

---

# **MJOLNIR Documentation**

***Release 1.0***

**Jakob Lass**

**Aug 03, 2023**





---

## Contents

---

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Software Structure . . . . .	1
1.2	Installation . . . . .	1
1.3	License . . . . .	2
1.4	Bug Report . . . . .	2
<b>2</b>	<b>Tutorials</b>	<b>3</b>
2.1	Scripting Tutorials . . . . .	3
2.2	Command Line Tutorials . . . . .	25
<b>3</b>	<b>MJOLNIR Module</b>	<b>31</b>
3.1	Geometry Module . . . . .	31
3.2	Statistics Module . . . . .	36
3.3	Data Module . . . . .	37
3.4	Tools functions . . . . .	59
<b>4</b>	<b>In depth description of core functionalities</b>	<b>61</b>
4.1	Geometry . . . . .	61
4.2	Energy normalization procedure . . . . .	63
4.3	Data file conversion . . . . .	65
4.4	Voronoi tessellation and plotting functionality . . . . .	70
4.5	Visualization methods . . . . .	72
<b>5</b>	<b>Optimizations</b>	<b>73</b>
5.1	Optimizing of the plotA3A4 routine . . . . .	73
5.2	Voronoi Tessellation subroutine . . . . .	74
5.3	Timing function . . . . .	74
<b>6</b>	<b>Commissioning</b>	<b>77</b>
6.1	29/10-18 - Start of hot commissioning . . . . .	77
6.2	30/10-18 - Opening of shutter and background . . . . .	79
6.3	31/10-18 - Data wrangling and measurement . . . . .	80
6.4	01/11-18 - First Vanadium normalization scan . . . . .	81
6.5	02/11-18 - Background hunting . . . . .	82
6.6	05/11-18 - Energy normalization . . . . .	82
6.7	06/11-18 - Determination of A4 + Be filter cooling . . . . .	85
6.8	09/11-18 - First magnon in YMnO <sub>3</sub> . . . . .	86

6.9	10/11-18 - Currat Axe Spurion in YMnO <sub>3</sub>	87
6.10	12/11-18 - No beam	87
6.11	13/11-18 - No beam	87
6.12	14/11-18 - No beam	88
6.13	15/11-18 - No beam	88
6.14	16/11-18 - Diffuse scattering	88
6.15	17/11-18 - Magnon in YMnO <sub>3</sub>	89
6.16	18/11-18 - Spinwaves in PbTi	90
6.17	19/11-18 - Spinwaves in PbTi	90
6.18	20/11-18 - Vacuum problems at SINQ	90
6.19	21/11-18 - Measurement of CuSeO <sub>3</sub>	93
6.20	21/11-18 - Measurement of CuSeO <sub>3</sub> II	93
6.21	23/11-18 - Startup of Ni <sub>3</sub> TeO <sub>6</sub>	93
6.22	24/11-18 - Measurment of Ni <sub>3</sub> TeO <sub>6</sub> II	93
6.23	25/11-18 - Measurment of Ni <sub>3</sub> TeO <sub>6</sub> III	93
6.24	26/11-18 - Measurment of YMnO <sub>3</sub> Startup	93
6.25	27/11-18 - Measurment of YMnO <sub>3</sub> II	93
6.26	28/11-18 - Measurment of YMnO <sub>3</sub> III	93
6.27	29/11-18 - Measurment of YMnO <sub>3</sub> IV	93
6.28	30/11-18 - Measurment of YMnO <sub>3</sub> V	94
6.29	01/12-18 - Measurment of YMnO <sub>3</sub> VI	94
6.30	02/12-18 - Startup of Ming Purple	94
6.31	03/12-18 - Ming Purple II	94
6.32	04/12-18 - Magnet force test and Startup of LSCO	94
6.33	05/12-18 - LSCO II	94
6.34	06/12-18 - Christmas and Ming Purple	95
6.35	07/12-18 - Christmas and Ming Purple	95
6.36	08/12-18 - Christmas and Ming Purple	96
6.37	09/12-18 - Startup of ???	96
6.38	10/12-18 - Beam Down	96
6.39	11/12-18 - Beam Down	96
6.40	12/12-18 - Beam Down	96
6.41	13/12-18 - Beam development	96
6.42	14/12-18 - Startup of K <sub>2</sub> Ni <sub>2</sub>	96
6.43	15/12-18 - K <sub>2</sub> Ni <sub>2</sub> II	96
6.44	16/12-18 - K <sub>2</sub> Ni <sub>2</sub> III	96
6.45	17/12-18 - Start of SCBO	96
6.46	18/12-18 - SCBO II	96
6.47	19/12-18 - Start of MnF <sub>2</sub>	96
6.48	20/12-18 - MNF <sub>2</sub> II	97
6.49	21/12-18 - MNF <sub>2</sub> III and Beam Shutdown	97
6.50	Shielding Issues	97
6.51	Electronic logbook of scans files	98

<b>Python Module Index</b>	<b>109</b>
----------------------------	------------

<b>Index</b>	<b>111</b>
--------------	------------

# CHAPTER 1

---

## Introduction

---

This is the introductory page for the MOLJNIR software package. The main purpose of this document is to give an overview of different features of the software and how you can contribute to it.

The software is currently developed solely by Jakob Lass, PhD student at both the Niels Bohr Institute, Copenhagen - Denmark, and the Paul Scherrer Institute, Villigen - Switzerland, and is not developed by a professional team. The software is intended to be used for data treatment and visualization for the CAMEA upgrad at the RITA II instrument at SINQ, PSI Villigen - Switzerland.

The software is found at [GitHub](#) and is intended to be used together with Python versions 2.7, (3.4,) 3.5, and 3.6. This compability is ensured by the use of automated unit test through the Travis project ([Travis](#)). Python 3.4 is no longer tested due to updates in the Travis testing framework. Further than just testing, as to ensure a thorough testing the coverage of these are monitored using Coveralls ([Coveralls](#)). However, certain algorithms and methods are not suited to be tested through simple tests. This includes graphical methods where one for example uses a plotting routine to generate a specific output. Though the visual inspection is far outside of the testing scope for this software, some of the methods are still tested by simple run through test. That is, if they can be run and generate a plot without crashing and throwing an error, it is believed that they work as intended. This is where acutal user testing is needed.

## 1.1 Software Structure

The software is devided into individual modules being Instrument, DataSet, and Statistics. With this division it is intended that each part of the software suit is to be fully independent of the others but may be used together. The same goes for the tutorials that are intended to cover all of the methods and workflows a user would come into contact with while using the software.

## 1.2 Installation

The inteded way for the software to be installed on a computer is currently to navigate to the github page [GitHub](#) and download the latest release, usually found in the main branch. For an up to date version with new features, the 1.0.0 branch can be used, but is not to be assumed complete. It is recommended to create a new Anaconda environment or similar for the installation, but this is not required. By utilizing the pip package manager:

```
cd MJOLNIR
pip install .
```

Alternatively one can install it directly by running

```
cd MJOLNIR
python install.py install
```

Both of the above methods installs the software to the specified environment. If only a temporary local build is wanted, the latter approach can be used with ‘install’ replaced by ‘build’.

## 1.3 License

The software package is released under the software lincense Mozilla Public License Version 2.0 to allow for redistribution and usage of the code. If this lincses file has not been shipped with your distribution of the code, please find it here: [licence](#).

## 1.4 Bug Report

If an un error or unexpected behaviour of the software is observed, or if a feature is needed, you are more than welcome to create an issue or feature request at the GitHub page ([Issues](#)). Dealing and fixing the reported bugs will be most easily done if both operation system, software version, a minimal working example, and other relevant informations are provided. Further as time goes by, it is hoped that this page will also contain explanations and answers to the most frequently asked question of the software.

Currently there are the following open issues and closed

Below is a list of different types tutorials in order to familiarize users with the objects and methods used in MJOLNIR.

## 2.1 Scripting Tutorials

Following is a list of scripting tutorials covering a given method but does not necessarily present a normal workflow.

### 2.1.1 Build a simple instrument

In order to build a complete instrument using the MJOLNIR *Geometry Module* module, a lot of different objects need to come together. This tutorial sets out to introduce these step by step with each example increases complexity. The following is an example of how to build a simple instrument consisting of one *Detectors* and one *Analysers* grouped together by a *Wedge* object:

```

1 from MJOLNIR.Geometry import Instrument,
  ↳Detector,Analyser,Wedge
2 def test_Build_a_simple_
  ↳instrument(saveFig = False):
3     import matplotlib.pyplot as plt
4     import numpy as np
5     from mpl_toolkits.mplot3d import _
  ↳Axes3D
6
7     Instr = Instrument.Instrument()
8
9     Det = Detector.
  ↳TubeDetector1D(position=(1,0,1),
  ↳direction=(1,0,0))
10    Ana = Analyser.
  ↳FlatAnalyser(position=(1,0,0),
  ↳direction=(1,0,1))
11
12    wedge = Wedge.Wedge(position=(0,0,0),
  ↳detectors=Det,analysers=Ana)
13
14    Instr.append(wedge)
15
16    fig = plt.figure()
17    ax = fig.gca(projection='3d')
18
19    Instr.plot(ax)
20
21    ax.set_xlim(-0.1,1.1)
22    ax.set_ylim(-0.1,1.1)
23    ax.set_zlim(-0.1,1.1)
24
25    if saveFig:
26        plt.savefig('../docs/_templates/
  ↳Build_a_simple_instrument.png',format=
  ↳'png',dpi=300)
27        plt.show()
28
29 if __name__=='__main__':
30     test_Build_a_simple_
  ↳instrument(saveFig = True)
31

```

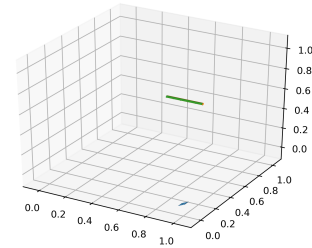


Fig. 1: The figure produced by the current code example.

### 2.1.2 Calculating A4 and Ef

This is an example of initialization of an instrument with two detectors and two analysers. The detectors each have 10 pixels, where the first 5 are looking at the first analyser and the last 5 are looking at the second analyser. This gives rise to different energies. As the second detector is moved away from the straight line through the analyser its A4 values

also changes.

<pre> 1  from MJOLNIR.Geometry import Instrument,     ↪Detector,Analyser,Wedge 2  def test_CalcualteA4Ef(): 3 4      Instr = Instrument.Instrument() 5      Det = Detector.     ↪TubeDetector1D(position=(1,0,1),     ↪direction=(1,0,0),pixels=10,split=[2,5,     ↪8],length=1.0) 6      Det2 = Detector.     ↪TubeDetector1D(position=(1,0.1,1),     ↪direction=(1,0,0),pixels=10,split=[2,5,     ↪8],length=1.0) 7      Ana = Analyser.     ↪FlatAnalyser(position=(0.5,0,0),     ↪direction=(1,0,1)) 8      Ana2 = Analyser.     ↪FlatAnalyser(position=(1,0,0),     ↪direction=(1,0,1)) 9 10     wedge = Wedge.Wedge(position=(0,0,0),     ↪detectors=[Det,Det2],analysers=[Ana,     ↪Ana2]) 11 12     Instr.append(wedge) 13 14     Instr.initialize() 15     print(Instr.A4) 16     print(Instr.Ef) 17 18 if __name__=='__main__': 19     test_CalcualteA4Ef() </pre>	<pre> [[array([-1.57079633, -1.57079633, -1.57079633, - 1.57079633, -1.57079633, -1.57079633]), array([- 1.55451765, -1.55481497, -1.55518368, -1.52099283, -1.52123672, -1.5217106 ])] [[array([ 4.81164763, 5.44262522, 6.18119629, 3.83622353, 4.27947022, 4.81164763]), array([ 4.81454296, 5.44345335, 6.17926234, 3.84580202, 4.29004983, 4.82331398)]] </pre>
--	--

### 2.1.3 Load instrument from XML file

In order to not having to create an instrument from scratch each time a data treatment is performed, one way is to load the instrument from a XML file formatted as below. What is important is the structure of the XML file, where the outer object is the instrument with all of its settings; middle part is the wedge(s) and inner part all of the detectors and analysers. All objects in the instrument have their attributes defined in the opening bracket of the XML object and nothing between it and the closing bracket.

```

1 <?xml version="1.0"?>
2 <Instrument Initialized='False' Author='Jakob Lass' Date ='16/03/18' position='0.0,
  ↳0.0,0.0'>
3   <Wedge position='0.0,0.0,0.0' concept='ManyToMany'>
4     <FlatAnalyser position='0.054020677125896165,0.9284297315590769,0.0' direction='0.
  ↳707106781187,0.0,0.707106781187' d_spacing='3.35' mosaicity='60' width='0.05'
  ↳height='0.1'></FlatAnalyser>
5     <FlatAnalyser position='0.05773242042519161,0.9922218389210393,0.0' direction='0.
  ↳707106781187,0.0,0.707106781187' d_spacing='3.35' mosaicity='60' width='0.05'
  ↳height='0.1'></FlatAnalyser>
6     <FlatAnalyser position='0.06139188564984909,1.0551154658976218,0.0' direction='0.
  ↳707106781187,0.0,0.707106781187' d_spacing='3.35' mosaicity='60' width='0.05'
  ↳height='0.1'></FlatAnalyser>
7     <FlatAnalyser position='0.06502811617466747,1.1176097682584802,0.0' direction='0.
  ↳707106781187,0.0,0.707106781187' d_spacing='3.35' mosaicity='60' width='0.05'
  ↳height='0.1'></FlatAnalyser>
8     <FlatAnalyser position='0.06869919874924452,1.1807030575429251,0.0' direction='0.
  ↳707106781187,0.0,0.707106781187' d_spacing='3.35' mosaicity='60' width='0.05'
  ↳height='0.1'></FlatAnalyser>
9     <FlatAnalyser position='0.07235285529894223,1.2434968533655766,0.0' direction='0.
  ↳707106781187,0.0,0.707106781187' d_spacing='3.35' mosaicity='60' width='0.05'
  ↳height='0.1'></FlatAnalyser>
10    <FlatAnalyser position='0.07608202462311699,1.3075884541893323,0.0' direction='0.
  ↳707106781187,0.0,0.707106781187' d_spacing='3.35' mosaicity='60' width='0.05'
  ↳height='0.1'></FlatAnalyser>
11    <FlatAnalyser position='0.07985185467201017,1.3723788730906055,0.0' direction='0.
  ↳707106781187,0.0,0.707106781187' d_spacing='3.35' mosaicity='60' width='0.05'
  ↳height='0.1'></FlatAnalyser>
12    <TubeDetector1D position='0.13917281377075363,1.1919022308508072,0.7' direction='0.
  ↳13917281377075363,1.1919022308508072,0.0' pixels='1024' length='0.883' diameter=
  ↳'0.02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>
13    <TubeDetector1D position='0.11605723555894083,1.1943746137935185,0.7' direction='0.
  ↳11605723555894083,1.1943746137935185,0.0' pixels='1024' length='0.883' diameter=
  ↳'0.02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>
14    <TubeDetector1D position='0.09289810020961405,1.1963987391239779,0.7' direction='0.
  ↳09289810020961405,1.1963987391239779,0.0' pixels='1024' length='0.883' diameter=
  ↳'0.02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>
15    <TubeDetector1D position='0.06970409951728537,1.1979738471730024,0.7' direction='0.
  ↳06970409951728537,1.1979738471730024,0.0' pixels='1024' length='0.883' diameter=
  ↳'0.02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>
16    <TubeDetector1D position='0.04648393836168752,1.19909934679091,0.7' direction='0.
  ↳04648393836168752,1.19909934679091,0.0' pixels='1024' length='0.883' diameter='0.
  ↳02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>
17    <TubeDetector1D position='0.02324633144076829,1.1997748155693826,0.7' direction='0.
  ↳02324633144076829,1.1997748155693826,0.0' pixels='1024' length='0.883' diameter=
  ↳'0.02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>
18    <TubeDetector1D position='-0.0,1.2,0.7' direction='-0.0,1.2,0.0' pixels='1024'
  ↳length='0.883' diameter='0.02' split='55,158,278,397,515,634,755,877,973'></
  ↳TubeDetector1D>
19    <TubeDetector1D position='0.1276210119438152,1.1931944004689414,0.71' direction='0.
  ↳1276210119438152,1.1931944004689414,0.0' pixels='1024' length='0.883' diameter='0.
  ↳02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>
20    <TubeDetector1D position='0.10448256963424918,1.195442760086247,0.71' direction='0.
  ↳10448256963424918,1.195442760086247,0.0' pixels='1024' length='0.883' diameter='0.
  ↳02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>
21    <TubeDetector1D position='0.08130491424476227,1.1972424612081096,0.71' direction=
  ↳'0.08130491424476227,1.1972424612081096,0.0' pixels='1024' length='0.883'
  ↳diameter='0.02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>
22    <TubeDetector1D position='0.05809674452057436,1.1985928283934086,0.71' direction=
  ↳'0.05809674452057436,1.1985928283934086,0.0' pixels='1024' length='0.883'
  ↳diameter='0.02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>
23    <TubeDetector1D position='0.03486677065916047,1.1994933548393678,0.71' direction=
  ↳'0.03486677065916047,1.1994933548393678,0.0' pixels='1024' length='0.883'
  ↳diameter='0.02' split='55,158,278,397,515,634,755,877,973'></TubeDetector1D>

```



## 2.1.4 Generate normalization table from data

Before real data can be converted from pixel position and counts into  $S(q, \omega)$ , one needs decide the binning used for the data as well as generate the normalization tables used. This is done using a Vanadium scan file containing a suitable number of energy steps. Three different binnings for each energy at all of the detectors are default for the CAMEA backend:

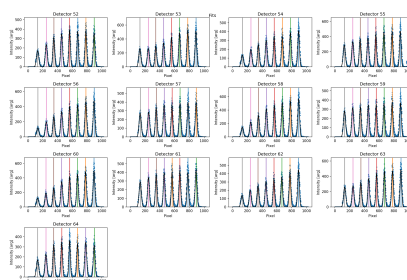
- 8 pixels ('PrismaticHighDefinition')
- 3 pixels ('PrismaticLowDefinition')
- 1 pixel ('Single')
- n pixels (n is integer)

Having chosen binning(s) one creates the tables either with or without creating fit plots at the same time. Creating these does indeed increase runtime a lot but is needed when one wants to inspect the fitting performed. A error will be raised if the number of peaks found in the data file does not match the number of analyser the detectors are exposed to.

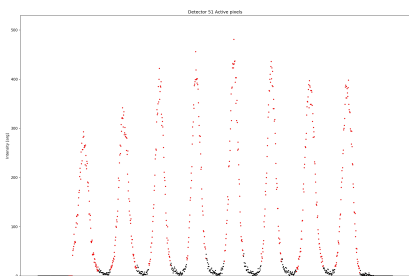
```

1 from MJOLNIR.Geometry import Instrument
2 def Generate_normalization(plot=False):
3     Instr = Instrument.
4     ↳Instrument(fileName='/home/lass/
5     ↳Dropbox/PhD/Software/MJOLNIR/Data/
6     ↳CAMEA_Updated.xml') #'TestData/1024/
7     ↳CAMEA_Full.xml')
8     Instr.initialize()
9
10    VanNormFile = '/home/lass/Dropbox/
11    ↳PhD/CAMEADData/camea2018n000119.hdf' #'/'
12    ↳home/lass/Dropbox/PhD/CAMEADData/
13    ↳camea2018n000084.hdf' #'/'TestData/1024/
14    ↳camea2018n000038.hdf' #'/'TestData/1024/
15    ↳EScanRunDoubleFocusHS.h5'
16    Instr.
17    ↳generateCalibration(Vanadiumdatafile=VanNormFile,
18    ↳savelocation='/home/lass/Dropbox/PhD/
19    ↳CAMEADData/NormalizationUpdated/',
20    ↳plot=plot, tables=[1, 3, 8])
21
22 if __name__ == '__main__':
23     Generate_normalization(False)

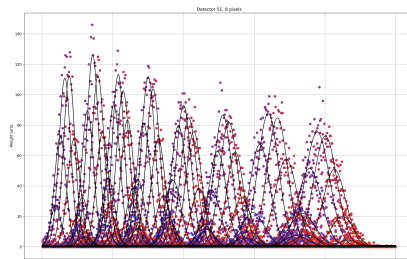
```



Plot of fit to data integrated in the energy direction for wedge 4.



Active area of detector 51 as defined by 3 sigmas away from center pixel, where red denotes active and black inactive.



Fit of peaks in vanadium data for detector 51 when using a a binning of 8 pixels per analyser.

In the end, it is the data in the normalization file, in the above case denoted EnergyNormalization\_8.calib and located in the TestData, folder that is essential. It contains the normalization and energy location of all peaks on all detectors in the format:

- Detector (int)
- Energy (int)
- Pixel (int)
- Amplitude [Arb]
- Center [meV]
- Width [meV]
- Background [Arb]
- lowerBin [pixelId]
- upperBin [pixelId]
- A4Offset [deg]

on each line starting with detector 0, analyser 0, pixel 0 increasing index of pixel, then analyser and lastly detector.

## 2.1.5 Raw plotting and fitting

For specific details on the object c.f. *Viewer1D*.

The visualizer is intended to take a data-set, plot a single cut from it and perform fitting on it with simple functions (currently the Gaussian and Lorentzian are available). During initial initialization this is of great importance as to refine the UB matrix, find the A3 offset or set up the goniometers. As it is only intended to deal with one 1D set at a time, no functionality is currently developed to display more than 1 at a time. However, one can cycle through the provided data both along the different scan parameters and data values.

Below is a table of the shortcuts available in the different states of the program:

All States	Key
Quit	q
Copy current values	ctrl+c
Cycle up in data	up
Cycle down in data	down
Cycle up in scan parameter	right
Cycle down in scan parameter	left

The initial window shown consists of a text part at the top and a plot of the current data below. This initial window allows the following key presses:

Initial State	Key
Initialize fitting	i or ctrl+i

By pressing 'i' or 'ctrl+i' one starts the fitting. Here one can change between different fit functions. These are currently limited to a Gaussian and a Lorentzian function. Having chosen the fitting function one moves the mouse onto the canvas and left clicks corresponding to the guess one has for the parameter(s) shown in bold font (multiple parameters chosen depending on the fit function).

Fitting State	Key
Choose Gaussian function	0
Choose Lorentzian function	1
Execute fit	e
Print current fit to terminal	i or ctrl+i
Choose guess for bold parameters	leftmouseclick
Cycle backwards in parameters	r

When the initial guess is satisfactory one presses 'e' to execute the fit. The x and y data is then fitted with the shown parameters as initial guesses and the errorbars as absolute standard deviations by the `scipy.optimize.curve_fit` function.

---

**Note:** Errorbars being zero is reset to 1 and fit is performed with the least squares algorithm.

---

After a fit execution the found parameters are written in place of the guess parameters together with the square-root of the covariance matrix. The diagonal elements are given by the diagonal elements. By pressing 'ctrl+c' the fitted parameters are copied to the clipboard. If another fit is wanted, one can press 'i' or 'ctrl+i' to initialize another fit.

Executed State	Key
Initialize another fit	i

Further shortcuts corresponds to the standard keys of [Matplotlib shortcuts](#).

## 2.1.6 Convert data to Q and Omega

With the above normalization table created, one can easily convert raw data files using the method in the *DataSet* called `ConvertDatafile` as

```

1 from MJOLNIR.Data import DataSet
2 def test_Convert_Data(save=False):
3     DataFile = ['Data/comea2018n000137.hdf']
4
5     dataset = DataSet.DataSet(dataFiles=DataFile)
6     dataset.convertDataFile(saveLocation='Data/', saveFile=save)
7
8
9 if __name__ == '__main__':
10     test_Convert_Data(True)

```

The code then converts the scan files, be it either Ei, A4, or A3 scan, and saves it into a new HDF files following the Nexus NXsqom data convention. This is followed in order to facilitate easy interfacing with other software later used. The location of the converted file is the same as the original but the new file has the ending `.nxs`. Furthermore, in order to store all information and ensure that no data is lost, all information already present in the raw data file is copied into the new. This also include the original raw data counts.

## 2.1.7 Bin data and visualize

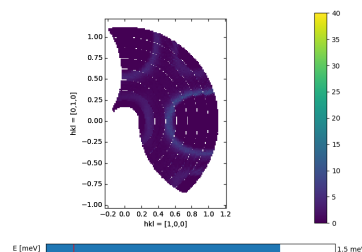
Having converted the data into the Nexus NXsqom format, one wants to both rebin the data and visualize it. As different detector pixels covers different positions in reciprocal space, one needs to rebin the data in order to avoid large areas of no data. This is done using the method in the *DataSet* called `binData3D`. As input one needs to provide the step size in the x, y, and z directions, the 3D position and intensity. Furthermore, normalization and monitor count

can be specified in order to also bin these. Returned is the rebinned data together with normalization and monitor count, if applicable, and the bins used.

```

1  from MJOLNIR.Data import
    ↳DataSet, Viewer3D
2  def test_Binning_data(view =
    ↳False):
3      import numpy as np
4      import h5py as hdf
5      import matplotlib.pyplot
    ↳as plt
6      fileName = 'Data/
    ↳camea2018n000137.hdf'
7      ds = DataSet.
    ↳DataSet(dataFiles=fileName)
8      ds.convertDataFile()
9
10     I = ds.convertedFiles[0].I
11     qx = ds.convertedFiles[0].
    ↳qx
12     qy = ds.convertedFiles[0].
    ↳qy
13     energy = ds.
    ↳convertedFiles[0].energy
14     Norm = ds.
    ↳convertedFiles[0].Norm
15     Monitor = ds.
    ↳convertedFiles[0].Monitor
16     title = 'Magnon
    ↳ComponentA3Scan'
17
18     pos = [qx,qy,energy]
19
20     Data,bins = DataSet.
    ↳binData3D(0.02,0.02,0.1,pos,
    ↳I,norm=Norm,mon=Monitor)
21     import warnings
22     warnings.simplefilter(
    ↳"ignore")
23     Intensity = np.
    ↳divide(Data[0]*Data[3],
    ↳Data[1]*Data[2])
24     warnings.simplefilter('once
    ↳')
25
26     Viewer = Viewer3D.
    ↳Viewer3D(Intensity,bins,
    ↳axis=2)
27
28     Viewer.caxis=(0,40)
29
30     Viewer.ax.set_
    ↳title(str(title)[2:-1])
31     if view:
32         plt.show()
33     else:
34         if os.path.exists(
    ↳'Data/camea2018n000137.nxs'):
35             os.remove('Data/
    ↳camea2018n000137.nxs')
36
37

```



Cut through data along the energy direction showing Qx and Qy for a phonon scan at the energy 1.5 meV.

The bins and calculated intensity is then passed on to the Viewer3D object, that generates a matplotlib figure. This plot is made interactive through the slider in the bottom, that shows the current position along the axis as well as the value of this. By pressing the up/down arrows (or +/- buttons as well as scrolling) one can change this value and thus investigate the third dimension. By default the energy direction is chosen but it can be changed by pressing the 0, 1, or 2 numerical buttons. This is to be understood as which direction is to be sliced. Further, by clicking on the plot, the x and y value corresponding to the point is printed to the terminal. This is then intended to be used in further data treatment when cuts are needed.

### **2.1.8 Full data treatment**

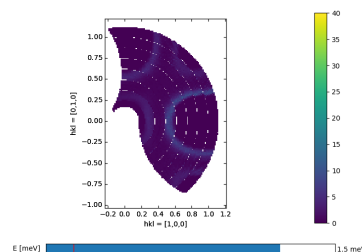
Full example of the data treatment starting from the instrument definitions provided in the XML file, through generation of normalization table using 8 software pixels, and to data conversion, rebinning and visualization.

```

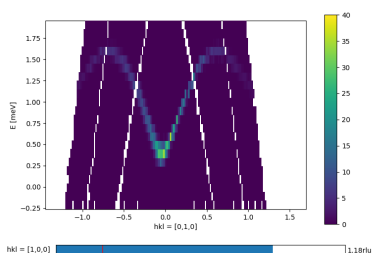
1  from MJOLNIR.Data import
    ↳DataSet, Viewer3D
2  def test_Full_
    ↳example(save=False,
    ↳show=False):
3      import warnings
4      import matplotlib.pyplot
    ↳as plt
5      import numpy as np
6      DataFile = 'Data/
    ↳camea2018n000017.hdf'
7
8      dataset = DataSet.
    ↳DataSet(dataFiles=DataFile)
9      dataset.
    ↳convertDataFile(saveLocation=
    ↳'Data/', saveFile=save)
10     viewer = dataset.View3D(0.
    ↳02, 0.02, 0.1, rlu=False)
11
12     viewer.caxis=(0, 40)
13     if show:
14         plt.show()
15
16 if __name__=='__main__':
17     test_Full_example(False,
    ↳True)
18

```

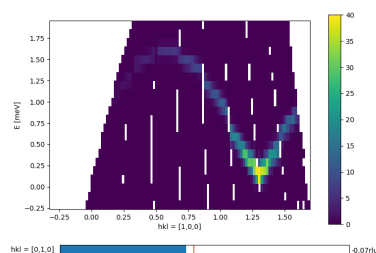
Binning of the converted data into Qx and Qy bins of size 0.02 ÅÅ and energy in 0.1 meV. Intensity is calculated and with the bins passed to the visualizer.



Cut through data along the energy direction showing Qx and Qy for a phonon scan at the energy 1.5 meV.



Cut of data along the Qx direction.



Cut of data along the Qy direction.

## 2.1.9 Full data treatment without Viewer3D

Full example of a data treatment workflow starting from raw h5 files and ending in a plot. The treatment is done in four steps signified in the code by the comments:

- Convert raw data from h5 to nxs (NXSqm) data format
- Open converted files and extract intensities and measurement positions from them
- Bin the data using polar binning and calculated the intensity for each point
- Plot an energy slice of the data set in a regular Matplotlib figure

```

1 from MJOLNIR.Data import DataSet
2 def test_Full_example_without_Viewer3D (show=False):
3     import numpy as np
4     import matplotlib.pyplot as plt
5
6     # Convert raw data to NXSqom
7
8     DataFile=['Data/comea2018n000137.hdf']
9     DS = DataSet.DataSet(dataFiles=DataFile)
10    DS.convertDataFile(saveFile=False)
11
12    # Extract all the data
13    I,qx,qy,energy,Norm,Monitor = DS.I,DS.qx,DS.qy,DS.energy,DS.Norm,DS.
↪Monitor
14
15    # Reshape it
16    I = np.concatenate(I)
17    qx = np.concatenate(qx)
18    qy = np.concatenate(qy)
19    energy = np.concatenate(energy)
20    Norm = np.concatenate(Norm)
21    Monitor = np.concatenate(Monitor)
22
23    # Bin data in polar coordinates
24
25    r = np.linalg.norm([qx,qy],axis=0)
26    theta = np.arctan2(qy,qx)
27
28    [I_bin,Monitor_bin,Normalization_bin,NormCount_bin],[r_bin,theta_bin,
↪energy_bin] = \
29    DataSet.binData3D(0.01,np.deg2rad(1.0),0.5,[r.flatten(),theta.flatten(),
↪energy.flatten()],data=I,norm=Norm,mon=Monitor)
30    Qx = np.cos(theta_bin)*r_bin
31    Qy = np.sin(theta_bin)*r_bin
32
33
34    Int = np.divide(I_bin*NormCount_bin,Monitor_bin*Normalization_bin)
35
36    # Plot energy slice of data
37    Eslice=2
38
39    VMIN=0
40    VMAX=20
41
42    fig=plt.figure(figsize=(8,8))
43    pc = plt.pcolormesh(Qx[:, :, Eslice].T,Qy[:, :, Eslice].T,Int[:, :, Eslice].T,
↪vmin=VMIN,vmax=VMAX,zorder=10)
44    ax = fig.add_subplot(111)
45
46
47    plt.ylabel('$Q_y$ [1/Å]')
48    plt.xlabel('$Q_x$ [1/Å]')
49    plt.title('$\hbar \omega$ = $ {:.02f}$'.format(np.mean(energy_bin[:, :,
↪Eslice])) + ' meV')
50    plt.axis([-1.8, 1.8, -1.8, 1.8])
51    ax.set_aspect('equal', 'datalim')
52    plt.grid(True)

```

(continues on next page)



(continued from previous page)

```

53
54     plt.colorbar(pc)
55     if show:
56         plt.show()
57
58 if __name__ == '__main__':
59     test_Full_example_without_Viewer3D(True)

```

In the above example, there are two key points in the treatment; first the binning in polar coordinates which takes advantage of the measurement positions of the instrument setup (equidistant  $A_3$  or theta steps), second that the binned data from the binning method is of e.g. shape (20,23,6) while the position arrays contains the bin edges and have thus the shape (21,24,7). This makes it easy to perform 2D cuts of the data along the primal axis (radius, angle and energy) by simply slicing data using regular slicing tools.

The arguments vmin and vmax controls the colorbar and thus the colors corresponding to different intensities.

