specifies molecular type the bond belongs to.

BondNumber - For B- or A-record, specifies a relative number of bond to which this record belongs. Note that bonds are indexed locally, with respect to the molecular type, the same way as in the corresponding mcm-file for the given molecular type

NPairs or NTriplets - Number of atom pairs (triplets) involved in the bond (B- or A-bond)

Pairs or Triplets - List of atom pairs (triplets) involved in the bond. Pairs/Triplets are comma separated, and atom numbers are separated by dash symbol (-) withing each pair/triplet. For triplets, atom numbers are specified in a direct way, so the central atom of the triplet is a central atom of the angle. Atoms are numbered locally, with respect to the molecular type, same way as in the corresponding mcm-file for the given molecular type).

&Fixed - Potential file only. If stated, the potential will be excluded from the inverse procedure (e.g. it will be fixed in potential update/refinement)

&InitZero - RDF-file only. If the corresponding potential is not provided, initiate it with zero. Default for NB-potentials.

&InitPMF - RDF-file only. If the corresponding potential is not provided, initiate it with Potential of Mean Force. Default for bond-potentials (both A- and B-).

&Table...&EndTable - Actual table defining the RDF/potential. The first column specifies distance (Å) or angle (deg), the second column specifies value, which is unitless for RDF and kJ/mol for potential. The table shall be uniformly spaced, and the same grid resolution should be used in all RDFs and potentials of the same kind (NB, B or A) for the whole system

&IncludePotential=IncludeFileName or

&IncludeRDF=IncludeFileName - Instead of providing data table in the record, one can import it from an external file `<filename>`. This feature allows to incorporate easily potentials previously obtained for other systems into the current one.

6.4.3 Included Potential/RDF

`&IncludedPotential` ... `&EndIncludedPotential`
`&IncludedRDF` ... `&EndIncludedRDF` The included record has nearly identical format as the parental RDF/potential section. The record specification values shall be also in agreement with the corresponding values of the parental section, to provide consistency of the whole set of RDF/potentials for the studied system.

Name

Type

Min, Max

NPoints

AtomTypes - NB interactions only

MolType - For bonds only

BondNumber - For bonds only

NPairs or NTriplets - For bonds only

Pairs or Triplets - For bonds only

&Table...&EndTable

Note that &Fixed, &InitZero and &InitPMF keywords are not applicable in the included record, but shall be used in the main potential/RDF file instead.

6.5 exclusions.dat

The file format for describing exclusions between all interaction sites of the system. This file is created by rdf.py and used a input to MagiC core. It can be edited manually if necessary.

The first two lines of the file state exclusion rules for each molecular type of the system, i.e. define the maximum number of bonds between atoms which is enough to exclude them from the non-bonded interactions. All remaining lines specify site-site exclusions:
¡site number¿ : ¡list of sites which do not have NB interactions to the given site¿
Example of exclusion file for DMPC-Cholesterol mixture:

```
NAngleBondsExclude=Chol.CG:1,DMPC.CG:1
NPairBondsExclude=Chol.CG:-1,DMPC.CG:1
1:2,3,4
```

```
2:1,3,4,5,8
3:1,2,4,5,6,8
4:1,2,3,5,8,9
5:2,3,4,6,7
6:3,5,7
7:5,6
8:2,3,4,9,10
9:4,8,10
10:8,9
11:12,13,14,15
12:11,13,14,15
13:11,12,14,15
14:11,12,13,15
15:11,12,13,14
```

The last five lines of this example imply that therer are no non-bonded interaction within coarse-grained cholesterol molecule defined by sites 11-15.