RMCSANS user guide

Orsolya Gereben

24.06.2010

I. The Reverse Monte Carlo algorithm

The Reverse Monte Carlo (RMC) computer simulation technique is capable of building 3-dimensional structural models in agreement with the experimental (mainly diffraction) data. The description of the algorithm is given elsewhere[1],[2],[3]. Several computer version of RMC exist, one of the newest implementation (RMC++)[4] was written in C++ and was parallelised and improved with new capabilities (RMC++_new and RMC++_multi)[5]. These later software were used as the staring point of the RMCSANS program. The theoretical background is discussed elsewhere[6], here the usage of the program will be described.

II. The RMCSANS program

The RMCSANS program was written in C++, using only the standard statements as far as it was possible. There were however few cases, where operation system-specific statements have to be used, these are located in *altern.h* and *altern.cpp* files. The program was tested on Windows and Linux platform, if other platform is used, some alteration might be necessary!

The RMCSANS program can be used in normal (consecutive) or multi-threaded environment depending on the compilation, which will be discussed later.

A. The structure of the program

The program consists of several header and source files listed below.

Header files

altern.h	header file including the necessary built-in headers, containing the options regulating the building of the code, declaration of the functions differing according to the options chosen
Coordinates.h	declaration of the SimpleCfg class which stores the coordinates, and the class' destructor
Grid.h	declaration of the Grid class related to the gridding of the simulation box, and the class' destructor
Move.h	declaration of the Move class responsible for making the movement
PrimaryClasses.h	declaration of basic classes: ExptsData, CoordNumbConst, AvCoordConst, RunParams and the classes' destructors
SecondaryClasses.h	declaration of the classes involved in the calculation: HistoSet, DataMat, PPCFSet, CalcPart, CalcData, ChiSquared, History and the classes' destructors
Threads.h	declaration of the Threads class for multi-threading and the class' destructor
units.h	declaration of the used constants
utilities.h	declaration of the auxiliary functions

Source files

altern.cpp	definition of the function differing according to the chosen options
AvCoordConst.cpp	definition of the AvCoordConst class describing the average coordination number constraint

CalcData.cpp	definition of the CalcData class holding and calculating the calculated data comparable with the experimental
CalcPart.cpp	definition of the CalcPart class holding and calculating the partial *S(Q)* and *SPA(Q)*
ChiSquared.cpp	definition of the ChiSquared class holding and calculating difference between the calculated and the experimental data
CoordNumbConst.cpp	definition of the CoordNumbConst class describing the coordination number constraint
DataMat.cpp	definition of the DataMat class containing the conversion table for the normalization of the *g(r)*, and the Fourier transformation matrices
ExptsData.cpp	definition of the ExptsData class containing the experimental data
Grid.cpp	definition of the Grid class containing the gridding of the simulation box
History.cpp	definition of the History class gathering and outputting information about the run
HistoSet.cpp	definition of the HistoSet class containing and calculating the histogram
MakeMove.cpp	containing the makemove function creating the movement of the spheres
Move.cpp	definition of the Move class, regulating the movements of the spheres, functions connected with the "too close" spheres, other functions of the Move class
PPCFSet.cpp	definition of the PPCFSet class, containing the partial pair correlation functions (ppcf)
RMCSANS.cpp	the main program for RMCSANS regulating the run, creating the instances of the classes, containing the simulation loop
RunParams.cpp	definition of the RunParams class describing the parameters of the simulation, calculating some necessary parameters
SimpleCfg.cpp	definition of the SimpleCfg class containing the coordinates of the spheres
Threads.cpp	definition of the Threads class containing the variables and function necessary for multi-threading, but a basic Thread class without multi-threading-specific variables is used even in case of consecutive compilation
utilities.cpp	definition of the auxiliary functions

B. Files used by RMCSANS

The *_X* in some of the file names refers to the time frame index. In case of the **.hst*, **.datmat*, the same file is used during a simulation for all the time frames, indexing is only added, if the simulation was not started from the initial data.

****.dat***	INPUT: the data file containing the parameters of the run.
****.cfg, *.cfg_X***	The text type configuration file containing the coordinates.
****.bcf, *.bcf_X***	The binary type configuration file containing the coordinates.
anyname_X	INPUT: The files containing the experimental data, name can be free choice.
**.ind*	INPUT: only if AtomEye option 3 is used, containing the indices of the

