SpecSwap-RMC Version 1.0a0
User Manual

Mikael Leetmaa <leetmaa@fysik.su.se>

2013-01-21

Contents

1 About this document

This document is intended to serve as a guideline for the use and input of the SpecSwap-RMC program. It is not intended as a tutorial or a full-fledged description of the SpecSwap-RMC method. But it is still our hope that it can serve as a useful guidance to get started with setting up your first SpecSwap-RMC simulations. And if ever in doubt about the intended behavior of a particular input keyword this document should be consulted. The aim has been to write an as up to date and accurate description as possible of all the input parameters and file formats, however there might still be errors in this document. If you believe a keyword or format behaves differently from what is described here you could check the source code to see what it really does. And if you do find bugs,

issues, inconsistencies or errors in this document, or in the source code, please don't hesitate to contact the SpecSwap-RMC authors so that the problem can be fixed. You find our contact information in the README file in the source code distribution.

2 The method in brief

The SpecSwap-RMC method [1] is a flavor of reverse Monte-Carlo (RMC) [2] that makes use of a finite discrete set of local structures, for which all relevant data is pre-computed, to expand the configuration space in these structures. The main reason for using the SpecSwap-RMC method compared to conventional RMC is to avoid expensive re-calculation of the data, making it practically manageable to fit data sets for which the theoretical evaluation is very CPU expensive. SpecSwap-RMC thus makes it possible to use RMC with data such as full multiple scattering EXAFS, XAS and NMR, where the nature of the problem and the need for accuracy demands expensive electronic-structure calculations, typically too costly to include in a straight forward way in a conventional RMC simulation.

All relevant data for the local structures are pre-computed. A sub-set of the available local configurations (the *sample set*) is taken out to represent the configuration of the system. A trial move in the Monte-Carlo run is performed by replacing ("swapping") a local configuration present in the sub-set with a local structure that is not. The re-calculation of the data for the new configuration of the system (needed for comparing against a provided reference to evaluate the success of the move - by far the most time consuming step of a conventional RMC simulation) is reduced to a simple add and subtract operation of the data that corresponds to the swapped local structures.

When equilibrium is reached in the SpecSwap-RMC simulation (this is typically achieved very fast given the parameters are set to yield a reasonable acceptance rate) the *sample set* is probed repeatedly on a given interval to record which local structures are present. Divided by the number of times the *sample set* was probed these records converge towards a weighting of each local structure proportional to its importance within the total set of local structures to represent/fit the reference data. These weights are the main result from the SpecSwap-RMC run. Structural properties such as pair-correlation functions and angle distributions can then be calculated and compared weighted and unweighted from the total set of local structures. For further details about the method see ref [1] and for an example of the method in use see ref [3].

3 Naming conventions

This document (and also the comments in the source code) will try to be as consistent as possible with the naming conventions of ref [1].

3.1 library

We will use the term *library* for all available local structures taken together, each with its corresponding pre-calculated data and compiled in to a binary file. The *raw library* or *library raw data* are the files making up the library before being compiled into a binary library file.

3.2 basis set

The set of all available local structures in the library will be referred to as the *basis set*.

3.3 basis element

Each local structure with its corresponding pre-calculated data will be called a *basis element*.

3.4 sample set

The sub-set of the *basis set* that, at any given moment during the SpecSwap-RMC run, represents the system configuration, is referred to as the *sample set*.

4 Before you start

Since the SpecSwap-RMC method uses pre-computed data from a (large) set of local structures you will need to select the local structures and generate data before you start the simulation.

4.1 Finding suitable basis structures

For a successful (and meaningful) SpecSwap-RMC simulation the structures taken as the basis set must be chosen with some care. They must be close enough to the real structure (the structure corresponding to the reference data modeled) to get a reasonable fit without a too low acceptance rate, yet be diverse enough for having the possibility of representing the real diversity of the local geometries in the material under study.

All to date practical applications of the SpecSwap-RMC method known to the authors has used structures cut out from MD simulations, conventional RMC simulations, or generated via random distortions around a known or assumed equilibrium geometry see e.g. refs [1, 3]. However, at the time of writing there exists no standard way to obtain suitable basis structures, so the best advice is to be careful and use your creativity.

