**SuPerRod User Manual**

Canrong Qiu

1. Introduction of SuPerRod

SuPerRod is a nickname of GenX, which is originally developed by Matts Björck from Swedish Nuclear Waste and Management Company AB. GenX is developed for modeling surface scattering data (X-ray and neutron) using the global optimization fitting algorithm (differential evolution). SuPerRod is built on the backbone of GenX to fit X-ray surface diffraction data, including crystal truncation rod and resonant anomalous X-ray reflectivity. A lot of other functionalities have been incorporated to the based backbone of GenX in SuPerRod to allow it for dealing with a range of extra tasks, including data processing (background subtraction) in a semi-automatic way, building model files in a customized way, fitting surface models based on both local computer and super computation facilities, outputting bestfit model results for quick-checking purpose and for publication purpose.

To use SuPerRod, you need some basic knowledge of Python scripting language. Any research work in a publication, which has benefited from SuPerRod, should cite Matts Björck ‘s paper about GenX (Journal of Applied Crystallography, 2007, 40, 1174-1178) as an acknowledgement.

This tutorial will walk you through main features of SuPerRod.

1. SuPerRod installation

To launch SuPerRod, make sure you have installed Python (2.7 recommended) as well as following Python site-packages. Note the following combination of package versions is just one possible option, you can have other combinations as long as these packages are compatible to each other.

numpy: 1.13.3 or 1.14.0

matplotlib: 1.5.1

scipy: 1.0.0 or 1.1.0

wxpython: 2.8.12.0 (msw-unicode) or 4.0.2 msw (phoenix) wxWidgets 3.0.5

If you are using version earlier than the 2.8.12.0, you will need to change some code lines in event\_handlers.py, where the wx.FD\_OPEN and wx.FD\_CHANGE\_DIR should be replaced with wx.OPEN and wx. CHANGE\_DIR. Don’t bother to do that for later versions.

ipython: 4.0.1

pandas: 0.22.0 or 0.23.3

Step by step installation procedures:

install Python 27🡪set environmental variable of system path to include python path (usually C://Python27)🡪install pip (download get-pip.py and run python get-pip.py in a terminal)🡪install ipython using (pip install ipython==4.0.1), and same for the other package. If installation of a package using pip fails, you can download the wheel files (\*\*.whl) from some python package repository sites, then run pip install \*\*.whl to start the installation process.

The source code of SuPerRod can be found in my github website: <https://github.com/jackey-qiu/SuPerRod>, where you can download the whole package in a zip file or do ‘git clone <https://github.com/jackey-qiu/SuPerRod.git>’ in a git bash terminal window.

1. CTR/RAXR data processing

Location of the associated module: SuPerRod//accessory\_functions//ctr\_data\_processing//data\_integration.py

The background subtraction algorithm is developed by Vincent Mazet (Chemom. Intell. Lab. Syst. 76 (2), 2005). The code was originally written by Vincent Mazet based on MATLAB. I translated the scripts to Python language. Correction factors are calculated using TDL modules, developed and maintained by GSECARS 13IDC beamline at APS (Peter Eng and Joanne Stubbs are responsible persons).

1. Modify the information in the module:

spec\_path: absolute path to the folder holding the spec file

spec\_name: spec file name

substrate: the name of the substrate of your sample (either ‘hematite’ or ‘muscovite’ at this moment.

beamline: either ‘APS’ or ‘ESRF’

You may also need to fine-twick other parameters (read the documentation in the module file)

1. Run the file:

>>>Import data\_integration

>>>data=data\_integration.data\_integration()

1. Manually check the background subtracted profiles:

>>>data.integrate\_images\_twick\_mode(scan\_number=[‘feed in a list of scan number here’])

>>>use hotkeys to adjust the data integration (mostly adjust the integration window)

1. Do q correction

>>>data.q\_correction(scan\_number=?)

Interactively change the scaling factor, if you are satisfied, type SL(L is the Bragg peak L you are considering now), then move on to another L. Once you are done, type q to finish the q correction.