---

**Note:** The way that Matplotlib currently has implemented the pcolormesh method for plotting 2D data requires the position matrices and data matrix to be transposed before plotting. Further, applying the ‘gouraud’ interpolation scheme provided by the method, one needs to give as arguments the centers of the binned data instead of the bin edges. This can simply be given by defining e.g. a new  $Q_x$  by  $QX = 0.5*(Qx[:-1,:-1,Eslice]+Qx[1:,1:,Eslice])$

---

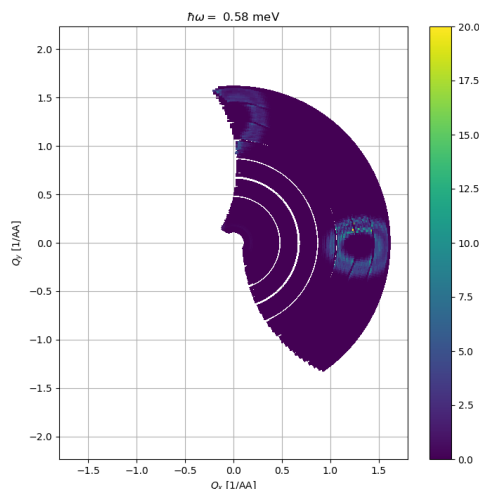


Figure created in the above script showing the second energy (5.60 meV) transfer plane.

## 2.1.10 Powder cutting and visualization

When having a converted data file of a powder sample, one might be interested in the powder signal instead of the (qx,qy) intensities. These can both be found and plotted by using the cutPowder and plotCutPowder methods of the DataSet object. The three different calls below all produce the same data treatment, i.e. intensity, monitor, normalization, and normalization count as a function of length of  $q$  and energy transfer. It has been chosen, with the use of  $qMinBin=0.01$  and  $tolerance=0.125$  that the length of  $q$  and the energies are binned with minimum sizes 0.01 1/Å and 0.125 meV.

```

1 from MJOLNIR.Data import DataSet
2 from MJOLNIR import _tools
3 def test_Powder(show=False):
4     import matplotlib.pyplot as plt
5     file = 'Data/comea2018n000137.hdf'
6
7     DataObj = DataSet.DataSet(dataFiles=file)
8     DataObj.convertDataFile()
9     I = DataObj.I
10    qx = DataObj.qx
11    qy = DataObj.qy
12    energy = DataObj.energy
13    Norm = DataObj.Norm
14    Monitor = DataObj.Monitor
15
16    EBinEdges = _tools.binEdges(energy,tolerance=0.125)
17
18    ax,Data,qbins = DataObj.plotCutPowder(EBinEdges,qMinBin=0.05)
19    plt.colorbar(ax.pmeshs[0])
20
21    ax2,Data2,qbins2 = DataSet.plotCutPowder([qx,qy,energy],I,Norm,Monitor,
22    ↪EBinEdges,qMinBin=0.05)
23    plt.colorbar(ax2.pmeshs[0])
24
25    Data3,qbins3 = DataObj.cutPowder(EBinEdges)
26
27    ax2.set_clim(0,0.01)
28    if show:
29        plt.show()
30
31    if __name__=='__main__':
32        test_Powder(True)

```

Notice in line 21 the call to the axis object changing all of the c-axis for the figure. This method has been added in the plotting routine to ease the change of intensity as the plot consists of many constant energy plots. Furthermore, when hovering over a pixel in the plot, the  $q$  length, energy and intensity of the nearest pixel center is shown. This is to make the binning size somewhat transparent and also ensure that the user sees the current value of the pixel, and not an interpolation.

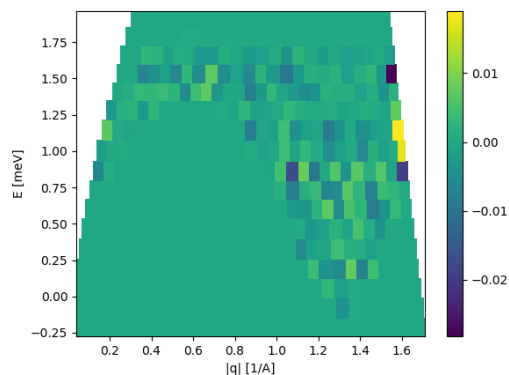
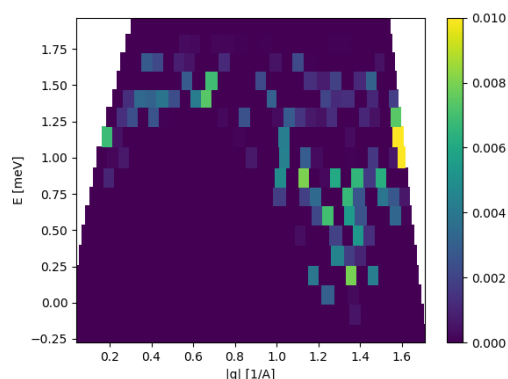


Figure created by the DataSet method plotCutPowder showing a phonon dispersion as well as a spurious signal.

Figure created by the function plotCutPowder showing the same data but with the c-axis changed.



### 2.1.11 Q-energy cutting and visualization in 1 and 2 dimensions

One feature needed when dealing with 3D intensity data is to be able to cut from q one point to another and investigate the energy dependency of the intensity. This can be done by invoking the `cutQE` or `plotCutQE`. These methods perform constant energy cuts between the given q points (q1 and q2) and then stitches them together. When hovering over a position, the nearest qx, qy, and energy center is shown as well as its intensity.

```

1  from MJOLNIR.Data import DataSet
2  from MJOLNIR import _tools
3  def test_cut2D(show=False):
4      import numpy as np
5      import matplotlib.pyplot as plt
6      file = 'Data/comea2018n000137.hdf'
7      DataObj = DataSet.DataSet(dataFiles=file)
8      DataObj.convertDataFile()
9      energy = DataObj.energy
10
11     EnergyBins = _tools.binEdges(energy, tolerance=0.125)
12     q1 = np.array([1.0, 0])
13     q2 = np.array([0, 1.0])
14     width = 0.1 # 1/A
15     minPixel = 0.01
16
17     ax, DataList, qBnLit, centerPos, binDistance = DataObj.plotCutQE(q1, q2, width,
18     ↪ minPixel, EnergyBins, rlu=False)
19     plt.colorbar(ax.pmeshs[0])
20
21 if __name__ == '__main__':
22     test_cut2D(True)

```

Figure created by the `DataSet` method `plotCutQE` showing a phonon dispersion when cutting from (1,0) to (0,1). The c-axis is simply found from the minimal and maximal values of the binned intensities.

... figure:: ../Tutorials/cut2DPlotCLim.png .. :width: 45%

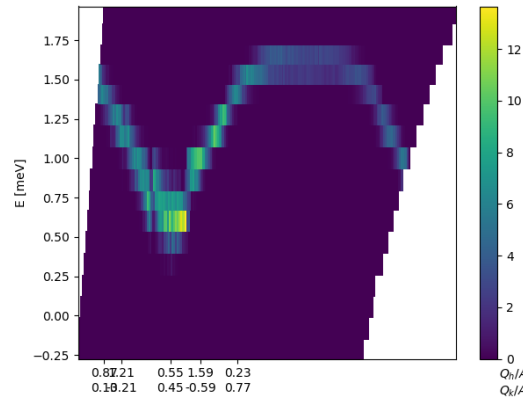
If one instead of a full map is only interested in a 1D cut, this can be achieved by the use of the `(plot)cut1D` method. It takes the same types of arguments as the 2D cutter with the exception of maximal and minimal energies instead of energy bins.

```

1  from MJOLNIR.Data import DataSet
2  from MJOLNIR import _tools
3  def test_cut2D(show=False):

```

(continues on next page)



(continued from previous page)

```

4  import numpy as np
5  import matplotlib.pyplot as plt
6  file = 'Data/comea2018n000137.hdf'
7
8  ## Cut and plot 1D
9  ax2, DataList, Bins, binCenter, binDistance = DataObj.plotCut1D(q1, q2, width,
10 minPixel, rlu=False, Emin = 0.2, Emax = 1.7, plotCoverage=True)
11  if show:
12
13  if __name__ == '__main__':
14      test_cut2D(True)

```

The above code cuts along the same direction as the 2D tool, but produces the two pictures below

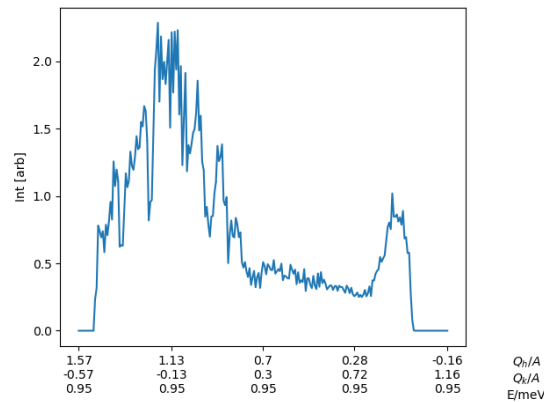
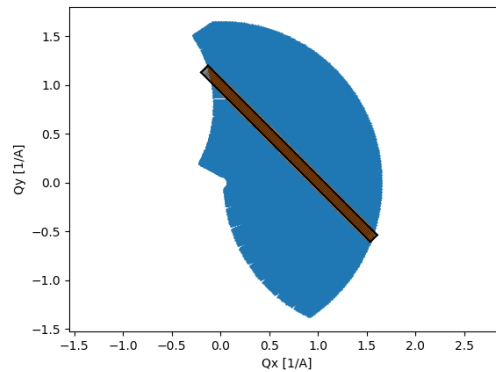


Figure created by the DataSet method plotCut1D showing a cut through a phonon dispersion when cutting from (1,0) to (0,1) and summing energies between 5.2 meV and 5.7 meV.

The points used in the binning algorithm where the black boxes denotes individual bins.

## 2.1.12 Plotting of Q plane using binnings



```

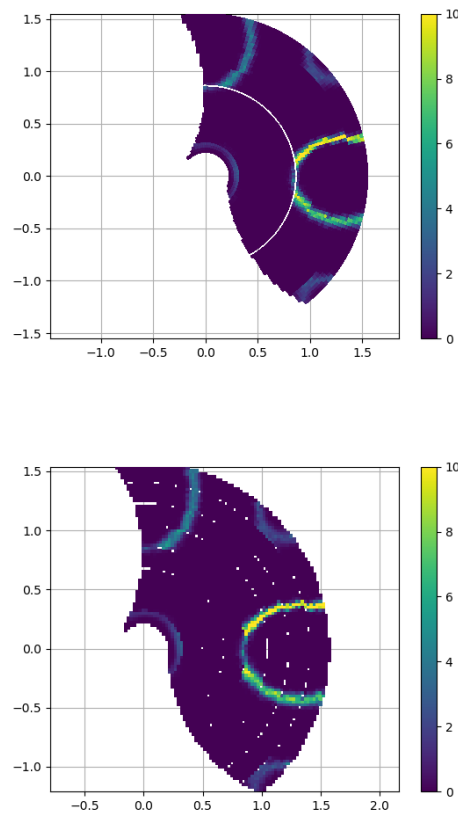
1  from MJOLNIR.Data import DataSet
2  def test_Plot_Q_Plane(save=False):
3      import numpy as np
4      import matplotlib.pyplot as plt
5      file = 'Data/comea2018n000137.hdf'
6      Data = DataSet.DataSet(dataFiles=file)
7      Data.convertDataFile()
8      EMin = np.min(Data.energy)+1.5
9      EMax = EMin+0.05
10
11     ax = Data.plotQPlane(EMin,EMax,binning='polar',xBinTolerance=0.025,
12 ↪ yBinTolerance=0.025,
13                               enlargen=False,log=False,ax=None,RLUPlot=True,vmin=0,
14 ↪ vmax=10)
15     plt.colorbar(ax.pmeshs[0])
16     ax.set_clim(0,10)
17
18     ax2 = Data.plotQPlane(EMin,EMax,binning='xy',xBinTolerance=0.025,
19 ↪ yBinTolerance=0.025,
20                               enlargen=False,log=False,ax=None,RLUPlot=True,vmin=0,
21 ↪ vmax=10)
22     plt.colorbar(ax2.pmeshs[0])
23
24     if save:
25         fig1 = plt.figure(1)
26         fig2 = plt.figure(2)
27         fig1.savefig('Tutorials/PlotQPlanePolar.png')
28         fig2.savefig('Tutorials/PlotQPlaneXY.png')
29         plt.show()
30
31 if __name__ == '__main__':
32     test_Plot_Q_Plane(True)

```

Creation of two plots of the Magnon\_ComponentA3Scan.nxs converted data. Firstly, a polar plot is created on a reciprocal lattice axes, with adaptive bins and non-logarithmic intensities.

Plotting of intensities measured between ~1.29 meV and ~1.34 meV using adaptive polar binning.

Plotting of intensities measured between ~1.29 meV and ~1.34 meV using equi-sized rectangular binning. As the binning is chosen too fine a lot of holes appear in the plot.



### 2.1.13 Plotting all pixels binned in A3 and A4

Instead of performing rebinning of the measured datapoints in to some sort of regular grid, one could instead try to create a grid that fits the data measured. This is the basis idea behind the plotA3A4 method. It takes a list of files, creates a common tessellation using the voronoi method in A3-A4 coordinates and maps this into Q-space.

```

1  from MJOLNIR.Data import DataSet
2  def test_PlotA3A4(save=False):
3      import matplotlib.pyplot as plt
4      import numpy as np
5
6      File = 'Data/comea2018n000137.hdf'
7      DS = DataSet.DataSet(dataFiles=File)
8      DS.convertDataFile()
9      files = DS.convertedFiles
10
11
12     planes2 = list(np.arange(64).reshape(8,8)) # Plot all planes binned with
↪ 8 pixels together
13     ax = [DS.createRLUAxes() for _ in range(len(planes2))] # Create custom
↪ axes for plotting
14
15     ax2 = DS.plotA3A4(files,planes=planes2,ax=ax)
16
17     counter = 0
18     for ax in ax2: # loop through axes to increase size and save

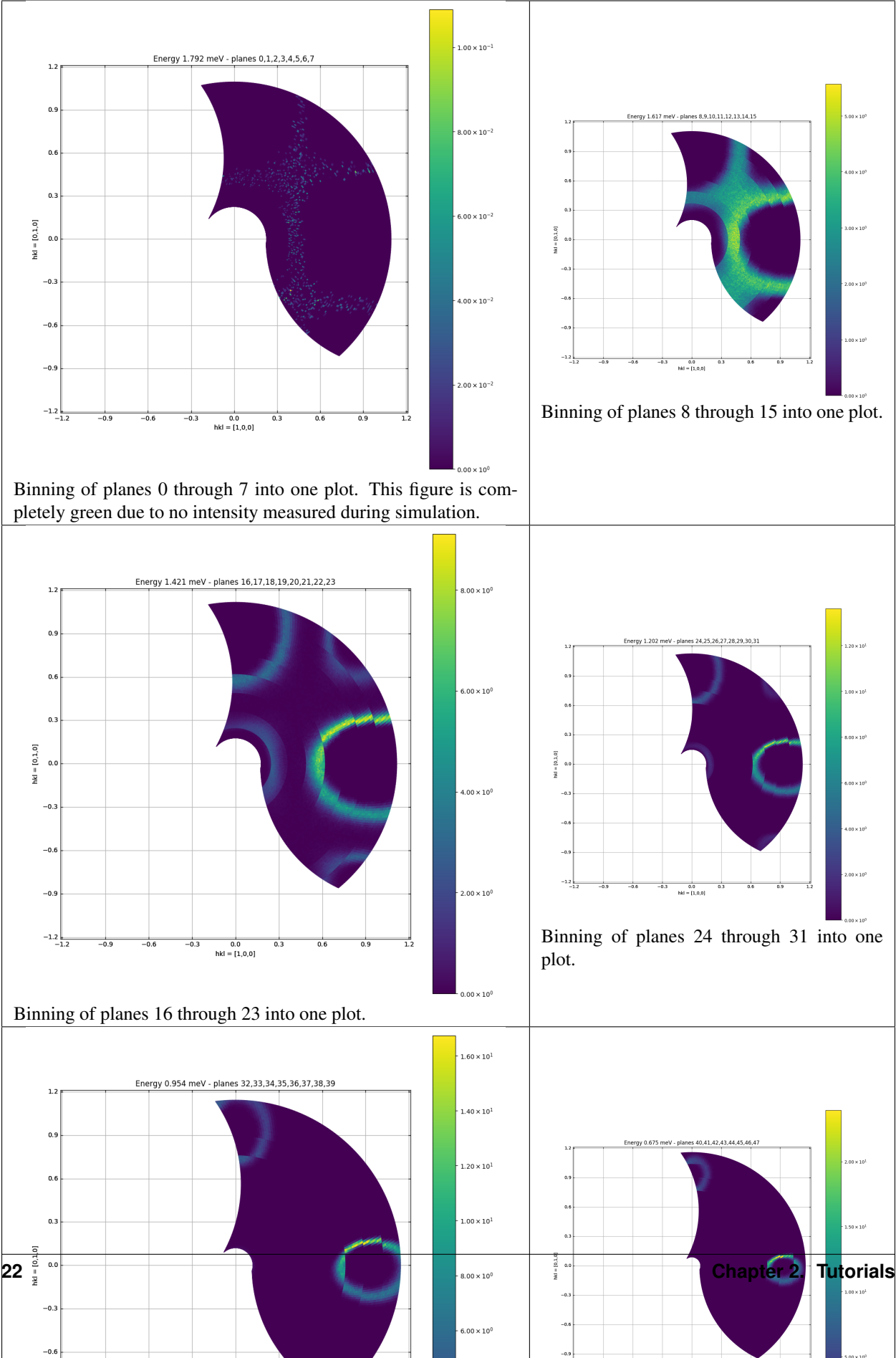
```

(continues on next page)

(continued from previous page)

```
19     fig = ax.get_figure()
20     fig.set_size_inches(10.5, 10.5, forward=True)
21
22     if save:
23         fig.savefig('A3A4/{:03d}.png'.format(counter), format='png')
24         counter+=1
25     if save:
26         plt.show()
27
28 if __name__=='__main__':
29     test_PlotA3A4(True)
30
31
```

As shown above one can provide an axis or a list of axes into which the plot is to be made. This is especially usefull if combined with the RLU axis method calculated in the DataSet object as one then gets the plot in reciprocal lattice units directly.





**Warning:** This Page is not up to date!

## 2.1.14 Plotting all pixels binned in Q space

Instead of performing rebinning of the measured datapoints in to some sort of regular grid, one could instead try to create a grid that fits the data measured. This is the basis idea behind the plotA3A4 method. It takes a list of files, creates a common tessellation using the voronoi method in Q coordinates.

```

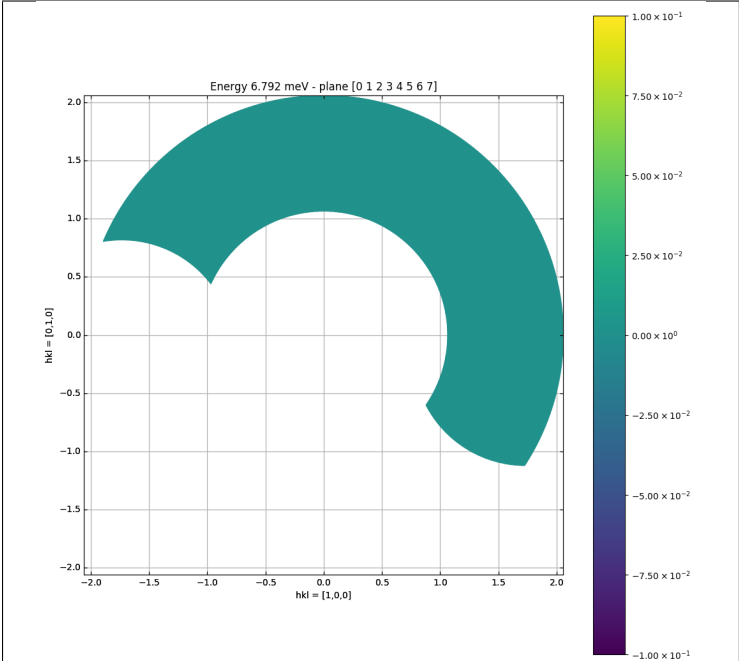
1
2     import matplotlib.pyplot as plt
3     import numpy as np
4     from MJOLNIR.Data import DataSet
5     File1 = '../TestData/T0Phonon10meV.nxs'
6     File2 = '../TestData/T0Phonon10meV93_5A4.nxs'
7
8
9     DS = DataSet.DataSet(convertedFiles=[File1,File2])
10
11     files = DS.convertedFiles
12
13
14     planes2 = list(np.arange(64).reshape(8,8))[1:] # Plot all planes binned
↪with 8 pixels together
15     ax = [DS.createRLUAxes() for _ in range(len(planes2))] # Create custom
↪axes for plotting
16
17     ax2 = DS.plotQPatches([files[0]],planes=planes2,ax=ax,binningDecimals=2,
↪A4Extend=2,A3Extend=3)
18
19     counter = 0
20     for ax in ax2: # loop through axes to increase size and save
21         fig = ax.get_figure()
22         fig.set_size_inches(10.5, 10.5, forward=True)
23         fig.tight_layout()
24         fig.savefig('QPatches/{:03d}.png'.format(counter),format='png')
25         counter+=1
26     plt.show()

```

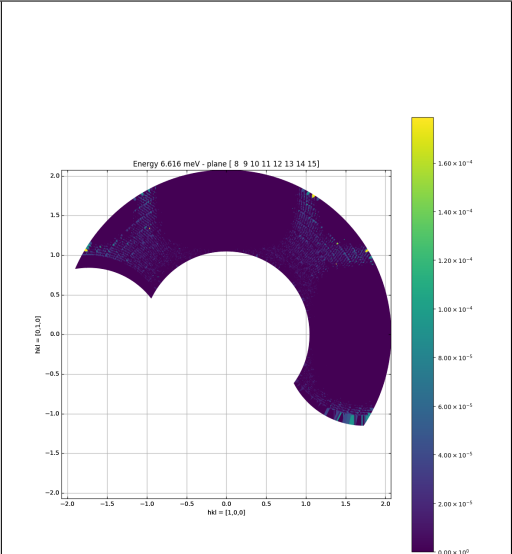
As shown above one can provide an axis or a list of axes into which the plot is to be made. This is especially useful if combined with the RLU axis method calculated in the DataSet object as one then gets the plot in reciprocal lattice units directly.

**Warning:** However, this method is really slow and takes approximately 3.5 minutes when combining 2 files and 8 planes.....

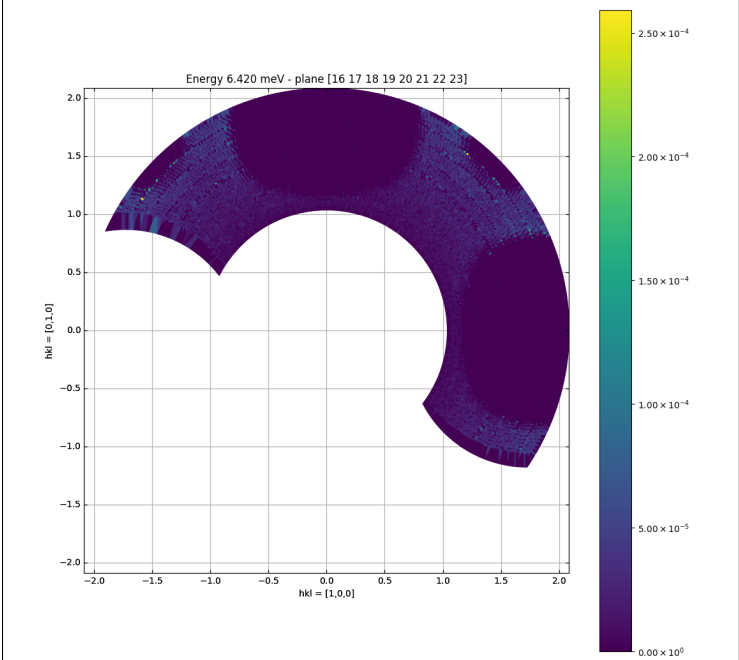
**Warning:** This method does not work fully yet due to inconsistencies in the calibration file of the data.



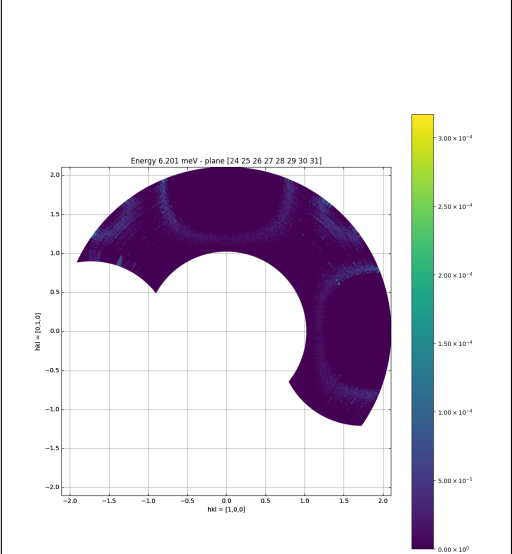
Binning of planes 0 through 7 into one plot. This figure is completely green due to no intensity measured during simulation.



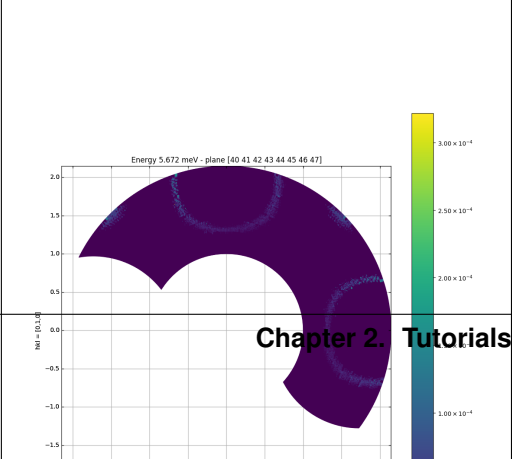
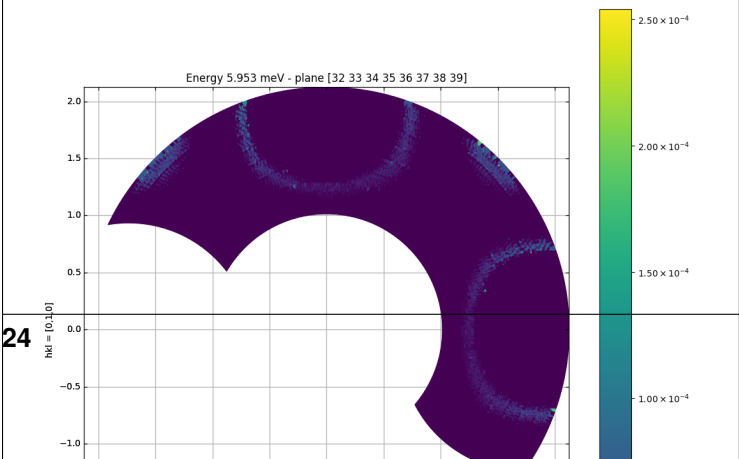
Binning of planes 8 through 15 into one plot.



Binning of planes 16 through 23 into one plot.



Binning of planes 24 through 31 into one plot.



## 2.2 Command Line Tutorials

Despite the full usability and customizability of using a scripting interface to the visualization software, during an experiment or in order to quickly get an overview of data one is more interested in just inspecting the data using a standardized set of plotting parameters. It could also be that Python is not a language the user masters and thus a command line interface might be easier when simple visualization is needed.

In any case, the following tutorials seek to introduce and explain the possibilities of using the command line scripts to quickly plot different parts of the data. However, running the scripts from the command line is operation system dependent. That is, if you are running either Linux or Mac, chances are that you can simply run:

```
python ScriptFile.py *args
```

or if python has been installed as an environment variable:

```
./ScriptFile.py *args
```

where in the latter case the shebang-command in the top of the scripting file is run.

### 2.2.1 Normalization inspection

Inspection of the current normalization table used to convert the raw data measured into actual scattering intensities is of great importance. It holds the information of energy and A4 value of all of the individual pixels and does thus govern the conversion into reciprocal space. Furthermore, it contains the normalization needed to take care of detector sensitivity, space angle coverage, as well as analyser efficiency. Due to this vital role, it has been made possible to easily plot the contents of the calibration table.

The script *CalibrationInspector.py* and it has the following help text:

```
$ python CalibrationInspector.py -h
usage: CalibrationInspector.py [-h] [-s SAVE] [-p [PLOTLIST [PLOTLIST ...]]]
                               [-b BINNING]
                               [DataFile]

Inspection tool to visualize calibration tables in a data file.

positional arguments:
  DataFile              Data file from which calibration table is to be
                        plotted. If none provided file dialog will appear.

optional arguments:
  -h, --help            show this help message and exit
  -s SAVE, --save SAVE  Location to which the generated file will be saved.
  -p [PLOTLIST [PLOTLIST ...]], --plot [PLOTLIST [PLOTLIST ...]]
                        List of wanted plots to be generated. Should be
                        "A4", "Normalization", "Ef", "EfOverview". Default "A4".
  -b BINNING, --binning BINNING
                        Binning to be inspected. Default '8'
```

First of all, if the program is run without any arguments, a file dialog will appear asking for a file to be plotted. From this file, the default plots will be created and the file directory is saved in an external settings file such that next time the program is run the file dialog will open on this location.

As seen in the description, the user can choose which plots are to be generated and whether to save the results or simply inspect them visually. As a default the script uses the 8 pixel binning and creates an A4 plot. This can be changed by invoking the *-b binning* or *-p PlotType* flags for binning and plotting respectively. The binning chosen should as

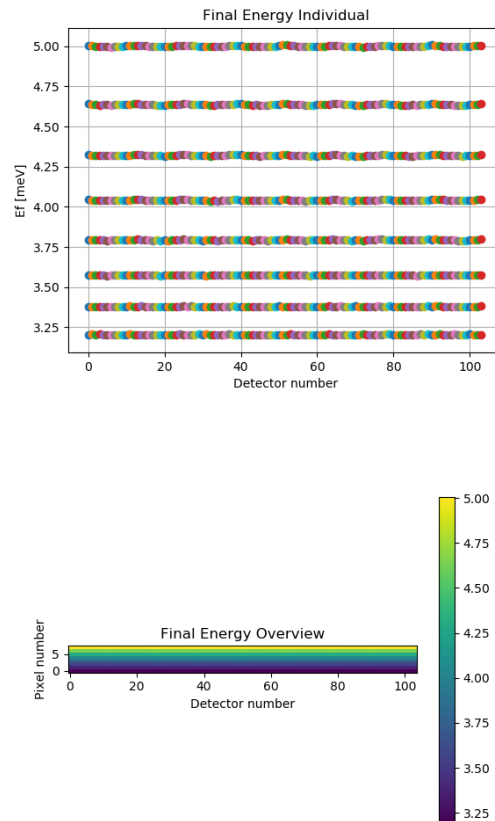
one of the provided in the file (usually 1, 3, or 8), and the plotting possibilities are “A4”, “Normalization”, “Ef”, and “EfOverview”.

Providing a saving location with the use of the `-s savelocation` or `-save savelocation` makes the script save the figures in the corresponding path. It should be a directory and not a file path. The figures are saved with selfexplanatory titles in a png format such as they can be opened on all OS’s without problems.

