MAGIC: User manual. Version 1.0.3

Alexander Mirzoev[1,3], Alexander Lyubartsev[1,2]

2014-01-13

[1] *Division of Physical Chemistry, Arrhenius Laboratory, Stockholm University, Stockholm, SE-10691, Sweden.*

[2] alexander.lyubartsev@mmk.su.se

[3] alexander.mirzoev@mmk.su.se

Contents

1 Introduction

MAGIC is a software package designed to perform systematic structure-based coarse graining of molecular models and to compute effective potentials for a coarse-grained model which reproduce structural information (radial distribution functions, distributions of intramolecular degrees of freedom) obtained in high resolution (fine grained) simulations of the system of interest [1]. The package implements the inverse Monte-Carlo method [2] or Iterative Boltzmann inversion [3] method to compute the effective potentials. Such potentials can be further used for a large scale simulation. In general the systematic coarse graining can be considered as a multi-stage process which leads from a high resolution system description to a low resolution. A general scheme of this process, implemented in MagiC, is shown in figure 1). Each step of the coarse-graining (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).

As a first step the system of interest is modelled at high resolution, e.g. using Molecular Dynamics simulation with all-atomistic force field. Such a simulation results in an all-atom (AA) trajectory which is supposed to be long enough to provide representative sampling of the system of interest. This simulation can be performed with in principle any suitable MD or MC software on user's choice which can produce atomistic trajectories in one of the supported formats. At the second stage the atomistic trajectory is converted to a coarse-grained trajectory. A mapping scheme defining CG sites from the atomistic representation should be provided at this stage. The conversion is performed by *cgtraj* module of MAGIC. This stage results in a coarse-grained trajectory and assignment of mass/charge properties to CG-beads. At the next stage the structural reference distribution functions are calculated by *RDF* module of MagiC. At this stage the user assigns interaction types to all CG sites and defines intramolecular bonds and eventually angles in each molecule type of the CG model. After that the main procedure begins: solution of the inverse problem using Iterative Boltzmann Inversion and/or Inverse Monte Carlo methods. This is done by a kernel of the package which is also called *magic*. It results in a set of effective potentials, which reproduce the reference distribution functions, and an extended log-file which reports details of each IMC/IBI iteration. The log-file can be analyzed by a set of post-processing tools *MagicTools*, which allows to plot convergence rate, effective potentials, potential corrections, intermediate RDFs, etc. Once the effective potentials are obtained, they can be exported by *MagicTools* to an external MD software and used for further study of the large scale system. At the moment the post-processing tools provide export to GROMACS format of tabulated potentials, but for those familiar with python it can be straightforwardly to extend it to any other MD software which accepts tabulated potentials.

Since MAGIC is implemented as a set of separate modules, the user can perform preparatory tasks (creating coarse-grained trajectory, computation of RDFs), inversion procedure and analysis on different computer systems: computational clusters or local desctops. The package is written in a mixture of Python and Fortran 95/2003, with Fortran-based computationally intensive parts (magic,cgtraj and partially rdf), and Python-based interface class (MagicTools and partially rdf).

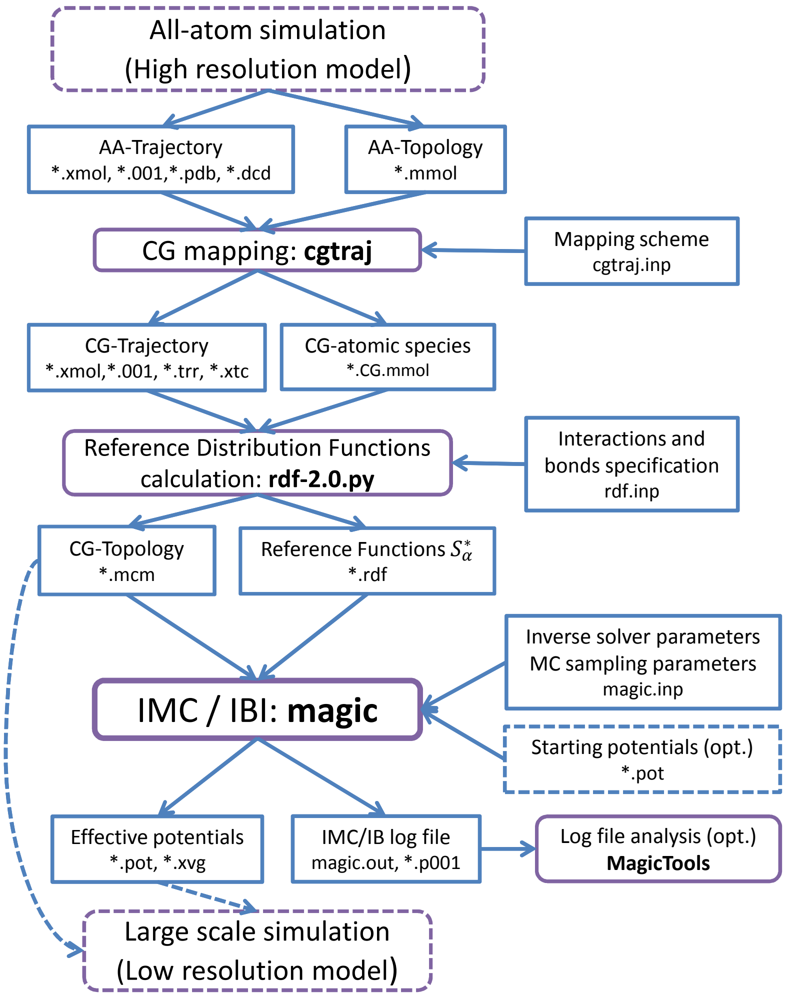

Figure 1: Systematic Coarse-Graining with MagiC: General outline. Blue rectangles denote input/output data; purple rectangles denote data processing procedures. Optional input data and use of external software are indicated by dashed frame.

2 Compilation

2.1 Prerequirements

To get the source code download and unpack the latest stable version of the software from our website (currently `http://code.google.com/p/magic`) or retrieve a (developers) copy from the repository:
`hg clone https://code.google.com/p/magic/`
This will result in a folder `magic-<current-version>`, containing the whole software. Below we will refer to MAGIC as a full path to the folder at your

computer.

For the python-based part of MagiC the following python software is required: numpy, matplotlib and ipython. If you are using Ubuntu, add these packages:
`sudo apt-get install ipython python-numpy python-matplotlib`. For Fedora, use the corresponding "yum" command.

To compile the Fortran-based part of MagiC you need a Fortran compiler implementing Fortran-95 standard. The following compilers were tested to provide successful compilations: Intel Fortran (v.$\geq$ 11), Oracle Solaris Studio, gfortran (v.$\geq$ 4.5). Also, LAPACK or equivalent linear algebra library is required. Lapack is included in Intel MKL which comes with Intel Fortran and it is also included in Oracle Solaris Performance Library, which comes with Oracle Solaris Studio. If you are using GNU Fortran (gfortran), you need a Linux repository build of Lapack (liblapack).

To compile the main module of MagiC in parallel, parallel MPI environment is required, for example OpenMPI or Mpich.

2.2 Fast Installation

- Open file `install.sh` for editing.
- Define the path to desired location of MagiC in variable `MAGIC`.
- Go to "INSTALL CGTRAJ" section and uncomment the line which refers to your Fortran compiler.
- Go to "INSTALL MAGIC" section and specify make option according to your Fortran compiler and MPI-library.
- Run the script. It will result in compilation of subparts of the package, linking executable in folder `$MAGIC/bin` and exporting `PATH`, `LD_LIBRARY_PATH` and `PYTHONPATH` environment variables.

If the fast installation script has run without errors, you can update environmental variables in your startup profile, see sec. Environment variables.

If you face errors during the execution of install.sh, you can try to compile MagiC step by step as described below.

2.3 *CGTraj*

Enter the `CGtraj` folder and run `make` with one of the options specifying compiler: `make intel` for Intel Fortran, `make oracle` for Oracle Solaris studio, or `make gfortran` for GNU Fortran (default option). For other compilers, try to edit one of Makefiles to match programming environment of your computer. If compilation has finished successfully you will get a binary file called `cgtraj`.

2.4 *RDF*

The rdf module is written in python and does not need compilation. You need only to specify environmental variables according to:

- add directory `$MAGIC/bin` to your `$PATH` environment variable.

- add `$MAGIC/RDF/rdf-2.0-lib` to your `$PYTHONPATH` environment variable.
- add `$MAGIC"/RDF/xdrfile-1.1.1` to your `LD_LIBRARY_PATH` environment variable.