1. Save data

>>>data.save\_data(labels={'First\_point':[scan\_number\_start,scan\_number\_end]})

1. Pickle dump the processed data (can be restored in the future)

>>>data.dump\_data\_info()

1. Data formatting

I used a data format pattern used for loading both CTR and RAXR datasets.

In this format, there are 8 columns, labeling as X, h, k, Y, I, eI, LB, dL.

X: column representing energy for RAXR datasets, and Miller index L for CTR datasets

h: Miller index H

k: Miller index K

Y: Miller index L for RAXR datasets, and no meaning for CTR datasets (all having values 0)

I: background subtracted scattering intensity

eI: the error bar for the scattering intensity

LB: the Miller index L for the first Bragg peak, used for roughness calculation.

dL: the L spacing between two adjacent Bragg peaks, used for roughness calculation.

1. Build model files

The associated module is @ SuPerRod//scripts//build\_gx\_file\_hematite(muscovite).py

For the case of hematite (build\_gx\_file\_hematite.py), specify the following parameters:

* gx\_file\_name: name of the gx file you give
* path\_of\_data: the location of the data to use (only in the format described above)
* domain\_setup\_HLT(FLT): specify the mixture of domain types (half layer and full layer)
* water\_layer\_number: number of adsorbed water molecules for each domain

For the case of muscovite (build\_gx\_file\_muscovite.py), specify the following parameters:

* gx\_file\_name
* path\_of\_data
* RAXR\_EL: resonant element, e.g. ‘\’Th\’’
* wal: wave length of x-ray
* E0: absorption energy of resonant element
* F1F2\_FILE: f1f2 file (to be dumped in batch\_file folder)
* c\_lattice: c lattice parameter
* NUMBER\_RAXR\_SPECTRA: how many RAXR datasets
* NUMBER\_GAUSSIAN\_PEAK: how many O-equivalent Gaussian peaks (for CTR data fit)
* NUMBER\_GAUSSIAN\_PEAK\_FREEZE: how many Gaussian peaks for Resonant element (for RAXR model dependent fit)
* RAXR\_FIT\_MODE: model dependent (‘\’MI\’’) or model independent fit (‘\’MI\’’) mode
* FREEZE: True or False, if True then resonant element makes no contribution to the total CTR structure factor
* a\_range, b\_range, c\_range, A\_range: the range [min, max] of parameters for RAXR data fit

After you specify the model parameters, in the ipython terminal (cd to the right folder) do one of the following commands to build gx files:

>>>execfile(“build\_gx\_file\_hematite.py”)

>>>execfile(“build\_gx\_file\_muscovite.py”)

1. Model refinement

After previous step, you should have a customized gx file created in the folder SuPerRod//scripts. Fire up the GenX GUI (double click genx\_gui.bat for win32) and load the file, and double-check the setup of parameters. Maybe you want to click out some redundant parameters, which you don’t want to fit.

Once you are satisfied with the model file, you can submit the file to a remote folder, where a supercomputer clusters can get access to. In the compute nodes in a super computer system, you should also have a copy of SuPerRod package. Then you can navigate to the folder SuPerRod//scripts//mpi\_scripts//bash\_scripts, and first change the tag for the gx files to be run (you should tag your gx files properly. I usually put the current date at the end of the gx file, eg gx\_file\_Jun21.gx. Then run one command to change the tag info in the mpi bash scripts:>>>python tag\_pbs.py \*\_Jun21.gx.

After tagging procedure, you are ready to submit a mpi job for model refinement. Just do one of following two commands:

For model dependent fitting (CTR or MD RAXR fit), use

>>> qsub -q short-laser -l nodes=4:ppn=64 -l walltime=04:00:00 mpi\_run\_model\_dependent.pbs

For model independent fitting, use

>>> qsub -q short-laser -l nodes=2:ppn=64 -l walltime=00:20:00 mpi\_run\_model\_independent.pbs

After the completion of model refinement, the refined gx file will saved and the original gx file will be discarded. For model-dependent fit, it takes about 30 min to finish, while for model-independent fit, it taks less than 3 mins to finish.

