User Guide to the Diamond Scisoft Data Analysis Plugin Documentation

Release 1.0

Diamond Scientific Software Team

October 27, 2010

CONTENTS

Contents:

CHAPTER

ONE

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

CHAPTER

TWO

SCISOFTPY

This section documents version 1.0 of the Scisoft analysis plugin or version 8.8 of GDA.

The whole Jython package will emulate behaviour of NumPy [1] 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`, `fit`, `signal`, `image`, `plot`, `roi` and `io` provide random number generators, function fitting, 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

CHAPTER

THREE

DATASET

The dataset classes aim to emulate some of the functionality of NumPy's `ndarray` class. In addition, datasets can have items which contain multiple elements - these are known as compound datasets.

3.1 Generic dataset

Capabilities

- Indexing
- Slicing
- Boolean dataset selection
- Flat index integer dataset selection
- Arithmetic operations
- Comparison and logic 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 except from single items
- No ellipse
- Incomplete implementation all NumPy's methods

Implemented NumPy methods (1.3)

- Array attributes: shape, ndim, data, size, itemsize, nbytes, T, view, indices
- 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

- Array comparisons and logic operations: all, any, greater, greater_equal, less, less_equal, equal, not_equal, logical_not, logical_and, logical_or, logical_xor, allclose, nonzero, select, where
- Maths: add, subtract, multiply, divide, negative, power, absolute, exp, log, log2, log10, expm1, log1p, sqrt, square, reciprocal, angle, conjugate, floor_divide, reminder, 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
```

[1] JAMA: http://math.nist.gov/javanumerics/jama/

```
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))

# comparison and logic
# create boolean datasets where items are true where condition applies
c = a >= 0.3
d = a < 0.2
# check if all or any of items in these are true
all(c)
any(c)
all(d)
all(d)
# how many were true
c.sum()
d.sum()
# flatten 1D dataset of items which were >= 0.3
a[c]
# assign value where items are < 0.4
a[a < 0.4] = 0

#
```

3.2 RGB dataset

When a colour image is loaded (as described in the next chapter), a RGB dataset is created. This type of dataset has items which are tuples of three 16-bit integers. Each integer represents a value of one of the colour channels. The channels are ordered as red, green and blue. There are four extra attributes to an RGB dataset, `red`, `green`, `blue` and `grey` which retrieve copies of the colour channel or a weighted mixture of channels in the grey case.

There are also four extra methods:

```
get_red(self, dtype=None)
get_green(self, dtype=None)
get_blue(self, dtype=None)
get_grey(self, cweights=None, dtype=None)
```

where `dtype` is an optional dataset type (default is `int16`) and `cweights` is an optional set of weight for combining the colour channel. The default weights are (0.299, 0.587, 0.114) which correspond to the NTSC formula for convert RGB to luma values.

3.3 References

CHAPTER

FOUR

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, ascolour=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

ascolour is a boolean value to dictate whether to return any coloured image data as an RGB dataset or convert it to greyscale

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 in exactly the same way:

```
di = dio.load(name)
```

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

A NeXus [1] file can be loaded with:

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

which creates a NeXus tree.

4.1 References

[1] NeXus: http://www.nexusformat.org

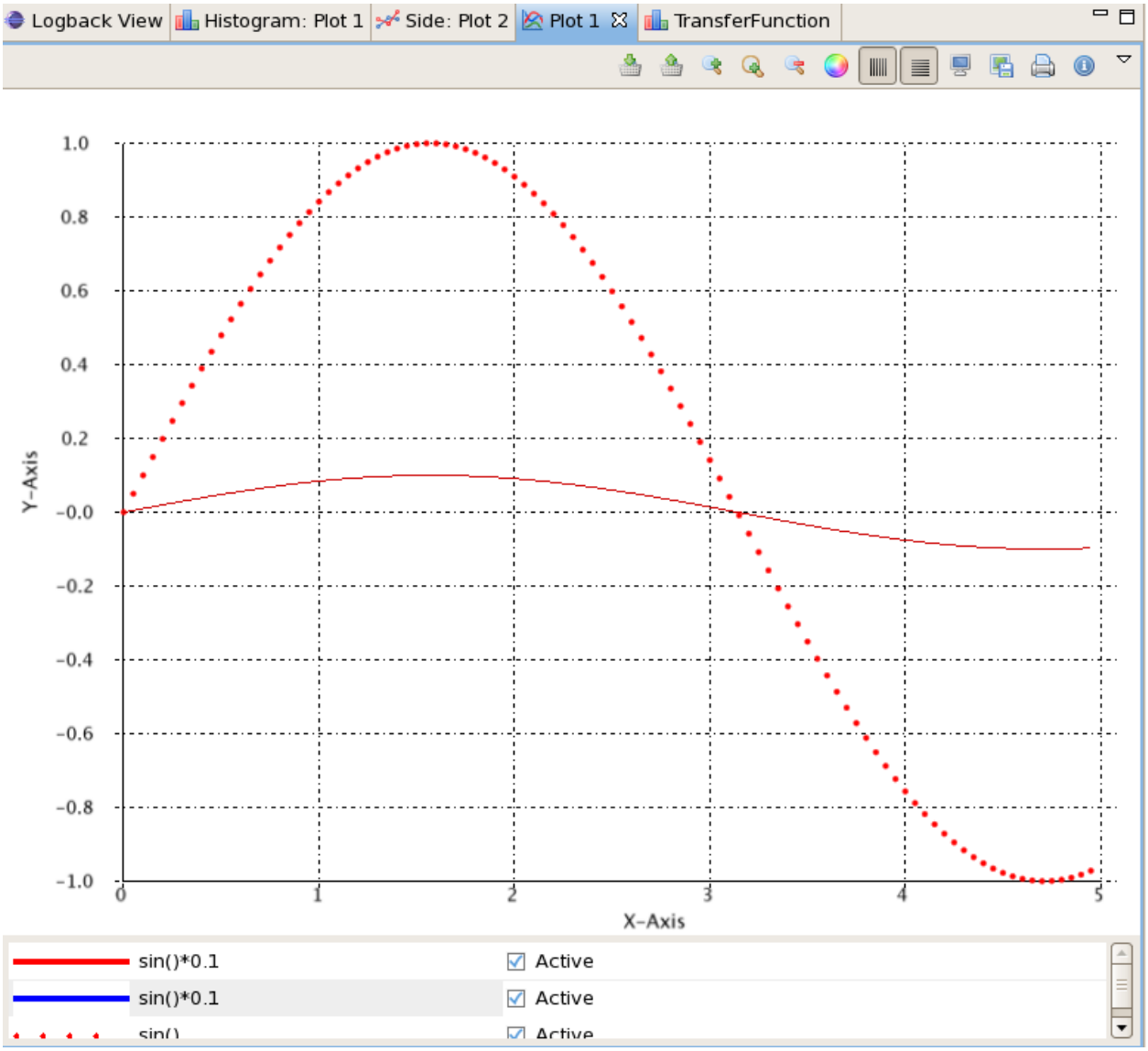
CHAPTER

FIVE

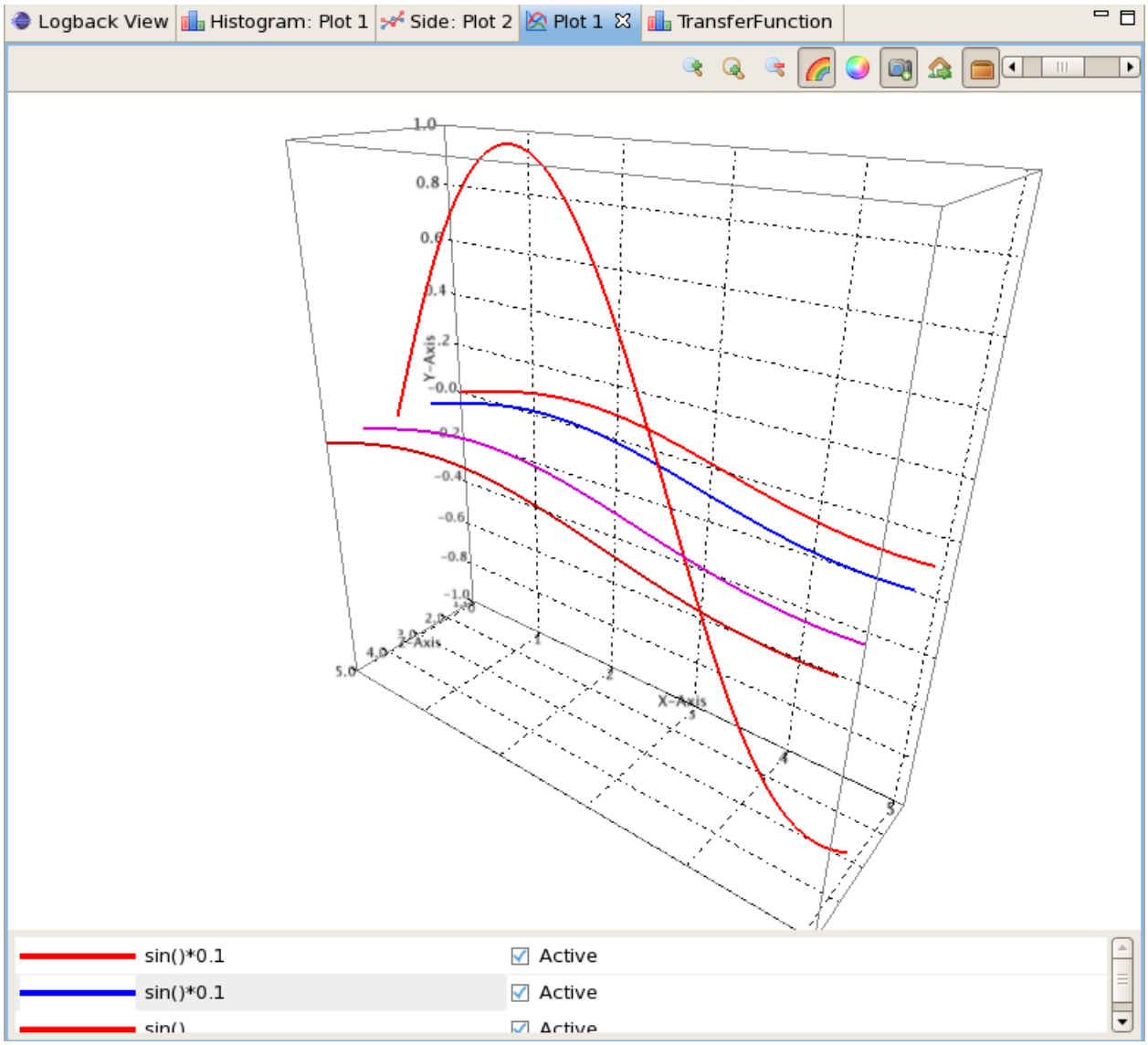
PLOT VIEW

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:

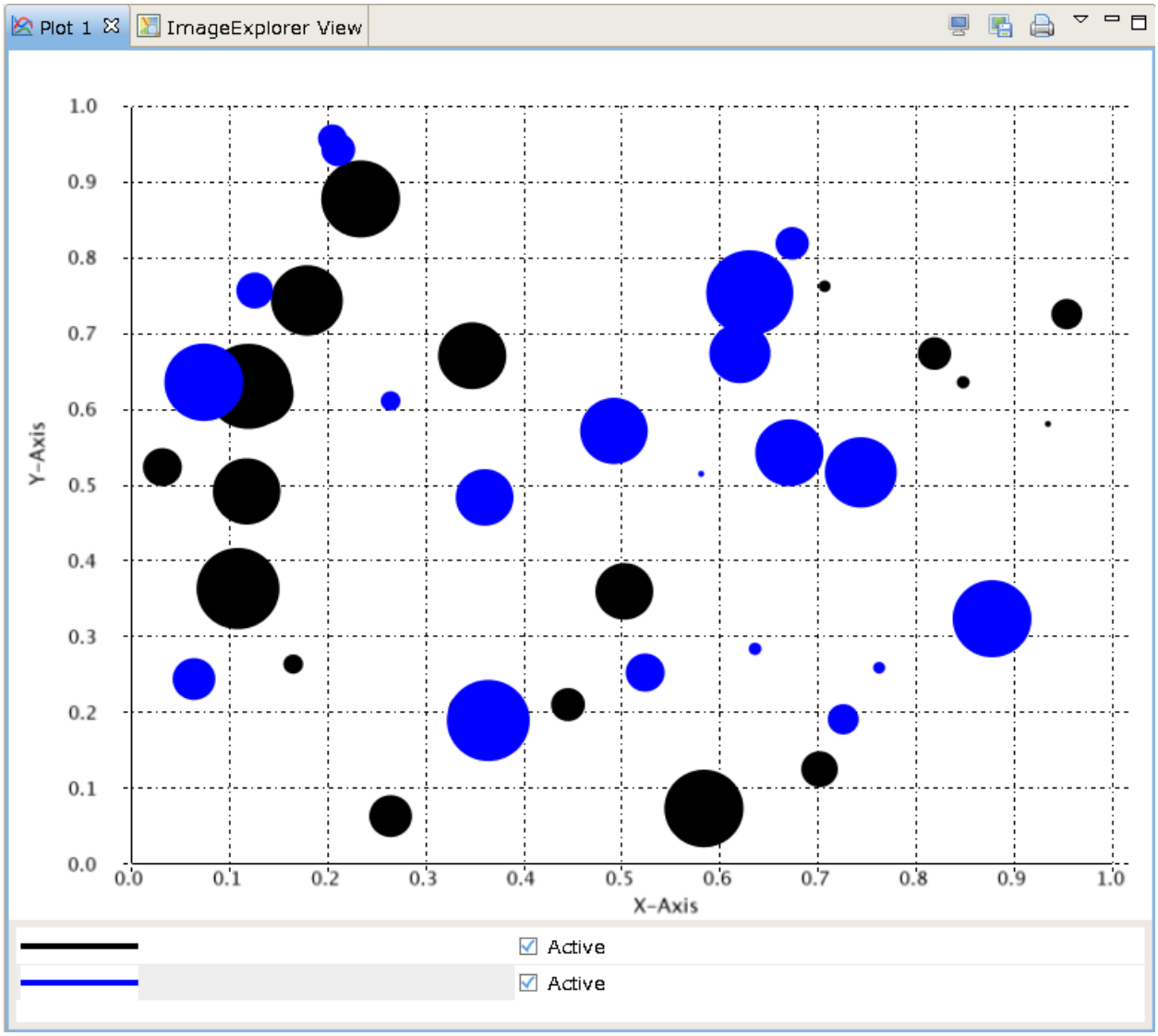
- multiple 1D scalar as lines

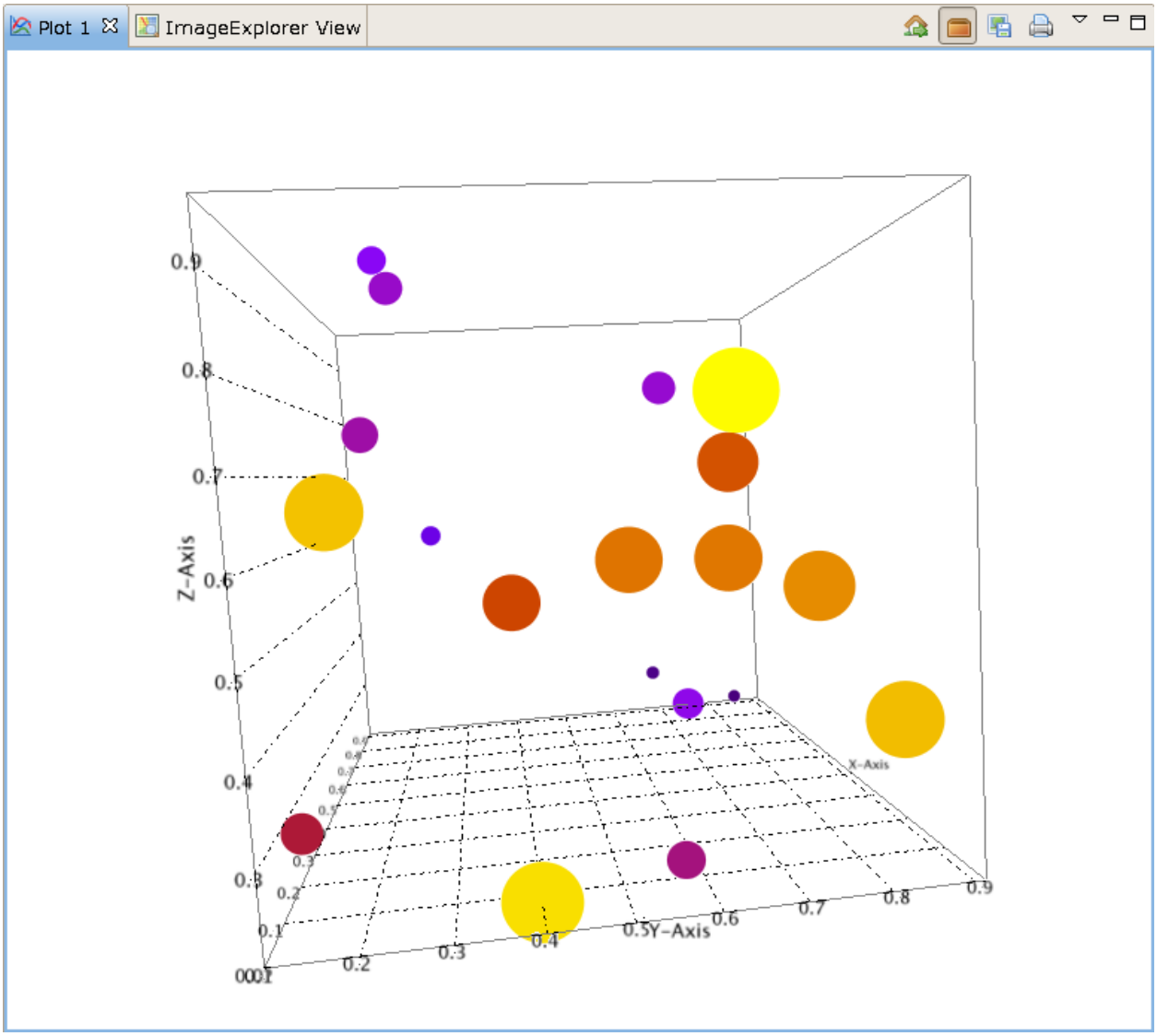


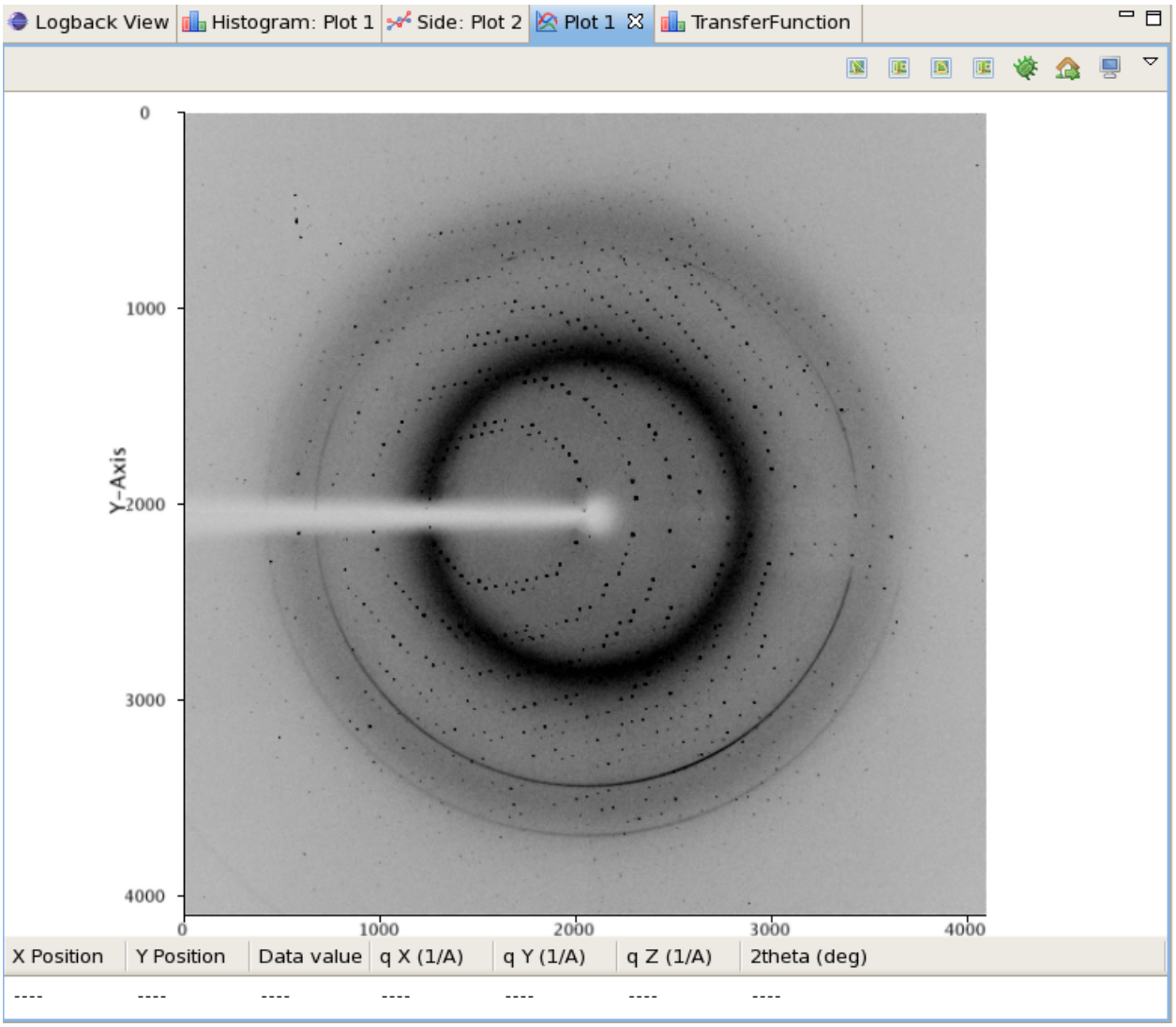
- multiple 1D scalar as a series of lines in 3D

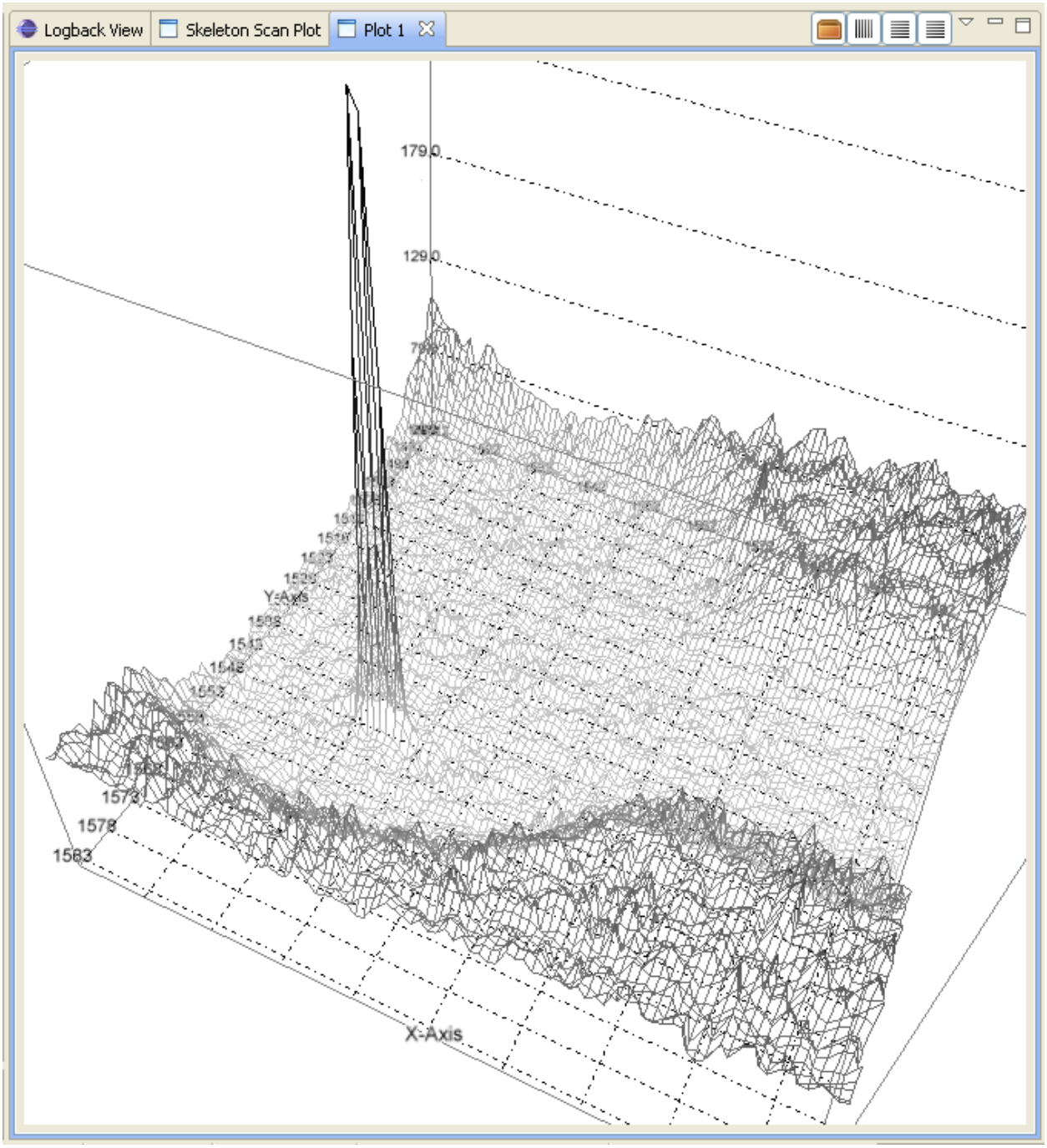


- multiple 1D scalar as points in 2D
- multiple 1D scalar as points in 3D
- 2D scalar as image
- 2D scalar as 3D surface plot


Plot 1
ImageExplorer View
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Y-Axis
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
X-Axis
Active
Active


Plot 1
ImageExplorer View
Z-Axis
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
Y-Axis
X-Axis


Logback View
Histogram: Plot 1
Side: Plot 2
Plot 1
TransferFunction
0
1000
2000
3000
4000
Y-Axis
0
1000
2000
3000
4000
X Position
Y Position
Data value
q X (1/A)
q Y (1/A)
q Z (1/A)
2theta (deg)


Logback View
Skeleton Scan Plot
Plot 1
179.0
129.0
Y-Axis
1573
1578
1583
X-Axis

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 line plots:

  ```
  import scisoftpy as dnp

  dnp.plot.line([x,] y)
  ```

  plots the given `y` axis dataset (or list of datasets) against a `x` dataset (if given)

- multiple 1D scalar line plots as 3D series:

  ```
  dnp.plot.stack([x,] y, z=None)
  ```

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

- 2D scalar image plots (also 2D compound datasets such as RGB datasets are supported and shown in colour):

  ```
  dnp.plot.image(image, x=None, y=None)
  ```

  plots the 2D dataset as an image

- 2D scalar points plots:

  ```
  dnp.plot.points(x, y, size=0)
  ```

  plots the points defined by `x` and `y` datasets. `size` can be a number or a dataset and defines the size of each point plotted

- 2D scalar 3D surface plots:

  ```
  dnp.plot.surface(data, x=None, y=None)
  ```

  plots the 2D dataset as a surface

- 3D scalar points plots:

  ```
  dnp.plot.points(x, y, z, size=0)
  ```

  plots the points defined by `x`, `y` and `z` datasets. `size` can be a number or a dataset and defines the size of each point plotted. The colours of each point depends on its size and the colour mapping used

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

```
dnp.plot.setdefname('Plot 2')
```

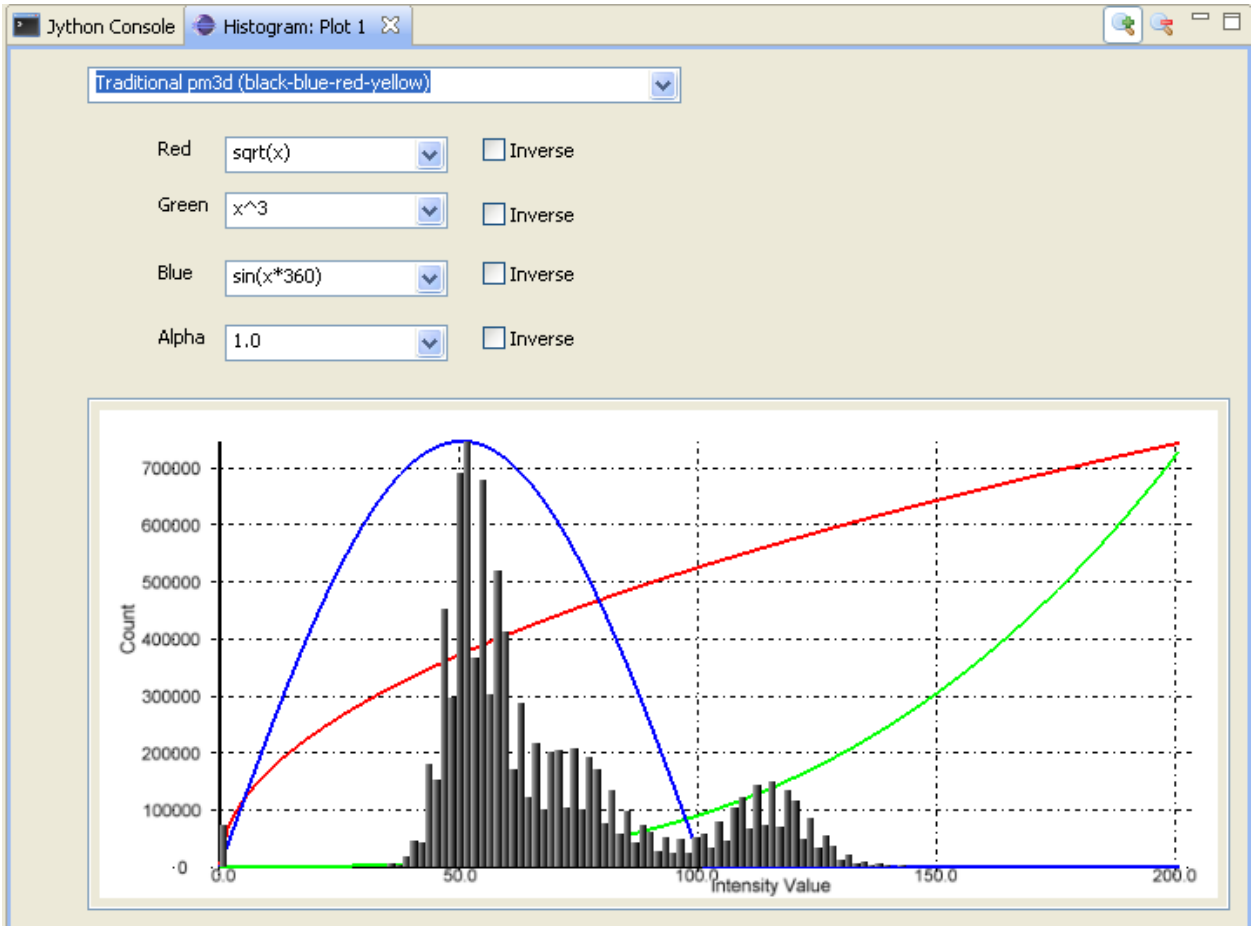
Otherwise, data can be sent to other plot views on a plot-by-plot basis using the optional keyword argument, `name`. For example:

```
dnp.plot.plot(y, name="Plot 2")
```

In points plots, more points can be added with:

```
dnp.plot.addpoints(x, y, z=None, size=0)
```

Both 2D image plots, 2D surface plots and 3D points 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.


Jython Console
Histogram: Plot 1
Traditional pm3d (black-blue-red-yellow)
Red
sqrt(x)
Inverse
Green
x^3
Inverse
Blue
sin(x*360)
Inverse
Alpha
1.0
Inverse
700000
600000
500000
400000
300000
200000
100000
0
Count
0.0
50.0
100.0
150.0
200.0
Intensity Value

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. Note, for compound datasets, only the first element of each item is analysed by the profile tools. In the case of coloured images, the red channel is profiled.

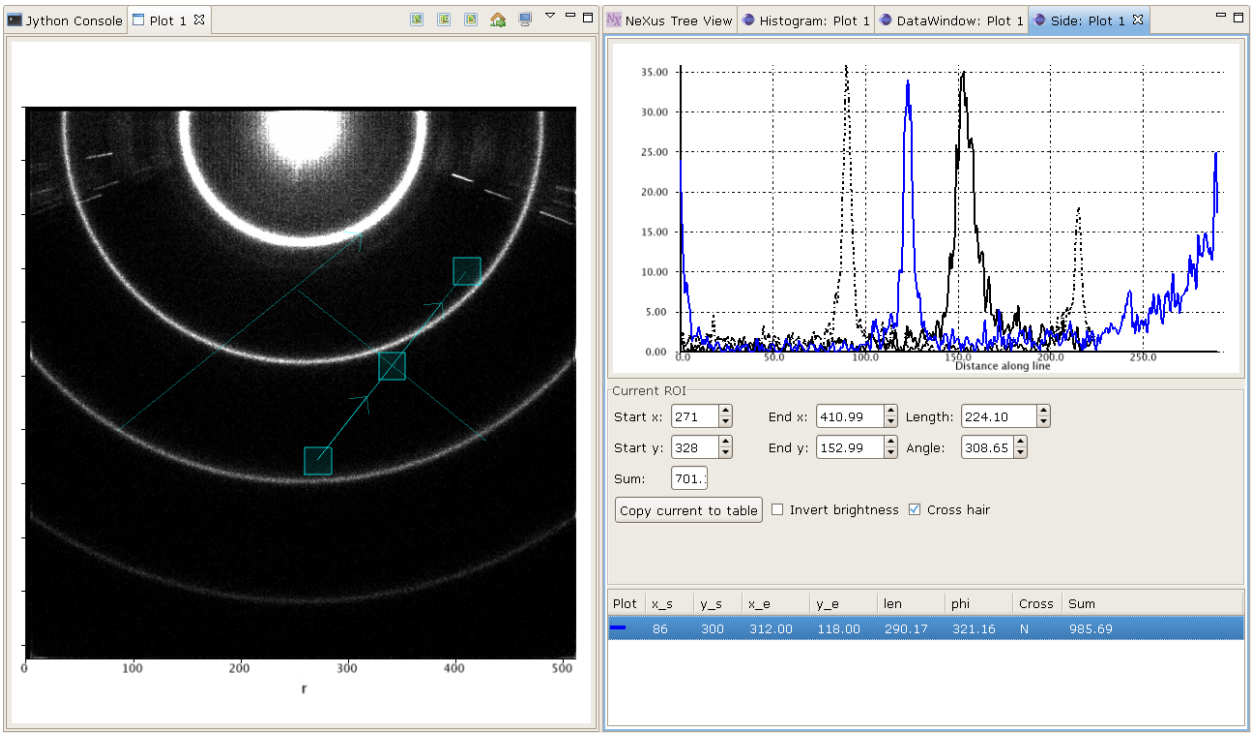


Figure 5.1: Line profile tool

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.

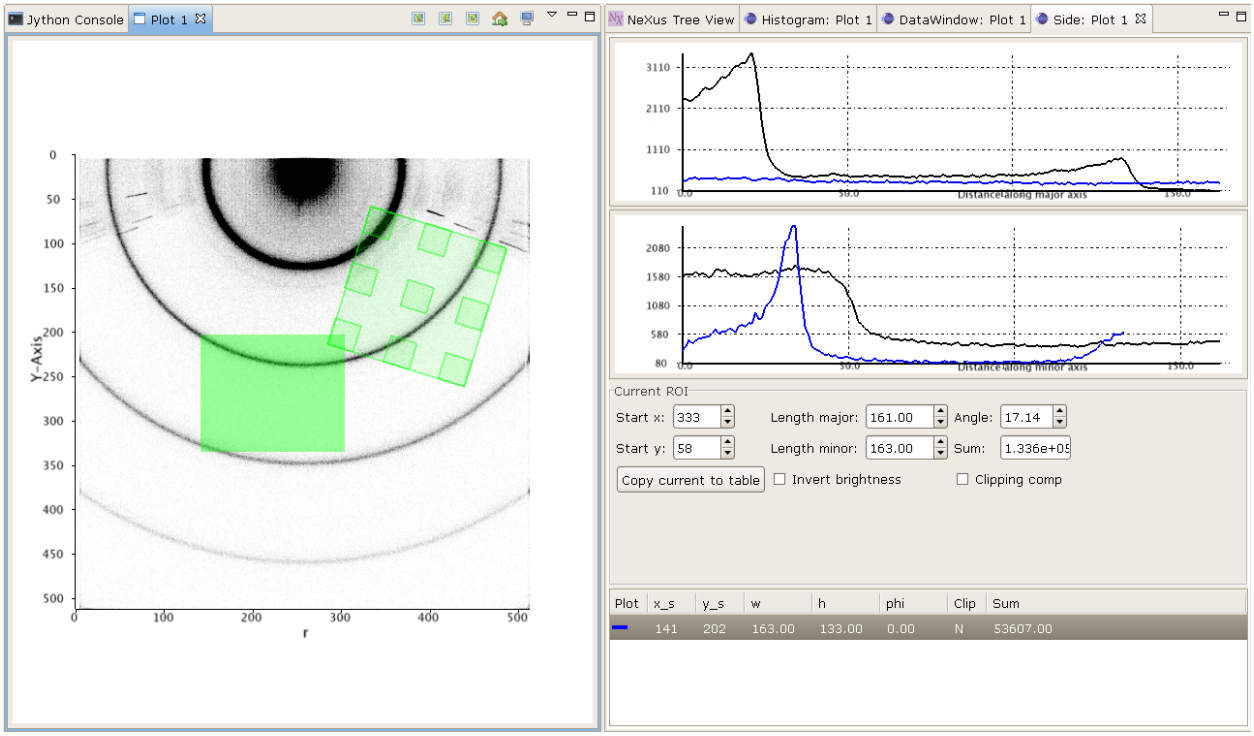


Figure 5.2: Box profile tool

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 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 *phi*) 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.

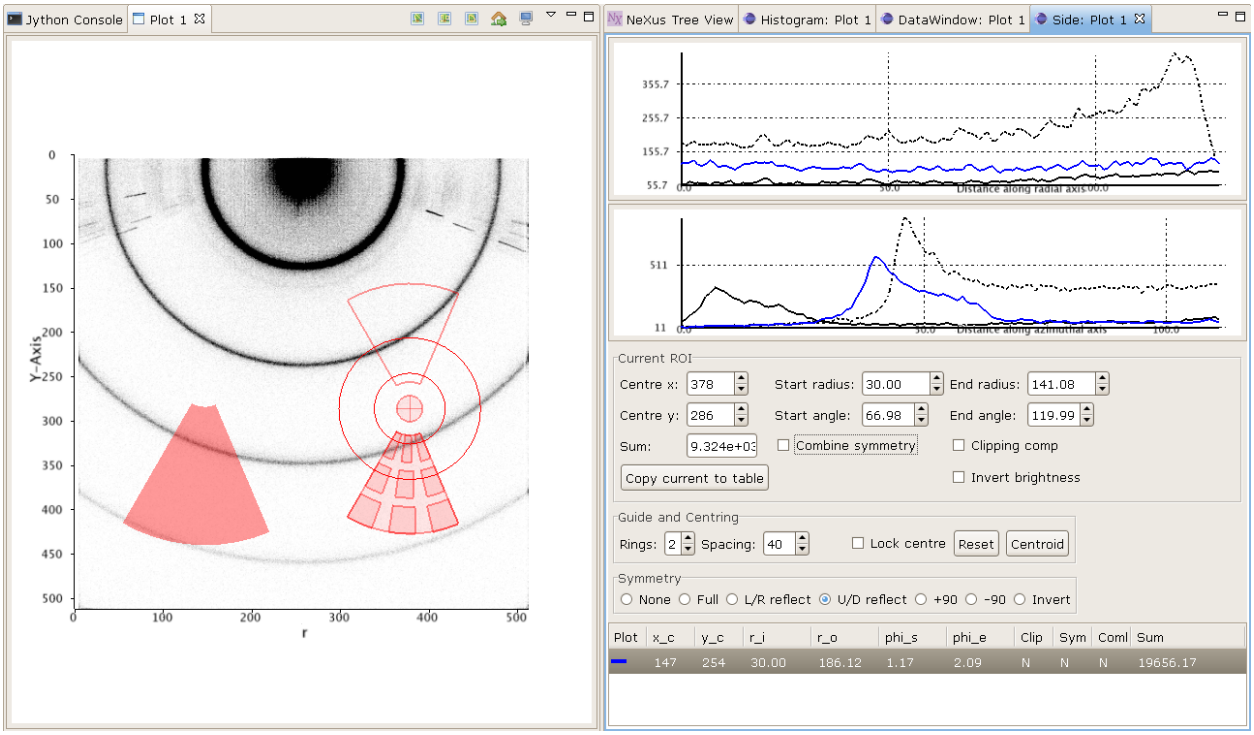


Figure 5.3: Sector profile tool

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.

All profile plots allow zooming using the left-hand mouse button to drag out a rectangular area to magnify. A right-hand mouse button click brings up a dialogue box where there are buttons for switching between linear and logarithmic scales on the `y` axis, undoing previous zooms and resetting the plot. The initial choice of y axis scale used in all the profile plots is controlled by the setting found within Windows > Preferences > Scisoft Settings > Side Plotter.

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)
```

