

16 July 2013

## **DL\_POLY molecular dynamics simulation package**

### **cgm-rdf – tool for coarse-grain mapping and RDF calculation**

**by Andrey Brukhno**

***University of Bath & Daresbury Laboratory, United Kingdom***

Coarse-graining implies grouping of atoms into coarse-grain (CG) units called “beads”, whereby a vast amount of the “uninteresting” degrees of freedom can be excluded from consideration. This makes simulation of a CG system significantly less resource/time demanding. The utility provides the means to (i) prepare a CG representation of an atomistic system simulated with the use of DL\_POLY, and (ii) to calculate pairwise radial distribution functions (RDF),  $g(r_{ij})$ , based on the obtained CG representation.

The coarse-graining definitions are given in the input file **CG\_MAP.xml** which specifies the CG mapping between the original and coarse-grained systems. The xml format has been chosen for compatibility with VOTCA package – a powerful tool for systematic coarse-graining applications becoming popular nowadays. Thus, the format of **CG\_MAP.xml** is supposed to comply with that described in the manual of VOTCA (section 3.1 Mapping files) to which the user should refer if this brief reference is insufficient. Note, though, that tags related to bonded and angular interactions (`<cg_bonded>`, `<bond>`, `<angle>` etc.) are not supported yet by **cgm-rdf** tool, although their implementation is planned/pending.

### **General workflow of CG mapping with cgm-rdf**

The utility starts by, first, scanning the original force-field (topology) file called **FIELD** and, then, analysing the mapping scheme specified in **CG\_MAP.xml**. If the latter is consistent with the molecule definitions in file **FIELD**, **cgm-rdf** carries out the mapping of configurations found in files **CONFIG**, **REVCON** and, optionally, **HISTORY** (the trajectory file) previously obtained for the full-atom system.

The produced output includes the following: **FIELD\_CG**, **CONFIG\_CG**, **REVCON\_CG** and, optionally, **HISTORY\_CG**. Regarding the produced **FIELD\_CG** file, it must be again noted that the (missing) definitions for bond and angle dependent intra-molecular interactions remain to be introduced manually.

While running, **cgm-rdf** produces some messages to inform the user about its progress and errors encountered, if any. Since these messages are barely formatted and include expanded tags and long mapping strings read from **CG\_MAP.xml**, to avoid obscuring the terminal view, it is recommended to redirect the standard output of **cgm-rdf** into a log-file.

To resume, provided the current working directory contains the output of DL\_POLY for the full-atom system, the CG tool can be invoked as follows:

```
user@host<dl_poly_sim_dir>$ cgm-rdf.x > cgm-rdf.log &
```

Then, if anything goes not as expected, one can check the log file (**cgm-rdf.log**) for clues. Most often the errors will arise due to misspellings in **CG\_MAP.xml** – checking and correcting tags, names of molecules, atoms and maps should normally resolve the issues (see below for the format details).

## Expected format of CG\_MAP.xml

The CG mapping input file for `cgm-rdf` assumes XML format, i.e. using xml-tags (e.g. `<tag></tag>`), where any meaningful value must happen only in between the opening (`<tag>`) and closing (`</tag>`) tags. An example of the correct `CG_map.xml` file in the case of mapping a three-site water molecule (TIP3P) onto one CG bead is given below.

```
<cg_molecule>      <!-- start with this tag section, can have a few -->
                    <!-- section describing a CG molecule type      -->
  <name>Water</name> <!-- have this tag *once* in section          -->
                    <!-- name of the CG molecule after mapping      -->
  <ident>water tip3p</ident> <!-- have this tag *once* in section  -->
                    <!-- name of the original molecule in FIELD      -->
  <topology>          <!-- have this tag subsection *once*        -->
    <cg_beads>        <!-- have this tag subsection *once*        -->
      <cg_bead>       <!-- at least one instance = 1 CG bead        -->
        <name>H2O-com</name> <!-- *once* in subsection              -->
                                <!-- type of the CG bead in created FIELD.CG -->
        <type>H2O</type>   <!-- *once* in subsection              -->
                                <!-- name of the CG bead in created FIELD.CG -->
        <mapping>W</mapping> <!-- *once* in subsection            -->
                                <!-- name of the CG map to be used for the bead -->
        <beads>         <!-- specification of atoms in the CG bead, *once*-->
          1:w_tip3p:OW 1:w_tip3p:HW 1:w_tip3p:HW
          <!-- 1*:OW 1*:HW* -->
        </beads>
      </cg_bead>
    </cg_beads>
  </topology>
  <maps>              <!-- maps container subsection; only *once* -->
    <map>              <!-- map specs; at least one, can have a few -->
                        <!-- at least one bead must refer to the map name-->
      <name>W</name>
      <weights>        <!-- atom weights in the CG bead, one per entry -->
        16. 1.008 1.008
        <!-- 16. 1.008 -->
      </weights>
    </map>
  </maps>
</cg_molecule>
```

Here the tags, their values and user remarks (embraced in `<!-- -->`) are highlighted in blue, black and grey, respectively. The comments briefly describe the tag rules and meaning. Molecule, atom and map names can be any character string complying with the DL\_POLY format (*sic. bead names and types must be of no more than 8 characters*). Blank spaces, digits and most of the special characters are allowed, except for the backslash (`' / '`; not to mess up the tags), column symbol (`' : '` used as a delimiter in the `<beads>` specification), tilde and hash symbols (`' ~ '` and `' # '` used as delimiters by `cgm-rdf` internally).

