

Manual of SuPerRod_v2.0 software

Canrong(Jackey) Qiu, kiel University, Kiel, Germany
crqiu2@gmail.com

“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.	<i>Motivation of software development</i>	2
2.	<i>Glimpse of SuPerRod software</i>	3
3.	<i>Benchmark features of SuPerRod</i>	3
4.	<i>Targeted users of SuPerRod</i>	4
5.	<i>Installation/setup guide</i>	4
6.	<i>Operation guide</i>	5
(1)	Introduction of widgets	6
1)	Data Frame	8
2)	Graphic Frame	10
3)	Code Frame	12
(2)	Data format	12
(3)	Script module	13
(4)	Fit parameters	14
(5)	Logic flow map	14
7.	<i>Setup a model from scratch</i>	15
8.	<i>Four examples</i>	15
(1)	Multi-Gaussian peak fitting problem	16
(2)	CO₂RR on Cu(100) surface	16
(4)	Zr nanoparticle on muscovite surface	16
9.	<i>Introduction of differential evolution</i>	16
10.	<i>How to formulate your model script?</i>	17
11.	<i>API functions for CTR structure factor calculation</i>	18

1. 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 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 programming 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 programming, 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.

2. 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.

3. 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 processors 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 SuPerRod 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.

4. 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.

5. Installation/setup guide

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

Install following Python Site-packages:

PyQt5: GUI package

pyqtgraph: GUI package for graphing

qdarkstyle: GUI decoration tool

pyopengl:3D molecular rendering engine

numpy

matplotlib

scipy

pandas

nexusformat

PyMca5(warning: Seems it is incompatible with Python 3.8)

wxPython (required due to some legacy import statements in GenX modules)

This may not be a completed dependent Python packages. If some required package is missing, you should just install it as well.

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 decompress 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.

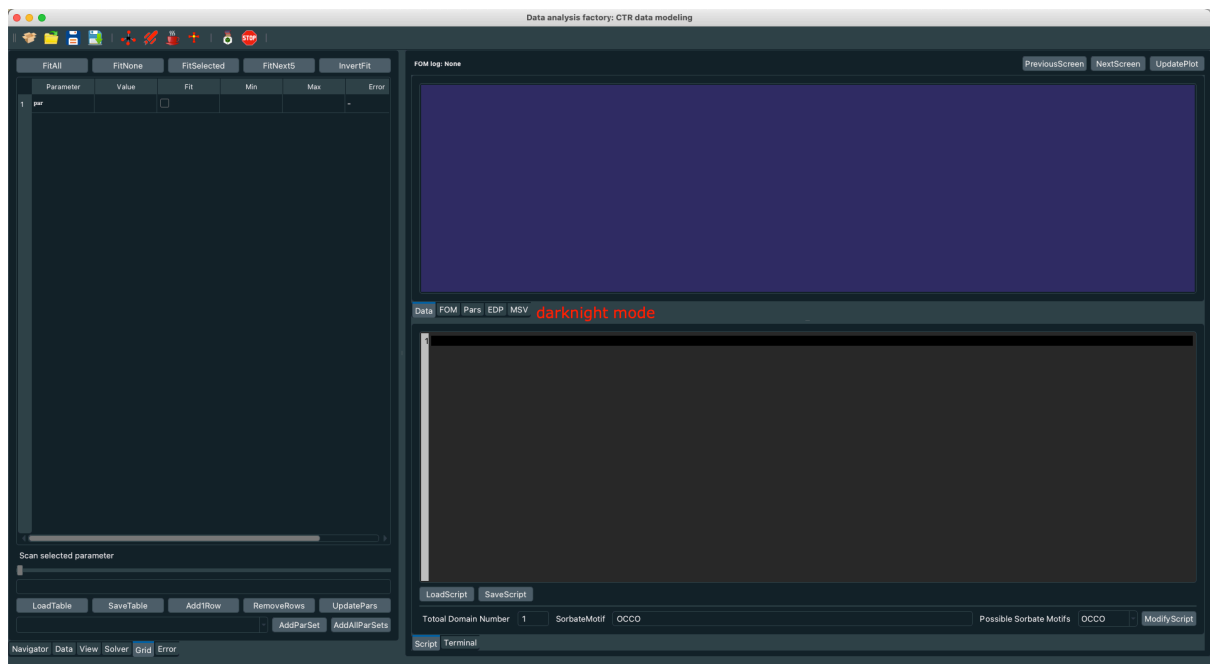
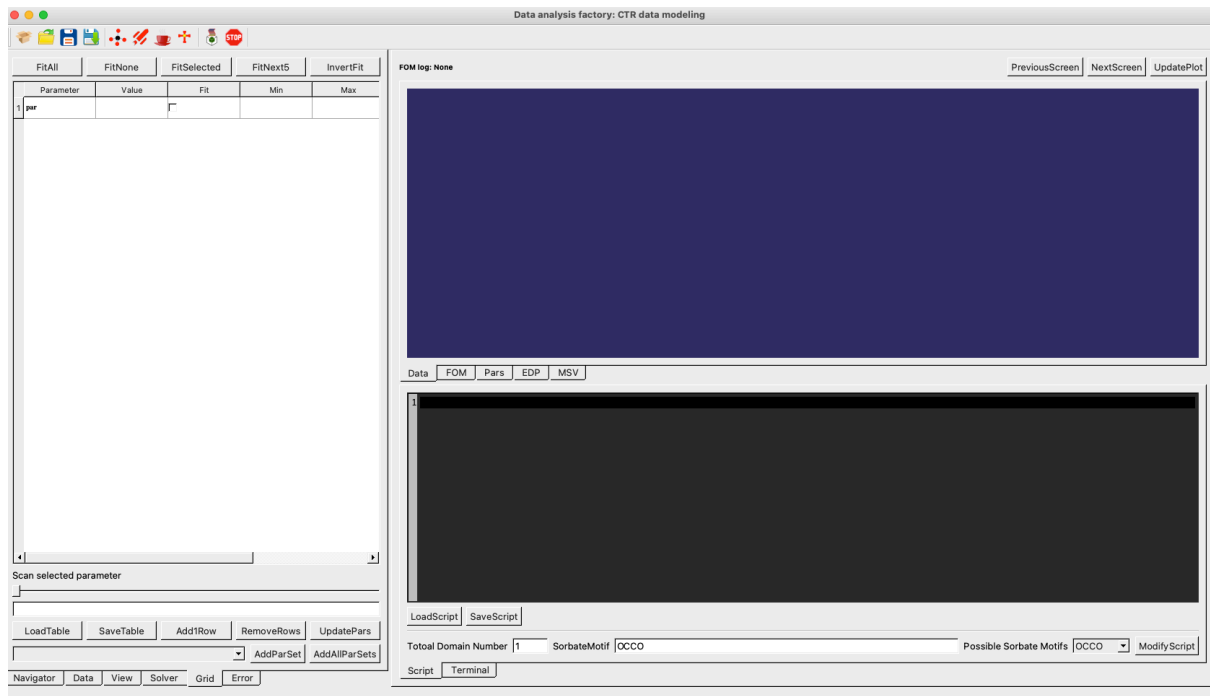


