nxs-analysis-tools

Release 0.0.32

Steven J. Gomez Alvarado

Feb 26, 2024

CONTENTS

CHAPTER

ONE

OVERVIEW

nxs-analysis-tools provides a suite of tools for slicing (2D), cutting (1D), and transforming (e.g., symmetrizing, interpolating, deltaPDF) nexus format (.nxs) scattering data.

CHAPTER

TWO

CONTENTS

2.1 Examples

2.1.1 Visualizing data using the plot_slice function

`plot_slice` is a function that is capable of plotting 2-D datasets on axes which need not be orthogonal. This was designed originally for plotting X-ray scattering datasets much in the way that the NeXpy package does.

Importing `plot_slice`

```
from nxs_analysis_tools import plot_slice, load_data
from nexusformat.nexus import NXdata, NXfield
import numpy as np
```

Import data

```
data_cubic = load_data('example_data/plot_slice_data/cubic_hkli.nxs')
data_hex = load_data('example_data/plot_slice_data/hex_hkli.nxs')
```

```
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
```

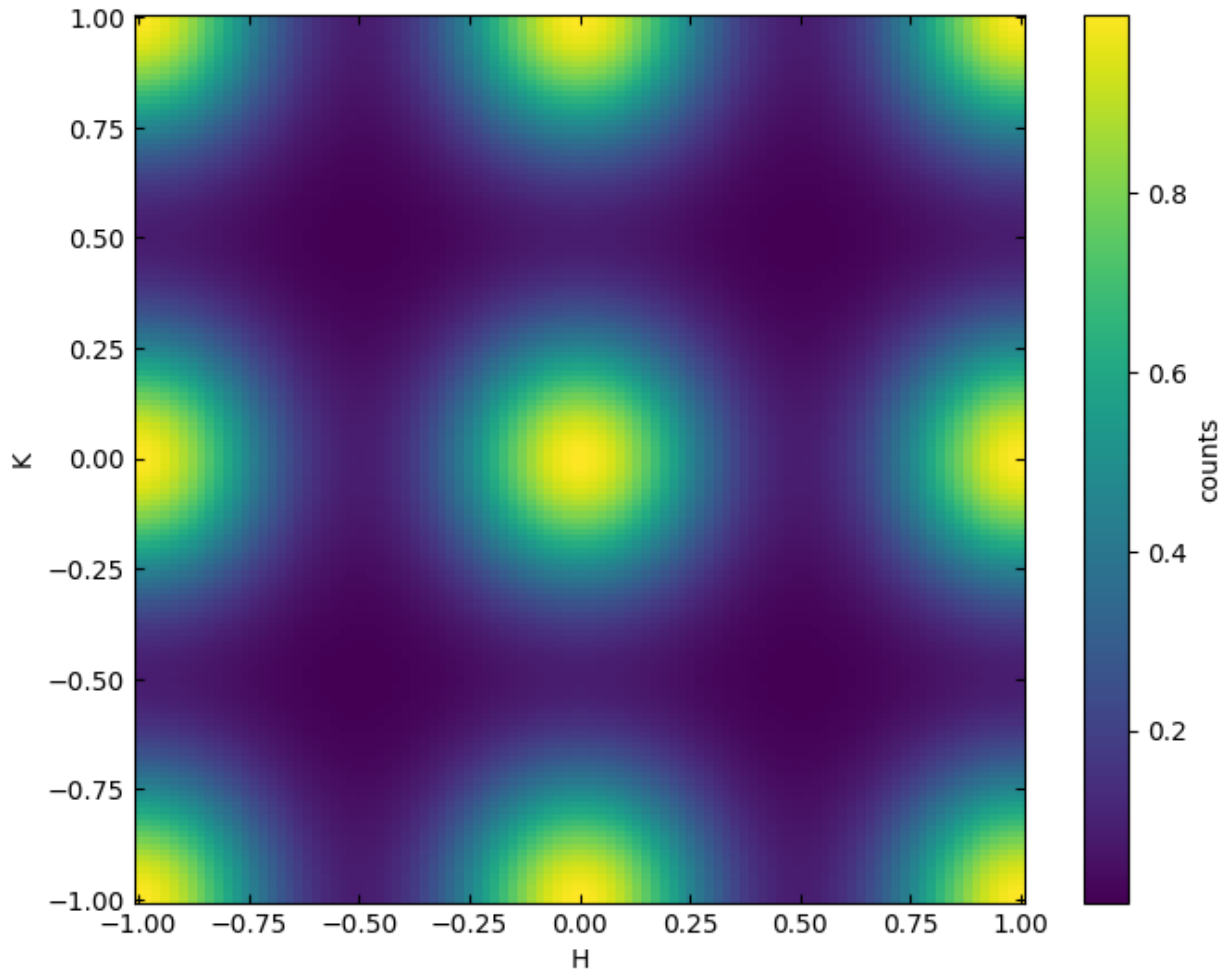
Basic plotting

Plot slice accepts an `NXdata` object which hold 2-D data. Thus, if working with a 3D scattering dataset, you must index one of the axes.

Here we plot the K=0 plane.

```
plot_slice(data_cubic[:,:,0.0])
```

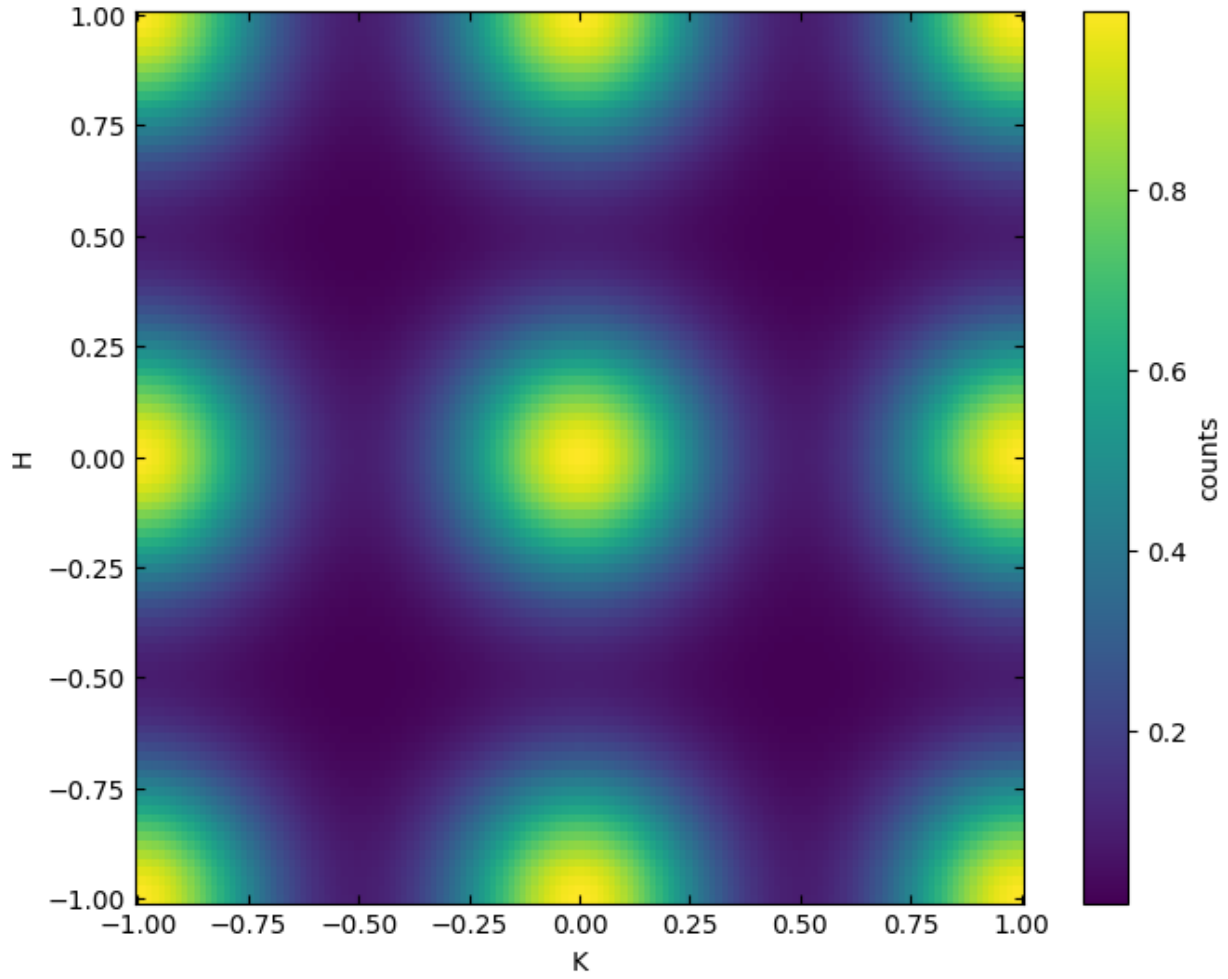
```
<matplotlib.collections.QuadMesh at 0x7fa29e132650>
```



Transposing the X and Y axes

```
plot_slice(data_cubic[:,:,0.0], transpose=True)
```

```
<matplotlib.collections.QuadMesh at 0x7fa29c0f1b40>
```



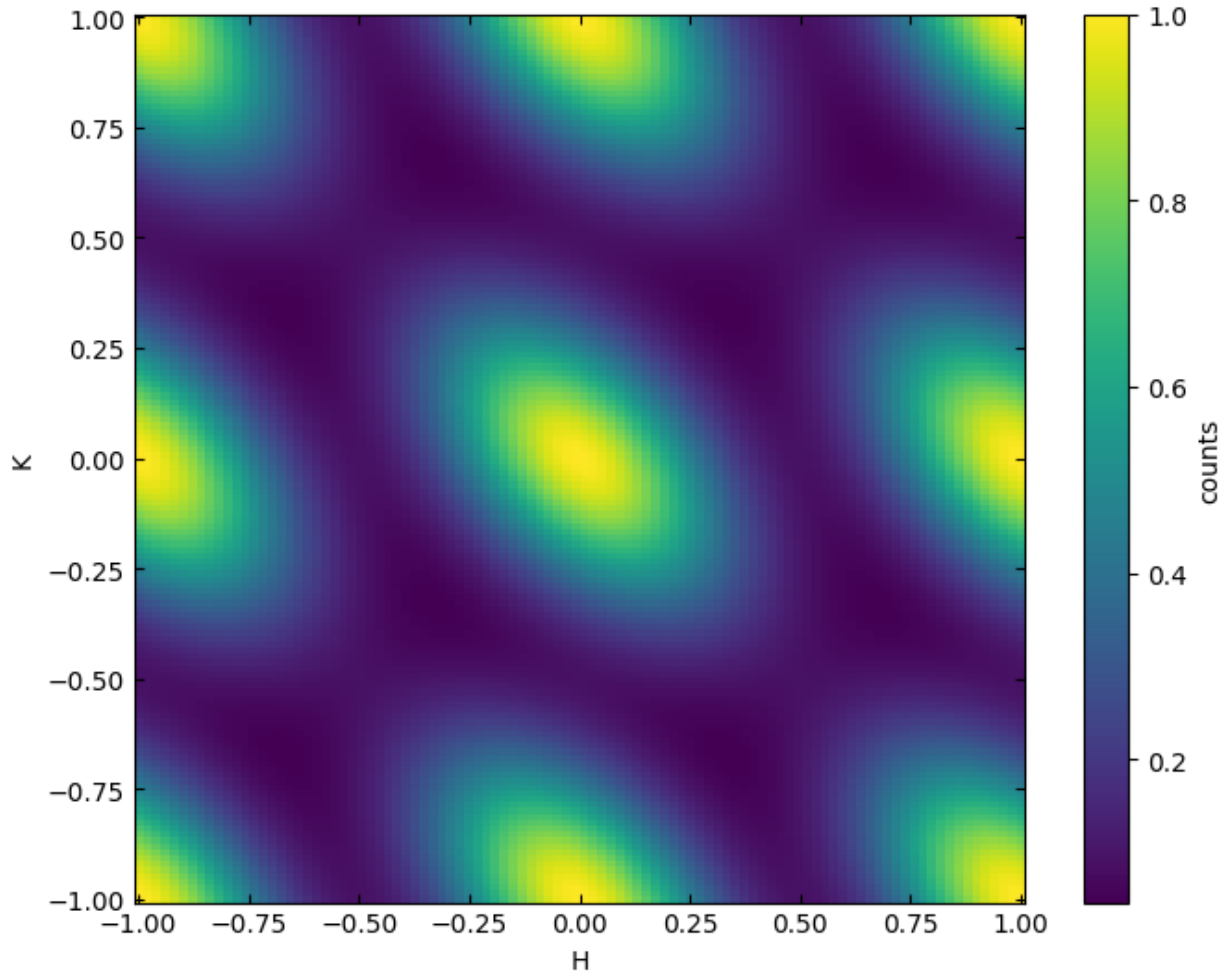
Plotting on non-orthogonal axes

When working in a non-orthorhombic cell, it may be necessary to specify the skew angle between the axes of interest. Here, we consider a hexagonal crystal structure, for which H and K are 60° apart.

If we plot the data using no additional arugments, we see that the Bragg reflections are skewed.

```
plot_slice(data_hex[:,:,0])
```

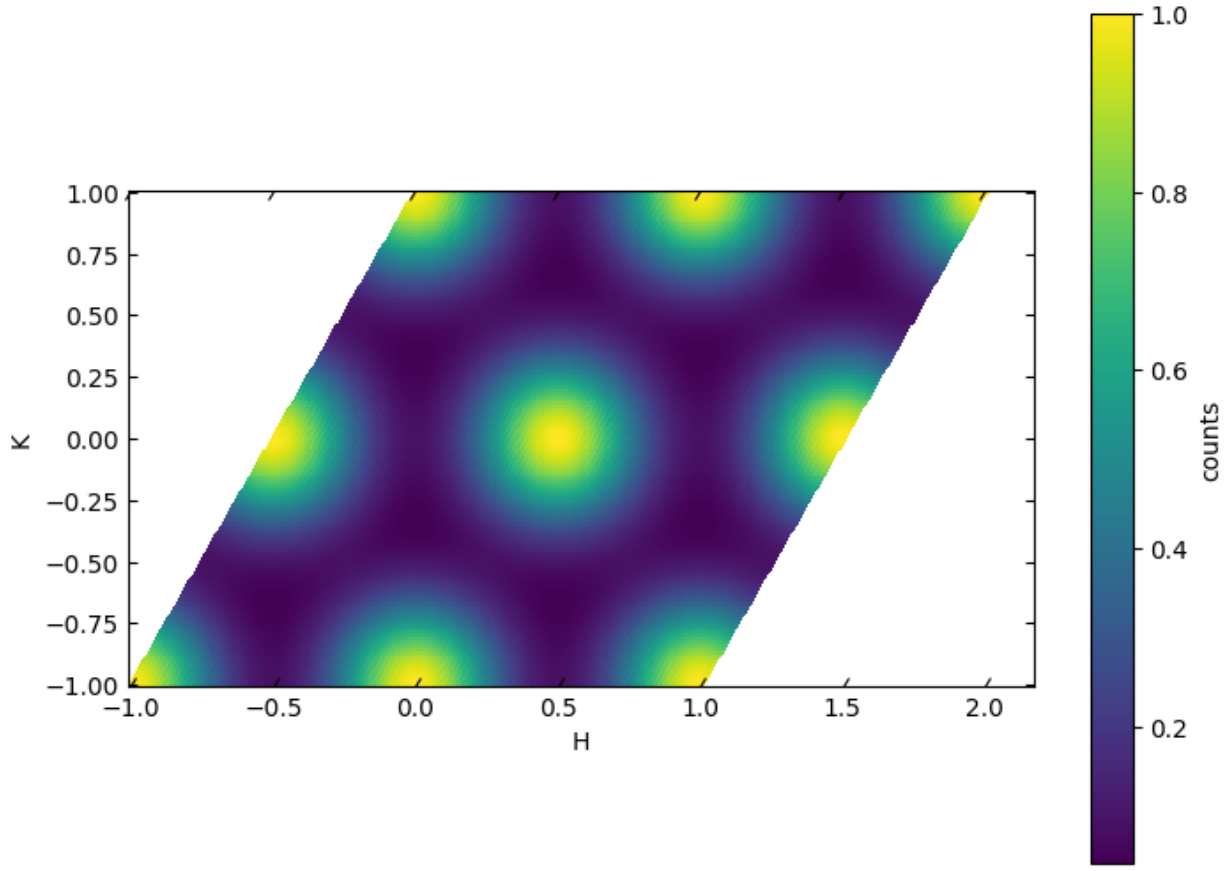
```
<matplotlib.collections.QuadMesh at 0x7fa29be84ca0>
```



Use the `skew_angle` parameter to correct the angle between the plotted X and Y axes.

```
plot_slice(data_hex[:,:,0], skew_angle=60)
```

```
<matplotlib.collections.QuadMesh at 0x7fa29a509690>
```

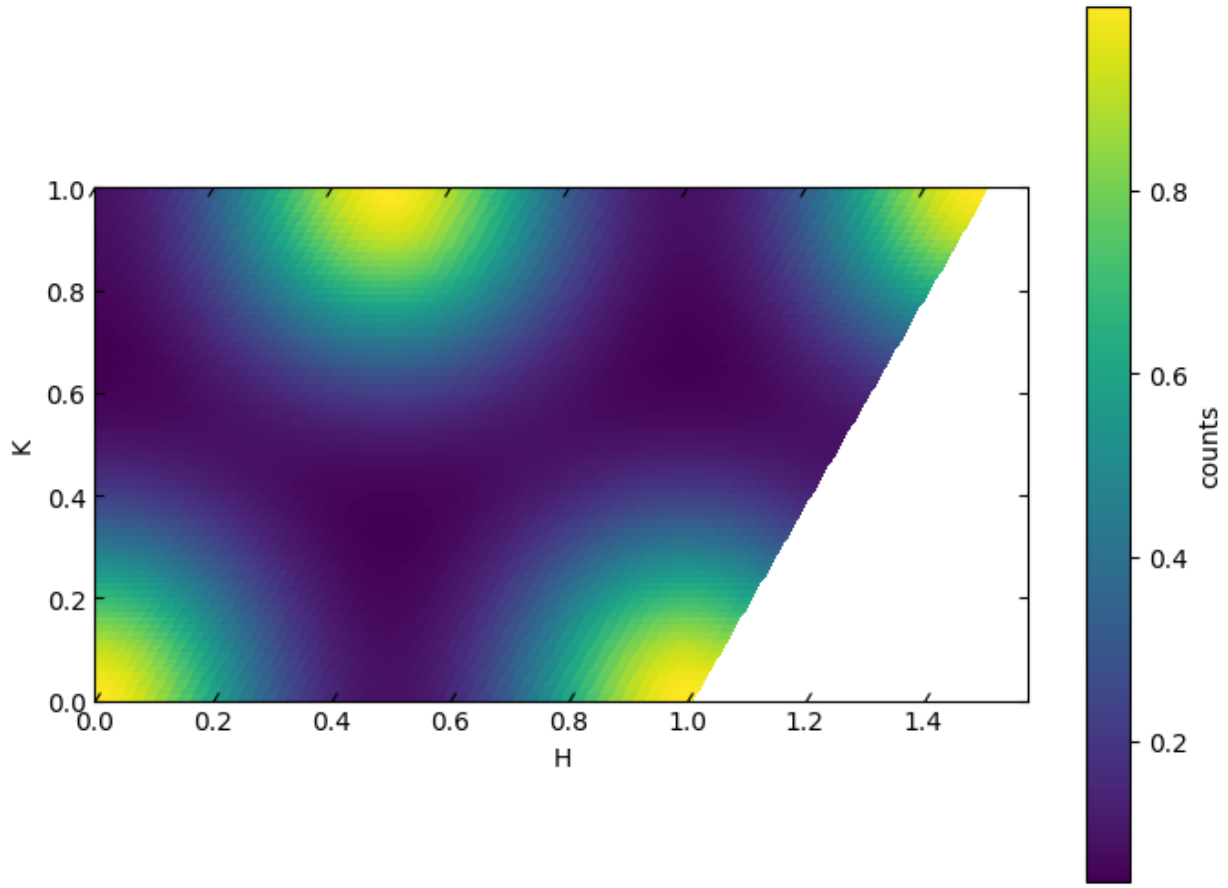


Adjusting axis limits

Note that the x-axis limits are interpreted in a way that displays the full area covered by the specified x and y limits. Thus, when working with a skewed dataset, the actual tick marks on the x-axis will extend longer than the specified limits.

```
plot_slice(data_hex[:,:,0.0], skew_angle=60, xlim=(0,1), ylim=(0,1))
```

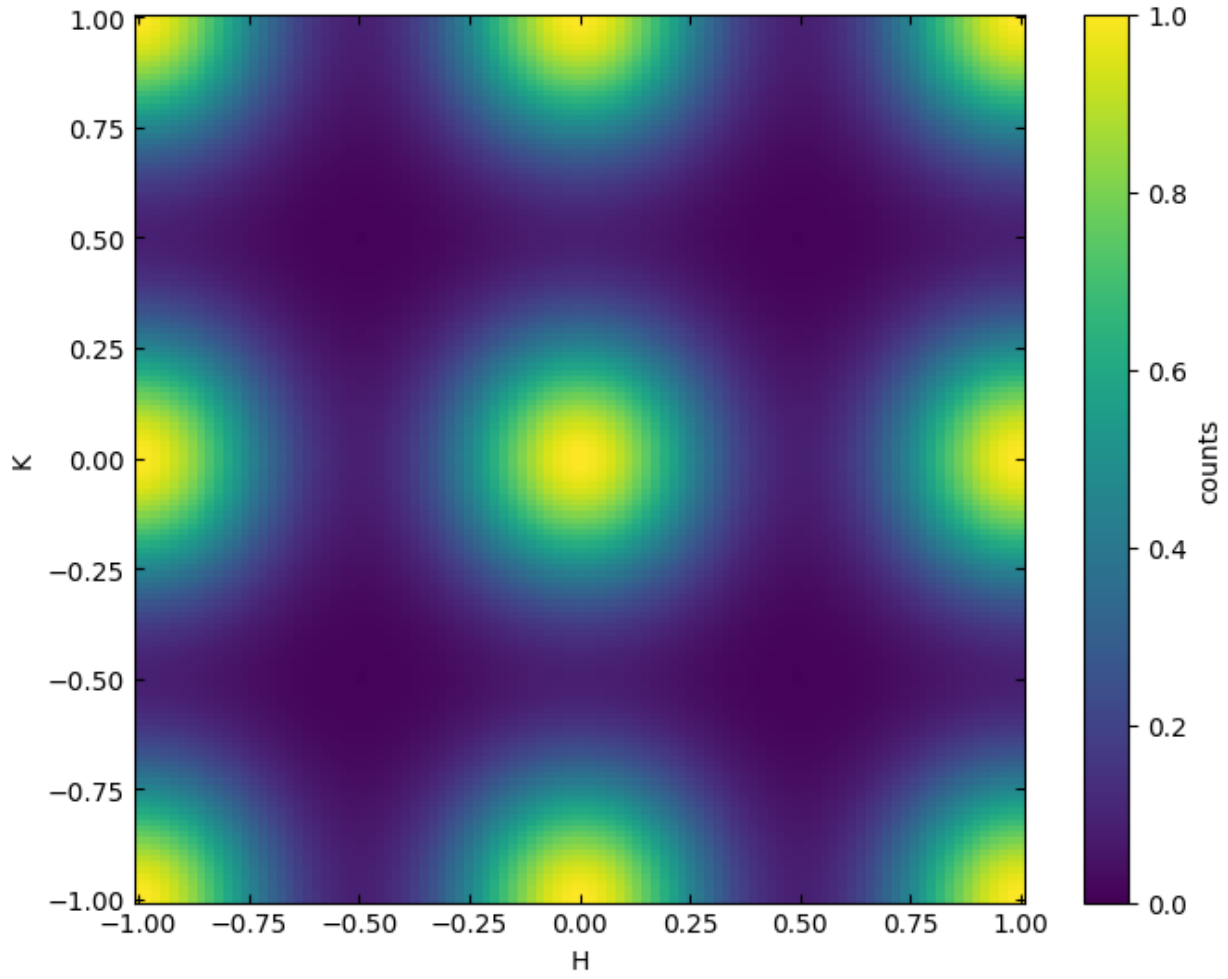
```
<matplotlib.collections.QuadMesh at 0x7fa29a408880>
```



