
GDA Analysis Guide

Release 8.6

Jun Aishima
Mark Basham
Peter Chang
Joachim Diepstraten
Richard Fearn
Matthew Gerring
Paul Gibbons
Karl Levik
Geoff Mant
Vasanthi Nagalingam
Bill Pulford
Eric Ren
Tobias Richter
Duncan Sneddon
Robert Walton
Matthew Webber
Richard Woolliscroft
Fajin Yuan

July 01, 2010

CONTENTS

1	User Guide to the Diamond Scisoft Data Analysis Plugin	3
1.1	How to do stuff with uk.ac.diamond.scisoft.analysis	3
2	Scisoftpy	5
2.1	References	5
3	Dataset	7
3.1	References	8
4	File Input/Output	9
4.1	References	10
5	Plot view	11
5.1	2D Image Plot Profiles	16
5.2	Plot GUI information	18
5.3	ROI objects	18
6	NeXus Tree View	21
6.1	Introduction	21
6.2	Interaction	21
6.3	References	22
7	Indices and tables	23

Contents:

USER GUIDE TO THE DIAMOND SCISOFT DATA ANALYSIS PLUGIN

1.1 How to do stuff with `uk.ac.diamond.scisoft.analysis`

1.1.1 Introduction

Diamond Light Source Ltd (DLS) ¹ is a not-for-profit company jointly owned by the UK government's STFC and Wellcome Trust. Its purpose is to handle all aspects of planning, building and running of a major scientific research facility: a synchrotron light source.

The u.a.d.s.a plugin holds the DLS Scientific Software team's data analysis and visualization package. It is implemented in Java using the Eclipse framework, Jython, jReality and other libraries. The analysis package originally was part of the Generic Data Acquisition (GDA) suite ². With their move to a plugin architecture, the code was split into relatively independent parts.

The basic data analysis was conceived to be driven by Jython, the Java implementation of the Python programming language. The package has been designed as a client/server model with multiple graphical clients which can replicate each other's actions.

The command server can be controlled by Jython commands sent by clients. It, in turn, can direct certain actions of the clients.

1.1.2 Jython

The Jython aspect of the plugin comprises a Jython console and a command line with an editor (taken from PyDev).

Everything from Jython is available on the console. In addition, there are a number of Java classes available to aid data analysis. The following sections details those Java classes as wrapped for Jython.

See the Jython documentation at its website ³.

1.1.3 References

¹ DLS: <http://www.diamond.ac.uk>

² GDA: <http://www.gda.ac.uk>

³ Jython: <http://www.jython.org>

SCISOFTPY

This section documents version 1.0 of the Scisoft analysis plugin or version 8.6 of GDA.

The whole Jython package will emulate behaviour of NumPy¹ together with some I/O and plotting routines. At its core, it contains an ndarray class and a set of subclasses that wrap around the Scisoft generic dataset classes.

The set of utility modules: random, fft, signal, image, plot, roi and io provide random number generators, fast Fourier transforms, signal processing, image processing, plotting, regions of interest, and file loading and saving.

The basic way to start at the Jython console is to enter:

```
import scisoftpy as dnp
```

This imports basic analysis tools into the Jython namespace under *dnp*. There is some help available on the console with the various packages:

```
import scisoftpy as dnp
help(dnp)
help(dnp.random)

import scisoftpy.random as drd
help(drd)
```

2.1 References

¹ NumPy: <http://www.numpy.org>

DATASET

The dataset classes aims to emulate some of the functionality of NumPy's ndarray class.

Capabilities

- Indexing
- Slicing
- Arithmetic operations
- Mathematical and statistical functions

Key differences

- DataSet only holds multi-dimensional array of doubles
- Expandable with reserved space - makes arrays non-contiguous
- No strides
- No broadcasting
- No ellipse
- Incomplete implementation all NumPy's methods

Implemented NumPy methods (1.3)

- Array attributes: shape, ndim, data, size, itemsize, nbytes
- Array methods: copy, fill, reshape, resize, transpose, flatten, squeeze, take, put, max, min, sum, prod, all, any, argmax, argmin
- Array creation: array, zeros, ones, linspace, logspace, arange, diag, diagflat, meshgrid, indices
- Array manipulation: tile, repeat, concatenate, vstack, hstack, dstack, array_split, split, vsplit, hsplit, dsplit
- Array modification: fill, append
- Maths: add, subtract, multiply, divide, negative, power, absolute, exp, log, log2, log10, expm1, log1p, sqrt, square, reciprocal, angle, conjugate, floor_divide, remainder, phase, signum, diff
- Trig: sin, cos, tan, arcsin, arccos, arctan, arctan2, hypot, sinh, cosh, tanh, arcsinh, arccosh, arctanh, deg2rad, rad2deg
- Rounding: rint, ceil, floor
- Stats: amax, amin, ptp, mean, std, var, cumprod, cumsum
- Random: rand, randint, random_integer, randn, exponential, poisson, seed

Non-NumPy

- Array methods: cast, minpos, maxpos, index
- Maths: cbrt

- Stats: rms, skew, kurtosis, median, iqr
- Import/export JAMA ¹

This is an example of DataSet usage:

```
import scisoftpy as dnp

# create a 1D dataset from 0 to 9 of doubles
a = dnp.arange(10)
# make it have 2 rows and 5 columns
a.shape = 2,5
a
# create new dataset with each element raised to power of 5/2
b = dnp.power(a,2.5)
# create new dataset that is sum of two datasets
c = a + b
# modify dataset in-place by dividing each element by corresponding
# element in other dataset
c /= a

# reassign a to a new (integer) dataset of 2 rows and 3 columns
a = dnp.array([[0, 1, 2], [3, 4, 5]])
# create new dataset from a slice of dataset which takes just the 2nd column
d = a[:,1]
# modify dataset and set a slice to a given value
a[1,0:1] = -2

# b is a double dataset (highest type that can contain all entries is used)
b = dnp.array([[ -2, 2.3], [1.2, 9.3]])
# modify dataset and set a slice to the values in another dataset
a[:,1:] = b
# notice the values from b are truncated to integers
a

import Jama.Matrix as Matrix
# make a dataset from a Jama Matrix
m = Matrix([[0,1,1.5],[2,3,3.5]])
dj = dnp.array(m) # directly
da = dnp.array(m.getArray()) # from Java array
db = dnp.array([m.getArray(), m.getArray()]) # from list of arrays

import scisoftpy.random as drd
# create dataset of shape 3,12 of uniform random numbers between 0 and 1
a = drd.rand((3,12))
# take item-wise sine
dnp.sin(a)
# create dataset of shape 3,4 of random integers from 0 to 11 inclusive
drd.randint(12, size=(3,4))
```

3.1 References

¹ JAMA: <http://math.nist.gov/javanumerics/jama/>

FILE INPUT/OUTPUT

The primary package is `scisoftpy.io`.

