**Manual of DaFy/SuPerRod software**

Canrong(Jackey) Qiu, kiel University, Kiel, Germany

“It doesn’t matter how good your software is, because if the documentation is not good enough, people will not use it.” — Daniele Procida

Table of Contents

[1. Motivation of software development 1](#_Toc36116090)

[2. Glimpse of SuPerRod software 1](#_Toc36116091)

[3. Benchmark features of SuPerRod 1](#_Toc36116092)

[4. Targeted users of SuPerRod 1](#_Toc36116093)

[5. Installation/setup guide 1](#_Toc36116094)

[6. Operation guide 1](#_Toc36116095)

[(1)Introduction of widgets 1](#_Toc36116096)

[1) Data Frame 1](#_Toc36116097)

[2) Graph Frame 1](#_Toc36116098)

[3) Code Frame 1](#_Toc36116099)

[(2)Logic flow map 2](#_Toc36116100)

[(3)Introduction of differential evolution 2](#_Toc36116101)

[(4)Data format 2](#_Toc36116102)

[(5)Script module 2](#_Toc36116103)

[(6)Fit parameters 2](#_Toc36116104)

[7. Setup a model from scratch 2](#_Toc36116105)

[8. Two examples 2](#_Toc36116106)

[(1)Multi-Gaussian peak fitting problem 2](#_Toc36116107)

[(2)CO2RR on Cu(100) surface 2](#_Toc36116108)

# Motivation of software development

I have been dreaming of a user-friendly and powerful software for CTR fitting since 2010 when I embarked on CTR work as a PhD student at University of Alaska Fairbanks. Unlike many other X-ray techniques, eg. X-ray absorption spectroscopy, GISAXS, for which there are already numerous nice software from which you can pick to do the modeling work, the available software to fit CTR data is very limited. I started my PhD program with Prof. Tom Trainor in Alaska in 2010. My PhD project involved resolving metal binding mechanism on hematite(1-102) surface using CTR technique. To start with my CTR modeling, I first used ROD, which is written in C language. It runs fast, and it is still used by some groups nowadays, but the limitations of ROD are apparent. Firstly, it doesn’t have a GUI interface but works with a bunch of commands in terminal, which makes things obscure especially for beginners. In addition, it is cumbersome to start a model fit with ROD. Furthermore, it does not support constraints, eg. bond valence constraints, during the fit. The biggest drawback, in my opinion, is the fit algorithm that is based on non-linear least square routine, which comes up with solutions probably representing local minimum. After a while of using ROD, I learned of GenX software, which was first developed for fitting both X-ray/neutron reflectivity data. The functionality of the software had been extended to also fit CTR data afterwards. GenX is written in Python scripting language by Matts Bjorck, who is now working for Swedish Nuclear and Fuel Management Company. Once I started using GenX, I cannot stop using it. The software is user-friendly with a wxpython-based GUI. It becomes very easy to get a good hand on it. In addition, GenX is equipped with a powerful fitting engine based on differential evolution algorithm, a global optimization algorithm aiming to find solutions of global minimum. The software is well written with highly modularizable functionalities, which can be extended and/or customized easily. For example, I made some contributions to this software during my PhD years, when I parallelized the code to allow for running a model on a supercomputer cluster systems thanks to the mpi4py Python package. GenX is fit for users of different levels. For beginners, you can always follow the logics while playing with the GUI widgets; for advanced users, you can customize the functionality accordingly to live up to your full potentials to accomplish different purpose. The transition from beginner to advanced user is a matter of time, which should be relatively short for a heavy user.