	selected atoms, one for a line, (indexing starting with 1).
**.fdat*	INPUT: for frame specific coordination or average coordination number data.
sfactorcube	INPUT: Only needed, if $x_{max}>\sqrt{2}$.
**.hgm, *.hgm_X*	Partial histograms.
**.grid, *.grid_X*	Gridding of the simulation box.
**.cnc, *.cnc_X*	The coordination number constraint.
**.acn, *.acn_X*	The average coordination number constraint.
**.tcs, , *.tcs_X*	The indices of the tooclose spheres in case of the moveout option.
**.sind*	OUTPUT: the indices of the selected spheres in case AtomEye option 1
**.fit_X*	OUTPUT: the total calculated *I(Q.*
**.expt, *.expt_X*	OUTPUT: The experimental data is saved to it for checking.
**.ppcf, *ppcf_X*	OUTPUT: The initial radial distribution function.
**.psq, *.psq_X*	OUTPUT: The calculated *S(Q)* partials.
**.spa, (*.spa_X)*	OUTPUT: The calculated *SPA(Q)* partials.
**.hst, (*.hst_X)*	OUTPUT: The history file containing information about the run.
****.chi, (*.chi_X)***	OUTPUT: The initial χ^2 for each data set.
**.datmat, (*.datmat_X)*	OUTPUT: The conversion tables of the DataMat object.
**.calcdat, (*.calcdat_X)*	OUTPUT: The initial calculated *I(Q)* data.

- The input files written in **bold** are mandatory, but it is enough, if either the text or the binary coordinates files is given, text type has precedence over binary, if both is given and they are not compatible.
- The software can be started without giving any *I(Q)* data, in this case a hard sphere simulation is performed, constraints can be present.
- The files given in (brackets) are produced only at the beginning of a whole simulation process, if the simulation does not start from the beginning, but it is a continuation of the simulation starting from frame *X*. In this case the starting frame index is appended to these file names. In case of the history file, the same history file created at the beginning of a simulation is used during all the time frames during a continuous simulation.

C. Visualization using AtomEye

Changes of particle coordinates during the simulation can be visualized. The whole simulation box or a chosen subset of particles, depending on the AtomEye options specified in the parameter file of RMCSANS, can be displayed during or after the simulation. For this purpose the Atomistic Configuration viewer[7,8], AtomEye 3.0, was modified. AtomEye is a linux-based free software capable of making 3D colour images based on a specific, AtomEye-format configuration file containing, among others, the coordinates and atomic symbols of atoms. There are lots of possibilities to govern the appearance of the pictures. As RMCSANS does not use atoms but larger particles, the program had to be modified to be able to handle them. Several other modifications were performed; the most important are making AtomEye to check if the image belonging to the next simulation stage is available and automatically display it; it is now also possible to display transparent particles. The new, modified version is called AtomEye3.1. RMCSANS and AtomEye are two completely different program working separately.

D. The structure of the *.dat file

The example will show the **.dat* parameter file for a two-component system, the first component having 3 and the second 2 shells. The first column will contain a serial number to help referencing the lines later, but it is not part of the file. As there are parameters, which has to be given in separate line (see coloured lines) for each instance of a parameter specified before them in the **.dat* file, the index of a line can be subject to the structure of the file.

1	test_run	
2	8	! total number of threads
3	10	! number of time frames
4	1.2148e-6	! number density
5	0.375 0.625	! molar fraction
6	3 2	! number of shells/type
7	13 10	! mean core sphere radius for each type
8	15 16 17	! mean shell radius(es) sphere radius for a given type
9	12 14	! mean shell radius(es) sphere radius for a given type
10	0 0	! maximum change in sphere radius
11	-1 -1	! z parameter (controlling the width of the size distribution), -1 for monodispersity
12	0 0	! maximum change in z parameter (integer)
13	1.0 1.2	! maximum moves
14	1.0	! r spacing (Å)
15	0	! initial bin shift
16	1.0	! xmax used in the run (in reduced units)
17	0	! whether to use moveout option (0: false, 1: true)
18	1	! whether to load the histogram (and coordination numbers, if there is a constraint), if the files are available (0: false, 1: true)
19	14	! maximum number of atoms in a gridcell
20	0.3 1	! fraction of swaps, ntypes*(ntypes-1)/2 0 (not allowed) or 1 (allowed) for the possible mixed partial (in order 1-2, 1-3,... 2-3...)
21	1 0.5 s1 s4 501.0 502.0 300 400	! whether to generate AtomEye3.1 input file, weight/type in amu, number of used spheres/type (needed for option 1, optional for option 2)
22	100	! step for printing (number of moves)
23	20000 30 0.5	! time limit, step for saving in minutes , multiplication factor for the time limit of the first time frame
24	100	!size of the history buffer (0->no history record, 200 is good)
25	50	!number of savings between each history buffering (>=0)
26	2	! no. of neutron data series
27	s2_non_eq.iq	
28	10 101	! range of Q points (start with 1)
29	0.000	! constant to subtract
30	0.2 -0.1	! core relative scattering length density/type ($Å^{-2}$)
31	0.4 0.6 0.8	! shell relative scattering length densities for the given type
32	0.3 -0.3	! shell relative scattering length densities for the given type
33	3e-12	! standard deviation
34	0	! whether to vary amplitudes (0: false, 1: true)
35	0	! whether to vary constant
36	0	! whether to vary linear
37	0	! whether to vary quadratic
38	s2.iq	
39	1 139	! range of Q points (start with 1)
40	0.000	! constant to subtract