Clearly, the CG mapping is done molecule-wise, i.e. one cannot unite two or more molecules into one CG entity, nor include atoms from different molecules into the same CG bead. Scenario are, though, possible where the same atom can be included into different CG beads within the same molecule. Note also that molecule type given as value in `<ident>` tag must be found in the original FIELD file, otherwise `cgm-rdf` will terminate with an error.

## Bead names and types within <cg\_bead> tag

Since each CG bead is mapped onto a specific set of atoms, which implies unique atomic indices, the bead names set by <cg\_bead><name> tag must be unique within the molecule specification. Otherwise `cgm-rdf` tool will halt with an error: “repetitive CG bead name”. However, DL\_POLY treats atom names essentially as atom types, and therefore *it is the bead types specified by <cg\_bead><type> tag that are stored in the output files in place of the “atom names”, whereas the bead names from CG\_map.xml take place of the “atom types” in the FIELD.CG file.*

This way it becomes possible not only to have the same “atom name” for physically identical beads (in CONFIG.CG, REVCON.CG and HISTORY.CG files), but also to opt for unique “atom names”, if necessary, by specifying unique bead types in CG\_map.xml. This feature, although being a *deviation from the original VOTCA specification*, adds some flexibility when using `cgm-rdf` tool – consider calculating RDFs for either all pairs of physically identical beads or only certain bead pairs (say, due to their “importance”).

## Atom lists in <beads> tag

According to the VOTCA xml specification, each atom included in a CG bead is identified by a triple ID as follows: `residue_index:residue_name:atom_name`, all the three IDs being found in the original topology file (this is the case for Gromacs MD simulation package). In the case of DL\_POLY residue name is not used in FIELD files at all, while *residue index is equivalent with the index of a charge/neutral group within a molecule*. Therefore, for the lack of a better option, **`cgm-rdf` ignores the value of `residue_name` and only uses `residue_index` and `atom_name`** from the VOTCA's triplet for atom identification. Note, though, that *while residue name is ignored, it has to be present in some way, e.g. the wild-card character '\*' or molecule name (or type) can be used for it.*

**The `residue_index` should match the index of the charge/neutral group** that the picked up atom belongs to. However, if the residue/group index cannot be taken into account, e.g. when the entire molecule is the only group, or group numbers are absent in FIELD, *the wild-card '\*' can be used to indicate that residue index must be ignored.*

**Various ways to identify atoms within the <beads> tag.**

**1) Most straightforward is to copy the group index and name of each atom in the same order as they appear in the FIELD file**, see the example above. *In this case the atom order within the molecule is presumed to be the same as in the <beads> tag.*

**2) It is also possible to specify the index of atom after its name**, with a hash '#' symbol preceding the index (`res_index:res_name:atom_name#atom_index`), e.g. `<beads> 1:*:OW#1 1:*:HW#2 1:*:HW#3 </beads>`. Alternatively, one can skip atom names altogether and opt for “`:#atom_index`” representation only. *When atom indices are present, atoms can enter the <beads> tag in arbitrary order.*

**NOTE:** if `residue_index` is also provided, `#atom_index` represents the atom *position within the group*, defined by the order of appearance of atoms within the group in the FIELD file. In contrast, in the case of undefined `residue_index` (\*), `#atom_index` should be set equal to the atom *positional index within the molecule*, defined by the order of appearance of atoms in the FIELD file.

**3) The most compact way of specifying atom lists is by using the wild-card '\*' instead of atom indices** (but *not for parts of indices or names!*). For instance, in order to include all atoms with the same name belonging to the same group, one can use

“:atom\_name#\*” or even simpler form “:atom\_name\*”, see the commented-out list of the <beads> tag in the example for water above. Evidently, by skipping the atom names and using “: #\*” in the place of atom\_name, one can define one CG bead for the entire group (residue), or even molecule (*sic.* “\*: \*: \*”)!

**NOTE:** the allowance of using wild-cards and hash-preceded atom indices is also an *extension to the VOTCA specification*, which does not, though, restrict the use of the VOTCA tools after the CG mapping has been done for DL\_POLY input files with `cgm-rdf`.

