
SAXS_routines

Release v1.0.0+1.gb324d82.dirty

Kevin Pounot

May 03, 2023

CONTENTS

1	SAXS analysis routines	1
1.1	Installation:	1
1.2	Quick start	1
1.3	Support	3
1.4	Contributions	3
1.5	API reference	3
2	Indices and tables	15
	Python Module Index	17
	Index	19

SAXS ANALYSIS ROUTINES

The package contains Python-based analysis routines for small-angle X-ray scattering data.

The routines are initially intended to be used with SAXS data, where a liquid chromatography column was connected before the injection into the capillary, but can be used with standard measurement data as well.

1.1 Installation:

Simply use `python3 setup.py install`.

1.2 Quick start

The experimental data can be imported using the functions present in the *data_parsers* module. For instance, a size-exclusion chromatography and small-angle scattering (SEC-SANS) experiment file performed on the BM29 beamline at the ESRF can be imported using:

```
from saxs_routines.data_parsers.esrf_bm29 import read_HPLC

data = read_HPLC('my_data_file.h5', name='my_protein_name')
```

The function returns an instance of the *sample.Sample* class which is a subclass of the NumPy ndarray. Hence, various operations are available with the data (slicing, mean, addition, division, transpose, ...). Along with the intensities, the class *sample.Sample* stores various metadata such as errors, incoming beam intensity, beamline name, momentum transfer q-values or time (see documentation for details). An important feature of the class *sample.Sample* is that the error propagation is done automatically for most of the operators applied on the data. Also, the momentum transfer q-values and the elution time are automatically sliced with the data.

```
from saxs_routines.data_analysis import FindPeaks

peaks = FindPeaks(data)
peaks.run()

sample = peaks.get_sub_arrays()[0] # select the first peak found

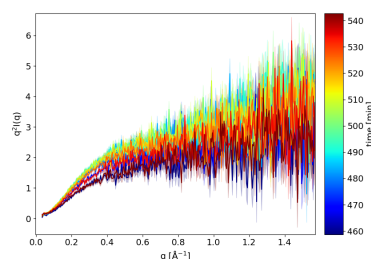
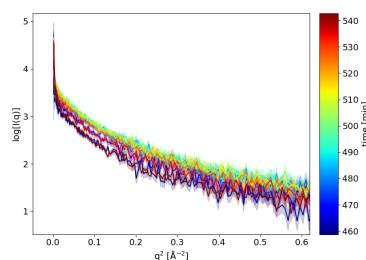
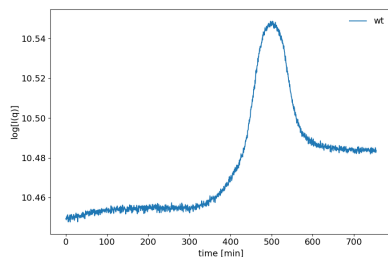
# select the buffer region (first 50 minutes) and perform the subtraction
buffer = data.get_time_range(0, 50)
sample = sample - 0.95 * buffer
```

The treated data can be saved in text format using the following:

The class `sample.Sample` also contains a method for quick plotting:

```
# for a log plot of the signal integrated over q
data.sum(1).plot('log')

# for a guinier and a kratky plot of the signal over time
data.plot('guinier', axis=1, max_lines=10, new_fig=True)
data.plot('kratky', axis=1, max_lines=10, new_fig=True)
```



To automatically find a peak in a SEC-SANS experiment and subsequently subtract a rescaled buffer signal, you can use:

```
sample.write_csv('output_file_name')
```

Additional data analysis routines will be found in `data_analysis` module.

User-defined model can also be constructed and fitted to the data. To this end, please refer to the documentation of the `model` module.

1.3 Support

In case of bugs or obvious change to be done in the code use GitHub Issues.

1.4 Contributions

See [contributing](#).

1.5 API reference

1.5.1 API reference

sample

Handle data associated with a sample.

class `sample.Sample(input_arr, **kwargs)`
Handle the measured data along with metadata.

This class is a subclass of the `numpy.ndarray` class with additional methods and attributes that are specific to small-angle X-ray scattering experiments on biological samples.

It can handle various operations such as addition and subtraction of sample data or numpy array, scaling by a scalar or an array, indexing, broadcasting, reshaping, binning, sliding average or data cleaning.