For reading trajectories produced by Gromacs (xtc and trr) RDF relies on xdrfile library provided by GROMACS . If you are planning to use this trajectory format, you need to compile `xdrfile`. Enter `MAGIC/RDF/xdrfile` folder and run

```
./configure --enable-shared.
```

You can also specify location for the compiled library files by adding

```
--prefix=/path/to/library/folder
```

to configure arguments. Build the library by `make install`. If you have specified non-default location for the library, add it to `$LD_LIBRARY_PATH`. Finally you need to add python wrapper for the library in `$PYTHONPATH`:

```
export PYTHONPATH=MAGIC/RDF/xdrfile-1.1.1/src/python:$PYTHONPATH
```

2.5 *Magic*

To compile the main part of the package, a Fortran compiler and lapack linear algebra library are required. Lapack is included in Intel MKL which comes with Intel Fortran and it is also included in Oracle Solaris Performance Library, which comes with Oracle Solaris Studio. Other option is GNU Fortran (gfortran) and a repository build of Lapack (liblapack). If you have any of them, use appropriate Makefile for compilation or use `make` with an argument, for example `make intel`. Run `make` without arguments to see all options available. The `magic` module is the most computationally intensive part of the package, and most probably you will want to compile it in parallel which can be done acccording to one of Makefiles with mpi-extension. For ohter compilation options edit one of the available Makefiles. The result of successful compilation is a binary file `magic-<extension>` which you need to copy/link to your working directory or default `bin` folder.

2.6 *MagicTools*

MagicTools is a Python-based library, which only needs to be added to PYTHONPATH environment variable:

```
export PYTHONPATH=MAGIC/MagicTools:$PYTHONPATH
```

You can also add this line to your `.bashrc` file to perform the export operation automatically. To check if it is added successfully, open terminal, run `ipython` and load the library:

```
import MagicTools
```

If no error message appeared, the module is ready for use.

2.7 Environment variables

In order to keep exported values of environment variables defined above you may wish to add these lines to startup files of your shell:

Bash: Edit `$HOME/.bashrc` in case of interactive session or `.profile` in case of remote login session Add the following lines:

```
# Define location of MAGIC below.
MAGIC= <PATH_TO_LOCATION_OF_MAGIC>
export PATH=$PATH:$MAGIC/bin
export PYTHONPATH=$MAGIC/RDF/rdf-2.0-lib:$PYTHONPATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MAGIC/RDF/xdrfile-1.1.1
export PYTHONPATH=$MAGIC/RDF/xdrfile-1.1.1/src/python:$PYTHONPATH
export PYTHONPATH=$MAGIC/MagicTools:$PYTHONPATH
```

Csh/Tcsh:

```
# Define location of MAGIC below.
set MAGIC= <PATH_TO_LOCATION_OF_MAGIC>
setenv PATH "$PATH":"$MAGIC"/bin
setenv PYTHONPATH  "$MAGIC"/RDF/rdf-2.0-\lib:$PYTHONPATH
setenv PYTHONPATH  "$MAGIC"/RDF/xdrfile-1.1.1/src/python:$PYTHONPATH
setenv PYTHONPATH  "$MAGIC"/MagicTools:$PYTHONPATH
setenv LD_LIBRARY_PATH "$LD_LIBRARY_PATH":"$MAGIC"/RDF/xdrfile-1.1.1
```

3 Using MagiC

3.1 *CGTRAJ*: Coarse graining of high resolution trajectory

Input: *CGtraj* module requires the following input:

- Trajectory file in on of the following formats

  MDYN - MDynaMix trajectory binary format.

  XMOL - Text-based XYZ trajectory format, see details in section 4.1.

  PDB - Text-based trajectory format `http://www.rcsb.org/pdb` with headers as generated by *trjconv* utility of Gromacs

  DCD - NAMD binary trajectory file format.

  A number of consequential trajectory files should have extensions as *.001,*.002,*.003,...

  The order of atoms in each configuration of the trajectory should be:

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

  and within each molecular type

  ```
  <atoms of molecule 1> <atoms of molecule 2> ...
  ```

  within each molecule, the order of atoms is the same as in the molecular topology file `.mmol`.

- Molecular type descriptions for each type involved: *.mmol files. See details in section 4.2.

- Parameter file defining the input trajectory, output and the mapping scheme (refered here as `cgtraj.inp`).

Output:

- Coarse grained trajectory file in XMOL format.

- Coarse grained molecular types descriptions in mmol format, but without any bond and force field information: *.CG.mmol

Usage:

```
cgtraj cgtraj.inp
```

3.1.1 cgtraj.inp: input parameter file

The main input file for *cgtraj* consists of two independent parts. The first part describes the input atomistic trajectory, and the second part describes CG-bead mapping scheme. This 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 must be defined:

- `NFORM = <format>` where `<format>` is one of:
  - `MDYN` - MDynaMix binary trajectory
  - `XMOL` - XMOL trajectory. It is implied, 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 periodic box sizes.
  - `PDBT` - PDB trajectory as generated by `trjconv` utility of Gromacs.
  - `DCDT` - DCD trajectories as generated by e.g. NAMD
- `FNAME = <file_name>` set the base name of the trajectory files. The trajectory must be written as a sequence of files `<file_name>.001` , `<file_name>.002` and so on, the largest possible number being `<file_name>.999` .
- `PATHDB = <value>` (Optional) Directory with molecular description files (.mmol). Default is the current directory (.) .
- `NTYPES = <value>` Number of molecule types in the trajectory
- `NAMOL = <name1> [,<name2>,...] NTYPES` names of molecules. It is supposed that files `<name1>.mmol`, `<name2>.mmol`,... describing the molecular structure (MDynaMix format, see http://www.fos.su.se/mdynamix) are present in the directory defined by `PATHDB`. For analyzing trajectories generated by other programs, .mmol files are not compulsory, but it is still desirable to have the first section of .mmol files containing names of atoms and information about atom masses and charges. If this information not present, the program implies that all atom masses are equal to 1 (affect center of mass calculation for definition of CG sites) and charges are set to zero (can be reconstructed later manually).

- `NSPEC = <n1>[,<n2>,...]` Number of molecules of each type (`NTYPES` numbers). This parameter is not necessary in MDynaMix binary trajectories.
- `NSITS = <n1>[,<n2>,...]` Number of atoms in each molecule of the given type (`NTYPES` numbers). This parameter is not necessary if .mmol files are provided.
- `NFBEG = <value>` Number of the first trajectory file (integer between 0 and 999)
- `NFEND = <value>` Number of the last trajectory file (integer between 0 and 999)
- `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 Å if it is not present in the trajectory (implies constant-volume simulation)
- `ISTEP = <value>` Specifies that only each `ISTEP`-th configuration from the trajectory is taken for the analysis.

The second part, which describes CG bead mapping scheme, has a hypertext-like format. The keywords/tags are not case sensitive, and spaces will be automatically removed from the text. The whole section starts with keyword: `BeadMapping` and ends with `EndBeadMapping`. Every coarse grained molecular type has to be described in a separate section, which starts with tag
`CGMolecularType: <CGMolecularTypeName>`
and ends with `EndCGMolecularType`. Inside each section, parental molecular type name and CG beads definition should be given. The parental name is defined by tag `ParentType: <ParentMolecularTypeName>`. CG beads are defined in a line-per-bead way, where every line has the following structure:
`<Bead name>:<Number of atoms in the bead>:<list of atoms>`, where list of atoms is a comma separated list of atom numbers `atom1, atom2,...` according to the *mmol*-file describing the parental molecular type. The names of the CG beads should be unique for a given system.

An output `<CGMolecularTypeName>.mmol}`-file will be created for every defined CG-molecular type. This file will contain the following information of each CG molecular type: the numer of CG sites; names of the CG sites; possible coordinates of CG sites (e.g. to be used at startup of CG simulations), masses and charges of CG sites. These files will be used on the next stage in calculation of the reference distribution functions.

3.1.2 Example: cgtraj.inp

This is an example of input file for *cgtraj* module, which reads a trajectory written in binary MDynaMix format consisting of 10 files: `dmpc16.001, dmpc16.002,...,dmpc16.010`.

The system presented in the trajectory, consists of 16 DMPC-lipid molecules dissolved in 1600 water molecules, as described in [1]. The bead mapping scheme is shown on figure 2.

```
  &TRAJ
 NFORM='MDYN',
 FNAME='./MDynamix/dmpc16',
 PATHDB='./',
 NTYPES=2,
 NAMOL='dmpc','H2O',
 NSPEC=16,1600,
 NFBEG=1,
 NFEND=10,
 ISTEP=1,
 IPRINT=6,
 &END

 BeadMapping
 CGTrajectoryOutputFile:cgtraj.001
  CGMolecularType:dmpc_NM.CG
    ParentType: dmpc_NM
    N:16:43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58
    P:11:59,60,61,62,63,64,65,66,67,68,69
    C1:9:1,2,3,4,5,73,74,75,76
    C2:12:6,7,8,9,10,11,12,13,14,15,16,17
    C3:12:18,19,20,21,22,23,24,25,26,27,28,29
    C4:13:30,31,32,33,34,35,36,37,38,39,40,41,42
    C5:8:70,71,72,77,78,79,80,81
    C6:12:82,83,84,85,86,87,88,89,90,91,92,93
    C7:12:94,95,96,97,98,99,100,101,102,103,104,105
    C8:13:106,107,108,109,110,111,112,113,114,115,116,117,118
  EndCGMolecularType
  CGMolecularType:H2O.CG
    parenttype:H2O
    H2O:3:1, 2, 3
  endcgmoleculartype
EndBeadMapping
```

Note, that even if molecules of a certain type are present in the atomistic trajectory but are completely excluded in the coarse-graining (for example, solvent), the `CGMolecularType` section for this molecule type should still be present in the input file, but without definitions of CG sites.