Images and recorded data from various detectors can be read into the data analysis plugin. This is done using the following method:

```
import scisoftpy.io as dio
dl = dio.load("datafile")
```

This loads up the file as a list of datasets. Optionally arguments can be specified:

```
dl = dio.load(name, formats=None, asdict=False, withmetadata=False)
```

where:

name is a file name

format is a list of strings that specify the formats to try

- `png, gif, jpeg, tiff` - standard image formats
- `adsc` - ADSC Quantum area series detector format
- `crysalis` - Oxford Diffraction CrysAlis processing software format
- `mar` - Rayonix's MarCCD detector
- `pilatus` - Dectris Pilatus detector version of TIFF
- `cbf` - Crystallographic Binary Format (IUCR)
- `xmap` - XIA's DXP-xMAP format for x-ray spectra
- `srs` - Daresbury Laboratory's Synchrotron Radiation Source format
- `binary` - raw single dataset format

or `None` (default) to attempt all above formats

***asdict* is a boolean value to dictate whether to return datasets in a dictionary or list (default)**

withmetadata is a boolean value to dictate whether to return metadata in addition to datasets

The metadata is present as a dictionary with keys of strings. Currently, the metadata items follow a Nexus naming convention.

The four standard image loaders will load and convert RGB images to grey-scale (luma).

Datasets can be saved:

```
dio.save(name, data, format=None, range=(), autoscale=False)
```

where:

name is a file name

data is a dataset or sequence of datasets

format is one of following strings

- `png`, `'gif'`, `jpeg`, `tiff` - standard image formats
- `text` - raw ASCII output
- `binary` - raw binary dump

or `None` (default) to guess format from file name extension

***range* is a tuple for minimum and maximum values for clipping a dataset** before saving

***autoscale* is a boolean value to dictate whether to scale automatically** dataset values to fit the chosen format (only `png` and `jpeg` are supported for auto-scaling).

If there are multiple datasets specified then multiple images will be saved, suffixed with a number representing the number of the dataset in the sequence.

In some of the formats supported (CBF, MarCCD, ADSC), information is supplied on the position and orientation of the detector that took the image, the size of pixels, the position and wavelength of an illuminating beam in a diffraction experiment. This diffraction data can be loaded with:

```
di = dio.loaddiffimage(name, asdict=False)
```

which returns a list (or dictionary) of diffraction images. These images can be plotted using the plot package.

A NeXus¹ file can be loaded with:

```
nt = dio.loadnexus(name)
```

which creates a NeXus tree.

