Advanced RMC analysis tools Documentation

Release rc-0.1

Yuanpeng Zhang

May 24, 2022

CONTENTS:

This documentation aims to provide useful information about those advanced analysis tools for RMCProfile.

CHAPTER

ONE

INSTALLATION

1.1 RMC Modules

All RMC modules have been packaged up and made available through *conda*. Users can simply install the package through *conda*,

```
conda install -c apw247 rmc_tools
```

1.2 RMC Tools

All RMC tools have been included in RMCProfile release package available at, https://sourceforge.net/projects/rmcprofile/, **except** *topas4rmc* and *sofq_calib* (for these two packages, installation instructions can be found in their dedicated documentation page at *Topas4RMC* and *SofQ_Calib*, respectively). Users can just download the RMCProfile releaes package on a specific platform and no extra installation is needed to get access those RMC analysis tools (again, *topas4rmc* and *sofq_calib* not in the list here - they do need extra bit of installation).

CHAPTER

TWO

RMC MODULES

2.1 Bulk RMC6F configuration processing

This modules holds stuff relevant to processing bulk RMC6F configurations.

class rmc_tools.bulk_stuff.**BulkStuff**

RMC6F bulk configuration processing class.

This class contains methods for processing bulk RMC6F configurations.

bulk_to_shells(*rmc6f_config*)

Method for grabbing shells from bulk RMC6F configuration.

Provided bulk RMC6F configuration, this method can extract shells from it. Since this program is to deal with bulk configuration, the center does not matter that much and therefore it is randomly placed in the bulk configuration.

Arguments: rmc6f_config {Object} – Instance of *RMC6FReader* class

Output: shell_X.rmc6f – RMC6F configuration for generated shells.

np_shell_gen.log – Log information about shells generation.

2.2 Nano RMC6F configuration processing

This modules holds stuff relevant to processing nano RMC6F configurations.

class rmc_tools.nano_stuff.**NanoStuff**

RMC6F nano configuration processing class.

This class contains methods for processing naon RMC6F configurations.

cent_rad_config(*rmc6f_config*)

Method for figuring out the center and radius of the input RMC6F nano configuration.

Provided nano RMC6F configuration, this method can extract information about the center and radius of the input nanoparticle configuration.

Arguments: rmc6f_config {Object} – Instance of *RMC6FReader* class

Output: The center and radius of input nanoparticle configuration can be accessed from instance variable *centPos* and *NPRadius*.

np_to_shells(*rmc6f_config*)

Method for grabbing shells from nano RMC6F configuration.

Provided nano RMC6F configuration, this method can extract shells from it. The center of the provided nanoparticle will be figured out automatically.

Arguments: rmc6f_config {Object} – Instance of *RMC6FReader* class

Output: shell_X.rmc6f – RMC6F configuration for generated shells.

np_shell_gen.log – Log information about shells generation.

2.3 General routine for RMC6F configuration processing

This modules holds stuff relevant to processing RMC6F configurations in general.

class rmc_tools.rmc6f_stuff.**RMC6FReader**(*file_name*)

RMC6F configuration reader

Given the full path of RMC6F configuration file as input, when declaring instance to this class, it will read in read in the RMC6F configuration. Several instance variables will be made available, as detailed below,

Variable name	Property	Type
self.atomsCoord	Atomic coordinates for all	list
self.atomsCoordInt	RMC Internal atomic coordinates	list
self.atomsEle	Element symbol for all atoms	list
self.atomsLine	All atom lines	list
self.atomTypes	Atom types present	list
self.fileName	The input RMC6F file name	string
self.header	Head lines of RMC6F file	string
self.initNumRho	Number density	float
self.lattPara	Lattice parameters	list
self.numAtoms	Number of atoms	int
self.numAtomEachType	Number of each atom type	list
self.numTypeAtom	Number of types of atoms	int
self.scDim	Supercell dimensions	list
self.uniq_ref	Compressed info for all atoms	dict
self.vectors	Lattice vectors	list

rmc_tools.rmc6f_stuff.**dist_calc_coord**(*coord1*, *coord2*, *vectors*)

Distance calculator

Parameters

- **coord1** (*1D list or numpy.array with 3 entries.*) – Coordinate of first atom.
- **coord2** (*1D list or numpy.array with 3 entries.*) – Coordinate of first atom.
- **vectors** (*2D list or numpy.array*) – Lattice vectors - x, y and z, respectively.

Returns Distance between the two input atoms.

Return type float

CHAPTER

THREE

RMC TOOLS

3.1 Nanoparticle generator

This routine provides capability of generating nanoparticle(s) from provided RMC6F configuration file as input. This script has been embodied into the RMCProfile release package, so that one can launch the RMCProfile terminal and then execute

```
np_gen RMC6F_FILE_FULL_NAME
```

to run the generator.