Figure 1. Main GUI frame popping out upon launching SuPerRod program (top: daylight mode, applied during 8am-5pm; bottom: dark-night mode, applied after 5pm).

6. 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.

(1) Introduction of widgets

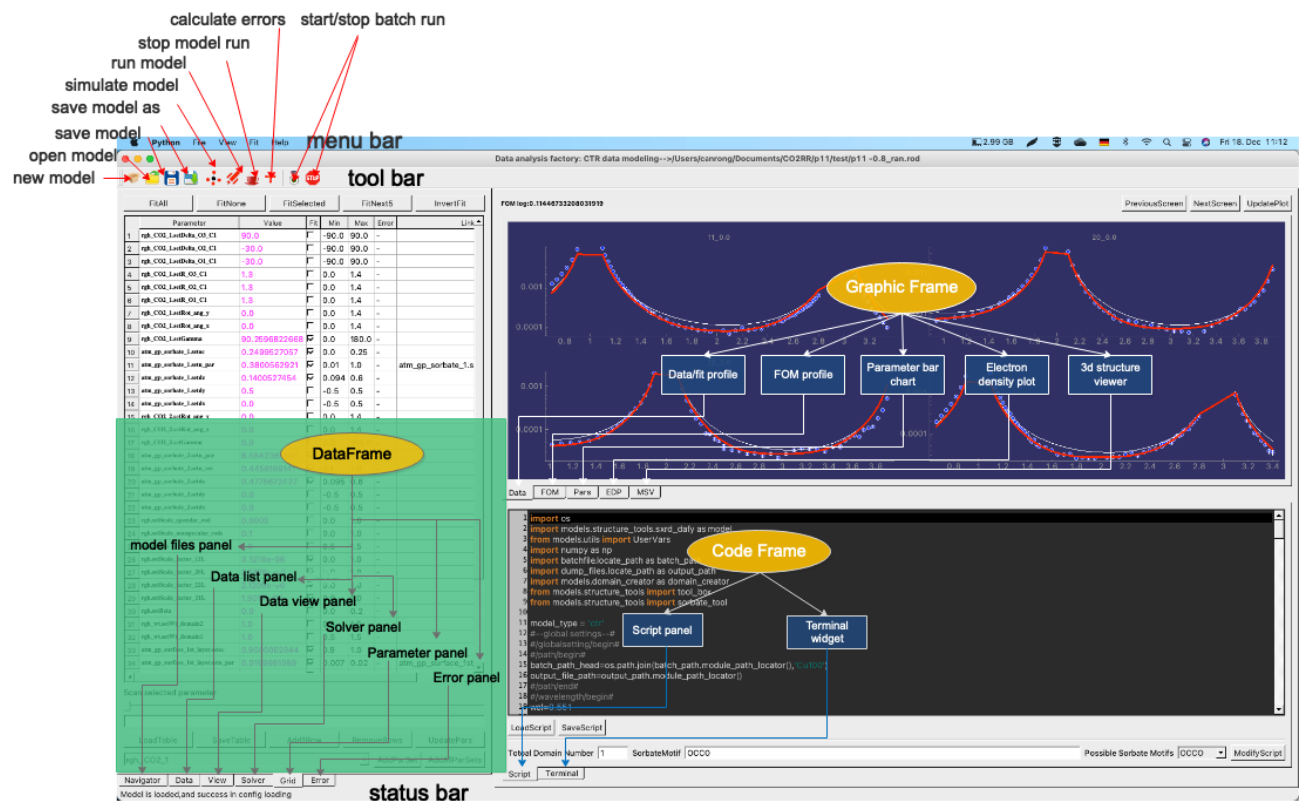


Figure 1. Layout of main GUI frame, three main widget frames, including Dataframe, Graphicframe and Codeframe, are labelled along with the associated constituent tab-widgets.