The single most important aspect when constructing a *basis set* library is that it must be possible to fit the data. If no satisfactory fit can be obtained the basis set must be expanded. Consider a case with a *basis set* containing 10000 *basis elements* and a *sample set* of 100 *basis elements*, and we assume for the argument that the data has been calculated with infinite accuracy. If it turns out impossible to obtain a reasonable fit to the reference that would be equivalent to say that there are no (or way too few) combinations of 100 *basis elements*

drawn from the set of 1000 that can represent the data. This means there are not enough *basis elements* with a geometry corresponding to the reference or that we are missing some very important structural components in the *basis set*. If the fit improves drastically when reducing the size of the *sample set* this is typically an indication of an insufficient basis set. There are simply to few *basis elements* with some important quality missing. The weights can is in such a case still be quite useful. Examining the structures of the highest weighted *basis elements* would give an indication of what type of structures are missing in the *basis set*.

4.2 Computing the data

For each chosen local geometry all relevant data must be calculated before the library is generated, with one important exception. When generating the library (see below), the `mklib` program calculates and stores the distributions of distances with respect to the atoms defined as the central molecule for each basis element.

Since all other data to use in SpecSwap-RMC must be computed before the library is generated the program is necessarily general such that it can not know what the data represents. Any type of data that can be stored as a function of one variable, i.e. any type of curve data, can be included. Any higher dimensional data must be mapped down to two dimensions before inclusion is possible. Two dimensional data goes here under the name *curve data*. Data represented as a single number on each *basis element* is named *scalar data*. *Scalar data* could e.g. be a frequency, a number classifying the local structure, the local density or Voronoi polyhedra volume. All data included in the fit (except the pair correlation functions) must be represented either as *curve data* or *scalar data*.

4.2.1 A note on accuracy

It is important to remember that the accuracy of the results from structure modeling with RMC is determined by the accuracy with witch the data to fit can be calculated. Structure modeling with SpecSwap-RMC is not and exception. If the data is not calculated accurately enough from a given structure, the resulting structure from the RMC simulation will not correspond to the real physical structure, and conclusions drawn can be quite wrong. For a sound use of RMC one needs to be aware of the particular limitations and error bars for the type of data which is used for the fit, and conclusions drawn should always be evaluated with the accuracy of the method used to calculate the data in mind. For SpecSwap-RMC simulations it is also crucial that a good enough *basis set* library is used.

5 Generating a library with Mklib

When all data is calculated form the chosen local structures to use as *basis set* the data needs to be compiled into a library file together with the corresponding geometries, to be used in the SpecSwap-RMC simulation. The SpecSwap-RMC program uses a binary file format defined by the binary I/O implementation

in the `library.cpp` file. The SpecSwap-RMC software package includes a program, `mklib`, that can generate the binary library files from the raw library data.

To generate the library several data files are needed for each *basis element*; a file holding the geometry, a file holding the curve data for each curve and a file holding all scalar data. The files corresponding to the same *basis element* must have the same base name, with endings corresponding to the names of the curves, `.xyz` for the geometry and `.sclr` for the scalars. For each curve data a file giving the scale is also needed. All files are then placed in a folder with the same name as the intended library name, and a manifest file, `libraryname.info` (the *info file*) with specific info about the library and the constituent basis elements is placed in the directory. The syntax of the `.info` file is rather strict fixed format so extra care must be taken to type every thing in correctly. The `mklib` program can then, with the information from the `info` file, be used to compile all data into a binary library file for use in SpecSwap-RMC, by typing:

```
./mklib3.0.x -c libraryname
```

This makes `mklib` read the file `libraryname.info` from the `./libraryname` folder where also all other files to compile must be present. Please see 6 below, as well as the examples shipped in the `/functest/testmklib` folder in the SpecSwap-RMC source, for details.

6 Mklib input

This describes the input format of the `.info` file for `mklib`, as well as the files refered to from that file.