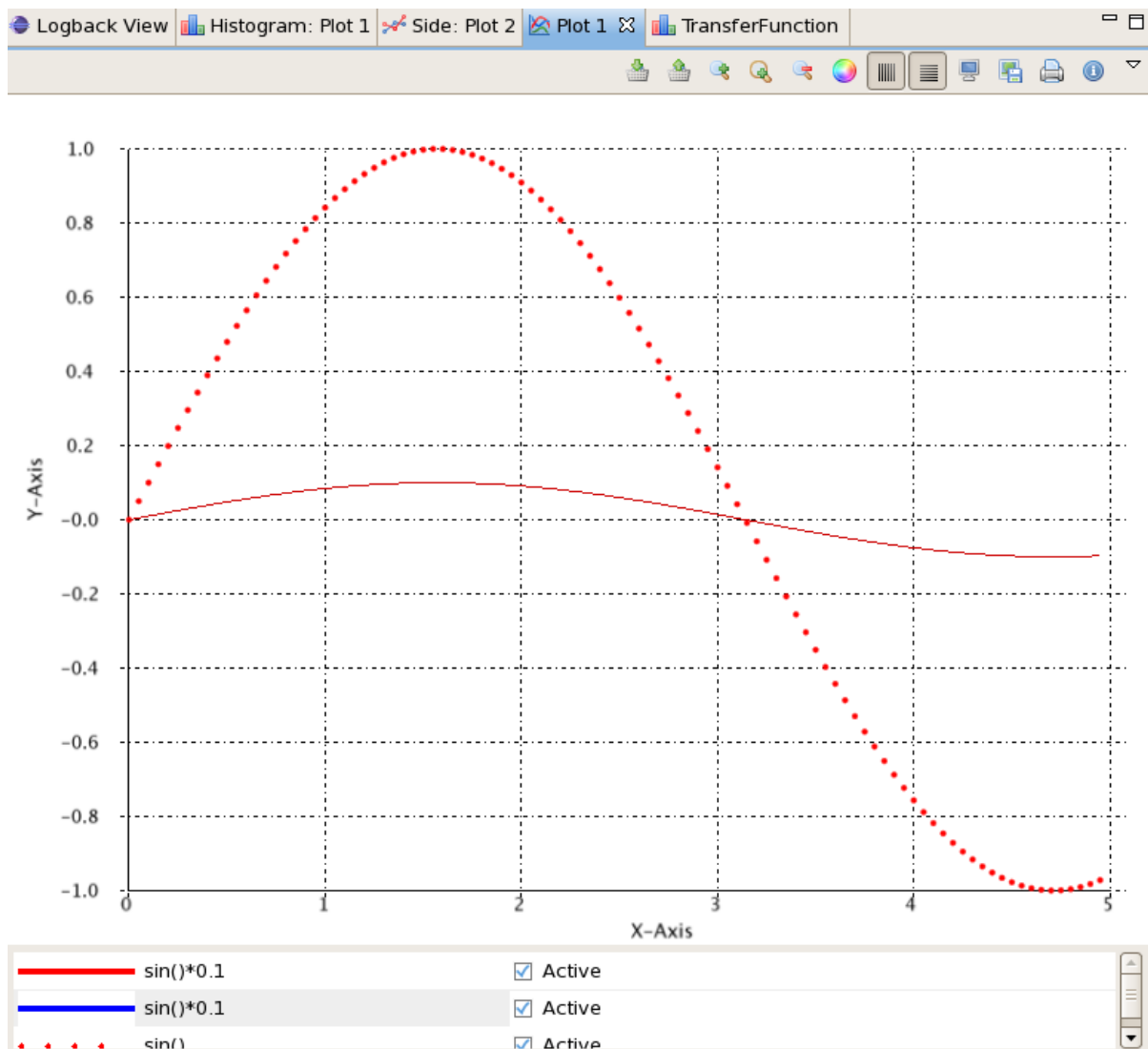
4.1 References

¹ NeXus: <http://www.nexusformat.org>

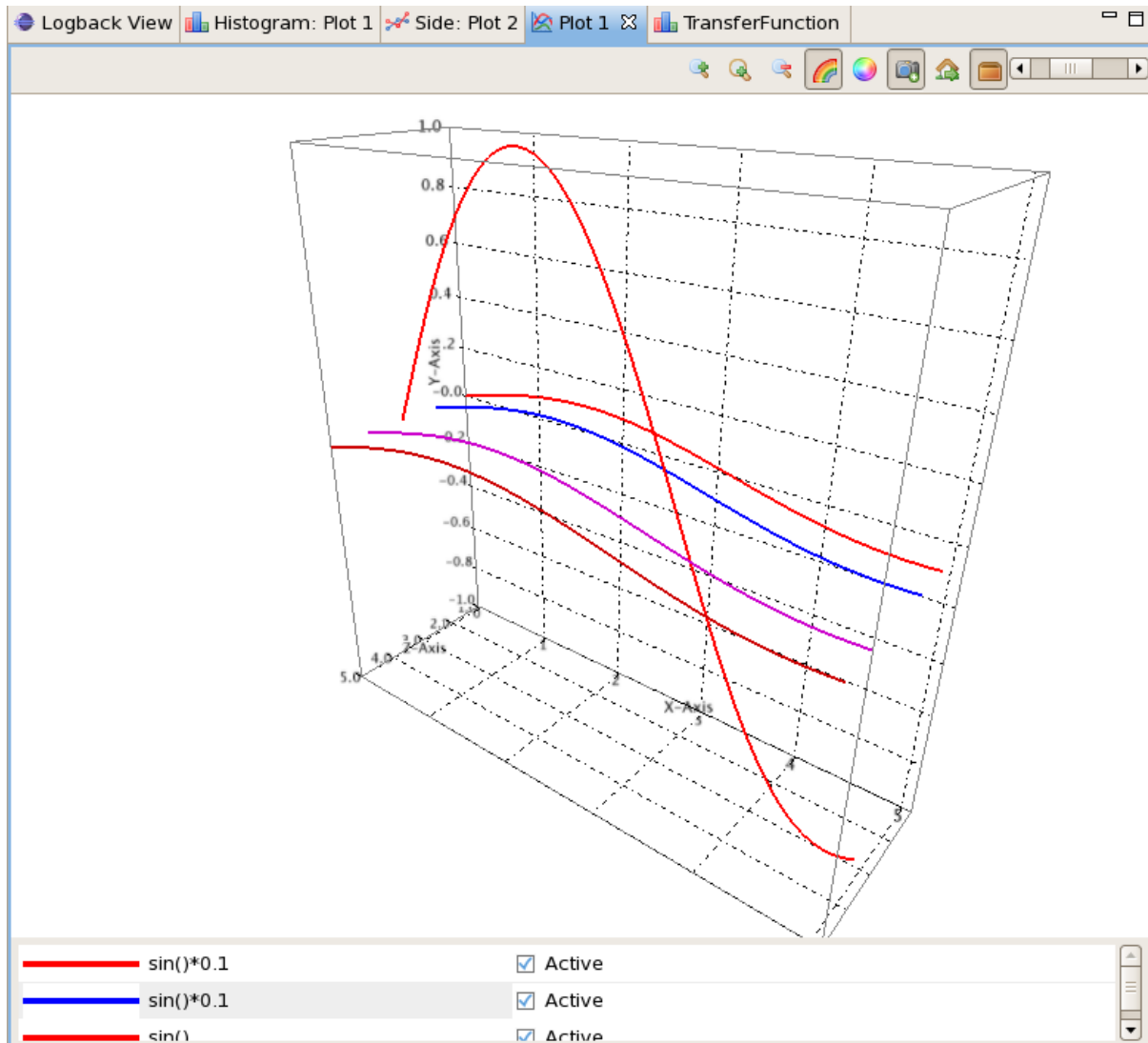
PLOT VIEW

The plot view is the main window where all graphical plotting is displayed. A plot view is a generic plotting UI, that allows graphical plotting of different scalar dataset types. Currently supported scalar plots are:

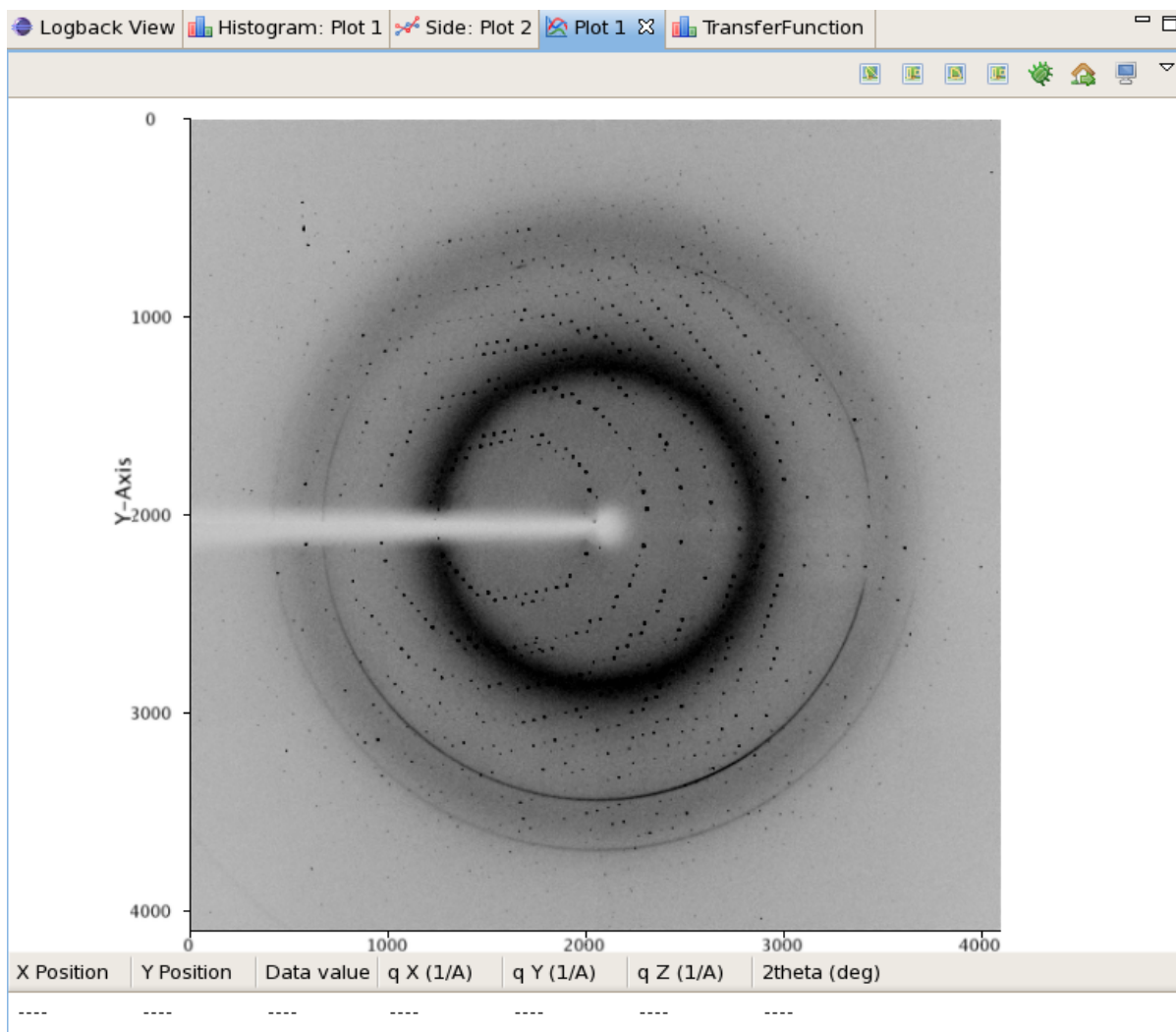
- multiple 1D scalar



- multiple 1D scalar as a series in 3D



- 2D scalar as image



- 2D scalar as 3D surface plot

It is possible to have more than one plot view open and plot to them simultaneously and usually they are named Plot 1, Plot 2, ..., Plot n. The name is important since it is used to send data to via the Jython terminal.

Plotting any data in any form to one of the plot views can be done using the plotting package:

- 1D scalar plots:

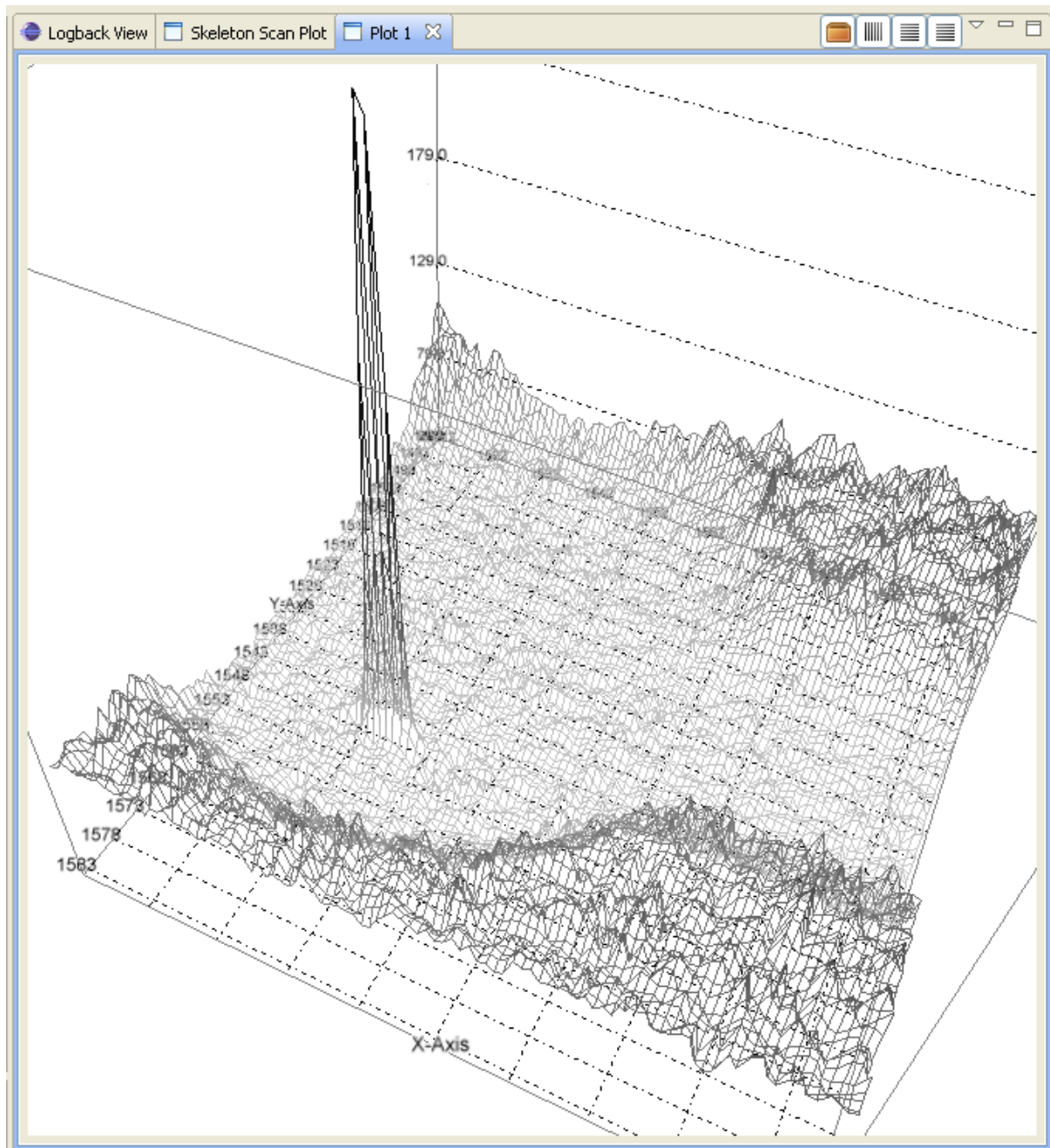
```
import scisoftpy.plot as dpl
dpl.plot([xAxis,] yAxes)
```

plots the given yAxes dataset (or list of datasets) against a xAxis dataset (if given)