The main GUI frame consist of three tab-widget frames, including Dataframe, Graphicframe and Codeframe, as illustrated above. You can hide or show these frames 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.
- **save model:** this will save the changes in the opened model with a same model file name (*.rod). A model fit is a zip file, which archive data, script and parameter instance using pickle dump.
- **save model as:** this will save the changes in the opened model with a different model file name (*.rod). A navigator dialog will pop up for locating the file.
- **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. However, the error bar values should give you a good sense of how sensitive the model fit is dependent on each fit parameter. Higher error bar means low sensitivity, and versa vice.

- **Start/stop batch run:** With SuPerRod, it is possible to run your model in a batch mode. The batch run can be started with two different modes. Either run on a single model file, where the fit parameter will be rolling stepwise in each batch task. It is reserved to run a model-independent RAXS model, where each RAXS spectrum is only dependent on five parameters. In this mode, the used data and fitted parameters (5 in one set) will move forward accordingly in the batch process. The other mode is a real batch mode, where you load in a bunch of model files from the model files panel. These model files will be run one after another until the optimization of every loaded model is finished.

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

1) Data Frame

Data widget (top left): A table with columns: DataID, logY, Use, Errors, and fmt. It lists 22 datasets. A yellow box highlights the following instructions:

- DataID:** generated automatically
- logY:** show y axis in log scale
- Use:** use this dataset for fitting
- Errors:** show error bars or not
- fmt:** string for plotting style

Viewer widget (top right): A table titled 'A list of datasets' showing data points for 'Data-001_2'. A yellow box highlights the following instruction:

Last column is the mask column, you can set the cell values to False. Then push this button to update the data profile view. The masked data point is not deleted, but labeled as ones that are not used for plotting and fitting.

Solver widget (bottom left): A configuration panel for the 'Diff.Ev' algorithm. It includes settings for 'k_m', 'k_r', 'Method', 'FOM' (Figure of merit, Error bar level, Auto save interval, save evals, buffering, weighting factor, weighting region), and 'Fitting' (start guess, Use (Max, Min), Population size, Generation size).

Grid widget (bottom right): A table showing fit results for various parameters. A yellow box highlights the following instruction:

Parameters for differential evolution algorithm. Refer to documentation to learn more.

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).

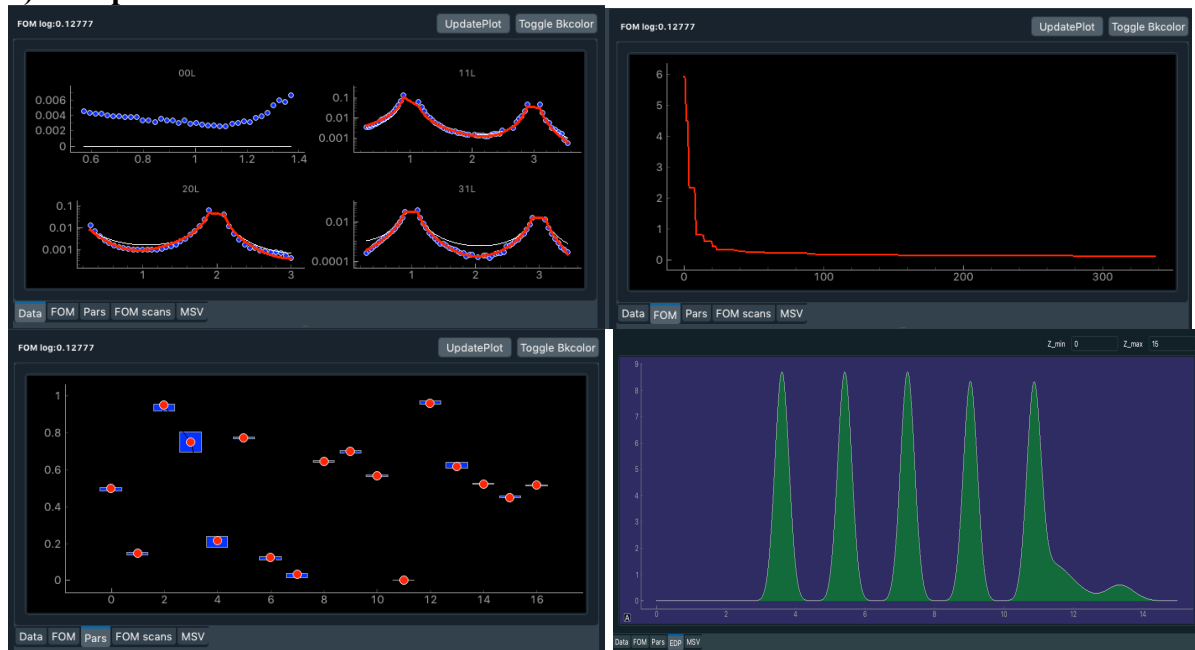
Dataframe contains one tabWidget, which consist of a Navigator widget, Data widget, a Viewer widget, a Solver widget and a Grid widget.

- Navigator widget:** a container, where you can load all model (*.rod) files in a folder. Double click of the model will launch this model file. If you want to push them to a queue, then click 'Hook to queue', which is a 'must do' step to start a batch run. If you want to remove these models from queue, then click 'purge from queue'.