## Example

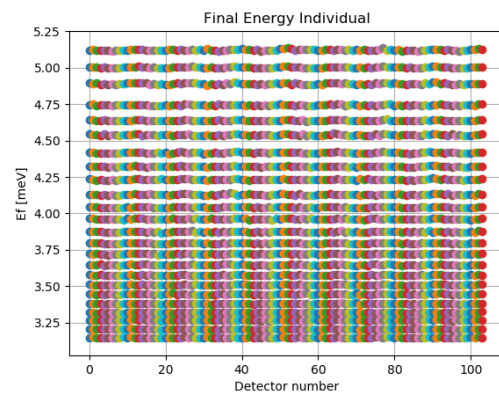
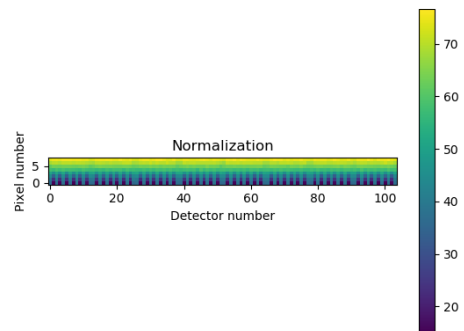
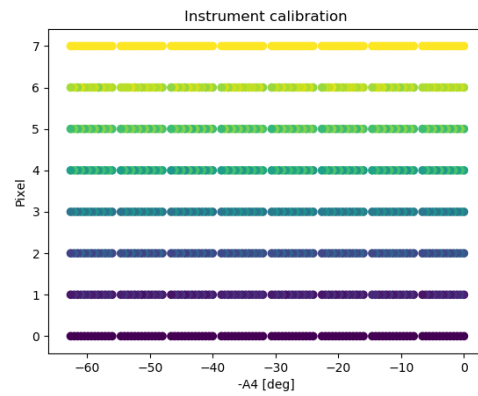
An example of the four different figures produces is the following, when choosing binning 1, 3, and 8:

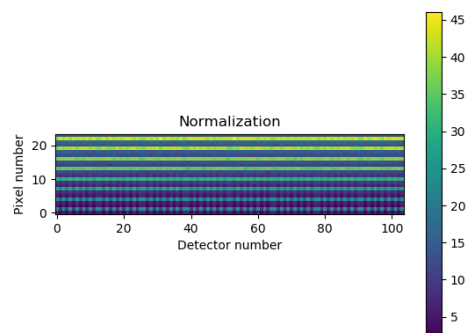
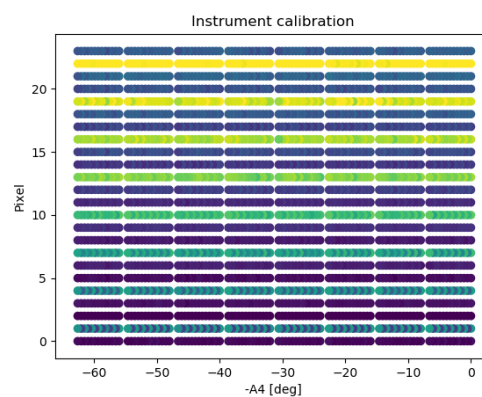
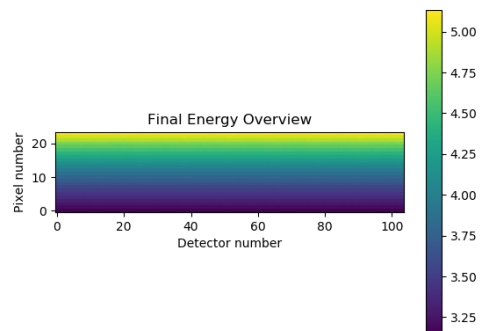
Binning 1:

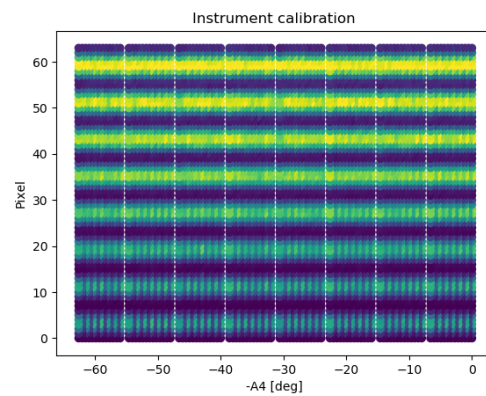
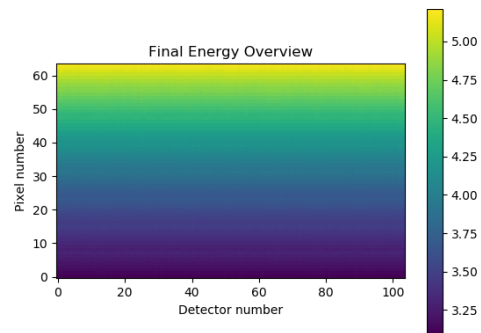
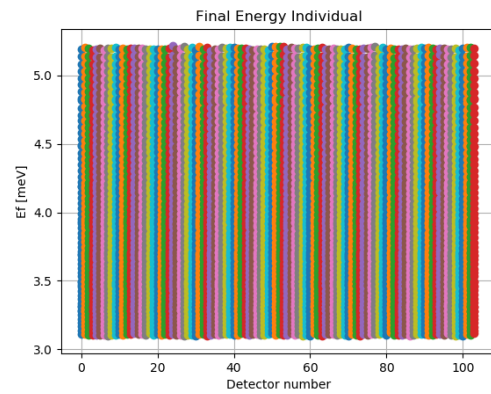


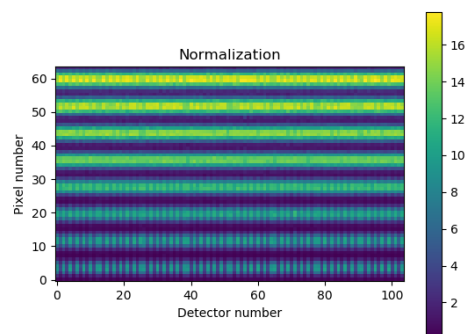
Binning 3

Binning 8











Something about MJOLNIR

## 3.1 Geometry Module

The geometry module is created in order to build a virtual copy of the instrument in the MJOLNIR. This is done using the following classes

<code>GeometryConcept.GeometryConcept</code>	Abstract geometry concept.
<code>GeometryConcept.GeometryObject</code>	Physical geometry object on which other physical MJOLNIR components are build.
<code>Detector.Detector</code>	Generic detector being the base class of all detectors.
<code>Detector.TubeDetector1D</code>	1D Tube detector used at PSI.
<code>Analyser.Analyser</code>	Generic analyser object.
<code>Analyser.FlatAnalyser</code>	Simple flat analyser.
<code>Wedge.Wedge</code>	Wedge object to keep track of analysers and detectors.
<code>Instrument.Instrument</code>	Instrument object used to calculated analytic scattering coverage.

Below is an extended description of the different classes and their methods.

### 3.1.1 Geometry base classes

General object from which other geometry objects inherits.

```
class GeometryConcept.GeometryConcept (position=(0, 0, 0))
    Abstract geometry concept. Used as base class for Wedge and Instrument.

    load (filename)
        Method to load an object from a pickled file.
```

**plot** (*ax*)

Args:

- *ax* (matplotlib axis): 3D matplotlib axis into which plotting is performed

**Warning:** Method not incorporated, but acts as virtual method.

**position**

Kwargs:

- Position (3vector): Position of object (default [0,0,0])

Raises:

- AttributeError
- NotImplementedError

```
>>> GenericConcept = GeometryConcept(position=(0.0, 1.0, 0.0))
>>> print(GenericConcept.position)
(0.0, 1.0, 0.0)
```

---

**class** `GeometryConcept.GeometryObject` (*position=(0.0, 0.0, 0.0), direction=(0, 0, 1)*)

Physical geometry object on which other physical MJOLNIR components are build. All of the components needed to create an instrument should inherit from this class in order enforce a uniform interface.

Kwargs:

- Position (3vector): Position of object (default [0,0,0])
- Direction (3vector): Direction along which the object points (default [0,0,1])

Raises:

- AttributeError

```
>>> GenericObject = GeometryObject(position=(0.0, 1.0, 0.0), direction=(1.0, 0, 0))
>>> print(GenericObject.position)
(0.0, 1.0, 0.0)
```

### 3.1.2 Detectors

**class** `Detector.Detector` (*position, direction*)

Generic detector being the base class of all detectors.

args:

- Position (3vector): Position of object (default [0,0,0])
- Direction (3vector): Direction along which the object points (default [0,0,1])

raises:

- NotImplementedError

**plot** (*ax, offset=(0.0, 0.0, 0.0)*)

Args:

- *ax* (matplotlib.pyplot 3d axis): Axis object into which the detector is plotted

Kwargs:

- offset (3vector): Offset of detector due to bank position (default [0,0,0])

```
>>> GenericDetector = Detector(position=(0.0,1.0,0.0),direction=(1.0,0,0))
>>> GenericDetector.plot(ax)
Plots detector tube in provided axis object.
```

**class** Detector.**TubeDetector1D**(*position, direction, length=0.25, pixels=1024, diameter=0.02, split=[]*)

1D Tube detector used at PSI. The detector is assumed to be a perfect cylinder consisting of pixels.

Args:

- Position (3vector): Position of object (default [0,0,0])
- Direction (3vector): Direction along which the object points (default [0,0,1])

Kwargs:

- length (float): Length of detector tube in meters (default 0.25)
- pixels (int): Number of pixels (default 1024)
- diameter (float): Diameter of tube in meters (default 0.02)
- split (list int): Edge pixels for slitting the tube into areas lidd by analysers (default [0,57,57\*2,57\*3,57\*4,57\*5,57\*6,57\*7,57\*8])

split (list int): Edge pixels for slitting the tube into areas lidd by analysers (default [0,57,57\*2,57\*3,57\*4,57\*5,57\*6,57\*7,57\*8])

Raises:

- AttributeError

**getPixelPositions** ()

Return pixel positions relative to center.

**plot** (*ax, offset=(0.0, 0.0, 0.0), n=100*)

Args:

- ax (matplotlib.pyplot 3d axis): Axis object into which the detector is plotted

Kwargs:

- offset (3vector): Offset of detector due to bank position (default [0,0,0])
- n (int): Number of points on the surface to be plotted (default 100)

```
>>> Detector = TubeDetector1D(position=(0.0,1.0,0.0),direction=(1.0,0,0))
>>> Detector.plot(ax,offset=(0.0,0.0,0.0),n=100)
Plots detector tube in provided axis object.
```

### 3.1.3 Analysers

**class** Analyser.**Analyser**(*position, direction, d\_spacing=3.35, mosaicity=60*)

Generic analyser object. Base class from which all analysers must inherit.

Args:

- position (float 3): Position of analyser in meters

- direction (float 3): Direction of analyser
- d\_spacing (float): The d spacing in Angstrom (default 3.35)
- mosaicity (float): The standard deviation of mosaicity in arcminutes (default 60)

**plot** (*ax*, *offset*=(0.0, 0.0, 0.0))

Args:

- ax (matplotlib.pyplot 3d axis): Axis object into which the analyser is plotted

Kwargs:

- offset (3vector): Offset of analyser due to bank position (default [0,0,0])

```
>>> GenericAnalyser = Analyser(position=(0.0, 1.0, 0.0), direction=(1.0, 0, 0))
>>> GenericAnalyser.plot(ax)
```

---

**class** `Analyser.FlatAnalyser` (*position*, *direction*, *d\_spacing*=3.35, *mosaicity*=60, *width*=0.05, *height*=0.1)

Simple flat analyser.

Args:

- Position (3vector): Position of object (default [0,0,0])
- Direction (3vector): Direction along which the object points (default [0,0,1])

Kwargs:

- d\_spacing (float): D spacing of analyser in Angstrom
- mosaicity (float): Mosaicity in arcminutes
- length (float): Length of detector tube in meters (default 0.25)
- pixels (int): Number of pixels (default 456)
- diameter (float): Diameter of tube in meters (default 0.02)

Raises:

- AttributeError
- NotImplementedError

**plot** (*ax*, *offset*=array([0, 0, 0]), *n*=100)

Args:

- ax (matplotlib.pyplot 3d axis): Axis object into which the analyser is plotted

Kwargs:

- offset (3vector): Offset of detector due to bank position (default [0,0,0])
- n (int): Number of points on the surface to be plotted (default 100)

```
>>> Analyser = FlatAnalyser(position=(0.0, 1.0, 0.0), direction=(1.0, 0, 0))
>>> Analyser.plot(ax, offset=(0.0, 0.0, 0.0), n=100)
```

### 3.1.4 Wedge

**class** `Wedge.Wedge` (*position*=(0.0, 0.0, 0.0), *detectors*=[], *analysers*=[], *concept*='ManyToMany', *\*\*kwargs*)

Wedge object to keep track of analysers and detectors. To be used as a storage object and facilitate easy movement of multiple detectors and analysers as once.

Args:

- *position* (float 3): Position of wedge (default (0,0,0))

Kwargs:

- *detectors* (list or single detector): Either a list or a single detector (default empty)
- *analysers* (list or single analyser): Either a list or a single analyser (default empty)
- *concept* (string "ManyToMany" or "OneToOne"): Setting to control if there is a "one to one" correspondence between analysers and detectors or a "many to many" relationship.

---

**Note:** A wedge does not have a direction. The direction of analysers and detectors are to be set individually.

---

**append** (*Object*)

Append Object(s) to corresponding list.

Args:

- *object* (Detector(s)/Analyser(s)): Single detector/analyser or list of detectors/analysers

**calculateDetectorAnalyserPositions** ()

Find neutron position on analyser and detector. Assuming that the analyser is in the z=0 plane.

**plot** (*ax*, *offset*=(0, 0, 0))

Recursive plotting routine.

### 3.1.5 Instrument

**class** `Instrument.Instrument` (*position*=(0, 0, 0), *wedges*=[], *fileName*="", *\*\*kwargs*)

Instrument object used to calculate analytic scattering coverage. Based on the GeometryConcept object it contains all needed information about the setup used in further calculations.

Kwargs:

- *position* (float 3d): Position of the instrument always at origin(?) (default (0,0,0))
- *wedges* (list of wedges or single wedge): Wedge or list of wedges which the instrument consists of (default empty)
- *fileName* (string): Filename of xml file (ending in xml). To load binary files use `self.load(filename)`.

Raises:

- `AttributeError`

**append** (*wedge*)

Append wedge(s) to instrument.

Args

- *wedge* (Wedge(s)): Single wedge or list of wedges

**generateCAMEXML** (*fileName*)

Generate CAMEA XML file to be used as instrument file.

Args:

- *fileName*: Name of file to be saved (required)

**generateCalibration** (*Vanadiumdatafile*, *A4datafile=False*, *savelocation='calibration'*, *tables=['Single', 'PrismaticLowDefinition', 'PrismaticHighDefinition']*, *plot=False*, *mask=True*)

Method to generate look-up tables for normalization. Saves calibration file(s) as 'Calibration\_Np.calib', where Np is the number of pixels.

Generates 4 different tables:

- Prismatic High Definition (8 pixels/energy or 64 pixels/detector)
- Prismatic Low Definition (3 pixels/energy or 24 pixels/detector)
- Single (1 pixel/energy or 8 pixels/detector)
- Number (integer)

Args:

- *Vanadiumdatafile* (string): String to single data file used for normalization, Vanadium Ei scan (required).

Kwargs:

- *A4datafile* (string): String to single data file used for normalization, ALO A4 scan (default False).
- *savelocation* (string): String to save location folder (calibration)
- *tables* (list): List of needed conversion tables (Default: ['Single', 'PrismaticLowDefinition', 'PrismaticHighDefinition'], increasing number of pixels).
- *plot* (boolean): Set to True if pictures of all fit are to be stored in *savelocation*
- *mask* (boolean): If True the lower 100 pixels are set to 0

**Warning:** At the moment, the active detector area is defined by *NumberOfSigmas* (currently 3) times the Guassian width of Vanadium peaks.

**initialize** ()

Method to initialize and perform analytical calculations of scattering quantities. Initializes:

- A4: Matrix holding pixel A4. Shape (len(Wedges),len(detectors),pixels)
- Ef: Matrix holding pixel Ef. Shape (len(Wedges),len(detectors),pixels)

**plot** (*ax*)

Recursive plotting routine.

**saveXML** (*fileName*)

Method for saving current file as XML in *fileName*.

## 3.2 Statistics Module

This is the documentation part intended for the explanation of the statistics module. Due to the philosophy of this software suit where data visualization and actual data fitting and feature extraction are decoupled, this section is

dedicated to explanation of the needed methods and workflows. The intended methods could include:

- **Fit using either Gaussian or Poisson statistics for:**
  - 1D extraction of data from two data points for constant energy
  - 1D extraction of data from 1 Q position and all energies
  - other..
- **Advanced fitting routines for multidimensional data (2, 3, or n(?) dimensions) using:**
  - Modified costfunctions to ensure continuity and breaks allowed for elastic peaks
  - Sequential fitting with blurring and rebinning of data to allow for initial fitting and guesses later used on the full data set
  - Constant signal to noise binning using voronoi tessellation to perform importance sampling
- Using swarm or other algorithms with all data fitting tool

### 3.2.1 Statistics Module

This is the statistics module.

## 3.3 Data Module

Something about the functionality of the Data module

### 3.3.1 Data Module

The DataSet object is the interface between the data files and the data treatment and visualization. It is both responsible for the conversion of raw '.h5'-files into '.nxs'-files as well as plotting these. Extracting values from this object results in a list of values where the first dimension is determined from the number of data files provided.

<code>DataSet.DataSet</code>	DataSet object to hold all informations about data.
<code>DataSet.DataSet.convertDataFile</code>	Conversion method for converting scan file(s) to hkl file.
<code>DataSet.DataSet.cut1D</code>	Wrapper for 1D cut through constant energy plane from q1 to q2 function returning binned intensity, monitor, normalization and normcount.
<code>DataSet.DataSet.plotCut1D</code>	Plotting wrapper for the cut1D method.
<code>DataSet.DataSet.cutQE</code>	Wrapper for cut data into maps of q and intensity between two q points and given energies.
<code>DataSet.DataSet.plotCutQE</code>	Plotting wrapper for the cutQE method.
<code>DataSet.DataSet.cutPowder</code>	Cut data powder map with intensity as function of the length of q and energy.
<code>DataSet.DataSet.plotCutPowder</code>	Plotting wrapper for the cutPowder method.
<code>DataSet.DataSet.createRLUAxes</code>	Wrapper for the createRLUAxes method.
<code>DataSet.DataSet.plotQPlane</code>	Wrapper for plotting tool to show binned intensities in the Q plane between provided energies.
<code>DataSet.DataSet.cutQELine</code>	Method to perform Q-energy cuts from a variable number of points.
<code>DataSet.DataSet.plotCutQELine</code>	Plotting wrapper for the cutQELine method.

Continued on next page

Table 2 – continued from previous page

<code>DataSet.binData3D</code>	3D binning of data.
<code>DataSet.boundaryQ</code>	Calculate the boundary of a given scan in Q space A4Extend: in degrees A3Extend: in degrees
<code>DataSet.calculateGrid3D</code>	Generate 3D grid with centers given by X,Y, and Z.
<code>DataSet.createRLUAxes</code>	Create a reciprocal lattice plot for a given DataSet object.
<code>DataSet.cut1D</code>	Perform 1D cut through constant energy plane from q1 to q2 returning binned intensity, monitor, normalization and normcount.
<code>DataSet.cut1DE</code>	Perform 1D cut through constant Q point returning binned intensity, monitor, normalization and normcount.
<code>DataSet.cutPowder</code>	Cut data powder map with intensity as function of the length of q and energy.
<code>DataSet.cutQE</code>	Cut data into maps of q and intensity between two q points and given energies.
<code>DataSet.plotCut1D</code>	Plotting wrapper for the cut1D method.
<code>DataSet.plotCutPowder</code>	Plotting wrapper for the cutPowder method.
<code>DataSet.plotCutQE</code>	Plotting wrapper for the cutQE method.
<code>DataSet.plotQPlane</code>	Plotting tool to show binned intensities in the Q plane between provided energies.
<code>DataFile.DataFile</code>	Object to load and keep track of HdF files and their conversions

This is the data module intended to be used as a stand-alone data reduction and visualization.

## DataSet Object and Methods

Object to take care of all data conversion and treatment taking it from raw hdf5 files obtained at the instrument into rebinned data sets converted to  $S(q, \omega)$ .

**class** `DataSet.DataSet` (*dataFiles=None, normalizationfiles=None, calibrationfiles=None, convertedFiles=None, \*\*kwargs*)

DataSet object to hold all informations about data.

Kwargs:

- `dataFiles` (string, DataFile or list of strings or DataFiles): List of datafiles or DataFile objects to be used in conversion (default None).
- `normalizationfiles` (string or list of strings): Location of Vanadium normalization file(s) (default None).
- `calibrationfiles` (string or list of strings): Location of calibration normalization file(s) (default None).
- `convertedFiles` (string, DataFile or list of strings): Location of converted data files (default None).

Raises:

- `ValueError`
- `NotImplementedError`

**View3D** (*dx, dy, dz, rlu=True, log=False, grid=False*)

View data in the Viewer3D object.

Args:

- `dx` (float): step size in qx



- `dx` (float): step size in `qx`
- `dz` (float): step size in `E`

Kwargs:

- `rlu` (Bool): If true a `rlu` axis is used for plotting otherwise `qx, qy` (Default True).
- `log` (Bool): If true logarithm of intensity is plotted

If one plots not using RLU, everything is plotted in real units ( $1/AA$ ), and the `Qx` and `Qy` is not rotated. That is, the `x` axis in energy is not along the projection vector. The cuts of constant `Qx` and `Qy` does not represent any symmetry directions in the sample. However, if one utilizes the RLU flag, first `Qx` and `Qy` are rotated with first HKL vector along the `x`-axis. This thus means that cuts of constant `Qx` (or more correctly along the principal HKL vector) represents a symmetry direction. However, as the data is binned in equal sized voxels, constant `Qy` does not necessarily correspond to HKL vector 2 (it will in systems with 90 degrees between the two vectors).

**binData3D** (*dx, dy, dz, rlu=True, dataFiles=None*)

Bin a converted data file into voxels with sizes `dx*dy*dz`. Wrapper for the `binData3D` functionality.

Args:

- `dx` (float): step sizes along the `x` direction (required).
- `dy` (float): step sizes along the `y` direction (required).
- `dz` (float): step sizes along the `z` direction (required).

Kwargs:

- `rlu` (bool): If True, the rotate `QX, QY` is used for binning (default True)
- `datafile` (string or list of strings): Location(s) of data file to be binned (default converted file in `DataSet`).

Raises:

- `AttributeError`

Returns:

- `Datalist`: List of converted data files having 4 sub arrays: Intensity(counts), Monitor, Normalization, Normalization count
- `bins`: 3 arrays containing edge positions in `x`, `y`, and `z` directions.

**convertDataFile** (*dataFiles=None, binning=8, saveLocation=None, saveFile=True*)

Conversion method for converting scan file(s) to `hkl` file. Converts the given `hdf` file into `NXSqom` format and saves in a file with same name, but of type `.nxs`. Copies all of the old data file into the new to ensure complete redundancy. Determines the binning wanted from the file name of normalization file.

Kwargs:

- `dataFiles` (`DataFile`, string or list of): File path(s), file must be of `hdf` format (default `self.dataFiles`).
- `binning` (int): Binning to be used when converting files (default 8).
- `saveLocation` (string): File path to save location of data file(s) (defaults to same as raw file).
- `saveFile` (bool): If true, the file(s) will be saved as `nxs`-files. Otherwise they will only persist in memory.

Raises:

- `IOError`

- AttributeError

**convertToQxQy** (*HKL*)

Convert array or vector of HKL point(s) to corresponding Qx and QY

Args:

- HKL (array): array or vector of HKL point(s)

Returns

- Q (array): Converted HKL points in Qx QY of unrotated coordinate system.

**createQEAxes** (*axis=0, figure=None*)

Wrapper for the createRLUAxes method.

Returns:

- ax (Matplotlib axes): Created reciprocal lattice axes.

---

**Note:** Uses sample from the first converted data file. However, this should be taken care of by the comparison of datafiles to ensure same sample and settings.

---

**createRLUAxes** (*figure=None*)

Wrapper for the createRLUAxes method.

Returns:

- ax (Matplotlib axes): Created reciprocal lattice axes.

---

**Note:** Uses sample from the first converted data file. However, this should be taken care of by the comparison of datafiles to ensure same sample and settings.

---

**cut1D** (*q1, q2, width, minPixel, Emin, Emax, rlu=True, plotCoverage=False, extend=True, dataFiles=None*)

Wrapper for 1D cut through constant energy plane from q1 to q2 function returning binned intensity, monitor, normalization and normcount. The full width of the line is width while height is given by Emin and Emax. the minimum step sizes is given by minPixel.

---

**Note:** Can only perform cuts for a constant energy plane of definable width.

---

Args:

- q1 (3D or 2D array): Start position of cut in format (h,k,l) or (qx,qy) depending on rlu flag.
- q2 (3D or 2D array): End position of cut in format (h,k,l) or (qx,qy) depending on rlu flag.
- width (float): Full width of cut in q-plane in 1/AA.
- minPixel (float): Minimal size of binning along the cutting direction. Points will be binned if they are closer than minPixel.
- Emin (float): Minimal energy to include in cut.
- Emax (float): Maximal energy to include in cut

Kwargs:

- rlu (bool): If True, coordinates given are interpreted as (h,k,l) otherwise as (qx,qy)

- `plotCoverage` (bool): If True, generates plot of all points in the cutting plane and adds bounding box of cut (default False).
- `extend` (bool): Whether or not the cut from `q1` to `q2` is to be extended throughout the data (default true)
- `dataFiles` (list): List of dataFiles to cut (default None). If none, the ones in the object will be used.

Returns:

- Data list (4 arrays): Intensity, monitor count, normalization and normalization counts binned in the 1D cut.
- Bin list (3 arrays): Bin edge positions in plane of size (n+1,3), orthogonal positions of bin edges in plane of size (2,2), and energy edges of size (2).

**cut1DE** (*E1, E2, q, rlu=True, width=0.02, minPixel=0.1, dataFiles=None*)

Perform 1D cut through constant Q point returning binned intensity, monitor, normalization and norm-count. The width of the cut is given by the width attribute.

---

**Note:** Can only perform cuts for a constant energy plane of definable width.

---

Args:

- `E1` (float): Start energy.
- `E2` (float): End energy.
- `q` (3D or 2D vector): Q point

Kwargs:

- `rlu` (bool): If True, provided Q point is interpreted as (h,k,l) otherwise as (qx,qy), (Default true)
- `width` (float): Full width of cut in q-plane (default 0.02).
- `minPixel` (float): Minimal size of binning along the cutting direction. Points will be binned if they are closer than minPixel (default 0.1).
- `dataFiles` (list): Data files to be used. If none provided use the ones in self (default None)

Returns:

- Data list (4 arrays): Intensity, monitor count, normalization and normalization counts binned in the 1D cut.
- Bin list (1 array): Bin edge positions in energy

**cutPowder** (*EBinEdges, qMinBin=0.01, dataFiles=None*)

Cut data powder map with intensity as function of the length of q and energy.

Args:

- `EBinEdges` (list): Bin edges between which the cuts are performed.

Kwargs:

- `qMinBin` (float): Minimal size of binning along q (default 0.01). Points will be binned if they are closer than qMinBin.
- `dataFiles` (list): List of dataFiles to cut (default None). If none, the ones in the object will be used.

Returns:

- Data list (n \* 4 arrays): n instances of [Intensity, monitor count, normalization and normalization counts].
- qbins (n arrays): n arrays holding the bin edges along the length of q

**cutQE** (*q1, q2, width, minPixel, EnergyBins, rlu=True, extend=True, dataFiles=None*)

Wrapper for cut data into maps of q and intensity between two q points and given energies. This is performed by doing consecutive constant energy planes.

Args:

- q1 (3D or 2D array): Start position of cut in format (h,k,l) or (qx,qy) depending on rlu flag.
- q2 (3D or 2D array): End position of cut in format (h,k,l) or (qx,qy) depending on rlu flag.
- width (float): Full width of cut in q-plane.
- minPixel (float): Minimal size of binning along the cutting direction. Points will be binned if they are closer than minPixel.
- EnergyBins (list): Bin edges between which the 1D constant energy cuts are performed.

Kwargs:

- rlu (bool): If True, coordinates given are interpreted as (h,k,l) otherwise as (qx,qy)
- extend (bool): If True, cut is extended to edge of measured area instead of only between provided points.
- dataFiles (list): List of dataFiles to cut (default None). If none, the ones in the object will be used.

Returns:

- Data list (n \* 4 arrays): n instances of [Intensity, monitor count, normalization and normalization counts].
- Bin list (n \* 3 arrays): n instances of bin edge positions in plane of size (m+1,3), orthogonal positions of bin edges in plane of size (2,2), and energy edges of size (2).
- center position (n \* 3D arrays): n instances of center positions for the bins.
- binDistance (n arrays): n instances of arrays holding the distance in q to q1.

**cutQELine** (*QPoints, EnergyBins, width=0.1, minPixel=0.01, rlu=True, dataFiles=None*)

Method to perform Q-energy cuts from a variable number of points. The function takes both qx/qy or hkl positions. In the case of using only two Q points, the method is equivalent to cutQE.

Args:

- QPoints (list of points): Q positions between which cuts are performed. Can be specified with both qx, qy or hkl positions dependent on the choice of format.
- EnergyBins (list of floats): Energy bins for which the cuts are performed

Kwargs:

- width (float): Width of the cut in 1/A (default 0.1).
- minPixel (float): Minimal size of binning along the cutting directions. Points will be binned if they are closer than minPixel (default=0.01)
- rlu (bool): If True, provided QPoints are interpreted as (h,k,l) otherwise as (qx,qy), (default True).
- dataFiles (list): List of dataFiles to cut. If none, the ones in the object will be used (default None).

**Warning:** The way the binning works is by extending the end points with  $0.5 * \text{minPixel}$ , but the method sorts away points not between the two Q points given and thus the start and end bins are only half filled. This might result in discrepancies between a single cut and the same cut split into different steps. Further, splitting lines into sub-cuts forces a new binning to be done and the bin positions can then differ from the case where only one cut is performed.

Returns:  $m = Q$  points,  $n = \text{energy bins}$

- Data list ( $m * n * 4$  arrays):  $n$  instances of [Intensity, monitor count, normalization and normalization counts].
- Bin list ( $m * n * 3$  arrays):  $n$  instances of bin edge positions in plane of size  $(m+1,3)$ , orthogonal positions of bin edges in plane of size  $(2,2)$ , and energy edges of size  $(2)$ .
- center position ( $m * n * 3D$  arrays):  $n$  instances of center positions for the bins.
- binDistance ( $m * n$  arrays):  $n$  instances of arrays holding the distance in  $q$  to  $q1$ .

---

**Note:** If an HKL point outside of the scattering plane is given, the program will just take the projection onto the scattering plane.

---

**extractData** (*A4=None, A4Id=None, Ef=None, EfId=None, raw=False, A4Tolerance=0.1, EfTolerance=0.1*)

Extract data given A4 value and Ef (or the corresponding indices).

Kwargs:

- A4 (float): Wanted A4 value in degrees (default None)
- A4Id (int): Id of wedge which is a number between 0 and 103 (default None)
- Ef (float): Wanted Ef value in meV (default None)
- EfId (int): Wanted Id of analyser energy, number between 0-7 (default None)
- raw (bool): If true method returns Intensity,Normalization,Monitor, else returns Intensity/(Norm\*Monitor) (default False)
- A4Tolerance (float): Tolerance between found and wanted A4 value in degrees (default 0.1)
- EfTolerance (float): Tolerance between found and wanted Ef value in meV (default 0.1)

---

**Note:** If A4 or Ef is provided, then these will be used instead of A4Id or EfId.

---

**plotA3A4** (*dataFiles=None, ax=None, planes=[], log=False, returnPatches=False, binningDecimals=3, singleFigure=False, plotTessellation=False, Ei\_err=0.05, temperature\_err=0.2, magneticField\_err=0.2, electricField\_err=0.2*)

Plot data files together with pixels created around each point in A3-A4 space. Data is binned in the specified planes through their A3 and A4 values. This can result in distorted binning when binning across large energy regions. Data is plotted using the pixels calculated for average plane value, i.e. binning 7,8,9,10, and 11 patches for plane 9 are used for plotting.

Kwargs:

- dataFiles (DataFiles): single file or list of files to be binned together (Default self.convertedFiles)
- ax (matplotlib axis): Axis into which the planes are to be plotted (Default None, i.e. new)
- planes (list (of lists)): Planes to be plotted and binned (default [])

- `log` (bool): Whether or not to plot intensities as logarithm (default False)
- `returnPatches` (bool): If true the method returns the patches otherwise plotted in the given axis (default False)
- `binningDecimals` (int): Number of decimal places Q positions are rounded before binning (default 3)
- `singleFigure` (bool): If true, all planes are plotted in same figure (default False)
- `plotTessellation` (bool): Plot Tessellation of points (default False)
- `Ei_err` (float): Tolerance of  $E_i$  for which the values are equal (default = 0.05)
- `temperature_err` (float): Tolerance of temperature for which the values are equal (default = 0.2)
- `magneticField_err` (float): Tolerance of magnetic field for which the values are equal (default = 0.2)
- `electricField_err` (float): Tolerance of electric field for which the values are equal (default = 0.2)

Returns:

- `ax` (matplotlib axis or list of): axis (list of) containing figures for plotted planes.

Raises:

- `NotImplementedError`

Examples:

The following example will combine the two files and plot all of the available planes in different figures.

```
>>> DS = DataSet.DataSet(convertedFiles=[--.nxs, ---.nxs])
>>> plt.figure()
>>> ax = plt.gca()
>>> DataSet.plotA3A4(DS.convertedFiles, ax=ax)
```

If only a subset of planes or different planes are to be combined the following will achieve this:

```
>>> DataSet.plotA3A4(DS.convertedFiles, ax=ax, planes=[0, 1, 2, 3, [4, 5, 6], [8, 9]])
```

Here planes 0 through 3 are plotted separately while 4,5, and 6 as well as 8 and 9 are binned.

---

**Note:** Binning planes from different analysers might result in nonsensible binnings.

---

**plotCut1D** (*q1*, *q2*, *width*, *minPixel*, *Emin*, *Emax*, *rlu=True*, *ax=None*, *plotCoverage=False*, *extend=True*, *dataFiles=None*, *\*\*kwargs*)

Plotting wrapper for the cut1D method. Generates a 1D plot with bins at positions corresponding to the distance from the start point. Adds the 3D position on the x axis with ticks.

---

**Note:**

Can only perform cuts for a constant energy plane of definable width.

Args:

- `q1` (3D or 2D array): Start position of cut in format (h,k,l) or (qx,qy) depending on `rlu` flag.
- `q2` (3D or 2D array): End position of cut in format (h,k,l) or (qx,qy) depending on `rlu` flag.
- `width` (float): Full width of cut in q-plane in 1/Å.

- minPixel (float): Minimal size of binning along the cutting direction. Points will be binned if they are closer than minPixel.
  - Emin (float): Minimal energy to include in cut.
  - Emax (float): Maximal energy to include in cut
- 

Kwargs:

- rlu (bool): If True, coordinates given are interpreted as (h,k,l) otherwise as (qx,qy)
- ax (matplotlib axis): Figure axis into which the plots should be done (default None). If not provided, a new figure will be generated.
- kwargs: All other keywords will be passed on to the ax.errorbar method.
- dataFiles (list): List of dataFiles to cut (default None). If none, the ones in the object will be used.