I have been a heavy user of GenX until the end of my first postdoc position in HZDR in Dresden, Germany. Five CTR papers (one is under second run of review) have been published, where I used GenX for CTR modeling work. Since I started my postdoc position in Kiel, I have been getting involved in several projects, which required a wealth of scripting work. Not long ago, I started with GUI programming using PyQt5. Three GUI applications (CTR, XRV, PXRD, all shipped with DaFy) have been developed for processing synchrotron data. Having accumulated some GUI programing experience, I was amazed by the fact that how much better you feel when you are using a program with nice GUI compared to working with pure code files. GUI programming with pyqt5 is much simpler than I thought, and the signal/slot protocol employed in PyQt5 is not only simple but also efficient for signal transfer among GUI widgets compared to “callback” mechanism used in wxPython. I don’t intent to make detailed comparisons between PyQt5 and wxPython in terms of the performance for GUI programing, but there is a nice article if you want to learn more (<https://opensource.com/article/17/4/pyqt-versus-wxpython>). As said in the article, both have pros and cons, but I myself prefer the advantages PyQt5 has to those coming with wxPython. I can endure the disadvantages of PyQt5 without pain.

I realized the speed issue of wxPython-based GUI program when I was using GenX. That’s also the reason why I developed the mpi code to speed up the model running with GenX. Except for the speed, I do like all the concepts of GenX software design and also the powerful fitting engine. Realizing these, a glimpse of crazy idea came to my mind one day, “How about equipping GenX’s powerful heart with a PyQt5-GUI shell to improve the script performance?”. Since the fit engine is already hard coded in GenX, I only need to pull the program apart into pieces of modules, and re-assemble them into the new GUI shell. After around 3 month coding work, I made a brand new SuPerRod application with a PyQt5-based GUI, where all nice concepts found in GenX are maintained. More than just making a new skin, I optimized some central modules from bottom in SuPerRod, and some new features are implemented during the software development. Upon comparing the performance between SuPerRod and GenX, I have a strong feeling that SuPerRod runs much faster than GenX(more than 10 times faster). In a word, SuPerRod consists of GenX’s central fitting codes and a new GUI based on PyQt5.

# Glimpse of SuPerRod software

The purpose of SuPerRod may be misunderstood by its name, which sounds like a tool to fit rod data. In reality, SuPerRod provides a platform to solve all general scientific problems, where model fitting is required to get the answers. SuPerRod is like a powerful car, which runs on different roads. As mentioned above, SuPerRod is shipped with the powerful fitting engine taken from GenX. The differential evolution (DE) fitting algorithm will be discussed later. DE algorithm can be used to achieve solutions of global minimum, and it also features an intrinsic metadata structure that can be easily employed in a parallelized computation to improve the fit performance. While SuPerRod can be used as a general fitting tool, it does provide a fully functionable API that is dedicated to resolve CTR modeling, e.g structure factor calculation, dealing with water layering structure, consideration of surface roughness, just name a few. If you are fitting CTR data, numerous functions/modules are already available to help you set up your model in no time. If you would like to fit other data, and you have implemented the calculation of the variable you need to fit, then it is trivial to set up your model in SuPerRod (refer to later section for details). If you start your project from scratch, you won’t miss all the fun parts to crank your codes, since Python is the easiest, if not the best, programing language in the world. At last, enjoy your SuPerRod trip. Be bold, and you will get there soon.

# Benchmark features of SuPerRod

Not just having a new GUI-shell, SuPerRod comes with some new benchmark features that do not belong to GenX. First, SuPerRod uses a thread program to separate the manipulating GUI widgets from the fitting processes. As a result, the main GUI won’t freeze during fitting process, since the GUI widgets reside in a separated thread. In addition, the burden of the fitting process (most computation expensive) is distributed to different processers in a multiple-core architecture CPU system, which is usually equipped in modern PC’s. In addition, you can modify your script easily by pushing some buttons rather than making edits directly in the script editor panel. The idea is you predefine different sorbates, which are wrapped and collected inside the sorbate module. Then by tagging through the standard script with unique name-tag, which you use to locate the corresponding code block, you can locate and replace the code block in a way defined by the given sorbate tag. Speaking of model script, the figure of merit function is structured to return a tuple, that contains a simulated result and a penalty factor. You are free to define the penalty factor as a constraint for the fitting process. Constraining your fit is a very robust way to avoid overfitting. Having said that, let me say something about fit parameters. Clinging to the concept of user-friendly concept, I implemented in SuPerRod auto-filling fit parameters in the parameter panel. Now setting up a model in SuPerRos is really a few easy steps away: loading data, modifying script, and defining parameters, which are (semi)-automatic process. Last but not least, a Python terminal widget is imbedded to allow for exploration of modules, variables and many others on your model. It is a nice/easy way for either debugging or testing new feature on SuPerRod. All these features will be detailed in later sections.