41	-0.2 -0.5	! core relative scattering length density/type ($Å^{-2}$)
42	0.0 0.2 0.4	! shell relative scattering length densities for the given type
43	-0.1 -0.7	! shell relative scattering length densities for the given type
44	2e-12	! standard deviation
45	0	! whether to vary amplitudes (0: false, 1: true)
46	0	! whether to vary constant
47	0	! whether to vary linear
48	0	! whether to vary quadratic
49	1 1 5	! number of coordination constraints, number of neighbour type/constraint, number of sub-constraint/constraint
50	1 1 34.0 40.0 8 9 10 11 12 0.4 0.5 0.6 0.8 1.0 0.0002 0.0003 0.0004 0.0005 0.0005	! central type, neighbour type(s), rmin[first_neightype]… rmin[last_neightype], rmax[first_neightype]… rmax[last_neightype], desired coordination number[first_subconst] ...desired coordination number[last_subconst], fraction[first_subconst] ...fraction[last_subconst], sigma[first_subconst] ...sigma[last_subconst]
51	2	! number of average coordination constraints
52	1 1 34.0 40.0 12 0.004	! type of the central atom, type of neighbour, min dist, max dist, desired average coord numb, sigma
53	1 2 34.0 40.0 12 0.004	! type of the central atom, type of neighbour, min dist, max dist, desired average coord numb, sigma

Detailed description of some of the parameters, the parameters will be referenced by the serial number in the first column.

2.) Number of threads for parallel execution.

3.) Number of consecutive time frames to fit, which is the number of experimental data snap shots separated by a time interval.

6.) Number of shells for the onion sphere model (see Ref 6). It can be zero, in this case no lines like 8 and 9 can be in the file.

7.) The radius of the core for each type of particle.

8-9.) As many line like this, as the number of shells given in line 6.

10-12.) Related to the poly-dispersity, which is not implemented in the software yet, leave the values, as they are.

14.) The histogram bin size in Ångstrom.

15.) Number of imaginary bins to leave out from the calculation at the small distance end (before the first used histogram bin). Can be a fraction!

16.) Largest distance between the particles to include in the histogram calculation in reduced unit (maximum is $\sqrt{3}$).

17.) If there are particles closer to each other than the cut-off distance, using the moveout option (1) means to move more frequently the "too close" particles to increase their distance above the cut-off. (0) means not to use moveout option. There can be more, than one "too close" particles among the n_{moved} moved particles!

18.) If the histogram file **.hgm*, {the **.cnc* and **.acn* files if there is/are (average) coordination number constraint(s)} are available, in case of (1) loading of them will be attempted, and if they are compatible with the given constraints and parameters, then initial histogram calculation will not be done. In case of option 0 initial histogram calculation will be performed, and if they were existing files, they will be overwritten. The values of the initial calculation will be saved in the **start.hgm*, **start.cnc* and **start.acn* files as well to be preserved. It has to be emphasized, that in

case of loading the histogram and coordination constraint files, the validity of the actual values cannot be checked, so care must be taken to use files corresponding to the **.cfg* and/or **.bcf* files!

19.) Gridding means that the simulation box is divided into a given number of sub-cells in each direction. If we know about each particle, in which grid cell it is located, and we want to calculate certain properties for only those particles that are not farther from the chosen particle than a given distance, then it is enough to calculate only for those particles, which are located in the same, and a given number of neighbouring grid cells. As checking, whether the move can be acceptable based on the satisfaction of the cutoff distances falls in this category, calculation can be quicker, if the grid-based cutoff check is performed before entering the lengthy histogram change calculation. The number of particles in the grid cell is regulated by this parameter. Based on this, the average number density of the sample and an additional safety parameter, SAFE_ADD located in the *units.h*, the program calculates the number of grid cells in each direction t. As due to inhomogenity in the sample these numbers can vary, the program always checks, whether the maximum is not exceeded, before attempting to write into the arrays. If the maximum was exceeded, the program gives a warning message, resizes the necessary arrays automatically, and continues execution. Gridding can only increase speed, if the number of particles in a grid cell is chosen adequately! It is preferable to set the desired maximum number of particles in a grid cell in the **.dat* file to a relatively low value (5-10) depending of course on the system size to ensure at least 5 or preferably more grid cell in each direction. The actual number of grid cells is calculated by the program and printed on screen during the initialisation period, or can be found in the **.grid* file. It has to be kept in mind, that even if the largest cutoff distance is smaller than the length of one grid cell, not just the cell containing the central particle, but all its closest neighbour cells are checked, as the central particle can be close to the edge. This way normally at least 27 cells are checked. Speed increase due to gridding can only be expected, if the number of grid cells to be checked is smaller, than the total number of grid cells!