Returns:

- ax (matplotlib axis): Matplotlib axis into which the plot was put.
- Data list (4 arrays): Intensity, monitor count, normalization and normalization counts binned in the 1D cut.
- Bin list (3 arrays): Bin edge positions in plane of size (n+1,3), orthogonal positions of bin edges in plane of size (2,2), and energy edges of size (2).
- binCenter (3D array): Array containing the position of the bin centers of size (n,3)
- binDistance (array): Distance from centre of bins to start position.

**plotCutPowder** (*EBinEdges*, *qMinBin=0.01*, *ax=None*, *dataFiles=None*, *\*\*kwargs*)

Plotting wrapper for the cutPowder method. Generates a 2D plot of powder map with intensity as function of the length of q and energy.

---

**Note:** Can only perform cuts for a constant energy plane of definable width.

---

Args:

- EBinEdges (list): Bin edges between which the cuts are performed.

Kwargs:

- qMinBin (float): Minimal size of binning along q (default 0.01). Points will be binned if they are closer than qMinBin.
- ax (matplotlib axis): Figure axis into which the plots should be done (default None). If not provided, a new figure will be generated.
- dataFiles (list): List of dataFiles to cut (default None). If none, the ones in the object will be used.
- kwargs: All other keywords will be passed on to the ax.pcolor mesh method.

Returns:

- ax (matplotlib axis): Matplotlib axis into which the plot was put.
- Data list (4 arrays): Intensity, monitor count, normalization and normalization counts binned in the 1D cut.
- Bin list (3 arrays): Bin edge positions in plane of size (n+1,3), orthogonal positions of bin edges in plane of size (2,2), and energy edges of size (2).

**plotCutQE** (*q1, q2, width, minPixel, EnergyBins, rlu=True, ax=None, dataFiles=None, \*\*kwargs*)  
Plotting wrapper for the cutQE method. Generates a 2D intensity map with the data cut by cutQE.

---

**Note:** Positions shown in tool tip reflect the closes bin center and are thus limited to the area where data is present.

---

Args:

- *q1* (3D or 2D array): Start position of cut in format (h,k,l) or (qx,qy) depending on *rlu* flag.
- *q2* (3D or 2D array): End position of cut in format (h,k,l) or (qx,qy) depending on *rlu* flag.
- *width* (float): Full width of cut in q-plane.
- *minPixel* (float): Minimal size of binning along the cutting direction. Points will be binned if they are closer than *minPixel*.
- *EnergyBins* (list): Bin edges between which the 1D constant energy cuts are performed.

Kwargs:

- *rlu* (bool): If True, coordinates given are interpreted as (h,k,l) otherwise as (qx,qy)
- *ax* (matplotlib axis): Figure axis into which the plots should be done (default None). If not provided, a new figure will be generated.
- *dataFiles* (list): List of dataFiles to cut (default None). If none, the ones in the object will be used.
- *kwargs*: All other keywords will be passed on to the *ax.errorbar* method.

Returns:

- *ax* (matplotlib axis): Matplotlib axis into which the plot was put.
- Data list (*n* \* 4 arrays): *n* instances of [Intensity, monitor count, normalization and normalization counts].
- Bin list (*n* \* 3 arrays): *n* instances of bin edge positions in plane of size (m+1,3), orthogonal positions of bin edges in plane of size (2,2), and energy edges of size (2).
- center position (*n* \* 3D arrays): *n* instances of center positions for the bins.
- binDistance (*n* arrays): *n* instances of arrays holding the distance in q to *q1*.

**plotCutQELine** (*QPoints, EnergyBins, width=0.1, minPixel=0.01, rlu=True, ax=None, dataFiles=None, \*\*kwargs*)

Plotting wrapper for the cutQELine method. Plots the scattering intensity as a function of Q and E for cuts between specified Q-points.

Args:

- *QPoints* (list): List of Q points in either RLU (3D) or QxQy (2D).
- *EnergyBins* (list): List of bin edges in the energy direction.

Kwargs:

- *width* (float): Width in Q-direction for cuts (default 0.01)
- *rlu* (bool): If True, provided points are interpreted as (h,k,l) otherwise (qx,qy), (Default RLU)
- *ax* (matplotlib axis): Axis into which the data is plotted. If None a new will be created (default None).
- *dataFiles* (DataFile(s)): DataFile or list of, from which data is to be taken. If None all datafiles in self is taken (default None).



- vmin (float): Lower limit for colorbar (default min(Intensity)).
- vmax (float): Upper limit for colorbar (default max(Intensity)).
- tickRound (int): Number of decimals ticks are rounded to (default 3).
- ticks (int): Number of ticks in plot, minimum equal to number of Q points (default 10).
- plotSeperator (bool): If true, vertical lines are plotted at Q points (default True).
- seperatorWidth (float): Width of seperator line (default 2).
- log (bool): If true the plotted intensity is the logarithm of the intensity (default False)

Return: m = Q points, n = energy bins

- ax: matplotlib axis in which the data is plotted
- Data list (m \* n \* 4 arrays): n instances of [Intensity, monitor count, normalization and normalization counts].
- Bin list (m \* n \* 3 arrays): n instances of bin edge positions in plane of size (m+1,3), orthogonal positions of bin edges in plane of size (2,2), and energy edges of size (2).
- center position (m \* n \* 3D arrays): n instances of center positions for the bins.
- binDistance (m \* n arrays): n instances of arrays holding the distance in q to q1.

---

**Note:** The ax.set\_clim function is created to change the color scale. It takes inputs vmin,vmax.

---

**plotQPlane** (*EMin, EMax, binning='xy', xBinTolerance=0.05, yBinTolerance=0.05, enlargen=False, log=False, ax=None, RLUPlot=True, dataFiles=None, \*\*kwargs*)

Wrapper for plotting tool to show binned intensities in the Q plane between provided energies.

Args:

- EMin (float): Lower energy limit.
- EMax (float): Upper energy limit.

Kwargs:

- binning (str): Binning scheme, either 'xy' or 'polar' (default 'xy').
- xBinTolerance (float): bin sizes along x direction (default 0.05). If enlargen is true, this is the minimum bin size.
- yBinTolerance (float): bin sizes along y direction (default 0.05). If enlargen is true, this is the minimum bin size.
- enlargen (bool): If the bin sizes should be adaptive (default False). If set true, bin tolerances are used as minimum bin sizes.
- log (bool): Plot intensities as the logarithm (default False).
- ax (matplotlib axes): Axes in which the data is plotted (default None). If None, the function creates a new axes object.
- RLUPlot (bool): If true and axis is None, a new reciprocal lattice axis is created and used for plotting (default True).
- other: Other key word arguments are passed to the pcolormesh plotting algorithm.

Returns:

- ax (matplotlib axes)

---

**Note:** The axes object has a new method denoted ‘set\_clim’ taking two parameters (VMin and VMax) used to change axes coloring.

---

## Module Functions

The following is a list of the available functions in the DataSet module. Some of them have wrappers in the DataSet-object methods.

DataSet.**OxfordList** (*list*)

Create a comma separated string from the strings provided with last comma trailed by ‘and’.

DataSet.**binData3D** (*dx, dy, dz, pos, data, norm=None, mon=None, bins=None*)  
3D binning of data.

Args:

- dx (float): Step size in x (required).
- dy (float): Step size in x (required).
- dz (float): Step size in x (required).
- pos (2D array): Position of data points as flattened lists (X,Y,Z) (required).
- data (array): Flattened data array (required).

Kwargs:

- norm (array): Flattened normalization array.
- mon (array): Flattened monitor array.
- bins (list of arrays): Bins locating edges in the x, y, and z directions.

returns:

Rebinned intensity (and if provided Normalization, Monitor, and Normalization Count) and X, Y, and Z bins in 3 3D arrays.

Example:

```
>>> pos = [Qx,Qy,E]
>>> Data,bins = DataSet.binData3D(0.05,0.05,0.2,pos,I,norm=Norm,mon=Monitor)
```

DataSet.**boundaryQ** (*file, plane, A4Extend=0.0, A3Extend=0.0*)

Calculate the boundary of a given scan in Q space A4Extend: in degrees A3Extend: in degrees

DataSet.**calculateGrid3D** (*X, Y, Z*)

**Generate 3D grid with centers given by X,Y, and Z.** Args:

X (3D array): 3D array of x values generated by np.meshgrid.

Y (3D array): 3D array of y values generated by np.meshgrid.

Z (3D array): 3D array of z values generated by np.meshgrid.

Example:

```

>>> x = np.linspace(-1.5,1.5,20)
>>> y = np.linspace(0,1.5,10)
>>> z = np.linspace(-1.0,5.5,66)
>>> X,Y,Z = np.meshgrid(x,y,z,indexing='ij')
>>> XX,YY,ZZ = calculateGrid3D(X,Y,Z)

```

Now XX is a 21x11x67 array containing all x coordinates of the edges exactly midway between the points. Same goes for YY and ZZ with y and z coordinates respectively.

`DataSet.convertToQxQy (sample, QPoints)`

Convert a given list of QPoints to QxQy from UB matrix of sample

Args:

- sample (MJOLNIR.DataFile.Sample): Sample from which the UB matrix is to be used
- QPoints (list): List of HKL point to be converted

Returns:

- Q (list): List of QxQy points in same shape as provided

`DataSet.convexHullPoints (A3, A4)`

Calculate the convex hull of rectangularly spaced A3 and A4 values

`DataSet.createQEAxes (Dataset, axis=0, figure=None)`

Function to create Q E plot

Kwargs:

- axis (int): Whether to create axis 0 or 1 (projection vector 0 or orthogonal to this, default 0)
- figure

`DataSet.createRLUAxes (Dataset, figure=None)`

Create a reciprocal lattice plot for a given DataSet object.

Args:

- Dataset (DataSet): DataSet object for which the RLU plot is to be made.

Returns:

- ax (Matplotlib axes): Axes containing the RLU plot.

`DataSet.cut1D (positions, I, Norm, Monitor, q1, q2, width, minPixel, Emin, Emax, plotCoverage=False, extend=True)`

Perform 1D cut through constant energy plane from q1 to q2 returning binned intensity, monitor, normalization and normcount. The full width of the line is width while height is given by Emin and Emax. the minimum step sizes is given by minPixel.

---

**Note:** Can only perform cuts for a constant energy plane of definable width.

---

Args:

- positions (3 arrays): position in Qx, Qy, and E in flattened arrays.
- I (array): Flatten intensity array
- Norm (array): Flatten normalization array
- Monitor (array): Flatten monitor array
- q1 (2D array): Start position of cut in format (qx,qy).

- **q2** (2D array): End position of cut in format (qx,qy).
- **width** (float): Full width of cut in q-plane.
- **minPixel** (float): Minimal size of binning along the cutting direction. Points will be binned if they are closer than minPixel.
- **Emin** (float): Minimal energy to include in cut.
- **Emax** (float): Maximal energy to include in cut

Kwargs:

- **plotCoverage** (bool): If True, generates plot of all points in the cutting plane and adds bounding box of cut (default False).
- **extend** (bool): Whether or not the cut from q1 to q2 is to be extended throughout the data (default true)

Returns:

- **Data list** (4 arrays): Intensity, monitor count, normalization and normalization counts binned in the 1D cut.
- **Bin list** (3 arrays): Bin edge positions in plane of size (n+1,3), orthogonal positions of bin edges in plane of size (2,2), and energy edges of size (2).

`DataSet.cut1DE` (*positions, I, Norm, Monitor, E1, E2, q, width, minPixel*)

Perform 1D cut through constant Q point returning binned intensity, monitor, normalization and normcount. The width of the cut is given by the width attribute.

---

**Note:** Can only perform cuts for a constant energy plane of definable width.

---

Args:

- **positions** (3 arrays): position in Qx, Qy, and E in flattend arrays.
- **I** (array): Flatten intensity array
- **Norm** (array): Flatten normalization array
- **Monitor** (array): Flatten monitor array
- **E1** (float): Start energy.
- **E2** (float): End energy.
- **q** (2d vector): Q point in (qx,qy)
- **width** (float): Full width of cut in q-plane.
- **minPixel** (float): Minimal size of binning along the cutting direction. Points will be binned if they are closer than minPixel.
- **Emin** (float): Minimal energy to include in cut.
- **Emax** (float): Maximal energy to include in cut

Returns:

- **Data list** (4 arrays): Intensity, monitor count, normalization and normalization counts binned in the 1D cut.
- **Bin list** (1 array): Bin edge positions in energy

`DataSet.cutPowder` (*positions, I, Norm, Monitor, EBinEdges, qMinBin=0.01*)

Cut data powder map with intensity as function of the length of q and energy.

Args:

- positions (3 arrays): position in Qx, Qy, and E in flattend arrays.
- I (array): Flatten intensity array
- Norm (array): Flatten normalization array
- Monitor (array): Flatten monitor array
- EBinEdges (list): Bin edges between which the cuts are performed.

Kwargs:

- qMinBin (float): Minimal size of binning along q (default 0.01). Points will be binned if they are closer than qMinBin.

Returns:

- Data list (n \* 4 arrays): n instances of [Intensity, monitor count, normalization and normalization counts].
- qbins (n arrays): n arrays holding the bin edges along the lenght of q

`DataSet.cutQE(positions, I, Norm, Monitor, q1, q2, width, minPix, EnergyBins, extend=True)`

Cut data into maps of q and intensity between two q points and given energies. This is performed by doing consecutive constant energy planes.

Args:

- positions (3 arrays): position in Qx, Qy, and E in flattend arrays.
- I (array): Flatten intensity array
- Norm (array): Flatten normalization array
- Monitor (array): Flatten monitor array
- q1 (2D array): Start position of cut in format (qx,qy).
- q2 (2D array): End position of cut in format (qx,qy).
- width (float): Full width of cut in q-plane.
- minPixel (float): Minimal size of binning aling the cutting direction. Points will be binned if they are closer than minPixel.
- EnergyBins (list): Bin edges between which the 1D constant energy cuts are performed.

Kwargs:

- extend (bool): Whether or not the cut from q1 to q2 is to be extended throughout the data (default true)

Returns:

- Data list (n \* 4 arrays): n instances of [Intensity, monitor count, normalization and normalization counts].
- Bin list (n \* 3 arrays): n instances of bin edge positions in plane of size (m+1,3), orthogonal positions of bin edges in plane of size (2,2), and energy edges of size (2).
- center position (n \* 3D arrays): n instances of center positions for the bins.
- binDistance (n arrays): n instances of arrays holding the distance in q to q1.

`DataSet.load(filename)`

Function to load an object from a pickled file.

---

**Note:** It is not possible to unpickle an object created in python 3 in python 2 or vice versa.

---

`DataSet.plotA3A4` (*files*, *ax=None*, *planes=[]*, *binningDecimals=3*, *log=False*, *returnPatches=False*, *singleFigure=False*, *plotTessellation=False*, *Ei\_err=0.05*, *temperature\_err=0.2*, *magneticField\_err=0.2*, *electricField\_err=0.2*)

Plot data files together with pixels created around each point in A3-A4 space. Data is binned in the specified planes through their A3 and A4 values. This can result in distorted binning when binning across large energy regions. Data is plotted using the pixels calculated for average plane value, i.e. binning 7,8,9,10, and 11 patches for plane 9 are used for plotting.

Args:

- `files` (DataFiles): single file or list of files to be binned together

Kwargs:

- `ax` (matplotlib axis): Axis into which the planes are to be plotted (Default None, i.e. new)
- `planes` (list (of lists)): Planes to be plotted and binned (default [])
- `binningDecimals` (int): Number of decimal places A3-A4 positions are rounded before binning (default 3)
- `log` (bool): Whether or not to plot intensities as logarithm (default False)
- `returnPatches` (bool): If true the method returns the patches otherwise plotted in the given axis (default False)
- `singleFigure` (bool): If true, all planes are plotted in same figure (default False)
- `plotTessellation` (bool): Plot Tessellation of points (default False)
- `Ei_err` (float): Tolerance of  $E_i$  for which the values are equal (default = 0.05)
- `temperature_err` (float): Tolerance of temperature for which the values are equal (default = 0.2)
- `magneticField_err` (float): Tolerance of magnetic field for which the values are equal (default = 0.2)
- `electricField_err` (float): Tolerance of electric field for which the values are equal (default = 0.2)

Returns:

- `ax` (matplotlib axis or list of): axis (list of) containing figures for plotted planes.

Raises:

- `AttributeError`

Examples:

The following example will combine the two files and plot all of the available planes in different figures.

```
>>> DS = DataSet.DataSet(convertedFiles=[--.nxs, ---.nxs])
>>> plt.figure()
>>> ax = plt.gca()
>>>
>>> DataSet.plotA3A4(DS.convertedFiles, ax=ax)
```

If only a subset of planes or different planes are to be combined the following will achieve this:

```
>>> DataSet.plotA3A4(DS.convertedFiles, ax=ax, planes=[0, 1, 2, 3, [4, 5, 6], [8, 9]])
```

Here planes 0 through 3 are plotted separately while 4,5, and 6 as well as 8 and 9 are binned.

---

**Note:** Binning planes from different analysers might result in nonsensible binnings.

---

`DataSet.plotCut1D` (*positions, I, Norm, Monitor, q1, q2, width, minPixel, Emin, Emax, rlu=True, ax=None, plotCoverage=False, extend=True, \*\*kwargs*)

Plotting wrapper for the cut1D method. Generates a 1D plot with bins at positions corresponding to the distance from the start point. Adds the 3D position on the x axis with ticks.

---

**Note:** Can only perform cuts for a constant energy plane of definable width.

---

Args:

- `positions` (3 arrays): position in Qx, Qy, and E in flattened arrays.
- `I` (array): Flatten intensity array
- `Norm` (array): Flatten normalization array
- `Monitor` (array): Flatten monitor array
- `q1` (2D vector): Starting position in (qx,qy)
- `q2` (2D vector): Ending position in (qx,qy)
- `width` (float): Width of binning orthogonal to cut direction
- `minPixel` (float): Width of binning along cut direction
- `Emin` (float): Minimal energy of cut
- `Emax` (float): Maximal energy of cut

Kwargs:

- `rlu` (bool): If True, extract points are plotted in RLU # TODO: Make this work
- `ax` (matplotlib axis): Figure axis into which the plots should be done (default None). If not provided, a new figure will be generated.
- `kwargs`: All other keywords will be passed on to the `ax.errorbar` method.

Returns:

- `ax` (matplotlib axis): Matplotlib axis into which the plot was put.
- `Data list` (4 arrays): Intensity, monitor count, normalization and normalization counts binned in the 1D cut.
- `Bin list` (3 arrays): Bin edge positions in plane of size (n+1,3), orthogonal positions of bin edges in plane of size (2,2), and energy edges of size (2).
- `binCenter` (3D array): Array containing the position of the bin centers of size (n,3)
- `binDistance` (array): Distance from centre of bins to start position.

`DataSet.plotCutPowder` (*positions, I, Norm, Monitor, EBinEdges, qMinBin=0.01, ax=None, \*\*kwargs*)

Plotting wrapper for the cutPowder method. Generates a 2D plot of powder map with intensity as function of the length of q and energy.

---

**Note:** Can only perform cuts for a constant energy plane of definable width.

---

Args:

- `positions` (3 arrays): position in Qx, Qy, and E in flattened arrays.
- `I` (array): Flatten intensity array
- `Norm` (array): Flatten normalization array

- Monitor (array): Flatten monitor array
- EBinEdges (list): Bin edges between which the cuts are performed.

Kwargs:

- qMinBin (float): Minimal size of binning along q (default 0.01). Points will be binned if they are closer than qMinBin.
- ax (matplotlib axis): Figure axis into which the plots should be done (default None). If not provided, a new figure will be generated.
- kwargs: All other keywords will be passed on to the ax.pcolormesh method.

Returns:

- ax (matplotlib axis): Matplotlib axis into which the plot was put.
- Data list (4 arrays): Intensity, monitor count, normalization and normalization counts binned in the 1D cut.
- Bin list (3 arrays): Bin edge positions in plane of size (n+1,3), orthogonal positions of bin edges in plane of size (2,2), and energy edges of size (2).

`DataSet.plotCutQE` (*positions, I, Norm, Monitor, q1, q2, width, minPix, EnergyBins, rlu=True, ax=None, \*\*kwargs*)

Plotting wrapper for the cutQE method. Generates a 2D intensity map with the data cut by cutQE.

---

**Note:** Positions shown in tool tip reflect the closes bin center and are thus limited to the area where data is present.

---

Args:

- positions (3 arrays): position in Qx, Qy, and E in flattend arrays.
- I (array): Flatten intensity array
- Norm (array): Flatten normalization array
- Monitor (array): Flatten monitor array
- q1 (2D array): Start position of cut in format (qx,qy).
- q2 (2D array): End position of cut in format (qx,qy).
- width (float): Full width of cut in q-plane.
- minPixel (float): Minimal size of binning aling the cutting direction. Points will be binned if they are closer than minPixel.
- EnergyBins (list): Bin edges between which the 1D constant energy cuts are performed.

Kwargs:

- ax (matplotlib axis): Figure axis into which the plots should be done (default None). If not provided, a new figure will be generated.
- rlu (bool): If True, data is plotted in RLU. # TODO: Make this work!
- kwargs: All other keywords will be passed on to the ax.errorbar method.

Returns:

- ax (matplotlib axis): Matplotlib axis into which the plot was put.
- Data list (n \* 4 arrays): n instances of [Intensity, monitor count, normalization and normalization counts].



- Bin list ( $n * 3$  arrays):  $n$  instances of bin edge positions in plane of size  $(m+1,3)$ , orthogonal positions of bin edges in plane of size  $(2,2)$ , and energy edges of size  $(2)$ .
- center position ( $n * 3D$  arrays):  $n$  instances of center positions for the bins.
- binDistance ( $n$  arrays):  $n$  instances of arrays holding the distance in  $q$  to  $q1$ .

`DataSet.plotQPlane` (*I, Monitor, Norm, pos, EMin, EMax, binning='xy', xBinTolerance=0.05, yBinTolerance=0.05, enlargen=False, log=False, ax=None, \*\*kwargs*)

Plotting tool to show binned intensities in the Q plane between provided energies.

Args:

- I (array): Intensity of data.
- Monitor (array): Monitor of data.
- Norm (array): Normalization of data.
- pos (3 array): Position of data in  $qx$ ,  $qy$ , and energy.
- EMin (float): Lower energy limit.
- EMax (float): Upper energy limit.

Kwargs:

- binning (str): Binning scheme, either 'xy' or 'polar' (default 'xy').
- xBinTolerance (float): bin sizes along x direction (default 0.05). If enlargen is true, this is the minimum bin size.
- yBinTolerance (float): bin sizes along y direction (default 0.05). If enlargen is true, this is the minimum bin size.
- enlargen (bool): If the bin sizes should be adaptive (default False). If set true, bin tolerances are used as minimum bin sizes.
- log (bool): Plot intensities as the logarithm (default False).
- ax (matplotlib axes): Axes in which the data is plotted (default None). If None, the function creates a new axes object.
- other: Other key word arguments are passed to the pcolormesh plotting algorithm.

Returns:

- ax (matplotlib axes)

---

**Note:** The axes object gets a new method denoted 'set\_clim' taking two parameters (VMin and VMax) used to change axes coloring.

---

`DataSet.voronoiTessellation` (*points, plot=False, Boundary=False, numGroups=False*)

Generate individual pixels around the given datapoints.

Args:

- points (list of list of points): Data points to generate pixels in shape  $[files, XY, N]$  i.e.  $[1,2,N]$  for one file with  $N$  points

Kwargs:

- plot (bool): If True, method plots pixels created with green as edge bins and red as internal (default False)
- Boundary (list of Polygons): List of Shapely polygons constituting the boundaries (Default False)

## DataFile Object and Methods

**class** `DataFile.DataFile` (*fileLocation*)

Object to load and keep track of HdF files and their conversions

**calculateEdgePolygons** (*addEdge=True*)

Method to calculate bounding polygon for all energies. The energies are split using the bin-edges method of DataSet. Hereafter, the outer most points are found in polar coordinates and a possible addition is made creating the padded bounding polygon.

Kwargs:

- `addEdge` (bool/float): If true, padding is found as difference between outer and next outer point. If `addEdge` is a number, generate padding a padding of this value (default True)

Returns:

- `edgePolygon` (list): List of shapely polygons of the boundary
- `EBins` (list): Binning edges in energy

**difference** (*other, keys={'Ei', 'I', '\_A3', '\_A4', 'binning', 'instrument', 'sample', 'scanParameters'}*)

Return the difference between two data files by keys

**loadBinning** (*binning*)

Small function to check if current binning is equal to wanted binning and if not reloads to binning wanted

**plotA4** (*binning=None*)

Method to plot the fitted A4 values of the normalization table

Kwargs:

- `binning` (int): Binning for the corresponding normalization table (default self.binning or 8)

returns:

- `fig` (matplotlib figure): Figure into which the A4 values are plotted

**plotEf** (*binning=None*)

Method to plot the fitted Ef values of the normalization table

Kwargs:

- `binning` (int): Binning for the corresponding normalization table (default self.binning or 8)

returns:

- `fig` (matplotlib figure): Figure into which the Ef values are plotted

**plotEfOverview** (*binning=None*)

Method to plot the fitted Ef values of the normalization table

Kwargs:

- `binning` (int): Binning for the corresponding normalization table (default self.binning or 8)

returns:

- `fig` (matplotlib figure): Figure into which the Ef values are plotted

**plotNormalization** (*binning=None*)

Method to plot the fitted integrated intensities of the normalization table

Kwargs:

- `binning` (int): Binning for the corresponding normalization table (default self.binning or 8)

returns:

- fig (matplotlib figure): Figure into which the Ef values are plotted

**saveNXsqom** (*saveFileName*)

Save converted file into an NXsqom.

Args:

- saveFileName (string): File name to be saved into.

### 3.3.2 Graphical User Interfaces

Below follows an overview of the developed graphical interfaces to be used in the data treatment.

#### 3.3.3 Viewer3D

Quick visualization tool designed to deal with the difficulties of handling 3D data. Through simple keyboard inputs one can look through the data along the principal axis.

**class** Viewer3D.**Viewer3D** (*Data, bins, axis=2, log=False, ax=None, grid=False, \*\*kwargs*)

3 dimensional viewing object generating interactive Matplotlib figure. Keeps track of all the different plotting functions and variables in order to allow the user to change between different slicing modes and to scroll through the data in an interactive way.

Args:

- Data (3D array): Intensity array in three dimensions. Assumed to have Qx, Qy, and E along the first, second, and third directions respectively.
- bins (List of 1D arrays): Coordinates of the three directions as returned by the BinData3D functionality of DataSet.

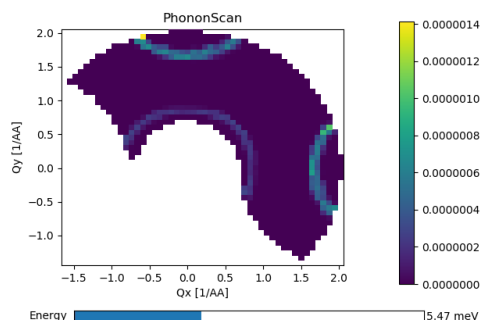
Kwargs:

- axis (int): Axis along which the interactive plot slices the data (default 2).
- log (bool): If true, the log 10 of the intensity is plotted (default False)
- ax (matplotlib axis): Matplotlib axis into which one plots data (Default None)

Example:

```
>>> from MJOLNIR.Data import DataSet, Viewer3D
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>>
>>> DataFile = ['./TestData/comeasim2018n000011.nxs']
>>> Data, bins = dataset.binData3D(0.08, 0.08, 0.25)
>>>
>>> Intensity = np.divide(Data[0]*Data[3], Data[1]*Data[2])
>>>
>>> Viewer = Viewer3D.Viewer3D(Intensity, bins, axis=2)
>>> Viewer.ax.set_title(str(title) [2:-1])
>>> plt.show()
```

Interactive plot generated by above function call with a Intensity being 3D rebinned data using the simple phonon component, Ei of 10 meV and 180 steps of 1 degree in A3, A4 at -60 degrees.



### 3.3.4 Viewer1D

Visualization tool for 1D extractions of data. This tool is intended to be used during the experiment but especially in the initial phase where optimal settings for goniometers, A3, crystal parameters, e.t.c. are to be found.

**class** `Viewer1D.Viewer1D` (*XData*, *YData*, *YErr*, *fitFunction*=<*MJOLNIR.Statistics.FittingFunction.Gaussian* object>, *xLabel*=", *dataLabel*=", *xID*=0, *plotAll*=False, **\*\*kwargs**)

Interactive visualization of 1D data with fitting capabilities. Currently only intended for 1 scan file.

Args:

- *XData* (list): List of x-values in shape (m,n) for m data series and n scan points.
- *YData* (list): List of y-values in shape (m,n) for m data series and n scan points.
- *YErr* (list): List of y errors in same shape as *YData*.

Kwargs:

- *fitFunction* (*FittingFunction*): Customized object to perform fitting (default Gaussian).
- *xLabel* (list): X label text in shape (m) for m scan parameters (default "", nothing plotted).
- *dataLabel* (list): Label to be shown in legend in shape (m) or (m,l), m and l free (default "", nothing plotted)
- *xID* (int): Index of x axis to be used (default 0)
- *yID* (int): Index of y axis to be plotted first (default 0)
- *plotAll* (bool): Boolean deciding whether or not to plot all data (default False)

Raises:

- `AttributeError`

Example: # TODO: REDO!!

```
>>> from MJOLNIR.Data import DataSet, Viewer1D
>>> file = 'TestData/ManuallyChangedData/A3.h5'
>>> ds = DataSet.DataSet(dataFiles = file)
>>> ds.convertDataFile(binning=1)
>>> data = ds.extractData(A4Id=30)
>>>
>>> Y = data[:, :5] # Only first 5 energies
>>> Y_err = np.sqrt(Y) # Calculate errors
>>> X = np.arange(Y.shape[0])
>>>
>>> xlabel = ['Arbitrary [arb]']
```

(continues on next page)

(continued from previous page)

```
>>> dataLabel = np.array(['Detector 0: pos 0', 'Detector 0: pos 1', 'Detector 0:
↳pos 2', 'Detector 0: pos 3', 'Detector 0: pos 4'])
>>>
>>> # Initialize the viewer
>>> Viewer = Viewer1D.Viewer1D(XData=X, YData=Y, >>> YErr=Y_err,
↳xLabel=xlabel, dataLabel = dataLabel, plotAll=True)
```

For a walkthrough of the interface see *Raw plotting and fitting*.

**initData()**

Update with new indices both X and Y (+Yerr)

**plotData()**

Plot current data. First destroy previous plot if possible

**plotFit()**

Plot current guess or fit

**removeFitPlot()**

Try to remove previous fitPlot if it exists

## 3.4 Tools functions

Below is a list of the general tools developed for the MJOLNIR software package.

<i>KwargChecker</i>	Function to check if given key-word is in the list of accepted Kwargs.
<i>my_timer_N</i>	Timer function to measure time consumption of function.
<i>binEdges</i>	Generate binning of values array with minimum bin size of tolerance.

**\_tools.KwargChecker** (*function=None, include=None*)

Function to check if given key-word is in the list of accepted Kwargs. If not directly therein, checks capitalization. If still not match raises error with suggestion of closest argument.

Args:

- func (function): Function to be decorated.

Raises:

- AttributeError

**\_tools.beautifyArgs** (*args=(), kwargs={}*)

Beautify arguments and keyword arguments. Returns formatted string with arguments and keyword arguments separated with commas as called in a function

**\_tools.binEdges** (*values, tolerance*)

Generate binning of values array with minimum bin size of tolerance. Binning starts at values[0]-tolerance/2.0 and ends at values[-1]+tolerance/2.0.

Args:

- values (array): 1D array to be binned.
- tolerance (float): Minimum length of bin sizes.

Returns:

- bins (array)

`_tools.fileListGenerator` (*numberString*, *folder*, *year*, *format*='{:}camea{:d}n{:06d}.hdf')

Function to generate list of data files.

Args:

- numberString (str): List if numbers seperated with comma and dashes for sequences.
- folder (str): Folder of wanted data files.
- year (int): Year if wanted data files

Kwargs:

- format (str): format of data files (default '{:}camea{:d}n{:06d}.hdf')

returns:

- list of strings: List containing the full file string for each number provided.

**Example:**

```
>>> numberString = '201-205,207-208,210,212'
>>> files = fileListGenerator(numberString,'data/',2018)
['data/comea2018n000201.hdf', 'data/comea2018n000202.hdf',
'data/comea2018n000203.hdf', 'data/comea2018n000204.hdf',
'data/comea2018n000205.hdf', 'data/comea2018n000207.hdf',
'data/comea2018n000208.hdf', 'data/comea2018n000210.hdf',
'data/comea2018n000212.hdf']
```

`_tools.my_timer_N` (*N=0*)

Timer function to measure time consumption of function.

Kwargs:

- N (int): Number of iterations to perform.

Raises:

- AttributeError

---

## In depth description of core functionalities

---

Below is a list of links to the documentation for individual core functionalities. These are not code dependent and should be understandable from a physics/mathematics perspective.