6.1 Curves

The first line in the `.info` file is the

```
NCURVES N
```

keyword, where N is the number of specific curve data types to include in the library, e.g. 2 if including both say EXAFS and XAS.

After the `NCURVES N` line a set of four lines appear N times, once for each curve. These are the lines:

```
CONVOLUTE D1 D2 D3 D4
FORMAT start stop steps
SCALE scale_filename
ENDING ending
```

where `D1 D2 D3 D4` is any set of four floating point numbers for the user to classify the curve. The name of the keyword stems from the use of a particular type of broadening scheme for TP-XAS calculations. The values are stored in the library file but in the present version they are not used. The `start`, `stop`

ands `steps` values for the `FORMAT` keyword specifies the first and last x values and the number of data points to expect in each curve file (one for each `basis element`, see below) of the type given as parameter to the `ENDING` keyword. If say EXAFS is fitted the ending could e.g. be `.exafs`, and the keyword would the be given as

```
ENDING exafs
```

All files with the `.exafs` extension would then need to have the data format with two columns, the first column giving the x values, with the first and last values as well as number of lines as specified with the `FORMAT` keyword.

The argument to the `SCALE` keyword is the file name of a data file to use as scale for the x axis. The scale file must have the number of data lines (the same value as the `steps` value to the `FORMAT` keyword) on the first line. The data must be given in the correct number of lines, in two columns. The x values (the data in the first column) will be stored in the library while the values in the second column will not. All basis element curve data of the same type must match this x scale, and all reference curves of this type used in subsequent SpecSwap-RMC runs with this library must also match in x values. This is to ensure that no mistakes are made by comparing data which are not normalized to the same x scale.

As an example, if we fit EXAFS and XAS the whole curves part of the `mklib` input file could look like this:

```
NCURVES 2
CONVOLUTE 1.0 2.0 1.0 3.0
FORMAT 2.3 7.65 108
SCALE dummy_exafs_ref.data
ENDING exafs
CONVOLUTE 0.5 535.0 1.0 545.0
FORMAT 530.0 550.0 201
SCALE xas_scale_file
ENDING xas
```

If no curves data should be included in the library the line `NCURVES 0` is given at the top, followed directly by the scalars information (see below).

6.2 Scalars

After all the curve information is given the scalar information is presented in two lines as:

```
NSCALARS N
NAMES name1 name2 ...  nameN
```

Where `N` is the number of scalars and a unique name for each scalar must be given as a space separated list after the `NAMES` keyword. Say that we have two scalars, the Voronoi polyhedra volume and area this could look like:

```
NSCALARS 2
NAMES VP-area VP-volume
```

If there are no scalar data in the library `NSCALARS 0` should be given and the `NAMES` line should be skipped.

6.3 Geometry

The scalars input is followed by a line indicating how many atom types there are. The names of the elements could be any strings but it important that the atom type names correspond to the data given later in the geometry `.xyz` files. The format is:

```
NATOMTYPES N name1 name2 ...  nameN
```

If all local configurations have the same number of atoms and if the same indices should be considered included in the central molecule an optional keyword `ATOMS` should be given after the `NATOMTYPES`. The `ATOMS` keyword takes the arguments:

```
ATOMS M M1 M2 ...  MN MOL Q Q1 Q2 ...
```

where `M` is the number of atoms (the same for all `.xyz` files), `M1` the number of atoms of the first atom type, assumed to be the first `M1` entries in the `.xyz` file, `M2` the number of atoms of the second atom type, etc. After the last integer indicating the number of atoms of a specific type the word `MOL` is written, followed by the number of atoms in the central molecule `Q`, and then `Q` integers telling the indices (atom numbers indexed from zero) in the `.xyz` of the central molecule atoms. Note that the first index in this list should always be 0 (zero). To clarify with an example, if we have 39 water molecules the input could look like this.

```
NATOMTYPES 2 O H
ATOMS 117 39 78 MOL 3 0 39 40
```

6.4 Basis name listing

Lastly the keyword `START` is given followed by the number of basis elements to compile into a library. Then all the base names (without endings) are given, one on each line, followed by the final line with the word `END`. In a realistic situation this list is likely to be several thousand lines long but we give here as an example of the format a basis name listing with 5 entries.

```
START 5
example_basename1
example_basename2
example_basename3
example_basename4
example_basename5
END
```

6.5 Data for each basis element

The data for the library is given in files named after the listed base names of the basis elements, with endings corresponding to the curve, scalar and xyz data.