- multiple 1D scalar plots as 3D series:

```
dpl.stack([xAxes,] yAxes, zAxis=None)
```

plots all of the given 1D yAxes datasets against corresponding xAxes (if given) as a 3D stack with specified z coordinates



- 2D scalar image plots:

```
dpl.image(image, xAxis=None, yAxis=None)
```

plots the 2D dataset as an image

- 2D scalar 3D surface plots:

```
dpl.surface(data, xAxis=None, yAxis=None)
```

plots the 2D dataset as a surface

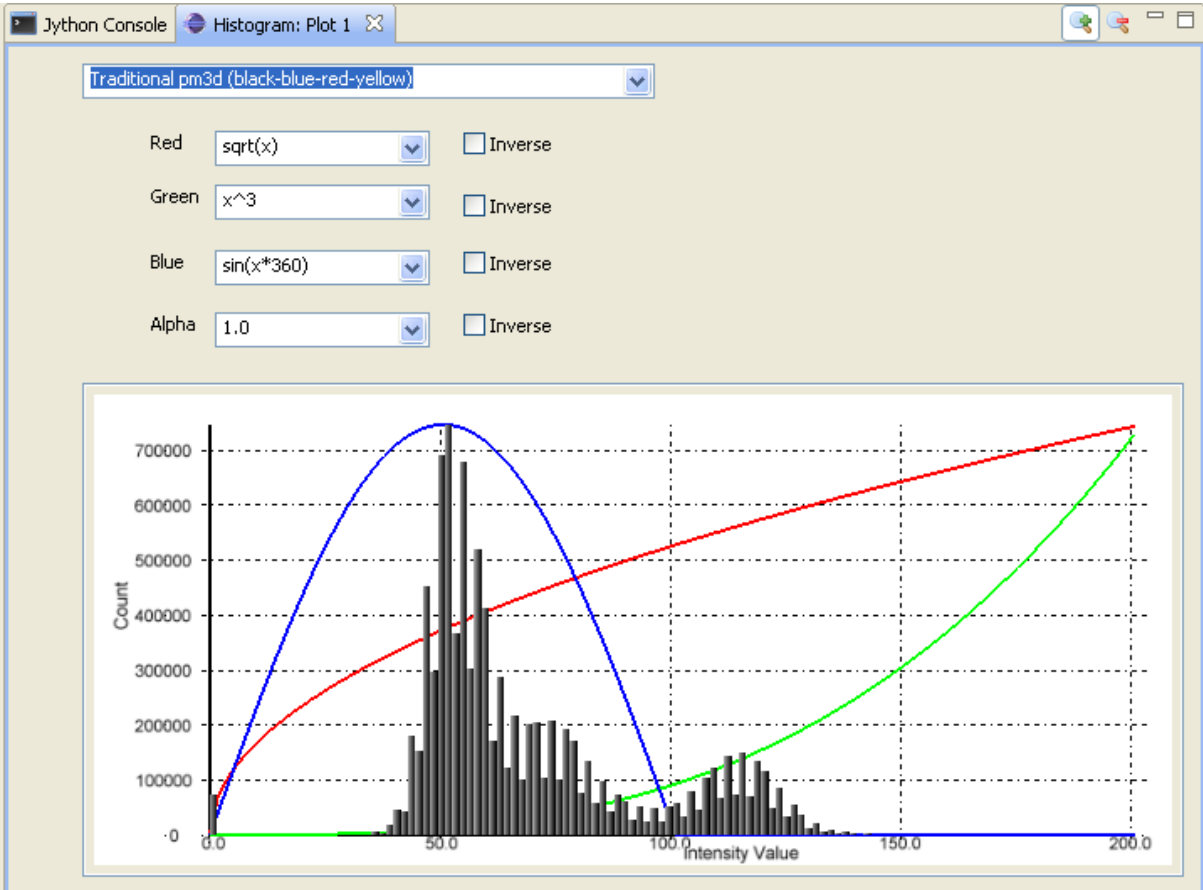
By default, these functions sent data to Plot 1. This default can be changed using:

```
dpl.setdefname('Plot 2')
```

Otherwise, data can be sent to other plot views on a plot-by-plot basis using the optional keyword argument, name. For example:

```
dpl.plot(yAxis, name="Plot 2")
```

Both 2D image plots and 2D surface plots will open automatically a histogram view panel that is associated to the plot view. Through the histogram view it is possible to control the mapping of the data values in the plotted image or surface to the different colours.



5.1 2D Image Plot Profiles

The plot profile tools inhabit a side plot panel. The tools are activated by clicking on the toolbar buttons in the plot view. These buttons become visible when an image is plotted.

The coordinate system used in the image plot is in pixels starting from the upper left at (0,0) with x increasing when moving left and y increasing moving down. Angles are measured from the horizontal and increases when moving clockwise.

There are three profile tools: line, box and sector tools. Each allows the selection of multiple regions of interest (ROIs). The purpose of the ROIs is to allow profiles of the image within a ROI to be plotted. These plots reside in the top part of the panel.