20.) Only has meaning for multi-component systems, where particles from different types can be swapped with each other to help the mixing of the simulation box. First swap fraction has to be given, a real number (between 0-1), which regulates the fraction of swaps related to the moves. In case of (1) only swaps, (0) no swap at all. After this in case of swaps as many integer as the number of mixed partials (their number is ntypes*(ntypes-1)/2) specify, which mixed partials can participate in the swap, (0) not involved, (1) involved. The order of the partials is the same, as usual, without the 'clean' partials (1-2, 1-3,…2-3,…).

21.) The parameters concerning AtomEye are given in one line following each other. These are the AtomEye_option, AtomEye_timesave, particle_name/type, weight/type, [pnused_spheres/type for AtomEye_option=1,2,3] or [select_axis, coord_limit1, coord_limit2 for AtomEye_option=4]. Description of the options:

- AtomEye_option controls, whether AtomEye type cfg file will be created, and if it will be, how the particles are chosen.
  - 0: No AtomEye cfg file is created, the rest of the line is skipped.
  - 1: The indices of the N_p used particles will be spread among the particles of this type with as equal step as possible. The particles can be found everywhere in the simulation box.(default). NUMBER of used spheres/type is needed after weight/type!
  - 2: The first N_p particle will be used from a type. (Most probably particles close to each other are chosen this way!) NUMBER of used spheres/type is needed after weight/type!
  - 3: Particles of given indices specified in the index file will be selected. (Index can range from 1 → ntotal). NUMBER of used spheres/type is needed after weight/type!
  - 4: Only particles between two parallel planes will be saved into the file. The planes have to be parallel to one of the sides of the simulation box! In case of option 4 after the weight/type has to be specified by an integer, which axis is perpendicular to the plane (x:0, y:1, z:2), followed by two real numbers (between -1 and +1) to specify the reduced coordinates according to the selected axis, between which the atoms will be outputed. Example for choosing the atoms, which reduced y-coordinates are between -0.7 → -0.5, after the weight/types: 1 -0.7 -0.5

- AtomEye configuration saving time as the fraction of RMC saving time (0.5 means a configuration is saved between RMC savings, and an other at RMC saving).
- particle_name has to be the standard chemical symbol for atoms, or s1, s2, s3, s4, s5 for compound particles.
- weight for each type is needed by the AtomEye extended configuration file, but the value is not relevant, if only the visualization of the configuration is wanted.
- number of used spheres for each type is needed for AtomEye_option=1,2,3. If number of used spheres is equal or greater total number of spheres for this type, than all the spheres of this type will be selected!
- select_axis only for AtomEye_option=4, to specify, which axis is perpendicular to the cutting planes, 0 for *x*, 1 for *y,* 2 for *z*.
- coord_limit1, coord_limit2 only for AtomEye_option=4, values between -1 and +1 to specify the reduced coordinates according to the selected axis, between which the atoms will be outputted.

23.) Time limit (minutes) is the total running time for one time frame. Saving time (minutes) regulates the interval the data is saved to disc. The duration of the first time frame can be different from the others, the multiplication factor has to be given here, it is a real number. Can be useful, is the simulation is stopped at a time frame and continued from it, but there is no need to complete a whole time limit interval.

24.) Size of the history buffer, each containing the χ^2 for a given phase of the simulation. When the buffer is full, the data is written to disc. Large value means less frequent disc writing, so greater program speed. (0->no history record, 200 is good).

25.) Regulates the interval, the χ^2 is saved to the history buffer. Integer number, denotes the number of savings between each history buffering.

26.) This is the number of different experimental SANS data sets. Each set has to have the same number of time frames (number of consecutive experiments) given in line 3. The following two green blocks in this example (27-37,38-48) specifies the parameters for a set.

27.) Name of the experimental data file (*filename*). Each time frame has to be in a separate file, named *filename_X*, where X is the time frame index.

28.) Range of the data points used from the files (index begins with 1).

29.) Constant to be subtracted from the experimental data after it was read, shifting the data along the *y*-axis. The *S(Q)*, and consequently the *I(Q)* calculated by RMC tends to zero at larger Q, so should the experimental data!

33.) Standard deviation. The χ^2 is divided by its square, it is used for scaling the contribution of the different data series and constraint to each other. The smaller the value, the larger the contribution of the given data set will have in the total χ^2.

34-37.) Whether to use renormalization of the data sets, see RMC.

49-53.) See the details later, where the frame-specific constraints are discussed.

1. Frame-specific average and normal coordination number constraints

Coordination number constraint

The coordination number constraint try to enforce the system, that a specified fraction of the central particle of the given type should have a specified number of neighbours of a desired type between r_{min} and r_{max}.

To save memory and increase speed the coordination number constraint was generalized, so not only one, but also a specified number of neighbour types can exist for a constraint. For example in case of 3 atom types it is possible to define a constraint that the central atom of type1 should have 3 atoms belonging to type2 and type3 between $r_{min}[type2] \rightarrow r_{max}[type2]$ and $r_{min}[type3] \rightarrow r_{max}[type3]$ respectively, which means, that there are several possibilities satisfying this constraint (type2-type2-type2, type2-type2-type3, type2-type3-type3 and type3-type3-type3).