# Targeted users of SuPerRod

SuPerRod targets users, who are working on a complicated fitting problem, which means large number of fit parameters (let’s say >8). In reality, the heavier your duty is, the better performance we can achieved with SuPerRod. It is not recommended to use SuPerRod to fit simple problem (although it can), e.g Gaussian peak fitting, since you can accomplish the fit with a much lighter code, e.g scipy.curve\_fit, lmfit, etc. The user should have some basic knowledge of Python scripting language. Some site-packages, like numpy, pandas, scipy, are heavily used in SuPerRod, so it is recommended to master these packages.

# Installation/setup guide

Install Python 3 (at least>3.5, 3.7 is the best) first.

Install following Site-packages:

PyQt5: GUI package

pyqtgraph: GUI package for graphing

qdarkstyle: GUI decoration skin

numpy

matplotlib

scipy

pandas

I didn’t check the compatibility of these packages of different versions, but it should be safer to install the newest version to avoid compatibility issue. One combination that is working on win32 system may fail on Linux system. You need to try this out by yourself.

Location of DaFy/SuPerRod package: <https://github.com/jackey-qiu/DaFy>. You can run ‘git clone <https://github.com/jackey-qiu/DaFy>’ in your terminal or download the package directly. You don’t need further installation of DaFy. After you unzip the DaFy package, in a terminal you change directory to superrod folder (DaFy/projects/superrod), and run “python SuPerRod\_GUI.py”. Then you should be able to see the main GUI pop out like below. Read more below to learn further steps after this.

A screenshot of a computer screen

Description automatically generated

Figure 1. Main GUI frame popping out upon launching SuPerRod program.

# Operation guide

In this section, the functions of different components will be illustrated, including widgets, modules (data, script and parameter instance). Understanding these components will help you understand the underlying logic of working with SuPerRod.

## Introduction of widgets

A screen shot of a computer

Description automatically generated

Figure 1. Layout of main GUI frame, three main widget frames, including Dataframe, Graphicframe and Codeframe, are labelled along with the associated constituent tab-widgets.

The main GUI frame consist of three tab-widget frame, including Dataframe, Graphicfram and Codeframe, as illustrated above. You can hide or show these frame by clicking the associated menu item in View tab. On the tool bar, you can find tool icons for, from left to right,

* **initializing a new model**: this will empty all the metadata holding by the current model, and a dialog will be prompted to ask for saving current model or not.
* **opening a saved model:** this will load all metadata saved in a model file (\*.rod). A model fit is a zip file, which archive data, script and parameter instance using pickle dump.
* **simulating a model:** script will be compiled, and the fit variable will be calculated. The fit results will be plotted in the Graphicframe. The model structure will also be updated.
* **running a model**: you launch a model run. Figures in Graphicframe will be updated lively (every 3 secs), so do the parameter values in the Parameter panel.
* **stopping a model run**: you stop a running model. The looping will be killed, and the associated timer will be stopped. The control is given back to user.
* **calculating the errors**: after you finish the model run, you press this button to calculate the errors for each fit parameter. Note the error bar values are only estimated from all intermediate fit results from all fit generations, and the error may not accurately represent the statistic errors. If you want to get statistical errors of each fit parameter, you can run a further NLLS fit using the best fit parameters, which is not implemented in SuPerRod.

To follow up, functions of widgets on each frame will be illustrated in detail.