Parameters

- **input_arr** (*np.ndarray, list, tuple or scalar*) – Input array corresponding to sample scattering data.
- **kwargs** (*dict (optional)*) – Additional keyword arguments either for `np.asarray()` or for sample metadata. The metadata are:
 - **filename**, the name of the file used to extract the data.
 - **errors**, the errors associated with scattering data.
 - **time**, the experimental time.
 - **elution_volume**, if available, the elution volume of the HPLC.
 - **I0**, the fitted intensity at $q = 0$.
 - **I0_std**, the uncertainty on I0 value(s).
 - **rg**, the gyration radius.
 - **rg_std**, the uncertainty on Rg value(s).
 - **wavelength**, the wavelength of the X-ray beam.
 - **name**, the name for the sample.
 - **temperature**, the temperature(s) used experimentally.
 - **concentration**, the concentration of the sample.
 - **pressure**, the pressure used experimentally.

- **buffer**, a description of the buffer used experimentally.
- **q**, the values for the momentum transfer q .
- **detector**, the detector used.
- **beamline**, the name of the beamline used.
- **flow_rate**, the flow rate inside the capillary.
- **observable**, for 2D dataset the name of the attribute corresponding to the first axis.

Note: The **errors** metadata is special as it is updated for various operations that are performed on the data array such as indexing or for the use of universal functions. For instance, indexing of the data will be performed on **errors** as well if its shape is the same as for the data. Also, addition, subtraction and other universal functions will lead to automatic error propagation. Some other metadata might change as well, like **q**, but only for the use of methods specific of the *Sample* class and not for methods inherited from numpy.

Examples

A sample can be created using the following:

```
>>> s1 = Sample(  
...     np.arange(5),  
...     dtype='float32',  
...     errors=np.array([0.1, 0.2, 0.12, 0.14, 0.15])  
... )
```

```
>>> buffer = Sample(  
...     [0., 0.2, 0.4, 0.3, 0.1],  
...     dtype='float32',  
...     errors=np.array([0.1, 0.2, 0.05, 0.1, 0.2])  
... )
```

where *my_data*, *my_errors* and *q_values* are numpy arrays. A buffer subtraction can be performed using:

```
>>> s1 = s1 - buffer  
Sample([0. , 0.80000001, 1.60000002, 2.70000005, 3.9000001], dtype=float32)
```

where *buffer1* is another instance of *Sample*. The error propagation is automatically performed and the other attributes are taken from the first operand (here *s1*). Other operations such as scaling can be performed using:

```
>>> s1 = 0.8 * s1  
Sample([0. , 0.80000001, 1.60000002, 2.40000001, 3.20000005], dtype=float32)
```

You can transform another *Sample* instance into a column vector and look how broadcasting and error propagation work:

```
>>> s2 = Sample(  
...     np.arange(5, 10),  
...     dtype='float32',  
...     errors=np.array([0.1, 0.3, 0.05, 0.1, 0.2])  
... )  
>>> s2 = s2[:, np.newaxis]
```

(continues on next page)

(continued from previous page)

```
>>> res = s1 * s2
>>> res.errors
array([[0.5      , 1.00498756, 0.63245553, 0.76157731, 0.85      ],
       [0.6      , 1.23693169, 0.93722996, 1.23109707, 1.5      ],
       [0.7      , 1.40089257, 0.84593144, 0.99141313, 1.06887792],
       [0.8      , 1.60312195, 0.98061205, 1.15948264, 1.26491106],
       [0.9      , 1.81107703, 1.1516944 , 1.3955644 , 1.56923548]])
```

property T

The transposed array.

Same as `self.transpose()`.

Examples

```
>>> x = np.array([[1.,2.],[3.,4.]])
>>> x
array([[ 1.,  2.],
       [ 3.,  4.]])
>>> x.T
array([[ 1.,  3.],
       [ 2.,  4.]])
>>> x = np.array([1.,2.,3.,4.])
>>> x
array([ 1.,  2.,  3.,  4.])
>>> x.T
array([ 1.,  2.,  3.,  4.])
```

See also:

``transpose`_`

bin(*bin_size*, **metadata*, *axis=0*)

Bin data with the given bin size along specified axis.