- **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) weighting factor/weighting region, define the region where you would like to weight the FOM. You may want to set a higher weighting factor at some anti-Bragg peak region for better fit performance. In the weighting region, this is a dictionary with the keys of (H, K) and values of a list of L segment to be considered for weighting. (6) 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, (7) “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. (8) 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 5 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 7 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 6th column is the error column, which will be filled after the model run is finished. Don't forget the push the button (a cross-like push button found on the tool bar) to calculate the error values. The last column is the linked functions, which can be left blank. This linked function will be executed the same time as the set function in the first column. It is a nice quick way if you want to group two fit parameters together during fit. For example, if you put `rgl.setA2` in the last column in the row of `rgl.setA1`, it literally means to group A1 and A2 together to have identical values during model fit.

2) Graphic Frame



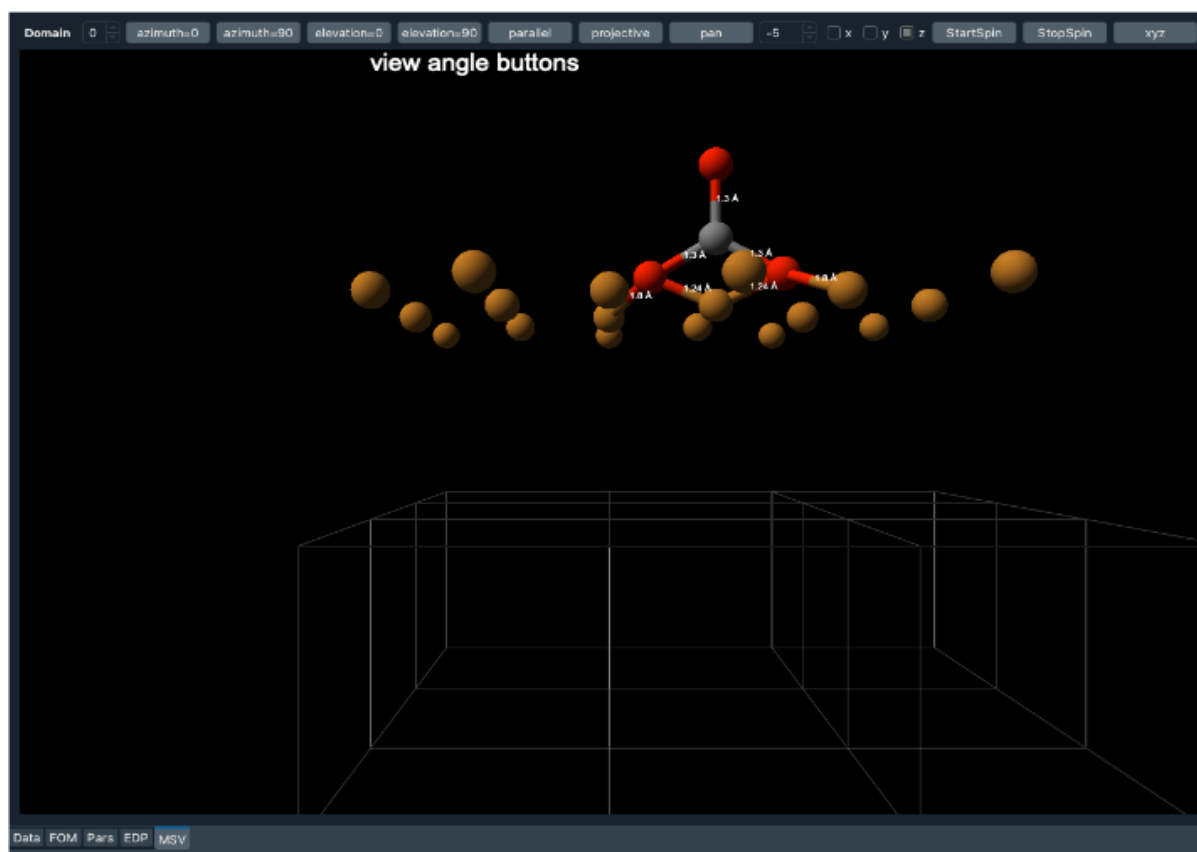


Figure 3. Layout of each constituent tabWidget in the GraphicFrame, including Data widget(top left), FOM widget (top right), Pars widget (middle left), EDP widget (middle right) and MSV widget (bottom).

The Graphic Frame contains a tab Widget, which consist of Data widget, FOM widget, Pars widget, FOM scan widget (not implemented yet) and MSV widget.

- **Data widget:** display the data profiles with filled circle symbols, simulated profiles in solid red lines (color can be changed), and calculated profiles for unrelaxed structure (for CTR data only). The figure layout will be expanded accordingly when more datasets are loaded.
- **FOM widget:** display the evolution of FOM (figure of merit) values during model running. It is a nice way to tell whether or not the model optimization is converged. You will see a more or less straight line spanning most part of the profile if the model fitting has converged.
- **Pars widget:** display the current best fit values of fit parameters in red filled circles. The values are normalized to range from 0 to 1 using the boundary of each parameter. Besides that, the bar chars illustrate the searching range at the current generation. An aggressive searching corresponds to a long searching bar. When a model is converged, you will most likely see all searching bars converged to a narrow line. This means you cannot improve your fit anymore, and you can stop the model run now.
- **MSV widget:** visualize the molecular structure on the side view (left) and top view (right). You can change the view angle by pushing those buttons on the widget panel. If you have defined several domains, you can also switch to a different domain by tweaking the spin box. The molecular structure in side view does not show chemical bonds, which give you an overview of how sorbate atoms are registering at substrate surface. The structure on top view only display the atoms on the top substrate layer

and one sorbate molecule, and the chemical bonds are displayed in the sorbate molecule. From sideview structure, you can quickly tell the geometrical relationship between the sorbate and the substrate surface.