### Data Frame

A screen shot of a computer

Description automatically generated ![A screen shot of a computer

Description automatically generated]()

A screenshot of a computer screen

Description automatically generated A screen shot of a monitor

Description automatically generated

Figure 2. Layout of each constituent tabWidget in the DataFrame, including Data widget(top left), Viewer widget (top right), Solver widget (bottom left) and Grid widget (bottom right).

Dataframe contains one tabWidget, which consist of a Data widget, a Viewer widget, a Solver widget and a Grid widget.

* **Data widget:** a table widget of 5 columns, most are self-explainable. DataID assigns the id of each dataset taking the naming rule of form “Data-HKL”. fmt column by default are occupied by string of “sym:6bw;l:3r”, which can be changed to the plotting style you want. The first part of the string before “;” specifies the symbol size (6 here) and symbol filled color (“w”) and symbol edge color (“b”). The second part of the string specify the line color (‘r’) and line width (3) for simulated profiles. Change these string tags accordingly. Acceptable color tags are: “r”, “b”, “k”, “w”, “y”, “m”, “g”.
* **View widget:** a table widget to show the selected dataset, including 8 columns. X column is L column for CTR data, but it can represent other physical variables for other data type. Y column is the intensity column for CTR data. Error column displays the error values. dL and BL column are used for calculating Robinson’s roughness factor. BL is the first Bragg peak L values in a rod, while dL is the L spacing between two adjacent Bragg peaks in a rod. The last column is mask column, which takes bool values (True, or False). A mask value of True means the associated data point will be displayed and used for fitting, while a data point with a False mask value will be masked out for plotting and fitting. But note that masking a data point will not delete the data point.
* **Solver widget:** this is a parameter tree widget to specify the intrinsic parameters for performing differential evolution fitting. Refer to last part of this section to learn of the meaning of k\_r, k\_m, and methods. (1) Figure of merit function is the cost function you use for model optimization. To fit ctr data, we most often use chi2bar cost function. It is also common to use diff and log cost functions for problems of general purpose. Refer the DaFy/util/fom\_funcs.py to learn more details about the definition of different cost functions. (2) Error bar level (>1) is used to estimate the error bars for fit parameters. (3) “Auto save, interval” define the frequency of save model during model run. (4) “Save evals, buffer” defines the maximum generations allowed. It is a huge number by default. (5) Under Fitting branch, you will see a check box of “start guess”, which toggle the two different ways to initialize the values of fit parameter in the first fit generation, either based on random generation (unchecked) or taking current values in the parameter panel. For a new model, we should always uncheck this button, while for a half-done model it may be wise to check it. In addition, (6)“Use(Max,Min)” checkbox should be always checked to tell the fit algorithm search for values within the boundary you define in the parameter panel. It is always better to set boundary for each fit parameter. (7) Population size and generation size define the size of model refinement (refer to last part of this section). Model runs with larger population size converge more slowly but could essentially avoid finding local minimum. Rule of thumb is the population size should be at least 8 times higher than the total number of independent fit parameters. You can set an arbitrary large generation size, and stop the model run whenever you see FOM profile has converged to a flat line.
* **Grid widget:** this is a table widget of 6 columns, where you define all fit parameters in terms of set function names, boundaries, fit or freeze check. In the parameter column, we provide the set function names for all fit parameters. For example, a function name of rgh.SetA1 will mean a parameter instance rgh containing an attribute A1 to be set by function rgh.SetA1. Therefore, remember the first column is not the name of a fit parameter but the name of a set function for setting the value of the associated parameter. In value column, the current best fit parameter values will be displayed in this column. When setting your model, ensure that the value of a parameter should be within the boundary you set for the parameter. The last column is the error column, which will be filled after the model run is finished. Don’t forget the push the button (a calculator-like push button found on the tool bar) to calculate the error values.

### Graph Frame

### Code Frame

## Data format

## Script module

## Fit parameters

## Logic flow map

A screenshot of a cell phone

Description automatically generated

# Setup a model from scratch

# Two examples

## Multi-Gaussian peak fitting problem

## CO2RR on Cu(100) surface

# Introduction of differential evolution

A screen shot of a computer

Description automatically generated