The concept of sub-constraints was also introduced. More than one sub-constraint means, that more than one desired coordination number (each with its own desired fraction and sigma) can be specified for

a constraint to make the calculation quicker and use less memory and disk space, if constraints should only differ in their desired coordination number. In the original concept each constraint had only one sub-constraint. The introduction of sub-constraint will not result in additional functionality.

The **.dat* file should contain the constraint the following way, for example for 3 constraint, the first having 1 neighbour type and 1 sub-constraint, the second 1 neighbour type and 2 sub-constraints and the third 2 neighbour type and 1 sub-constraint:

```
3 1 1 2 1 2 1  ! no of coordination constraints, number of neighbour types for
        each constraint, number of sub-constraint for each constraint
1 2    2.2  3.05 2  1.0 0.00025  !     central     type,     neighbour     type(s),
        r_min[first_neightype]…      r_min[last_neightype],      r_max[first_neightype]…
        r_max[last_neightype],                desired                coordination
        number[first_subconst]...desired   coordination   number[last_subconst],
        fraction[first_subconst]...fraction[last_subconst],
        sigma[first_subconst]...sigma[last_subconst]
1 3   2.1   3.1 3 4   0.4 0.6 0.00025 0.00015! central type, neighbour type(s),
        r_min[first_neightype]…      r_min[last_neightype],      r_max[first_neightype]…
        r_max[last_neightype],                desired                coordination
        number[first_subconst]...desired   coordination   number[last_subconst],
        fraction[first_subconst]...fraction[last_subconst],
        sigma[first_subconst]...sigma[last_subconst]
1 2  3 2.2  2.1  3.05  3.1 3  1.0 0.00001    ! central type, neighbour type(s),
        r_min[first_neightype]…      r_min[last_neightype],      r_max[first_neightype]…
        r_max[last_neightype],                desired                coordination
        number[first_subconst]...desired   coordination   number[last_subconst],
        fraction[first_subconst]...fraction[last_subconst],
        sigma[first_subconst]...sigma[last_subconst]
```

If coordination number constraint is applied, the number of types cannot exceed 7 if long int type is 4 byte or 10 if long int type is 8 bytes (this depend on the computer applied and the compiler) as the 8*size of internal variable CoordNumbConst::cctype (defined as long int) cannot exceed the number of partials! If more, than 7 types needed, and long int is only 4 byte, then change the type of this variable to 64bit integer, which is unfortunately called by different names in different C++ environment, so it is not used presently in the program.

If we have several time frames in a data set, then it can be necessary to define different desired *fraction* and/or *sigma* for each time frame. The indicator for frame specific data is different for the *fraction* and the σ, as for *fraction* zero can be a valid value. So for *fraction* < 0 means, that frame specific data is given, for σ ~0 (absolute value smaller than LOAD_TOL specified in the *units.h*) indicates, that frame specific data is given. If σ is negative value either in the **.dat* or in **.fdat* file, it means that not the actual value of σ is given, but the percent of the initial χ^2 of the given constraint to the initial χ^2 of the first *I(Q)* data set. This means, that for example σ= -0.8 in the **.dat* file means, that the initial χ^2 at the beginning of each time frame of the given coordination constraint has to be 80% of the initial χ^2 of the first *I(Q)* constraint of the given time frame. σ=-0.8 for a given time frame in the **.fdat* file means, that the initial χ^2 for this time frame of the given coordination constraint has to be 80% of the initial χ^2 of the first *I(Q)* constraint of the given time frame. Specifying the actual σ value or the starting percentage can be used mixed even for one coordination constraint, you can specify actual value for a given frame and percentage for the other frame, also this might not make too much sense.

Each sub-constraint of a constraint is handled separately. The **.fdat* file has to contain the frame related data for the coordination constraint(s) in the following format described below beginning in the first line of the file. For a sub-constraint of a constraint which has any frame-specific data there has to be as many lines in the **.fdat* file, as the total number of time frames (not just the used ones). Each line has to have the format containing the data for the given frame:

```
constraint_index sub-constarint_index    fraction  σ
```

Even if there is not frame specific data for the *fraction*, but there is for σ, some value for the *fraction* has to be given, it will not be used by the program, but needed for reading the file. This way the format of

the file can be the same, if you change your mind after using both frame specific *fraction* and σ only to use frame specific σ.

If there are more than one constraints with frame specific data, then they have to follow each other in the same order as in the **.dat* file. For those coordination constraints, where there is not any frame specific data, the **.fdat* file will not contain any line for them.

Average coordination number constraint

The concept is similar to the coordination constraint, but here not the actual number of neighbours/particle is constrained, but only the average coordination number. The indicator for frame specific data is different for the required coordination number and the σ, as for acn_{req} zero can be a valid value. So for $acn_{req} < 0$, and/or $\sigma \sim 0$ (absolute value smaller than LOAD_TOL) indicates, that frame specific data is given in the **.fdat* file.