## Mapping specification in <map><name> and <map><weights> tags

The weights allow to weigh each atom contribution to a CG bead as necessary. In the simplest case of centring the bead on the center of mass (COM) of the group of included atoms, weights must be set equal to the atom masses. Common sense suggests that each map name (defined by <map><name>) should be present within the <cg\_beads> subsection at least once (named by <cg\_bead><mapping>). Note that `cgm-rdf` will halt if it's not the case! Obviously, the sequence and number of entries in <map><weights> must match those specified in <cg\_bead><beads> for the same <cg\_bead><mapping> name, i.e. **weights are assigned atom-by-atom or entry-by-entry** (if '\*' characters are used). *To set all weights equal to masses for all atoms in a bead, put '\*' in <weights>.*

Being the counterpart of atom masses, weights are accounted for in the same fashion, i.e. all atom coordinates and velocity components (if present in HISTORY file) are re-weighted by normalised weights, so that the sum of all atom weights within a CG bead equals 1. Note that this way the total momentum of the atomic group included in a CG bead is preserved, whereas the corresponding kinetic energy is not. Forces are treated as conservative, i.e. dependent on atom-atom separations only, and hence, the total force on the CG bead center is set to the sum of all the atomic forces (if present in HISTORY file).

## Using `cgm-rdf` tool for RDF calculation

The utility allows also to calculate radial distribution functions,  $g(r_{ij})$ , for pairs of CG sites, based on a set of full-atom configurations (frames) stored in the trajectory file HISTORY. As will be detailed shortly, in a single run of `cgm-rdf` one can obtain *either a single RDF* for a particular pair of beads identified in file `CG_map.xml` by their “bead names”, *or several RDFs* for a set of bead pairs, as is prescribed by the user in an additional input file named `inp.cgm-rdf`. **NOTE:** *if the latter file is not found in the working directory, the tool can still be used for CG mapping, without RDF calculations.*

## Format of the RDF input file for `cgm-rdf`

Before using `cgm-rdf` for RDF calculus one needs to create the input file for it, called **inp.cgm-rdf**, containing RDF-specific input. This input file should have the following “fixed” format and contain all the parameters shown in the example below. Note that for running `cgm-rdf` without reading errors *it is essential to adhere to the specified line sequence in the input file and keeping the number and the meaning of the parameters on each line exactly as exemplified.*

Each parameter-containing line, or “record”, is preceded by a descriptive remark line with a short reference for the parameters read in next. To make the input file easier readable, the format assumes exactly one(!) empty line to separate a previously read record and the next remark line. All remark lines and single empty lines are simply skipped by `cgm-rdf`. **Thus, the format is based on line triplets containing subsequently: descriptor, record and separator lines. Provided the parameter values in a record are put in the**

**correct order, the figures do not assume any particular formatting**, i.e. they are read in a “free style” format: `read(istd,*)val1,val2 etc..`

Trajectory file name (HISTORY or HISTORY\_CG)

**"HISTORY\_CG"**

bead1, bead2

**H2O ALL**

`r_cut (<= Lmin/2), delr (Angstrom)`

**10. 0.05**

`Nbeads , Conf0, ConfN, Nskip`

**-1 1 1000 10**

The first record should contain a string specifying the name of the trajectory file, either HISTORY or HISTORY\_CG. **NOTE:** specifying HISTORY here implies *skipping the creation of HISTORY\_CG file!* The second record provides two bead names (must be present in CG\_map.xml) as `character*(8)` words. *Single or double quotes can be used to embrace strings containing blank spaces, otherwise the quotes are unnecessary.*

The third record defines the cutoff distance (Å), `r_cut`, and the bin size, `delr`, to be used for the *r*-grid, from which the number of bins is calculated. Most often `delr` = 0.05 Å would be suitable, as producing sufficiently detailed RDFs, and still keeping the statistical noise level under control.

**NOTE:** the cutoff value does not need to be related in any way to the actual cutoff(s) used in the DL\_POLY simulation (*cf.* CONTROL file). But, of course, *the cutoff for RDF cannot be greater than half the minimum cell dimension in any frame found in the HISTORY file*, i.e. one has to be careful with the cutoff choice in the case of *NpT* trajectories!

The last (forth) record sets the (integer) numbers for: `Nbeads` – the number of beads per frame; `Conf0` and `ConfN` – the first and the last frames to account for, and `Nskip` – the skipping step between the frames included in the RDF calculation. Any of the numbers can be set to -1, in which case `cgm-rdf` will assume one of the following: for `Nbeads` – reading it from the first frame in the HISTORY file; for `Conf0` and `ConfN` – starting with the first frame and/or finishing with the last frame found in HISTORY; and `Nskip=-1` will be translated as `Nskip=1`.

### Alternative ways of RDF(s) calculation

If two actual bead names (present in CG\_map.xml) are given, then a single RDF will be calculated and stored in file called **RDFDAT\_bead1\_bead2** (two-column ASCII file), where “bead1” and “bead2” stand for the bead names used in the input. It is, however, possible to replace one or both bead names in the input with a directive word: “**ANY**” or “**ALL**”. Thereby `cgm-rdf` will be directed to calculate RDF(s) not for a particular bead pair but as follows.

**1)** Self-evidently, “**ANY**” invokes the substitution of the actual bead name by “any bead name”, implying that all possible pairs (consistent with the reference to the other bead) will be included into a single RDF file: **RDFDAT\_bead1\_any** (or **RDFDAT\_any\_any**).

**2)** In contrast, directive “**ALL**” tells `cgm-rdf` to run through “all bead names” one-by-one and, hence, calculate the (multiple) separate RDFs stored as **RDFDAT\_bead1\_bead2**.