Parameters

- **bin_size** (*int*) – The size of the bin (in number of data points).
- **metadata** (*strings*) – List of metadata names that should be binned as well.
- **axis** (*int*, *optional*) – The axis over which the binning is to be performed. (default, 0)

Returns **out_arr** – A binned instance of [Sample](#) with the same metadata except for **errors**, which are binned as well and possibly other user provided metadata names.

Return type [Sample](#)

get_q_range(*qmin*, *qmax*)

Helper function to select a specific momentum transfer range.

The function assumes that q values correspond to the last dimension of the data set.

Parameters

- **qmin** (*int*) – The minimum value for the momentum transfer q range.
- **qmax** (*int*) – The maximum value for the momentum transfer q range.

Returns out – A new instance of the class with the selected q range.

Return type *Sample*

get_time_range(*tmin*, *tmax*)

Helper function to select a specific time range.

The function assumes that time values correspond to the first dimension of the data set.

Parameters

- **tmin** (*int*) – The minimum value for time.
- **tmax** (*int*) – The maximum value for time.

Returns out – A new instance of the class with the selected time range.

Return type *Sample*

plot(*plot_type*='standard', *axis*=0, *xlabel*=None, *ylabel*='I(q)', *new_fig*=False, *max_lines*=10, *colormap*='jet')

Helper function for quick plotting.

Parameters

- **plot_type** (*str*) – Type of plot to be generated (for q on x-axis). Possible options are:
 - 'standard', I(q) vs. q
 - 'q2', I(q) vs. q^2
 - 'log', log[I(q)] vs. q
 - 'guinier', log[I(q)] vs. q^2
 - 'kratky', $q^2 \cdot I(q)$ vs. q
- **axis** (*int*) – The axis along which to plot the data. If *xlabel* is None, then for 1D data, 0 is assumed to be q values, and for 2D data, 0 is assumed to correspond to time and 1 to q values.
- **xlabel** (*str*) – The label for the x-axis. (default None)
- **ylabel** (*str*) – The label for the y-axis. (default 'log(I(q))')
- **new_fig** (*bool*) – If true, create a new figure instead of plotting on the existing one.
- **max_lines** (*int*) – For 2D data, maximum number of lines to be plotted.
- **colormap** (*str*) – The colormap to be used for 2D data.

range_selector(*axis*=0, *sel_data*=None, ***kwargs*)

Interactive plot to manually select a data range.

Parameters

- **axis** (*int*, *optional*) – The axis along which to plot the data. If *xlabel* is None, then for 1D data, 0 is assumed to be q values, and for 2D data, 0 is assumed to correspond to time and 1 to q values. (default, 0)
- **sel_data** (*Sample*, *optional*) – An instance of *Sample* class that will be used to perform the range selection. If None, the instance from which the method was called will be used. (default, None)
- **kwargs** (*dict*, *keywords arguments*) – Additional arguments to be passed to the *Sample.plot()* method.

sliding_average(*window_size*, **metadata*, *axis=0*)

Performs a sliding average of data and errors along given axis.

Parameters

- **window_size** (*int*) – Size of the window for the sliding average.
- **metadata** (*strings*) – List of metadata names that should be binned as well.
- **axis** (*int*, *optional*) – The axis over which the average is to be performed. (default, 0)

Returns **out_arr** – An averaged instance of [Sample](#) with the same metadata except for **errors**, which are processed as well.

Return type [Sample](#)

write_csv(*filename*, *header=None*, *comments='# '*, *footer=None*)

Write the data in text format.

By default, the text contains three columns, the first one corresponding to q values, the second to $I(q)$ and the third to the associated errors.

If sample data are 2D, the data will be averaged over the first dimension before writing the file.

Parameters

- **filename** (*str*) – The name of the file to be written.
- **header** (*str*, *optional*) – A string defining the header of the file.
- **comments** (*str*, *optional*) – The string to be prepend to header lines to indicate a comment.
- **footer** (*str*, *optional*) – A string defining the footer of the file.

data_parsers

Data parsers for SAXS performed on the BM29 beamline at the ESRF with a Pilatus detector.

esrf_bm29.read_HPLC(*filepath*, ***sample_kwargs*)

Reader for an experiment with HPLC in HDF format.