For σ negative value in either in the **.dat* or in **.fdat* file means, that not the actual value of σ is given, but the required percent of the initial χ^2 of the given constraint to the initial χ^2 of the first *I(Q)* data set. This means, that for example σ -0.8 in the **.dat* file means, that the initial χ^2 at the beginning of each time frame of the given average coordination constraint has to be 80% of the initial χ^2 of the first *I(Q)* constraint of the given time frame. σ -0.8 for a given time frame in the **.fdat* file means, that the initial χ^2 for this time frame of the given average coordination constraint has to be 80% of the initial χ^2 of the first *I(Q)* constraint of the given time frame. Specifying the actual σ value or the starting percentage can be used mixed even for one average coordination constraint, you can specify actual value for a given frame and percentage for the other frame, also this might not make too much sense.

The **.fdat* file has to contain the frame related data for the average coordination constraint(s) in the following format described below, after the last line of the frame specific information for the coordination constraint, if there is any. For a constraint which has any frame specific data there has to be as many lines in the **.fdat* file, as the total number of time frames (not just the used ones). Each line has to have the format containing the data for the given frame:

`constraint_index` acn_{req} σ

Even if there is not frame specific data for the acn_{req}, but there is for σ, some value for the acn_{req} has to be given, it will not be used by the program, but needed for reading the file. This way the format of the file can be the same, if you change your mind after using both frame specific acn_{req} and σ only to use frame specific σ.

If there are more than one constraints with frame specific data, then they have to follow each other in the same order as in the **.dat* file. For those average coordination constraints, where there is not any frame specific data, the **.fdat* file will not contain any line for them.

*E. The structure of the *.cfg file*

The format of the text-type coordinate file is identical to the RMC version 3 format, except few places in the text, so RMCSANS can read the RMC file, and vice versa.

First a header can be found with general information. The coordinates of the spheres are arranged according to types, first are all the coordinates of the first type, then the second and so on. It has to be noted, that regardless the three separate box vectors, only cubic simulation box can be handled, and only the first box vector is read.

```
(RMCSANS version 1 format configuration file) !file created by SimpleCfg::save !
test


       1000       962       376 moves generated, tried, accepted
          0                    configurations saved

      32000 spheres of all types
```

```
          2 types of spheres
          1 (to be compatible with RMC)
          0 (to be compatible with RMC)

          F (box is cubic)
            Defining vectors are:
            690.534004   0.000000   0.000000
              0.000000 690.534004   0.000000
              0.000000   0.000000 690.534004

      12000 spheres of type  1
          1 sphere sites
              0.000000   0.000000   0.000000

      20000 spheres of type  2
          1 sphere sites
              0.000000   0.000000   0.000000

 -0.987633336403535   0.925880785212280  0.909503049321865
 -0.977325226155868  -0.953697637327416 -0.990475192582183
  0.887719053851144   0.997024791676164 -0.974844995102832
 -0.883800549659262  -0.936855638162484 -0.869230768516112
 -0.926418036537766   0.939895658760871 -0.921847223726014
 -0.867742379356644  -0.880791969987750  0.989364262371605
  0.985969129847551  -0.852903753907912 -0.902887487782913
 -0.822462269892600  -0.964524782323185 -0.894389827275485
 -0.766888765973420  -0.868694927876430  0.932139201516232
                              •
                              •
                              •
 -0.709210176450055  -0.991277379975797  0.994665938293646
 -0.819921420524851  -0.922772485395741 -0.989073934225886
 -0.790493404801405  -0.970241507519512  0.990513018228407
```

F. The structure of the experimental data file

First the title of the data series can be given, then an empty line followed by the Q and $I(Q)$ data. $I(Q)$ is used instead of the microscopic differential (scattering) cross section, which is the experimental data for RMCSANS, see [6] for details.

```
290 !hs_c2_cc12_2

0.001000000     1.726670E-08
0.001050000     1.717757E-08
0.001102500     1.707971E-08
0.001157625     1.697231E-08
0.001215506     1.685451E-08
0.001276282     1.672537E-08
0.001340096     1.658387E-08
0.001407100     1.642894E-08
              •
              •
              •

0.445000000     2.221700E-12
0.455000000     7.536251E-15
0.465000000     2.333310E-12
0.475000000     7.845965E-12
0.485000000     1.539494E-11
0.495000000     2.366464E-11
```

III. Usage of the RMCSANS program

A. Compilation of the program

As the programs was developed with the possibility to be run on different platforms, due to the differences of the operating systems and the available compilers some code changes are necessary before compilation. The program was tested both on PC having Windows operation system using Microsoft Visual C++ compiler, and on GNU/LINUX platform.