3.2 *rdf*: Reference Distribution functions calculation

Input:

- Molecular type description file for each CG type present in the system: **.CG.mmol*. These files are automatically created at the previous stage (*c*gtraj)
- Coarse Grained trajectory file: **.xmol*.

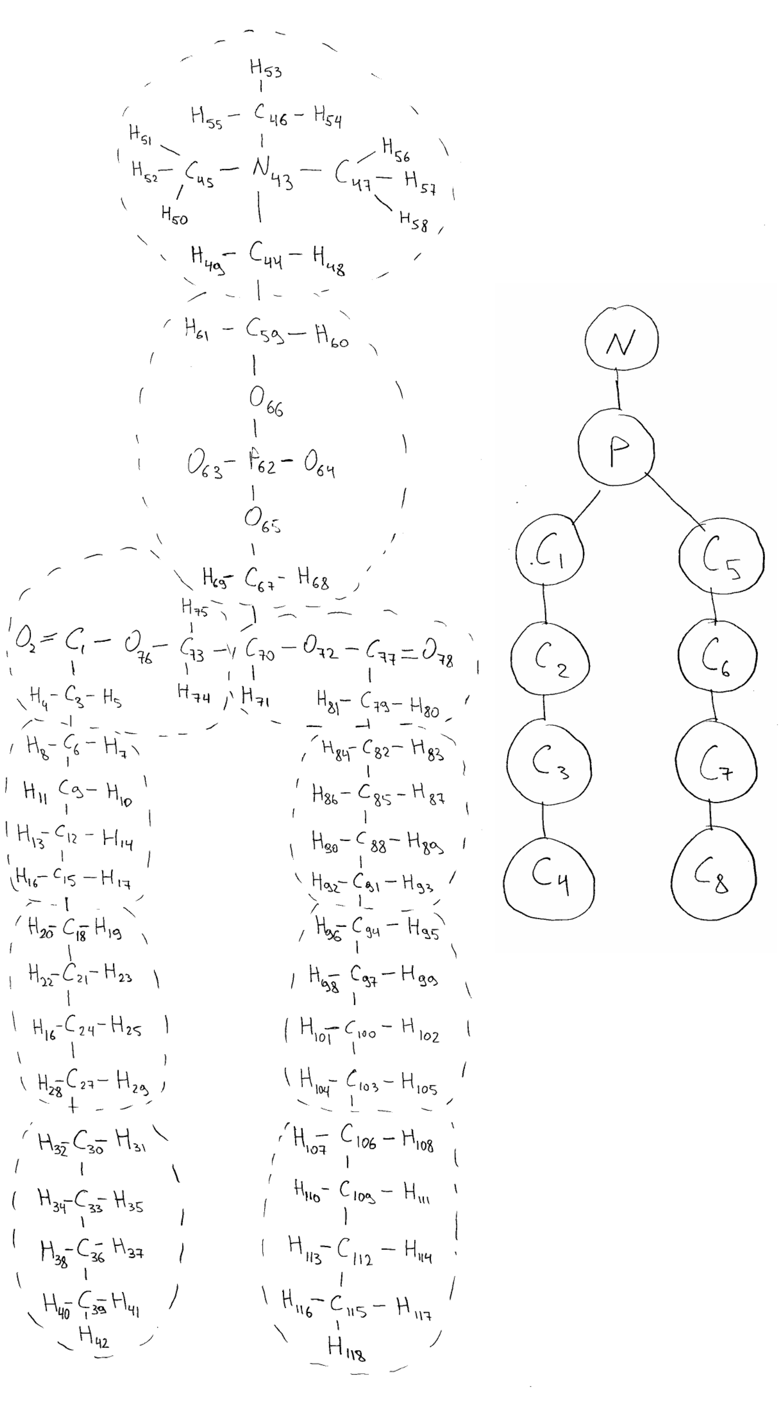

Figure 2: Simple illustration of mapping of DMPC phospholipid, which consists of 118 atoms, into 10-beads CG model.

- Parameter file defining the input trajectory, CG-atom types, assigning atoms pairs to different RDFs, defining bonds and angles in the coarse-grained model: *rdf.inp*

Output:

- File containing reference distribution functions: **.rdf* See format details in section RDF.
- Coarse grained topology files for each molecular type which are used in the main MC engine of MagiC **.mcm.* See details in section MCM.

Usage:

```
rdf-2.0.py -i rdf.inp
```

3.2.1 rdf.inp: main input file

RDF input file consists of three parts: Trajectory description, RDF-calculation parameters definition and interactions definition.

Trajectory: The section defines input trajectory file, in the same way as in *cgtraj*, but allowed file formats are **.xmol*, and GROMACS's **.trr, *.xtc.* The later two are defined by keyword `NFORM='TRR'` or `NFORM='XTC'`. It is supposed that input trajectory for $r$df module is already coarse-grained.

RDF-parameters: The format of this section was inherited from *rdf* utility of MDynaMix. Thus it looks like a NAMELIST section, but it accepts comment lines, and less strict to the syntax.

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

and contains the following parameters:

- `FOUTRDF = <filename>` The name of the output file
- `NATOMTYPE = <int.value>` Total number of different bead types in the system. Each pair of bead types has its own non-bonded interaction potential.
- `NRDF = <int.value>` Number of non-bonded RDFs to compute (a typical value $NATOMTYPE * (NATOMTYPE + 1)/2$ implying all-to-all interactions)
- `NRDFI = <int.value>` Number of intramolecular bond length distributions
- `NADF = <int.value>` Number of intramolecular bending angle distributions
- `RDFCUT = <value>` Cut-off distance for non-bonded RDF
- `RMI = <value>, RMAX = <value>` Maximum boundaries for intramolecular bond distributions

– `DELTAR = <value>` Resolution of the histogram for non-bonded RDF calculation (Å)

– `DELTARI = <value>` Resolution of the histogram for intramolecular bond distributions (Å)

– `DELTAPHI = <value>` Resolution of the histogram for intramolecular angular distributions (degrees)

Interactions: The section consists of a number of lines defining sites for reference distribution functions (DF). Since each DF will result in a separate interaction potential, the definition of groups of beads, which belong to a certain DF, is equivalent to definition of specific interactions in the system. First we introduce bead types (which can be called "coarse grained atom types") and specify CG beads belonging to each type. This is done by a list of lines written in format:
`<Name of CG-type>:<NameBead1 NameBead2 NameBead3>`,
with one line per each CG type. In total `NATOMTYPES` lines should be stated. **Important:** The CG interaction types (or CG force field types) introduced here will define interactions types for CG effective potentials and which will be further used in large-scale CG simulations. The order of appearance of CG interaction types in the list we will further refer as the index (or the number) of CG interaction type.

The DF definitions follow after the list defining CG interaction types. Each specific DF is specified by names of the molecular types involved in the DF, separated by `'--'`, and for intramolecular DF the molecular type number has to be specified as well. Then a single or several pairs (triplets in case of angle distribution) of sites which belongs to the DF are stated. If DF is specified by a single pair/triplet of sites, two/three number defining these sites are written in the input. The sites are specified by the "global CG site number" which is an index of the sequence:
`<CG sites of a molecule of type 1 > <CG sites of a molecule of type 2> ...`,
where the order of atoms in a molecule is determined by CG molecular topology files (.CG.mmol) obtained from the previous stage CGtraj (and which in turn is determined by the order of appearence of CG sites in the mapping scheme in *cgtraj.inp* file).

If a DF includes several pair of sites (in order to average them), it is written in the following way:

```
<Name of CG-type1>--<Name of CG-type2>
&<num of pairs>
<n1-1>  <n2-1>
<n1-2>  <n2-2>
...
(<NRDF> times for nonbonded RDF)

<Name of CG-type1>--<Name of CG-type2> <Num of molecular type>
&<num of pairs>
<n1-1>  <n2-1>
<n1-2>  <n2-2>
(<NRDFI> times for bond distributions)
```

```
<Name of CG-type1>--<Name of CG-type2> <Num of molecular type>
&<num of pairs>
<n1-1>  <n2-1> <n3-1>
<n1-2>  <n2-2> <n3-2>
...
(<NADF> times for angular distributions)
```

DFs within a group (introduced by `&<num of pairs>` statement) are averaged and counted as a single DF, and the corresponding sites will interact by the same potential. In case of triplets for angular distributions, the first two numbers `<n1-1>` `<n2-1>` defines the ends and the third number `<n3-1>` the middle of the angle.

CG sites of the same molecule included into list of bond or angle interactions are excluded from non-bonded RDF calculations, and they are not supposed to interact by a non-bonded potentials in the coarse-grained model.