Adjusting color mapping

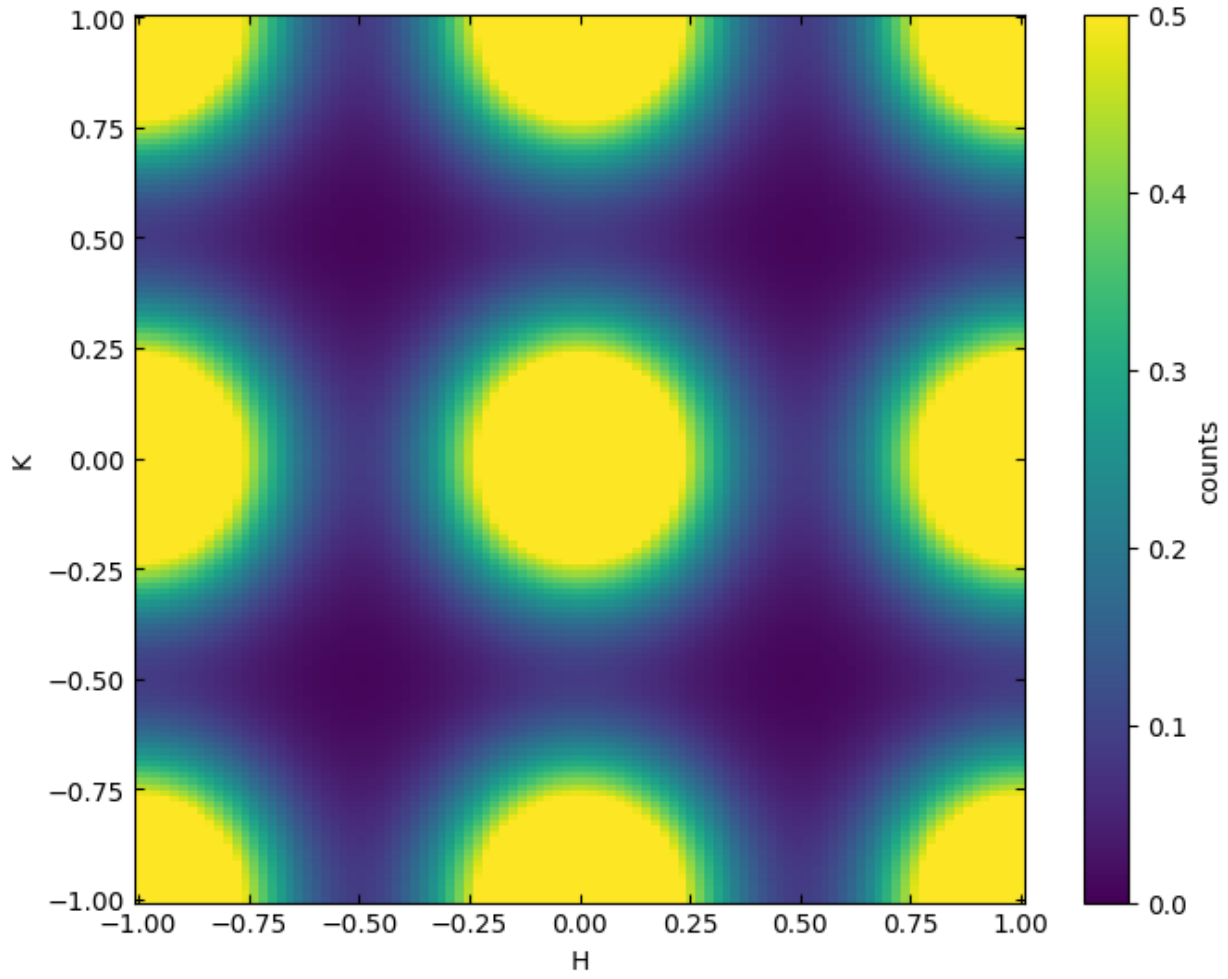
```
plot_slice(data_cubic[:,:,0.0], vmin=0,vmax=1)
```

```
<matplotlib.collections.QuadMesh at 0x7fa29a2ec610>
```



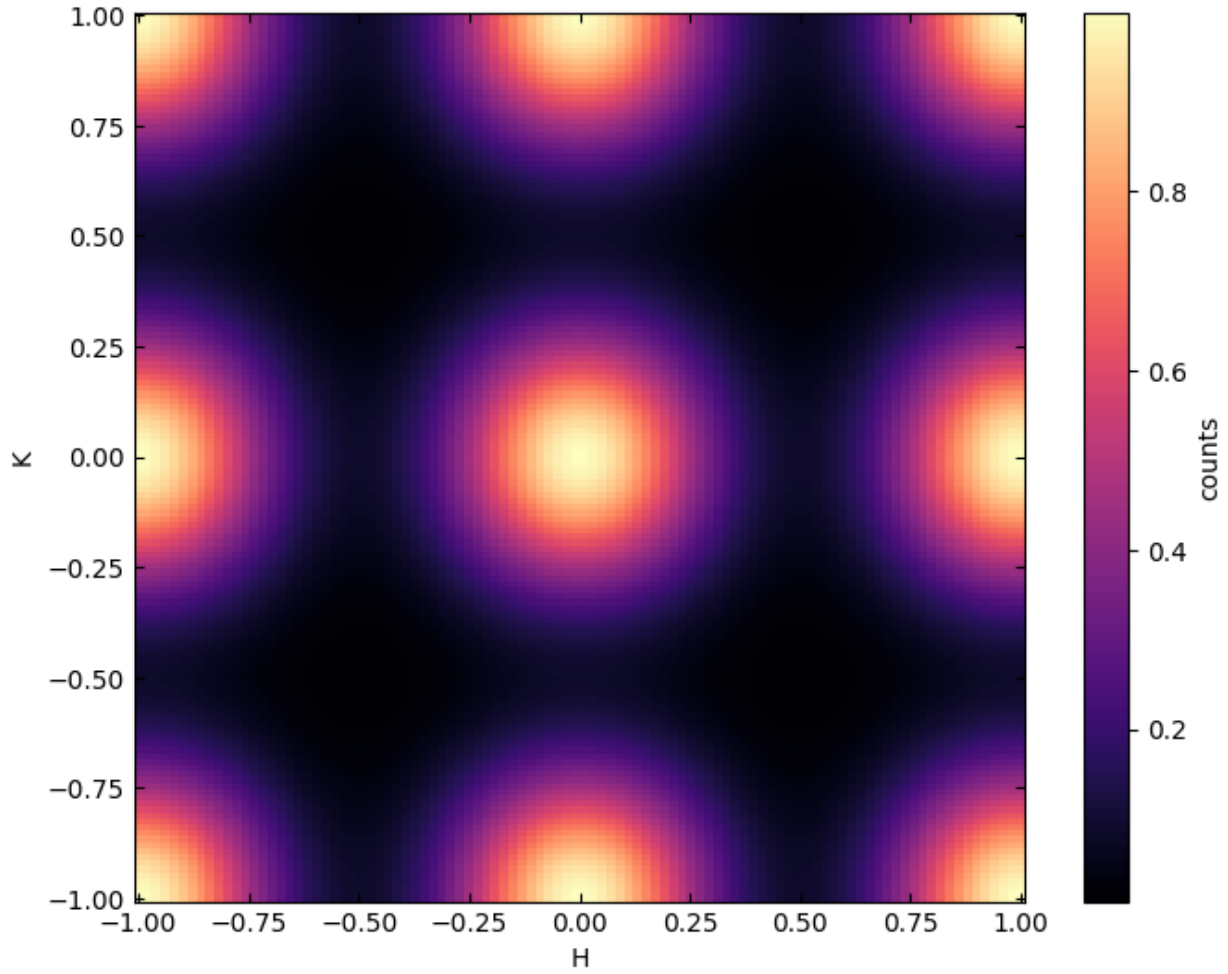

```
plot_slice(data_cubic[:,:,0.0], vmin=0,vmax=0.5)
```

```
<matplotlib.collections.QuadMesh at 0x7fa29a1dcfa0>
```




```
plot_slice(data_cubic[:,:,0.0], cmap='magma')
```

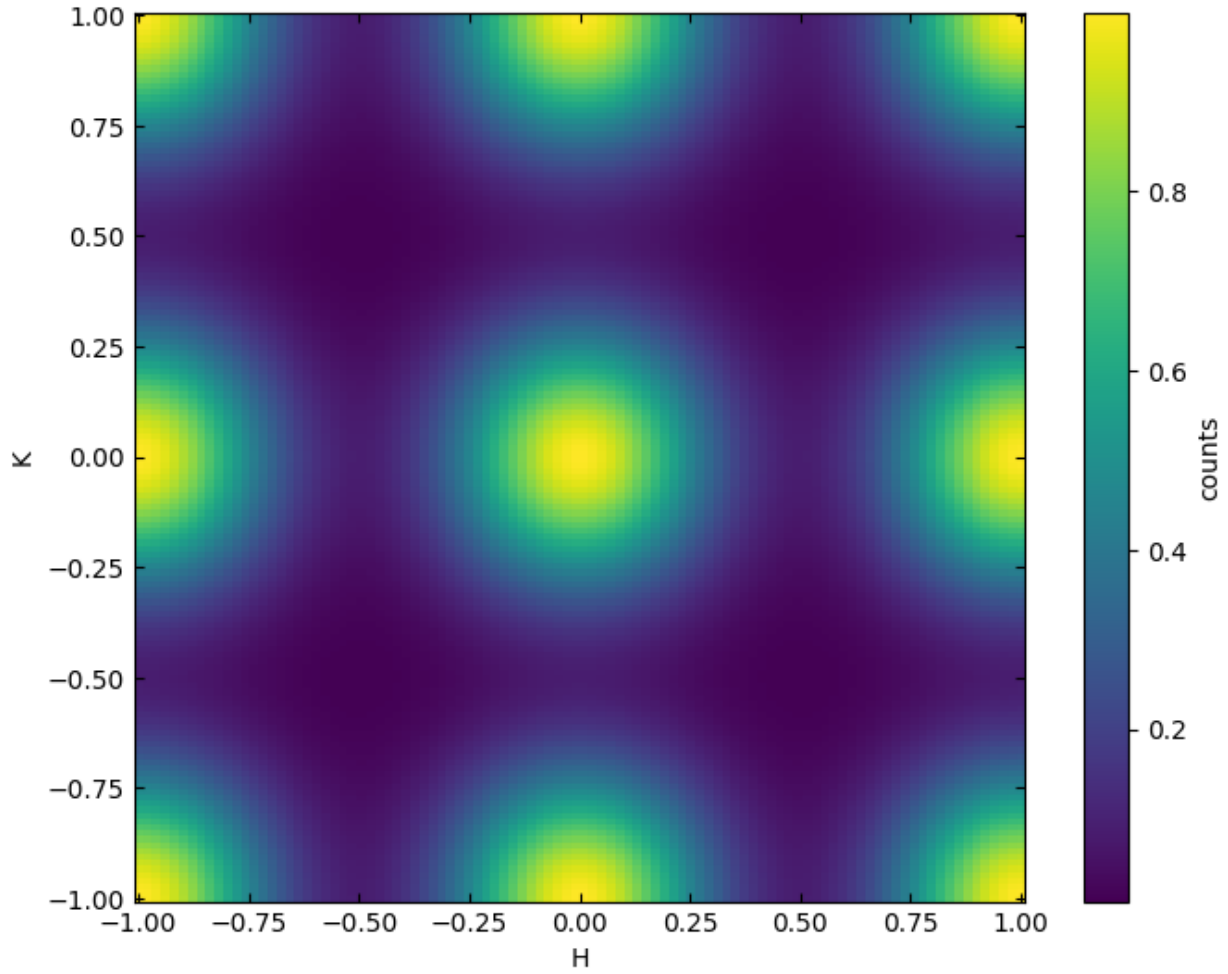
```
<matplotlib.collections.QuadMesh at 0x7fa29a0dd330>
```



Using logscale

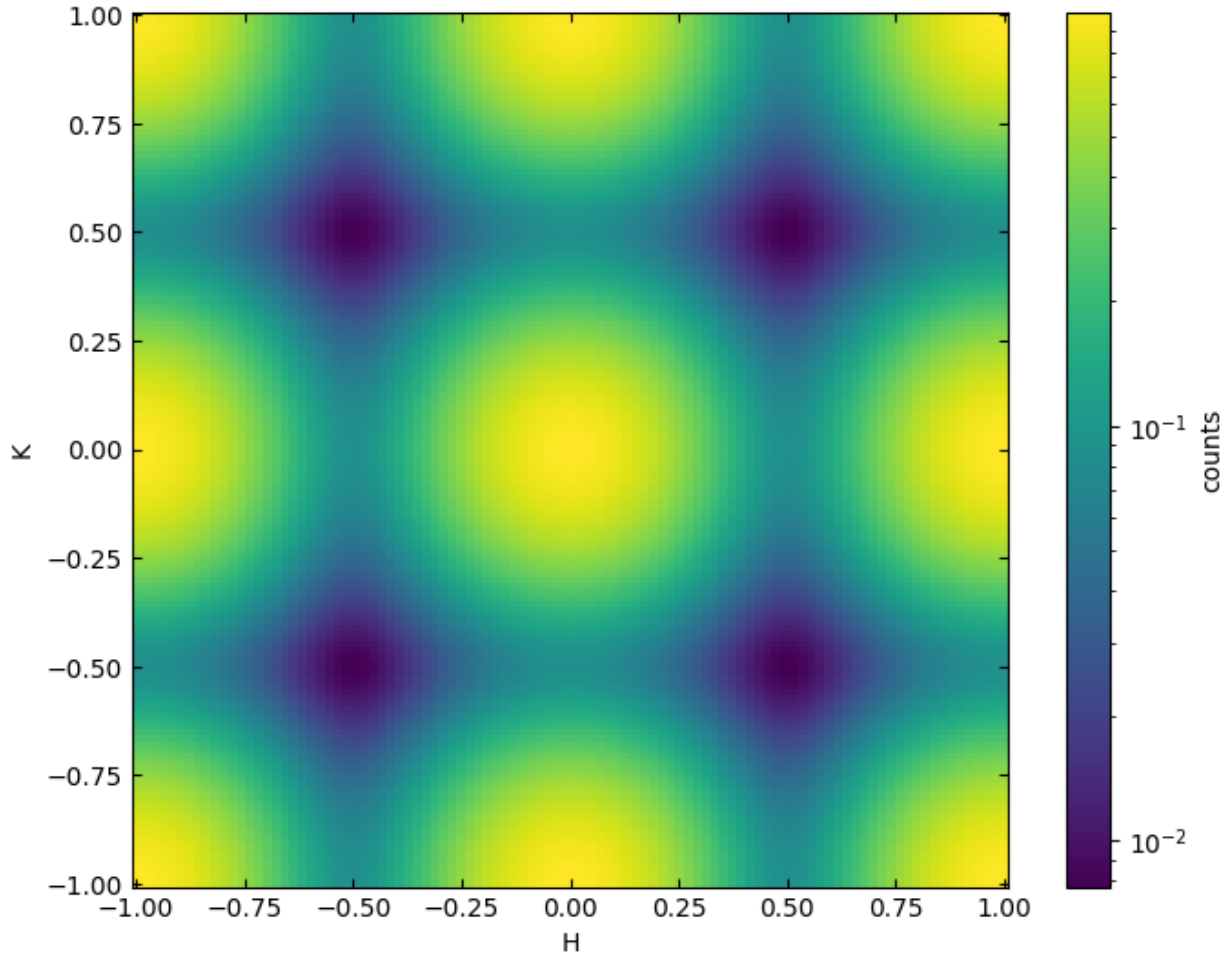
```
plot_slice(data_cubic[:,:,0.0], logscale=False)
```

```
<matplotlib.collections.QuadMesh at 0x7fa299fe0490>
```




```
plot_slice(data_cubic[:,:,0.0], logscale=True)
```

```
<matplotlib.collections.QuadMesh at 0x7fa299e6f880>
```



Using symlogscale

First, let's create a test dataset with both positive and negative values

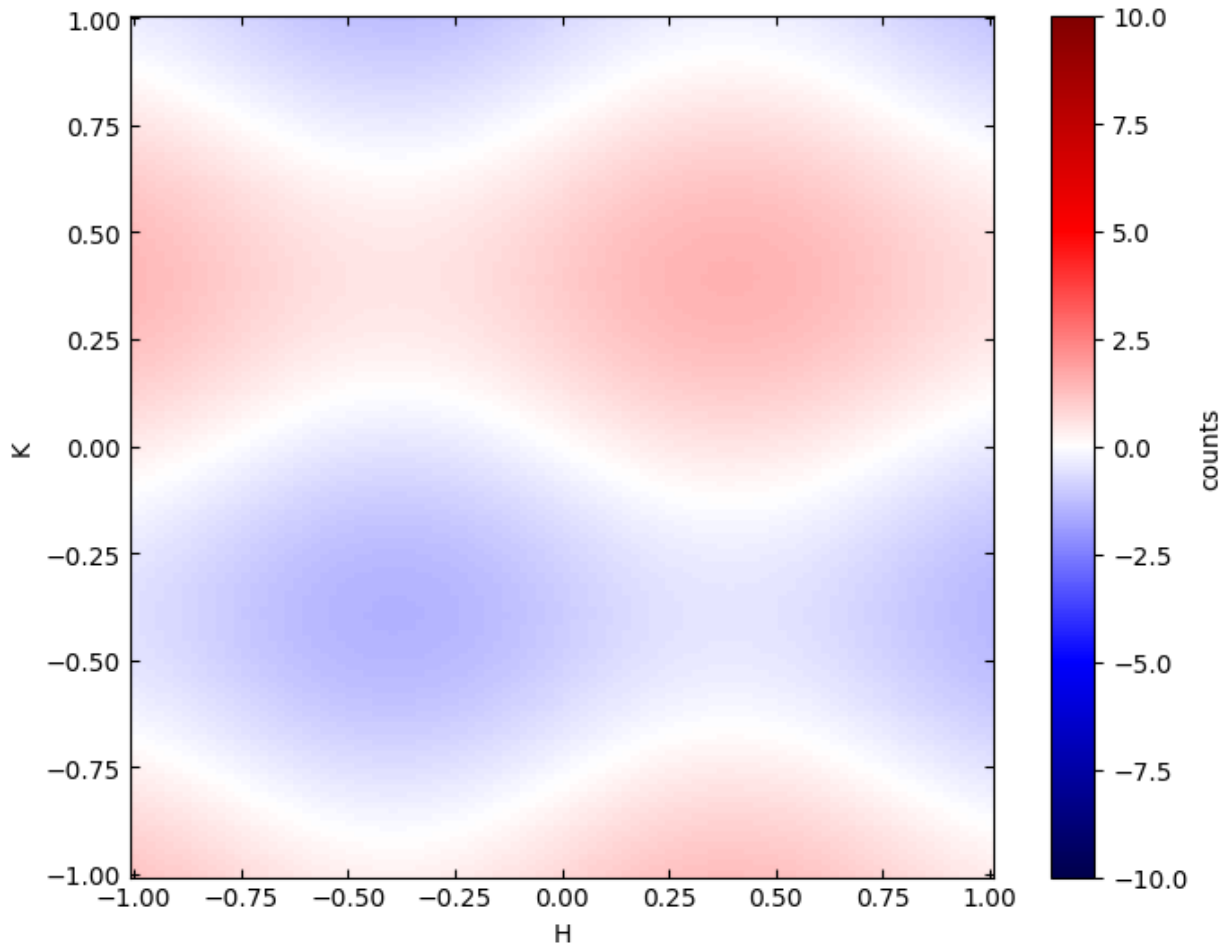
```
data_sym = load_data('example_data/plot_slice_data/sym_hkli.nxs')
```

```
data:NXdata
  @axes = ['H', 'K']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  counts = float64(100x150)
```

Without the `symlogscale`, the data is plotted on a linear colormap. Both a `vmin` and `vmax` can be provided for the limits of the colormap.

```
plot_slice(data_sym, cmap='seismic', symlogscale=False, vmin=-10, vmax=10)
```

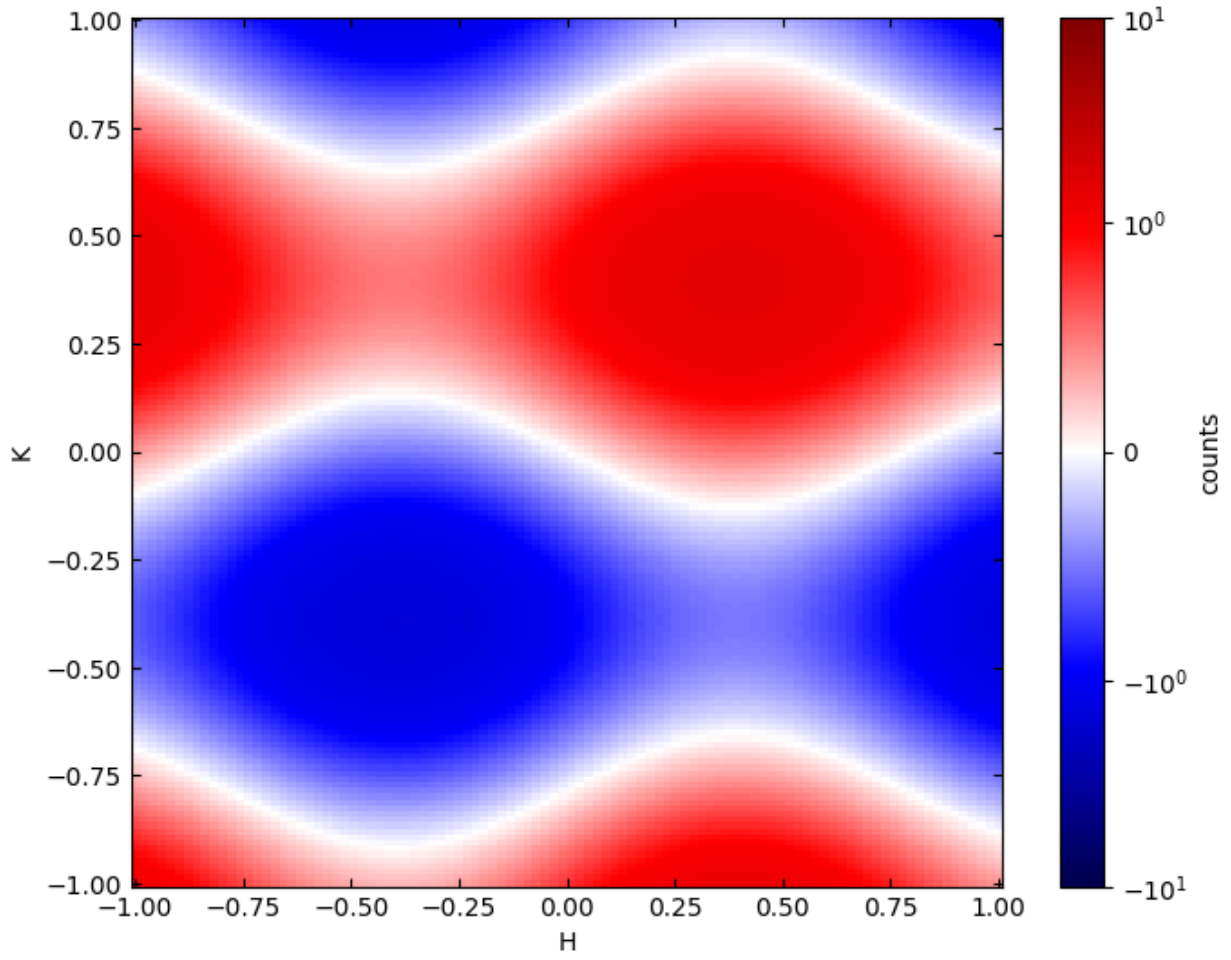
```
<matplotlib.collections.QuadMesh at 0x7fa29a28b6d0>
```



With `symlogscale` enabled, only the `vmax` parameter is used the colorbar is symmetric about zero.

```
plot_slice(data_sym, cmap='seismic', symlogscale=True, vmax=10)
```

```
<matplotlib.collections.QuadMesh at 0x7fa299d654e0>
```



2.1.2 Performing linecuts using the Scissors class

Importing the `Scissors` class

```
from nxs_analysis_tools import Scissors, load_data
from nexusformat.nexus import NXdata, NXfield
import numpy as np
```

Setting up a linecut - Method 1

You can set the data, linecut center, integration window, and integration axis when you initialize the `Scissors` object.

```
data = load_data('example_data/sample_name/15/example_hkli.nxs')
```

```
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
```

(continues on next page)

(continued from previous page)

```
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
```

```
scissors = Scissors(data, center=(0,0,0), window=(0.2,0.2,2), axis=None)
```

Setting up a linecut - Method 2

…or you can set them one by one after initializing the object.

```
scissors = Scissors()
```

```
scissors.set_data(data)
```

```
scissors.set_center((0,0,0))
```

```
scissors.set_window((0.1,1,0.2))
```

```
Linecut axis: K
Integrated axes: ['H', 'L']
```

Performing a linecut

You can use `.cut_data()` to perform the cut, returning an `NXdata` object.

```
linecut = scissors.cut_data()
```

```
Linecut axis: K
Integrated axes: ['H', 'L']
```

Plotting a linecut

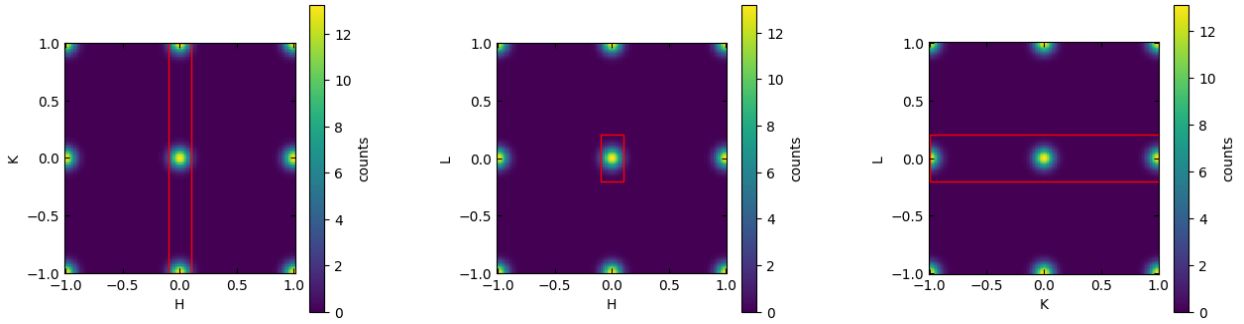
The `.integration_window` attribute stores a `tuple` of `slice` objects over which the integration was performed.

```
scissors.integration_window
```

```
(slice(-0.1, 0.1, None), slice(-1.0, 1.0, None), slice(-0.2, 0.2, None))
```

To show where the integration was performed, use the `.highlight_integration_window()` method.