1. Pre-processor directives for code building

Pre-processor directives regulate the conditional building of the code. The description of the options regulating the different building of the code can be found in the header file *altern.h* having the simple form of #define PARAMETER. The options can be turned on in LINUX environment by passing the appropriate command line argument to make. In WINDOWS environment, if we want to choose the given option, then the #define PARAMETER belonging to it has to be in the code, if not then it has to be commented out from the code preceding it with //. Here the available option will be given in their order appearing in the file.

First option: choosing the platform

#define _MICROSOFT_VC//If switched on, compiling using MS Visual C++ with WINDOWS is assumed. If option _GNU_LINUX is not passed to the compiler explicitly, than *MICROSOFT*WINDOWS is assumed. **DO NOT SWITCH THIS OFF**, as in case of the linux Makefile the _GNU_LINUX option is automatically switched on.

Second option: whether the code will be built for normal running (this is what usually needed), or for running in test mode. The later was introduced during the testing of the programs to ensure, that the random number generator start with the same value each time, and the program will run to a given number of generated steps specified by LAST_GEN to make the results produced by the different version comparable. It is also useful for performance testing.

#define _TEST_MODE //this has to be disabled, if the program is used in normal running mode

#define LAST_GEN 1000//the run will end at ngenerated=LAST_GEN in test mode (this does NOT have to be disabled!!!)

Third option: whether build a multi-threading application. If switched on, multi-threading is possible, the thread libraries needed, but the program can still run with only one thread! DO NOT TOUCH IT for WINDOWS with Microsoft Visual C++, as it is passed to the pre-processor by the project setting of the RMC_POT.dsw project file, and it is not passed by the consecutive RMC_POT_s.dsw. Do not touch this either, if you use the Linux Makefile provided with the source code, only pass the MULTI=0 to make, if you want to switch it on. ONLY switch it on or off, if you do not use the project files or Makefile provided! Do not compile for multi-threading only if you do not have more, than one processor and no thread libraries, in this case of course only one thread is assumed, (usual consecutive code) is generated..

#define _MULTI

Fourth option: the *ppcf*-s will be summed at each saving, and the average is calculated and saved too

//#define _SUM_PPCF

Fifth option: regulating the LINUX platform based ATLAS library usage. The libraries have to be installed separately (see the ATLAS web site), only use this option if they are installed! The installed ATLAS libraries are using the BLAS routines, optimised for the given platform, and can increase the speed of the vector-vector, matrix-vector and matrix-matrix operations. In RMCSANS ATLAS is applied for the *ppcf* calculation and the partial *S(Q)* Fourier-transformation

//#define _ATLAS//use the ATLAS libraries for matrix operations

Sixth option: only if old style header files are used (this depends on the compiler), which is most probably not the case.

//#define _OLD_HEADER //if the old style header (name.h) are used, THE NEW STYLE headers are used by default, this option has to be commented out normally

Seventh option: for Code Warrior on Macintosh, but the RMCt++ code was never tested on this platform, so there is no guarantee, that it can be compiled without any change! For example I have no idea how the integer represented on 64 bits is called in this case, so this was only kept for historical reasons!!!

//#define _CODE_WARRIOR_MAC //DEFAULT is the PC or UNIX version, this option has to be commented out normally

Eights option: additional to the normal output, a **.out* file will be created to be compatible with RMCA containing the PPCF-s and partial *S(Q)* data, then the RMC and (renormalised) experimental total *I(Q)*.

#define _OLDFORMAT_OUT

2. Constant values

There are some preset constant values in the *units.h* file, which can be altered if need arises. These are:

#define FILE_NAME_LENGTH 30 : length of the file names without extensions

#define TOLERANCE 1.0e-15 : the tolerable difference coming from the different number representation in the binary and decimal number system (if the numbers are represented by 15 digits after the decimal point)

#define GRID_TOL 1.0e-14 : this is used to ensure the accuracy of the bin->dr conversion

#define WAIT_FOR_FILE 2 : amount of time in seconds to wait before trying to open a file again, if file open failed

#define WAIT_LIMIT 10 : maximum amount of time in seconds to wait for a file without error message

#define SAFE_ADD 6 : this is a safety increase for array dimensions in NeighbouList object

#define LOAD_TOL 1.0e-9 : used during the load of CoordNumbConst and CoordConst and RunParams

The CACHE related things are architecture dependent, the given values and caching concept is for the Intel64 architecture.

#define NUMBER_OF_CACHE_LINES_TO_FETCH 1 : Number of cache lines to cache in the same time (pre-fetch)

#define CACHE_LINE_SIZE 64 : Byte .Size of the L1 cache, needed in some cases to optimise cache usage. False sharing between threads has to be prevented. This can happen, when although different threads are writing different memory addresses, but the addresses are so close to each other, that they would be cached together into the same cache line (are inside the same CACHE_ALIGNMENT block). Because of this, if one part of a cache line is modified, the whole cache line is written back to memory, so different threads may want to write the same part of the memory holding back each other causing if this happens too often to slow the performance down.