CHAPTER

SIX

FITTING

The `fit` module contains a `fit.fit` function and some built-in fitting functions.

6.1 Fitting functions

The fit function requires fitting functions to be specified. There are a number of built-in fitting functions defined in `fit.function` listed in the table below:

Name	Description	Parameters
offset	constant offset, a	a
linear	linear function, $ax + b$	a, b
quadratic	quadratic function, $ax^2 + bx + c$	a, b, c
cubic	cubic function, $ax^3 + bx^2 + cx + d$	a, b, c, d
step	double step profile	base, start, end, outer step, inner step, inner width fraction, inner offset fraction
gaussian	Gaussian profile (normal profile)	position, FWHM, area
lorentzian	Lorentzian profile (Cauchy or Breit-Wigner profile)	position, FWHM, area
pvoigt	pseudo-Voigt profile	position, Gaussian FWHM, Lorentzian FWHM, area, mixing
pearson7	Pearson VII profile	position, FWHM, mixing, area

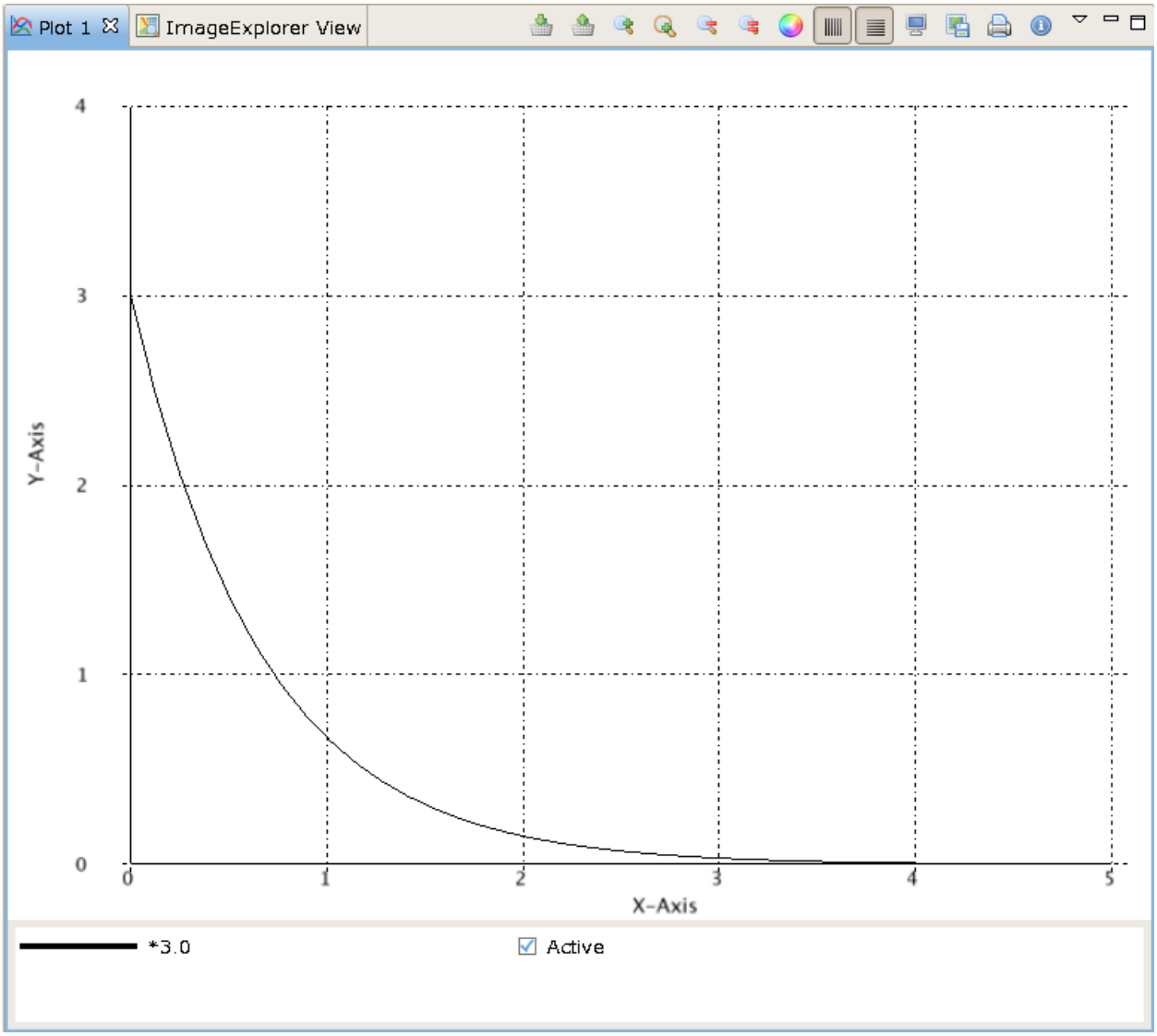
Functions defined in Jython can also be used as long as they conform to a standard argument signature:

```
import scisoftpy as dnp

def myfunc(p, x, *arg):
    '''p -- list of parameters
    x -- list of coordinate datasets
    arg -- tuple of additional arguments
    '''
    return p[0]*dnp.exp(-p[1]*x[0])
```

This function uses two parameters defines a negative exponential function. It can be tested in a line plot:

```
x = dnp.linspace(0, 5, 40)                # dataset of 40 coordinate points between 0 and 5
dnp.plot.line(x, myfunc([3., 1.5], [x])) # plot line of evaluated function
```


Plot 1
ImageExplorer View
4
3
2
1
0
Y-Axis
0
1
2
3
4
5
X-Axis
*3.0
Active

6.2 Fit usage

Once a fitting function (or a set of fitting functions) is chosen, the `fit` function can be invoked:

```
fr = dnp.fit.fit(func, coords, data, p0, bounds=[], args=None, ptol=1e-4, optimizer='local')
```

where `func` is a function or list of functions, `coords` is a coordinate dataset (or list of datasets), `data` is a dataset that contains the data to fit against, `p0` is a list of initial parameters, `bounds` is a list of tuples of lower and upper limits, `args` is optional arguments, `ptol` is fitting tolerance, and optimizer specifies the underlying methods used to make the fit.