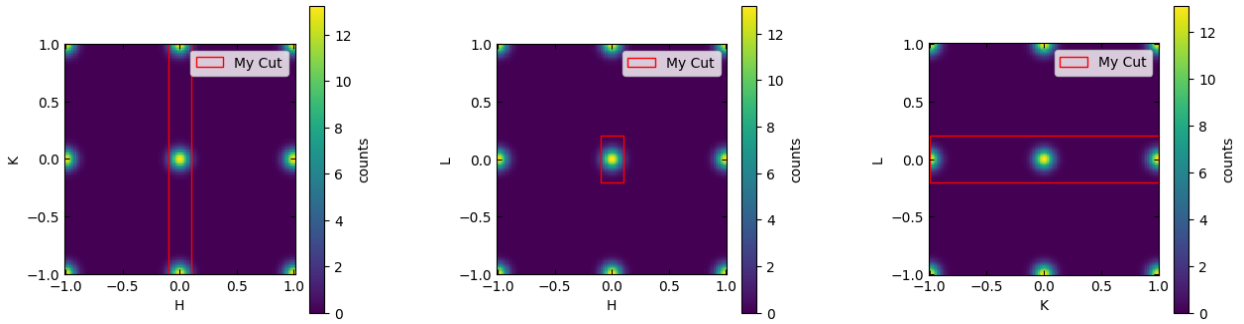
```
scissors.highlight_integration_window()
```




```
(<matplotlib.collections.QuadMesh at 0x7f863ab83e80>,
 <matplotlib.collections.QuadMesh at 0x7f8638a1a7a0>,
 <matplotlib.collections.QuadMesh at 0x7f8638a99990>)
```

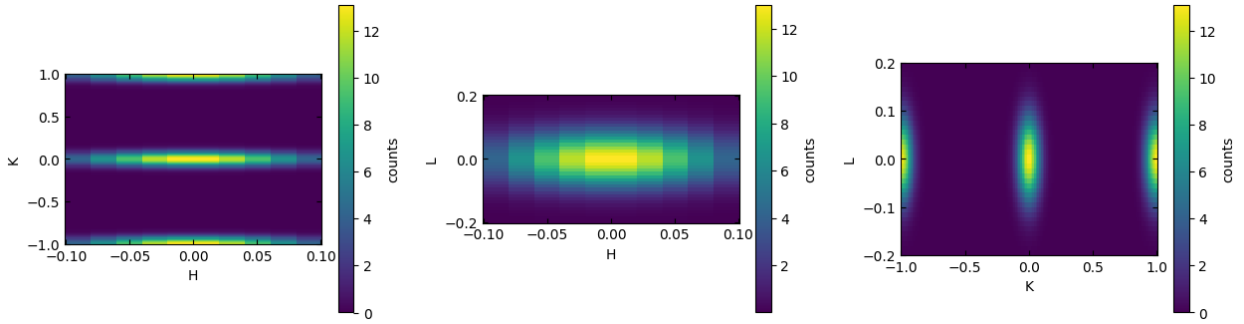
The `label` parameter allows you to specify a label for the highlighted region.

```
scissors.highlight_integration_window(label='My Cut')
```




```
(<matplotlib.collections.QuadMesh at 0x7f86384d8730>,
 <matplotlib.collections.QuadMesh at 0x7f863880a9e0>,
 <matplotlib.collections.QuadMesh at 0x7f863887dbd0>)
```

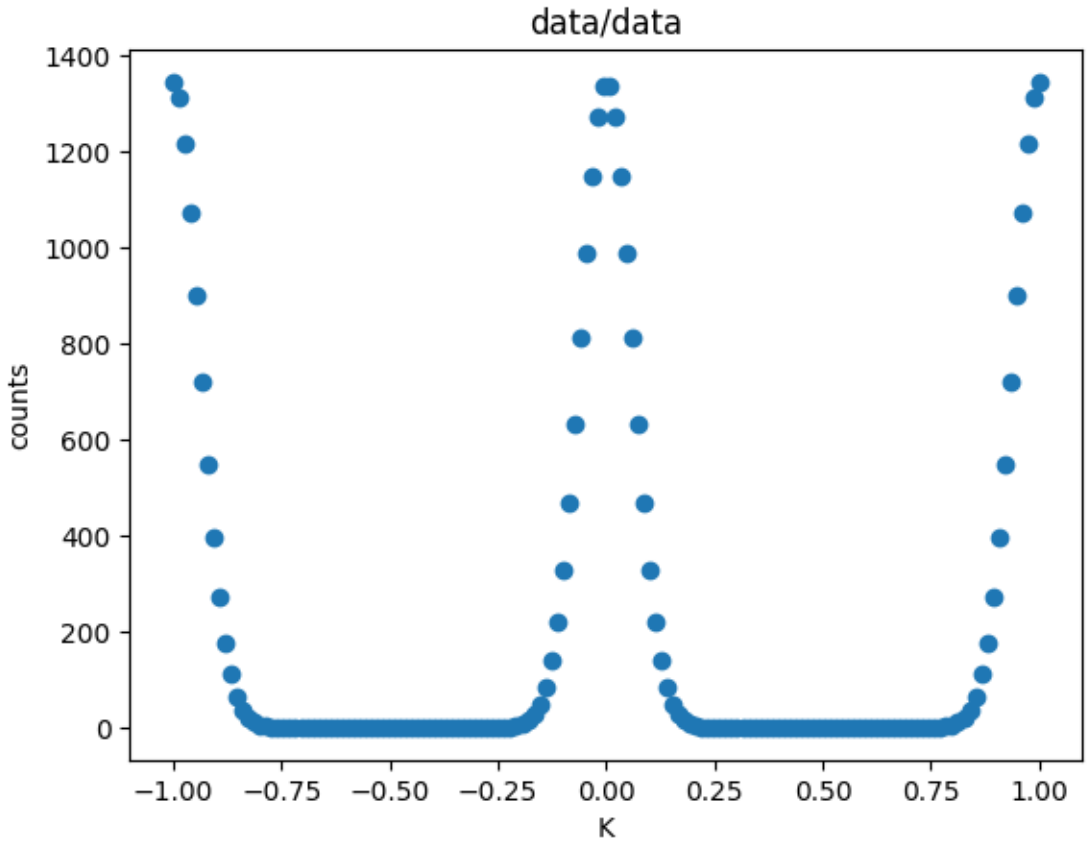
```
scissors.plot_integration_window()
```




```
(<matplotlib.collections.QuadMesh at 0x7f8636210ac0>,
 <matplotlib.collections.QuadMesh at 0x7f86362c57e0>,
 <matplotlib.collections.QuadMesh at 0x7f8636257a90>)
```

The `.plot()` method of the NXdata object can be used to generate a basic plot of the linecut.

```
linecut.plot()
```



2.1.3 Fitting linecuts using the lmfit package

Import functions

```
from nxs_analysis_tools.datareduction import load_data, Scissors
from nxs_analysis_tools.fitting import LinecutModel

from lmfit.models import GaussianModel, LinearModel
```

Load data

```
data = load_data('example_data/sample_name/15/example_hkli.nxs')
```

```
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
```

(continues on next page)

(continued from previous page)

```
  L = float64(200)
  counts = float64(100x150x200)
```

Perform linecuts

```
s = Scissors(data=data)
```

```
linecut = s.cut_data(center=(0,0,0), window=(0.1,0.5,0.1))
```

```
Linecut axis: K
Integrated axes: ['H', 'L']
```

The `LinecutModel` class

The workhorse of the fitting algorithm here is the `LinecutModel` class which wraps both the `Model` and `ModelResult` classes of the `lmfit` package, among other information.

```
lm = LinecutModel(data=linecut)
```

Create `lmfit` model

Use the `.set_model_components()` method to set the model to be used for fitting the linecut. The `model_components` parameter must be a `Model` or `list` of `Model` objects.

```
lm.set_model_components([GaussianModel(prefix='peak'), LinearModel(prefix='background')])
```

Set model constraints

Set constraints on the model using `.set_param_hint()`.

```
lm.set_param_hint('peakcenter', min=-0.1, max=0.1)
```

After the model and the hints have been specified, use the `.make_params()` method to initialize the parameters.

Initialize parameters

```
lm.make_params()
```

```
Parameters([('peakamplitude', <Parameter 'peakamplitude', value=1.0, bounds=[-inf:inf]>),
↪ ('peakcenter', <Parameter 'peakcenter', value=0.0, bounds=[-0.1:0.1]>), ('peaksigma',
↪<Parameter 'peaksigma', value=1.0, bounds=[0:inf]>), ('backgroundslope', <Parameter
↪'backgroundslope', value=1.0, bounds=[-inf:inf]>), ('backgroundintercept', <Parameter
↪'backgroundintercept', value=0.0, bounds=[-inf:inf]>), ('peakfwhm', <Parameter
↪'peakfwhm', value=2.35482, bounds=[-inf:inf], expr='2.3548200*peaksigma'>), (
↪'peakheight', <Parameter 'peakheight', value=0.3989423, bounds=[-inf:inf], expr='0.
↪3989423*peakamplitude/max(1e-15, peaksigma)'>), ('peakcorrlength', <Parameter
```

(continues on next page)

(continued from previous page)

```
↪'peakcorrlength', value=2.6682231793426188, bounds=[-inf:inf], expr='(2 * 3.
↪141592653589793) / peakfwhm'>)])
```

Perform initial guess

Use the `.guess()` method to perform an initial guess.

```
lm.guess()
```

The initial values of the parameters that result can be viewed using the `.print_initial_params()` method.

```
lm.print_initial_params()
```

```
peaksigma
        min: 0
peakfwhm
        expr: 2.3548200*peaksigma
peakheight
        expr: 0.3989423*peakamplitude/max(1e-15, peaksigma)
peakcenter
        min: -0.1
        max: 0.1
```

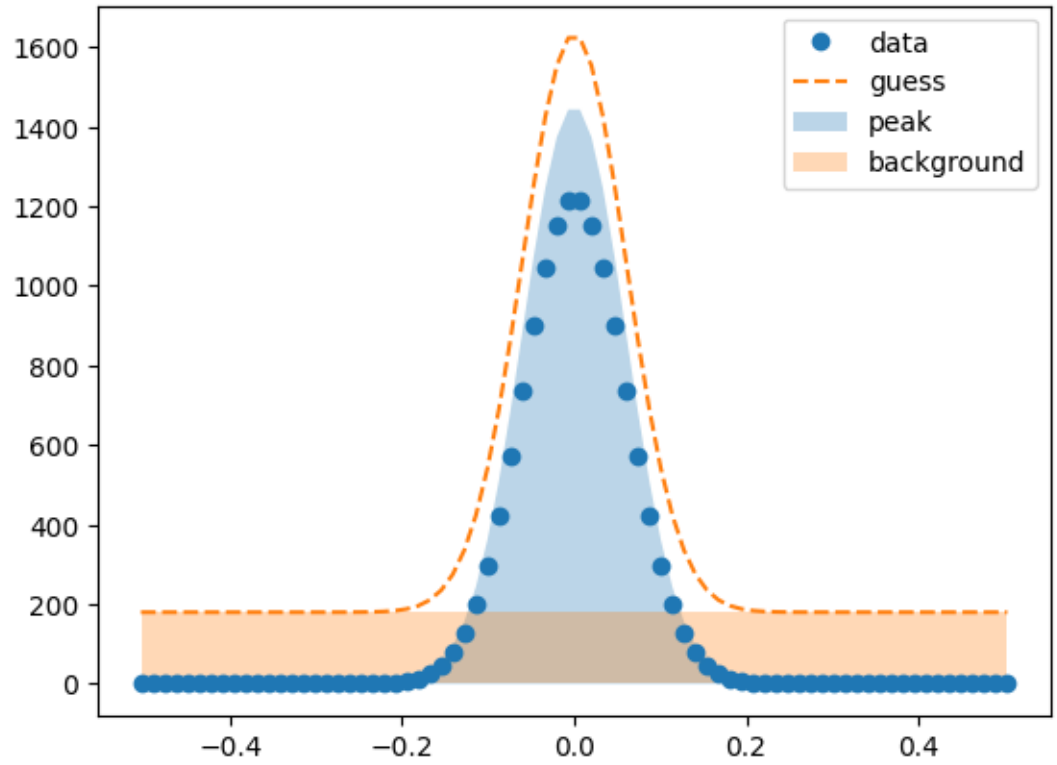
These can also be accessed through the individual `LinecutModel` objects.

```
lm.params
```

```
Parameters([('peakamplitude', <Parameter 'peakamplitude', value=219.88536567293792,␣
↪bounds=[-inf:inf]>), ('peakcenter', <Parameter 'peakcenter', value=0.0, bounds=[-
↪inf:inf]>), ('peaksigma', <Parameter 'peaksigma', value=0.06040268456375841, bounds=[0.
↪0:inf]>), ('backgroundslope', <Parameter 'backgroundslope', value=2.3853065274578794e-
↪14, bounds=[-inf:inf]>), ('backgroundintercept', <Parameter 'backgroundintercept',␣
↪value=180.0195329458017, bounds=[-inf:inf]>), ('peakfwhm', <Parameter 'peakfwhm',␣
↪value=0.14223744966442958, bounds=[-inf:inf], expr='2.3548200*peaksigma'>), (
↪'peakheight', <Parameter 'peakheight', value=1452.279383796392, bounds=[-inf:inf],␣
↪expr='0.3989423*peakamplitude/max(1e-15, peaksigma)'>), ('peakcorrlength', <Parameter
↪'peakcorrlength', value=44.17391708022779, bounds=[-inf:inf], expr='(2 * 3.
↪141592653589793) / peakfwhm'>)])
```

Visualize the initial guesses

```
lm.plot_initial_guess()
```

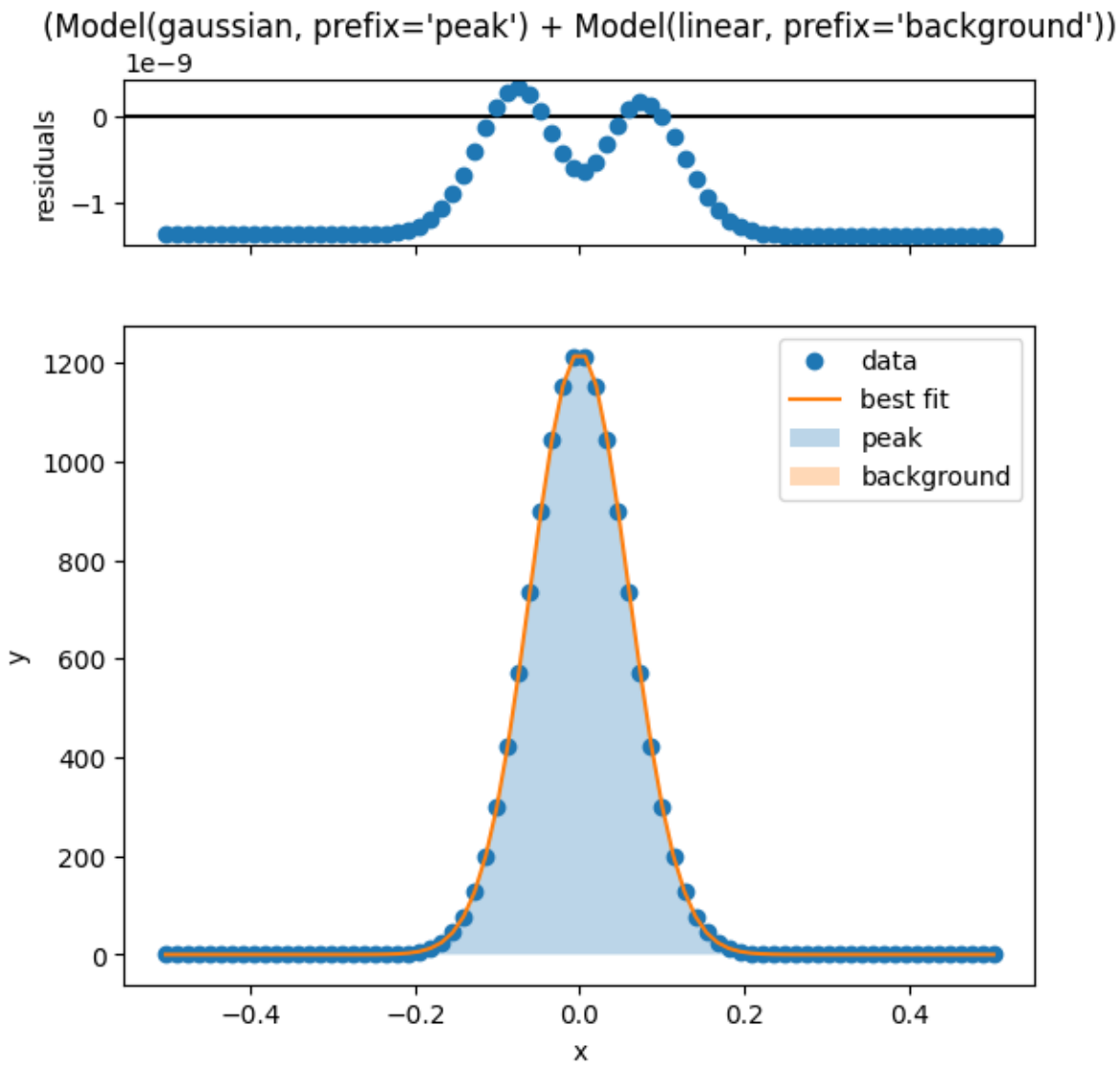


Perform the fit

```
lm.fit()
```

Visualize the fit

```
lm.plot_fit()
```




```
[[Model]]
    (Model(gaussian, prefix='peak') + Model(linear, prefix='background'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 25
    # data points      = 76
    # variables        = 5
    chi-square         = 1.0034e-16
    reduced chi-square = 1.4132e-18
    Akaike info crit   = -3118.82112
    Bayesian info crit = -3107.16745
    R-squared          = 1.00000000
##  Warning: uncertainties could not be estimated:
    backgroundslope:      at initial value
[[Variables]]
    peakamplitude:        183.644087 (init = 219.8854)
```

(continues on next page)

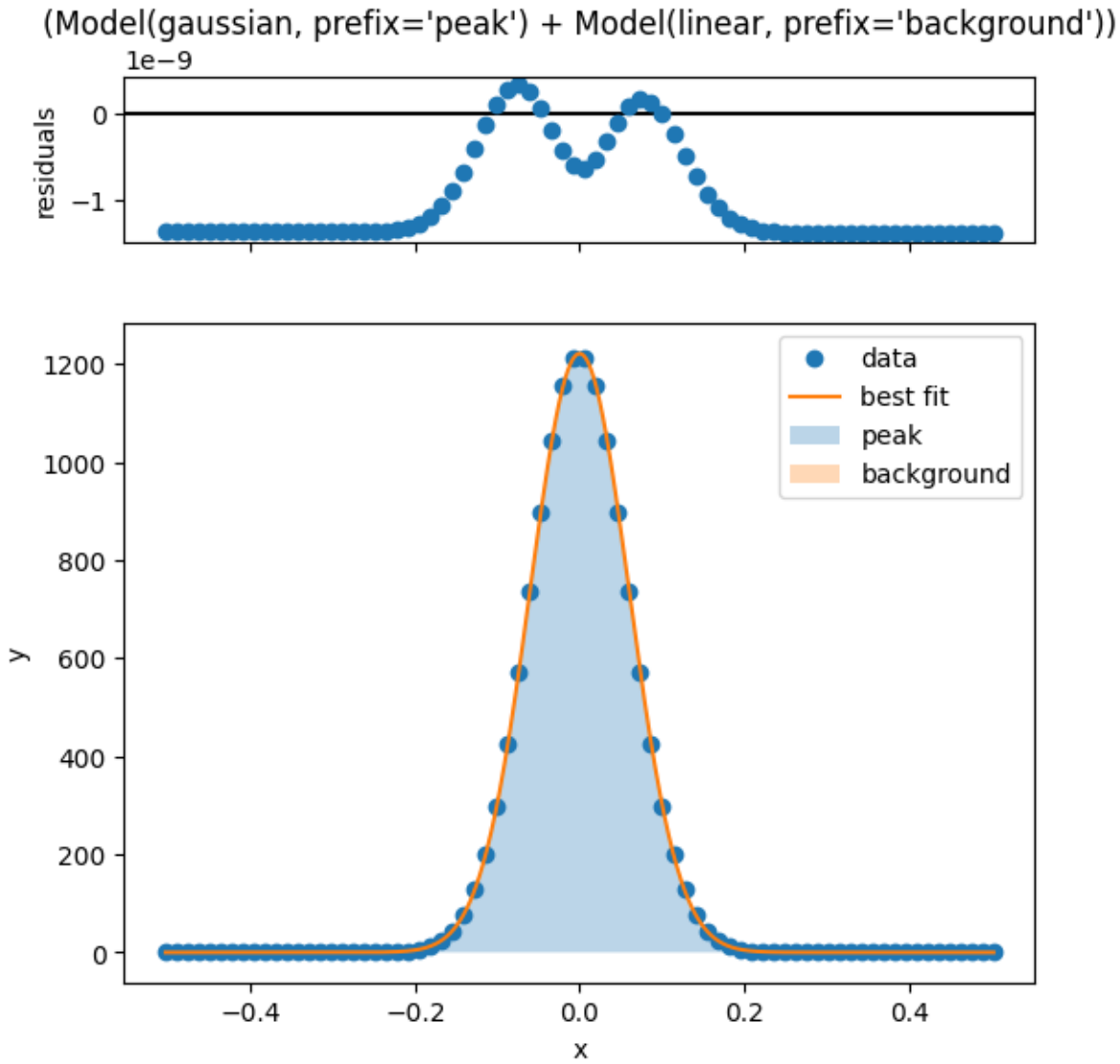
(continued from previous page)

```
    peakcenter:          -6.9385e-15 (init = 0)
    peaksigma:            0.06000000 (init = 0.06040268)
    backgroundslope:     -2.8140e-11 (init = 2.385307e-14)
    backgroundintercept: -1.3796e-09 (init = 180.0195)
    peakfwhm:             0.14128920 == '2.3548200*peaksigma'
    peakheight:           1221.05658 == '0.3989423*peakamplitude/max(1e-15, peaksigma)'
    peakcorrlength:       44.4703863 == '(2 * 3.141592653589793) / peakfwhm'
```

```
<Axes: xlabel='x', ylabel='y'>
```