6.5.1 Curve data

For each curve and listed basis name there must be a file name `basename.ending` giving the curve data. In our above example with `.xas` and `.exafs` data endings the files for the first listed base name would be:

```
example_basename1.xas
example_basename1.exafs
```

The data in these files are given in two columns with the number of lines corresponding to the `steps` value given to the `FORMAT` keyword in the `.info` file. The first column data must match the given values in the corresponding scale file.

6.5.2 Scalar data

For each basis element there must be a file named `basename.sclr` holding scalar information. The scalars are given as a space separated list of floating point numbers on one line, one number for each scalar in the system. All basis elements must have the same number of scalar values, as specified in the `.info` file.

6.5.3 Geometry data

The geometry data from each basis element is given in a file `basename.xyz`. The format of this file is the "standard" xyz format, the first line gives the number of atoms. Followed by a blank line. Then one line for each atom with the symbol and x, y and z coordinates.

```
N

symbol1 x1 y1 z1
symbol1 x2 y2 z2
...
symbolN xN yN zN
```

The types and number of atoms must match the `NATOMTYPES` and `ATOMS` information given in the `.info` file.

6.6 A full example

Putting all the above examples together the content of the `.info` file looks like this:

```
NCURVES 2
CONVOLUTE 1.0 2.0 1.0 3.0
```

```
FORMAT 2.3 7.65 108
SCALE dummy_exafs_ref.data
ENDING exafs
CONVOLUTE 0.5 535.0 1.0 545.0
FORMAT 530.0 550.0 201
SCALE xas_scale_file
ENDING xas
NSCALARS 2
NAMES VP-area VP-volume
NATOMTYPES 2 O H
ATOMS 117 39 78 MOL 3 0 39 40
START 5
example_basename1
example_basename2
example_basename3
example_basename4
example_basename5
END
```

And for this particular example it is also required that the files `dummy_exafs_ref.data` and `xas_scale_file` are present, as well as for each base name the files with endings: `.exafs .xas .sclr .xyz`

7 Running a simulation

Once all data is calculated and the library is set up running the actual SpecSwap-RMC simulation is fairly straight forward. The SpecSwap-RMC executable is run with the name of the input file `name.inp` withouth the `.inp` ending as the only argument, and the output to stdout can be sent to an output file of choice:

```
./SpecSwapRMC.x name > output
```

See 9 below for a description of the input file format. For a few proper working examples please see the examples in the `functest/testspecswap` folder in the SpecSwap-RMC source distribution.

8 Analyzing results

When the SpecSwap-RMC simulation is finished i.e. when it has run for a long enough time for the weights to approach convergence, there is a single, yet very useful, built in analysis option. The typical usage is to make a new restart from the converged weights, but instead of running the Monte-Carlo loop use the option available in the ANALYSIS input section to generate weighted and unweighted curves. See 9.6 below for more details. See also the examples in the `functest/testspecswap` folder in the SpecSwap-RMC source distribution.

9 SpecSwap-RMC input

The SpecSwap-RMC program takes the name of the input file without the `.inp` extension as the only command line argument. This file specify fully the behavior of the program.

9.1 Input structure

The input file is structured in keyword sections, each with a specified set of possible keywords.

9.1.1 Sections

Each section begins with a & character followed by its name, and ends with the &END keyword. Inside the section are one or more keywords given, some of which are optional, as described in more detail below. Some sections also take a keyword argument directly after the name.