By default, all parameters are limited to values between `-scisoftpy.floatmax` and `scisoftpy.floatmax`. The `bounds` list specify how to set each parameter's limits. `None` values in the list indicate the corresponding parameter is skipped over. Each item in the list can be a single number or a tuple - a single number is used to set the lower limit. For a tuple, the first value is the lower limit and, if the tuple has more than one item, the second value is the upper limit. `None` values in a tuple indicate the corresponding limit is skipped over. Thus if just an upper bound of 12.3 is required then use `(None, 12.3)`. These bounds are particularly important when using a global optimizer as this reduces the search space significantly.

By default, a local optimizer is used, which minimises the chi-squared value of fit vrs the data, to the nearest local minima. This means that the fit will not necessarily return the best result, however the local techniques are quite fast if a good start point can be guessed at. The global techniques take longer, but are much more likely to find the global minima, however they are slower, and need to be bounded to work effectively if at all. The current minimisers available are

Name	Description
local	points to the simplex method
global	points to the genetic method
simplex	uses the Simplex, or Nelder Mead local optimisation technique
genetic	uses an implementation of a differential evolution genetic algorithm

Built-in functions are specified solely by their names; Jython functions are specified as tuples with each tuple as a pair of function name and number of parameters. If the aforementioned Jython function was used, it would be given as `(myfunc, 2)`. The following example shows a fitting of a composite function - a negative exponential function that is offset by a constant amount:

```
x = dnp.linspace(0, 5, 20)
d = dnp.array([ 3.5733e+00, 2.1821e+00, 1.8313e+00, 1.9893e+00, 8.3145e-01,
    9.8761e-01, 7.1809e-01, 8.6756e-01, 2.3144e-01, 6.6659e-01, 3.8420e-01,
    3.2560e-01, 6.0712e-01, 5.0191e-01, 5.1308e-01, 2.8631e-01, 2.3811e-01,
    7.6472e-02, 2.1317e-01, 9.1819e-02 ])
fr = dnp.fit.fit([(myfunc, 2), dnp.fit.function.offset], x, d, [2.5, 1.2, 0.1], [(0,4), 0, (-0.2,0.7)

print fr  # print fit result
fr.plot() # plot fit result
```

The coloured lines in the plot are black for original data, blue for fitted composite, red for error, magenta for error offset, dark red for the negative exponential and green for the offset.

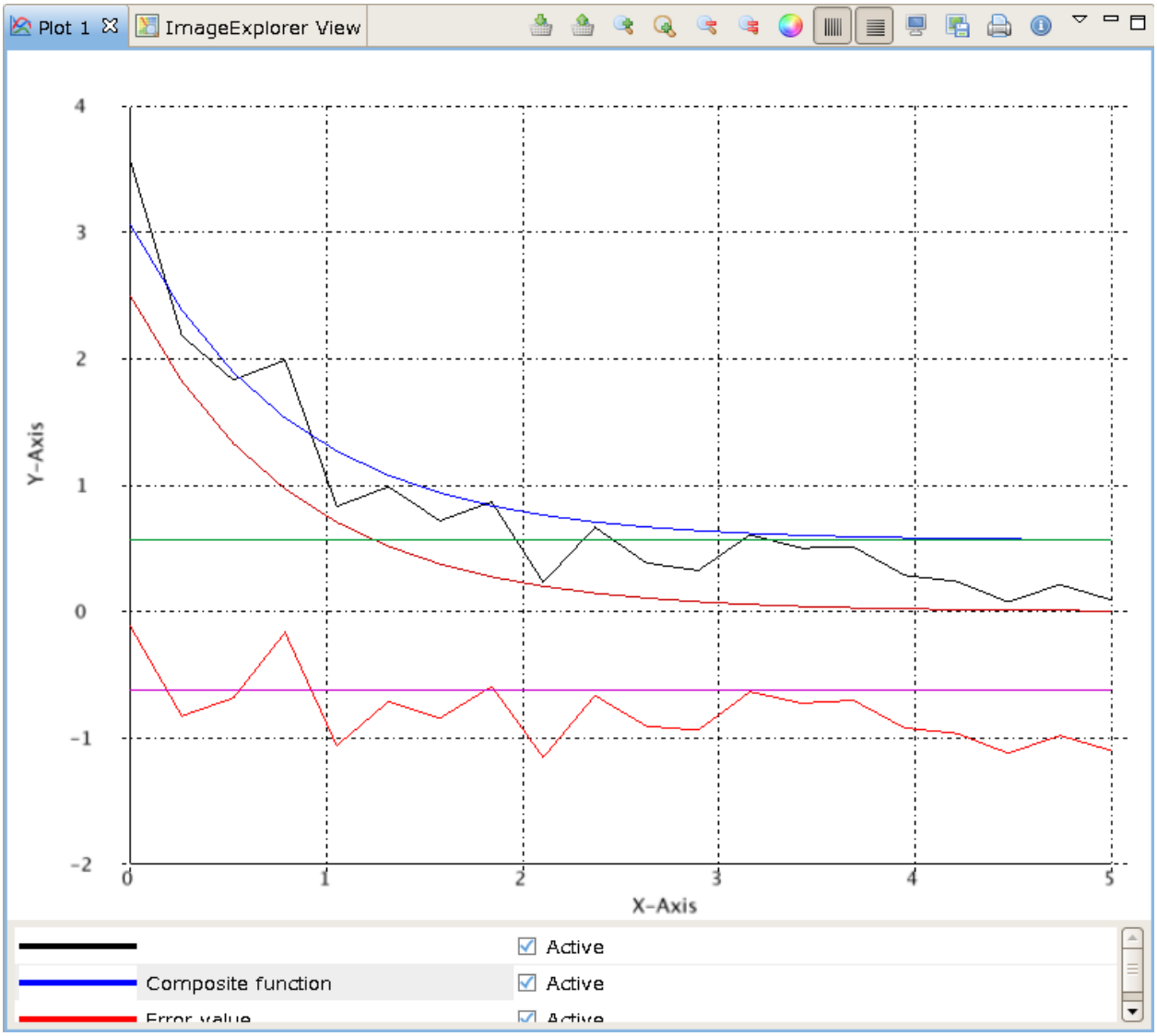
The object return by `fit` is a fit result object. It has a printable string version, a number of attributes and some useful methods:

func Fitted (composite) function

coords List of coordinate datasets used in the fitting function

data Dataset containing data that was fitted to

parameters List of fitted parameter values


Plot 1
ImageExplorer View
Y-Axis
X-Axis
4
3
2
1
0
-1
-2
0
1
2
3
4
5
Active
Composite function
Active
Error value
Active

residual Value of final residual (sum of squared differences between fitted function and data)

area Area (or hyper-volume) under fitted function assuming the coordinates were uniformly spaced

plot() Plot data, fitted function, error and its offset, each component of the fitted composite function

makefuncdata() Create list of datasets evaluated using the composite function and each of its components

makeplotdata() Create list of datasets for plotting (used by plot())

Also, the parameter values can be accessed directly with square brackets – that is, the object acts like a list:

```
len(fr) # number of parameters in fit result
fr[0]   # 0th parameter value
```