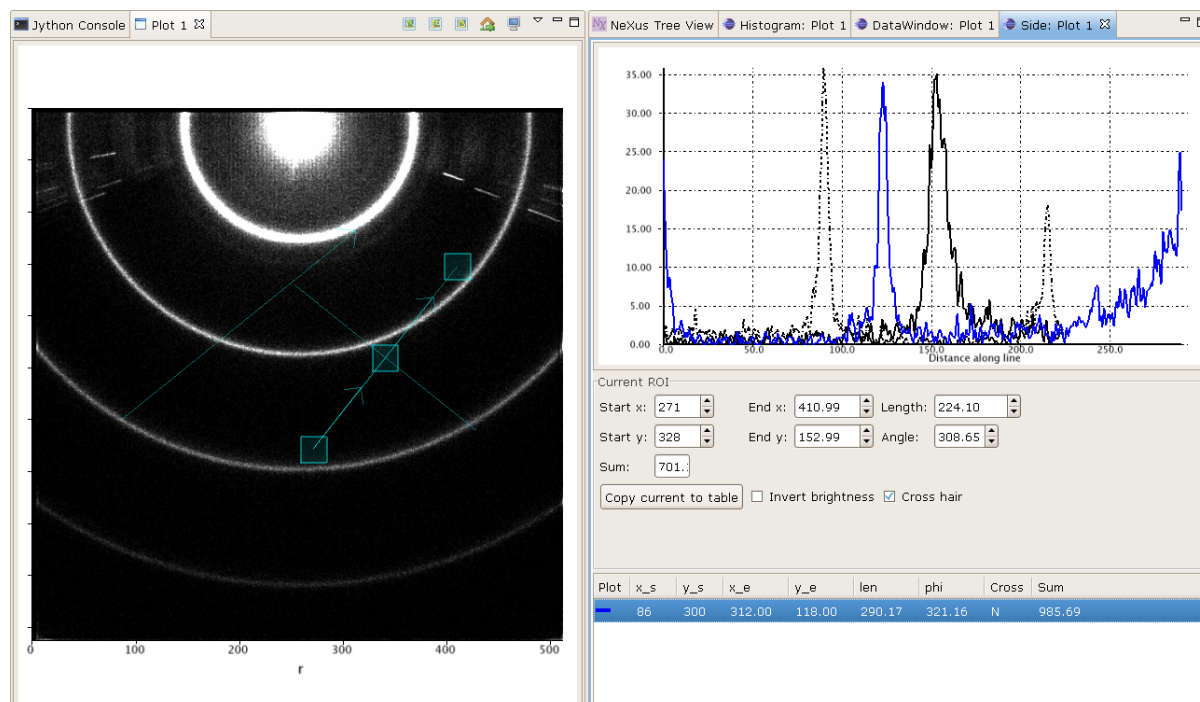


Figure 5.1: Line profile tool

When a profile tool is active, a region of interest can be specified using the mouse to click and drag out a ROI. The ROI is shown as an overlay on the image. Once done, the ROI can be further manipulated with use of its handle areas. The brightness of the ROI outline can be inverted using the “Invert brightness” checkbox to improve its contrast with the image.

The handle areas operate in two ways: a left click on an area enables that area, and the part of the ROI to which it is attached, to be moved; a right click (or alternatively, simultaneous holding a shift key and left clicking) cause some type of rotation to occur. Generally, a central handle area allows translation of the ROI or rotation about that handle area. A handle area at a vertex will allow resize of the ROI (leaving the opposing vertex fixed) or rotate about the opposite vertex.

Once a profile is plotted, it can be added to a store using a toolbar button above the plotting area. The oldest item in the store also can be removed using a toolbar button. There are separate stores for each type of profile.

Each linear ROI can have an optional cross, linear ROI defined to form a cross-hair. This cross ROI is a perpendicular bisector of the same length as its partner. The line profile is plotted in the graph and dashed lines are used for cross ROIs.

A rectangular ROI defined in the box profile tool is defined by its starting point, width (major axis length), height (minor axis length) and orientation angle of its major axis. The upper graph shows the integration values over the minor axis as the position on the major axis is varied. The lower graph shows the converse. There is a “clipping comp” checkbox available that attempts to compensate for the situation where a ROI lies partially

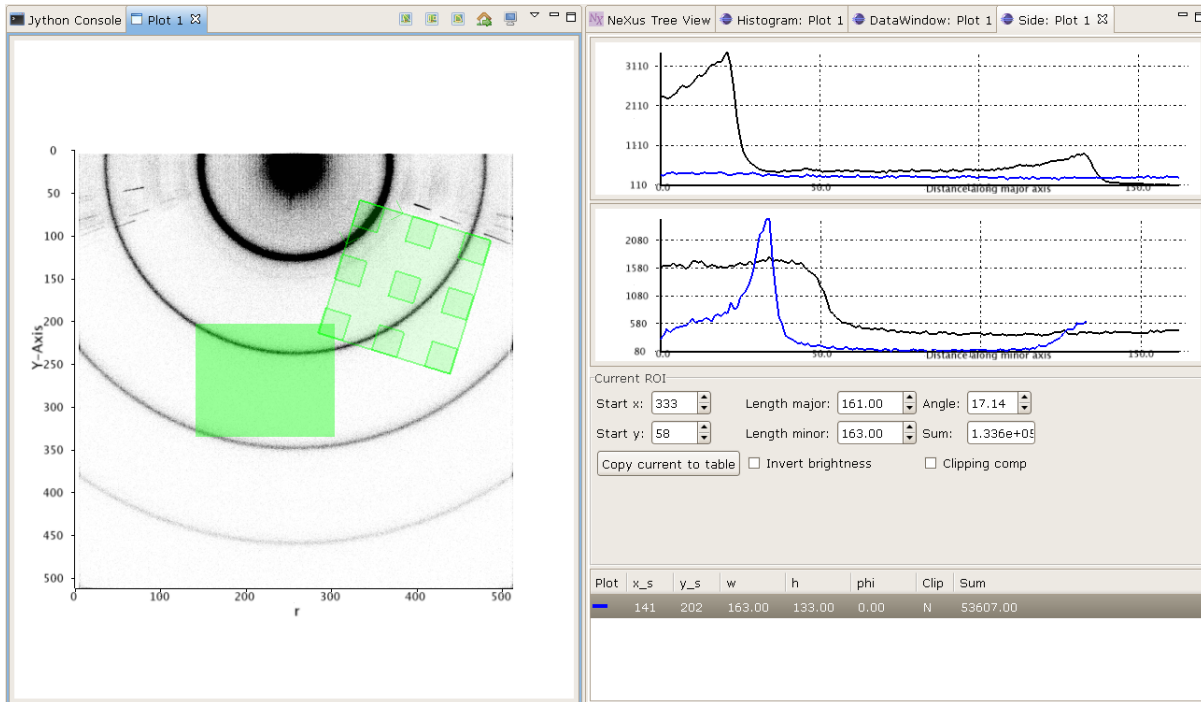


Figure 5.2: Box profile tool

outside the image, i.e. the ROI is clipped by the boundaries of the image. In this case, some of the integration values are subdued by the lack of pixels (they are represented by zeros in the ROI) outside the image and the compensation scheme boosts those values by the ratio of the full integration length to the clipped length. Note that this compensation can introduce extrapolation errors and is prone to erroneous results where the clipped length is short and when the pixel values are noisy.