Parameters

- **filepath** (*str*) – The path of the file to be read
- **sample_kwargs** (*dict*, *optional*) – Additional keywords to be passed to the [sample.Sample](#) class after file reading.

esrf_bm29.read_abs_data(*filepath*)

Reader for text file of absorption data from Shimadzu HPLC.

Parameters **filepath** (*str*) – The path of the file to be read.

Returns

- **times** (*np.array*) – The elution time in minutes.
- **wavelengths** (*np.array*) – The wavelengths at which absorption is measured in nm.
- **dataset** (*np.ndarray*) – The 2D dataset of the absorption data with time along the first axis and wavelength along the second.

esrf_bm29.read_processed_1d(*filepath*, ***sample_kwargs*)

Reader for an experiment with HPLC in HDF format.

Parameters

- **filepath** (*str*) – The path of the file to be read
- **sample_kwargs** (*dict*, *optional*) – Additional keywords to be passed to the [sample.Sample](#) class after file reading.

data_analysis

AutoRg

This module provides a routine to compute the radius of gyration.

```
class auto_rg.AutoRg(data, qmin=0.005, qmax=0.25, min_n=4, qRg_limits=(0.1, 1.3), model=None,  
                    fit_kws=None)
```

Computes a radius of gyration for the provided data.

The methods consists in fitting a radius of gyration (Rg) and an intensity at the limit of $q = 0 \text{ \AA}^{-1}$ (I0). This is done by considering several block sizes that will slide along the q range. For each block position, a fit is performed. Subsequently, the distribution of Rg and I0 are computed and the mean values for these two parameters is obtained from the distribution.

The class can treat both 1D and 2D datasets.

Note: By default, the class uses a linear model to fit the Rg, such that the user should provide the logarithm of the data if no user-defined model is provided.

Parameters

- **data** ([sample.Sample](#)) – An instance of [sample.Sample](#) containing the data to fit. No transformation is performed on the data except slicing. Hence, if the model requires to use the log of the data, it is up to the user to apply the operator before passing the data to the class.
- **qmin** (*float*, *optional*) – Minimum value for the range of q values to be used. (default 0.005)
- **qmax** (*float*, *optional*) – Maximum value for the range of q values to be used. (default 0.2)
- **min_n** (*int*, *optional*) – Minimum number of points to use for the fit. (default 10)
- **qRg_limits** (*2-tuple of floats*, *optional*) – Minimum and maximum limit of $q \cdot Rg$ value to consider the fit valid. (default (0.0005, 1.3))
- **model** ([model.Model](#)) – An instance of Model class to be used to fit the data. If none, a built-in model from SAXS_routines will be used.
- **fit_kws** (*dict*, *optional*) – Additional keywords to pass the fit function (default None)

fitResult

A list of fitted models, each being an instance of the [model.Model](#) class.

Type list of [model.Model](#)

prob

A list of arrays giving the posterior probability that the Rg and I0 values are correct given the data as a function of Rg and I0. There is one array per 1D scattering curve in the dataset.

Type list of np.array

rg

The values of Rg determined by the routine for each 1D scattering curve in the dataset.

Type list of float

rg_std

The errors for Rg determined by the routine for each 1D scattering curve in the dataset.

Type list of float

I0

The values of I0 determined by the routine for each 1D scattering curve in the dataset.

Type list of float

I0_std

The errors for I0 determined by the routine for each 1D scattering curve in the dataset.

Type list of float

Examples

The class can be initialized from a 2D dataset using:

```
>>> import numpy as np
>>> from saxs_routines.data_parsers.esrf_bm29 import read_HPLC
>>> from saxs_routines.data_analysis.auto_rg import AutoRg
>>>
>>> data = read_HPLC('myFile', name='myProtein')
>>> # buffer subtraction
>>> sub = data[480:520] - data[:,50].mean(0)
>>>
>>> # AutoRg part
>>> autorg = AutoRg(np.log(sub))
>>> autorg.run()
>>> autorg.rg
[6.625, 6.232, 6.457, ..., 6.890, 7.011]
```

The result can be plotted as well as other attributes:

```
>>> autorg.plot()
```

plot(*plot_errors=True, new_fig=False*)

Plot the fitted parameters, Rg and I0.