Here following are the major features of this program,

- Generate spherical particle(s) from RMC6F config.
- Generate particles with facets, assuming cubic symmetry, or no symmetry.
- Keep the surface terminated.
- Generate randomly packed multiple particles in box.

Attention: We need to generate a huge RMC6f configuration (using, e.g. data2config), say, with the dimension of 50x50x50.

Attention: The program will ask a bunch of questions during execution, as detailed below.

1. **The generation scheme – '0' for generating multiple particles (as close packing as possible), '1' for generating a single nan**

   If '1' is selected

   2. Particle shape – '0' for sphere, '1' for polygon.

      If '0' is selected 2.1. Particle radius

      If '1' is selected 2.1. Roughly estimated particle quasi-radius (of the polygon).

      2.2. Input file containing the facets list.

      Here follows is provided an example facet list file,

      ```
      Example facet list file
      ```

      Explanation for entries is provided below,

Entry	Description
DISTANCE_MEAN_VAL	Expected perpendicular distance from the center to each facet
DISTANCE_VAR_VAL	Variation of the aforementioned distance
CUBIC_SYMMETRY	By specifying cubic symmetry for the system, equivalent facets will be generated automatically
FACETS	The number of facets to be specified in the list. List of facets should be provided following this line, consecutively

3. Location of the particle center. A list will be given here, we only need to refer to the list for input options.
4. Estimated surface layer thickness in angstrom. This is for the purpose of searching for surface termination bonding later on. Usually, '3.0' is a good estimation.
5. Atom type to be fully coordinated. Usually, we may just want to guarantee metal atoms are fully coordinated and leave, e.g. oxygen dangling. Again, a list of options will be given here.
6. Atom type we want to terminate the surface with, e.g. oxygen. Here, the utility can only terminate the surface with atoms already existing in the original configuration.
7. Lower limit to check for surface bonding.
8. Upper limit to check for surface bonding.

If '0' is selected

2. Particle radius and its variation in angstrom.
3. Minimum and maximum of the gap between particles.
4. Estimated surface layer thickness.
5. Atom type to be fully coordinated. Usually, we may just want to guarantee metal atoms are fully coordinated and leave, e.g. oxygen dangling. Again, a list of options will be given here.
6. Atom type we want to terminate the surface with, e.g. oxygen. Here, the utility can only terminate the surface with atoms already existing in the original configuration.
7. Lower limit to check for surface bonding.
8. Upper limit to check for surface bonding.

Along with the output RMC6F file for the generated nanoparticle(s), there are several other output files, as described below,

Single particle generation:

np_single.out: Output information about the generated single nanoparticle and the file content should be self-explanatory.

Multiple particles generation:

np_rpm.out: Output information about the generated particles, where all relevant coordinates are given in fractional form.

np_rpm_cart.out: Output information about the generated particles, where all relevant coordinates are given in Cartesian form.

np_rpm_par_belong.out: Output information about to which particle a certain atom in the output RMC6F configuration belongs.

np_rpm_rot.out: Output information about the rotation angles for each generated particle, as compared to their original orientation in the input RMC6F configuration.

3.2 Nanoparticle linear expansion analyzer

This python routine takes the initial and fitted nano RMC6F configuration as inputs and will analyze the linear expansion of lattice in the fitted nano configuration. The goal is to find a lattice that best matches the initial one through linear expansion of the fitted lattice. Input from command line will be needed during execution and the CLI prompt should be already self-explaining.

The linear fitted configuration will be stored in whatever file name specified in the very last step. The linear fitting information can be found in a file named 'np_lin_analyzer.out'. Specially, the diagonal elements of the inverse Hessian matrix will give the uncertainty of the corresponding fitted variable.

3.3 Nanoparticle to shells

This little Python script is basically just calling the method implemented in *rmc_tools* for dividing nano RMC6F configuration to shells. The script has been embodied into RMCProfile release package so that it can be executed within RMCProfile environment, simply as,

```
np_shells RMC6F_CONFIG
```

Output: shell_X.rmc6f – RMC6F configuration for generated shells.

np_shell_gen.log – Log information about shells generation.

3.4 Bulk to shells

This little Python script is basically just calling the method implemented in *rmc_tools* for dividing bulk RMC6F configuration to shells. The script has been embodied into RMCProfile release package so that it can be executed within RMCProfile environment, simply as,

```
bulk_shells RMC6F_CONFIG
```

Output: shell_X.rmc6f – RMC6F configuration for generated shells.

np_shell_gen.log – Log information about shells generation.