The sector ROI is distinguished by the necessity of defining a centre point. Once defined, the sector ROI operates in a manner dictated by a polar coordinate system (radius r , angle ϕ) so rotation operations on the handle areas act like translations in polar coordinates. Also, the angular symmetry can be selected for a sector ROI that can alter the ROI or make a copy subject selected symmetry:

- None** No symmetry
- Full** 360 degrees
- L/R reflect** Left/right reflection
- U/D reflect** Up/down reflection
- +90** Rotate 90 degrees clockwise
- 90** Rotate 90 degrees anti-clockwise
- Invert** Invert through centre

The upper graph shows the azimuthal integration as the radius is varied and the lower graph shows the radial integration as the azimuth angle is changed. Ticking the “combine symmetry” checkbox allows any separate symmetry-selected ROI to be combined in the profile plots, otherwise the separate ROI is plotted as dashed lines.

The current ROI can also be modified using the spinner widgets that are displayed in the centre part of the side plot panel. Each spinner is editable and can alter a parameter of the ROI. Once the ROI has been defined, it can be saved and then displayed in the table at the bottom of the panel.

Multiple ROIs can have their profiles plotted by clicking on the checkboxes in the table. Any ROI in the table can be selected and replace the current ROI, copied in place of the current ROI or deleted using a right mouse click anywhere on the row of the ROI.

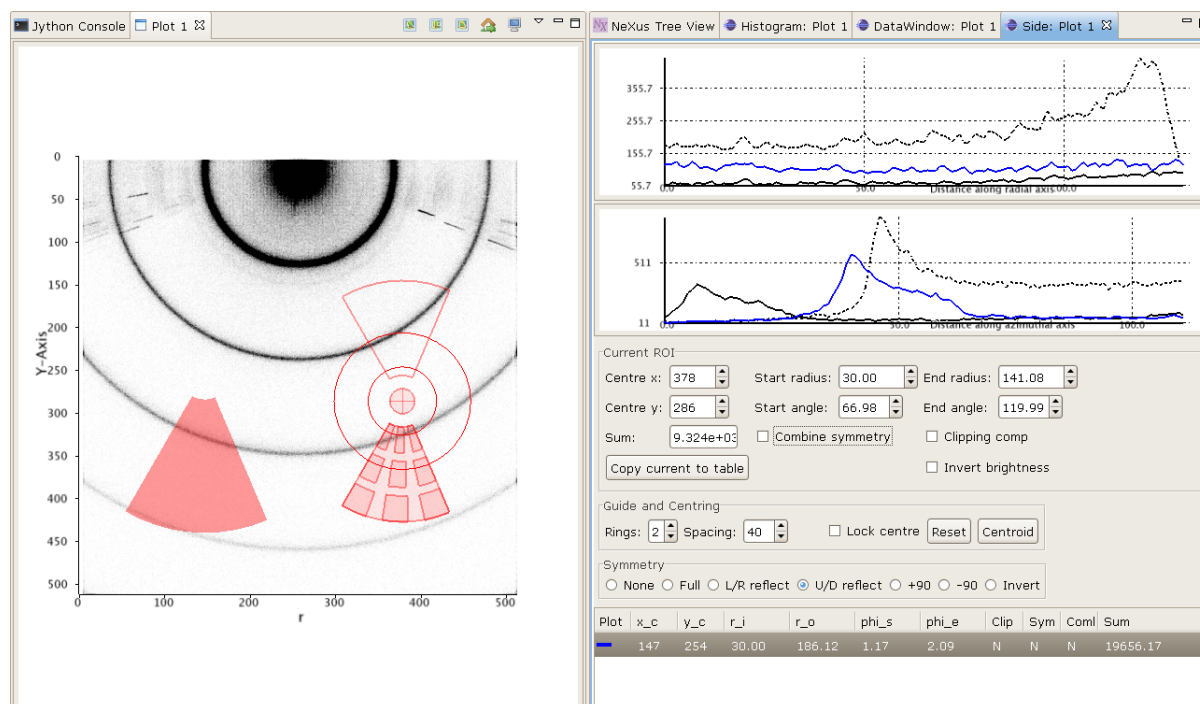


Figure 5.3: Sector profile tool

5.2 Plot GUI information

GUI information from interactions with the plot view and side panels can be passed back and forth from the view to the Jython console.

The plot client regularly updates the console with GUI information. This can be obtained using the plotting package:

```
import scisoftpy.plot as dpl
```

```
# grab a GUI bean
gb = dpl.getbean()
```

By default, this function returns information from Plot 1 - use the keyword argument `name` to obtain information from other named plot views. Again, the default view name can be changed with `dpl.setdefname`. The GUI bean is a dictionary object with a set of possible keys listed in the GUI parameters class. None is returned if there is no dictionary present. You can add in new entries or overwrite existing ones. Modified GUI beans can be pushed back to a plot view:

```
dpl.setbean(gb)
```

and the view will respond appropriately to the updated GUI information. The keys for the dictionary are listed as strings in the GUI parameters class:

```
dir(dpl.parameters)
```

5.3 ROI objects

The regions of interest defined are in the ROI package:

```
import scisoftpy.roi as droi
```

These are

line A line segment defined by its starting point, length and angle

rect A rectangle defined by its starting point, width, height and angle

sect A sector defined by its centre point, bounds on radius and azimuthal angle

As mentioned in the previous section, the current ROI and any ROIs stored in the table are sent via a GUI bean back to the plot view.