Note you can also run the model refinement on you local computer, but it will take up to several days to finish the refinement. Therefore, it is not recommended to run gx file fitting on a local computer.

1. Output model results

The associated module is located at SuPerRod//accessory\_functions//plotting\_results//create\_plots\_gx\_file.py.

Modify the parameters before you run this file.

plot\_e\_model: True or False, would you like to plot electron density profiles?

plot\_e\_FS: True or False, would you like to plot resonant element electron density profiles based on model independent results (Fourier synthesis)?

plot\_ctr: True or False, would you like to plot CTR profiles?

plot\_raxr: True or False, would you like to plot RAXR results?

plot\_AP\_Q: True or False, would you like to plot Fourier components as a function of q? (Make sure you have temp\_plot\_raxr\_A\_P\_Q file saved in the dump\_files folder. You have to do it manually in the GenX shell. Read the standard scripts (around Line 244 for muscovite) for details.)

gx\_file\_path: the absolute path to the gx file under consideration.

With all these parameters being specified, you can now run the following command for plotting results.

>>>execfile(“create\_plots\_gx\_file.gx”)

There are also functions available to plot results for publication purpose. Read the documentation inside the SuPerRod//accessory\_functions//plotting\_results//create\_plots.py for details.

1. Some confused group names

Take sorbate Pb on domain2 as an example.

* gp\_Pb\_set1\_D2 contains four group members as ['Pb1\_D2A', 'Pb1\_D2B', 'Pb2\_D2A', 'Pb2\_D2B'] covering all Pb atoms on symmetrical domains. It is always set1 if there is only one type of sorbate. You can use this group to set u, dxdydz for metal atoms. The symmetry matrix has been set properly. You can use this group to fit dxdydz after initial geometry fitting to get the associated errors.
* gp\_HO\_set1\_D2 contains its group members as ['HO1\_Pb1\_D2A', 'HO1\_Pb1\_D2B'] covering all distal oxygen atoms on symmetrical domains associating with sorbate Pb1(set1). Therefore, there is a second group named gp\_HO\_set2\_D2 for distal oxygen ligands associated with Pb2(set2). The set is interpreted in a different way than the previous sorbate set. Such a group is used to group u for all distal oxygen. Ensure you have this command in the sim function ‘gp\_HO\_set2\_D2.setu(gp\_HO\_set1\_D2.getu())’
* gp\_sorbates\_set1\_D2 contains its group members as ['Pb1\_D2A', 'HO1\_Pb1\_D2A', 'Pb1\_D2B', 'HO1\_Pb1\_D2B'] covering sorbate atoms on symmetrical domains. There is a set2 as well. So the set here is interpreted in the same way as in distal oxygen set. Such a group is used to group occupancy for all sorbate atoms. Ensure you have this command in the sim function ‘gp\_sorbates\_set2\_D2.setoc(gp\_sorbates\_set1\_D2.getoc())’
* gp\_waters\_set1\_D2 contains its group members as ['Os1\_D2A', 'Os2\_D2A', 'Os1\_D2B', 'Os2\_D2B'] covering first adsorbed water pair on symmetrical domains. The number of set depends on how many water pair you have considered in the model. Different set of waters should be fit independently.

1. How to extract dxdydz errors for sorbate atoms (metal or ligands)?

To maintain the coordination environment for sorbates, we usually fit the associated geometrical parameters (eg bond angles, bond length). Therefore, we will only have the errors for those geometrical parameters. Then how to extract the associated dxdydz errors for sorbate atoms. The idea is to scan the geometrical parameters within the errors, and record the x y z values of sorbates. After a complete scan, you should be able to extract the boundary of x y z, from which you can easily achieve the dxdydz errors. The associated script file is located in SuPerRod/accessory\_functions/other\_acc\_funs/extract\_errors\_of\_sorbate\_position.py.