- **EDP widget:** draw the electron density profile of the system you are fitting. The substrate surface is not necessarily at $z=0$, it is determined by the atom heights in the associated surface file and the offset height you used in the script. Only total electron density profile is displayed. If you need a fancier EDP, where you show EDP from different contributions, such as substrate, sorbate, water and even Fourier Synthesis from RAXR model independent model results, you need to implement your script for this purpose. Contact me for a solution.

3) Code Frame

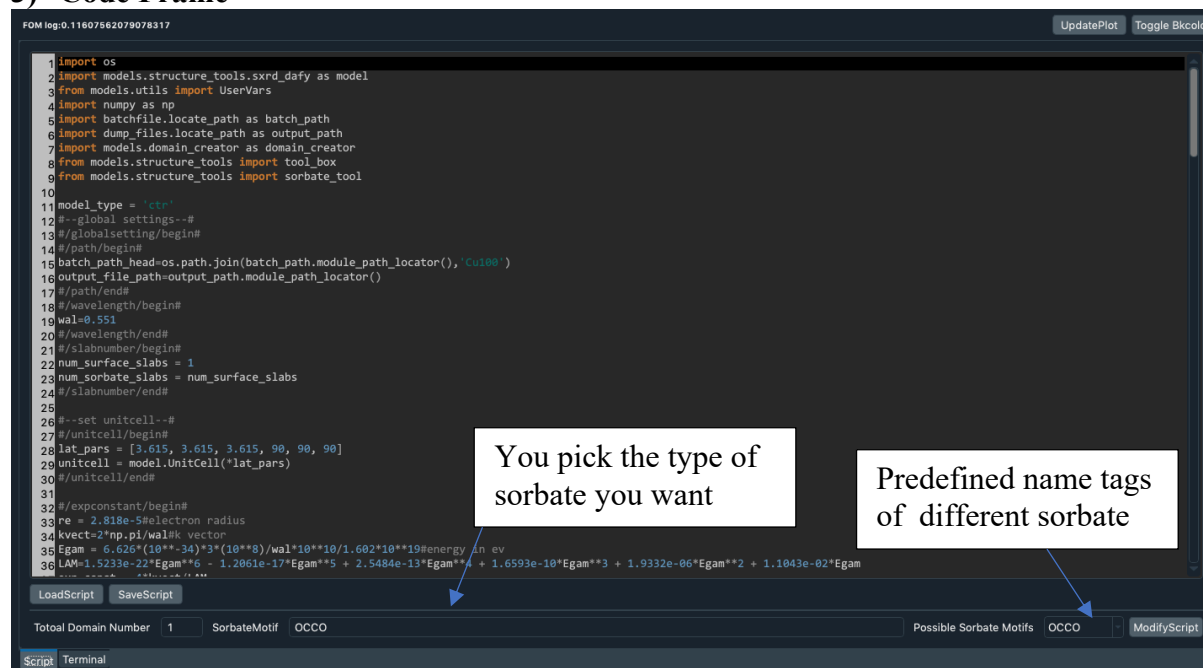


Figure 4. Layout of each constituent tabWidget in code frame.

Code Frame is the place where you display, modify your script. The main widget here is a plaintext editor, which has been styled to highlight Python key words. To use the `modifyscript` button, you need to define sorbate types, which are stored in a script module to be loaded in main GUI name space. In addition, you should also define name tags in the standard script file, so that the script block for defining sorbate can be located. It is a handy feature, but you don't have to use it if you don't know how to get around with the coding trick. The functionality of choosing predefined sorbate type is now only implemented in the model related to CO₂ reduction on Cu(100) surface. In addition, it should be noted that you cannot mix-use tab-space and normal space in the editor, otherwise error will be prompted. Therefore, you should always use normal space for indentation.

The script will be compiled upon model simulation through "`exec script_string`". After compiling, the variables and functions will be stored in the name space of `model.script_module`, which is defined in the main GUI frame class. I will elaborate below the guides to setup the script for model fitting.

(2) Data format

Data loader loads data file of contents in a special format. Data file is a normal ascii file, which must contain 8 columns of space separated numeric values. When the data file is loaded, the data loader will split the data values into multiple date sets according to the values

of second and third columns. The rows with same values in both 2nd and 3rd columns will be grouped into one dataset. For example, rows with the 2nd column values of 0 and 3rd column values of 0 will be assigned to a separated data instance, which will be named as Dataset_00L. For each dataset, a special name will be assigned to each column in the associated data instance. From column 1 to column 8, the corresponding names are 'x', 'h', 'k', 'Y', 'y', 'error', 'BL' and 'dL'. While 'x', 'y', 'error' are direct attributes of the data instance, the others are stored in the extra_data attribute, which is a normal Python dictionary. For example, if we define a data instance with a name of *data*, one-to-one corresponding match between the original column in the data file and the name space defined in *data* instance is as follows:

1st column \leftrightarrow *data.x*,
 2nd column \leftrightarrow *data.extra_data['h']*,
 3rd column \leftrightarrow *data.extra_data['k']*,
 4th column \leftrightarrow *data.extra_data['Y']*,
 5th column \leftrightarrow *data.y*,
 6th column \leftrightarrow *data.error*,
 7th column \leftrightarrow *data.BL*,
 8th column \leftrightarrow *data.dL*.