Initially intended to be used with 2D dataset with time as first axis.

Parameters

- **plot_errors** (*bool, optional*) – Whether to plot the errors on the parameters or not. (default False)
- **new_fig** (*bool, optional*) – If True, a new figure will be created. Otherwise, use the existing one if any. (default False)

plot_I0_posterior_prob(*idx=0*)

Plot the computed posterior probability for the I0 value.

The posterior probability is computed from the Bayes rule for each fit at a given index. The plot gives the probability as a function of fitted IO.

Parameters `idx (int)` – Index of the data to be plotted for 2D dataset of time series.

plot_fit(`idx=0, best=False, new_fig=True`)

Plot the fitted model against experimental data.

Parameters

- **idx (int)** – Index of the data to be plotted for 2D dataset of time series.
- **best (bool, optional)** – If True, plot only the best estimate of the Rg and IO. If False, plot all fitted models. (default, False)

plot_rg_posterior_prob(`idx=0`)

Plot the computed posterior probability for the Rg value.

The posterior probability is computed from the Bayes rule for each fit at a given index. The plot gives the probability as a function of fitted radius of gyration.

Parameters `idx (int)` – Index of the data to be plotted for 2D dataset of time series.

run()

Run the algorithm to extract the radius of gyration.

The result is stored in the `fitResult`, `prob`, `rg`, `IO`, `rg_std` and `IO_std` attributes.

FindPeaks

Automatic peak finding for 2D SEC-SAS data.

class `find_peaks.FindPeaks`(`data, height_factor=0.5`)

Finds the peaks in a time series from a 2D SEC-SAS dataset.

Parameters

- **data** (`sample.Sample`) – An instance of `sample.Sample` containing a 2D dataset where the first axis is assumed to be the elution time and the second axis is assumed to be the momentum transfer q values.
- **height_factor (float)** – A float between 0 and 1. This will be used to define the minimum height of the peak by taking `height = height_factor * data.max()`.

get_sub_arrays()

Return the array corresponding to the region within peak borders.

run(`find_peaks_kws=None, peak_widths_kws=None`)

Run the algorithm to find the peaks and associated widths.

Parameters

- **find_peaks_kws (dict)** – Additional keywords to be passed to scipy `find_peaks` keywords.
- **peak_widths_kws (dict)** – Additional keywords to be passed to scipy `peak_widths` keywords.

models

Model

This module provides a template class to build models that can be used to fit the data.

class `model.Component`(*name, func, skip_convolve=False, **funcArgs*)

Component class to be used with the `Model` class.

Parameters

- **name** (*str*) – Name for the component.
- **func** (*callable*) – The function to be used for this component.
- **skip_convolve** (*bool*) – If True, no convolution is performed for this model. It can be useful for background or normalization terms.
- **funcArgs** (*dict of str, int, float or arrays*) – Values to be passed to the function arguments. This is a dictionary of argument names mapped to values. The values can be of different types:
 - **int, float or array**, the values are directly passed to the function.
 - **str**, the values are evaluated first. If any word in the string is present in the `Model.params` dictionary keys, the corresponding parameter value is substituted.

Examples

For a `Model` class that has the following key in its `params` attribute: ('amplitude', 'sigma'), the component for a Lorentzian, the width of which depends on a defined vector *q*, can be created using:

```
>>> def lorentzian(x, scale, width):
...     return scale / np.pi * width / (x**2 + width**2)
>>> myComp = Component(
...     'lor', lorentzian, scale='scale', width='width * q**2')
```

If the Lorentzian width is constant, use:

```
>>> myComp = Component('lor', lorentzian, scale='scale', width=5)
```

Some math functions can be used as well (below the exponential):

```
>>> myComp = Component('lor', lorentzian, scale='np.exp(-q**2 * msd)')
```

eval(*x, params, **kwargs*)

Evaluate the components using the given parameters.

Parameters

- **params** (*Parameters* instance) – Parameters to be passed to the component
- **kwargs** (*dict*) – Additional parameters to be passed to the function. Can override params.

processFuncArgs(*params, **kwargs*)

Return the evaluated argument for the function using given parameters and keyword arguments.

class `model.FindParamNames`(*key, params*)

Helper class to parse strings to evaluation for function arguments in `Component`.