&SECTIONNAME <argument>
KEYWORD1 <arg1 arg2 ...>
KEYWORD2 <arg1 arg2 ...>
...
&END

The first section is always the RUN section which controls the general flow of the program. For each curve fit there must be a CURVE section and for each scalar fit there must be a SCALAR section. If fitting a pcf there is a PCF section to add and if interested in post run analysis one adds an ANALYSIS section. Comments starting with a # character are allowed in the input file between but not inside the sections.

9.1.2 Keywords and arguments

Each section has a particular set of valid keywords. Most keywords take one more arguments as specified below. Also some of the sections take arguments, e.g. the SCALAR section, to determine which scalar to fit. Below is a list of all sections and their keywords.

9.2 &RUN

The RUN section control all basic run parameters, how many steps to take, which library file to use, when to save results, etc. A RUN section must be present as the first section in any input file. Below is a listing of all valid RUN section keywords.

9.2.1 TITLE < *title* >

Set a title to the run. This title is used as the base name of the generated restart and weights files. This keyword is compulsory. The title must be given as a single string (no spaces).

9.2.2 LIBPATH < *filepath* >

Set a path to a library file to use for the run. This keyword is compulsory. The path is given relative to the directory in which the program is executed.

9.2.3 SEED < *value* >

Set a seed value to the random number generator. This keyword is compulsory. The value must be given as a positive integer.

9.2.4 NSAMPLE < *value* >

Set the number of elements in the sample set. This keyword is compulsory. The value must be given as a positive integer.

9.2.5 MOVES < *value* >

Set the limit of the run, given as the number of *attempted* moves to take. This keyword is compulsory. The value must be given as a positive integer.

9.2.6 PRINT < *value* >

Set the interval, in number of *accepted* moves, for printing to screen / stdout. This keyword is compulsory. The value must be given as a positive integer.

9.2.7 PROBE < *value* >

Set the interval, in number of *accepted* moves, for probing the sample set to collect weights. This keyword is compulsory. The value must be given as a positive integer.

9.2.8 DUMP < *value* >

Set the interval, in number of *accepted* moves, for writing the momentaneous curve and pcf averages from the sample set. This keyword is compulsory. The value must be given as a positive integer.

9.2.9 RESTART < *filepath* >

Set the path to a restart file to make a restart from a previous run. This keyword is optional. With no RESTART keyword given the run will start from a random samples set, with all weights starting from zero. The file path is given relative to the folder the program is executed from.

9.3 &PCF

The PCF section is added to the input file to fit a reference pair correlation function. These are the valid keywords:

9.3.1 RMIN < *value* >

Give the first r value of the reference. This keyword is compulsory.

9.3.2 RMAX < *value* >

Give the last r value of the reference. This keyword is compulsory.

9.3.3 DR < *value* >

Give the spacing between the bins. This keyword is compulsory. It is important that the number of bins as calculated from (RMAX-RMIN)/DR is integer.

9.3.4 FIT < *lowervalue* > < *uppervalue* >

Give the r interval to fit. This keyword is optional. If not given the whole interval from RMIN to RMAX will be used.

9.3.5 SIGMA < *value* >

The sigma value to use in the fitting. This keyword is compulsory. The value must be given as a positive floating point number.

9.3.6 NUMBERDENSITY < *value* >

The number density needed for normalizing the PCFs. This keyword is compulsory. The value must be given as a positive floating point number.

9.3.7 PARTIAL < *firsttype* >< *secondtype* >

Specify the atom types of the partial to fit. This keyword is compulsory. The atom types are referred to in the order they have in the library, i.e., as given in the `.info` file when the library was setup.

9.3.8 PATH < *filepath* >

The path to the file holding the reference data. This keyword is compulsory. The path is given relative to the directory in which the program is executed. The reference data file should have two columns, the first giving the r points and the second the corresponding g(r) data. The bin size and max and min values must match the values given in the RMIN and RMAX keywords.

9.4 &CURVE < *name* >

The CURVE section is added to the input file to fit a reference curve. The curve name, as present in the library (given as the ending of this curve type in the `.info` file when constructing the library) is given as argument. Valid keywords are:

9.4.1 EXPPATH < *filepath* >

The path to the file holding the reference data. This keyword is compulsory. The path is given relative to the directory in which the program is executed. The reference data file should have two columns, the first giving the x points and the second the corresponding curve points. The bin size and max and min values must match the values given in the `.info` file when creating the library.