When dealing with **CTR/RAXR data**, you stick to the data format as follows:

- 1) For CTR data, x column is L column, Y column is filled with 0
- 2) For RAXR data, x column is energy column, Y column is L column
- 3) h, k columns are holding H, K values
- 4) y column is holding background-subtracted intensity of CTR signal
- 5) BL and dL are two values for roughness calculation, of which BL represents smallest Bragg peak L along one rod, dL represents L interval spacing between two adjacent Bragg peak.

If the data you want to load is **not in CTR format**, to make successful loading, assure:

- 1) your data file has 8 columns
- 2) columns are space-separated (or tab-separated)
- 3) you can add comment lines heading with "#"
- 4) if your data has <8 columns, then fill the unused columns with 0
- 5) to assess your data column, you should use the naming rule described above,

although the real meaning of each column, eg x column, could be arbitrary depending on your data. For example, if you put frequency values to the first column(x column), then to access this column, you use *data.x* to achieve the frequency values.

Data file of 8 columns should be enough to cover many different data format situations. You can freely format your datafile, as long as you remember which column corresponds to what and how to extract them from a data instance. If you fill, for example, frequency data in first column, then to extract them using *data.x*. If you do it differently by filling the frequency data in 8th column, then to extract them using *data.extra_data['dL']*, which has no literal connection with the name "dL" to the meaning of your values (frequency). If you know the naming rule for a data instance, you should know how to extract the data column you would like to use in your model script.

(3) Script module

Script module (named *script_module*) is a module attribute of the *model* instance, which is defined in the main GUI object. *script_module* holds a name space, which include all object variables defined in your script. Besides that, it also holds the data list (named *data*), which is a list dataset instances. For example, you can achieve the first dataset instance using *model.script_module.data[0]*. Then obviously, *model.script_module.data[0].x* will return the

first column of this dataset. As a user, you don't need to manipulate script module for your modeling, but it can help you understand the logic flow map shown below.

(4) Fit parameters

Defining fit parameters is an indispensable part of setting up a model. It is extremely flexible and convenient to define a fit parameter in SuPerRod. All you need to do is implementing the associated set function for one parameter attribute residing in different object. In other words, the parameter attributes can live in different objects. Let's take one class instance objects (*obj1*) as an example here. I define and initialize an attribute named *attr1* in this object through:

```
self.attr1 = 1
```

At the same time, I also define the associated set function for this attribute.

```
def setattr1(self, val):  
    self.attr1 = val
```

With these steps, you have already defined a fit parameter and its set function. In the parameter table (Figure 2, bottom right), you can filled one row of the first column with the set function name, i.e *obj1.setattr1*, then the program knows you want to fit the attribute *attr1* living in the name scope of *obj1*. Besides this way, there is a built-in class object (*UserVars*) you can import to quickly setup a fit parameter. In your script, you can simply import it through *from models.utils import UserVars*. Then the same work shown above can be done in a much more concise and easier way:

```
obj1 = UserVars()  
obj1.new_var("attr1", 1)
```

In summary, you can create your fit parameters through defining the attribute of any class object along with its set function or using built-in class object (*UserVars*). Both ways could be helpful in different situations.

(5) Logic flow map

With the knowledge of script module, data format and fit parameter, let's put them together in one snapshot to understand the underlying logic flowing path while the program is running. As shown in Figure 5, SuPerRod is highly modularized with different objects living inside different name space. Data exchange between different object makes all the logic flowing pattern shown in Figure 5. The main GUI object is the place you enter when you launch this program. The main GUI object holds the GUI widgets, which live in one of the three qt frames: Dataframe, Codeframe and Graphicframe. In addition, the main GUI object define a *model* attribute, which stores information of data, parameter, script and model fit configurations. We have discussed all these components above, so refer to the associated section if anything is unclear to you. To accomplish the fitting, *run_fit* is defined as well in the main GUI object. In *run_fit*, there is a solver attribute that connect to the defined model attribute. The fit looping is triggered by a switch (run button widget on the tool bar). Once the fit looping starts, a new thread is open for the optimization, where the calculation task will be chopped into sub-tasks being completed in parallel by different processors if available. The optimization evolves through many generations. After each generation, a pyqt signal will be emitted to be caught by main frame widgets. To respond to the signal, the fit parameter values will be updated. To avoid potentially long overhead time caused by 3D structure visualization, a timer is created to connect the signal, when it is emitted the first time. The connection is open only once, and then will be close. Once the timer starts, it will trigger the updating of molecular structure rendering every 3 seconds. The optimization looping will continue moving forward until receiving a stop run signal, which can be triggered by pressing the stop model button on the tool bar.