Parameters **params** (Parameters) – An instance of Parameters from which the parameter names are to be found and substituted by the corresponding values.

visit_Name(*node*)

Name visitor.

class **model.Model**(*params*, *name*='Model', *convolutions*=None, *on_undef_conv*='numeric')

Model class to be used for fitting.

The model is structured in components that can be added together, each component consisting of a name, a callable function and a dictionary of parameters. The parameters of two different components can have the same name such that they can be shared by several components just like for the switching diffusive state model.

Also, the components are separated in two classes, namely *eisfComponents* and *qisfComponents*, in order to provide the possibility to separately extract the elastic and quasi-elastic parts for analysis and plotting.

Parameters

- **params** (Parameters instance) – Parameters to be used with the model
- **name** (*str*, *optional*) – A name for the model.
- **convolutions** (*dict of dict*) – Dictionary that defines the mapping '(function1, function2)' to 'convolutionFunction(function1, function2)'. Analytic convolutions or user defined operators can be defined this way.
- **on_undef_conv** (*{'raise', 'numeric'}*) – Defines the behavior of the class on missing convolution function in the 'convolutions' attribute. The option 'raise' leads to a *KeyError* and the option 'numeric' to a numerical convolution.

addComponent(*comp*, *op*='+')

Add a component to the model.

Parameters

- **comp** (*Component*) – An instance of *Component* to be added to the model.
- **op** (*{'+' , '-' , '*' , '/'}*, *optional*) – Operator to be used to combine the new component with the others. If this is the first component, the operator is ignored. (default "+")

property **bic**

Return the bayesian information criterion (BIC).

property **components**

Return the model components.

copy()

Return a copy of the model.

eval(*x*, *params*=None, *convolve*=None, ***kwargs*)

Perform the assembly of the components and call the provided functions with their parameters to compute the model.

Parameters

- **x** (*np.ndarray*) – Values for the x-axis variable
- **params** (*list*, *np.array*, *optional*) – Parameters to be passed to the components. Will override existing parameters in *self.params*.
- **convolve** (*Model*) – Another model to be convolved with this one.

- **kwargs** – Additional keyword arguments to be passed to the components. Can override params too.

Returns

- If *returnComponents* is False – The computed model in an array, the dimensions of which depend on *x* and *params* attributes and the function called.
- *else* – A dictionary with key being the component names and the values are the evaluated components.

eval_components(*x*, *params=None*, *convolve=None*, ***kwargs*)

Alias for *eval* with 'returnComponents' set to True.

Perform the computation of the components with the given x-axis values, parameters and convolutions.

Returns

- *A dictionary with key being the component names and the values*
- *are the evaluated components.*

fit(*x*, *data=None*, *weights=None*, *fit_method='curve_fit'*, *fit_kws=None*, *params=None*, ***kwargs*)

Fit the experimental data using the provided arguments.

Parameters

- **x** (*np.ndarray*) – Values for the independent variable.
- **data** (*np.ndarray*) – Experimental data to be fitted.
- **weights** (*np.ndarray*, *optional*) – Weights associated with the experimental data (the experimental errors).
- **fit_method** (*str*, *optional*) – The method to be used for fitting. Currently available methods are (from Scipy): - “curve_fit” - “basinhopping” - “differential_evolution” - “shgo” - “minimize”
- **fit_kws** (*dict*, *optional*) – Additional keywords to be passed to the fit method.
- **params** (*Parameters* class instance, *optional*) – Parameters to be used (default None, will use the parameters associated with the model).
- **kwargs** (*dict*, *optional*) – Additional keywords arguments to give for the evaluation of the model. Can override parameters too.

Returns

Return type A copy of the fitted model instance.

property fitResult

Return the full result of the fit.

property on_undef_conv

Return the class behavior on undefined convolution.

property optParams

Return the result of the fit.

property userkws

Return the keywords used for the fit.

Params

The module contains a Parameter class to be used with the Model class.

class `params.Parameters`(*params=None, **kwargs*)

A parameter class that handles names, values and bounds.