3.2.2 Example: rdf.inp

This is an example of rdf.inp file which defines reference distribution function calculation for a 10-site CG DMPC lipid model shown on figure 3 which originates from the atomistic model displayed in figure 2.

```
 cat rdf.inp
# Trajectory description
 &TRAJ
 NFORM='XMOL',
 FNAME='cgtraj.mdyn',
 PATHDB='.'
 NTYPES=1
 NAMOL='dmpc_NM.CG',
 NSPEC=16,
 NFBEG=1,
 NFEND=1,
 ISTEP=1,
 IPRINT=5
# End of trajectory description section
 &END
# RDF calculation Description
 &RDFIN
# Number of intermolecular RDFs
 NRDF=10,
# Number of intramolecular bond length distributions
 NRDFI=5,
# Number of intramolecular bending angle distributions
 NADF=5,
# Output file name
 FOUTRDF='dmpc16-100aa-rdf-mdyn',
# Cutoff distance for intermolecular RDF: 20Å
 RDFCUT=20.,
# Boundaries for intramolecular bond length distributions
 RMI=0.0,
 RMAX=10.,
# Resolution of the histogram for intermolecular RDF calculation Å
 DELTAR=0.1,
# Resolution of the histogram for intramolecular RDF calculation Å
 DELTARI=0.02,
# Resolution of the histogram for intramolecular ADF calculation degrees
 DELTAPHI=1.0,
# Number of different bead types. Each pair of bead types has their
#own intermolecular interaction potential
NATOMTYPES=4,
# End of RDF calculation description section
 &END
# Third section - define interactions
# First we define coarse grained molecular types as a list
# CG-type name: Names of beads which belong to the CG-type
# CG mol type named "N" consists of one bead called "N"
N:N
P:P
# CG mol type named "CH" consists of six beads called
CH:C2 C3 C4 C6 C7 C8
CO:C1 C5
#  List of Intermolecular RDFs : There are 10 different RDFs, as stated above in NRDF
#
# Pair of CG-type names separated by "--"
N--N
# Pair of atom numbers that belongs to N--N interaction
  1  1
#
N--P
  1  2
# N--CH intermolecular interaction
N--CH
# six pairs of atoms belong to N--CH interaction
& 6
# list of atom pairs
  1 4
  1 5
  1 6
  1 8
  1 9
  1 10
N--CO
& 2
  1 3
  1 7
P--P
  2  2
P--CH
```

```
& 6
  2 4
  2 5
  2 6
  2 8
  2 9
  2 10
P--CO
& 2
  2 3
  2 7
CH--CH
& 21
  4 4
  4 5
  4 6
  4 8
  4 9
  4 10
  5 5
  5 6
  5 8
  5 9
  5 10
  6 6
  6 8
  6 9
  6 10
  8 8
  8 9
  8 10
  9 9
  9 10
  10 10
CH--CO
& 12
  4 3
  4 7
  5 3
  5 7
  6 3
  6 7
  8 3
  8 7
  9 3
  9 7
  10 3
  10 7
CO--CO
& 3
```

```
  3 3
  3 7
  7 7
#  List of Intramolecular pairwise bond length distribution function
# Bond N--P, which belongs to the molecular type 1
N--P 1
# Two atom pairs belongs to this bond
  1 2
P--CO 1
& 2
  2 3
  2 7
# Bond CH--CH, which belongs to the molecular type 1
CH--CH 1
# Four atom pairs belongs to this bond
& 4
  4 5
  5 6
  8 9
  9 10
CH--CO 1
& 2
  3 4
  7 8
CO--CO 1
  3 7
#  List of Intramolecular ADFs
# Angle bending bond N--P--CO, which belongs to the molecular type 1
# Names are given for the end atoms of the angle
N--CO 1
# Two atom triplets belongs to this bond
& 2
# Triplets. The central atom in the angle is given third in the triplet
  1 3 2
  1 7 2
P--CH 1
& 2
  2 4 3
  2 8 7
CH--CO 1
& 2
  3 5 4
  7 9 8
CH--CH 1
& 2
  4 6 5
  8 10 9
CH--CO 1
& 2
  4 7 3
```

8 3 7

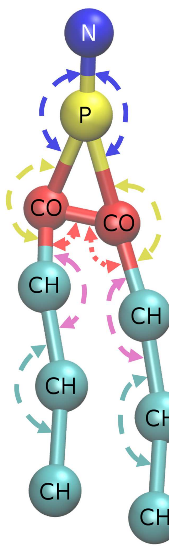

Figure 3: Example: 10-beads CG model of DMPC-lipid. Beads and bonds of same color have same type; solid lines denote covalent bonds; dashed arrows denote angle bending bonds.

3.3 *magic*: Inverse Solver IMC/IB

This is the main part of the package. It performs iterative Metropolis Monte-Carlo sampling of the system described by a trial set of potentials, then compares sampled distribution functions with the reference ones, and computes correction to the set of potential. Then the new iteration starts, with the updated set of potentials. The process is repeated by a given number of iterations.

Input:

- Parameters of the Monte-Carlo sampling and inverse solver: *magic.inp*
- Coarse grained topology files, one file for each CG-molecular type: **.mcm*
- Reference distribution functions: **.rdf*
- Initial structure for MC-process (optional). In a parallel run a separate structure file should be provided for each parallel process: *name-of-the-system.p<process-number>.i<iteration-number>.start.xmol*
- Starting interaction potential (optional) **.pot.* If not stated, the potential of mean force or zero non-bonded potentials will be used at start. See format details in section .pot.

Output:

- Log file for each parallel process: name-of-the-system.p<process-number>
- General log file collecting summary information from all parallel processes (as a standard output)
- Updated effective potentials after each iteration: *name-of-the-system.i<iteration-number>.pot*
- Computed correction to effective potential obtained after a certain number of MC steps: **.potcorrcheck* (optional). This can be used to control convergence of the MC sampling at each iteration.
- Monte-Carlo trajectory of each parallel process (optional): *name-of-the-system.p<process-number>.xmol*
- The latest 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 continuation of MC run. In case of a parallel MC run, a set of files for each processor is generated.

Usage:

serial execution:
```
magic magic.inp
```

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

3.3.1 magic.inp: main input file

The input file should be written in strict format for "Namelist" type of input in Fortran program. The following parameters need to be defined:

System parameters:

NTYP Number of different molecule types (species) present in the system.

PATHDB Directory with molecule description files .mcm

NAMOL Names of molecules present in the system. Every molecule type should have a description file, with the corresponding name and extension .mcm, located in PATHDB directory. E.g. `NAMOL = 'DMPC','CHOL'` defines 2 names: `DMPC`-for the first molecule type, and `CHOL` for the second one. The description files should be named `DMPC.mcm` and `CHOL.mcm`.

NSPEC Number of molecules of each type, written as a comma-separated list,(1,2,...,NTYP). For example, `NSPEC=392,3,` defines a system consisting of 392 molecules of the first type and 3 molecules of the second type.

LMOVE Which molecules are allowed to move. List of comma-separated logical values. Default `LMOVE = .true., .true.,`, i.e. all molecules are moving. Frozen molecules coordinates has to be specified in a **.xmol* file given by `FCRD` parameter.

NA Number of grid points for non-bonded interactions (RDFs and effective potentials). This value should be the same as the one stated in reference distribution function file, and in the input potential file if it is used in the input.

EPS Dielectric permittivity constant defining electrostatic interactions in the system.

TEMP Temperature of the system, K

BOXL,BOYL,BOZL ! Periodic cell dimensions, Å. The software uses rectangular periodic boundary conditions and NVT ensemble.

Monte Carlo sampling parameters:

NMKS The total number of Monte Carlo steps to be performed at each iteration per processor (including equilibration).

NMKS0 The number of Monte Carlo steps for equilibration (not included into averaging of the distribution functions).

LRDF Whether to read reference distribution functions file. If .true., the inverse mode is ON, the program will update the starting potential to match the given RDF. Otherwise it will perform a standard MC simulation with given potential. Default `LRDF=.true.,`

FILRDF Name of the input file with reference distribution functions

LPOT Whether to read trial potentials from a file (if `LPOT=.true.`), or to generate them from RDF as pontential of mean force $U(r) = -k_B T \ln(g(r))$ (if `LPOT=.false.`). If `LPOT=.false.`, `LRDF` should be set to true: `LRDF=.true.,` and .rdf file must be present.

FILPOT Name of the file with a set of starting potentials (not required if LRDF=.t. and LPOT=.f.)

LZEROPOT This option has an effect only in case (if `LPOT=.false.`). In case of `LZEROPOT=.true.`, the non-bonded trial potential is set to zero in the whole range where the non-bonded RDFs are not zero. The bonded potentials are still initiated as potentials of mean force. This option provides often a smoother initial iterative process than starting from the potential of mean force for non-bonded interactions.

LCRD If .true., starts the MC process from coordinates given in `FSTART` file. Otherwise, start MC from random configuration by using local coordinates from .mcm files and assigning random molecular center of mass positions and orientations. Default: `LCRD = .false.`

LCRDPass If .true., provides passing of final configuration of the MC sampling at each iteraction to the next iteration. Otherwise at the beginning of each iteration, a starting configuration will be generated randomly, or read from `FSTART` file (as given by `LCRD` parameter). Default: `LCRDPass = .true.,`

IAV How often to compute DFs and the cross-correlation term of the IMC algorithm. Computation of the cross-correlation term is rather expensive, that is why it is not advisable to use it too often. A good value for this parameter can be about the total number of CG sites in the system.

IOUT How often to recalculate the total energy, pressure and write the energies to the log-file. Within the program, the energy of the system is permanently updated after each MC step (since the energy difference is computed at each MC step). When the full recomputation of energy takes place, the new value is compared with the value hold by the program. If the difference is larger than 0.01 KT a warning message is given.

RECUT Cutoff for the electrostatic energy in the real-space Ewald sum. It should not be less than the RDF cutoff.