The number of x values used to evaluate the fit curve can be set using the `numpoints` parameter, and any keyword arguments here will be passed to the `.plot()` method of the underlying `ModelResult` object.

```
lm.plot_fit(numpoints=1000)
```



```
[[Model]]
    (Model(gaussian, prefix='peak') + Model(linear, prefix='background'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 25
    # data points      = 76
    # variables        = 5
    chi-square         = 1.0034e-16
    reduced chi-square = 1.4132e-18
    Akaike info crit   = -3118.82112
    Bayesian info crit = -3107.16745
    R-squared          = 1.00000000
##  Warning: uncertainties could not be estimated:
    backgroundslope:      at initial value
[[Variables]]
    peakamplitude:        183.644087 (init = 219.8854)
    peakcenter:          -6.9385e-15 (init = 0)
    peaksigma:            0.06000000 (init = 0.06040268)
    backgroundslope:     -2.8140e-11 (init = 2.385307e-14)
    backgroundintercept: -1.3796e-09 (init = 180.0195)
    peakfwhm:             0.14128920 == '2.3548200*peaksigma'
    peakheight:           1221.05658 == '0.3989423*peakamplitude/max(1e-15, peaksigma)'
    peakcorrlength:       44.4703863 == '(2 * 3.141592653589793) / peakfwhm'
```

```
<Axes: xlabel='x', ylabel='y'>
```

Access the fit values

The x and y values of the fit are stored in the `.x` (identical to the raw x values) and the `.y_fit` attributes.

```
lm.x
```

```
array([-0.503356, -0.489933, -0.47651 , ...,  0.47651 ,  0.489933,
        0.503356])
```

```
lm.y_fit
```

```
array([-1.364786e-09, -1.361743e-09, -1.341593e-09, ..., -1.368411e-09,
       -1.389316e-09, -1.393114e-09])
```

The residuals are stored within the `ModelResult` class of the lmfit package, which is stored in the `.modelresult` attribute of the `LinecutModel` class.

```
lm.modelresult.residual
```

```
array([-1.367040e-09, -1.366048e-09, -1.366214e-09, ..., -1.393032e-09,
       -1.393621e-09, -1.395368e-09])
```

Viewing the fit report

```
lm.print_fit_report()
```

```
[[Model]]
    (Model(gaussian, prefix='peak') + Model(linear, prefix='background'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 25
    # data points      = 76
    # variables        = 5
    chi-square         = 1.0034e-16
    reduced chi-square = 1.4132e-18
    Akaike info crit   = -3118.82112
    Bayesian info crit = -3107.16745
    R-squared          = 1.00000000
##  Warning: uncertainties could not be estimated:
    backgroundslope:      at initial value
[[Variables]]
    peakamplitude:        183.644087 (init = 219.8854)
    peakcenter:          -6.9385e-15 (init = 0)
    peaksigma:            0.06000000 (init = 0.06040268)
    backgroundslope:     -2.8140e-11 (init = 2.385307e-14)
    backgroundintercept: -1.3796e-09 (init = 180.0195)
    peakfwhm:             0.14128920 == '2.3548200*peaksigma'
    peakheight:           1221.05658 == '0.3989423*peakamplitude/max(1e-15, peaksigma)'
    peakcorrlength:       44.4703863 == '(2 * 3.141592653589793) / peakfwhm'
```

2.1.4 Visualizing CHESS temperature dependent data

Import functions

```
import matplotlib.pyplot as plt
from nxs_analysis_tools import TempDependence
```

The `TempDependence` class

It is assumed that the file structure of the temperature dependent scan is as follows:

```
folder
└── subfolder
    ├── 15
    ├── 100
    └── 300
```

Here we create a `TempDependence` objecct called `sample` whose temperature folders are found in the path `'example_data/sample_name'`.

```
sample = TempDependence()
```

Loading data

Use the `load_datasets()` method to load the ".nxs" files. By default, all files ending with "hkli.nxs" are imported, but this can be changed using the `file_ending` parameter.

```
sample.load_datasets(folder='example_data/sample_name', file_ending="hkli.nxs")
```

```
Found temperature folders:
[0] 15
[1] 100
[2] 300
-----------------------------------------------
Loading 15 K indexed .nxs files...
Found example_data/sample_name/15/example_hkli.nxs
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
-----------------------------------------------
Loading 100 K indexed .nxs files...
Found example_data/sample_name/100/example_hkli.nxs
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
-----------------------------------------------
Loading 300 K indexed .nxs files...
Found example_data/sample_name/300/example_hkli.nxs
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
```

A subset of temperatures can be imported using the `temperatures_list` parameter. Temperatures can be listed here as numeric values (`[15,300]`) or as strings (`['15','300']`).

```
sample.load_datasets(folder='example_data/sample_name', temperatures_list=[15,300])
```

```
Found temperature folders:
[0] 15
[1] 100
[2] 300
-----------------------------------------------
Loading 15 K indexed .nxs files...
```

(continues on next page)

(continued from previous page)

```
Found example_data/sample_name/15/example_hkli.nxs
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
------------------------------------------------
Loading 300 K indexed .nxs files...
Found example_data/sample_name/300/example_hkli.nxs
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
```

Accessing the data

```
sample.load_datasets(folder='example_data/sample_name')
```

```
Found temperature folders:
[0] 15
[1] 100
[2] 300
------------------------------------------------
Loading 15 K indexed .nxs files...
Found example_data/sample_name/15/example_hkli.nxs
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
------------------------------------------------
Loading 100 K indexed .nxs files...
Found example_data/sample_name/100/example_hkli.nxs
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
------------------------------------------------
Loading 300 K indexed .nxs files...
Found example_data/sample_name/300/example_hkli.nxs
```

(continues on next page)

(continued from previous page)

```
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
```

The datasets are stored under the `.datasets` attribute of the `TempDependence` object.

```
sample.datasets
```

```
{'15': NXdata('data'), '100': NXdata('data'), '300': NXdata('data')}
```

Use square brackets to index the individual datasets in the `NXentry`. Each dataset is a `NXdata` object and possesses the corresponding attributes and methods.

```
sample.datasets['15']
```

```
NXdata('data')
```

A list of temperatures is stored in the `.temperatures` attribute of the `TempDependence` object.

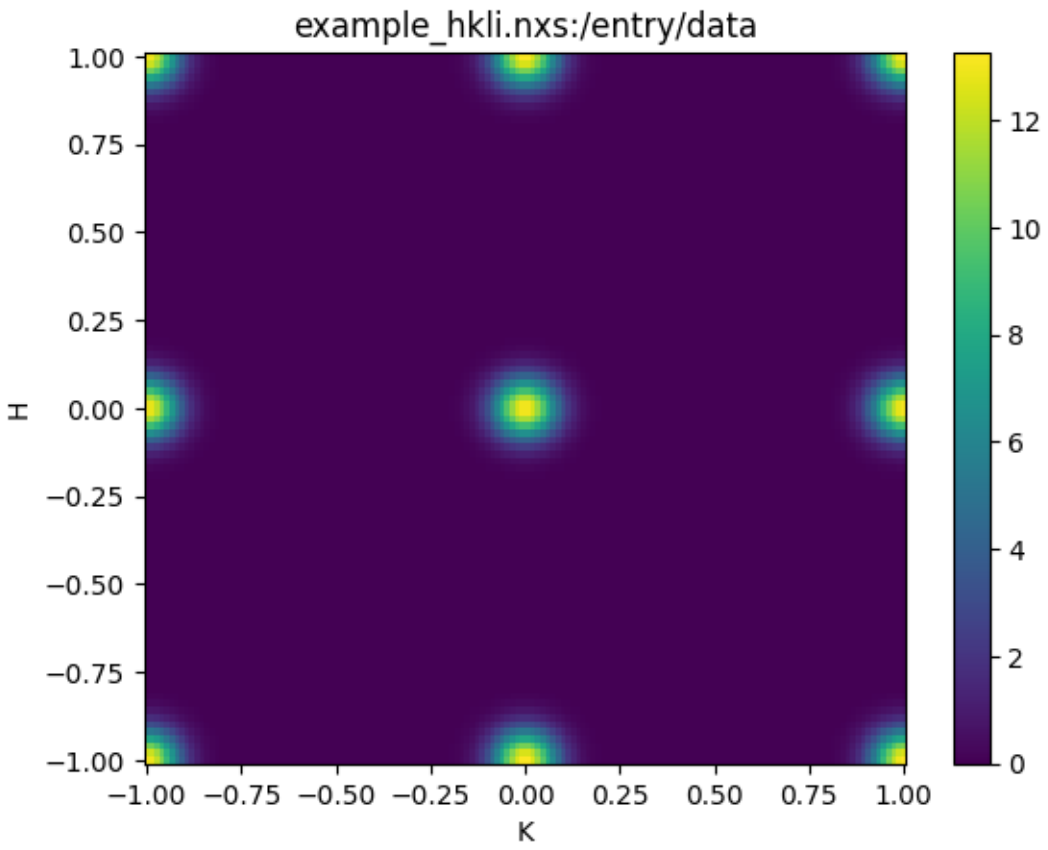
```
sample.temperatures
```

```
['15', '100', '300']
```

Visualizing the data

For example, each `NXdata` object has a `.plot()` method. Here we plot the L=0.0 plane.

```
sample.datasets['15'][:,:,0.0].plot()
```



Temperature dependent linecuts

Linecuts are performed using the `Scissors` class. An instance of the `Scissors` class is created for each temperature and stored in a dict attribute called `scissors`.

See documentation on the `Scissors` class for more information.

```
sample.scissors
```

```
{'15': <nxs_analysis_tools.datareduction.Scissors at 0x7fd01ad7fc40>,
 '100': <nxs_analysis_tools.datareduction.Scissors at 0x7fcfe40a8c70>,
 '300': <nxs_analysis_tools.datareduction.Scissors at 0x7fd01bee2200>}
```

The individual `Scissors` objects can be indexed using square brackets and the temperature.

```
sample.scissors['15']
```

```
<nxs_analysis_tools.datareduction.Scissors at 0x7fd01ad7fc40>
```

Batch linecuts are peformed using a method of the `TempDependence` class called `.cut_data()`, which internally calls the `.cut_data()` method on each of the `Scissors` objects at each temperature.

```
sample.cut_data(center=(0,0,0), window=(0.1,1,0.1))
```

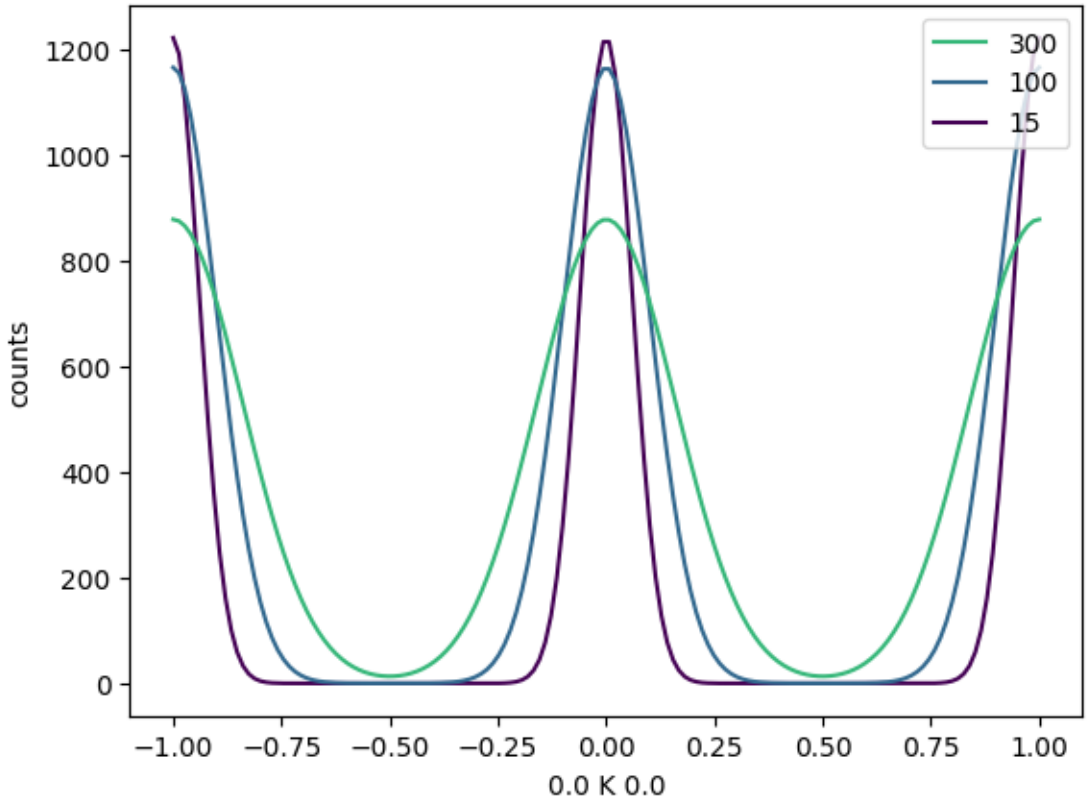
```
------------------------------
Cutting T = 15 K data...
Linecut axis: K
Integrated axes: ['H', 'L']
------------------------------
Cutting T = 100 K data...
Linecut axis: K
Integrated axes: ['H', 'L']
------------------------------
Cutting T = 300 K data...
Linecut axis: K
Integrated axes: ['H', 'L']
```

```
{'15': NXdata('data'), '100': NXdata('data'), '300': NXdata('data')}
```

Similarly, batch plotting of the linecuts can be achieved using the `.plot_linecuts()` method of the `TempDependence` object, which internally calls the `.plot_linecut()` method of the `Scissors` objects at each temperature.

```
sample.plot_linecuts()
```

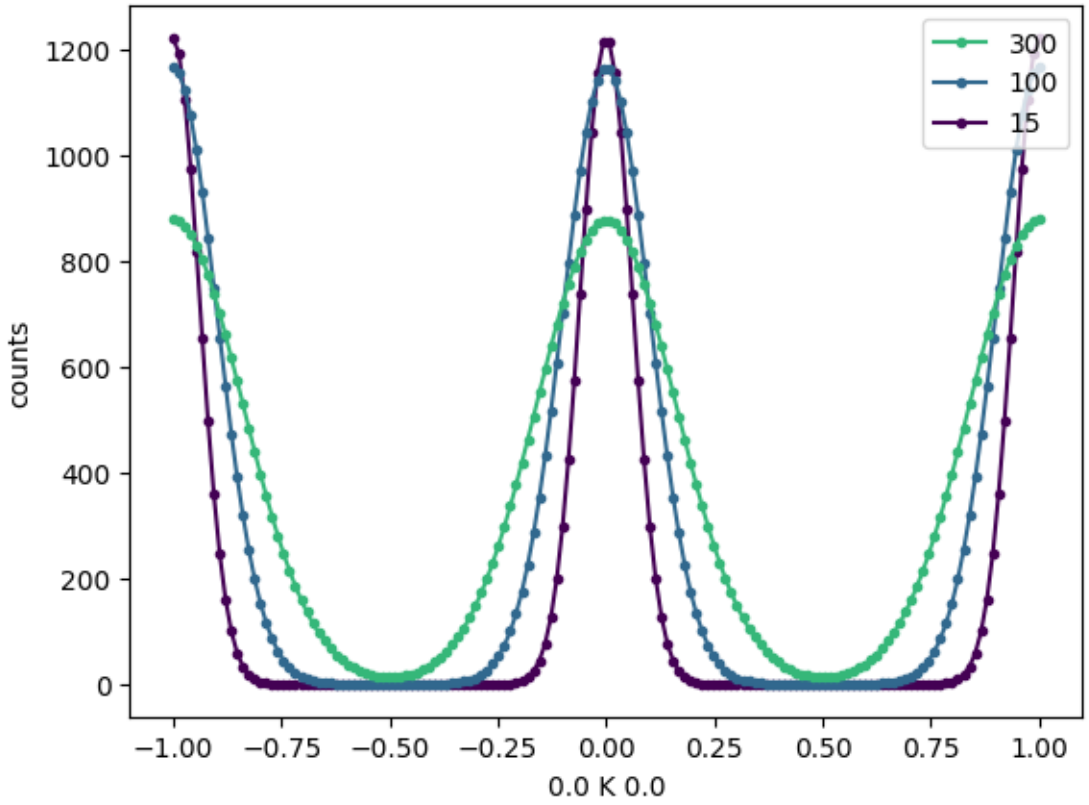
```
(<Figure size 640x480 with 1 Axes>,
 <Axes: xlabel='0.0 K 0.0', ylabel='counts'>)
```



Any keyword arguments are passed to a `matplotlib` function `ax.plot()` within `.plot_linecuts()`, so the usual `matplotlib` parameters can be used to change the formatting.

```
sample.plot_linecuts(linestyle='-', marker='.')
```

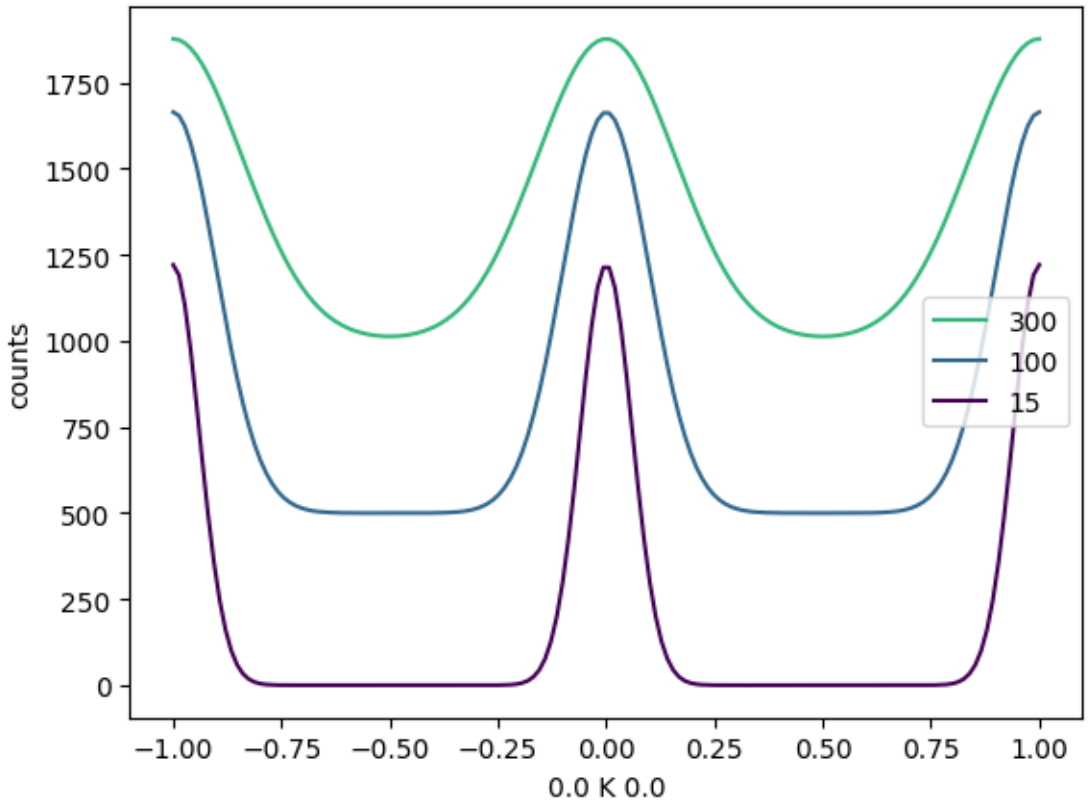
```
(<Figure size 640x480 with 1 Axes>,
 <Axes: xlabel='0.0 K 0.0', ylabel='counts'>)
```



You can also introduce a vertical offset using the `vertical_offset` parameter.

```
sample.plot_linecuts(vertical_offset=500)
```

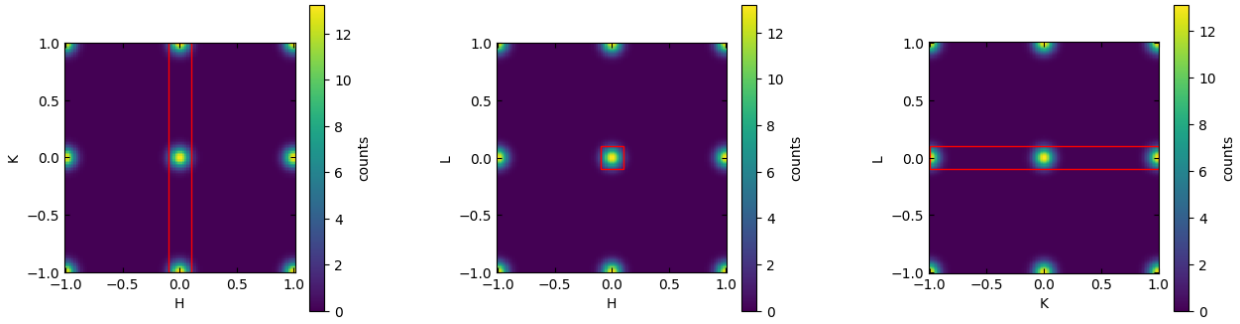
```
(<Figure size 640x480 with 1 Axes>,
 <Axes: xlabel='0.0 K 0.0', ylabel='counts'>)
```



Visualizing the integration window

To visualize where the integration was performed, use the `.highlight_integration_window()` method of the `Scissors` class.