### 4.1 Geometry

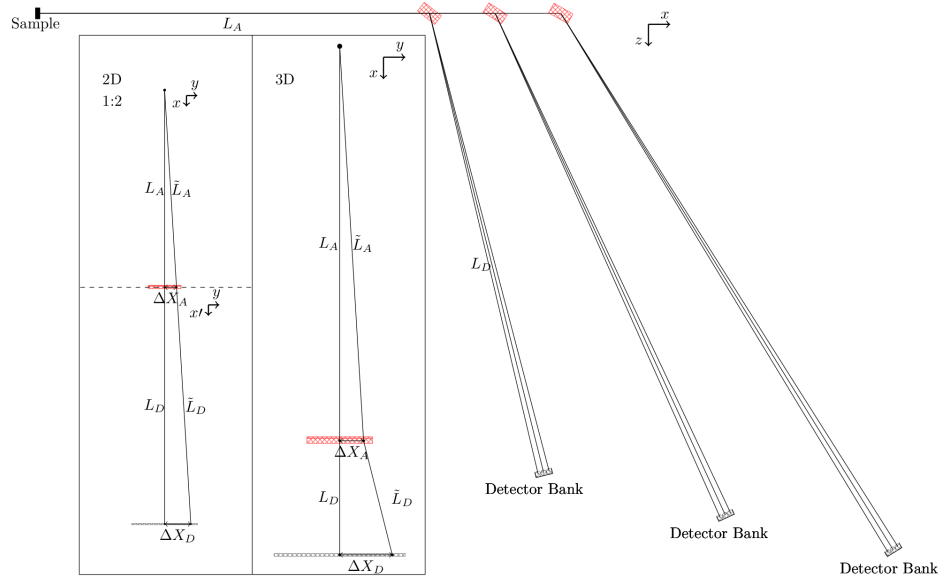
**Warning:****Assumptions made by the program:**

- Incoming beam is along  $(0, 1, 0)$ , i.e. the y-axis
- Sample is located at  $(0, 0, 0)$
- Only first order scattering
- Energies and A4 is calculated independently of the rotation of the analyser, i.e. infinite mosaicity is assumed
- Analysers are flat, tangential, and without focusing in the tangential plane.
- Detectors are horizontal
- All analysers in a wedge are made from the same material.

The calculation of energies and the corresponding scattering angles is automatically calculated when the initialization method is called on the *Instrument* object. That is, one first of all needs to create the wanted instrument with all of the detectors, analysers and wedges as described in the instrument creation tutorial *Build a simple instrument*. Having done all this, the procedure for the calculations is as follows.

Looping through all wedges, the concept attribute is checked. This tells whether one has a one to one correspondence between the detectors and the analysers or a many to many. The prior is true for secondary spectrometers as FlatCone or MultiFLEXX, where each detector optimally receives neutrons from only one analyser. The latter is, however only in the sense of many to one, true for the CAMEA concept applied at SINQ. Here one detector receives neutrons from

multiple analysers. The need for this distinction in the code is for the algorithm to know, if it should split the detector pixels up in bunches or not. These bunches are controlled by the attributes for the *Detector* object. In both cases, the relative position vector is found for each pixel and the corresponding analyser as well as the vector from origo to the analyser. As described below, the actual optimal scattering position is then calculated and returned to the instrument object. Having both the detector position as well as the scattering position at the analyser, it is straight forward to calculate scattering angle  $A4$  and corresponding final energy  $E_f$ .



Visualization of scattering planes used for the calculation of scattering angle and final energy.

The math behind finding the optimal scattering position for a given pixel at the analyser is as follows. Noticing that the neutron cannot have a change in its momentum perpendicular to the scattering plane, one can make a 2D drawing of the trajectory of the neutron from the sample to the detector as in the left of figure *InstrumentFig*. Here it is important to note that the dashed line signifies a bend of the trajectory as depicted in the 3D subfigure in the inset to the right. The discrepancy between the two is that in the latter a projection from 3D to 2D is used; the neutron is scattered out of the plane. Accepting the 2D depiction, one can notice, that the two triangles Sample-AnalyserCenter-DeltaXA and Sample-DetectorCenter-DeltaXD have the same angles. Thus

$$\frac{\Delta X_D}{L_A + L_D} = \frac{\Delta X_A}{L_A} \Rightarrow \Delta X_A = \frac{\Delta X_D}{\frac{L_D}{L_A} + 1}.$$

This calculation is slightly more complex if one does not assume that both the analyser and the detector are tangential to the sample-detector vector. One then needs to find the distance away from the scattering direction, the pixel is moved. Before, this was just given by the pixel position relative to its centre,  $\Delta X_D$ , but is now given as a dot product between the relative position and the vector perpendicular to the scattering direction:

$$\Delta X_D = (\vec{P}_{\text{pos}} - \vec{P}_{\text{det,centre}}) \cdot \vec{L}_{\perp},$$

where  $\vec{P}_{\text{pos}}$  is the pixel position,  $\vec{P}_{\text{det,centre}}$  is the center position of the pixel and  $\vec{L}_{\text{perp}}$  is the vector perpendicular to the scattering direction and is in the horizontal plane. With this correction, the above formula for position on the analyser still holds true, and one can thus find the scattering position. By simply using the cosine relation, where the angle  $\theta_{\text{vec}\{a\}}$  and  $\vec{b}$  is given by

$$\cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}| |\vec{b}|},$$

one can find the angle between the incoming beam and the scattering direction, denoted  $A4$ . Further, the final energy  $E_f$  is found in a similar sense, where the angle between sample-analyser and analyser-detector is found and converted



into an energy by the usual elastic scattering formula

$$\lambda = 2d \sin(\theta) \quad \text{and} \quad E = \left( \frac{9.0445678 \text{A} \sqrt{\text{meV}}}{\lambda} \right)^2.$$

Here the algorithm uses the d-value specified for the first analyser in the wedge. This could of course be generalized to allow for different analyser materials, but is not yet done as this would complicate the code further and is not believed to be relevant.

## 4.2 Energy normalization procedure

The following is a walk-through of the energy normalization method as performed by the MJOLNIR Data module when given a data file containing scattering data from scanning incoming energy,  $E_i$ , with a Vanadium sample.

The raw data file is opened and the intensities of all pixels and all detectors are extracted. Further, the number of pixels, detectors and wedges are found.

### 4.2.1 Determination of active area

In order to determine the active area of each detector, and indeed of each detector segment looking at a given analyzer, the intensity data as function of pixel, detector, and energy is collapsed along the energy direction. This results in graphs like in figure [EnergySummed](#) below. From this, it is clear that not all pixels are active and that the splitting into software pixels depend on the detector/analyser combination.

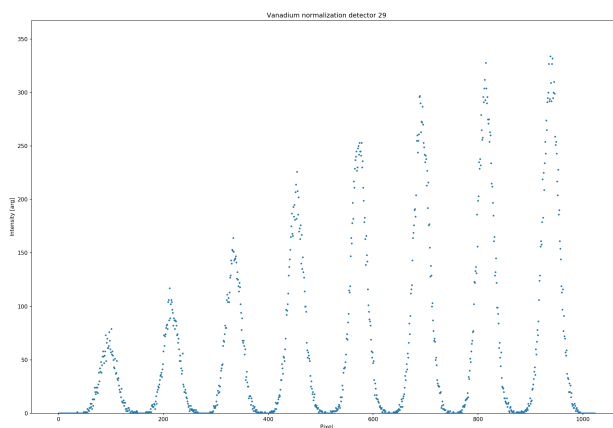


Fig. 1: Intensity of detector tube 29 when summing across energies for Vanadium sample.

By fitting a suitable amount of Gaussians to all of the peaks, one obtains the intensity center for each energy on the detectors and from this, one can determine the active area, as seen in figure [GaussFitWedge](#). The procedure to find the peaks and fit the Gaussian functions is to first find the maximal intensity, estimate the width from prior knowledge of the setup, fit the Gaussian and then subtract the fit. This is then repeated for the necessary number of times needed. This method does, however, depend on the signal being described by a Gaussian to an extent that the data with the fit subtracted has a small remainder. If the difference is too big, the algorithm cannot find all the peaks and an error is raised. Currently the active area is defined as pixels within the center  $\pm 3\sigma$ . This makes it possible to use around 99.74% of the intensity. However, making this area too broad allows pixels with very low statistics to be used in experiments introducing a high uncertainty on the measured intensity.

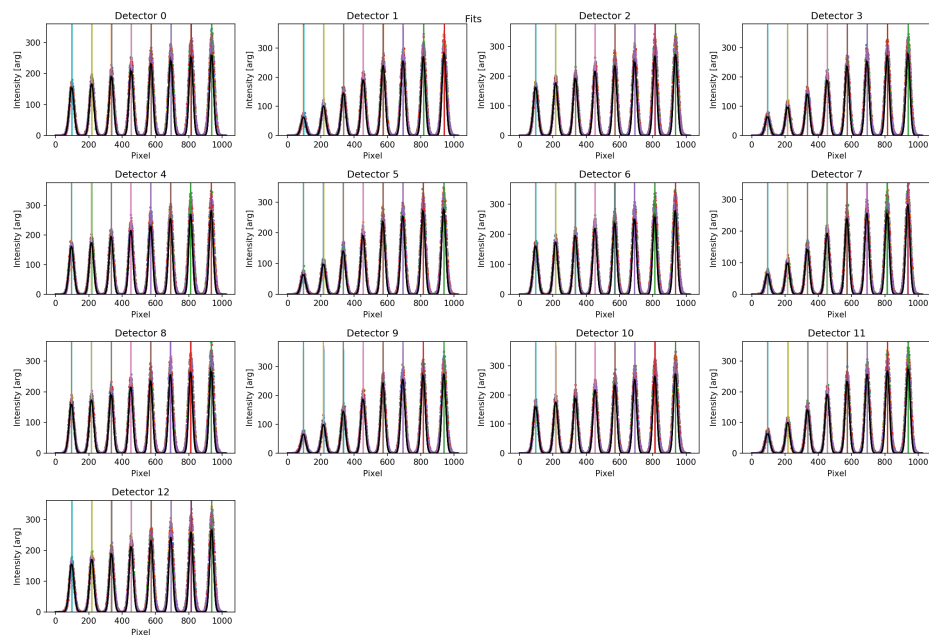


Fig. 2: Fit of all peaks for wedge 4 allowing determination of center and width.

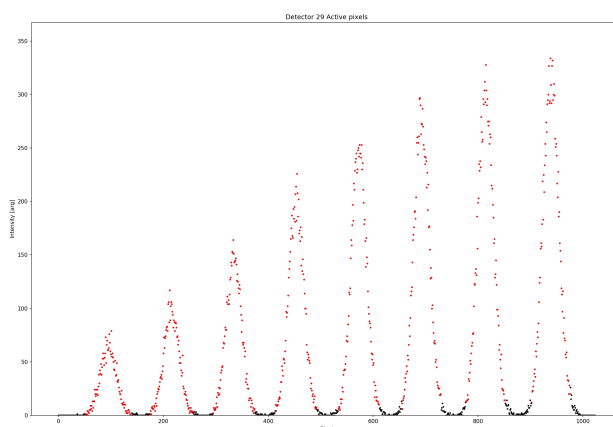
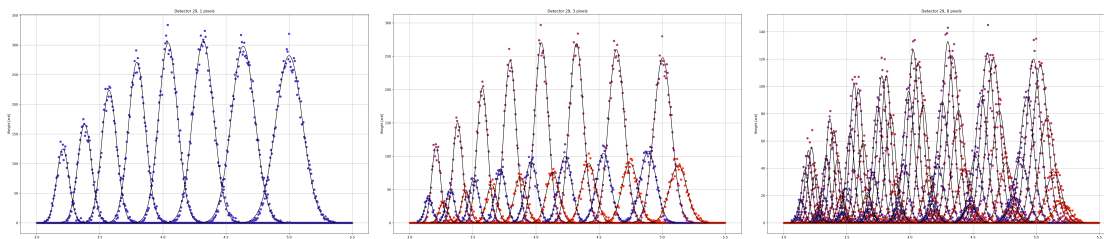


Fig. 3: Intensity of detector tube 29 with active area shown in red.

For the current width used for active area, the red points in the above figure [ActiveArea](#) is used.

## 4.2.2 Software pixel binning

With the knowledge of the positions and widths of the active areas on the detectors, one needs to define the pixel edges for all of the software pixels. The number of pixels in each software pixel depends on both the width of the active area on the detector and the number of software pixels into which the user wants to bin. Usually, the number of software pixels is between 1 and 8, where a case of 8 pixels is shown in figure [SoftwarePixels](#) below. Then, using the raw intensity signal is binned into software pixels as function of energy. These are then individually fitted with a Gaussian as to precisely determine the center energy, normalization, width, and possible background.



Fit of one, three, and eight software pixels to Vanadium normalization for the 29th detector tube.

It merely remains to save the obtained normalization into a file, which is done in the CSV format. For each detector, analyser, and software pixel the following parameters are saved:

```
Normalization for 8 pixel(s) using data TestData/VanNormalization.h5
Performed 2018-04-05 13:22:29.008053
Detector,Energy,Pixel,Amplitude,Center,Width,Background,lowerBin,upperBin
0,0,0,553.307499792,3.11194470068,0.0351916686546,-1.14865525492,25,30
0,0,1,3534.65749131,3.13586570375,0.0234845709327,2.79927766486,30,35
0,0,2,6707.93446507,3.17045382964,0.0296278355214,-2.44445514979,35,40
0,0,3,8449.34235339,3.19740050283,0.0279281924646,0.147005539459,40,44
0,0,4,7762.45025046,3.22095475304,0.029643268258,-3.43488524431,44,48
0,0,5,5700.97166402,3.25044106789,0.0305651479509,-0.633300325994,48,53
0,0,6,2117.92181626,3.28390443811,0.0270144206303,1.62528891194,53,58
0,0,7,269.377490747,3.31657107929,0.0341873820177,-0.0625227707394,58,63
...
```

The CSV file is saved and is used when converting experiment data from raw HDF files into NXqom files explained in the Data file conversion documentation. For a table of the found energies, see [291018](#)

## 4.3 Data file conversion

In general, it is expected that a CAMEA-like instrument is to be run during experiments in 2 different scan modes:

- A3 scans
- External parameter

However, in the initial phase of setup other scans might be conducted. The data conversion thus does not require a specific scan but allows for all types. This does then require the user to choose corresponding visualizations correspondingly.

### 4.3.1 HdF file format

The raw data from the instrument is expected to be provided in an HdF 5 format with the following structure:

```
cameasim2018n0000xx.h5
├── entry
│   └── CAMEA
│       ├── calib1
│       │   ├── a4offset
│       │   ├── amplitude
│       │   ├── background
│       │   ├── boundaries
│       │   ├── final_energy
│       │   └── width
│       ├── calib3
│       │   ├── a4offset
│       │   ├── amplitude
│       │   ├── background
│       │   ├── boundaries
│       │   ├── final_energy
│       │   └── width
│       ├── calib8
│       │   ├── a4offset
│       │   ├── amplitude
│       │   ├── background
│       │   ├── boundaries
│       │   ├── final_energy
│       │   └── width
│       ├── detector
│       │   ├── counts
│       │   └── summed_counts
│       ├── monochromator
│       │   ├── d_spacing
│       │   ├── energy
│       │   ├── gm
│       │   ├── gm_zero
│       │   ├── horizontal_curvature
│       │   ├── horizontal_curvature_zero
│       │   ├── rotation_angle
│       │   ├── rotation_angle_zero
│       │   ├── summed_counts
│       │   ├── tlm
│       │   ├── tlm_zero
│       │   ├── tum
│       │   ├── tum_zero
│       │   ├── type
│       │   ├── vertical_curvature
│       │   └── vertical_curvature_zero
│       └── monochromator_slit
│           ├── bottom
│           ├── bottom_zero
│           ├── left
│           ├── left_zero
│           ├── right
│           ├── right_zero
│           ├── top
│           ├── top_zero
│           └── x_gab
```

(continues on next page)

(continued from previous page)

```

└─ y_gab
— comment
— control
  └─ absolute_time
  └─ data
  └─ mode
  └─ preset
  └─ time
— data
  └─ counts
  └─ summed_counts
  └─ (Scan parameter)
— end_time
— experimental_identifier
— instrument
— local_contact
  └─ name
— proposal_id
— proposal_title
— proposal_user
  └─ name
— proton_beam
  └─ data
— sample
  └─ azimuthal_angle
  └─ name
  └─ orientation_matrix
  └─ plane_normal
  └─ plane_vector_1
  └─ plane_vector_2
  └─ polar_angle
  └─ polar_angle_zero
  └─ rotation_angle
  └─ rotation_angle_zero
  └─ sgl
  └─ sgl_zero
  └─ sgu
  └─ sgu_zero
  └─ (sample environment parameters)
  └─ unit_cell
— scancommand
— scanvars
— start_time
— title
— user
  └─ address
  └─ affiliation
  └─ email
  └─ name

```

From this file, raw plotting and a conversion algorithm is possible. Raw plotting is further explained in [Raw plotting and fitting](#).

### 4.3.2 NXsqom file format

The format into which data is converted is the **NXsqom** format. It is a standard of the nexus files and is designed for data converted into reciprocal space. With this choice of conversion it is believed that some pre-existing data handling routines exist in other software solutions already.

Below is a HDF converted file in the NXsqom format for a *A3* scan. Here *NP* is the number of scan points and *NNP* is the number of unique pixels converted.

```
cameasim2018n0000xx.nxs
├── entry
│   ├── CAMEA
│   │   ├── calib1
│   │   │   ├── a4offset
│   │   │   ├── amplitude
│   │   │   ├── background
│   │   │   ├── boundaries
│   │   │   ├── final_energy
│   │   │   └── width
│   │   ├── calib3
│   │   │   ├── a4offset
│   │   │   ├── amplitude
│   │   │   ├── background
│   │   │   ├── boundaries
│   │   │   ├── final_energy
│   │   │   └── width
│   │   ├── calib8
│   │   │   ├── a4offset
│   │   │   ├── amplitude
│   │   │   ├── background
│   │   │   ├── boundaries
│   │   │   ├── final_energy
│   │   │   └── width
│   │   ├── detector
│   │   │   ├── counts
│   │   │   └── summed_counts
│   │   ├── monochromator
│   │   │   ├── d_spacing
│   │   │   ├── energy
│   │   │   ├── gm
│   │   │   ├── gm_zero
│   │   │   ├── horizontal_curvature
│   │   │   ├── horizontal_curvature_zero
│   │   │   ├── rotation_angle
│   │   │   ├── rotation_angle_zero
│   │   │   ├── summed_counts
│   │   │   ├── tlm
│   │   │   ├── tlm_zero
│   │   │   ├── tum
│   │   │   ├── tum_zero
│   │   │   ├── type
│   │   │   ├── vertical_curvature
│   │   │   └── vertical_curvature_zero
│   │   └── monochromator_slit
│   │       ├── bottom
│   │       ├── bottom_zero
│   │       ├── left
│   │       └── left_zero
```

(continues on next page)

(continued from previous page)

```

      |
      |— right
      |— right_zero
      |— top
      |— top_zero
      |— x_gab
      |— y_gab
— comment
— control
  |— absolute_time
  |— data
  |— mode
  |— preset
  |— time
— data
  |— counts
  |— en
  |— monitor
  |— normalization
  |— qx
  |— qy
  |— summed_counts
  |— (Scan parameter)
— end_time
— experimental_identifier
— instrument
— local_contact
  |— name
— proposal_id
— proposal_title
— proposal_user
  |— name
— proton_beam
  |— data
— sample
  |— azimuthal_angle
  |— name
  |— orientation_matrix
  |— plane_normal
  |— plane_vector_1
  |— plane_vector_2
  |— polar_angle
  |— polar_angle_zero
  |— rotation_angle
  |— rotation_angle_zero
  |— sgl
  |— sgl_zero
  |— sgu
  |— sgu_zero
  |— (sample environment parameters)
  |— unit_cell
— scancommand
— scanvars
— start_time
— title
— user
  |— address
  |— affiliation

```

(continues on next page)

<div style="display: flex; align-items: center;"> <div style="margin-right: 10px;"> <div style="border-bottom: 1px solid black; width: 20px; height: 10px; margin-bottom: 2px;"></div> <div style="border-bottom: 1px solid black; width: 20px; height: 10px;"></div> </div> <div> <div>email</div> <div>name</div> </div> </div>
---

## 4.4 Voronoi tessellation and plotting functionality

With all of the effort put into building an instrument acquiring as much data and data points as possible, it is sensible to have a plotting algorithm that then shows all of these. This is exactly what the two methods `plotA3A4` and `plotQPatches` seek to do. However, performing calculations and plotting all of the measured points make the methods computationally heavy and slow as well as presents challenges for the visualization. Below is a list of difficulties encountered while building the two methods.

Difficulties:

- Handle (almost) duplicate point positions
- Generate suitable patch around all points
- Number of points to handle

The methods do address some of the above challenges in some way; the almost duplicate points are handled by truncating the precision on the floating point values holding the position. That is,  $\vec{q} = (0.1423, 2.1132)$  is by default rounded to  $\vec{q} = (0.142, 2.113)$  and binned with other points at the same position. This rounding is highly relevant when generating patches in *A3-A4* coordinates as the discretization is intrinsic to the measurement scans performed.

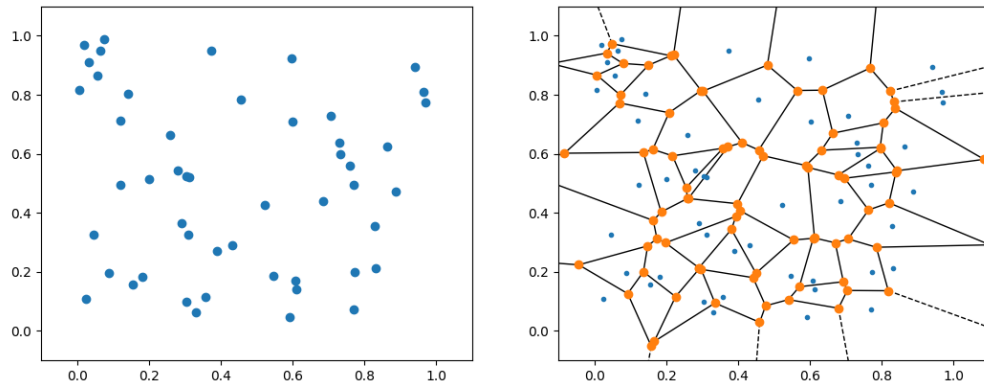
What is of real interest is the generation of a suitable patch work around all of the points for which this page is dedicated. The wanted method is to be agile and robust to be able to handle all of the different scenarios encountered. For these requirements to be met, the voronoi tessellation has been chosen.

### 4.4.1 Voronoi tessellation

First of all, the voronoi diagram is defined as the splitting of a given space into regions, where all points in one region is closer to one point than to any other point. That is, given an initial list of points, the voronoi algorithm splits of space into just as many regions for which all points in a given region is closest to the initial point inside it than to any other. This method is suitable in many different areas of data treatment, e.g. to divide a city map in to districts dependent on which hospital is nearest, or divide ????. This method can however also be used in the specific task for creating pixels around each measurement point in a neutron scattering dataset.

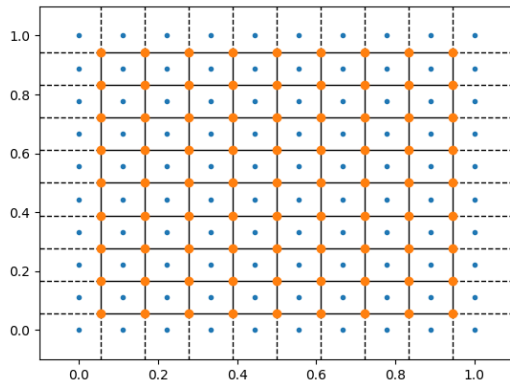
The method works in *n*-dimensional spaces, where hyper-volumes are created, and one can also change the distance metric from the normal Euclidean  $d = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2 \dots}$  to other metrics, i.e. the so-called Manhattan distance  $d = |\Delta x| + |\Delta y| + |\Delta z| + \dots$ . It has, however, been chosen that using multi-dimensional and non-Euclidean tessellations obscures the visualization of the data rather than enhance it. Furthermore, the used SciPi-package [spatial](#) does not natively support changes of metric and a rewriting is far outside of the scope of this software suite.





**Left:** 50 random points generated and plotted in 2D. **Right:** Voronoi diagram created for the 50 random points. Blue points are initial positions, orange are intersections, full lines are edges (denoted ridges) connecting two intersections, dashed lines go to infinity.

As seen above, for a random generated set of points, the voronoi tessellation is also going to produce a somewhat random set of edges. This is of course different, if instead one had a structured set of points as in [StructuredVoronoi](#) below. However, some of the edges still go to infinity creating infinitely large pixels for all of the outer measurements. This is trivially un-physical and is to be dealt with by cutting or in another way limiting the outer pixels.



Voronoi generated for regular set of data points as for instance an  $A3$  rotation scan with equidistant  $A4$  points.

From the above, it is even more clear that the edge pixels extend to infinity. This is to be taken care of and two ways comes into mind. First, one could define a boundary such that the pixel edges intersecting this boundary is cut in a suitable manor. Second, one could add an extra set of data points around the actual measurement points in such a way that all of the wanted pixels remain finite. Both of these methods sort of deals with the issue but ends up also creating more; when cutting the boundary it still remains to figure out how and where the infinite lines intersect with it and how to best cut; adding more points is in principle simple but how to choose these suitably in all case. In the reality a combination of the two is what is used. That is, first extra points are added all around the measurement area, generating a bigger voronoi diagram; secondly the outer pixels are cut by the boundary. Thus the requirement on the position of the additional points is loosened as one is free to only add a small amount of extra points (specifically 8 extra points are added: above, below, left, right, and diagonally, with respect to the mean position).

## 4.5 Visualization methods

This section is dedicated to give both an overview of the available visualization methods and also an in depth explanation of the workings of them.

### 4.5.1 Discussed visualization tools for ‘Live view’

As discussed in the technical report of F. Groitel and S. Tóth, the is following non-exhaustive list features and visualizations needed by the user during an experiment at the CAMEA instrument

- 1D line plots
  - Intensity as function of  $\Delta E$  - So-called Dimer plot
  - Constant energy between two  $Q$ -points
  - Constant  $Q$ -point against  $\Delta E$
- 2D Colour plots
  - Powder average, intensity against  $\Delta E$  and  $|Q|$
  - Constant energy planes, intensity against  $Q_x$  and  $Q_y$  or scattering plane vectors for given energy width
  - Intensity against detector index and unbinned pixel
  - Intensity against detector index and  $\Delta E$
  - Intensity against energy and  $Q$ -points, for a list of  $Q$ -points plot the intensity as function of  $\Delta E$  and  $\vec{Q}$

The last bullet under the 2D plots has been added since the last review of the document (medio February 2017).

Some of the above visualizations have already been met by the developed software, and will below be explained in some detail as to elucidate the underlying algorithms and their limitations. Software specifications can be found in the ‘Data Module’\_

For more details, see <DataSet>ShortAnchor\_.

This is a small set of notes explaining the methods used to optimize different parts of the software code. It is meant as an overview of the changes and thoughts that have gone into the code structure and changes made.

### 5.1 Optimizing of the plotA3A4 routine

In order to make the method *plotA3A4* usefull in a practical mannor, it needs to be somewhat fast in its computation. Otherwise, one will not use it and instead use binning, or even worse another program! And as it is already mentioned in the documentation, the computation time for generating all pixels individually for the methods *plotA3A4* and *plotQPatches* is too long for practical use. Thus, an optimization of the underlying algorithm made sense to persue. Before changing any code, the end test was set up: By first running the old method and then the new, two results are generated and they are checked to be idential. Is so, and the with a speed-up, the goal is reached.

Thus, before headlessly trying to perform optimization, a test of the current speed is needed. The following is a dump of times for the un-optimized function using four files and one plane with 8 subplanes (T0Phonon10meV.nxs, T0Phonon10meV90A4InterlaceA3.nxs, T0Phonon10meV93\_5A4.nxs, and T0Phonon10meV93\_5A4InterlaceA3.nxs, planes 8 through 15):

Function	Time [s]	Uncertainty [s]
testFiles	0.0003143	2.2458e-05
getA3A4	2.8371810e-05	1.0929e-05
getData	0.0561441	0.0118
concatINormMon	0.0405629	0.0068
A4Instr	0.0001758	2.9835-05
genPointsAndBoundary	0.2238357	0.0151
voronoiTessellation	14.070171	1.6175
calcCentroids	0.4847603	0.0366
sortPoints	25.009635	0.7448
calculateQ	3.4248633	0.0778
binPlanes	0.0295210	0.0007
genPatchesAndCollection	8.7695337	0.0920
plotter	0.1842771	0.0093

From this it is clear that three functions are to be looked at: *voronoiTessellation*, *sortPoints*, and *genPatchesAndCollection*.

## 5.2 Voronoi Tessellation subroutine

Individual functions are timed by the using *my\_timing\_N* decorator. This allows for a partitioning of the program into smaller pieces that are easier to be optimized individually. Also when changing code in these sub-functions, the test was of course that the output is identical to the old one. This method is not the optimal for speeding up the code as it is most probable that changing larger coding structures might allow for a quicker speedup.

Below is a table of the resulting computational times for the tests used showing a clear speed-up of around 1.7 for real data.

Tests (5 runs)	Original [s]	Optimized [s]	Gain
100000 random points between 0 and 1	10.27 $\pm$ 0.15	4.45 $\pm$ 0.08	2.310 $\pm$ 0.05
1000000 random points between 0 and 1	106.7 $\pm$ 0.8	47.75 $\pm$ 0.7	2.235 $\pm$ 0.04
4 Data files and 8 planes *	10.93 $\pm$ 0.08	6.3 $\pm$ 0.2	1.72 $\pm$ 0.06

\*: setup from above.

The speed-up comes mainly from two changes; when generating the return data the intersections points of all of the polygons were recalculated while only the polygons cut by the boundary needed to be calculated. Secondly, when testing if all data points are within the boundary a list comprehension is changed into the vectorized *contains* function from the *shapely.vectorized* sub-library.

## 5.3 Timing function

In order to quantify whether or not a speed-up has been achieved, one needs to time the method in question. This is most easily done through the use of the package *time* from Python, where one simply requests the current time in seconds before and after the method has been running. Subtracting the two then gives an estimate of the time consumption. The reason for the use of the word ‘estimate’ is that this is not the correct time used by the computer. The true time is the actual CPU time or even better, the number of CPU cycles needed to run the method. The reason the *time* module does not capture this is that if the program does not run as the only process and with a 100% usage of the core then a discrepancy between measured and actual time consumption is created. However, these technicalities

are not taken into account when I have performed the profiling of the code as I in no case am able to perform a superb optimization with a background in pure physics.

The decorator used for profiling is given below:

```

1 def my_timer_N(N=3):
2     if N<0:
3         raise AttributeError('Number of runs need to be bigger or equal to 1 or equal_
↳to 0 for no timing, but {} given.'.format(N))
4     def my_timer(func):
5         import time
6         def newFunc(*args,**kwargs):
7             Time = []
8             if N==0:
9                 returnval = func(*args,**kwargs)
10            else:
11                for i in range(N):
12                    startT = time.time()
13                    returnval = func(*args,**kwargs)
14                    stopT = time.time()
15                    Time.append(stopT-startT)
16                if N>1:
17                    print('Function "{}" took: {}s (±{}s)'.format(func.__name__, np.
↳mean(Time), np.std(Time)/np.sqrt(N)))
18                else:
19                    print('Function "{}" took: {}s'.format(func.__name__, Time[0]))
20                return returnval
21            return newFunc
22    return my_timer

```

With this definition of the decorator, using it to time a function is straight forward, i.e.

```

1 def untimedFunction(*args,**kwargs):
2     # some calculations
3     return True
4
5 @my_timer_N(N=5)
6 def timedFunction(*args,**kwargs):
7     # some calculations
8     return True

```

Thus, the above code allows for timing the function *timedFunction* called 5 times, producing the output: Function “timedFunction” took: 1.1920928955078125e-06s (±1.3571023436315258e-06s), where the first time is the average of the N=5 runs, and the parenthesis denotes the uncertainty on the mean,  $std(X)/\sqrt{N}$ .



---

## Commissioning

---

This part of the documentation is intended to contain a sort of commissioning log-book and will be updated (hopefully) frequently with the newest information about the process. Nice to have tables and figures are linked below table of contents.

### 6.1 29/10-18 - Start of hot commissioning

---

**Note:** No neutrons on the instrument due to the need of adjusting the motor controller for A4. The motor inertia is too small as it has been calibrated for RITA2 and not CAMEA.