#define CACHE_PADDING NUMBER_OF_CACHE_LINES_TO_FETCH * CACHE_LINE_SIZE : the thread segments of some arrays have to be separated at least with CACHE_PADDING-size(data_type) amount of bytes have to be kept between the threads segment data.

3. Compilation on Linux platform, the usage of the Makefile for RMCSANS

The supplied Linux Makefile can have the following command line options, which will regulate the building of the code. The status of the switches in *altern.h*, whether they are commented out or not is of no consequence, as the Makefile will always pass the Linux platform specific _GNU_LINUX switch to the compiler, and the options in the *altern.h* will be bypassed. Instead, an option can be switch on by passing command line arguments to the make. The name of the executable will contain indicators of the used option switches to avoid confusion. The file names will always begin with '*rmcsans*' and end with '*.exe*'.

The command line options for make, and the indicators in the executable name are the following:

Command line argument	file name indicator	option switch
TEST=X	_t	for _TEST_MODE, X is the number of generated steps to make
MULTI=0	_multi	for _MULTI threading
SUMP=0	_sp	for _SUM_PPCF
AT=0	_atlas	for _ATLAS
OH=0	_oh	for _OLD_HEADER
MAC=0	_mac	for _CODE_WARRIOR_MAC
OLDOUT=0	_oo	for _OLDFORMAT_OUT
ARCH=X	X	where X is a number, this adds the X to the end of the file name before extension to differenciate between different architecture, if necessary

For example compiling the RMCSANS code with ATLAS usage, summing the ppcf and compiling for consecutive execution on a 64-bit architecture use

make AT=0 SUMP=0 ARCH=64

This will result in executable named rmcsans_atlas_sp64.exe

4. Compilation using Windows operation system with Microsoft Visual C++

There will be two set of workspace files *(*.dsw, *.dsp, *.ncb, *.opt, *.plg*), the *: RMCSANS_s is for the standard consecutive version, and the *: RMCSANS is for the parallel version. The parallel version needs the thread libraries, this is the reason for having two workspaces. Both workspaces use the same header and source files.

The passing of the _MULTI compiler option switch by the parallel RMCSACS workspace is done automatically by the project setting, so do not touch the switched off _MULTI switch in *altern.h*!

Choose among the other preferred option switches in the *altern.h* file by turning them off (commenting out) or turning on. Then build the application. The consecutive executable will be named RMCSANS_s.exe, and the parallel RMCSANS.exe, regardless the chosen option switches.

B. Starting the program

The program can be started by the executable file name followed by the following variation of command line arguments:

- ***exename***
- ***exename filename***
- ***exename filename y***
- ***exename filename frame_offset***
- ***exename filename frame_offset y***
- ***exename filename frame_offset AtomEye_index***
- ***exename filename frame_offset AtomEye_index y***

If the program is started without any command line option, then it will ask for the ***filename***, give it without any extension. In this case the simulation will be started from the very beginning, so **.cfg* or **.bcf* file without any frame offset has to be present.

The ***frame_offset*** is for starting the program with the simulation of the ***frame_offset***-th time frame. For example, if we want to continue the simulation from the coordinates of the 2nd time frame, and lets assume, that the filename is ***mytest***, then start the program with

```
exename mytest _2
```

and the program will look for ***mytest.dat, mytest.cfg_2*** (***mytest.hgm_2, mytest.grid_2,*** if constraints present then for ***mytest.acn_2, mytest.cnc_2,*** but these are not necessary, if not found, they will be recalculated). The same **.dat* file can be used, regardless which time frame the simulation is started from, and do not reset the number of time frames!

'***y***' indicates to close the window at the end of simulation, which is useful, if the output is redirected to a file, and the program is running in the background.

AtomEye_index will be used as the serial index of the first saved AtomEye configuration file, if not specified, 0 is assumed. The AtomEye configuration files will be numbered continuously during the whole simulation series of the consecutive time frames, the name of the last AtomEye configuration file for a given frame will be written into the **.hst* file.

---

[1] McGreevy, R.L., Pusztai, L.: Molec. Simul. **1**, (1988) 359.
[2] Pusztai, L.: J. Non-Cryst. Sol. **227-230**, (1998) 88.
[3] McGreevy, R.L.: J. Phys.: Cond. Matter **13**, (2001), R877.
[4] Evrard, G., Pusztai, L.: J. Phys.: Cond. Matter. **17**, (2005) S1.
[5] Gereben, O., Jóvári, P., Temleitner, L., Pusztai, L.: J. Optoelectron. Adv. Mater. **9**, (2007) 3021
[6] Gereben, O., Pusztai, L., McGreevy, R.L.: J. Phys.: Condens. Matter 22 (2010) 404216
[7] Li J 2003 Modelling Simul. Mater. Sci. Eng. 11 173
[8] Futoshi S and Li J 2006 AtomEye 3.0 software, http://mt.seas.upenn.edu/Archive/Graphics/A3/A3.html