Parameters

- **params** (*dict of dict*) – A dictionary of parameter names, each being associated with a namedtuple containing the ‘value’, the ‘bounds’, the ‘fixed’, and the ‘error’ attributes.
- **kwargs** (*keywords*) – Additional keywords argument to set parameter names, values (and possibly bounds and fixed attributes). Can override params too.

listToParams(*pList, errList=None*)

Use the given list to convert a list of parameters to a dictionary similar to the current one.

property paramList

Accessor for parameter list.

set(*name, **kwargs*)

Set a parameter entry with given attributes in ‘kwargs’.

Parameters

- **name** (*str*) – Parameter name to be updated.
- **kwargs** (*dict of float, tuple or namedtuple*) – Parameters to be updates with the associated attributes. The call should be of the form:

```
>>> params.set('amplitude', value=1.2, fixed=True)
>>> params.set('width', value=2.3, bounds=(0., np.inf))
```

update(***kwargs*)

Update the parameters.

`params.pTuple`(*value=1, bounds=(- inf, inf), fixed=False, error=0.0*)

Helper function to create a namedtuple with default values.

Builtins

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

a

`auto_rg`, 8

e

`esrf_bm29`, 7

f

`find_peaks`, 10

m

`model`, 11

p

`params`, 14

s

`sample`, 3

A

`addComponent()` (*model.Model* method), 12
`auto_rg`
 module, 8
`AutoRg` (*class in auto_rg*), 8

B

`bic` (*model.Model* property), 12
`bin()` (*sample.Sample* method), 5

C

`Component` (*class in model*), 11
`components` (*model.Model* property), 12
`copy()` (*model.Model* method), 12

E

`esrf_bm29`
 module, 7
`eval()` (*model.Component* method), 11
`eval()` (*model.Model* method), 12
`eval_components()` (*model.Model* method), 13

F

`find_peaks`
 module, 10
`FindParamNames` (*class in model*), 11
`FindPeaks` (*class in find_peaks*), 10
`fit()` (*model.Model* method), 13
`fitResult` (*auto_rg.AutoRg* attribute), 8
`fitResult` (*model.Model* property), 13

G

`get_q_range()` (*sample.Sample* method), 5
`get_sub_arrays()` (*find_peaks.FindPeaks* method), 10
`get_time_range()` (*sample.Sample* method), 6

I

`I0` (*auto_rg.AutoRg* attribute), 9
`I0_std` (*auto_rg.AutoRg* attribute), 9

L

`listToParams()` (*params.Parameters* method), 14

M

`model`
 module, 11
`Model` (*class in model*), 12
`module`
 `auto_rg`, 8
 `esrf_bm29`, 7
 `find_peaks`, 10
 `model`, 11
 `params`, 14
 `sample`, 3

O

`on_undef_conv` (*model.Model* property), 13
`optParams` (*model.Model* property), 13

P

`Parameters` (*class in params*), 14
`paramList` (*params.Parameters* property), 14
`params`
 module, 14
`plot()` (*auto_rg.AutoRg* method), 9
`plot()` (*sample.Sample* method), 6
`plot_fit()` (*auto_rg.AutoRg* method), 10
`plot_I0_posterior_prob()` (*auto_rg.AutoRg* method), 9
`plot_rg_posterior_prob()` (*auto_rg.AutoRg* method), 10
`prob` (*auto_rg.AutoRg* attribute), 8
`processFuncArgs()` (*model.Component* method), 11
`pTuple()` (*in module params*), 14

R

`range_selector()` (*sample.Sample* method), 6
`read_abs_data()` (*in module esrf_bm29*), 7
`read_HPLC()` (*in module esrf_bm29*), 7
`read_processed_1d()` (*in module esrf_bm29*), 7
`rg` (*auto_rg.AutoRg* attribute), 9
`rg_std` (*auto_rg.AutoRg* attribute), 9
`run()` (*auto_rg.AutoRg* method), 10
`run()` (*find_peaks.FindPeaks* method), 10

S

sample

 module, [3](#)

Sample (*class in sample*), [3](#)

set() (*params.Parameters method*), [14](#)

sliding_average() (*sample.Sample method*), [6](#)

T

T (*sample.Sample property*), [5](#)

U

update() (*params.Parameters method*), [14](#)

userkws (*model.Model property*), [13](#)

V

visit_Name() (*model.FindParamNames method*), [12](#)

W

write_csv() (*sample.Sample method*), [7](#)