---

Monday 29th of October. Today, the following is planned:

- First neutrons on the instrument
- Initial check of old alignment of A1-A2

Possible difficulties to be discussed:

- **Tuning of the bias voltage of the detectors can be difficult as there is non-linearities in measurements.**
  - On RITA2 this was countered by putting a mask in front of the 2D detector.
  - The CAMEA back-end has a build-in mask due to the cross-talk shielding and the scattering of analysers into different parts of the detectors.
  - It could be possible to assume that the design energies of the back-end are true making it possible to figure out  $E_i$  independent of the A4 angle.
  - The energies expected is provided in table [EfTable](#)

Anal- yser	$A_5$ [deg]	$E_i$ calculated [meV]	$E_i$ article [meV]	$E_i$ McStas [meV]	$E_i$ Measurement [meV]
0	48.90	3.200	3.21	3.204	3.177270 ± 0.145755
1	47.21	3.374	3.38	3.379	3.358399 ± 0.159861
2	45.53	3.568	3.58	3.575	3.550246 ± 0.173190
3	43.84	3.787	3.80	3.794	3.763014 ± 0.188041
4	42.16	4.033	4.05	4.041	4.009775 ± 0.202941
5	40.47	4.313	4.33	4.320	4.287641 ± 0.227040
6	38.79	4.629	4.64	4.637	4.603493 ± 0.256536
7	37.10	4.993	5.01	5.000	4.960061 ± 0.289973

EffTable: Final energies are calculated from the angles defined in the McStas simulation and a d-spacing of 3.355 Å for PG. The article numbers are taken from [Groitl2016](#), while the McStas numbers are found from the normalization procedure using 1 software pixel as described in [Software pixel binning](#)

- Alignment of the A2 and A4 are not independent but need to be aligned together

### 6.1.1 Fitting routine for finding $A_2$ and $A_4$ offsets

As mentioned above, the determination of the offsets in  $A_2$  and  $A_4$  is coupled. That is, the offset in  $A_2$  results in an offset in the incoming energy thus changing the scattering angle  $A_4$ . One way of determining these offsets is to consider a sample with known lattice parameters. Further, it is assumed that the lattice parameter of the monochromator is  $d_{pg} = 3.355$  Å. The ‘out-going’ wave from the monochromator must be equal to the ‘in-coming’ wave of the sample (assuming first order scattering), meaning that according to Bragg’s law

$$\lambda_{pg} = 2d_{pg} \sin \frac{A_2 + \delta_{A_2}}{2}, \quad \lambda_S = 2d_S \sin \frac{A_4 + \delta_{A_4}}{2}.$$

$$\lambda_{pg} = \lambda_S \Rightarrow 2d_{pg} \sin \frac{A_2 + \delta_{A_2}}{2} = 2d_S \sin \frac{A_4 + \delta_{A_4}}{2}.$$

One is then left with one equation with two unknowns, but if two reflections in the crystal are known, one can solve the set of equations. However, this relies on the determination of  $A_2$  and  $A_4$  to be without any uncertainty. Often, multiple scattering vectors are possible for a given sample and one can thus over-determine the offsets. This is then formulated as:

$$2d_{pg} \sin \frac{A_{2,i} + \delta_{A_2}}{2} = 2d_{S,i} \sin \frac{A_{4,i} + \delta_{A_4}}{2},$$

where  $A_{2,i}$  denotes angle corresponding to incoming energy  $i$  and  $d_{S,i}$  and  $A_{4,i}$  are the  $i$  ‘th’ lattice spacing and corresponding scattering angle. Allowing  $A_2$  to be dependent on  $i$  allows for changes in incoming energies which might allow for more reflections inside of the limits of the secondary spectrometer. Formally the fitting is performed by minimizing the difference between the calculated wavelengths of the monochromator and sample reflections using least  $\chi^2$ . The uncertainties are then defined for the individual wavelengths as

$$\sigma_{\lambda_i} = \sqrt{4 \sin^2(\theta_i) \sigma_{d_i}^2 + 4 d_i^2 \cos^2(\theta_i) \sigma_{\theta_i}^2},$$



where  $\theta_i$  is either  $A2_i$  or  $A4_i$ . With the assumption of known d-spacing, the uncertainty  $\sigma_d$  can in principle be set to 0. When the optimal values of the two offsets have been found one could define the uncertainties of these as the change in offset resulting in an increase of the  $\chi^2$  value of one.

### Example of offset fitting

A possible sample to be used for such a fitting procedure is AIO in powder form. This ensures that the value of sample rotation,  $A3$ , is unimportant. This material has scattering planes with spacings given by  $d \in \{3.3993, 2.0371, 2.4915, 2.1145\}$  Å, which for an incoming energy of 5 meV gives scattering angles,  $A4$ , of 73.0, 166.2, 108.5, and 146.1 degrees. As to mimic a real measurement a random offset (but equal for all of the reflections) is added together with a Gaussian noise 0.05 degrees on both  $A2$  and  $A4$ . The used values are tabulated in [dataTable](#) below.

$d$ [Å]	$E_i$ [mev]	$A2$ true [deg]	$A2$ 'measured' [deg]	$A4$ true [deg]	$A4$ 'measured' [deg]
3.3993	5.0	74.1426	69.5344	73.0186	69.9029
2.0371	5.0	74.1426	69.4865	166.2374	163.2118
2.4915	5.0	74.1426	69.5796	108.5306	105.4428
2.1145	5.0	74.1426	69.5630	146.0592	142.9605

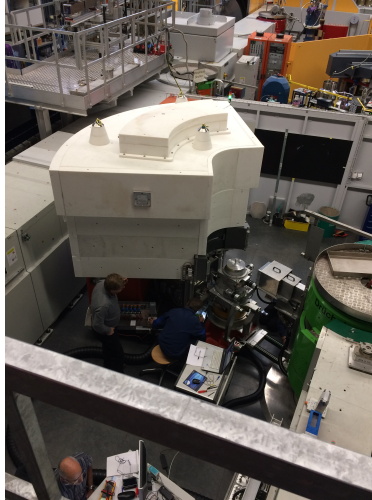
Running the minimization results in an  $A2$  offset of  $-4.620 \pm 0.004$  degrees and an  $A4$  offset of  $-3.14 \pm 0.03$  degrees. These are somewhat consistent with the simulated offsets of -4.608 and -3.067 for  $A2$  and  $A4$  respectively. It is seen that the found values are 3.2 and 2.3 sigmas away suggesting an underestimation of the uncertainties of the offsets.

## 6.2 30/10-18 - Opening of shutter and background

Things done today:

- **Opening of shutter**
  - The neutron guide is not suitably under vacuum and needs to be fixed by turning on pump(s).
  - After restart of pumps and security checks the main shutter and secondary shutter are allowed to be opened.
  - Test of shutter closing when triggered by alarm (beam stop button, malfunctioning external safety system).
- **Check of software**
  - Streaming of data through the UDP pipeline as well as direct logging tested to be functional.
  - Connections between Six and motors, monitors confirmed.
  - Slight reajustment of data path way from detector to histogram memory is needed as data becomes skewed.
- **Measurements**
  - Background measurement of full tank for 83 minutes with a total of around 2500 counts or 0.3 count/(minut tube).

The secondary spectrometer in the direct beam to test that it can go through the space between sample table and guide.



### 6.3 31/10-18 - Data wrangling and measurement

- During the night a background scan was measured with between 0.25 and 0.3 counts/minute in each tube. The distribution is homogenous except for at the ends of the tubes. Here it is believed that high voltage malfunctions add signal.
- The data transfer from the detector through streaming and histogram memory works but the format of the detector matrix in the data files is change from being (np,104,1024)  $\rightarrow$  (np,1024,104)
- Visit by Henrik:



Fig. 1: Visit of Henrik and the mandatory photo session. CAMEA has gotten a name tag as well.

- The first neutrons have been measured!



Fig. 2: First neutrons as observed in the RITA cabin

- No dispersions have yet been measured as those in lead are at too high A4.

## 6.4 01/11-18 - First Vanadium normalization scan

First scan has been conducted of a Vanadium sample with variable  $E_i$  as shown in figure [VanScan](#) with scan file name `camea2018n000038.hdf`

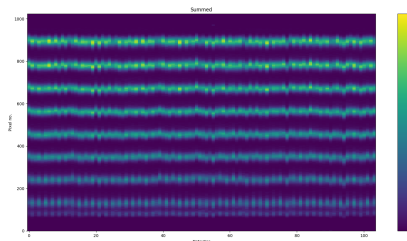


Fig. 3: Summed overview of neutron counts for scan of  $E_i$  from 3.0 meV to 5.5 meV in 501 steps ( $\Delta E_i = -0.005$ ).

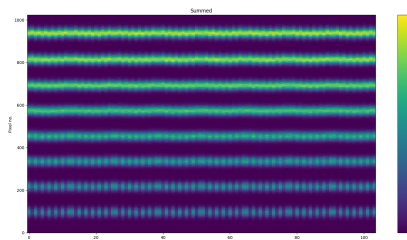


Fig. 4: SIMULATION: Summed overview of neutron counts for scan of  $E_i$  from 3.0 meV to 5.5 meV in 401 steps ( $\Delta E_i = -0.00625$ ).

Assuming that the primary instrument is somewhat well-aligned the above data set is used for the normalization and determination of  $E_f$ .

Further studies have been conducted in the pursuit for reducing the background below pixel 100 in all tubes. Following has been found:

- Background is present for all energies.
- Background seems to be independent of A4 (or slightly)
- Background is dependent on neutron count through slits. (Depends highly on slit openings)
- Background is gone when A1 is wrong, instrument shutter is in, or slits are closed
- Background increases with plastic or Vanadium as sample but is also present without sample
- Background remains despite shielding mat being in front of Be filter (see [MatPicture](#))

One of the current explanations is that air scattering moves neutrons from sample area to material under the tank generating gammas. These then enter the tank and are either wrongly interpreted at detector resulting in increased background at pixel 100. Or, they are weak enough to be absorbed by cross talk shielding thus only allowing gammas to reach detectors at pixel 100.

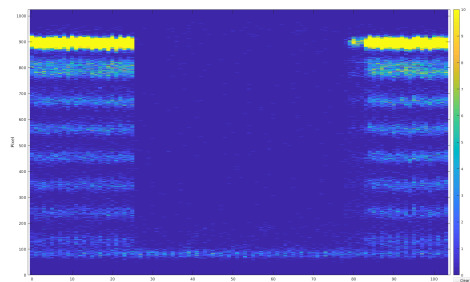


Fig. 5: Detector overview with shielding mat just in front of the middle part of Be filter. As seen, the background remains while neutrons are completely blocked.

## 6.5 02/11-18 - Background hunting

After many attempts to shield different parts of the instrument with borated plastic and boral it was found that the neutrons can actually go through a tiny ‘slit’ between the Be filter and the cross talk shielding. This gap only exists in the front as the shielding after the last analyser bank has a boral plate to cover the direct line.

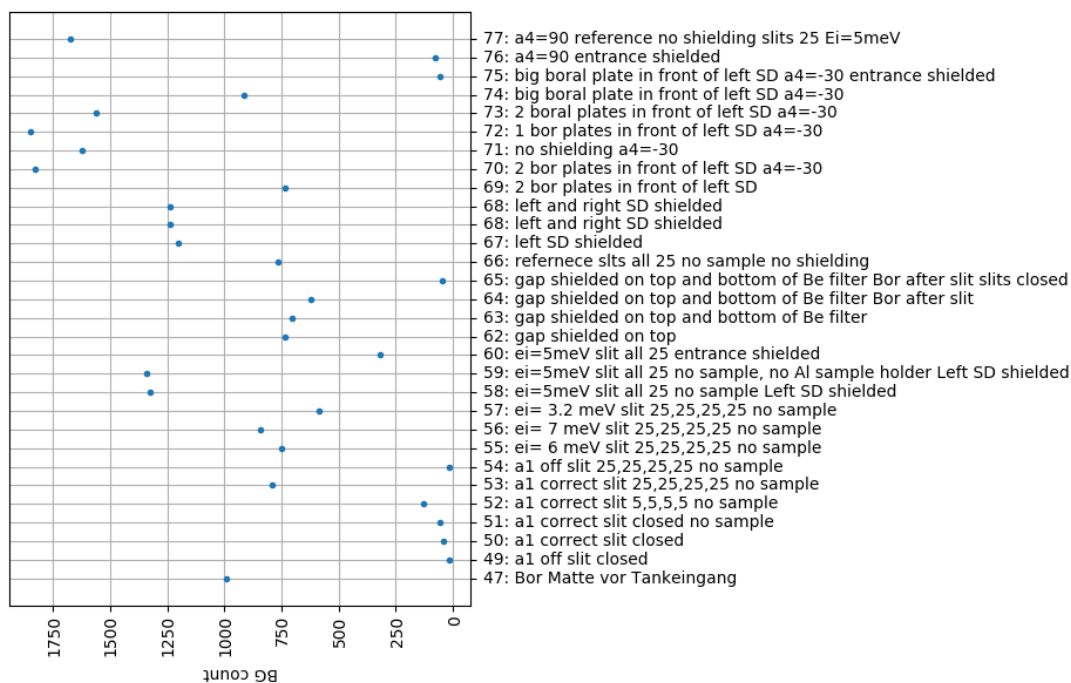
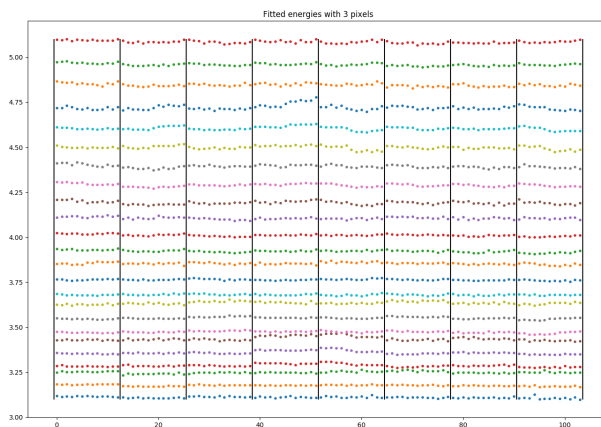
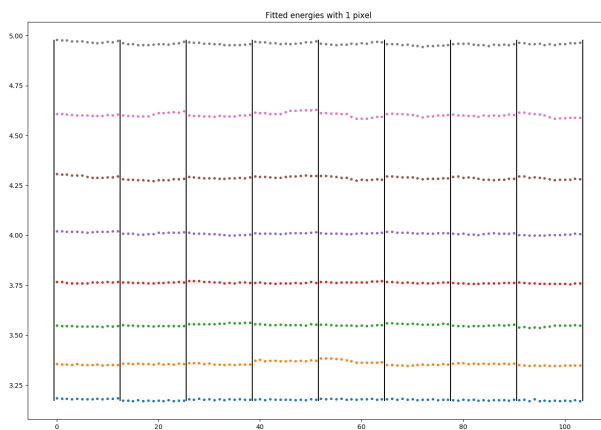


Fig. 6: Overview of background measurements with the scan title and number

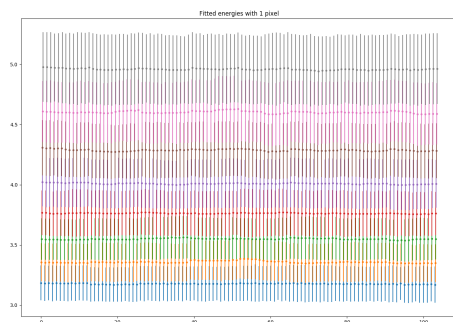
## 6.6 05/11-18 - Energy normalization

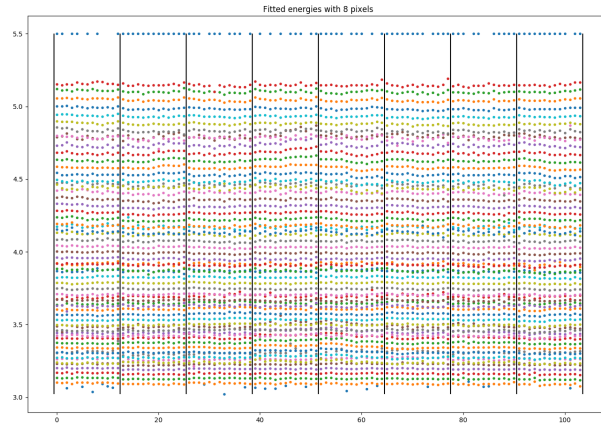
With the energy scan using a curved monochromator and Vanadium, data file ‘camea2018n000038.hdf’, the following three energy diagrams is found. It is noticable that due to the excess background around pixel 80 across all tubes,

and the crude method of masking these when finding energies, the lowest energy of the 3.2 meV analyser bank for 8 software pixel does not converge. This then results in rather arbitrary values found. The values of 1 pixel binning can be found in the table in [291018](#).



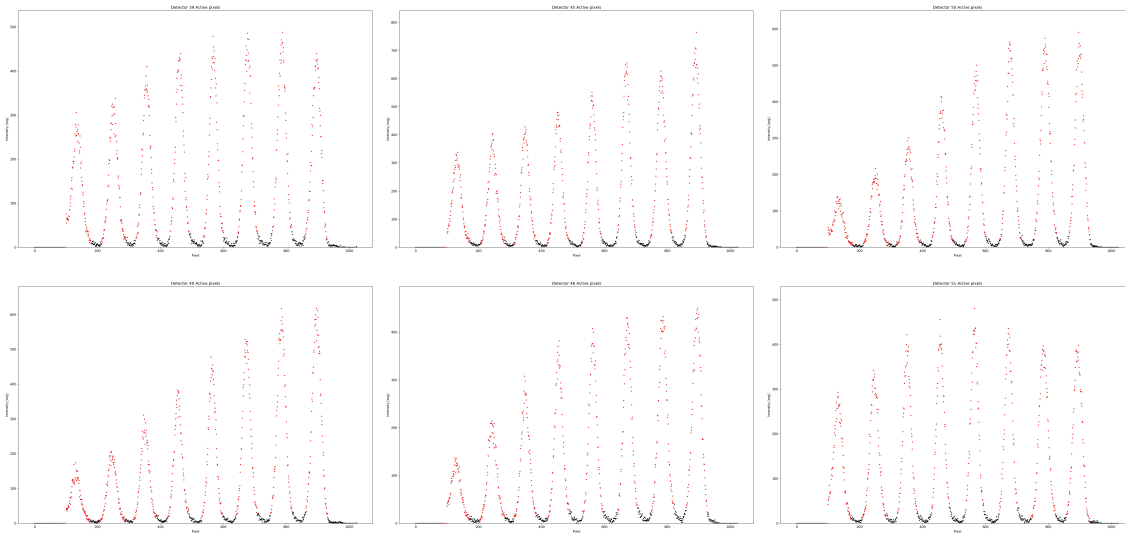
For the FWHM values, below the energy widths for 1 and 3 pixelation is shown.





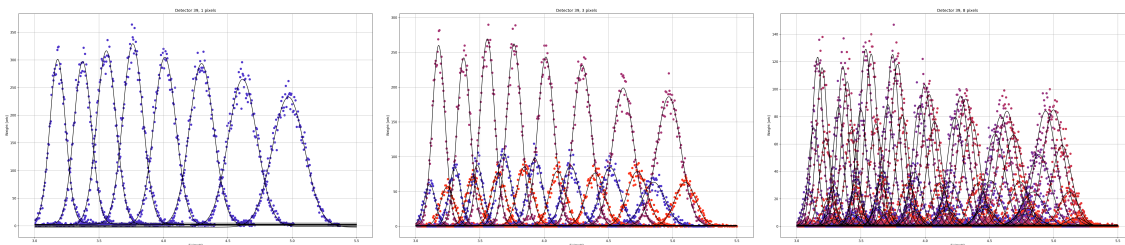
### 6.6.1 Pixel area and fit

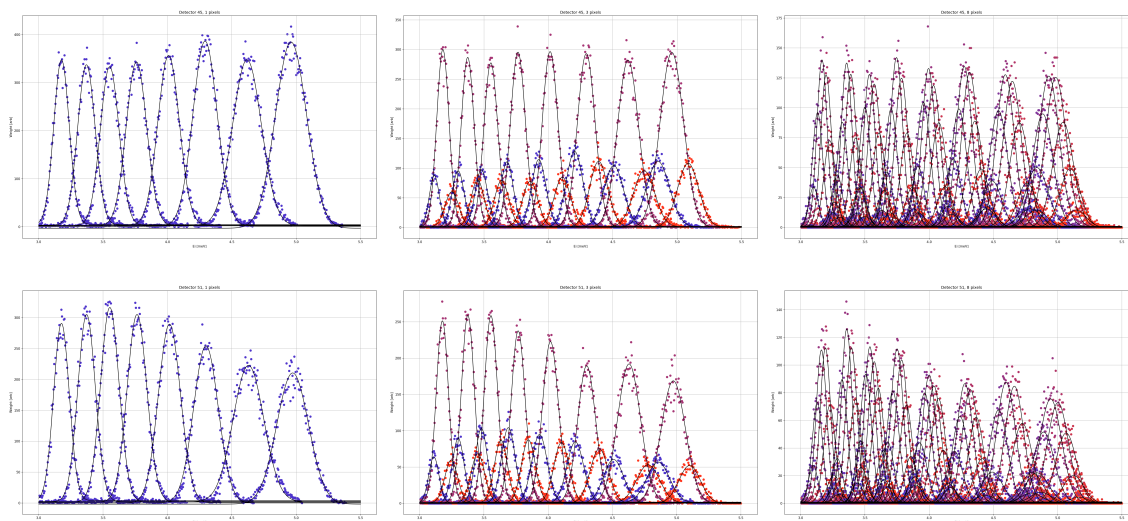
Below are the active areas of detector tubes 39, 40, 45, 46, 50, and 51. These tubes are the four outer most in wedge 4 as well as the middle in the upper and lower layers.



### 6.6.2 Pixel binning 1, 3, and 8

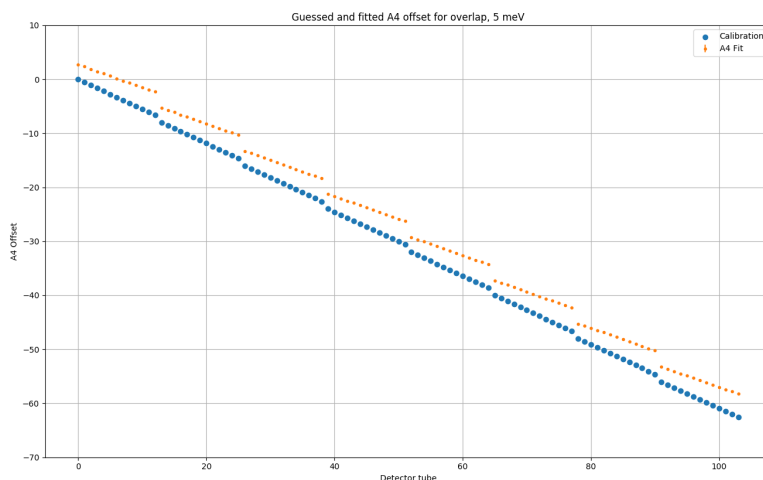
Using all of the pixel binnings for detector tubes 39, 45, and 51 results in the following positions





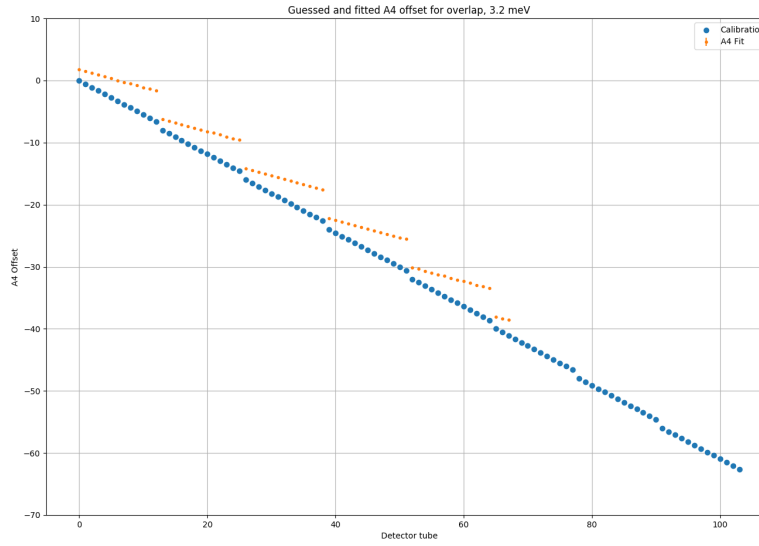
## 6.7 06/11-18 - Determination of A4 + Be filter cooling

Whilst the Be filter has been cooling, the  $2\theta$  data has been analysed. It is specifically files 87 and 88 containing  $2\theta$  scans for AlO at 5 meV. The files contain a scan on each side of the direct beam. As 5 tubes are measured on both sides, these form the basis for determining the proper sample  $2\theta$ . This follows from the fact that the scattering angle is to be negated when moving to the other side of the direct beam while the offset is the same. In an equation, if the AlO peak is found at  $C_1 = 2\theta_S + \theta_{off}$  and the other at  $C_2 = -2\theta_S + \theta_{off}$  then the actual scattering angle is  $2\theta = \frac{C_1 - C_2}{2}$ . For the given data, the value was found to be  $71.1082 \pm 0.0008$  degrees. By subtracting this from all of the other fits, the graph below is found. As seen, there is a general offset for all wedges of  $3.52 \pm 0.03$  degrees between the found and the calibration table. Further, the  $2\theta$  step size between the tubes is also different than the calibration files. This is due to the out of plane scattering as described in *Geometry*.



From the found sample scattering angle, assuming that the lattice parameters are known, the true incoming energy can be found through the regular Bragg scattering condition with the powder ring peak being (0,2,1) of AlO. This is found to be  $E_i = 4.993192$  meV, which compared to the one written in the data file as 4.999995 meV is quite close. However, when using the HFO sample kindly provided by Romain Franck, the calculated  $E_i = 3.1886$  meV as

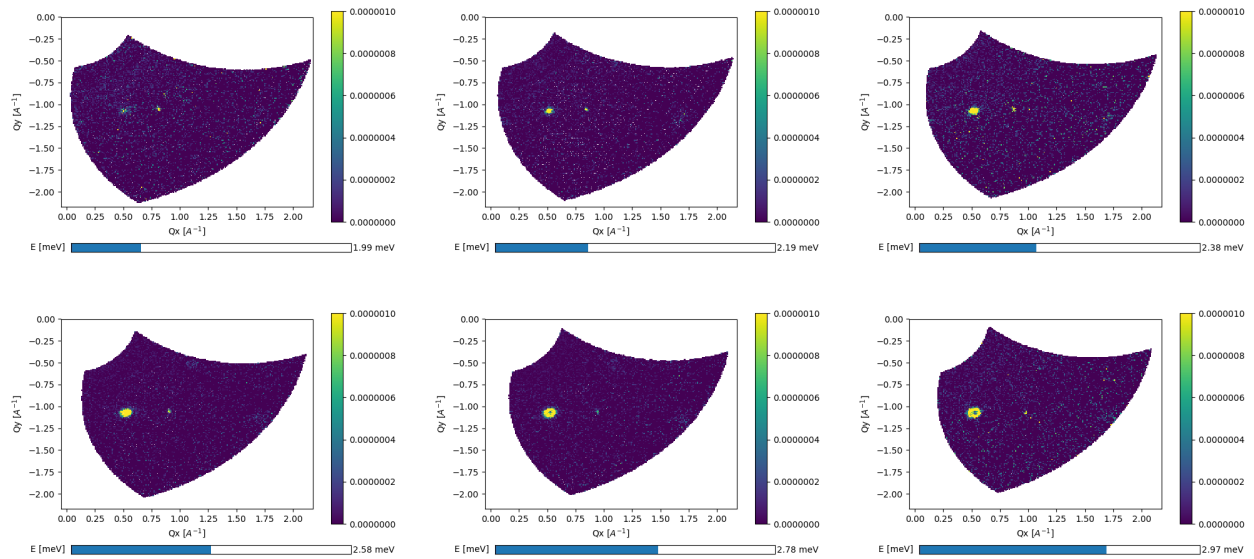




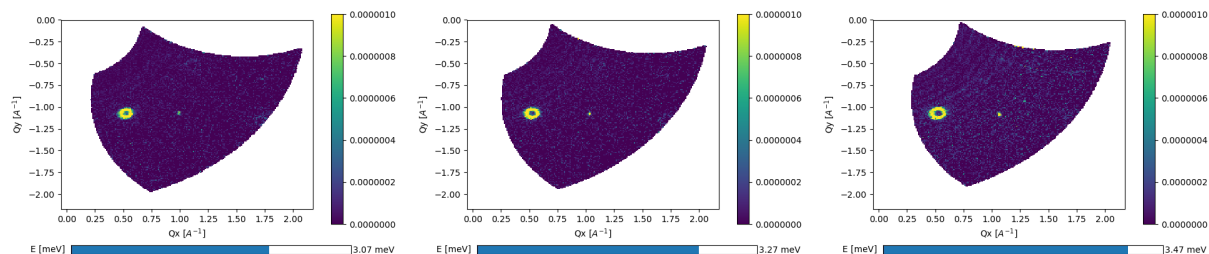
compared to the  $E_i = 3.199995$  meV wanted. This corresponds to an offset in  $A_1$  of about 0.12 degrees. This is to be investigated.

## 6.8 09/11-18 - First magnon in YMnO3

By performing 2 A3 scans with 0.5 step size and 121 points, counting roughly 3 minuts/point and  $2\theta$  of -20 and -40 degrees, the scans shown below are found. In total, 12 hours have been used for this scan.

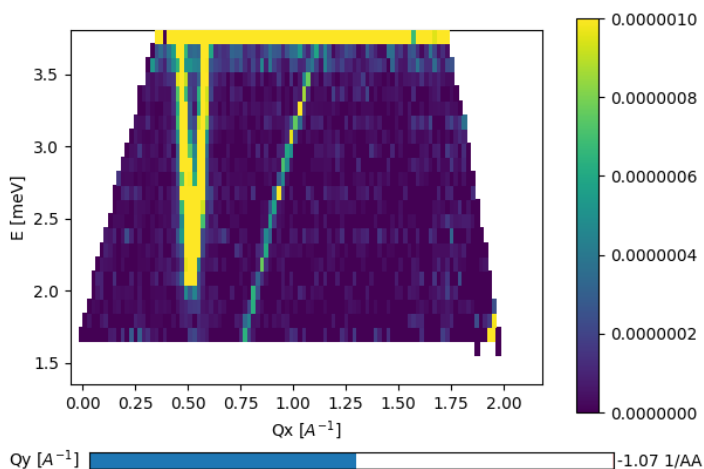






## 6.9 10/11-18 - Currat Axe Spurion in YMnO<sub>3</sub>

The previous scans has shown that the background is really good and low. However, there is a spurious signal which moves in Q-E space. It is seen in all of the constant energy plots as a sharp dot and moves towards the (1,0,0) magnetic Bragg peak when going down in energy transfer, c.f. below. Currently it is believed that the signal originates from the strong Bragg peak that is transmitted through the filter (tough increased in strength) and somehow gets scattered in the analysers.



## 6.10 12/11-18 - No beam

It is difficult to measure without neutrons....

## 6.11 13/11-18 - No beam

It is difficult to measure without neutrons....

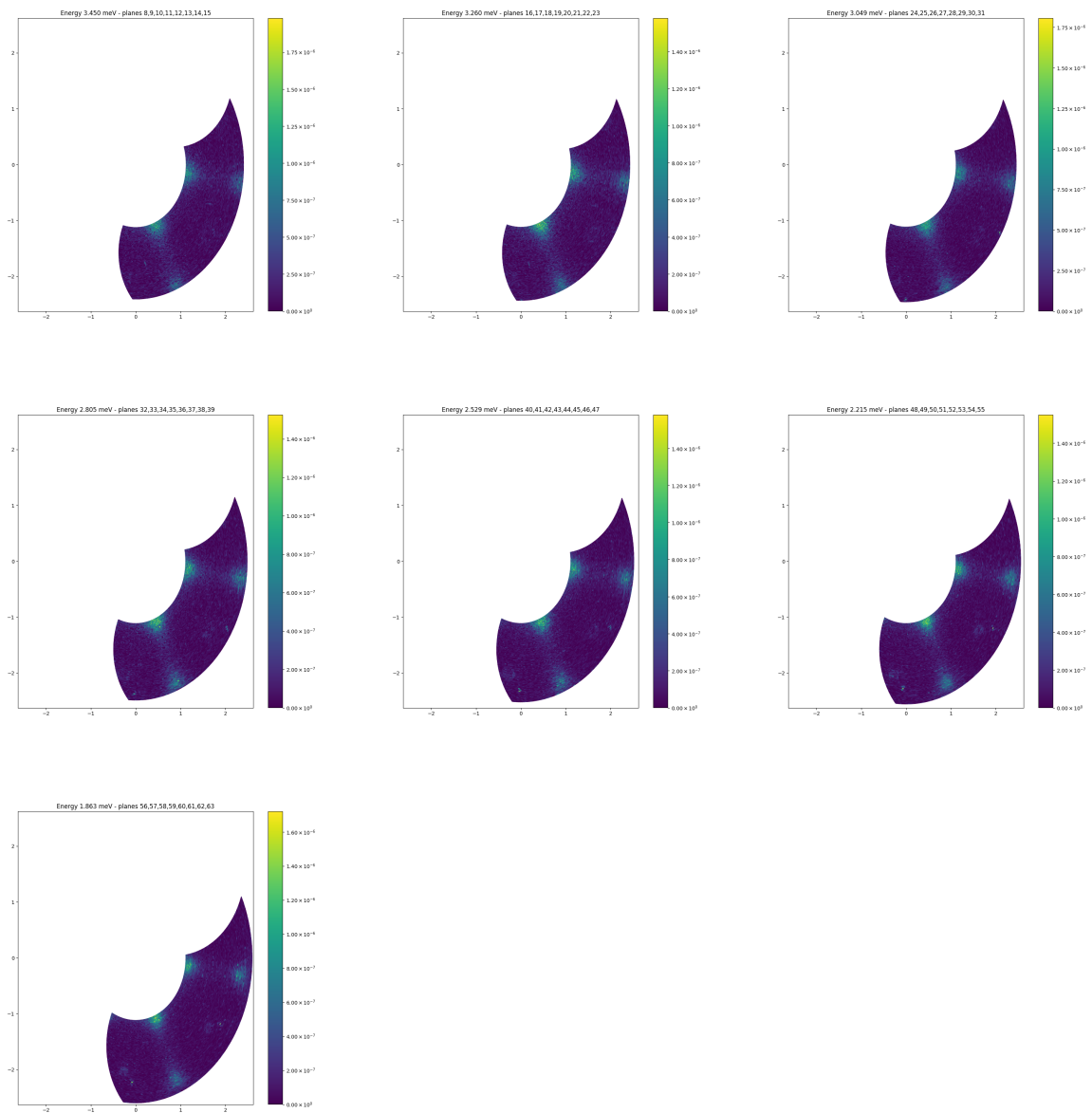
## 6.12 14/11-18 - No beam

It is difficult to measure without neutrons. . . .