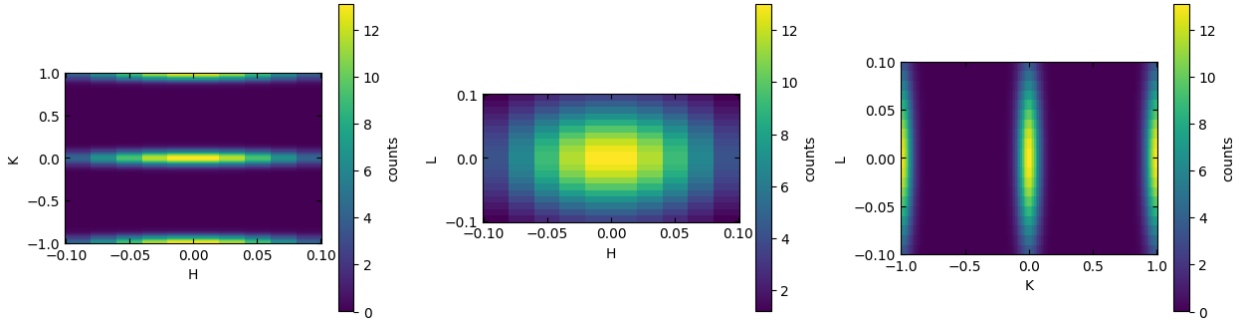
```
sample.scissors['15'].highlight_integration_window()
```




```
(<matplotlib.collections.QuadMesh at 0x7fcfe01f6800>,
 <matplotlib.collections.QuadMesh at 0x7fcfe0265660>,
 <matplotlib.collections.QuadMesh at 0x7fcfe029fa60>)
```

Similarly, to plot a heatmap of the integrated volume itself, use the `.plot_integration_window()` method of the `Scissors` class.

```
sample.scissors['15'].plot_integration_window()
```




```
(<matplotlib.collections.QuadMesh at 0x7fcfe03299c0>,
 <matplotlib.collections.QuadMesh at 0x7fcfdbd80130>,
 <matplotlib.collections.QuadMesh at 0x7fcfdbf038e0>)
```

2.1.5 Fitting CHESS temperature dependent linecuts

Import functions

```
from nxs_analysis_tools.datareduction import load_data, Scissors
from nxs_analysis_tools.chess import *

from lmfit.models import GaussianModel, LinearModel
```

Create TempDependece object and load data

```
sample = TempDependence()
```

```
sample.load_datasets(folder='example_data/sample_name')
```

```
Found temperature folders:
[0] 15
[1] 100
[2] 300
-------------------------------------------------
Loading 15 K indexed .nxs files...
Found example_data/sample_name/15/example_hkli.nxs
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
-------------------------------------------------
Loading 100 K indexed .nxs files...
```

(continues on next page)

(continued from previous page)

```
Found example_data/sample_name/100/example_hkli.nxs
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
-----------------------------------------------
Loading 300 K indexed .nxs files...
Found example_data/sample_name/300/example_hkli.nxs
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
```

Perform linecuts

```
sample.cut_data(center=(0,0,0), window=(0.1,0.5,0.1))
```

```
--------------------------------
Cutting T = 15 K data...
Linecut axis: K
Integrated axes: ['H', 'L']
--------------------------------
Cutting T = 100 K data...
Linecut axis: K
Integrated axes: ['H', 'L']
--------------------------------
Cutting T = 300 K data...
Linecut axis: K
Integrated axes: ['H', 'L']
```

```
{'15': NXdata('data'), '100': NXdata('data'), '300': NXdata('data')}
```

The `LinecutModel` class

Each `TempDependence` object has a `LinecutModel` for each temperature, stored in the `.linecutmodels` attribute.

```
sample.linecutmodels
```

```
{'15': <nxs_analysis_tools.fitting.LinecutModel at 0x7f90be581300>,
 '100': <nxs_analysis_tools.fitting.LinecutModel at 0x7f90866fa260>,
 '300': <nxs_analysis_tools.fitting.LinecutModel at 0x7f90866fa230>}
```

Create `lmfit` model

Use the `.set_model_components()` method to set the model to be used for fitting the linecut. The `model_components` parameter must be a `Model` or `list` of `Model` objects.

```
sample.set_model_components([GaussianModel(prefix='peak'), LinearModel(prefix='background
↪')])
```

Set model constraints

Set constraints on the model using `.set_param_hint()`.

```
sample.set_param_hint('peakcenter', min=-0.1, max=0.1)
```

After the model and the hints have been specified, use the `.make_params()` method to initialize the parameters.

The constraints can be different for each temperature - for example, here we set the constraints of the Gaussian sigma at 300 K only.

```
sample.linecutmodels['300'].set_param_hint('peaksigma', min=-1, max=1)
```

Initialize parameters

```
sample.make_params()
```

Perform initial guess

Use the `.guess()` method to perform an initial guess.

```
sample.guess()
```

The initial values of the parameters that result can be viewed using the `.print_initial_params()` method.

```
sample.print_initial_params()
```

```
peaksigma
        min: -1
        max: 1
peakfwhm
        expr: 2.3548200*peaksigma
peakheight
        expr: 0.3989423*peakamplitude/max(1e-15, peaksigma)
peakcenter
        min: -0.1
        max: 0.1
peaksigma
        min: -1
        max: 1
peakfwhm
        expr: 2.3548200*peaksigma
```

(continues on next page)

(continued from previous page)

```
peakheight
        expr: 0.3989423*peakamplitude/max(1e-15, peaksigma)
peakcenter
        min: -0.1
        max: 0.1
peaksigma
        min: -1
        max: 1
peakfwhm
        expr: 2.3548200*peaksigma
peakheight
        expr: 0.3989423*peakamplitude/max(1e-15, peaksigma)
peakcenter
        min: -0.1
        max: 0.1
```

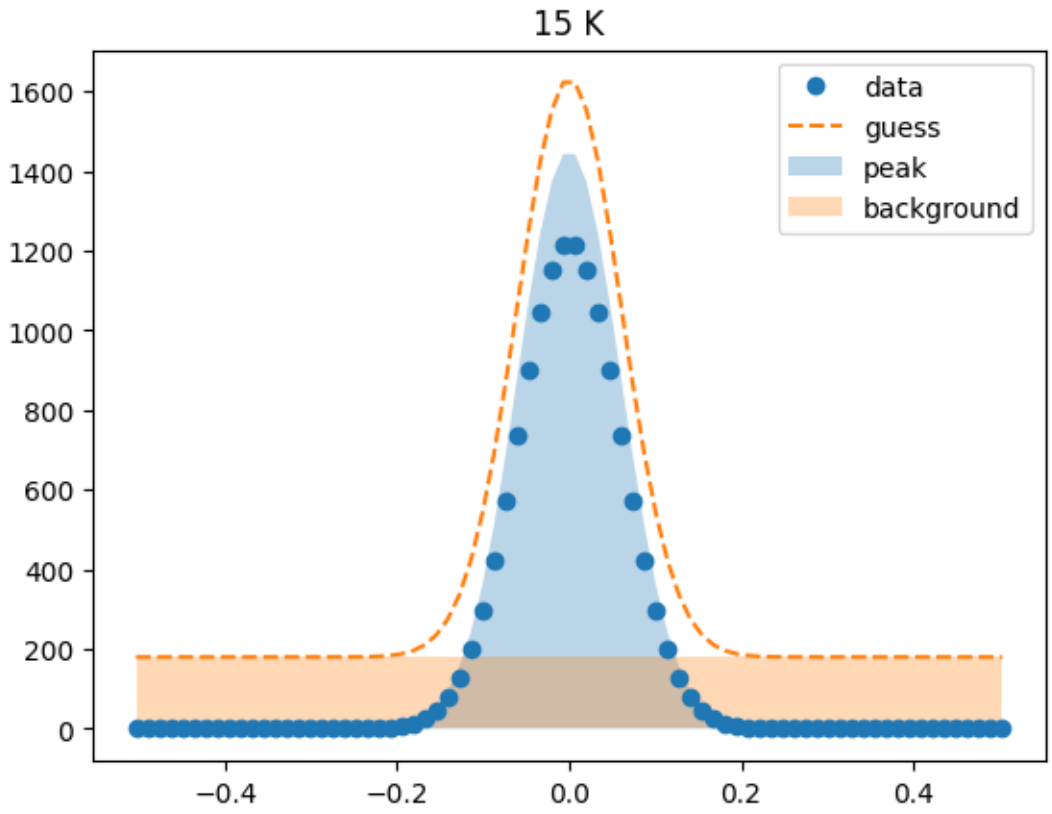
These can also be accessed through the individual `LinecutModel` objects.

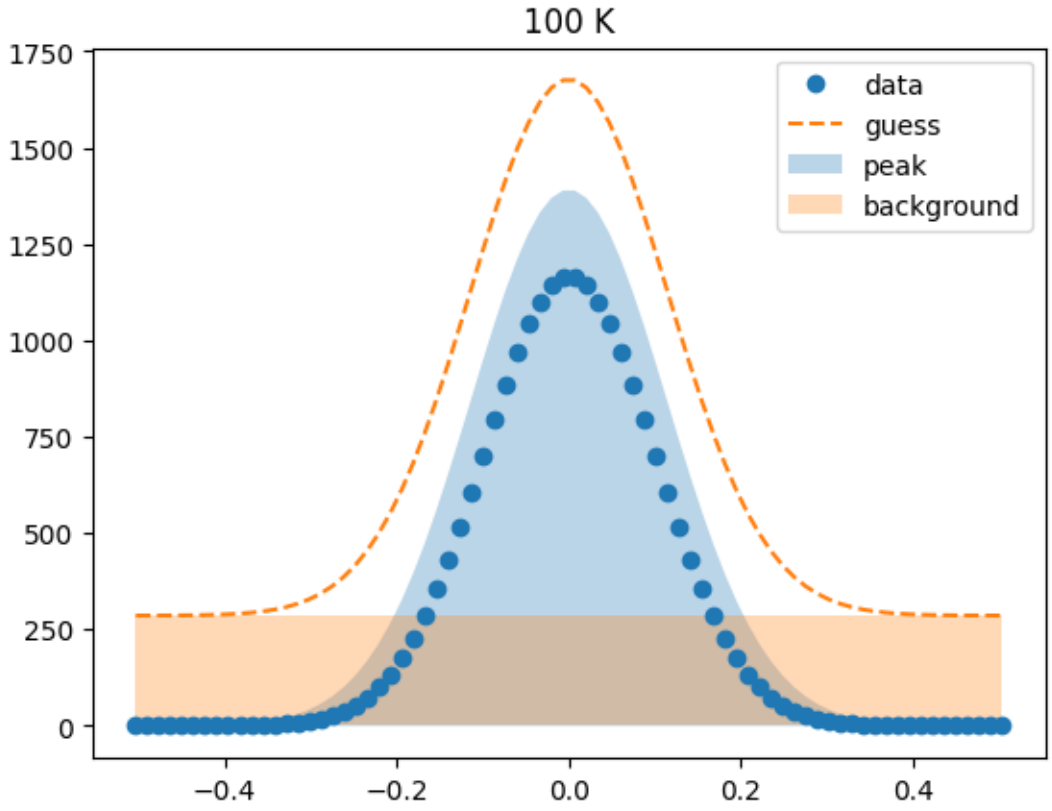
```
sample.linecutmodels['15'].params
```

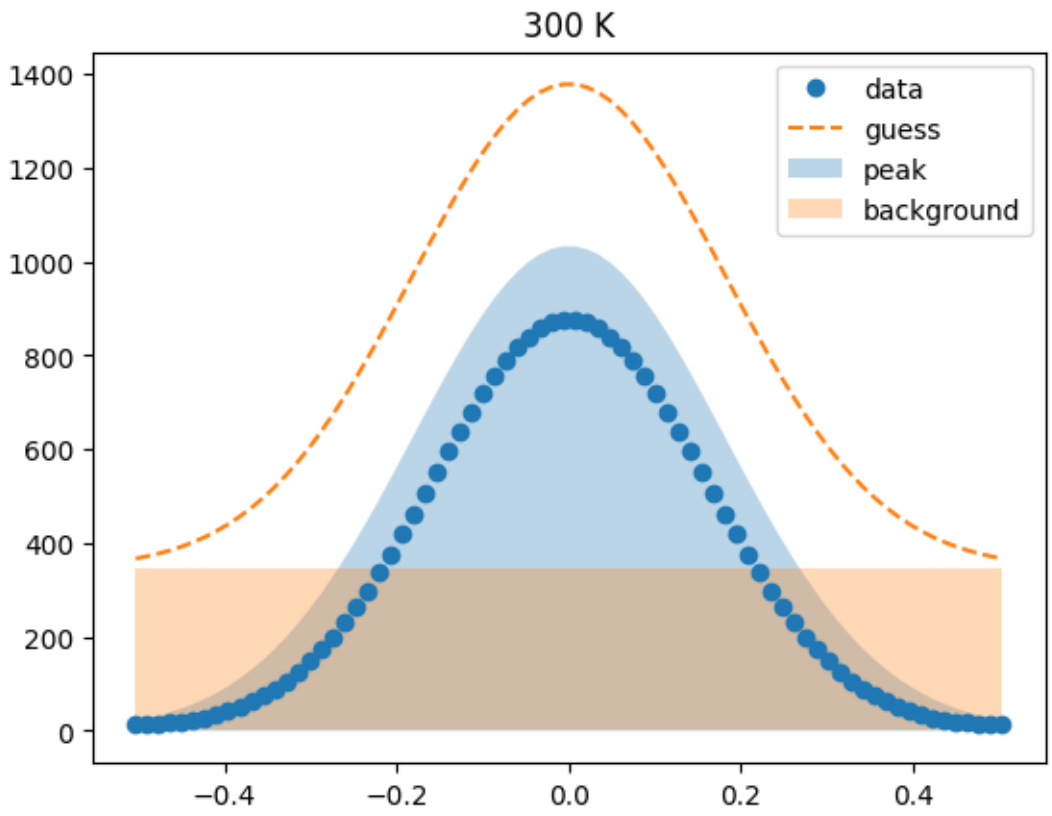
```
Parameters([('peakamplitude', <Parameter 'peakamplitude', value=219.88536567293792,␣
↪bounds=[-inf:inf]>), ('peakcenter', <Parameter 'peakcenter', value=0.0, bounds=[-
↪inf:inf]>), ('peaksigma', <Parameter 'peaksigma', value=0.06040268456375841, bounds=[0.
↪0:1]>), ('backgroundslope', <Parameter 'backgroundslope', value=2.3853065274578794e-14,
↪ bounds=[-inf:inf]>), ('backgroundintercept', <Parameter 'backgroundintercept',␣
↪value=180.0195329458017, bounds=[-inf:inf]>), ('peakfwhm', <Parameter 'peakfwhm',␣
↪value=0.14223744966442958, bounds=[-inf:inf], expr='2.3548200*peaksigma'>), (
↪'peakheight', <Parameter 'peakheight', value=1452.279383796392, bounds=[-inf:inf],␣
↪expr='0.3989423*peakamplitude/max(1e-15, peaksigma)'>), ('peakcorrlength', <Parameter
↪'peakcorrlength', value=44.17391708022779, bounds=[-inf:inf], expr='(2 * 3.
↪141592653589793) / peakfwhm'>)])
```

Visualize the initial guesses

```
sample.plot_initial_guess()
```


15 K
data
guess
peak
background
1600
1400
1200
1000
800
600
400
200
0
−0.4
−0.2
0.0
0.2
0.4


100 K
data
guess
peak
background
1750
1500
1250
1000
750
500
250
0
−0.4
−0.2
0.0
0.2
0.4



Perform the fit

```
sample.fit()
```

```
Fitting 15 K  data...
Done.
Fitting 100 K  data...
Done.
Fitting 300 K  data...
Done.
Fits completed.
```

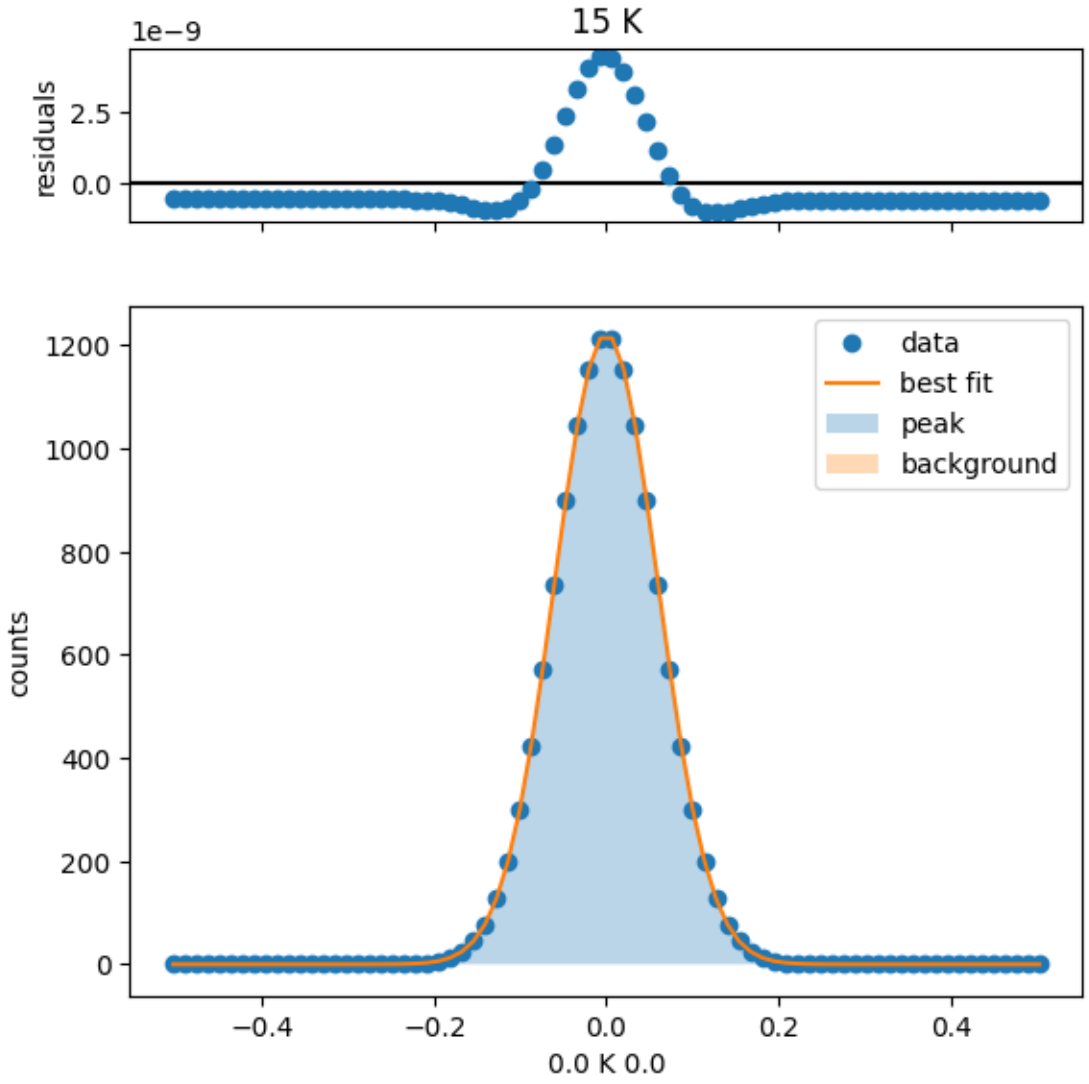
Visualize all fits

The `.plot_fit` function prints fit results for all temperatures. The parameter `fit_report` (default `True`) determines whether the fit report is also printed.

Additionally, an optional Markdown heading can be displayed using the `mdheadings` parameter (default `False`).

```
sample.plot_fit(mdheadings=True)
```

15 K Fit Results




```
[[Model]]
    (Model(gaussian, prefix='peak') + Model(linear, prefix='background'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 25
    # data points      = 76
    # variables        = 5
    chi-square         = 1.3162e-16
    reduced chi-square = 1.8539e-18
    Akaike info crit   = -3098.19567
    Bayesian info crit = -3086.54200
    R-squared          = 1.00000000
##  Warning: uncertainties could not be estimated:
    backgroundslope:      at initial value
[[Variables]]
```

(continues on next page)

(continued from previous page)

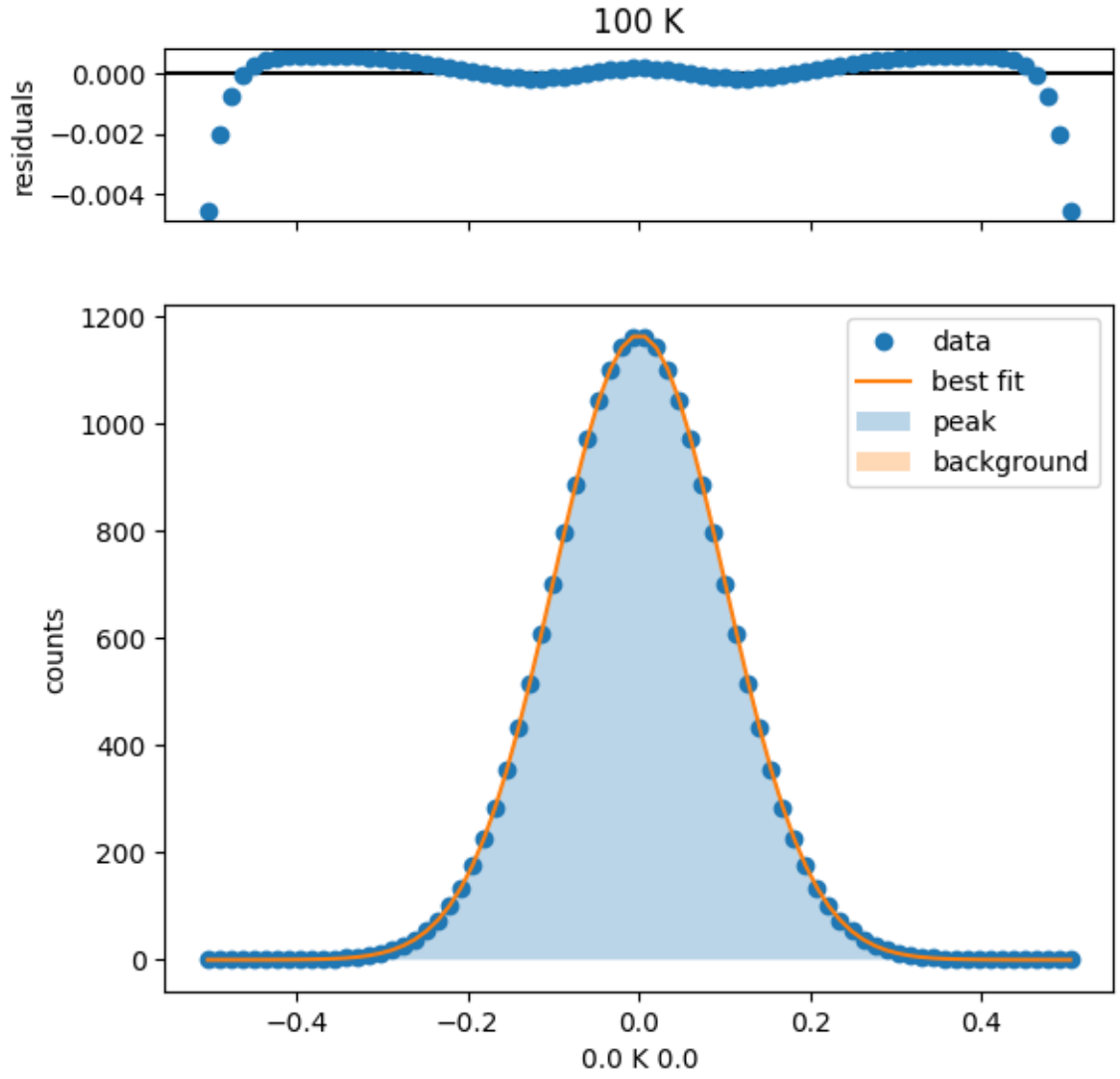
```
    peakamplitude:        183.644087 (init = 219.8854)
    peakcenter:          -9.7004e-15 (init = 0)
    peaksigma:            0.06000000 (init = 0.06040268)
    backgroundslope:     -5.2966e-11 (init = 2.385307e-14)
    backgroundintercept: -5.6044e-10 (init = 180.0195)
    peakfwhm:             0.14128920 == '2.3548200*peaksigma'
    peakheight:           1221.05658 == '0.3989423*peakamplitude/max(1e-15, peaksigma)'
    peakcorrlength:       44.4703863 == '(2 * 3.141592653589793) / peakfwhm'
[[Model]]
    (Model(gaussian, prefix='peak') + Model(linear, prefix='background'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 34
    # data points      = 76
    # variables        = 5
    chi-square         = 5.9182e-05
    reduced chi-square = 8.3356e-07
    Akaike info crit   = -1058.98702
    Bayesian info crit = -1047.33335
    R-squared          = 1.00000000
[[Variables]]
    peakamplitude:        291.982900 +/- 1.1230e-04 (0.00%) (init = 397.8052)
    peakcenter:          -3.6249e-12 +/- 3.0297e-08 (835795.34%) (init = -6.167906e-18)
    peaksigma:            0.09999993 +/- 3.5765e-08 (0.00%) (init = 0.114094)
    backgroundslope:      3.5175e-14 +/- 3.7545e-04 (1067379877973.20%) (init = 3.
↪517481e-14)
    backgroundintercept:  5.6407e-04 +/- 1.5350e-04 (27.21%) (init = 286.2205)
    peakfwhm:             0.23548184 +/- 8.4221e-08 (0.00%) == '2.3548200*peaksigma'
    peakheight:           1164.84410 +/- 3.2639e-04 (0.00%) == '0.3989423*peakamplitude/
↪max(1e-15, peaksigma)'
    peakcorrlength:       26.6822500 +/- 9.5430e-06 (0.00%) == '(2 * 3.141592653589793) /
↪ peakfwhm'
[[Correlations]] (unreported correlations are < 0.100)
    C(peakamplitude, backgroundintercept)   = -0.7268
    C(peakamplitude, peaksigma)             = +0.7173
    C(peaksigma, backgroundintercept)       = -0.5215
    C(backgroundslope, backgroundintercept) = -0.1658
    C(peakamplitude, backgroundslope)       = +0.1205
[[Model]]
    (Model(gaussian, prefix='peak') + Model(linear, prefix='background'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 31
    # data points      = 76
    # variables        = 5
    chi-square         = 85.5499422
    reduced chi-square = 1.20492876
    Akaike info crit   = 18.9958908
    Bayesian info crit = 30.6495575
    R-squared          = 0.99998814
##  Warning: uncertainties could not be estimated:
    peakcenter:           at initial value
```