9.4.2 AREARENORM

This optional keyword works as a switch. If the keyword is present this indicates that the fitted curve should be re-normalized to the same area (computed as the sum of all y values multiplied by bin size, thus assuming all values to be positive) before compared to the reference.

9.4.3 SIGMA $<$ *value* $>$

The sigma value to use in the fitting. This keyword is compulsory. The value must be given as a positive floating point number.

9.5 &SCALAR $<$ *name* $>$

The SCALAR section is added to the input file to fit a reference scalar, either as a mean value, a specific value or as a distribution. The scalar name, as present in the library (given as the name of the scalar type in the `.info` file when constructing the library) is given as argument.

The SCALAR section comes in three different flavors, depending on what type of fit is to be performed. This is controlled by adding one of the three keywords MEAN, VALUE or DISTRIBUTION as the first keywords in the SCALAR section. Only one of these keywords can be added to a given SCALAR section, but there is no limit to how many different SCALAR sections there can be in one input file.

9.5.1 MEAN $<$ *target* $>$

To fit the mean value of the specified scalar. The target value is given as a floating point number.

9.5.2 VALUE $<$ *lowerlimit* $>$ $<$ *upperlimit* $>$ $<$ *target* $>$

To fit to a fraction within a certain interval. The target value refers to the fraction of all element in the sample set that should have their scalar value between the given lower and upper limit.

9.5.3 DISTRIBUTION $<$ *filepath* $>$

To fit a reference distribution. The file path is given relative to the directory in which the program is executed. The reference data file should start with an integer on the first line giving the number of subsequent lines of data. The data should have three columns, the first giving x (bin) values and the second the distribution value. The third column gives a floating point number which will be multiplied with the error in that point/bin to allow for weighted error values. This is particularly useful if one wants to fit harder to some parts of the reference distribution. The distribution is area normalized with the corresponding scalar distribution when comparing. The first and last bins includes every thing below and above the distribution and the x (bin) values in the first column refer to the center of the bin.

9.5.4 SIGMA $< value >$

The scalar section must have the SIGMA keyword with the specified value. This keyword is compulsory.

9.6 &ANALYSIS

The ANALYSIS section is added to the input file to perform an analysis run after the Monte-Carlo run has finished. The ANALYSIS section only takes one keyword.

9.6.1 CHUNKS $< numberofchunks > < chunksize >$

This keyword is compulsory. All *basis elements* in the library are sorted according to their weight. The *basis elements* are then bunched into chunks and for all data sets, (curves, scalars, pcfs) specified in the input file the data for each chunk is written weighted and unweighted to file. The number of chunks and chunk size are given as arguments. Note that if the number of chunks times the chunk size is larger than the number of elements in the basis set library the last chunk will be truncated accordingly.

10 Additional sources of information

For a source code reference manual see the document `refman_vX.X.pdf` distributed with the SpecSwap-RMC source code. For a hint on how to extend the program to fit other types of data, or how to extend it with custom analysis functions there might be a development manual written in the future. Contact the SpecSwap-RMC authors if you are interested in such a document or further guidance.

References

[1] M. Leetmaa, K. T. Wikfeldt, and L. G. M. Pettersson. SpecSwap-RMC: A novel reverse Monte Carlo approach using a discrete set of local configurations and pre-computed properties. *J. Phys.: Cond. Matter*, 22:135001, 2010.

[2] R. L. McGreevy and L. Pusztai. Reverse Monte Carlo simulation: A new technique for the determination of disordered structures. *Mol. Simul.*, 1:359–367, 1988.

[3] K. T. Wikfeldt, M. Leetmaa, A. Mace, A. Nilsson, and L. G. M. Pettersson. Oxygen-oxygen correlations in liquid water: Addressing the discrepancy between diffraction and extended x-ray absorption fine-structure using a novel multiple-data set fitting technique. *J. Chem. Phys.*, 132:104513, 2010.