## 6.13 15/11-18 - No beam

It is difficult to measure without neutrons. . . .

## 6.14 16/11-18 - Diffuse scattering

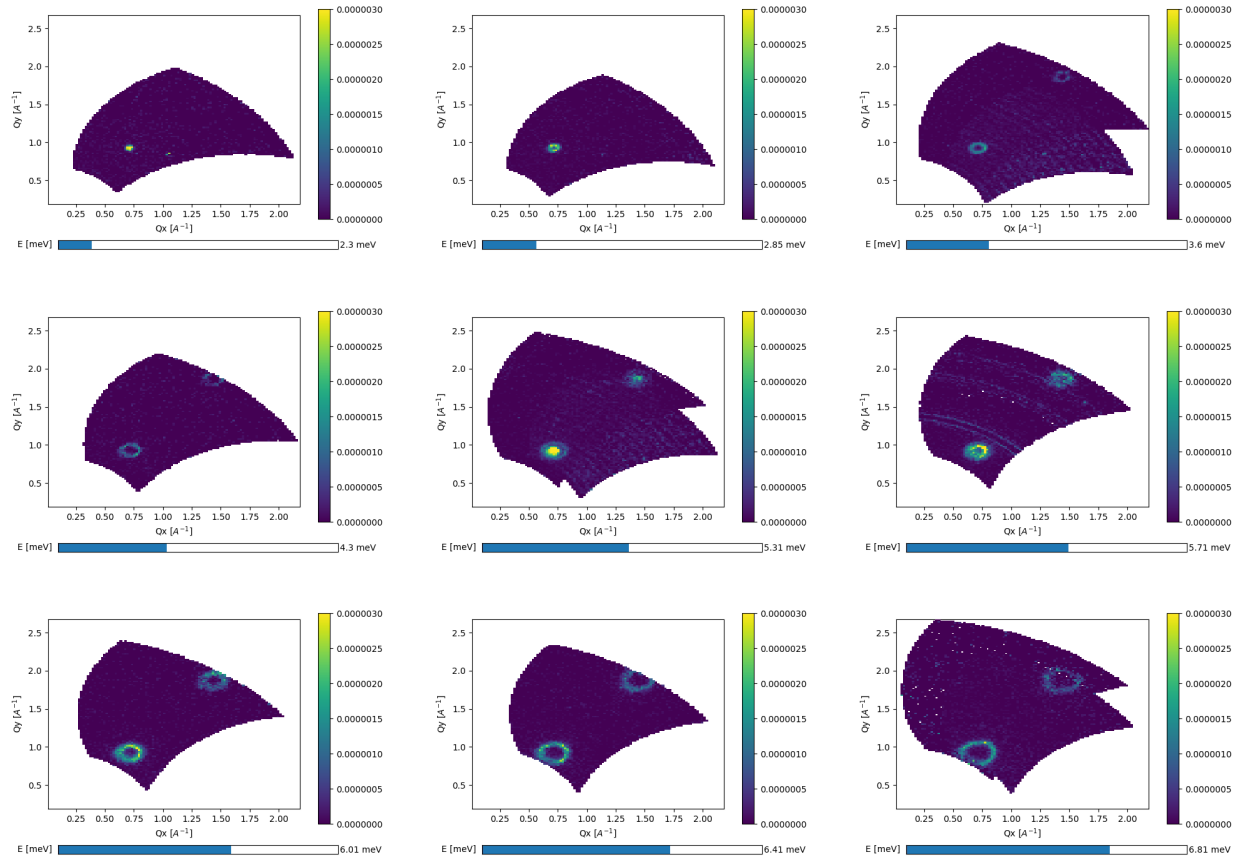


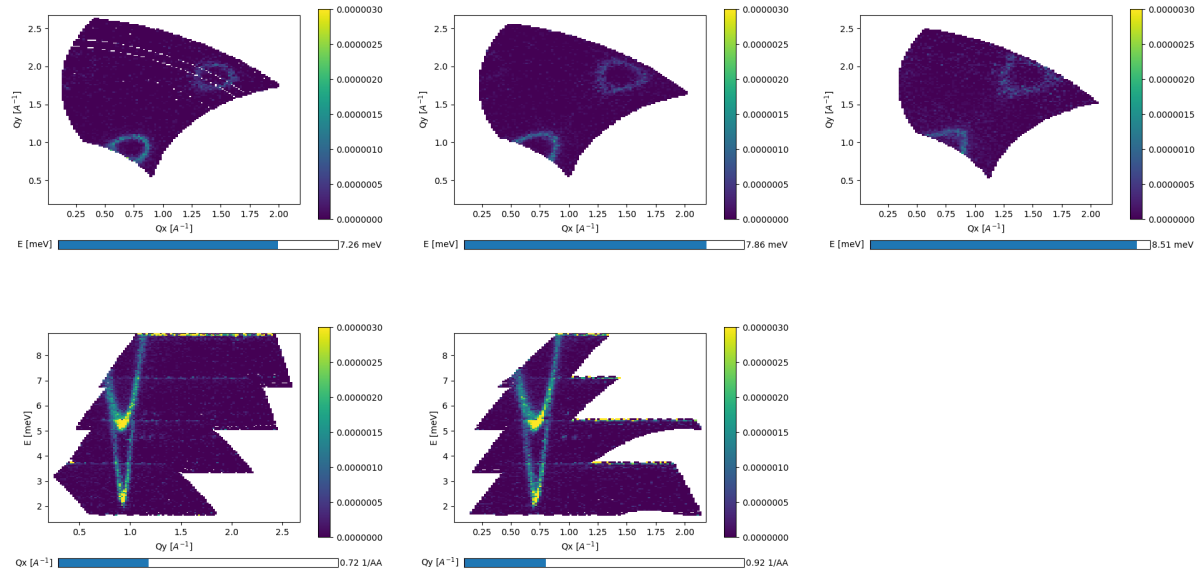
## 6.15 17/11-18 - Magnon in YMnO<sub>3</sub>

With the combination of incoming energies  $E_i$  and angles  $A4$  the following data has been taken. As seen, the two spin waves emerge and overlap as expected. The following settings has been measured:

- 161: A4: 84.000 deg, A3: -20.000 - 20.000 deg, Ei 6.800 meV
- 162: A4: 80.000 deg, A3: -20.000 - 20.000 deg, Ei 6.800 meV
- 163: A4: 84.000 deg, A3: -10.000 - 30.000 deg, Ei 8.500 meV
- 164: A4: 80.000 deg, A3: -10.000 - 30.000 deg, Ei 8.500 meV
- 165: A4: 76.000 deg, A3: -0.000 - -0.000 deg, Ei 10.200 meV
- 166: A4: 76.000 deg, A3: -0.000 - 40.000 deg, Ei 10.200 meV
- 167: A4: 80.000 deg, A3: -0.000 - 40.000 deg, Ei 10.200 meV
- 168: A4: 76.000 deg, A3: 7.000 - 47.000 deg, Ei 11.900 meV
- 169: A4: 80.000 deg, A3: 7.000 - 47.000 deg, Ei 11.900 meV

**Note:** The dispersion is measured on the + + scattering side to check resolution.





## 6.16 18/11-18 - Spinwaves in PbTi

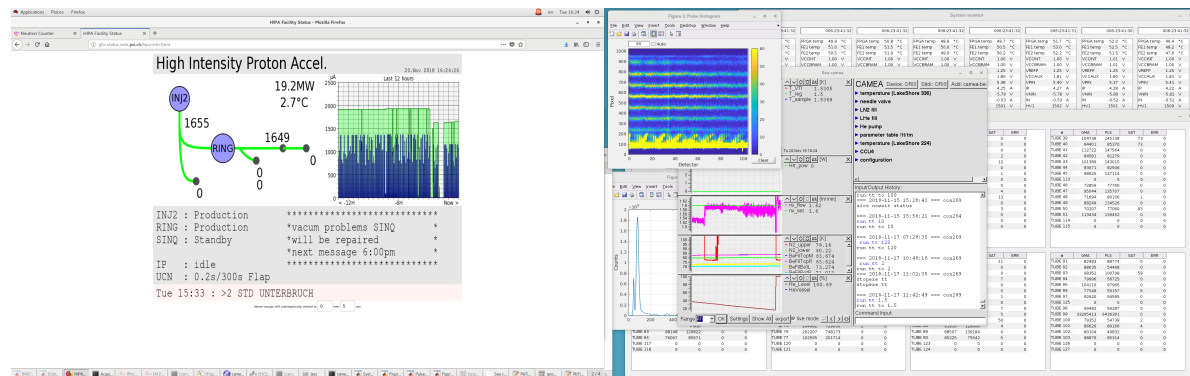
In data files 178 through 190 measurements of PbTi is saved showing nice spin waves.

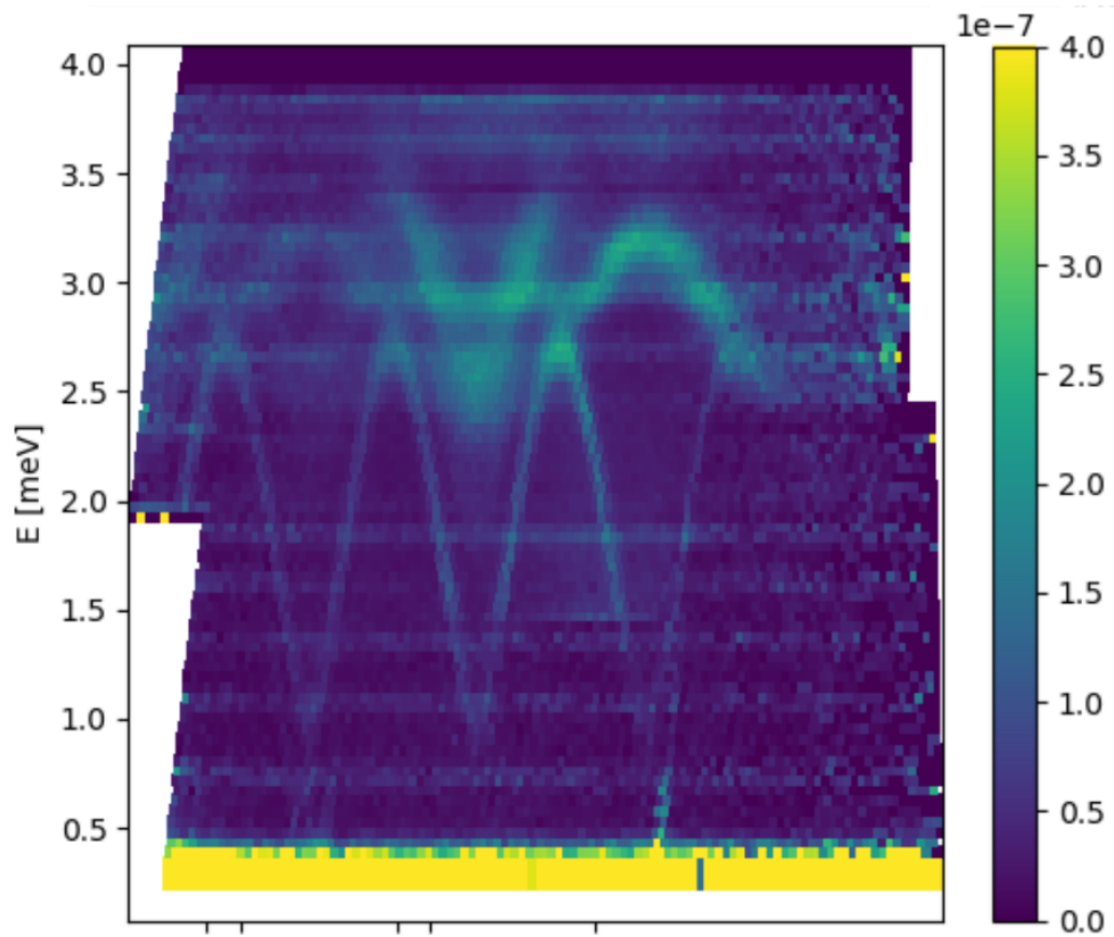
## 6.17 19/11-18 - Spinwaves in PbTi

Continuation of measurments of the nice spin waves. The following spin waves have been measured (summation of all H-values as the compound is independent of this value).

## 6.18 20/11-18 - Vacuum problems at SINQ

Today we have had problems with the beam. . .





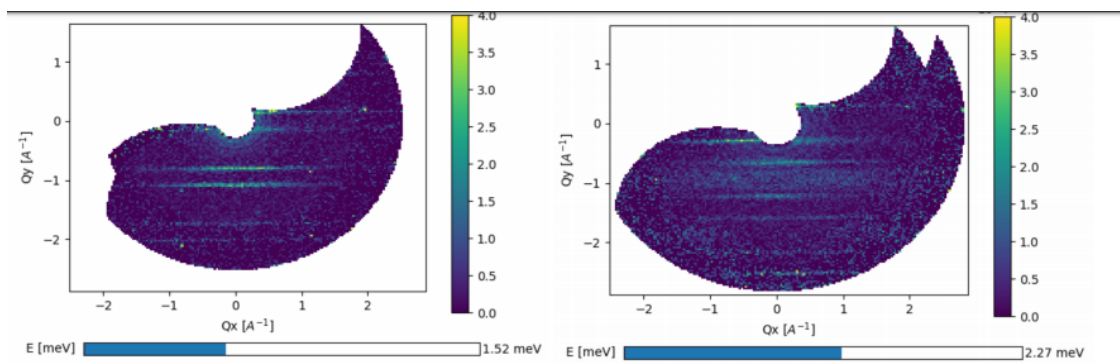


Figure 2

Figure 3

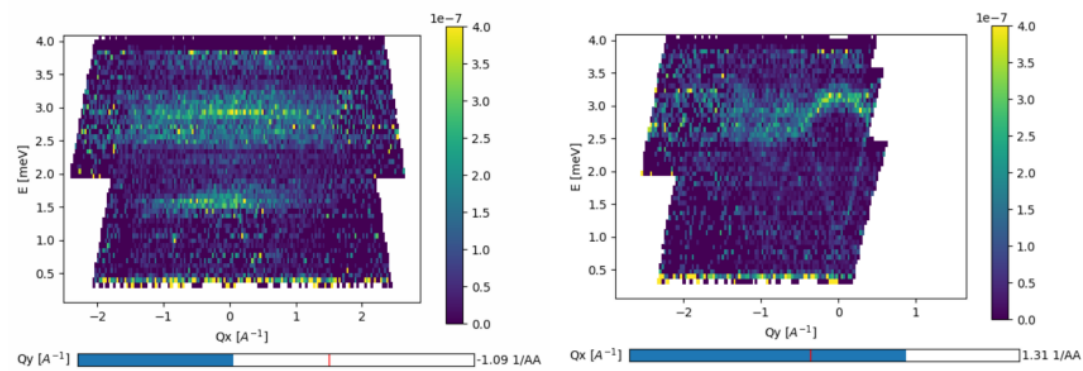


Figure 4

Figure 5

## 6.19 21/11-18 - Measurement of CuSeO<sub>3</sub>

With the small sample and weak signal, the excitations of CuSeO<sub>3</sub> is very difficult to measure. This has to be viewed in the light of the sample also being measured at Thales with 6 min/point. There, the signal was also found to be weak.

## 6.20 21/11-18 - Measurement of CuSeO<sub>3</sub> II

Continuation of measurements.

## 6.21 23/11-18 - Startup of Ni<sub>3</sub>TeO<sub>6</sub>

Today the measurement of Ni<sub>3</sub>TeO<sub>6</sub> has started. The experiment is done to make a direct comparison with the data measured at MultiFLEXX at HZB with the same exact sample.

## 6.22 24/11-18 - Measurement of Ni<sub>3</sub>TeO<sub>6</sub> II

Continuation of measurements.

## 6.23 25/11-18 - Measurement of Ni<sub>3</sub>TeO<sub>6</sub> III

Continuation of measurements.

## 6.24 26/11-18 - Measurement of YMnO<sub>3</sub> Startup

Start of YMnO<sub>3</sub> measurements.

## 6.25 27/11-18 - Measurement of YMnO<sub>3</sub> II

Continuation of YMnO<sub>3</sub> measurements.

## 6.26 28/11-18 - Measurement of YMnO<sub>3</sub> III

Continuation of YMnO<sub>3</sub> measurements.

## 6.27 29/11-18 - Measurement of YMnO<sub>3</sub> IV

Continuation of YMnO<sub>3</sub> measurements.

## 6.28 30/11-18 - Measurement of YMnO<sub>3</sub> V

Continuation of YMnO<sub>3</sub> measurements.

## 6.29 01/12-18 - Measurement of YMnO<sub>3</sub> VI

Continuation of YMnO<sub>3</sub> measurements.

## 6.30 02/12-18 - Startup of Ming Purple

Startup of Ming Purple measurements.

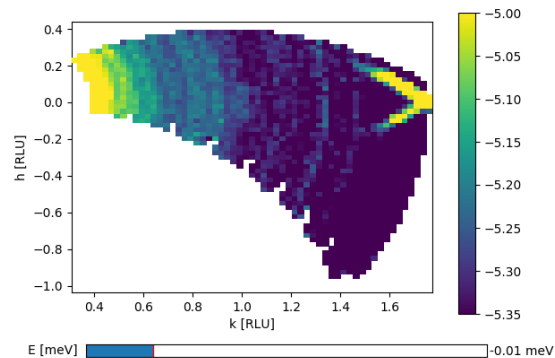
## 6.31 03/12-18 - Ming Purple II

Continuation of Ming Purple measurements.

## 6.32 04/12-18 - Magnet force test and Startup of LSCO

Today, the MA15 was mounted at the sample table to test forces exerted by the magnet on the monochromator and magnet on the secondary spectrometer. No malfunction of anything during ramping up til 12 T and forces are towards the monochromator and of same strengths as earlier with the RITA2 tank. Conclusion: Tank is not a problem in magnetic field.

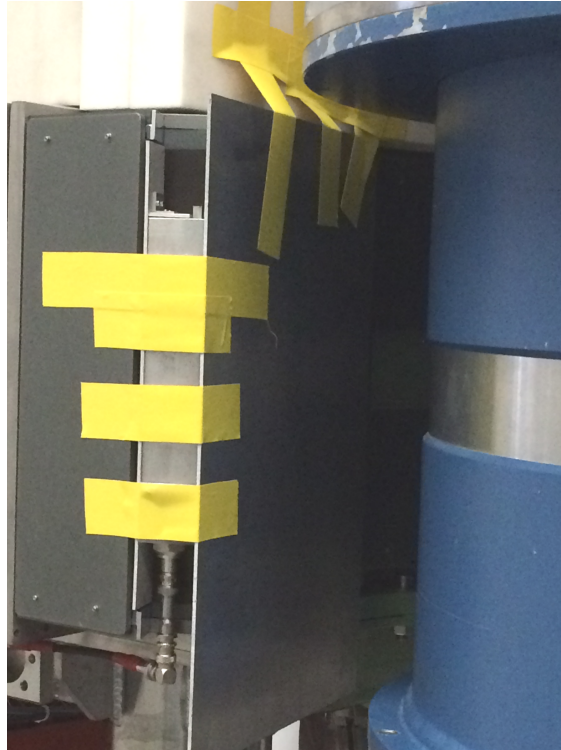
Startup of LSCO 5% doped sample. Measurements at  $E_i$  5 meV elastic. Signal is very weak but this was expected from measurements on Thales.



## 6.33 05/12-18 - LSCO II

Continuation of elastic measurements on LSCO. Changed  $E_i$  to 4.05 meV incoming to move Bragg peak to higher  $A_4$  values to avoid the excess background from the direct beam hitting the side of the tank opening. Further, a Boral plate was mounted.





### 6.34 06/12-18 - Christmas and Ming Purple

Christmas is upon us and this also shows, even in SINQ. The beautiful christmas hat has been masterfully created by Ana and will forever be her legacy at PSI. Startup of Ming Purple measurements.



### 6.35 07/12-18 - Christmas and Ming Purple

Continuation of measurements on Ming Purple. Due to low statistics, more measurement time is granted.

## **6.36 08/12-18 - Christmas and Ming Purple**

Continuation of measurements on Ming Purple.

## **6.37 09/12-18 - Startup of ???**

To be found out.

## **6.38 10/12-18 - Beam Down**

## **6.39 11/12-18 - Beam Down**

## **6.40 12/12-18 - Beam Down**

## **6.41 13/12-18 - Beam development**

## **6.42 14/12-18 - Startup of K2Ni2**

## **6.43 15/12-18 - K2Ni2 II**

Continuation of K2Ni2

## **6.44 16/12-18 - K2Ni2 III**

Continuation of K2Ni2

## **6.45 17/12-18 - Start of SCBO**

Startup of SCBO.

## **6.46 18/12-18 - SCBO II**

Continuation of SCBO

## **6.47 19/12-18 - Start of MnF2**

Startup of MnF2.

## 6.48 20/12-18 - MNF2 II

Continuation of MNF2 measurement.

## 6.49 21/12-18 - MNF2 III and Beam Shutdown

End of MNF2 measurement and beam shutdown at 06:00.

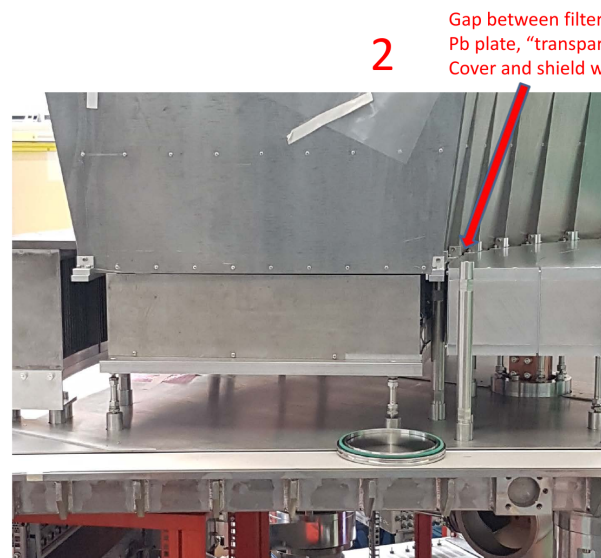
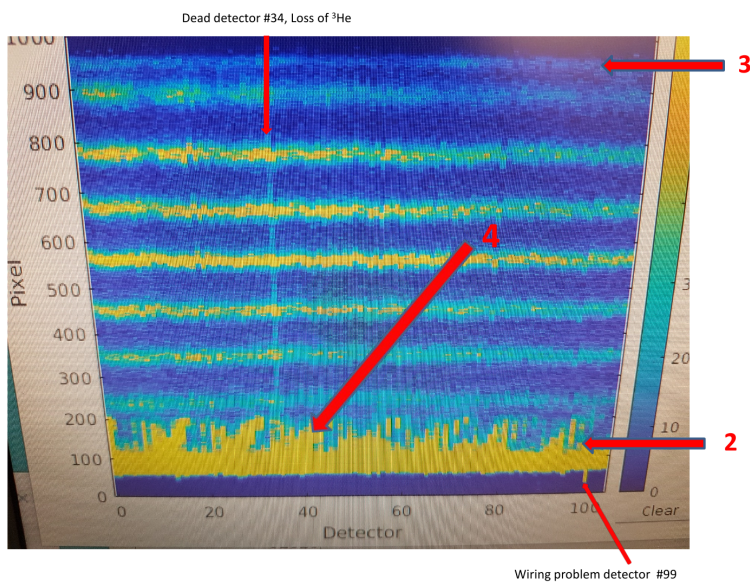
Important links:

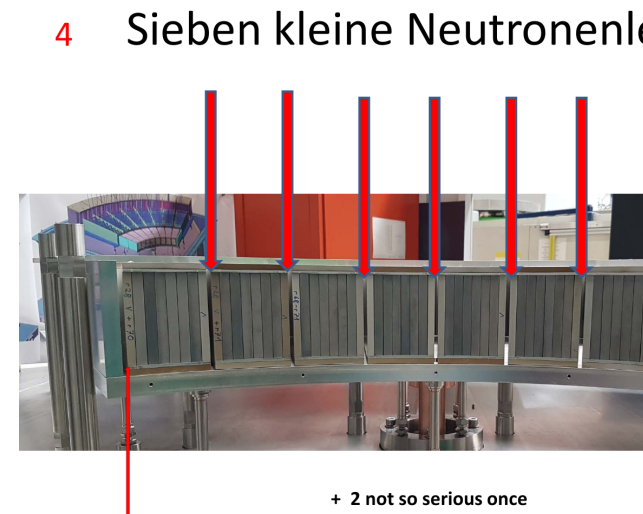
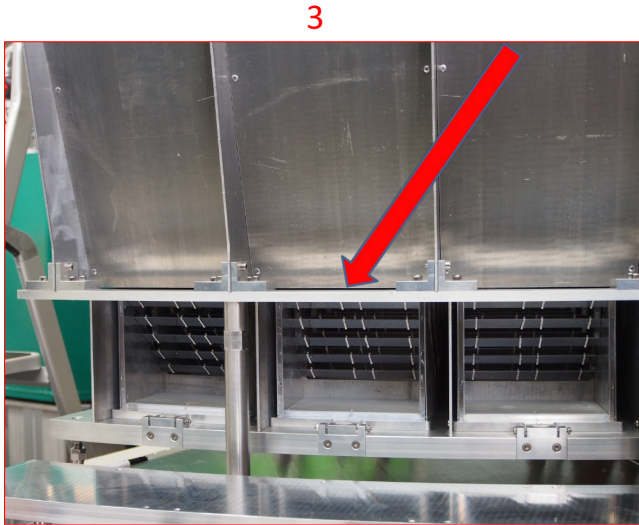
- Overview of calculation and McStas simulations of final energy averaged over all detectors ([291018](#))

Other links:

## 6.50 Shielding Issues

During the course of commissioning instrument specific spurions have been found.





## 6.51 Electronic logbook of scans files

Following is a raw overview of scan files with: name, scan command and title

- camea2018n000001.hdf: sc a3 67.67 da3 0 np 1 ti 30 UNKNOWN
- camea2018n000002.hdf: sc a3 67.67 da3 0 np 1 ti 30 UNKNOWN
- camea2018n000003.hdf: sc a3 67.67 da3 0 np 1 ti 30 UNKNOWN
- camea2018n000004.hdf Scan stopped before first point
- camea2018n000005.hdf: sc a3 28.3 da3 0 np 1 ti 30 UNKNOWN
- camea2018n000006.hdf: sc a3 28.3 da3 0 np 1 ti 30 UNKNOWN
- camea2018n000007.hdf: sc a3 28.3 da3 0 np 1 ti 30 UNKNOWN
- camea2018n000008.hdf: sc a3 28.3 da3 0 np 1 ti 30 UNKNOWN
- camea2018n000009.hdf: sc a3 28.3 da3 0 np 1 ti 30 UNKNOWN
- camea2018n000010.hdf: sc a3 28.3 da3 0 np 1 ti 30 UNKNOWN
- camea2018n000011.hdf Scan stopped before first point
- camea2018n000012.hdf: sc a3 25.7 da3 0 np 1 ti 30 UNKNOWN
- camea2018n000013.hdf: sc a3 0 da3 0 np 3 ti 10 UNKNOWN
- camea2018n000014.hdf Scan stopped before first point
- camea2018n000015.hdf: sc a3 0 da3 0 np 3 ti 10 UNKNOWN
- camea2018n000016.hdf Scan stopped before first point
- camea2018n000017.hdf: sc a3 0 da3 0 np 3 ti 10 UNKNOWN
- camea2018n000018.hdf Scan stopped before first point
- camea2018n000019.hdf: sc a3 0 da3 0 np 3 ti 10 UNKNOWN

- camea2018n000020.hdf Scan stopped before first point
- camea2018n000021.hdf: sc a3 0 da3 0 np 3 ti 10 UNKNOWN
- camea2018n000022.hdf Scan stopped before first point
- camea2018n000023.hdf: sc a3 0 da3 0 np 3 ti 10 UNKNOWN
- camea2018n000024.hdf Scan stopped before first point
- camea2018n000025.hdf: sc a3 0 da3 0 np 3 ti 10 UNKNOWN
- camea2018n000026.hdf Scan stopped before first point
- camea2018n000027.hdf: sc a3 0 da3 0 np 3 ti 10 UNKNOWN
- camea2018n000028.hdf Scan stopped before first point
- camea2018n000029.hdf: sc a3 0 da3 0 np 3 ti 10 UNKNOWN
- camea2018n000030.hdf Scan stopped before first point
- camea2018n000031.hdf: sc a3 0 da3 0 np 3 ti 10 UNKNOWN
- camea2018n000032.hdf Scan stopped before first point
- camea2018n000033.hdf Scan stopped before first point
- camea2018n000034.hdf Scan stopped before first point
- camea2018n000035.hdf Scan stopped before first point
- camea2018n000036.hdf: sc a3 25 da3 0.1 np 3 mn 1000 UNKNOWN
- camea2018n000037.hdf: sc ei 3.2 dei 0.05 np 3 mn 1000 UNKNOWN
- camea2018n000038.hdf: sc ei 4.25 dei -0.005 np 501 mn 100000 Vanadium normalization Be filter warm MCV 80
- camea2018n000039.hdf: sc a4 -52 da4 -0.5 np 5 mn 10000 Vanadium normalization Be filter warm MCV 80
- camea2018n000040.hdf: sc a4 -32.5 da4 0.2 np 226 mn 20000 a4 scan at 5 meV Al2O3 MV 80
- camea2018n000041.hdf: sc a4 -32 da4 0.2 np 221 mn 20000 a4 scan at 5 meV Al2O3 MV 80
- camea2018n000042.hdf: sc a3 25.1 da3 0 np 1 mn 10000 a4 scan at 5 meV Al2O3 MV 80
- camea2018n000043.hdf: sc gm 2 dgm 0.2 np 11 mn 10000 a4 scan at 5 meV Al2O3 MV 80
- camea2018n000044.hdf: sc gm 1.5 dgm 0.3 np 11 mn 10000 a4 scan at 5 meV Al2O3 MV 80
- camea2018n000045.hdf: sc a1 37.082 da1 0.1 np 11 mn 10000 a4 scan at 5 meV Al2O3 MV 80
- camea2018n000046.hdf: sc a1 37.082 da1 0.1 np 11 mn 10000 a4 scan at 5 meV Al2O3 MV 80
- camea2018n000047.hdf: sc a3 25.1 da3 0 np 1 mn 100000 Bor Matte vor Tankeingang
- camea2018n000048.hdf Scan stopped before first point
- camea2018n000049.hdf: sc a3 25.1 da3 0 np 1 ti 115 a1 off slit closed
- camea2018n000050.hdf: sc a3 25.1 da3 0 np 1 mn 100000 a1 correct slit closed
- camea2018n000051.hdf: sc a3 25.1 da3 0 np 1 mn 100000 a1 correct slit closed no sample
- camea2018n000052.hdf: sc a3 25.1 da3 0 np 1 mn 100000 a1 correct slit 5,5,5,5 no sample
- camea2018n000053.hdf: sc a3 25.1 da3 0 np 1 mn 100000 a1 correct slit 25,25,25,25 no sample
- camea2018n000054.hdf: sc a3 25.1 da3 0 np 1 ti 115 a1 off slit 25,25,25,25 no sample

- `camea2018n000055.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 ei= 6 meV slit 25,25,25,25 no sample`
- `camea2018n000056.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 ei= 7 meV slit 25,25,25,25 no sample`
- `camea2018n000057.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 ei= 3.2 meV slit 25,25,25,25 no sample`
- `camea2018n000058.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 ei=5meV slit all 25 no sample Left SD shielded`
- `camea2018n000059.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 ei=5meV slit all 25 no sample, no Al sample holder Left SD shielded`
- `camea2018n000060.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 ei=5meV slit all 25 entrance shielded`
- `camea2018n000061.hdf`: `sc ei 4.25 dei -0.005 np 501 mn 100000 MV80 Plexi normalization`
- `camea2018n000062.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 gap shielded on top`
- `camea2018n000063.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 gap shielded on top and bottom of Be filter`
- `camea2018n000064.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 gap shielded on top and bottom of Be filter Bor after slit`
- `camea2018n000065.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 gap shielded on top and bottom of Be filter Bor after slit slits closed`
- `camea2018n000066.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 refernece slts all 25 no sample no shielding`
- `camea2018n000067.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 left SD shielded`
- `camea2018n000068.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 left and right SD shielded`
- `camea2018n000069.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 2 bor plates in front of left SD`
- `camea2018n000070.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 2 bor plates in front of left SD a4=-30`
- `camea2018n000071.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 no shielding a4=-30`
- `camea2018n000072.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 1 bor plates in front of left SD a4=-30`
- `camea2018n000073.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 2 boral plates in front of left SD a4=-30`
- `camea2018n000074.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 big boral plate in front of left SD a4=-30`
- `camea2018n000075.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 big boral plate in front of left SD a4=-30 entrance shielded`
- `camea2018n000076.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 a4=90 entrance shielded`
- `camea2018n000077.hdf`: `sc a3 25.1 da3 0 np 1 mn 100000 a4=90 reference no shielding slits 25 Ei=5meV`
- `camea2018n000078.hdf`: `sc 2t 77.5 d2t -0.2 np 126 mn 20000 MV 80 Al2O3 positive 2t Ei=5meV`
- `camea2018n000079.hdf`: `sc 2t 77.5 d2t -0.2 np 126 mn 20000 MV 80 Al2O3 positive 2t Ei=4.62meV`
- `camea2018n000080.hdf`: `sc ei 4.25 dei 0.005 np 501 mn 100000 MV 80 Vanadium normalization scan 2t=70`
- `camea2018n000081.hdf`: Scan stopped before first point
- `camea2018n000082.hdf`: `sc ei 3.25 dei -0.005 np 101 mn 100000 continuation of run 61`
- `camea2018n000083.hdf`: Scan stopped before first point
- `camea2018n000084.hdf`: `sc ei 4.25 dei -0.01 np 251 mn 100000 Mono flat 80 V-TAS6`
- `camea2018n000085.hdf`: `sc ei 4.25 dei 0.01 np 251 mn 100000 Mono flat V-TAS6 a4=70`
- `camea2018n000086.hdf`: `sc 2t 77.5 d2t -0.05 np 501 mn 25000 Mono flat Al2O3 2t scan left side`
- `camea2018n000087.hdf`: `sc 2t 77.5 d2t -0.1 np 251 mn 50000 Mono flat Al2O3 2t scan left side`