3.5 RMC config strain analyzer

Python script for strain field analysis for RMC6F configuration. Details about strain analysis theoretical description can be found in the following paper,

https://doi.org/10.1107/S1600576719000372

1. For microstrain analysis (Eqn. 11 in the paper mentioned above), we need a single RMC6F configuration as the input. For the deformation gradient tensor analysis, we need the initial and fitted RMC6F configurations as inputs.

The script has been embodied into the RMCProfile release package so that it can be simply executed as,

```
rmc_strain RMC6F_CONFIG [OPTIONS]
```

For example, if we want to analyze the microstrain, we can type something like,

```
rmc_strain -i ceriaNano_NP.rmc6f -t rms
```

If we want to analyze deformation gradient tensor, we can type something like,

```
rmc_strain -i ceriaNano_NP.rmc6f -t dgt -r ceriaNano_NP_init.rmc6f
```

One can see a list of options by typing

```
rmc_strain -h
```

The full list of [OPTIONS] is presented below,

-h Show current help information.

-i [RMC6F config name] Input RMC6F configuration.

-t [rms/dgt] Analysis type.

- **-r** [Reference RMC6F config file] If 'dgt' analysis is selected, one needs to provide the reference RMC6F configuration for computing the deformation gradient tensor.

-v Show version information.

The program will then ask some questions interactively during execution,

a) If 'rms' type of analysis is selected, the program will ask whether to do the shell analysis. Then it will ask whether this is for bulk when executing shell analysis. The program was originally designed for analyzing nanoparticle, and for the analysis we need to figure out the center and radius of the particle first. Sometimes, we may also want to run the program against bulk as well just to make sure, with some confidence, what we obtain from nanoparticle shell analysis is real. However, for bulk model, it may be tricky to figure out the center and radius, so we need extra input here.

If we do want to do it for bulk, we need to first generate a nanoparticle model from the bulk configuration. To do this, we can use the 'bulk_shells.py' script in the 'NP_Shells' folder. The usage is similar to 'np_shells.py', and here we only need to generate one shell (by specifying 'by thickness' for particle generation and particle radius as the shell thickness), which is actually a nanoparticle in the center. Then we want to copy the 'shell_0.rmc6f' and 'np_shell_gen.log' to the same directory where we execute the 'rmc_strain.py' script.

Then the program will ask the minimum number of cells in each shell. We may want to figure out an estimation based on the total number of cells information printed out in the log file (see list above).

b) If 'dgt' type of analysis is selected, the program will ask for the cutoff (in angstrom) for the local deformation analysis. Usually, 10 angstrom should be good enough. This analysis will take a while since figuring out all neighbors for all atoms is time consuming.

The output is described as follows, assuming the input fitted RMC6F configuration is with the name of 'ceriaNano_NP.rmc6f'.

Output if 'rms' mode selected,

---

ceriaNano_NP_#.log – Overall microstrain result

ceriaNano_NP_#_shells.log – Microstrain for various shells.

---

where '#' represents the smallest integer that is not already existing in the output file names.

Output if 'dgt' mode selected,

---

ceriaNano_NP_dgt.out – Deformation gradient tensor for each single atom.

ceriaNano_NP_rot.out – Rotation tensor for each single atom.

ceriaNano_NP_strain.out – Strain tensor for each single atom.

ceriaNano_NP_strain_invar.out – Strain invariants for each single atom.

Explanation for the output quantities can be again found in the paper, https://doi.org/10.1107/S1600576719000372.

3.6 Topas4RMC

Graphical user interface for preparing Topas profiles to be used in Bragg pattern fitting in RMCProfile. **Running this GUI requires Topas to be already installed which is only available on Windows.** To install this GUI, users need to install *conda* first (refer to https://conda.io/projects/conda/en/latest/user-guide/install/index.html). Then the GUI can be simply installed by executing,

```
conda install -c apw247 topas4rmc
```

Detailed instruction about how to use this program has been inluced in the *Help* menu of the GUI and will not be reproduced here.

3.7 SofQ Calibration

Graphical user interface for calibrating S(Q) to Bragg pattern. To install this GUI, users need to install *conda* first (refer to https://conda.io/projects/conda/en/latest/user-guide/install/index.html). Then the GUI can be simply installed by executing,

```
conda install -c apw247 sofq_calib
```

Detailed instruction about how to use this program has been inluced in the *Help* menu of the GUI and will not be reproduced here.

CHAPTER

FOUR

EXTERNAL LINKS

- pystog for preparing total scattering data.

PYTHON MODULE INDEX

n

r

INDEX