(continues on next page)

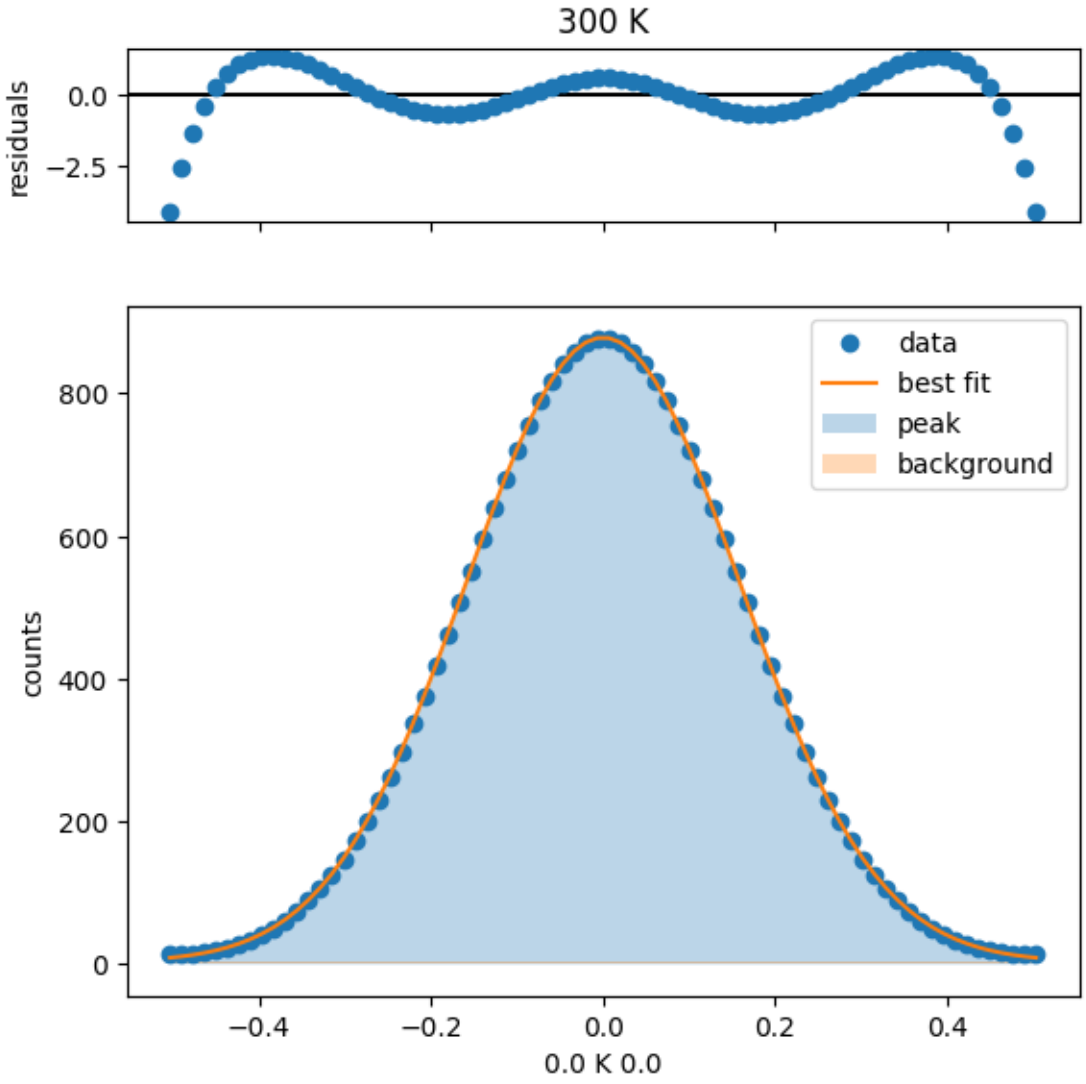
(continued from previous page)

```
    backgroundslope:      at initial value
[[Variables]]
    peakamplitude:        349.382304 (init = 469.3782)
    peakcenter:          -4.2492e-10 (init = -1.982541e-17)
    peaksigma:            0.15931867 (init = 0.1812081)
    backgroundslope:      1.1764e-14 (init = 1.176368e-14)
    backgroundintercept:  3.23456528 (init = 345.2546)
    peakfwhm:             0.37516679 == '2.3548200*peaksigma'
    peakheight:           874.871598 == '0.3989423*peakamplitude/max(1e-15, peaksigma)'
    peakcorrlength:       16.7477118 == '(2 * 3.141592653589793) / peakfwhm'
```

100 K Fit Results



300 K Fit Results



Access the fit values

The x and y values of the fit are stored in the `.x` (identical to the raw x values) and the `.y_fit` attributes.

```
sample.linecutmodels['15'].x
```

```
array([-0.503356, -0.489933, -0.47651 , ...,  0.47651 ,  0.489933,
        0.503356])
```

```
sample.linecutmodels['15'].y_fit
```

```
array([-5.331438e-10, -5.304346e-10, -5.106177e-10, ..., -5.610952e-10,
       -5.823341e-10, -5.864651e-10])
```

The residuals are stored within the `ModelResult` class of the lmfit package, which is stored in the `.modelresult` attribute of the `LinecutModel` class.

```
sample.linecutmodels['15'].modelresult.residual
```

```
array([-5.353978e-10, -5.347390e-10, -5.352385e-10, ..., -5.857160e-10,
       -5.866385e-10, -5.887192e-10])
```

Viewing the fit report

Use the `.print_fit_report()` method to print out the fit report for all temperatures.

```
sample.print_fit_report()
```

```
[[[15 K Fit Report]]]
[[Model]]
    (Model(gaussian, prefix='peak') + Model(linear, prefix='background'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 25
    # data points      = 76
    # variables        = 5
    chi-square         = 1.3162e-16
    reduced chi-square = 1.8539e-18
    Akaike info crit   = -3098.19567
    Bayesian info crit = -3086.54200
    R-squared          = 1.00000000
##  Warning: uncertainties could not be estimated:
    backgroundslope:      at initial value
[[Variables]]
    peakamplitude:        183.644087 (init = 219.8854)
    peakcenter:          -9.7004e-15 (init = 0)
    peaksigma:            0.06000000 (init = 0.06040268)
    backgroundslope:     -5.2966e-11 (init = 2.385307e-14)
    backgroundintercept: -5.6044e-10 (init = 180.0195)
    peakfwhm:             0.14128920 == '2.3548200*peaksigma'
    peakheight:           1221.05658 == '0.3989423*peakamplitude/max(1e-15, peaksigma)'
    peakcorrlength:       44.4703863 == '(2 * 3.141592653589793) / peakfwhm'
[[[100 K Fit Report]]]
[[Model]]
    (Model(gaussian, prefix='peak') + Model(linear, prefix='background'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 34
    # data points      = 76
    # variables        = 5
    chi-square         = 5.9182e-05
    reduced chi-square = 8.3356e-07
    Akaike info crit   = -1058.98702
    Bayesian info crit = -1047.33335
    R-squared          = 1.00000000
[[Variables]]
```

(continues on next page)

(continued from previous page)

```
    peakamplitude:        291.982900 +/- 1.1230e-04 (0.00%) (init = 397.8052)
    peakcenter:          -3.6249e-12 +/- 3.0297e-08 (835795.34%) (init = -6.167906e-18)
    peaksigma:            0.09999993 +/- 3.5765e-08 (0.00%) (init = 0.114094)
    backgroundslope:      3.5175e-14 +/- 3.7545e-04 (1067379877973.20%) (init = 3.
↪517481e-14)
    backgroundintercept:  5.6407e-04 +/- 1.5350e-04 (27.21%) (init = 286.2205)
    peakfwhm:             0.23548184 +/- 8.4221e-08 (0.00%) == '2.3548200*peaksigma'
    peakheight:           1164.84410 +/- 3.2639e-04 (0.00%) == '0.3989423*peakamplitude/
↪max(1e-15, peaksigma)'
    peakcorrlength:       26.6822500 +/- 9.5430e-06 (0.00%) == '(2 * 3.141592653589793) /
↪ peakfwhm'
[[Correlations]] (unreported correlations are < 0.100)
    C(peakamplitude, backgroundintercept)   = -0.7268
    C(peakamplitude, peaksigma)             = +0.7173
    C(peaksigma, backgroundintercept)       = -0.5215
    C(backgroundslope, backgroundintercept) = -0.1658
    C(peakamplitude, backgroundslope)       = +0.1205
[[[300 K Fit Report]]]
[[Model]]
    (Model(gaussian, prefix='peak') + Model(linear, prefix='background'))
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 31
    # data points      = 76
    # variables        = 5
    chi-square         = 85.5499422
    reduced chi-square = 1.20492876
    Akaike info crit   = 18.9958908
    Bayesian info crit = 30.6495575
    R-squared          = 0.99998814
##  Warning: uncertainties could not be estimated:
    peakcenter:           at initial value
    backgroundslope:      at initial value
[[Variables]]
    peakamplitude:        349.382304 (init = 469.3782)
    peakcenter:          -4.2492e-10 (init = -1.982541e-17)
    peaksigma:            0.15931867 (init = 0.1812081)
    backgroundslope:      1.1764e-14 (init = 1.176368e-14)
    backgroundintercept:  3.23456528 (init = 345.2546)
    peakfwhm:             0.37516679 == '2.3548200*peaksigma'
    peakheight:           874.871598 == '0.3989423*peakamplitude/max(1e-15, peaksigma)'
    peakcorrlength:       16.7477118 == '(2 * 3.141592653589793) / peakfwhm'
```

To obtain a fit report for just one temperature, use the `.fit_report()` method of the `ModelResult` object stored in the `.linecutmodels` attr

Or, to access a single temperature fit report:

```
sample.linecutmodels['15'].print_fit_report()
```

```
[[Model]]
    (Model(gaussian, prefix='peak') + Model(linear, prefix='background'))
```

(continues on next page)

(continued from previous page)

```
[[Fit Statistics]]
    # fitting method   = leastsq
    # function evals   = 25
    # data points      = 76
    # variables        = 5
    chi-square         = 1.3162e-16
    reduced chi-square = 1.8539e-18
    Akaike info crit   = -3098.19567
    Bayesian info crit = -3086.54200
    R-squared          = 1.00000000
##  Warning: uncertainties could not be estimated:
    backgroundslope:      at initial value
[[Variables]]
    peakamplitude:        183.644087 (init = 219.8854)
    peakcenter:          -9.7004e-15 (init = 0)
    peaksigma:            0.06000000 (init = 0.06040268)
    backgroundslope:     -5.2966e-11 (init = 2.385307e-14)
    backgroundintercept: -5.6044e-10 (init = 180.0195)
    peakfwhm:             0.14128920 == '2.3548200*peaksigma'
    peakheight:           1221.05658 == '0.3989423*peakamplitude/max(1e-15, peaksigma)'
    peakcorrlength:       44.4703863 == '(2 * 3.141592653589793) / peakfwhm'
```

2.1.6 Symmetrizing data using the Symmetrizer classes

Import functions

```
from nxs_analysis_tools import load_data, plot_slice
from nxs_analysis_tools.pairdistribution import Symmetrizer3D, Symmetrizer2D
```

Load data

```
data = load_data('example_data/pairdistribution_data/test_hkli.nxs')
```

```
data:NXdata
  @axes = ['H', 'K', 'L']
  @signal = 'counts'
  H = float64(100)
  K = float64(150)
  L = float64(200)
  counts = float64(100x150x200)
```

The `Symmetrizer2D` class

A `Symmetrizer2D` object can be initialized by defining a `theta_min` and `theta_max` representing the range of angles (clockwise relative the y-axis) which define the region to use for symmetrization.

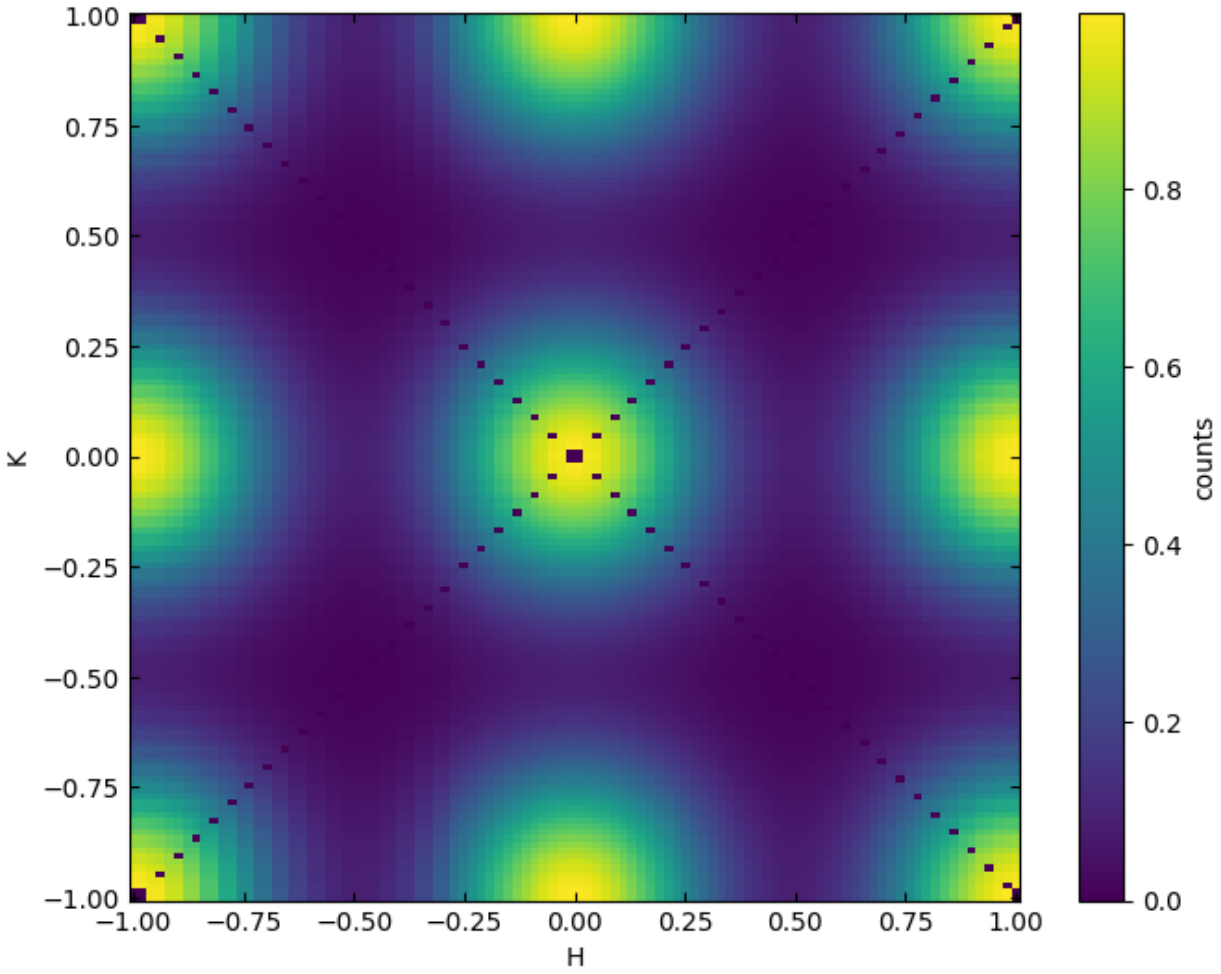
```
s2d = Symmetrizer2D(theta_min=0, theta_max=45)
```

Because data is not specified, the `Symmetrizer2D` object can be applied to any `NXdata` object. Perform the symmetrization using the `.symmetrize_2d()` method.

```
symmetrized_data = s2d.symmetrize_2d(data[:,:,0.0])
```

```
plot_slice(symmetrized_data)
```

```
<matplotlib.collections.QuadMesh at 0x7f1deea3ead0>
```

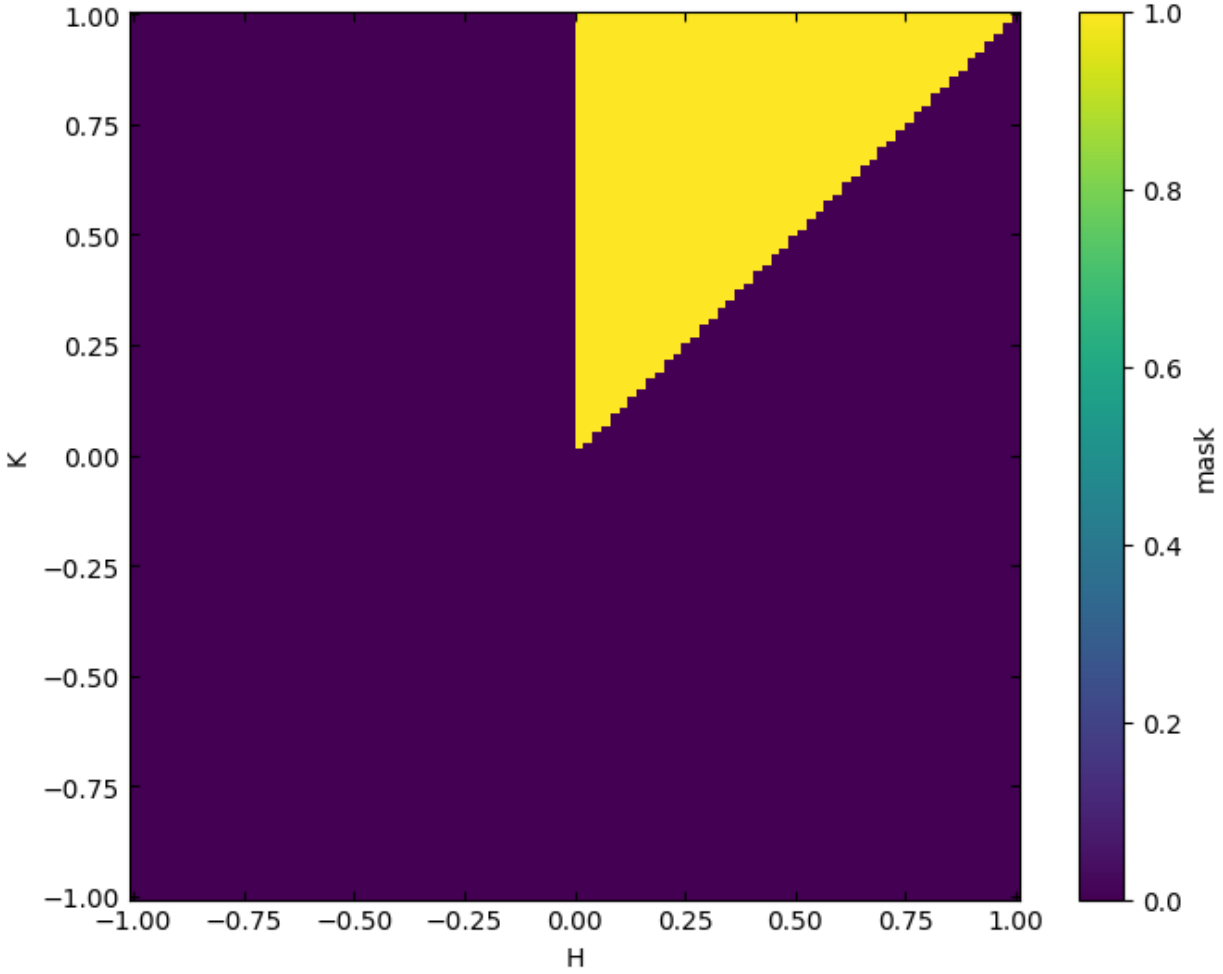


Viewing the symmetrization mask

The symmetrization mask for the most recent symmetrized dataset is stored in `symmetrization_mask`

```
plot_slice(data=s2d.symmetrization_mask)
```

```
<matplotlib.collections.QuadMesh at 0x7f1df0bfac50>
```