AF, FQ Ewald summation parameters. Within the Ewald summation method, the total electrostatic energy is expressed as

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

where, $\alpha = \frac{AF}{r_{cut}}$, and $k^2_{cut} = 4\alpha^2 \, FQ$. With these notations, the precision or the first sum is defined by $erfc(AF)$, while accuracy of the second sum is defined by $exp(-FQ)$. Default: `AF= 3.0, FQ=9.0`

DR Maximal displacement in a single atom displacement MC step, Å. Default `DR = 1.0,`

MCTRANSSTEP Maximal displacement in MC translation of the whole molecule, Å. Default `MCTRANSSTEP = 0.2,`

ITRANS How often (after which number of single-atom steps) to perform random translation of a random molecule. Default `ITRANS=0`, i.e. never. Single-atom molecules are not included into translation step (they are moved only by single-atom displacements)

MCROTSTEP Maximal angle of rotation of the whole molecule (degree). Default `MCRotStep=0.1`

IROT How often (after which number of single-atom steps) to perform random rotation of a random molecule (excluding single-atom). Default `IROT=0`, i.e. never

NRS Initial random seed for the random number generator.

Input-Output parameters:

BASEOUTFILENAME Base name (filename template) to use for writing output files. All names of output files will begin with —BASEOUTFILENAME—.

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

LPOUT If true, perform an input test: read all input data, initialize all data structures then finish. Default `LPOUT = .false.,`

IXMOL Whether to write trajectory files: 1 - yes, 0 - no. Default `IXMOL = 0,`. Trajectory file is written in XMOL format.

ITR How often to write current system's geometry to the trajectory file. Default `ITR=100,`

FSTART Name of input file (set of files) with starting coordinates, excluding 'p001.start.xmol' part.

FCRD Input file file with starting coordinates of frozen molecules (if they are present).

LXMOL If true, program dumps the last configuration of MC process in "start.xmol" file. It is done after every inverse iteration on every parallel process. Output filename is
`BASEOUTFILENAME.i<iteration>.p<process>.start.xmol`

Inverse solver parameters

LIMC Inverse solver selection. If `LIMC = .true.,` Inverse Monte Carlo method is used, otherwise iterative Boltzmann inversion is used. Default `LIMC = .true.,`

IREPT Number of inverse iterations to perform. Default `IREPT=1,`

DPOTM Cutoff for the change of potential value at every point during the correction procedure. kT units. Default `DPOTM = 5.,`. Recommended (safer) value: `DPOTM = 2.`.

REGP Regularization parameter for potential correction. This parameter scales down the difference between calculated and reference RDFs which is used in the inversion procedure, and has value between 0 and 1. In case of disconvergence, expressed in alternating signs of corrections to the potential at each next iteration, value of `REGP` need to be decreased. Also, if corrections to the potential systematically exceed a few kT units (or potential cutoff value set by `POTM` parameter) the valus of `REGP` should be decreased. Too small values of REGP will however lead to very slow convergence, and values below 0.01 are not recommended. Default (but seldom optimal) is `REGP=1.`.

IPotCorrCheck How often (after which number of MC steps) to perform potential correction check. The program gathers accumulated statistics from all the processes, and then calculates sampled distribution functions and corrections to the potential. However, these corrections are not applied, but just printed to the log file, while the run is continued with starting values of the interaction potential. This allows user to analyze how well the distribution functions and potential corrections are converged on every inverse iteration. The checks are performed after equilibration. Default `IPotCorrCheck=0,` i.e., no check.

POTCUT Relatively high value of the non-bonded potentials (in kJ/mol) representing a nearly hard wall in areas where RDF=0. Default `POTCUT=1000.`

3.4 *MagicTools*: Analysis of IMC results

This part of the package is completely python-based, and it should be run in a python interpreter. We recommend to use *ipython* as an interpreter, but standard python should also work. Once you have started ipython, you need to import MagicTools module. To do this type: `import MagicTools`, if no error message appeared, import completed correctly.

Analysis usually include few phases: Import of data from output files; Plotting the data; Numerical analysis/evaluation of the data; Export of the data; Below we will discuss how these actions can be taken by MagicTools.

3.4.1 Importing data

MagicTools can be used to load (for further analysis and visualization) reference distribution functions (both from .rdf file `GetDFsFromFile_rdf` and magic output file `GetDFsFromFile_magic`), computed DFs (from magic output `GetDFsFromFile_magic`), effective potentials (both from .pot file `GetPotsFromFile_pot` and magic output file `GetDFsFromFile_magic`), corrections of the potentials in the iterative process (magic output `GetDFsFromFile_magic`). You can also load previously saved list of distribution functions using `LoadDFs`. The import procedures are named by origin of the target file, and they return a list of requested distribution functions, which are instances of class `CDF`.

Typical import of data from *.rdf-file looks like:
`rdf=MagicTools.GetDFsFromFile\_rdf('dmpc16.rdf')`
Here all radial distribution functions and bond distribution functions will be read from file *dmpc16.rdf* and a resulting list of functions will be assigned to a variable *rdf*.

3.4.2 Plotting data

Once the data are loaded, they can be visualized via plotting. MagicTools have two plotting functions:
`PlotAllDFs`: It takes a nested list of distribution functions (RDF, potentials) and plot these functions in relevant groups. This is the easiest way to plot and compare several sets of DFs, for example from different iterations of IMC.
`PlotDFs`: This function simultaneously plots all functions which are given in list 'DFs' on the single plot.

3.4.3 Numerical analysis

MagicTools has a few procedures performing simple analysis of previously imported data or output files generated by magic. `Deviation` reads and plots total deviation between reference and computed DF obtained on different iterations of the inverse procedure. `AnalyzeIMCOuput` plots the reference and computed DFs obtained in inverse procedure. They both require magic's output file as input. `TotalPots` creates a set of total non-bonded potentials by adding the electrostatic term to short-range intermolecular potentials. `GetOptEpsilon` calculates the optimal value of dielectric permittivity which provides the fastest decay of the short-range intermolecular potentials [4]. `PotsEpsCorrection` creates a new set of short-range potentials with the corrected value of dielectric permittivity.

`PotsPressCorr` creates a list of new short-range potentials by adding a decaying linear term, which can be used to obtain "pressure-corrected" potentials providing correct total pressure in the CG system [5].

3.4.4 Exporting data

At present three operations are available: `DumpDFs`: dumps a list of DFs to a binary dump file, which can be read by `LoadDFs`; `SaveDFsAsText` saves a distribution function from a given list to a separate text-file; `PotsExport2Gromacs` exports a list of potentials to a GROMACS's .xvg format.

3.5 MagicTools procedures reference:

3.5.1 GetDFsFromFile_rdf(filename,basename='prefix')

Import a set of radial distribution functions and bond length/angle distributions from rdf-file *filename* produced by rdf-2.0.py tool. You can also specify an optional prefix `basename='some string'` to be given to all DFs read from the file. If basename is omitted, input filename will be used.
Example: `RDFs=MagicTools.GetDFsFromFile_rdf('dmpc16.rdf')`

3.5.2 GetDFsFromFile_magic(filename,iters=(),DF='',mtype='',basename='')

Analyse output file *filename* produced by MagiC and return a list of DFs for every required iteration. Every single distribution function can be accessed as `DFs[iteration number][pair number]`

filename - name of the file to import (mandatory argument).

iters - set of iterations which should be extracted from the file (optional argument). By default all iterations will be extracted. Example: iters=(1,2,3) or iters=(1,) Mind the comma in case of a single iteration number

basename - prefix given to all the DFs read from the file (optional argument). By default a name of the input file will be used.

DF - which kind of function to import (optional argument): `DF='RDF'` results in import of radial distribution functions and bond distribution functions obtained in iterative inverse Monte Carlo calculation; `DF='RDFref'` - import reference RDFs (ADFs); `DF='Pot'` - import potentials used in MC simulation on every iteration; `DF='PotNew'` - import effective potentials generated on every iteration. Default `DF='RDF'`.

mtype - Define the list of CG sites (optional argument). It defines the list of pairs/triplets of atomic names to search in output file for parsing RDF,potentials, etc. By default the program will automatically build the list, but if you have more than one interactions for the same type of a CG-atom, it is recommended to manually edit the subroutine in file MagicTools.py and add a list of beads specific to your system. The list consists of three sublists: first of them refer to non-bonded pair interactions, the second refers to bonded pair interactions, and the third refers to bending angle (1-3) bond interactions.

Examples:

`RDFs=MagicTools.GetDFsFromFile_magic('02.magic.out',DF='rdf',iters=(1,2))`
  This line will import two subsets of computed RDFs and bond length/angle distributions, obtained after 1-st and 2-nd iteration of the inverse procedure.

`potentials=MagicTools.GetDFsFromFile_magic('02.magic.out',DF='pot')`
  This line will import the whole set of potentials used on each iteration reported in the file '02.magic.out'.

`potentials=MagicTools.GetDFsFromFile_magic('02.magic.out',DF='newpot',mtype='dmpc')`
  This line will import the whole set of potentials obtained after each iteration reported in the file '02.magic.out'. A predefined list of bead names will be used for parsing of file '02.magic.out'.