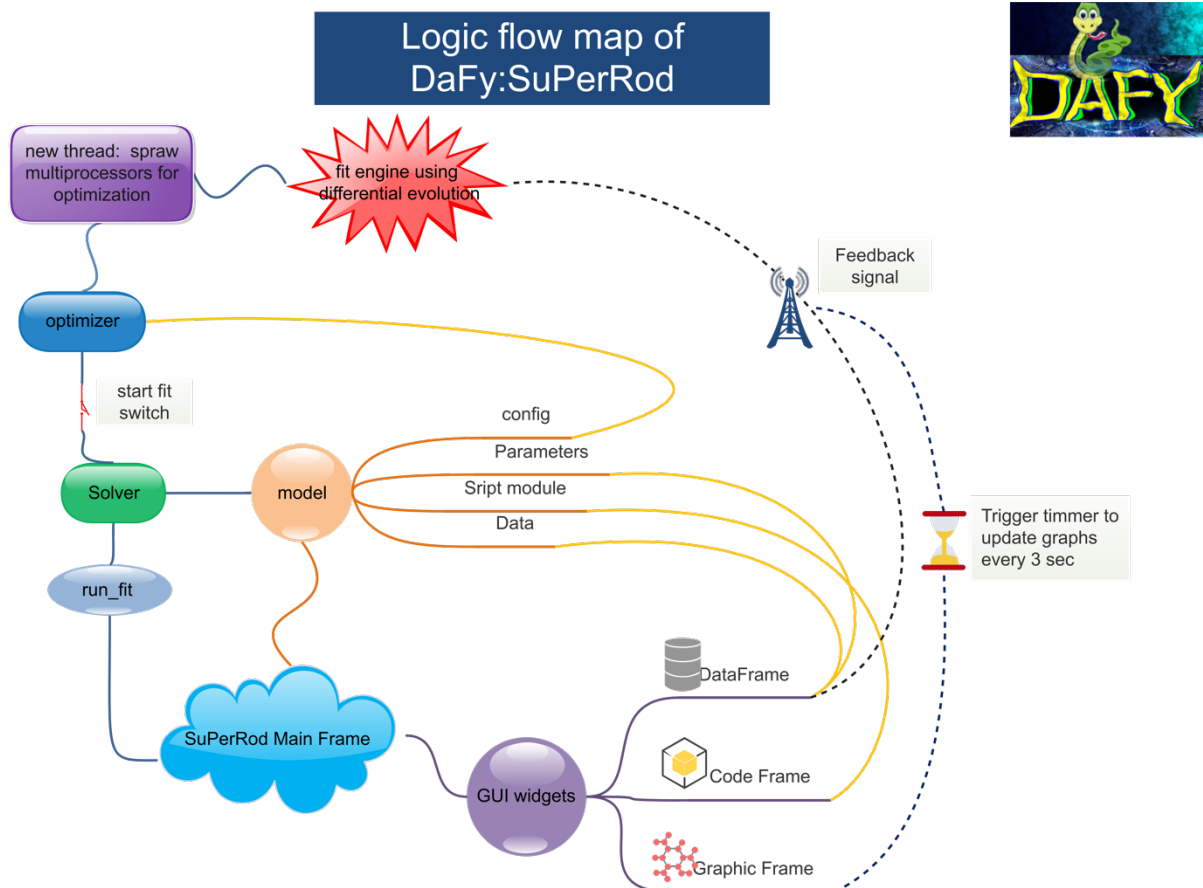


Figure 5. Logic flow map of SuPerRod. Solid lines indicate sharing name spacing between objects, dashed lines are signal transmitting path.

7. Setup a model from scratch

To set up a model, you need a data file of specific format (refer to Data format section before), model script, and fit parameters. You should define the fit parameters inside the model script. Play with those examples below to get yourself being more familiar with model setup. To run your model script well, you need two minimum components in your script. First is the definitions of fit parameters, and second is a function named *Sim* at the end of the script. This *Sim* function is mandatory. It takes the first argument of *data*, which is a list of dataset instance. The body of *Sim* function is the place where you implement the calculation of the *y* variable (eg. structure factor for CTR model) you want to fit. This function should return a three-member tuple. The 1st tuple member is a list of *y* variable values calculated from the model, 2nd tuple member is a penalty factor, which should be set to 1 if you don't have any penalty function of place a constraint. The last tuple member is a list of weighting factors scaled to the calculated FOM value for each dataset. You can assign different weighting factors for different datasets during fit. This way, the fit subroutine will bias towards making better fit to the dataset with larger weighting factor. You can simply set the scaling factor to 1 for each dataset to disable the weighting strategy.

8. Four examples

Three examples rod files are already created for you to play with. You can locate these files and load it using open model button.

(1) Multi-Gaussian peak fitting problem

The model file is located in DaFy/examples/others/four_gaus_peaks.rod The data is a simulated multi-Gaussian peak model. Four overlapping Gaussian peaks are summed up to produce the datafile. From this example, you will learn how to set up a model with a non-CTR dataset.

(2) CO2RR on Cu(100) surface

The model file is located in DaFy/examples/Cu100_CO2_EC/ctr_fit_mp6V_test.rod. This model deals with the adsorption of CO2 reduction product on Cu(100) surface under operando condition at reductive potential -0.6 V (w.r.t Ag/AgCl reference electrode). The data quality is not good enough to resolve the binding structure, but this example provide some insights into setting up CTR model.

(3) Zn(II) adsorption on hematite (1-102) surface

The model file is located in DaFy/examples/hematite_rcut_AD/ctr_example_Zn_7mm_cmp_hematite_substrate_ran_ran.rod. This model deals with the adsorption of Zn(II) adsorption on hematite (1-102) surface. From this example, you will learn how to constrain your model with bond valence rule.

(4) Zr nanoparticle on muscovite surface

The model file is located in DaFy/examples/muscovite001_AD/ctr_example_muscovite_ran.rod. This model deals with X-ray reflectivity data to solve the adsorption of ZrOx nanoparticle on muscovite(001) surface. In this example, you will learn how to fit CTR/RAXS data together.