- `camea2018n000088.hdf`: `sc 2t -32 d2t 0.1 np 441 mn 50000 Mono flat Al2O3 2t scan right side`
- `camea2018n000089.hdf`: `sc a3 25.1 da3 0 np 3 mn 5000 Mono flat Al2O3 2t scan right side`
- `camea2018n000090.hdf`: `sc a3 25.1 da3 0 np 3 mn 5000 Mono flat Al2O3 2t scan right side`
- `camea2018n000091.hdf`: `sc a3 25.1 da3 0 np 3 mn 5000 Mono flat Al2O3 2t scan right side`
- `camea2018n000092.hdf` Scan stopped before first point
- `camea2018n000093.hdf`: `sc a3 25.1 da3 0 np 3 mn 5000 Mono flat Al2O3 2t scan right side`
- `camea2018n000094.hdf`: `sc a3 25.1 da3 0 np 3 mn 5000 Mono flat Al2O3 2t scan right side`
- `camea2018n000095.hdf`: `sc a3 25.1 da3 0 np 3 mn 5000 Mono flat Al2O3 2t scan right side`
- `camea2018n000096.hdf`: `sc a3 25.1 da3 0 np 3 mn 5000 Mono flat Al2O3 2t scan right side`
- `camea2018n000097.hdf`: `sc a3 25.1 da3 0 np 3 mn 20000 Mono flat Al2O3 2t scan right side`
- `camea2018n000098.hdf`: `sc a3 25.1 da3 0 np 3 mn 20000 Mono flat Al2O3 2t scan right side`
- `camea2018n000099.hdf`: `sc a3 25.1 da3 0 np 200 mn 50000 Mono flat Al2O3 2t scan right side`
- `camea2018n000100.hdf`: `sc a3 30 da3 0.2 np 301 mn 100000 Mono flat Al2O3 2t scan right side`
- `camea2018n000101.hdf` Scan stopped before first point
- `camea2018n000102.hdf` Scan stopped before first point
- `camea2018n000103.hdf` Scan stopped before first point
- `camea2018n000104.hdf`: `sc a3 60 da3 0 np 200 mn 100000 Mono flat Al2O3 2t scan right side`
- `camea2018n000105.hdf`: `sc a3 -60 da3 0.5 np 121 mn 2000 Mono flat Al2O3 2t scan right side`
- `camea2018n000106.hdf`: `sc a3 -33.5 da3 0.2 np 11 mn 2000 Mono flat Al2O3 2t scan right side`
- `camea2018n000107.hdf`: `sc a3 -33.5 da3 0.5 np 11 mn 2000 Mono flat Al2O3 2t scan right side`
- `camea2018n000108.hdf` Scan stopped before first point
- `camea2018n000109.hdf`: `sc a3 -45 da3 0.1 np 21 mn 2000 Mono flat Al2O3 2t scan right side`
- `camea2018n000110.hdf`: `sc 2t -32 d2t 0.1 np 441 mn 10000 Mono flat Al2O3 2t scan right side`
- `camea2018n000111.hdf`: `sc a3 -25 da3 0.2 np 61 mn 2000 Two theta scan with PHO plus side`
- `camea2018n000112.hdf`: `sc a3 -25 da3 0.2 np 61 mn 2000 Two theta scan with PHO plus side`
- `camea2018n000113.hdf`: `sc a3 -67 da3 0.2 np 11 mn 5000 Two theta scan with PHO plus side`
- `camea2018n000114.hdf`: `sc 2t 72.5 d2t -0.1 np 151 mn 10000 Two theta scan with PHO plus side`
- `camea2018n000115.hdf`: `sc ei 4.15 dei -0.01 np 251 mn 100000 MV 80 V-TAS6 enrgy scan Be cold`
- `camea2018n000116.hdf` Scan stopped before first point
- `camea2018n000117.hdf`: `sc a1 37.08 da1 0.1 np 11 mn 1000 MV 80 V-TAS6 enrgy scan Be cold`
- `camea2018n000118.hdf`: `sc a1 37.08 da1 0.1 np 11 mn 2000 MV 80 V-TAS6 enrgy scan Be cold`
- `camea2018n000119.hdf`: `sc ei 4.15 dei -0.01 np 251 mn 100000 MV 80 V-TAS6 E scan Be cold after a1 a2 correction`
- `camea2018n000120.hdf`: `sc ei 5 dei -0.02 np 31 mn 20000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000121.hdf`: `sc a3 -67 da3 0.2 np 21 mn 10000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000122.hdf`: `sc a3 -67 da3 0.5 np 21 mn 10000 a1 softzero 0.391 a2 softzero 0.36`



- `camea2018n000123.hdf`: `sc a3 -72 da3 0.5 np 21 mn 10000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000124.hdf`: `sc a3 -72 da3 0.2 np 21 mn 1000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000125.hdf`: `sc a3 -87 da3 0 np 1 ti 1 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000126.hdf`: `sc a3 -87 da3 0.5 np 21 mn 100000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000127.hdf`: `sc a3 -87 da3 -0.5 np 21 mn 100000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000128.hdf` Scan stopped before first point
- `camea2018n000129.hdf`: `sc 2t -20 d2t 0.1 np 11 mn 2000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000130.hdf`: `sc a3 -87.315 da3 0.05 np 21 mn 2000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000131.hdf`: `sc mst 10 dmst 1 np 21 mn 2000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000132.hdf`: `sc msb 10 dmsb 1 np 21 mn 2000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000133.hdf`: `sc msl 10 dmsl 1 np 21 mn 2000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000134.hdf`: `sc msr 10 dmsr 1 np 21 mn 2000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000135.hdf`: `sc a3 0 da3 1 np 61 mn 150000 a1 softzero 0.391 a2 softzero 0.36`
- `camea2018n000136.hdf`: `sc a3 0 da3 0.5 np 121 mn 150000 A3 scan around 1 0 0 YMnO3 T=10, 2T= -20`
- `camea2018n000137.hdf`: `sc a3 0 da3 0.5 np 121 mn 150000 A3 scan around 1 0 0 YMnO3 T=10, 2T= -24`
- `camea2018n000138.hdf`: `sc a3 4 da3 0.2 np 21 mn 5000 A3 scan around 1 0 0 YMnO3 T=10, 2T= -24`
- `camea2018n000139.hdf`: `sc qh 0 -1 0 0 dqh 0 0.025 0 0 np 21 mn 10000 A3 scan around 1 0 0 YMnO3 T=10, 2T= -24`
- `camea2018n000140.hdf` Scan stopped before first point
- `camea2018n000141.hdf`: `sc a3 0 da3 1 np 121 mn 125000 A3 scan around 1 0 0 YMnO3 T=100K, 2T= -40, Ei = 6.8`
- `camea2018n000142.hdf`: `sc a3 0 da3 -1 np 121 mn 125000 A3 scan around 1 0 0 YMnO3 T=100K, 2T= -36, Ei = 6.8`
- `camea2018n000143.hdf`: `sc a3 0 da3 1 np 121 mn 125000 A3 scan around 1 0 0 YMnO3 T=100K, 2T= -40, Ei = 5.25 (el. line)`
- `camea2018n000144.hdf`: `sc a3 0 da3 1 np 21 mn 5000 A3 scan around 1 0 0 YMnO3 T=100K, 2T= -40, Ei = 5.25 (el. line)`
- `camea2018n000145.hdf`: `sc a3 -4 da3 0.5 np 61 mn 75000 YMnO3 T=10 2T=84 Ei=6.8 Resolution for ++ scattering`
- `camea2018n000146.hdf` Scan stopped before first point
- `camea2018n000147.hdf`: `sc a3 26 da3 0.5 np 61 mn 75000 YMnO3 T=10 2T=84 Ei=6.8 Resolution for ++ scattering`
- `camea2018n000148.hdf`: `sc a3 0 da3 0.5 np 61 mn 75000 YMnO3 magnon dispersion 2t=-16 Ei=8.6meV T=10K`
- `camea2018n000149.hdf`: `sc a3 0 da3 0.5 np 61 mn 75000 YMnO3 magnon dispersion 2t=-20 Ei=8.6meV t=10K`
- `camea2018n000150.hdf`: `sc a3 0 da3 0.5 np 61 mn 75000 YMnO3 magnon dispersion 2t=-12 Ei=8.6meV T=10K`
- `camea2018n000151.hdf`: `sc a3 0 da3 0.5 np 61 mn 75000 YMnO3 magnon dispersion 2t=-16 Ei=8.6meV t=10K`
- `camea2018n000152.hdf`: `sc a3 0 da3 -1 np 121 mn 100000 YMnO3 Diffuse 2t=-12 Ei=6.8meV tt=60K`



- `camea2018n000153.hdf`: `sc a3 0 da3 -1 np 121 mn 100000 YMnO3 Diffuse 2t=-16 Ei=6.8meV tt=60K`
- `camea2018n000154.hdf`: `sc a3 0 da3 -1 np 121 mn 100000 YMnO3 Diffuse 2t=-12 Ei=6.8meV tt=60K`
- `camea2018n000155.hdf`: `sc a3 0 da3 -1 np 121 mn 100000 YMnO3 Diffuse 2t=-16 Ei=6.8meV tt=60K`
- `camea2018n000156.hdf`: `sc a3 0 da3 -1 np 121 mn 100000 YMnO3 Diffuse 2t=-46 Ei=6.8meV tt=60K`
- `camea2018n000157.hdf`: `sc a3 0 da3 -1 np 121 mn 100000 YMnO3 Diffuse 2t=-50 Ei=6.8meV tt=60K`
- `camea2018n000158.hdf`: `sc a3 0 da3 -1 np 121 mn 100000 YMnO3 Diffuse 2t=-12 Ei=6.8meV tt=100K`
- `camea2018n000159.hdf`: `sc a3 0 da3 -1 np 121 mn 100000 YMnO3 Diffuse 2t=-16 Ei=6.8meV tt=100K`
- `camea2018n000160.hdf`: `sc a3 34 da3 0 np 3 ti 10 YMnO3 Diffuse 2t=-16 Ei=6.8meV tt=100K`
- `camea2018n000161.hdf`: `sc a3 0 da3 0.5 np 81 mn 75000 YMnO3 T=10K positive side 2t=84`
- `camea2018n000162.hdf`: `sc a3 0 da3 0.5 np 81 mn 75000 YMnO3 T=10K positive side 2t=80`
- `camea2018n000163.hdf`: `sc a3 10 da3 0.5 np 81 mn 75000 YMnO3 T=10K positive side 2t=84 Ei=8.5`
- `camea2018n000164.hdf`: `sc a3 10 da3 0.5 np 81 mn 75000 YMnO3 T=10K positive side 2t=80 Ei=8.5`
- `camea2018n000165.hdf`: `sc a3 20 da3 0.5 np 81 mn 75000 YMnO3 T=10K positive side 2t=76 Ei=10.2`
- `camea2018n000166.hdf`: `sc a3 20 da3 0.5 np 81 mn 75000 YMnO3 T=10K positive side 2t=76 Ei=10.2`
- `camea2018n000167.hdf`: `sc a3 20 da3 0.5 np 81 mn 75000 YMnO3 T=10K positive side 2t=80 Ei=10.2`
- `camea2018n000168.hdf`: `sc a3 27 da3 0.5 np 81 mn 75000 YMnO3 T=10K positive side 2t=76 Ei=11.9`
- `camea2018n000169.hdf`: `sc a3 27 da3 0.5 np 81 mn 75000 YMnO3 T=10K positive side 2t=80 Ei=11.9`
- `camea2018n000178.hdf`: `sc a3 0 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=5.5 2t=-10 HHL plane around 1 1 0`
- `camea2018n000179.hdf`: `sc a3 0 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=5.5 2t=-14 HHL plane around 1 1 0`
- `camea2018n000180.hdf`: `sc a3 0 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=5.5 2t=-50 HHL plane around 1 1 0`
- `camea2018n000181.hdf`: `sc a3 0 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=5.5 2t=-54 HHL plane around 1 1 0`
- `camea2018n000182.hdf`: `sc a3 -10 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=7.1 2t=-10 HHL plane around 1 1 0`
- `camea2018n000183.hdf`: `sc a3 -10 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=7.1 2t=-14 HHL plane around 1 1 0`
- `camea2018n000184.hdf`: `sc a3 -10 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=7.1 2t=-50 HHL plane around 1 1 0`
- `camea2018n000185.hdf`: `sc a3 -10 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=7.1 2t=-54 HHL plane around 1 1 0`
- `camea2018n000186.hdf`: `sc a3 -5 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=6.6 2t=-10 HHL plane around 1 1 0`
- `camea2018n000187.hdf`: `sc a3 -5 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=6.6 2t=-14 HHL plane around 1 1 0`
- `camea2018n000190.hdf`: `sc a3 -5 da3 1 np 181 mn 100000 PbTi T=1.5K Ei=6.6 2t=-54 HHL plane around 1 1 0`
- `camea2018n000191.hdf`: `sc a3 -1 da3 0.1 np 31 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000192.hdf`: `sc sgl -2 dsgl 0.5 np 15 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000193.hdf`: `sc sgu 0 dsgu 0.5 np 15 mn 2000 SeCuO3 hk0 plane Alignment`

- `camea2018n000194.hdf`: `sc sgu 0 dsgu 0.5 np 15 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000195.hdf`: `sc sgl -3 dsgl 0.5 np 15 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000196.hdf`: `sc a3 -1.85 da3 0.1 np 21 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000197.hdf`: `sc a3 69.7 da3 0.1 np 31 mn 50000 SeCuO3 hk0 plane Alignment`
- `camea2018n000198.hdf`: `sc a3 69.7 da3 2.5 np 5 mn 50000 SeCuO3 hk0 plane Alignment`
- `camea2018n000199.hdf`: `sc a3 69.7 da3 5 np 7 mn 50000 SeCuO3 hk0 plane Alignment`
- `camea2018n000200.hdf`: `sc a3 -23 da3 0.5 np 21 mn 5000 SeCuO3 hk0 plane Alignment`
- `camea2018n000201.hdf`: `sc a3 -5 da3 0.25 np 21 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000202.hdf`: `sc a3 -5.2 da3 0.1 np 21 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000203.hdf`: `sc a3 27 da3 0.1 np 21 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000204.hdf`: `sc a3 48 da3 0.25 np 21 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000205.hdf`: `sc a3 47.75 da3 0.1 np 21 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000206.hdf`: `sc a3 120 da3 0.1 np 21 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000207.hdf`: `sc a3 119.7 da3 0.05 np 21 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000208.hdf`: `sc a3 119.7 da3 0.05 np 21 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000209.hdf`: `sc sgu 1 dsgu 0.5 np 7 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000210.hdf`: `sc sgl -2.2 dsgl 0.5 np 7 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000211.hdf`: `sc sgl -2.2 dsgl 0.5 np 7 mn 2000 SeCuO3 hk0 plane Alignment`
- `camea2018n000212.hdf`: `sc a3 115 da3 0.5 np 51 mn 100000 SeCu3 Map of 020 2t=-35 Ei=5.25`
- `camea2018n000213.hdf`: `sc a3 115 da3 0.5 np 51 mn 100000 SeCu3 Map of 020 2t=-31 Ei=5.25`
- `camea2018n000214.hdf`: `sc a3 115 da3 0.5 np 51 mn 100000 SeCu3 Map of 020 2t=-35 Ei=5.25`
- `camea2018n000215.hdf`: `sc a3 120 da3 1 np 41 mn 200000 SeCu3 Map of 020 2t=-40 Ei=5.55`
- `camea2018n000216.hdf`: `sc a3 120 da3 1 np 41 mn 200000 SeCu3 Map of 020 2t=-44 Ei=5.5`
- `camea2018n000217.hdf`: `sc a3 120 da3 1 np 41 mn 200000 SeCu3 Map of 020 2t=-40 Ei=5.55`
- `camea2018n000218.hdf`: `sc a3 120 da3 1 np 41 mn 200000 SeCu3 Map of 020 2t=-44 Ei=5.5`
- `camea2018n000219.hdf`: `sc a3 120 da3 1 np 41 mn 200000 SeCu3 Map of 020 2t=-40 Ei=5.55`
- `camea2018n000220.hdf`: `sc a3 120 da3 1 np 41 mn 200000 SeCu3 Map of 020 2t=-44 Ei=5.5`
- `camea2018n000221.hdf`: `sc a3 112 da3 0.1 np 21 mn 2000 Ni3TeO6 alignment`
- `camea2018n000222.hdf`: `sc a3 62 da3 0.1 np 21 mn 2000 Ni3TeO6 alignment`
- `camea2018n000223.hdf`: `sc a3 2 da3 0.1 np 21 mn 2000 Ni3TeO6 alignment`
- `camea2018n000224.hdf`: `sc sgu 0 dsgu 0.5 np 7 mn 5000 Ni3TeO6 alignment`
- `camea2018n000225.hdf` not correct format
- `camea2018n000226.hdf`: `sc sgl 0 dsgl 0.5 np 7 mn 5000 Ni3TeO6 alignment`
- `camea2018n000227.hdf`: `sc sgl -1 dsgl 0.5 np 9 mn 5000 Ni3TeO6 alignment`
- `camea2018n000228.hdf`: `sc sgu 0 dsgu 0.5 np 9 mn 5000 Ni3TeO6 alignment`
- `camea2018n000229.hdf`: `sc sgu 0 dsgu 0.5 np 9 mn 2000 Ni3TeO6 alignment`

- `camea2018n000230.hdf`: `sc sgl 0 dsgl 0.5 np 9 mn 2000 Ni3TeO6 alignment`
- `camea2018n000231.hdf`: `sc a3 55 da3 0.5 np 181 mn 250000 Ni3TeO6 Ei=5.75 2t=-10 around 0 0 1.5`
- `camea2018n000232.hdf`: `sc a3 55 da3 0.5 np 181 mn 250000 Ni3TeO6 Ei=5.75 2t=-14 around 0 0 1.5`
- `camea2018n000233.hdf`: `sc a3 35 da3 0.5 np 181 mn 250000 Ni3TeO6 Ei=7.35 2t=-10 around 0 0 1.5`
- `camea2018n000234.hdf`: `sc a3 35 da3 0.5 np 181 mn 200000 Ni3TeO6 Ei=7.35 2t=-10 around 0 0 1.5`
- `camea2018n000235.hdf`: `sc a3 35 da3 0.5 np 181 mn 200000 Ni3TeO6 Ei=7.35 2t=-14 around 0 0 1.5`
- `camea2018n000236.hdf`: `sc a3 55 da3 0.5 np 181 mn 150000 Ni3TeO6 Ei=5.5 2t=-10 T=25K around 0 0 1.5`
- `camea2018n000250.hdf`: `sc msl 10 dmsl 1 np 21 mn 2000 Alignment YMnO3`
- `camea2018n000251.hdf`: `sc msr 10 dmsr 1 np 21 mn 2000 Alignment YMnO3`
- `camea2018n000252.hdf`: `sc a3 79.79 da3 0.2 np 31 mn 5000 Alignment YMnO3`
- `camea2018n000253.hdf`: `sc a3 79.79 da3 0.2 np 31 mn 5000 Alignment YMnO3`
- `camea2018n000254.hdf`: `sc a3 79.99 da3 0.2 np 31 mn 5000 Alignment YMnO3`
- `camea2018n000255.hdf`: `sc a3 79.99 da3 0.2 np 31 mn 5000 Alignment YMnO3`
- `camea2018n000256.hdf`: `sc a3 79.99 da3 0.2 np 31 mn 5000 Alignment YMnO3`
- `camea2018n000257.hdf`: `sc a3 79.99 da3 0.2 np 31 mn 5000 Alignment YMnO3`
- `camea2018n000258.hdf`: `sc a3 79.99 da3 0.2 np 31 mn 5000 Alignment YMnO3`
- `camea2018n000259.hdf`: `sc a3 79 da3 1 np 181 mn 100000 YMnO3 inelastics, Ei=5.25 2t=-12 around 1 0 0`
- `camea2018n000260.hdf`: `sc a3 79 da3 1 np 181 mn 100000 YMnO3 inelastics, Ei=5.25 2t=-16 around 1 0 0`
- `camea2018n000261.hdf`: `sc a3 79 da3 1 np 181 mn 100000 YMnO3 inelastics, Ei=5.25 2t=-50 around 1 0 0`
- `camea2018n000262.hdf`: `sc a3 79 da3 1 np 181 mn 100000 YMnO3 inelastics, Ei=5.25 2t=-54 around 1 0 0`
- `camea2018n000263.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=5.25 2t=-12 around 1 0 0 TT=2`
- `camea2018n000264.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=5.25 2t=-16 around 1 0 0 TT=2`
- `camea2018n000265.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=5.25 2t=-50 around 1 0 0 TT=2`
- `camea2018n000266.hdf`: `not correct format`
- `camea2018n000267.hdf`: `sc a3 30.8 da3 0.1 np 21 mn 2000 YMnO3 inelastics, Ei=5.25 2t=-54 around 1 0 0 TT=2`
- `camea2018n000268.hdf`: `sc a3 90 da3 0.5 np 361 mn 10000 YMnO3 elastics, Ei=4.96 2t=-30 around 1 0 0`
- `camea2018n000269.hdf`: `sc a3 90 da3 0.25 np 721 mn 5000 YMnO3 elastics, Ei=4.96 2t=-31 around 1 0 0`
- `camea2018n000270.hdf`: `sc a3 90 da3 0.25 np 721 mn 5000 YMnO3 elastics, Ei=4.96 2t=-35 around 1 0 0`
- `camea2018n000276.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=5.25 2t=-12 around 1 0 0`
- `camea2018n000277.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=5.25 2t=-16 around 1 0 0`
- `camea2018n000278.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=5.25 2t=-50 around 1 0 0`
- `camea2018n000279.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=5.25 2t=-54 around 1 0 0`
- `camea2018n000280.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=6.85 2t=-12 around 1 0 0`

- `camea2018n000281.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=6.85 2t=-12 around 1 0 0`
- `camea2018n000282.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=6.85 2t=-16 around 1 0 0`
- `camea2018n000283.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=6.85 2t=-50 around 1 0 0`
- `camea2018n000284.hdf`: `sc a3 79 da3 1 np 181 mn 80000 YMnO3 inelastics, Ei=6.85 2t=-54 around 1 0 0`
- `camea2018n000285.hdf`: `sc a3 95 da3 1 np 131 mn 80000 YMnO3 inelastics, Ei=6.85 2t=-12 around 1 0 0`
- `camea2018n000286.hdf`: `sc a3 95 da3 1 np 131 mn 80000 YMnO3 inelastics, Ei=6.85 2t=-16 around 1 0 0`
- `camea2018n000287.hdf`: `sc a3 95 da3 1 np 131 mn 80000 YMnO3 inelastics, Ei=6.85 2t=-50 around 1 0 0`
- `camea2018n000288.hdf`: `sc a3 95 da3 1 np 131 mn 80000 YMnO3 inelastics, Ei=6.85 2t=-54 around 1 0 0`
- `camea2018n000289.hdf`: `sc a3 48.5 da3 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000290.hdf`: `sc a3 132 da3 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000291.hdf`: `sc a3 49 da3 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000292.hdf`: `sc a3 48.995 da3 0.05 a4 -42.6642 da4 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000293.hdf`: `sc a3 136 da3 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000294.hdf`: `sc a3 136.30 da3 0.05 a4 -48.032 da4 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000295.hdf`: `sc a3 86.54 da3 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000296.hdf`: `sc a3 85.54 da3 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000297.hdf`: `sc a4 -66.195 da4 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000298.hdf`: `sc a3 48.6625 da3 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000299.hdf`: `sc a3 48.9586 da3 0.05 a4 -42.664 da4 0.1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000300.hdf`: `sc sgl 0 dsgl 0.5 np 7 mn 2000 SrBaCuSiO alignment`
- `camea2018n000301.hdf`: `sc sgl 0 dsgl 0.5 np 11 mn 2000 SrBaCuSiO alignment`
- `camea2018n000302.hdf`: `sc sgu 0 dsgu 0.5 np 11 mn 2000 SrBaCuSiO alignment`
- `camea2018n000303.hdf`: `sc sgl 0 dsgl 0.5 np 11 mn 2000 SrBaCuSiO alignment`
- `camea2018n000304.hdf`: `sc mst 10 dmst 1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000305.hdf`: `sc mst 17 dmst 1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000306.hdf`: `sc msb 15 dmsb 1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000307.hdf`: `sc msl 15 dmsl 1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000308.hdf`: `sc msl 5 dmsl 1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000309.hdf`: `sc msr 10 dmsr 1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000310.hdf`: `sc msl 5 dmsl 1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000311.hdf`: `not correct format`
- `camea2018n000312.hdf`: `sc msr 10 dmsr 1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000313.hdf`: `sc msl 5 dmsl 1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000314.hdf`: `sc msr 10 dmsr 1 np 21 mn 2000 SrBaCuSiO alignment`
- `camea2018n000315.hdf`: `sc a3 40 da3 0.5 np 221 mn 100000 MV80 SrBaCuSiO Ei=7.5 2t=-12 tt=1.5`
- `camea2018n000316.hdf`: `sc a3 40 da3 0.5 np 221 mn 100000 MV80 SrBaCuSiO Ei=7.5 2t=-16 tt=1.5`

- `camea2018n000317.hdf`: `sc a3 30 da3 0.5 np 221 mn 100000 MV80 SrBaCuSiO Ei=9 2t=-12 tt=1.5`
- `camea2018n000318.hdf`: `sc a3 30 da3 0.5 np 221 mn 100000 MV80 SrBaCuSiO Ei=9 2t=-16 tt=1.5`
- `camea2018n000319.hdf`: `sc a3 19 da3 0.5 np 101 mn 100000 MV80 SrBaCuSiO Ei=7.5 2t=-25 tt=1.5`
- `camea2018n000320.hdf`: `sc a3 19 da3 0.5 np 101 mn 100000 MV80 SrBaCuSiO Ei=7.5 2t=-29 tt=1.5`
- 

\* \*

---

**Note:** Python 3.4 is believed to be compatible but is not tested due to updates in testing framework. Python 3.7 is supported by the package but some of the dependencies might not be supported on all operating systems.

---



—  
\_tools, [59](#)

### **a**

Analyser, [33](#)

### **d**

Data, [38](#)

DataFile, [56](#)

DataSet, [48](#)

Detector, [32](#)

### **g**

Geometry, [31](#)

GeometryConcept, [31](#)

### **i**

Instrument, [35](#)

### **s**

Statistics, [37](#)

### **v**

Viewer1D, [58](#)

Viewer3D, [57](#)

### **w**

Wedge, [35](#)





## Symbols

`_tools` (module), 59

## A

`Analyser` (class in *Analyser*), 33

`Analyser` (module), 33

`append()` (*Instrument.Instrument* method), 35

`append()` (*Wedge.Wedge* method), 35

## B

`beautifyArgs()` (in module *\_tools*), 59

`binData3D()` (*DataSet.DataSet* method), 39

`binData3D()` (in module *DataSet*), 48

`binEdges()` (in module *\_tools*), 59

`boundaryQ()` (in module *DataSet*), 48

## C

`calculateDetectorAnalyserPositions()`  
(*Wedge.Wedge* method), 35

`calculateEdgePolygons()` (*DataFile.DataFile*  
method), 56

`calculateGrid3D()` (in module *DataSet*), 48

`convertDataFile()` (*DataSet.DataSet* method), 39

`convertToQxQy()` (*DataSet.DataSet* method), 40

`convertToQxQy()` (in module *DataSet*), 49

`convexHullPoints()` (in module *DataSet*), 49

`createQEAxes()` (*DataSet.DataSet* method), 40

`createQEAxes()` (in module *DataSet*), 49

`createRLUAxes()` (*DataSet.DataSet* method), 40

`createRLUAxes()` (in module *DataSet*), 49

`cut1D()` (*DataSet.DataSet* method), 40

`cut1D()` (in module *DataSet*), 49

`cut1DE()` (*DataSet.DataSet* method), 41

`cut1DE()` (in module *DataSet*), 50

`cutPowder()` (*DataSet.DataSet* method), 41

`cutPowder()` (in module *DataSet*), 50

`cutQE()` (*DataSet.DataSet* method), 42

`cutQE()` (in module *DataSet*), 51

`cutQELine()` (*DataSet.DataSet* method), 42

## D

`Data` (module), 38

`DataFile` (class in *DataFile*), 56

`DataFile` (module), 56

`DataSet` (class in *DataSet*), 38

`DataSet` (module), 38, 48

`Detector` (class in *Detector*), 32

`Detector` (module), 32

`difference()` (*DataFile.DataFile* method), 56

## E

`extractData()` (*DataSet.DataSet* method), 43

## F

`fileListGenerator()` (in module *\_tools*), 60

`FlatAnalyser` (class in *Analyser*), 34

## G

`generateCalibration()` (*Instrument.Instrument*  
method), 36

`generateCAMEAXML()` (*Instrument.Instrument*  
method), 35

`Geometry` (module), 31

`GeometryConcept` (class in *GeometryConcept*), 31

`GeometryConcept` (module), 31

`GeometryObject` (class in *GeometryConcept*), 32

`getPixelPositions()` (*Detector.TubeDetector1D*  
method), 33

## I

`initData()` (*Viewer1D.Viewer1D* method), 59

`initialize()` (*Instrument.Instrument* method), 36

`Instrument` (class in *Instrument*), 35

`Instrument` (module), 35

## K

`KwargChecker()` (in module *\_tools*), 59

## L

`load()` (*GeometryConcept.GeometryConcept method*), 31  
`load()` (*in module DataSet*), 51  
`loadBinning()` (*DataFile.DataFile method*), 56

## M

`my_timer_N()` (*in module \_tools*), 60

## O

`OxfordList()` (*in module DataSet*), 48

## P

`plot()` (*Analyser.Analyser method*), 34  
`plot()` (*Analyser.FlatAnalyser method*), 34  
`plot()` (*Detector.Detector method*), 32  
`plot()` (*Detector.TubeDetector1D method*), 33  
`plot()` (*GeometryConcept.GeometryConcept method*), 31  
`plot()` (*Instrument.Instrument method*), 36  
`plot()` (*Wedge.Wedge method*), 35  
`plotA3A4()` (*DataSet.DataSet method*), 43  
`plotA3A4()` (*in module DataSet*), 51  
`plotA4()` (*DataFile.DataFile method*), 56  
`plotCut1D()` (*DataSet.DataSet method*), 44  
`plotCut1D()` (*in module DataSet*), 52  
`plotCutPowder()` (*DataSet.DataSet method*), 45  
`plotCutPowder()` (*in module DataSet*), 53  
`plotCutQE()` (*DataSet.DataSet method*), 45  
`plotCutQE()` (*in module DataSet*), 54  
`plotCutQELine()` (*DataSet.DataSet method*), 46  
`plotData()` (*Viewer1D.Viewer1D method*), 59  
`plotEf()` (*DataFile.DataFile method*), 56  
`plotEfOverview()` (*DataFile.DataFile method*), 56  
`plotFit()` (*Viewer1D.Viewer1D method*), 59  
`plotNormalization()` (*DataFile.DataFile method*), 56  
`plotQPlane()` (*DataSet.DataSet method*), 47  
`plotQPlane()` (*in module DataSet*), 55  
`position` (*GeometryConcept.GeometryConcept attribute*), 32

## R

`removeFitPlot()` (*Viewer1D.Viewer1D method*), 59

## S

`saveNXsqom()` (*DataFile.DataFile method*), 57  
`saveXML()` (*Instrument.Instrument method*), 36  
`Statistics` (*module*), 37

## T

`TubeDetector1D` (*class in Detector*), 33

## V

`View3D()` (*DataSet.DataSet method*), 38  
`Viewer1D` (*class in Viewer1D*), 58  
`Viewer1D` (*module*), 58  
`Viewer3D` (*class in Viewer3D*), 57  
`Viewer3D` (*module*), 57  
`voronoiTessellation()` (*in module DataSet*), 55

## W

`Wedge` (*class in Wedge*), 35  
`Wedge` (*module*), 35