3.5.3 GetPotsFromFile_pot(filename, basename=", mcmfile=", Ucut=10000.0)

Import a set of potentials from *.pot file *filename* produced by MagiC. Return a list of potentials.

filename - name of the file to import (mandatory argument).

basename - prefix given to all the potentials read from the file (optional argument). By default a name of the input file will be used as a mark.

mcmfile - *.mcm molecular structure file to read bead names (optional argument). If not set, a warning will be given, and bead numbers will be stated instead of types.

Ucut - Criteria to recognize artificial core part of the potential (optional argument).

Example:
`pots=MagicTools.GetPotsFromFile_pot('01.dmpc16.pot',mcmfile='dmpc_NM.CG.mcm')`

3.5.4 LoadDFs(filename)

Loads a set of DFs from a file previously dumped with `DumpDFs`.

filename - name of the file to load (mandatory argument)

Example: `rdfs=MagicTools.LoadDFs('rdf.dmp')`

3.5.5 PlotAllDFs(listDFs, hardcopy=False, title=", linetype=", nolegendintra=False, figsize=(20,14), dpi=80)

It takes a nested list of distribution functions (RDF, potentials) and plot these functions in relevant groups.

listDFs - nested list of distribution functions, e.g. it is a list which keeps a few sublists with distribution functions inside (mandatory argument).
$[[f_1^{(1)}(r), f_2^{(1)}(r), f_3^{(1)}(r)], [f_1^{(2)}(r), f_2^{(2)}(r), f_3^{(2)}(r)], ..., [f_1^{(n)}(r), f_2^{(n)}(r), f_3^{(n)}(r)]]$

hardcopy - Defines that the plots should be saved as *.eps files (optional argument). Default value - False, no eps copies are made.

title - Prefix used in a title of each plot (optional argument).

linetype - String defining a type and color of lines according to matplotlib syntax (optional argument). See more about syntax here: matplotlib tutorial

nolegendintra - If set, a legend on the intramolecular plots will be omitted (optional argument). Default: False (show the legend everywhere). This option might be useful, when the legend overlaps with curves on the plot, usually it can happen when plotting intramolecular DF/potentials.

figsize - Size of the plot in inches (x,y) (optional argument). Default size is (20,14).

dpi - Resolution of the plot in dpi (optional argument). Default value: 80 dpi.

Example:
`imc=MagicTools.GetDFsFromFile_magic('magic.out')` - import RDFs obtained on every iteration of IMC to a list `imc`
`MagicTools.PlotAllDFs(imc)` - Plot RDFs from the list at the same plot grouped by origin (pair of beads involved in function).
`MagicTools.PlotAllDFs([rdfref]+[imc[4]],hardcopy=True,`
 `title='RDF convergence in IMC',linetype='.')` Plot RDFs calculated in IMC at the same plot together with reference RDFs, and save copies to eps-files. Plots will be drawn with dots instead of lines, and a title will be added to each plot.

3.5.6 PlotDFs(DFs,hardcopy=False, title='', linetype='', nolegendintra=False, figsize=(20,14), dpi=80):

Plots all distribution functions stated in a list *DFs* on a single plot.

listDFs - list of distribution functions (mandatory argument).
$[f_1^{(1)}(r), f_2^{(1)}(r), f_3^{(1)}(r)]$

hardcopy - Defines that the plots should be saved as *.eps files (optional argument). Default value - False, no eps copies are made.

title - Prefix used in a title of each plot (optional argument).

linetype - String defining a type and color of lines according to matplotlib syntax (optional argument). See more about syntax here: matplotlib tutorial

nolegendintra - If set, a legend on intramolecular plots will be omitted (optional argument). Default:False - show the legend everywhere. This option might be useful, when legend overlaps with curves on plot, usually it can happen when plotting intramolecular DF/potentials.

figsize - Size of the plot in inches (x,y) (optional argument). Default size is (20,14).

dpi - Resolution of the plot in dpi (optional argument). Default value: 80 dpi.

Example:
`MagicTools.PlotDFs(rdfref[0:5])` Draw first 5 functions from list `rdfref`

3.5.7 Deviation(filename,hardcopy=False)

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

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

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

Examples:
```
MagicTools.Deviation('01.magic.out')
MagicTools.Deviation(['01.magic.out','02.magic.out'],hardcopy=True)
```

3.5.8 AnalyzeIMCOuput(filename,mtype=”,iters=(),hardcopy=True)

Analyze output file produced by MagiC and plot resulting DFs of interest and reference DFs.

filename - name of the output file to analyse (mandatory argument)

iters - tuple of iterations which should be extracted from the file (optional argument). By default all iterations will be extracted. Example: iters=(1,2,3) or iters=(1,) Mind the comma in case of a single iteration number

mtype - type of the molecules in system (optional argument). It defines list of pairs/triplets of atomic names to search in output file for parsing RDF,potentials, etc. By default the program will automatically build the list, but if you have more than one interactions for same type of CG-atom, it is recommended to manually edit the subroutine in file MagicTools.py and add list of beads specific to your system. The list consists of three sublists: first of them refer to non-bonded pair interactions, the second refers to bonded pair interactions, and the third refers to bending angle (1-3) bond interactions.

hardcopy - if plots should be saved to a .eps files (optional argument). Default - True

Example:
```
MagicTools.AnalyzeIMCOuput('01.magic.out')
```

3.5.9 TotalPots(pots,eps)

Creates a set of total potentials by adding electrostatic interactions to short-range 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 non-bonded potentials, while bond potentials (both pairwise and angular) will be kept the same.

pots - list of potentials (mandatory argument)

eps - dielectric permittivity ϵ of implicit solvent used in inverse Monte-Carlo simulation (mandatory argument).

Example:
`pots=MagicTools.GetPotsFromFile_pot('01.dmpc16-100a.i010.pot', mcmfile='dmpc_NM.CG.mcm')` - importing potentials.
`TotalPots=MagicTools.TotalPots(pots,70)` - Creating total potentials

3.5.10 GetOptEpsilon(pots,eps_old,r1,eps_min=0,eps_max=0,npoints=100)

This procedure implement methodology to compute optimal value of the dielectric permittivity in an ion-like system providing fastest decay of short-range intermolecular potentialas as described in ref. [4]. A brief description of the procedure is given here. First, we introduce a numerical criteria of a short range potential deviation from zero at longer distances:

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

where r_1 and r_2 are the lower and upper boundaries of the range of distances defining the tail (by default the r_2 value is taken as the cut-off of RDFs and tabulated effective potentials). 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 short range potentials between charged species 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 analyse (mandatory argument) NB! The dielectric permittivity value calculation only takes the non-bonded 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 $\frac{\epsilon_{max}-\epsilon_{min}}{npoints}$

Example:
`pots=MagicTools.GetPotsFromFile_pot('01.dmpc.pot',mcmfile='dmpc.mcm')`
importing potentials.
`eps_opt=MagicTools.GetOptEpsilon(pots, 70.0, 15, eps_min=50, eps_max=100, npoints=1000)`
getting optimal epsilon

3.5.11 PotsEpsCorrection(pots,eps_old,eps_new)

This procedure creates a new set of potentials, with short-range intermolecular potentials corresponding to a changed value of the dielectric permittivity according to eq.5

pots - set of potentials to analyze (mandatory argument) NB! The correction only affects intermolecular potentials.

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

eps_new - new value of dielectric permittivity.

Example: `newpots=MagicTools.PotsEpsCorrection(pots,70,100)`

3.5.12 PotsPressCorr(pots,U_corr0)

This procedure creates a new set of short-range potentials by adding a decaying linear term to each non-bonded potential in a given set. Such a correction can be used to fit a correct pressure in the CG simulation as described in ref. [5]. The 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 - set of potentials to analyze (mandatory argument) NB! The correction only affects intermolecular potentials.

U_corr0 - Magnitude of the correction, kJ/mol (mandatory argument)

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

3.5.13 GromacsTopology(mcmfile,topfile)

Creates a Gromacs topology file *.top from a given list of mcm-files. Number of molecules in the resulting system should be stated manually once the top-file is created.

mcmfile - Name of the mcm-file or a list of such files (mandatory argument).

topfile - Name of the output GROMACS topology file (mandatory argument)

Examples:
`MagicTools.GromacsTopology('dmpc.mcm','dmpc.top')` - system of single molecular type
`MagicTools.GromacsTopology(('dmpc.mcm','dopc.mcm'),'dmpc_dopc.top')` - system having two different molecular types

3.5.14 DumpDFs(DFs,filename)

Dumps a given set of DFs into a file. The file can be read using `LoadDFs`.

DFs - set of distribution functions to dump (mandatory argument)

filename - name of the dump-file (mandatory argument)

Example: `MagicTools.DumpDFs(pots,'pots.dmp')`

3.5.15 SaveDFsAsText(DFs)

Save every function from a given list DFs into a separate text-file. Each function is saved in a tabulated format: First column - distances in Å, second column - values. The text file has the same name as according distribution function.