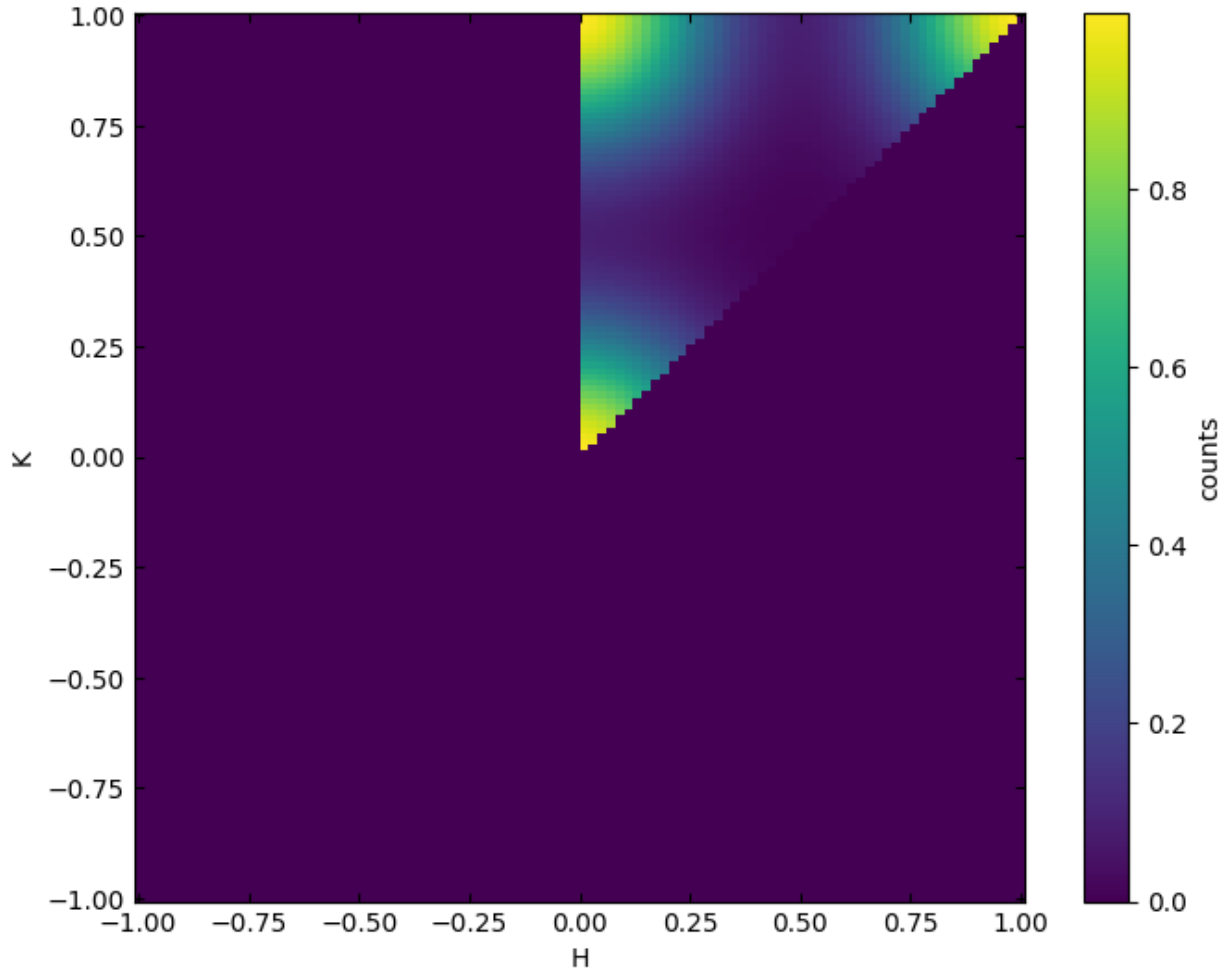


Viewing the wedge used for symmetrization

The `.wedge` attribute stores the wedge of data used to recreate the entire dataset.

```
plot_slice(s2d.wedge)
```

```
<matplotlib.collections.QuadMesh at 0x7f1df0bfadd0>
```



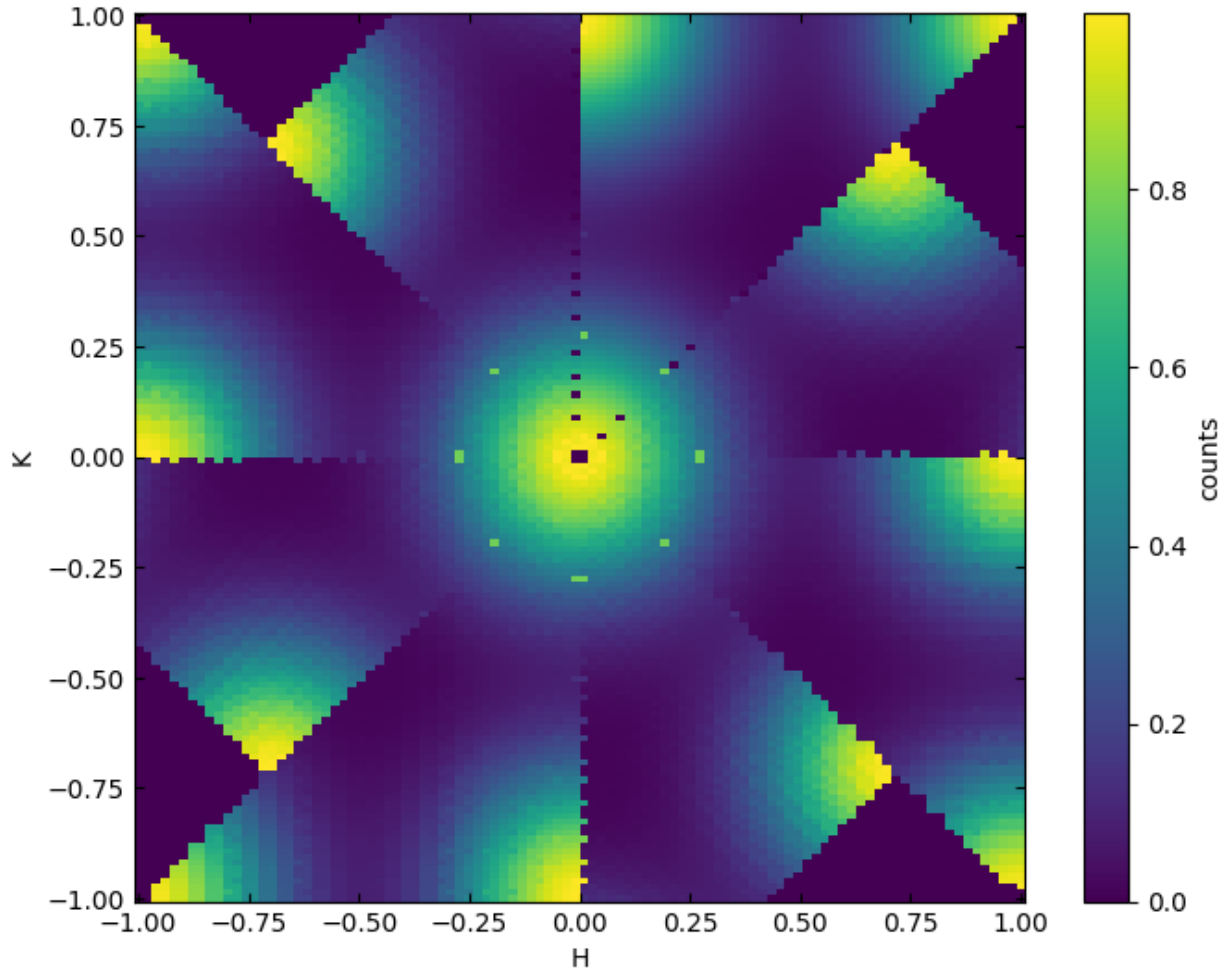
The mirror operation

The optional Boolean parameter `mirror` applies a mirror transformation to the wedge before it is rotated around the origin to reconstruct the dataset. The default value is `True`, in order to preserve the mirror symmetry common to diffraction patterns. If false, the wedge is rotated enough times to recreate all 360 degrees of the plane.

Here, the effect is noticeable because the wedge is only 45 degrees, and 8-fold rotational symmtery is not present in the parent dataset.

```
s2d = Symmetrizer2D(theta_min=0, theta_max=45, mirror=False)
plot_slice(s2d.symmetrize_2d(data[:,:,0.0]))
```

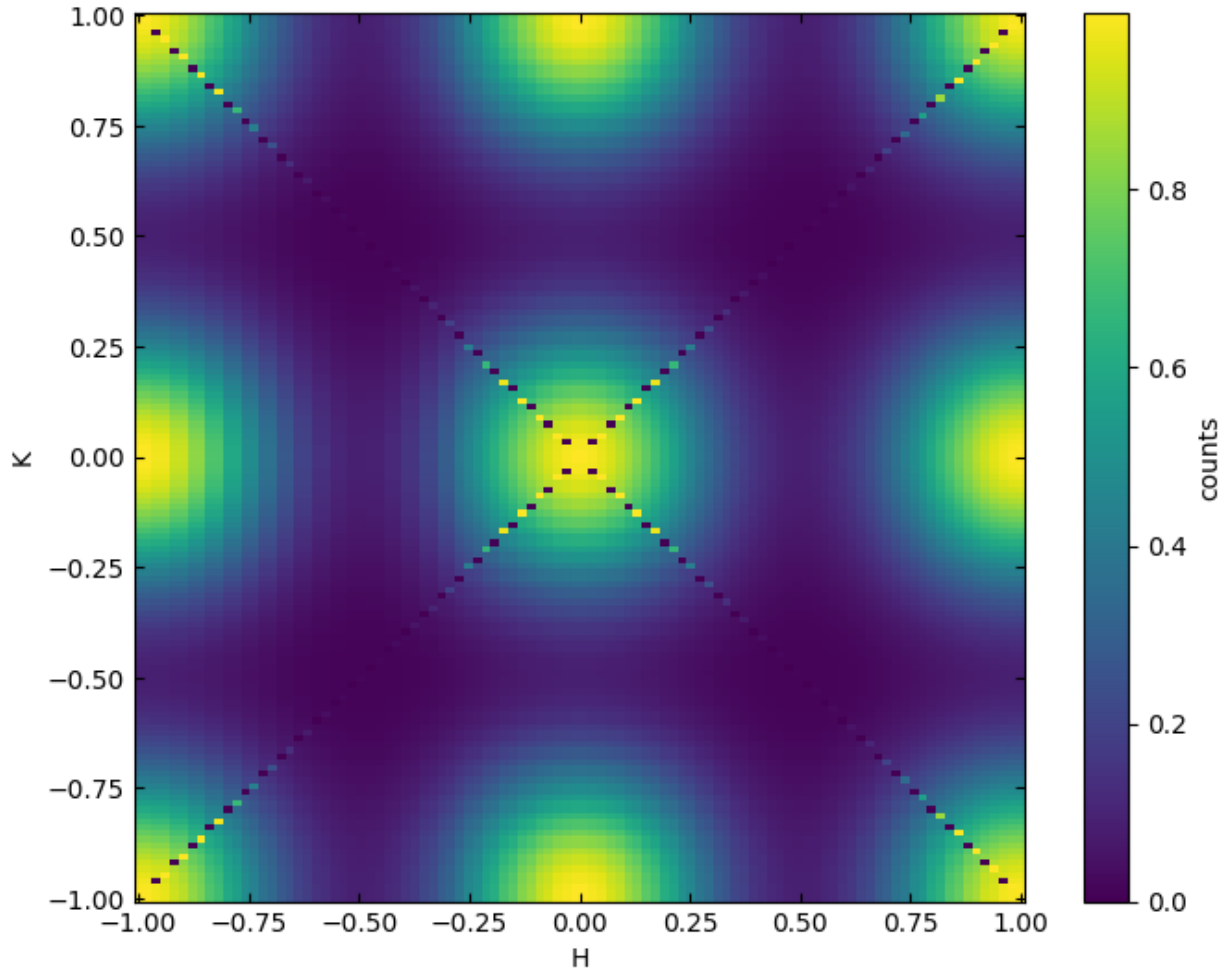
```
<matplotlib.collections.QuadMesh at 0x7f1dee7da0e0>
```



Here, we change the wedge to cover 90 degrees, and since 4-fold rotation is present in the parent dataset the symmetrization produces the expected result.

```
s2d = Symmetrizer2D(theta_min=45, theta_max=135, mirror=False)
plot_slice(s2d.symmetrize_2d(data[:,:,0.0]))
```

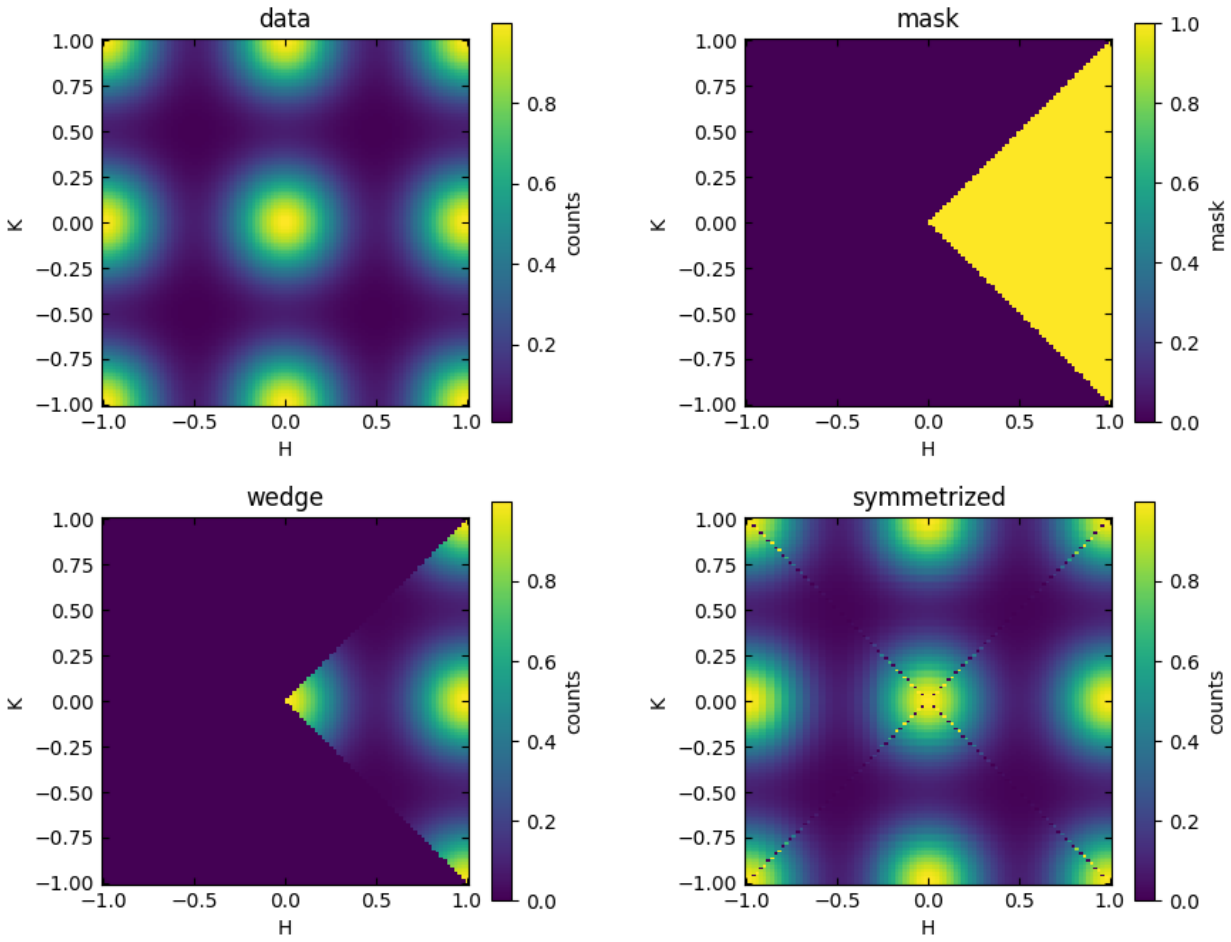
```
<matplotlib.collections.QuadMesh at 0x7f1dee6d3d00>
```



The `test` function

Use the `.test()` method to visualize an overview of the symmetrization process, including the raw data, symmetrization mask, wedge, and reconstructed dataset.

```
s2d.test(data[:,:,0.0])
```




```
(<Figure size 1000x800 with 8 Axes>,
 array([[<Axes: title={'center': 'data'}, xlabel='H', ylabel='K'>,
         <Axes: title={'center': 'mask'}, xlabel='H', ylabel='K'>],
        [<Axes: title={'center': 'wedge'}, xlabel='H', ylabel='K'>,
         <Axes: title={'center': 'symmetrized'}, xlabel='H', ylabel='K'>]],
       dtype=object))
```

The `Symmetrizer3D` class

The `Symmetrizer3D` class wraps three instances of the `Symmetrizer2D` class, which represent the three primary cross-sections of the dataset, called "Plane 1", "Plane 2", and "Plane 3".

```
s = Symmetrizer3D(data)
```

```
Plane 1: HK
Plane 2: HL
Plane 3: KL
```

Each `Symmetrizer2D` object is then accessible via the attributes `.plane1symmetrizer`, etc.

```
s.plane1symmetrizer
```

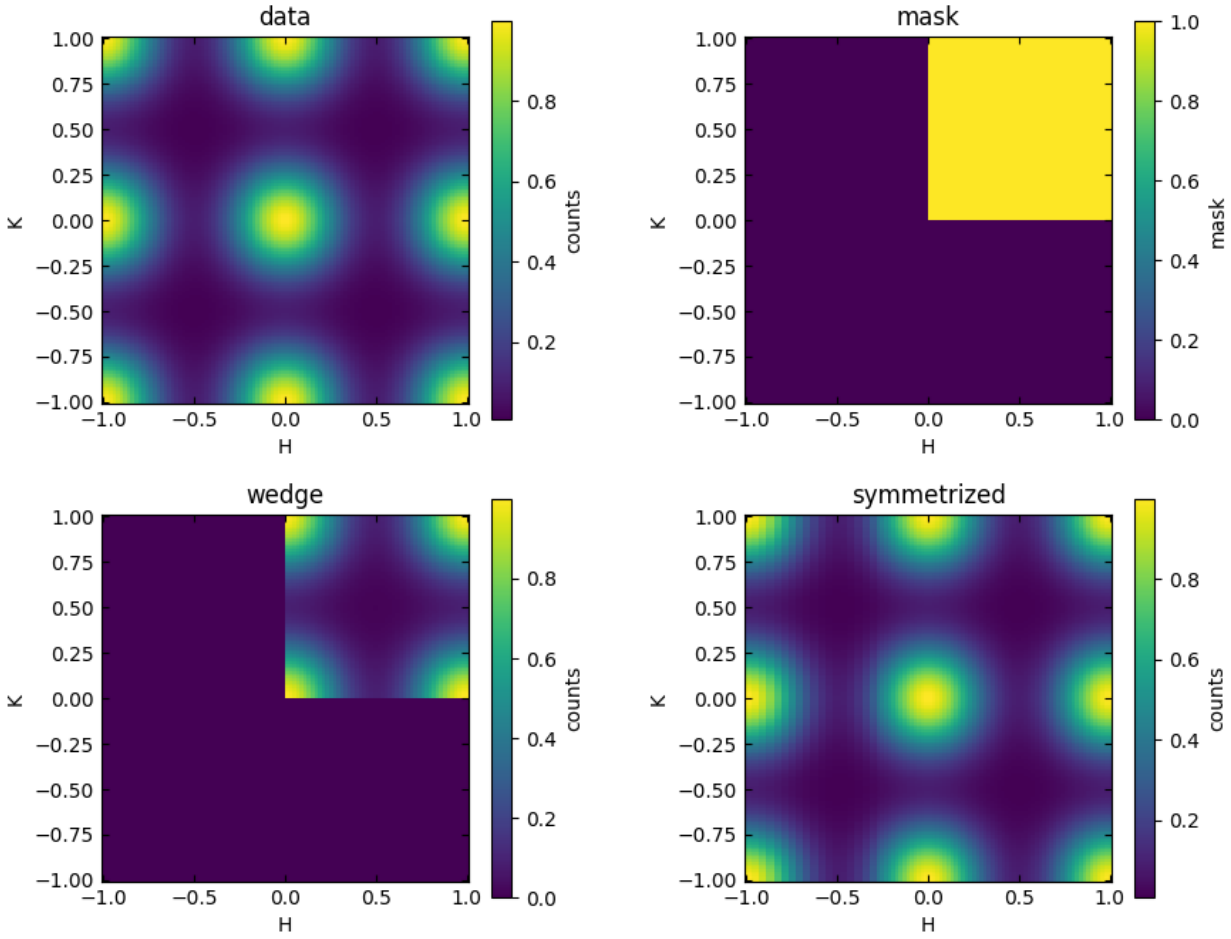
```
<nxs_analysis_tools.pairdistribution.Symmetrizer2D at 0x7f1decdcc190>
```

The parameters for each Symmetrizer2D object can then be set in the usual way.

```
s.plane1symmetrizer.set_parameters(theta_min=0, theta_max=90, mirror=True)
```

The .test() method of the Symmetrizer2D class should be used to verify the symmetry operations on each plane.

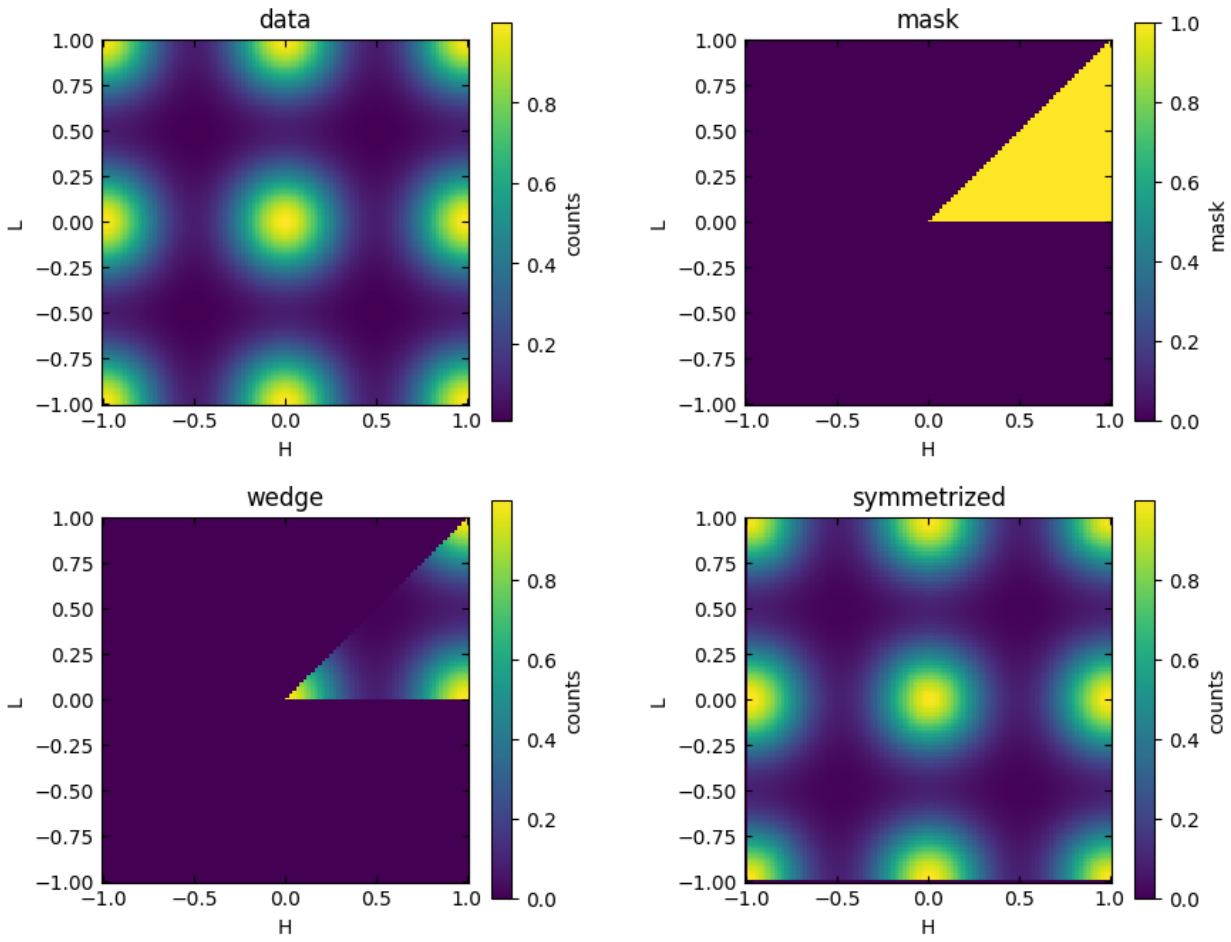
```
s.plane1symmetrizer.test(data[:,:,len(data.L)//2])
```




```
(<Figure size 1000x800 with 8 Axes>,
 array([[<Axes: title={'center': 'data'}, xlabel='H', ylabel='K'>,
         <Axes: title={'center': 'mask'}, xlabel='H', ylabel='K'>],
        [<Axes: title={'center': 'wedge'}, xlabel='H', ylabel='K'>,
         <Axes: title={'center': 'symmetrized'}, xlabel='H', ylabel='K'>]],
       dtype=object))
```