Read more at <https://pubs.acs.org/doi/10.1021/acs.langmuir.8b02277>

9. Introduction of differential evolution

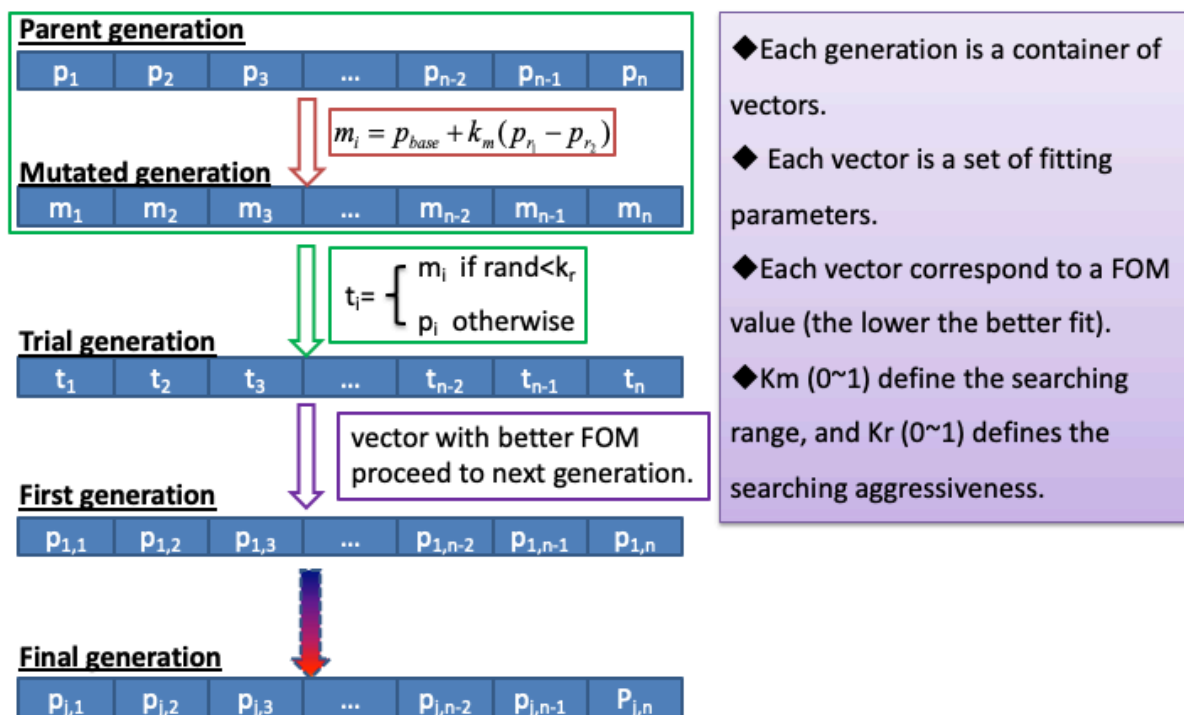


Figure 6. Differential evolution mechanism

The optimization algorithm used in SuPerRod is differential evolution, which is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. In this process, new generation with better fitting quality will be generated from the parent generation. Some basic concepts are listed here.

- Each generation is a container of vector, and each vector is a set of fit parameter values. Each vector corresponds to a FOM value, which determines the fitting quality. The population size is arbitrarily set.
- To start with, the parent generation is created on a random basis.
- Then each vector in the **parent** generation will be mutated by applying a mutation rule. The p_{base} here is the vector with best fit quality, K_m is a fit factor, ranging from 0 to 1, which determines the searching range. P_{r1} and P_{r2} are randomly selected from the parent generation.
- With this mutation step, we come up with the mutated generation. Now we apply a selection rule to each vector in the mutated generation to compute the trial generation. Here the K_r is the selection factor, ranging from 0 to 1, which defines the searching aggressiveness. $Rand$ is a random number from 0 to 1.
- As the last step, we simply compare the FOM value of each vector in parent generation to that in the trial generation. The vector with better FOM will proceed to the next generation.
- We repeat the mutation and trial method on the new generation to come to next generation. This process will be repeated again and again until we arrive at the final generation.

As you can see, the new generation always has a better FOM value than the old generation. That way, we get a better and better fit in differential evolution. This is basically how DE works.

10. How to formulate your model script?

This section is more advanced topic dealing with how to set up your model script. In the model script, you need to define your fit variables, and parameterize them in a mathematic formular. For CTR fitting, the mathematic formular is where you define the way to calculate structure factor. Basically, in the script you define numerous Python variables, which is pretty much of free-style without strict rules except for one. You are forced to define a function named **Sim**, and its first argument must be data without exception. Inside **Sim** function, you define how you calculate the model-dependent variable y (structure factor for CTR modeling). This subroutine can be defined in a separated Python module that should be imported in the script. In **Sim** function, you need to return a tuple of three members. 1st member is a list of calculated y values (length should matches that of datasets). 2nd member is a scaler that serves as a penalty factor (>1), which can be implemented in different ways (eg. Based on bond valence constrain). If you don't have any penalty function defined, just set this to 1. The 3rd member is a list of FOM scaler (same length as your datasets). Here, you can apply a different scaler to a specific dataset for some reason, this scaler will then be scaled to the FOM value for this dataset. If you want to use the same scaler for all datasets, then set this to a list of 1s. These are the minimum requirements to define a model script. We will talk about some API functions for CTR structure factor calculations, which are implemented in SuPerRod.

11.API functions for CTR structure factor calculation

To be added!