DFs - set of distribution functions to save (mandatory argument)

Example: `MagicTools.SaveDFsAsText(pots)`

3.5.16 PotsExport2Gromacs(pots, ofilename=”, zeroforce=False, npoints=2500, Umax=6000, Rmaxtable=2.5, sigma=0.5, interpol=True, noplot=False, hardcopy=True, figsize=(14,8), dpi=120)

This procedure exports a set of potentials into GROMACS .xvg format. Such an export is a few steps process, detailed description of which follows below. In general the question is following: we start with a tabulated potential which is defined on a r-grid $[r_{min} : r_{max}]$ with a given density; and as a result we need to obtain a tabulated potential, which is defined on a wider r-grid $[0 : r_{vdw} + r_{table-extension}]$ with a larger density as required by GROMACS for tabulated potentials. In order to do that, we need to introduce first left-side and right-side extensions of the potential which should be smoothly connected to the original potential. The left side extension should represent a strongly repulsive core, which is approximated by

$$U^{left}(r) = ar^2 + br + U_{max} \tag{7}$$

and coefficients a and b are chosen to provide continuity of $\frac{d}{dr}U(r)$ at $r = r_{min}$. For the right side extension we should take into account an origin of the potential. Short-range non-bonded potentials should decay to zero when $r > r_{max}$ which is achieved by:

$$U^{right}_{short\,range}(r) = U(r_{max}) \cdot \exp[-10\frac{(r - r_{max})}{r_{vdw+table-extension} - r_{max}}] \tag{8}$$

Here 10 is a pre-defined coefficient, $r_{vdw+table-extension}$ is a range of the table required by GROMACS.

Angle bending bond potentials should also decay to zero at $\phi = 180°$. The following smoothing expression is applied:

$$U^{right}_{angle}(\phi) = U(\phi_{max}) \cdot \exp[-100\frac{(\phi - \phi_{max})}{180° - \phi_{max}}] \tag{9}$$

In contrast, pairwise bond potentials should have attractive wall at the right side, which is approximated by harmonic wall in the same way as the repulsive wall:

$$U^{right}_{pair\,bond}(r) = ar^2 + br + U_{max} \tag{10}$$

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

Once the extension has been made, we interpolate all the points to a denser grid. Number of nodes in the grid is defined by `npoints`. If `interpol=False` a grid of the original density is used. The interpolation is made by Gaussian smoothing, e.g. every point of the resulting potential is obtained as:

$$U^{new}(r) = \frac{1}{Z(r)} \sum_{r_i=r_{min}}^{r_{max}} U^{orig}(r_i) \cdot \exp\frac{-(r - r_i)^2}{2\sigma^2} \tag{11}$$

$$Z(r) = \sum_{r_i=r_{min}}^{r_{max}} \exp\frac{-(r - r_i)^2}{2\sigma^2} \tag{12}$$

where σ defines how broad is the averaging. By default $\sigma = 0.5\Delta r_i$

After the interpolation is done, each resulting potential and force based on it are written to xvg-file. The xvg-file is named by name and type of the corresponding original potential. 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.

In order to control the results, each original potential, the extrapolated part and interpolated potential are plotted together. The plotting is controlled by parameters `noplot, hardcopy, figsize, dpi`, which are explained below.

pots - set of potentials to export into GROMACS .xvg format (mandatory argument)

ofilename - Prefix used for naming of the output .xvg files (optional argument).

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

npoints - number of points in the resulting table in .xvg file (optional argument). Default value - 2500 points.

Umax - height of the potential wall at r=0 in case of non-bonding potential and at $r = r_{min}$, $r = r_{max}$ in case of bonding potential (optional argument). Default value 6000 kJ/mol

Rmaxtable - cutoff range of the resulting potentials (both intermolecular and pair bonds) in nanometers (optional argument). Default value is 2.5 nm. Note that GROMACS requires tabulated potentials to be defined up to r=rvdw+table-extension (in terms of GROMACS parameters).

sigma - Gaussian smoothing parameter measured in resolution of the original potential (optional parameter). Default value 0.5

interpol - if the potentials should be interpolated, otherwise original resolution of the table will be kept and `npoints` value will be ignored (optional argument). Default - True, potentials are interpolated.

noplot - if interpolated potentials potential/forces should be plotted (optional argument). Default: False - the graphs are plotted.

hardcopy - defines that the plots should be saved as *.eps files (optional argument). Default value - True.

figsize - size of the plot in inches (x,y) (optional argument). Default size is (14,8).

dpi - resolution of the plot in dpi (optional argument). Default value: 120 dpi.

Examples:
`MagicTools.PotsExport2Gromacs(pots)` - simplest call with default parameters.
`MagicTools.PotsExport2Gromacs(pots[0:10], sigma=0.1, zeroforce=True, interpol=False )` - export only first 10 potentials from the set; do not use denser net for interpolation; use averaging with sigma=0.1 (very narrow gaussian); do not export forces, write zeros instead.

3.5.17 C_DF - Object Class representing Distribution Function

C_DF is a object-oriented class, which suppose to reproduce basic features of any distribution function (e.g. RDF, bond length distribution, angle distribution, intermolecular potential, angle-bending potential, correction to potential).

The class has a few properties and methods, which are common for every DF:

Properties:

Name - Name of the distribution function

type - Type of the distribution function: It is either 'inter' for intermolecular DF and 'intra' for intramolecular.

Npoints - Number of points in a table defining the function.

rmin,rmax - Range of distances/angles where the function is defined

resol - Resolution of the tabulated function

g[:, 0:2] - Table (numpy array) defining the function. g[:,0]-keeps argument (r,angle), g[:,1]-keeps function's value

Methods:

Normalize() - Calculate values of g[:,0:2] by normalizing accumulated histogram. Only available with rdf-2.0

Smooth() - Smooth values of g[:,1] using Savitzky Golay 5 points smoothing procedure

Trim(tolerance) - Cut function's (g[:,1]) left and right tails which have values smaller then ¡tolerance¿

Plot() - Plot the function, using matplotlib library

Save() - Write the function in a tabulated format to a text file.

3.6 Additional small tools

3.6.1 gro2xmol.py

Converts `*.gro` file to a `*.xmol` file. **Usage:** gro2xmol.py -i input.gro -o output.xmol

3.6.2 xmol2gro.py

Converts `*.xmol` file to a `*.gro` file. **Usage:** xmol2gro.py -i input.xmol -o output.gro -molnames "('DMPC','WAT')" -nmols "(98,2700)" -natimol "(118,3)" Since the later one has more information about system's structure, this script requires additional input:

-molnames - tuple of molecular types that are present in the system

-nmols - tuple stating how many molecules of each type are present in the system

-natimol tuple stating how many atoms each molecular type has

3.6.3 gdevi

Simple tool to check a convergence of iterative inverse process. It is nothing else, but a shortcut to a grep, which allows to parse deviation in inverse process.
`alias gdevi='grep -i --text '\''devi'\'''`
Usage: `gdevi magic.out`

4 File formats

4.1 .xmol

This is plain text trajectory file format, which can be produced by many computational chemistry software packages (sometimes is references as xyz). 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: Some commentary text. In MD trajectories used in the input to MagiC this line is supposed to have a form: after *time stamp* some text BOX: *box-x box-y box-z*

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.

The length unit is Å, and time unit is femtoseconds. MagiC can understand *.xmol* trajectory with commentary lines without time step and box sizes, but in the latter case the box sizes should be specified in the input file (*cgtraj* and *rdf* modules) and the simulations imply NVT-ensemble.

4.2 .mmol

.mmol is a molecular topology file format, which is used in MDynaMix MD software. It consists of two parts: first part describes atomic composition, geometry, charges, masses and non-bonded interactions; the second part defines bonded interactions in the molecule. MMOL-topology files are required by cgtraj (subsection 3.1) for converting high-resolution trajectory to a coarse grained one and for calculation of reference distribution functions rdf (subsection 3.2). In both cases only information about the atomic structure is used, so the bonding part may be omitted.

The first part of the file has the following structure: The lines started with '#' are commentaries 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 (effective atom diameter) in Å, 8) Lennard-Jones parameter in kJ/M. Two optional columns may present, the 9-th column with the chemical types of atoms according to the chosen force field and the 10th, with the atom numbers.

4.2.1 .mmol File Example (original): H2O.mmol (SPC-water)

```
#=================================================I
#       MDynaMix molecular Data Base              I
#       Configuration and interaction potential   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.
#    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
```

4.2.2 "fake" .mmol File Example:

For the purposes of MagiC, the only information which is needed is the number and type of atoms, their charges and masses. In case of trajectories prepared by other packages, a fake .mmol file can be created containing only the first part, with any numbers (e.g., zeroth) for the atom coordinates and LJ parameters. For example, the shown above true file for SPC water can be used in the following form:

```
 3
O       0.    0.    0.  15.9994      -0.82       0.        0.
H1      0.    0.    0.   1.008        0.41       0.        0.
H2      0.    0.    0.   1.008        0.41       0.        0.
```

4.3 .mcm

Coarse grained topology file, which is similar to CG.mmol, but includes information about non-bonded potential types and intramolecular bonds and angles. These files are created by *rdf* module of MagiC, but can be also prepared and used independently. In general a *.mcm* file consists of three parts: atom composition and geometry section, covalent-like bond section and angle-bending bond section. A separate *.mcm* file should be provided for each molecular type present in the system.

Format of *.mcm* file:

Lines beginning with '#' or '!' are commentaries.

The first non-commentary line of a *.mcm* 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 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) Index of the interaction type this atom belongs to, 8) Name of the atom species/type this atom belongs to (CG force field atom type). Atoms of the same atom type interact by the same non-bonded potential.