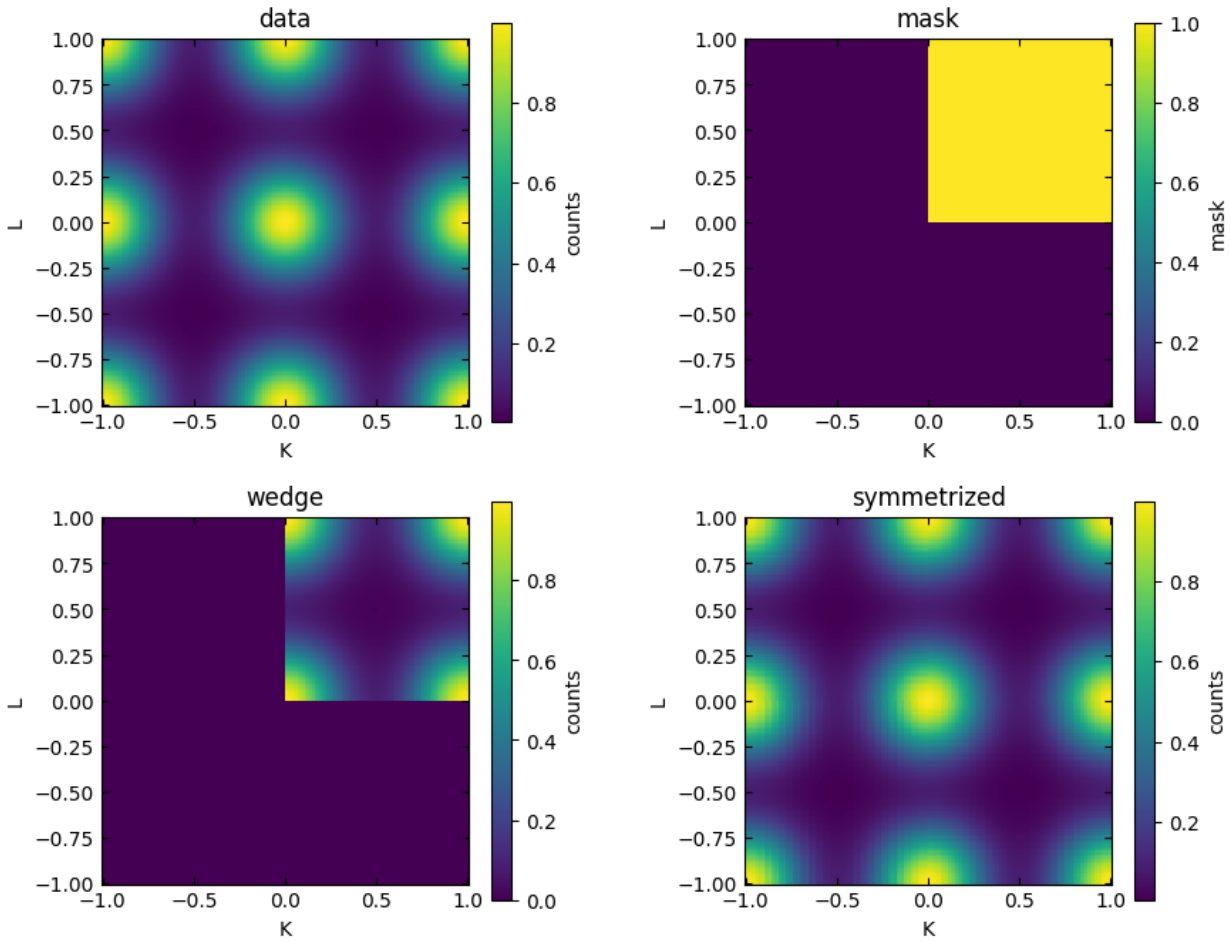
Once the Symmetrizer2D objects for each plane have been initialized, the .symmetrize() method of the Symmetrize3D object can be used to perform the full symmetrization.

```
s.plane2symmetrizer.set_parameters(theta_min=45, theta_max=90, mirror=True)
s.plane2symmetrizer.test(data[:,len(data.K)//2,:])
```




```
(<Figure size 1000x800 with 8 Axes>,
 array([[<Axes: title={'center': 'data'}, xlabel='H', ylabel='L'>,
         <Axes: title={'center': 'mask'}, xlabel='H', ylabel='L'>],
        [<Axes: title={'center': 'wedge'}, xlabel='H', ylabel='L'>,
         <Axes: title={'center': 'symmetrized'}, xlabel='H', ylabel='L'>]],
       dtype=object))
```

```
s.plane3symmetrizer.set_parameters(theta_min=0, theta_max=90, mirror=False)
s.plane3symmetrizer.test(data[len(data.H)//2,:,:])
```




```
(<Figure size 1000x800 with 8 Axes>,
 array([[<Axes: title={'center': 'data'}, xlabel='K', ylabel='L'>,
         <Axes: title={'center': 'mask'}, xlabel='K', ylabel='L'>],
        [<Axes: title={'center': 'wedge'}, xlabel='K', ylabel='L'>,
         <Axes: title={'center': 'symmetrized'}, xlabel='K', ylabel='L'>]],
       dtype=object))
```

```
s.symmetrize()
```

```
Symmetrizing HK planes...
Symmetrizing L=1.00...
Symmetrized HK planes.
Symmetrizing HL planes...
Symmetrizing K=1.00...
Symmetrized HL planes.
Symmetrizing KL planes...
Symmetrizing H=1.00...
Symmetrized KL planes.

Symmetriztaion finished in 1.05 minutes.
```

```
NXdata('data')
```

2.2 API Reference

2.2.1 datareduction

Reduces scattering data into 2D and 1D datasets.

nxs_analysis_tools.datareduction.**load_data**(*path*)

Load data from a specified path.

Parameters

path
[str] The path to the data file.

Returns

data
[nxdata object] The loaded data stored in a nxdata object.

nxs_analysis_tools.datareduction.**plot_slice**(*data*, *X=None*, *Y=None*, *transpose=False*, *vmin=None*, *vmax=None*, *skew_angle=90*, *ax=None*, *xlim=None*, *ylim=None*, *xticks=None*, *yticks=None*, *cbar=True*, *logscale=False*, *symlogscale=False*, *cmap='viridis'*, *linthresh=1*, *title=None*, *mdheading=None*, *cbartitle=None*, ***kwargs*)

Parameters

data
[nexusformat.nexus.NXdata object or ndarray] The NXdata object containing the dataset to plot.

X
[NXfield, optional] The X axis values. Default is first axis of *data*.

Y
[NXfield, optional] The y axis values. Default is second axis of *data*.

transpose
[bool, optional] If True, tranpose the dataset and its axes before plotting. Default is False.

vmin
[float, optional] The minimum value to plot in the dataset. If not provided, the minimum of the dataset will be used.

vmax
[float, optional] The maximum value to plot in the dataset. If not provided, the maximum of the dataset will be used.

skew_angle
[float, optional] The angle to shear the plot in degrees. Defaults to 90 degrees (no skewing).

ax
[matplotlib.axes.Axes, optional] An optional axis object to plot the heatmap onto.

xlim
[tuple, optional] The limits of the x-axis. If not provided, the limits will be automatically set.

ylim
[tuple, optional] The limits of the y-axis. If not provided, the limits will be automatically set.

xticks
[float, optional] The major tick interval for the x-axis. If not provided, the function will use a default minor tick interval of 1.

yticks
[float, optional] The major tick interval for the y-axis. If not provided, the function will use a default minor tick interval of 1.

cbar
[bool, optional] Whether to include a colorbar in the plot. Defaults to True.

logscale
[bool, optional] Whether to use a logarithmic color scale. Defaults to False.

symlogscale
[bool, optional] Whether to use a symmetrical logarithmic color scale. Defaults to False.

cmap
[str or Colormap, optional] The color map to use. Defaults to 'viridis'.

linthresh
[float, optional] The linear threshold for the symmetrical logarithmic color scale. Defaults to 1.

mdheading
[str, optional] A string containing the Markdown heading for the plot. Default *None*.

Returns

p
[matplotlib.collections.QuadMesh] A matplotlib.collections.QuadMesh object, to mimick behavior of matplotlib.pyplot.pcolormesh.

class nxs_analysis_tools.datareduction.**Scissors**

Scissors class provides functionality for reducing data to a 1D linecut using an integration window.

Attributes

data	(nexusformat.nexus.NXdata or None) Input nexusformat.nexus.NXdata.
center	(tuple or None) Central coordinate around which to perform the linecut.
window	(tuple or None) Extents of the window for integration along each axis.
axis	(int or None) Axis along which to perform the integration.
data_cut	(ndarray or None) Data array after applying the integration window.
integrated_axes	(tuple or None) Indices of axes that were integrated.
linecut	(nexusformat.nexus.NXdata or None) 1D linecut data after integration.
window_plane_slice_obj	(list or None) Slice object representing the integration window in the data array.

Methods

set_data(data)	Set the input `nexusformat.nexus.NXdata`
get_data()	Get the input `nexusformat.nexus.NXdata`.
set_center(center)	Set the central coordinate for the linecut.
set_window(window)	Set the extents of the integration window.
get_window()	Get the extents of the integration window.
cut_data(axis=None)	Reduce data to a 1D linecut using the integration window.
show_integration_window(label=No	Plot the integration window highlighted on a 2D heatmap of the full dataset.
plot_window()	Plot a 2D heatmap of the integration window data.

`__init__`(*data=None*, *center=None*, *window=None*, *axis=None*)

Initializes a Scissors object.

Parameters

data
[`nexusformat.nexus.NXdata` or `None`, optional] Input NXdata. Default is None.

center
[`tuple` or `None`, optional] Central coordinate around which to perform the linecut. Default is None.

window
[`tuple` or `None`, optional] Extents of the window for integration along each axis. Default is None.

axis
[`int` or `None`, optional] Axis along which to perform the integration. Default is None.

`set_data`(*data*)

Set the input NXdata.

Parameters

data
[`nexusformat.nexus.NXdata`] Input data array.

`get_data`()

Get the input data array.

Returns

`ndarray` **or** `None`
Input data array.

`set_center`(*center*)

Set the central coordinate for the linecut.

Parameters

center
[`tuple`] Central coordinate around which to perform the linecut.

`set_window`(*window*)

Set the extents of the integration window.

Parameters

window
[tuple] Extents of the window for integration along each axis.

`get_window()`

Get the extents of the integration window.

Returns

tuple or None
Extents of the integration window.

`cut_data(center=None, window=None, axis=None)`

Reduces data to a 1D linecut with integration extents specified by the window about a central coordinate.

Parameters

center
[float or None, optional] Central coordinate for the linecut. If not specified, the value from the object's attribute will be used.

window
[tuple or None, optional] Integration window extents around the central coordinate. If not specified, the value from the object's attribute will be used.

axis
[int or None, optional] The axis along which to perform the linecut. If not specified, the value from the object's attribute will be used.

Returns

integrated_data
[nexusformat.nexus.NXdata] 1D linecut data after integration.

`highlight_integration_window(data=None, label=None, highlight_color='red', **kwargs)`

Plots integration window highlighted on the three principal cross sections of the first temperature dataset.

Parameters

data
[array_like, optional] The 2D heatmap dataset to plot. If not provided, the dataset stored in *self.data* will be used.

label
[str, optional] The label for the integration window plot.

highlight_color
[str, optional] The edge color used to highlight the integration window. Default is 'red'.

****kwargs**
[keyword arguments, optional] Additional keyword arguments to customize the plot.

`plot_integration_window(**kwargs)`

Plots the three principal cross-sections of the integration volume on a single figure.

Parameters

****kwargs**
[keyword arguments, optional] Additional keyword arguments to customize the plot.

`nxs_analysis_tools.datareduction.rotate_data(data, lattice_angle, rotation_angle, rotation_axis, printout=False)`

Rotates 3D data around a specified axis.

Parameters

data
[nexusformat.nexus.NXdata] Input data.

lattice_angle
[float] Angle between the two in-plane lattice axes in degrees.

rotation_angle
[float] Angle of rotation in degrees.

rotation_axis
[int] Axis of rotation (0, 1, or 2).

printout
[bool, optional] Enables printout of rotation progress. If set to True, information about each rotation slice will be printed to the console, indicating the axis being rotated and the corresponding coordinate value. Defaults to False.

Returns

rotated_data
[nexusformat.nexus.NXdata] Rotated data as an NXdata object.

nxs_analysis_tools.datareduction.**array_to_nxdata**(*array*, *data_template*, *signal_name='counts'*)

Create an NXdata object from an input array and an NXdata template, with an optional signal name.

Parameters

array
[array_like] The data array to be included in the NXdata object.

data_template
[NXdata] An NXdata object serving as a template, which provides information about axes and other metadata.

signal_name
[str, optional] The name of the signal within the NXdata object. If not provided, the default signal name 'counts' is used.

Returns

NXdata
An NXdata object containing the input data array and associated axes based on the template.

class nxs_analysis_tools.datareduction.**Padder**

A class to pad and unpad datasets with a symmetric region of zeros.

Methods

pad(padding)	Symmetrically pads the data with zero values.
save([fout_name])	Saves the padded dataset to a .nxs file.
set_data(data)	Set the input data for symmetrization.
unpad(data)	Removes the padded region from the data.

__init__(*data=None*)

Initialize the Symmetrizer3D object.

Parameters

data
[NXdata, optional] The input data to be symmetrized. If provided, the *set_data* method is called to set the data.

set_data(*data*)

Set the input data for symmetrization.

Parameters

data
[NXdata] The input data to be symmetrized.

pad(*padding*)

Symmetrically pads the data with zero values.

Parameters

padding
[tuple] The number of zero-value pixels to add along each edge of the array.

save(*fout_name=None*)

Saves the padded dataset to a .nxs file.

Parameters

fout_name
[str, optional] The output file name. Default is padded_(Hpadding)_(Kpadding)_(Lpadding).nxs

unpad(*data*)

Removes the padded region from the data.

Parameters

data
[ndarray or NXdata] The padded data from which to remove the padding.

Returns

ndarray or NXdata
The unpadded data, with the symmetric padding region removed.

Notes

This method removes the symmetric padding region that was added using the *pad* method. It returns the data without the padded region.

2.2.2 chess

This module provides classes and functions for analyzing scattering datasets collected at CHESS (ID4B) with temperature dependence. It includes functions for loading data, cutting data, and plotting linecuts.

class nxs_analysis_tools.chess.**TempDependence**

Class for analyzing scattering datasets collected at CHESS (ID4B) with temperature dependence.

Methods

clear_datasets()	Clear the datasets stored in the TempDependence instance.
cut_data([center, window, axis])	Perform data cutting for each temperature dataset.
fit()	Fit the line cut models.
get_folder()	Get the folder path where the datasets are located.
guess()	Make initial parameter guesses for all line cut models.
highlight_integration_window([temperature])	Displays the integration window plot for a specific temperature, or for the first temperature if none is provided.
load_datasets(folder[, file_ending, ...])	Load scattering datasets from the specified folder.
make_params()	Make parameters for all line cut models.
plot_fit([mdheadings])	Plot the fit results.
plot_initial_guess()	Plot the initial guess for all line cut models.
plot_integration_window([temperature])	Plots the three principal cross-sections of the integration volume on a single figure for a specific temperature, or for the first temperature if none is provided.
plot_linecuts([vertical_offset])	Plot the linecuts obtained from data cutting.
print_fit_report()	Plot the fit results.
print_initial_params()	Print the initial parameter values for all line cut models.
set_center(center)	Set the central coordinate for the linecut.
set_model_components(model_components)	Set the model components for all line cut models.
set_param_hint(*args, **kwargs)	Set parameter hints for all line cut models.
set_window(window)	Set the extents of the integration window.

__init__()

Initialize TempDependence class.

get_folder()

Get the folder path where the datasets are located.

Returns

str:
The folder path.

clear_datasets()

Clear the datasets stored in the TempDependence instance.

load_datasets(*folder*, *file_ending='hkli.nxs'*, *temperatures_list=None*)

Load scattering datasets from the specified folder.

Parameters

folder
[str] The path to the folder where the datasets are located.

file_ending
[str, optional] The file extension of the datasets to be loaded. The default is 'hkli.nxs'.

temperatures_list
[list of int or None, optional] The list of specific temperatures to load. If None, all available temperatures are loaded. The default is None.

`set_window`(*window*)

Set the extents of the integration window.

Parameters

window
[`tuple`] Extents of the window for integration along each axis.

`set_center`(*center*)

Set the central coordinate for the linecut.

Parameters

center
[`tuple`] Central coordinate around which to perform the linecut.

`cut_data`(*center=None*, *window=None*, *axis=None*)

Perform data cutting for each temperature dataset.

Parameters

center
[`tuple`] The center point for cutting the data.

window
[`tuple`] The window size for cutting the data.

axis
[`int` or `None`, optional] The axis along which to perform the cutting. If None, cutting is performed along the longest axis in *window*. The default is None.

Returns

`list`
A list of linecuts obtained from the cutting operation.

`plot_linecuts`(*vertical_offset=0*, ***kwargs*)

Plot the linecuts obtained from data cutting.

Parameters

vertical_offset
[`float`, optional] The vertical offset between linecuts on the plot. The default is 0.

****kwargs**
Additional keyword arguments to be passed to the plot function.

`highlight_integration_window`(*temperature=None*, ***kwargs*)

Displays the integration window plot for a specific temperature, or for the first temperature if none is provided.

Parameters

temperature
[`str`, optional] The temperature at which to display the integration window plot. If provided, the plot will be generated using the dataset corresponding to the specified temperature. If not provided, the integration window plots will be generated for the first temperature.

****kwargs**
[`keyword arguments`, optional] Additional keyword arguments to customize the plot.

plot_integration_window(*temperature=None*, ***kwargs*)

Plots the three principal cross-sections of the integration volume on a single figure for a specific temperature, or for the first temperature if none is provided.

Parameters

temperature
[str, optional] The temperature at which to plot the integration volume. If provided, the plot will be generated using the dataset corresponding to the specified temperature. If not provided, the integration window plots will be generated for the first temperature.

****kwargs**
[keyword arguments, optional] Additional keyword arguments to customize the plot.

set_model_components(*model_components*)

Set the model components for all line cut models.

This method sets the same model components for all line cut models in the analysis. It iterates over each line cut model and calls their respective *set_model_components* method with the provided *model_components*.

Parameters

model_components
[Model or iterable of Model] The model components to set for all line cut models.

set_param_hint(**args*, ***kwargs*)

Set parameter hints for all line cut models.

This method sets the parameter hints for all line cut models in the analysis. It iterates over each line cut model and calls their respective *set_param_hint* method with the provided arguments and keyword arguments.

Parameters

***args**
Variable length argument list.

****kwargs**
Arbitrary keyword arguments.

make_params()

Make parameters for all line cut models.

This method creates the parameters for all line cut models in the analysis. It iterates over each line cut model and calls their respective *make_params* method.

guess()

Make initial parameter guesses for all line cut models.

This method generates initial parameter guesses for all line cut models in the analysis. It iterates over each line cut model and calls their respective *guess* method.

print_initial_params()

Print the initial parameter values for all line cut models.

This method prints the initial parameter values for all line cut models in the analysis. It iterates over each line cut model and calls their respective *print_initial_params* method.

plot_initial_guess()

Plot the initial guess for all line cut models.

This method plots the initial guess for all line cut models in the analysis. It iterates over each line cut model and calls their respective *plot_initial_guess* method.

fit()

Fit the line cut models.

This method fits the line cut models for each temperature in the analysis. It iterates over each line cut model, performs the fit, and prints the fitting progress.

plot_fit(*mdheadings=False*, ***kwargs*)

Plot the fit results.

This method plots the fit results for each temperature in the analysis. It iterates over each line cut model, calls their respective *plot_fit* method, and sets the xlabel, ylabel, and title for the plot.

print_fit_report()

Plot the fit results.

This method plots the fit results for each temperature in the analysis. It iterates over each line cut model, calls their respective *plot_fit* method, and sets the xlabel, ylabel, and title for the plot.

2.2.3 pairdistribution

Tools for generating single crystal pair distribution functions.

class nxs_analysis_tools.pairdistribution.**Symmetrizer2D**

A class for symmetrizing 2D datasets.

Methods

set_parameters(theta_min, theta_max[, ...])	Sets the parameters for the symmetrization operation.
symmetrize_2d(data)	Symmetrizes a 2D dataset based on the set parameters.
test(data, **kwargs)	Performs a test visualization of the symmetrization process.

__init__(***kwargs*)

set_parameters(*theta_min*, *theta_max*, *lattice_angle=90*, *mirror=True*, *mirror_axis=0*)

Sets the parameters for the symmetrization operation.

Parameters

theta_min
[float] The minimum angle in degrees for symmetrization.

theta_max
[float] The maximum angle in degrees for symmetrization.

lattice_angle
[float, optional] The angle in degrees between the two principal axes of the plane to be symmetrized (default: 90).

mirror
[bool, optional] If True, perform mirroring during symmetrization (default: True).

mirror_axis
[int, optional] The axis along which to perform mirroring (default: 0).

symmetrize_2d(*data*)

Symmetrizes a 2D dataset based on the set parameters.

Parameters

data
[NXdata] The input 2D dataset to be symmetrized.

Returns

symmetrized
[NXdata] The symmetrized 2D dataset.

test(*data*, ***kwargs*)

Performs a test visualization of the symmetrization process.

Parameters

data
[ndarray] The input 2D dataset to be used for the test visualization.

****kwargs**
[dict] Additional keyword arguments to be passed to the plot_slice function.

Returns

fig
[Figure] The matplotlib Figure object that contains the test visualization plot.

axesarr
[ndarray] The numpy array of Axes objects representing the subplots in the test visualization.

Notes

This method uses the *symmetrize_2d* method to perform the symmetrization on the input data and visualize the process.

The test visualization plot includes the following subplots: - Subplot 1: The original dataset. - Subplot 2: The symmetrization mask. - Subplot 3: The wedge slice used for reconstruction of the full symmetrized dataset. - Subplot 4: The symmetrized dataset.

Example usage: ` s = Scissors() s.set_parameters(theta_min, theta_max, skew_angle, mirror) s.test(data) `

class nxs_analysis_tools.pairdistribution.**Symmetrizer3D**

A class to symmetrize 3D datasets.

Methods

save([fout_name])	Save the symmetrized dataset to a file.
set_data(data)	Sets the data to be symmetrized.
symmetrize()	Perform the symmetrization of the 3D dataset.

set_lattice_params

`__init__`(*data=None*)

Initialize the Symmetrizer3D object.

Parameters

data
[NXdata, optional] The input 3D dataset to be symmetrized.

`set_data`(*data*)

Sets the data to be symmetrized.

Parameters

data
[NXdata] The input 3D dataset to be symmetrized.

`symmetrize`()

Perform the symmetrization of the 3D dataset.

Returns

symmetrized
[NXdata] The symmetrized 3D dataset.

`save`(*fout_name=None*)

Save the symmetrized dataset to a file.

Parameters

fout_name
[str, optional] The name of the output file. If not provided, the default name 'symmetrized.nxs' will be used.

nxs_analysis_tools.pairdistribution.`generate_gaussian`(*H*, *K*, *L*, *amp*, *stddev*, *lattice_params*, *coeffs=None*)

Generate a 3D Gaussian distribution.

Parameters

H, K, L
[ndarray] Arrays specifying the values of H, K, and L coordinates.

amp
[float] Amplitude of the Gaussian distribution.

stddev
[float] Standard deviation of the Gaussian distribution.

lattice_params
[tuple] Tuple of lattice parameters (a, b, c, alpha, beta, gamma).

coeffs
[list, optional] Coefficients for the Gaussian expression, including cross-terms between axes. Default is [1, 0, 1, 0, 1, 0], corresponding to (1*H**2 + 0*H*K + 1*K**2 + 0*K*L + 1*L**2 + 0*L*H)

Returns

gaussian
[ndarray] 3D Gaussian distribution.

2.3 License

CHAPTER

THREE

LICENSE

This project is *licensed* under the MIT License.

PYTHON MODULE INDEX

n

INDEX

Symbols

A

C

F

G

H

L

M

N

P

R

S

T

U