The current ROI is held in the GUI bean under the key `parameters.roi` and the table of ROIs under the key `parameters.roilist`. The values held under those keys depend on which side panel is active.

When the line profile tool is being used, the `parameters.roi` item is a linear ROI object and any stored ROIs are held in a Jython list of linear ROIs:

```
cr = gb[dpl.parameters.roi]

# or use convenience function
cr = dpl.getroi(gb)

# print current ROI's starting point, length and angle (in radians)
print cr.point, cr.length, cr.angle

lr = gb[dpl.parameters.roilist]

# or use convenience function
lr = dpl.getrois(gb)

# get first item
ra = lr[0]

print ra.length, ra.angleDegrees

# copy ROI from list
roi = gb[dpl.parameters.roilist][0].copy()

# or use convenience function
roi = dpl.getrois(gb)[0].copy()

# modify ROI
roi.setPoint(100,50)

# import region of interest package
import scisoftpy.roi as droi
list = droi.linelist()
list.add(roi)
gb[dpl.parameters.roilist] = list

# or use convenience function
dpl.setrois(gb, list)

# push bean back
dpl.setbean(gb)
```

The ROIs obtained from the client can be used with image datasets to calculate profile datasets in the console:

```
# for a linear ROI lroi, image dataset and a step size of 0.5 pixels,
# lprof is a list of datasets. The first element is the profile along the
# line and the second element is along the perpendicular bisector (if the
```

```
# crosshair option is set)
lprof = droi.profile(image, lroi, step=0.5)
```

NEXUS TREE VIEW

6.1 Introduction

The NeXus¹ file format is a common data storage format for neutron, x-ray and muon science.

This view allows the data held in a Nexus file to be explored with a graphical user interface and visualized with a simple set of plotting tools in a Plot view. A selected data item can be shown in a plot that has fewer dimensions than the item by choosing which data dimensions to use for plot axes.

6.2 Interaction

NeXus files can be loaded into the viewer by using the Jython console or by clicking on the toolbar button at the top right of the viewer.

To load and view a NeXus file:

```
import scisoftpy.io as dio
nt = dio.loadnexus("/path/to/file.nxs")

import scisoftpy.plot as dpl
dpl.viewnexus(nt)
```

where the tree is sent to a Nexus viewer called “nexusTreeViewer”. To use a viewer with another name, use the optional keyword argument `name`.

The table-tree representing the Nexus structure can be expanded node by node using a left mouse click on the node. The columns of the table-tree display the node name, class, value type, dimensions, value. Right clicking on the table header will bring up a context menu that allows columns to be hidden or made visible.

To select an item to plot, double click on a node that belongs to the NXdata. This will plot the data item if it has a signal attribute. Otherwise, double click on any item of class SDS (scientific data set) to plot that item alone.

The selected data item will have its name shown in the part of the panel below the table-tree at the left hand side. This is the axes selection panel and allows a choice of any (compatible) axis data item or an automatically configured axis to be selected for each dimension in the data.

Once the axes are chosen, the user then moves on to the right hand panel to configure a plot. There are a set of four tabs: one for each type of plot available. Within a tab, the plot axes can be selected from the drop-down combination boxes. These boxes allow different dimensions of the data to be used as plot axes.

Below the drop-down boxes in a plot tab, a set of sliders allows any remaining dimensions of the data (not chosen to act as plot axes) to have their index chosen. These sliders allow the user to visualize slices through their data.

¹ NeXus: <http://www.nexusformat.org>

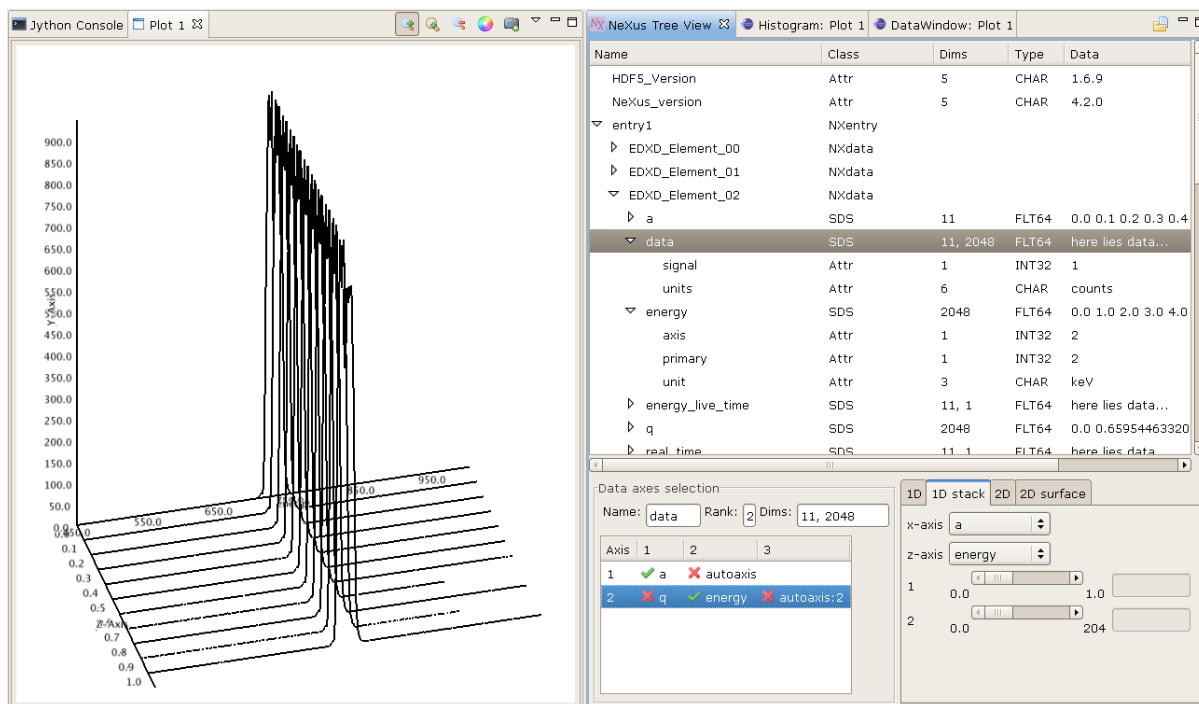


Figure 6.1: NeXus tree viewer

6.3 References

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*