Important: The index of the interaction type is a global number for the whole system. CG sites having similar fine-grained structure can be described by the same interaction type, then they have the same interaction index. Different types of molecules may have CG sites with the same interaction type.

Next line, coming after atoms section, defines the number of covalent-like bond types in the molecule. For each type of a covalent bond we need to specify: Number of atom pairs which belong to a bond of this type; List of triplets, where 1-st and 2-nd numbers define which atoms are connected by this bond and the 3-rd number is always 1 for a covalent bonds (this field is reserved for future use). In difference from the non-bonded section, the CG sites in the list of bond are given as CG atom numbers defined by the order of their appearence in the first part of the .mcm file, irrespectively of their non-bonded interaction type.

The angle bending section is similar to covalent-like bond section. It also starts with a line defining a total number of angle bending bond types. For each type of angle bending bond we need to specify: Number of atom triplets which belong to a bond of this type; List of triplets (CG atom number) defining which atoms are connected by this bond. The triplet has unusual order: central atom stands last in the triple, e.g. triplet 1 3 2, defines angle between 1-2 and 2-3. Thses numbers are given as global CG site numbers.

4.3.1 .mcm Example: DMPC.CG.mcm

The mcm-file listed below defines 10-beads model of DMPC-lipid, as shown on figure 3.

```
#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
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  1
# 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  1
2  10  1
# 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  1
4  5  1
6  7  1
7  8  1
# Covalent bond type 4
2
9  3  1
10  6  1
# Covalent bond type 5
1
9  10  1
#
# Here we define angle bending bonds
# Total number of angle bending bonds:5
5
# Two atom triplets belong to angle bond type 1:
# OBS! Triplet has unusual order: central atom stands last in the triplet, e.g.
# triplet 1 9 2 means angle 1-2-9, with 1,9 on sides, and 2 on corner.
2
1  9  2
1  10  2
# Two atom triplets belong to angle bond type 2:
2
2  3  9
2  6  10
# angle bond type 3:
2
9  4  3
10  7  6
# angle bond type 4:
2
3  5  4
6  8  7
# angle bond type 3:
```

```
2
3  10  9
6  9  10
```

4.4 .rdf

This file keeps reference distribution functions of three types: radial distribution functions for each non-bonded potential; intramolecular covalent bond distance distribution for each covalent-bond like potential; intramolecular bending angle distribution for each angle-bending potential. RDF file is necessary for the inverse solver mode of *magic*, otherwise it will only perform MC sampling with a look-up table potentials. RDF files are generated by program *rdf-2.0.py* and can be read and visualized by *MagicTools*.

Lines beginning with '#' or '!' are comments.

First line: 4 numbers:

1. number of different atom types over all molecules in the simulation (integer). Different molecules may have atom types of the same kind.
2. number of "bins" for intermolecular RDFs (integer). Must be the same as NA parameter in the main input file
3. Rmin : minimum distance for RDF; usually 0.
4. Rmax = Rcutoff for short-range interactions and RDFs (real)

Atom types: Then the number of lines, equal to the number of atom types follow, with one character parameter per line: name of the corresponding atom type.

Then the RDFs are read according to the following scheme:

1. Intermolecular (non-bonded) part: Each section contains RDF written in 4 columns, with the following meanings:

1. distance, Å
2. corresponding RDF value
3. columns 3-4: atom types involved in this RDF

The range of short distances where RDF has zero values may be omitted, these points will be automatically filled by zeros. The distance column is for human-reading only; the actual distances are computed from Rmin, Rmax values and the number of bins, implying equal size of all the bins. If an RDF between some pair of atomic types is missing, these types will not have a short-range interaction, but only Coulomb one. Atom types in columns 3,4 are those which are defined in the 7-th column of the first part of the .mcm files.

Covalent-like bond distributions: The section begins with keyword `intra`, which is followed by intramolecular bond length distributions.

- Each distribution starts by a line with 4 numbers:

1. the number of different bonds of this type (NST)
2. number of "bins" for bond DF between this atom pair (NPOTS)
3. 3 and 4: - Rmin and Rmax for this bonded DF

- Then follows NST lines, (each per bond of this type), with two numbers showing which atoms are connected by this bond: **NB:** in this and next section, global CG site numbers are used, not the interaction types as in the nonbonded section!
- Then follows NPOTS lines with 2 real numbers: distance and RDF

Bending angle bond distributions: The section begins with keyword `angle`, which is followed by intramolecular bond angle distributions.

- Each distribution starts by a line with 4 numbers:
  1. the number of different angles of this type (NST)
  2. number of "bins" for intramolecular ADF for angle of this type (NPOTS)
  3. 3 and 4. - Min and Max angle values for this intramolecular ADF
- Then follows NST lines, (each per bond of this type), with three numbers showing which atoms are involved in this angle. NB! The triplet has unusual order: central atom stands last in the triple, e.g. triplet 1 3 2, defines angle between 1-2 and 2-3.
- Then follows NPOTS lines with 2 real numbers: angle (degrees) and ADF

The .rdf file must end with the keyword "end"

En example of .rdf file can be found in the tutorial files.

4.5 .pot

File describes the whole set of interaction potentials for the given system in a tabulated format. It is created and used by *magic*, but can be imported from outside, plotted and exported to Gromacs by *MagicTools*.

A .pot file consists of three parts: short-range non-bonded potential, intramolecular bond potential and intramolecular angle bending potential. All potentials are given in kJ/mol, distances are given in A, and angles are given in degrees.

Electrostatic interactions are calculated independently from the charges, specified in .mcm files, and scaled by the value of dielectric constant, which is specified in the magic main input file

Format of .pot file: No comment lines allowed.

First line : Consists of 7 numbers:

1. number of different atom types over all molecules in the simulation (integer)
2. number of bins for potential (integer). Must be the same as NA parameter in magic's main input file and in the .rdf file

3. number of non-bonded potentials (integer; must be equal to the number of different interacting non-bonded pairs over all molecules)
4. number of intramolecular pairwise bond potentials (integer; must be equal to the number of different "bonds" over all molecules)
5. number of intramolecular angle bending potentials (integer; must be equal to the number of different "angles" over all molecules)
6. Rmin (usually 0.)
7. Rmax = Rcutoff for short-range interactions (real)

Intermolecular part: For each pairs of atom type, "NA" lines should be specified, and every line consists of 4 numbers :

1. distance, Å. Only for guidance, since the actual distance is always calculated by division of the interval [Rmin,Rmax] by NA parts
2. value of the potential (only short-range part) in kJ/mol
3. columns 3-4: two integer numbers specifying the CG atom types involved into this interaction.

Note that potential values should be given for all distances starting from Rmin, even in the core region. When a .pot file is created by *magic*, the values of the non-bonded potential in the core region are filled by very large numbers.

Intramolecular part, pair bonds: Consist of subsections, one per each type of bond potentials (their number is specified by the fourth parameter in the first line). For each intramolecular pair bond potential the first line consists of 6 numbers:

1. The number of different bonds of this type (NST)

2. 3. Contain the same numbers as the next line, see below

4. Number of "bins" for this bond potential

5. 6. - Rmin and Rmax for this bond potential

Then follows NST lines (each per bond of this type), with two numbers stating which atoms are connected by this bond, in terms of global CG site number, and NPOTS lines with 2 real numbers: distance and potential.

Intramolecular part, angle bending: Consist of subsections, one per each type of angle potentials (their number is specified by the fifth parameter in the first line of .pot file). For each intramolecular angle potential the first line consists of 6 numbers:

1. The number of different bonds of this type (NST)

2. 3. Put the same as the next line, see below

4. Number of "bins" for the potential (NPOTS)

5. 6. - Minimal and maximal angle for the potential

Then follows NST lines, (each per bond of this type), with three numbers stating which atoms are connected by this bond, and NPOTS lines with 2 real numbers: angle and potential. The central atom stands last in the triple, e.g. triplet 1 3 2, defines angle between 1-2 and 2-3.

All parameters of the .pot file must be consistent with the corresponding parameters in the .rdf file and the order of all distribution functions / corresponding interaction potentials should be the same. The program does only limited control about their consistency.

References

[1] A. Mirzoev, A.P.Lyubartsev *"MagiC: Software Package for Multiscale Modeling"* J. Chem. Theory and Computations, 9(3), 1512-1520 (2013). DOI: 10.1021/ct301019v

[2] A. P. Lyubartsev and A. Laaksonen *Calculation of Effective Interaction Potentials from Radial Distribution Functions: A Reverse Monte Carlo Approach* Phys.Rev.E, 52, 730 (1995).

[3] D. Reith, M. Pütz and F. Müller-Plathe *Deriving effective mesoscale potentials from atomistic simulations* J. Comput. Chem., 24, 1624-1636 (2003)

[4] A. Mirzoev, A.P.Lyubartsev *Effective solvent-mediated potentials of Na+ and Cl- ions in aqueous solution: temperature dependence* Phys. Chem. Chem. Phys., 13, 5722-5727 (2011).

[5] H. Wang, C. Junghans, K. Kremer *Comparative atomistic and coarse-grained study of water: what do we loose by coarse-graining?* Eur. Phys. J. E, 28, 221 – 229 (2009)