CHAPTER

SEVEN

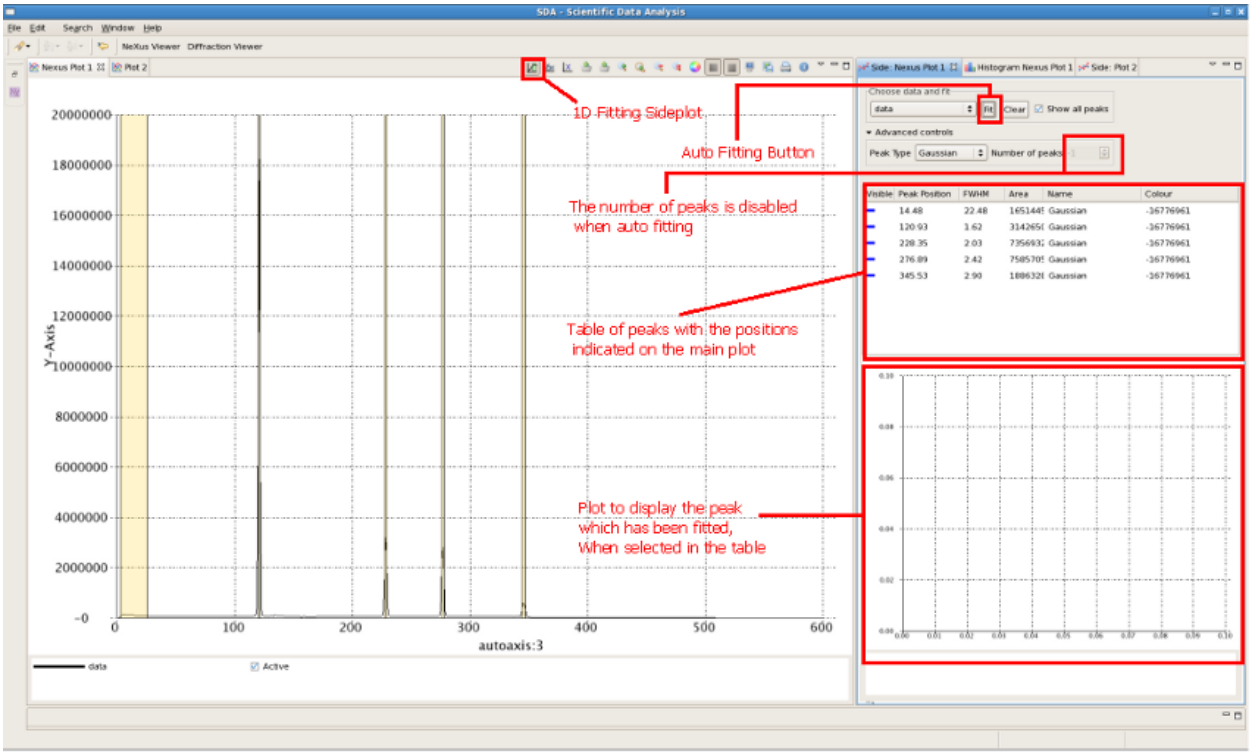
FITTING 1D SIDEPLOT

7.1 Overview

This side plot allows peaks to be fitted to a 1-dimensional dataset. There are 2 basic modes of operation:

- Auto Fitting.
- Click and drag fitting.

The fitting algorithm works in the following manner. Initially the first derivative of the one dimensional dataset is taken. A smoothing function is applied during this process. The default value for the smoothing value is 1% of the data length or this value can be set manually using the Fitting1D preferences page. Using the first derivative, the regions thought to contain peaks are found. These peak containing regions are identified by a region of positive gradient followed by a region of negative gradient. Using this information an optimisation routine is applied to the data in the region where the peak was found. The selected peaks is then fitted to the data. Currently, the types of peaks fitted can be chosen in the side plot or in the preferences page.



The results from the peak fitting are then shown in the table and their positions are indicated in the plot. An individual peak can be selected from the table and the region where the peak came from is highlighted in the main plot view, the data and the fitted peak are displayed in the plot at the bottom of the sideplot.

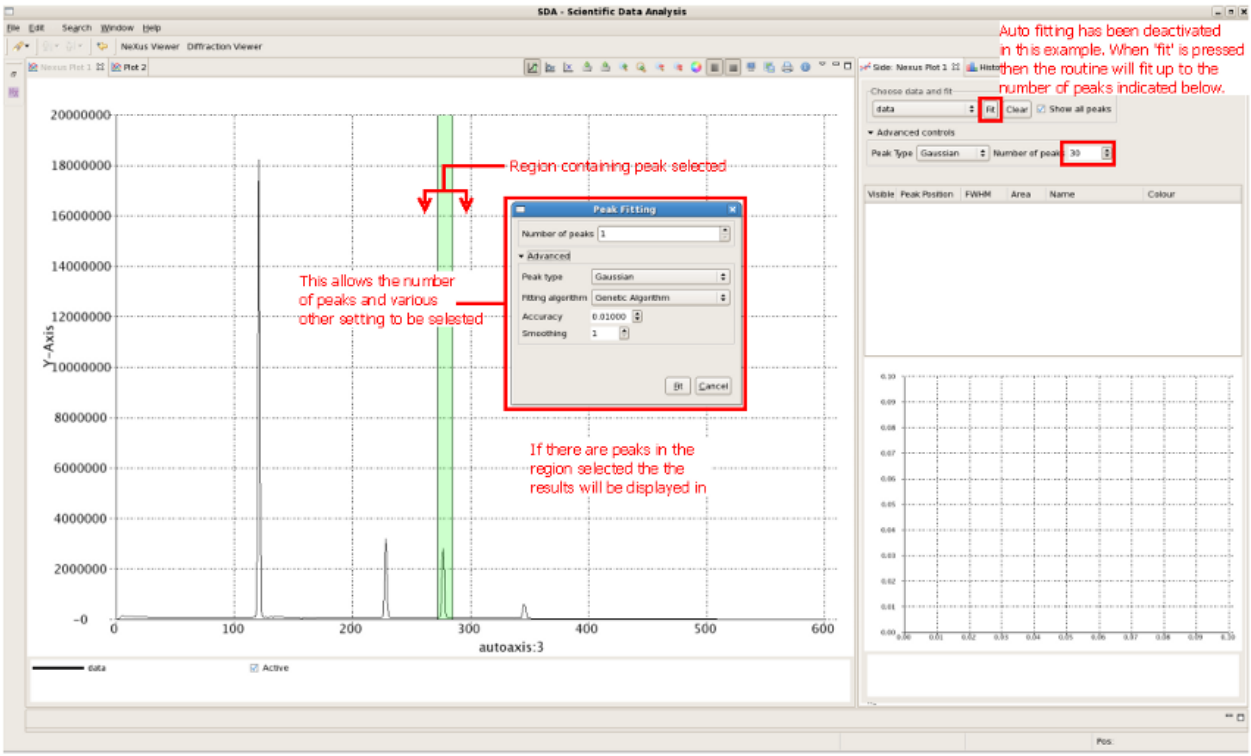
7.2 Auto Fitting

This mode of operation is designed to make an attempt at fitting peaks to the input data automatically but still allowing the user to manipulate the peaks found after this operation. There are two ways to do this: The first method is to specify the number of peaks that the fitting routing should fit. This will attempt to fit the *n* peaks with the largest area and then stop. If there are less than *n* peaks in the data then the routine will fit the maximum number of peaks that the peak finding routine has found.

The second method of automatic fitting is to continue to fit peaks to the data until a threshold is met. There are two different measures which are available through the preference page: **area** and **height**. Using this method will continue to fit peaks to the data until the peak being fitted is less than the threshold selected in the preference page. For example, if area is selected as the measure and the threshold is set 0.05 the peak fitting routine will continue until the next peak being fitted is less than the 5% of the area of the largest peak. The automatically found peaks are coloured blue in the table.

7.3 Click and Drag Fitting

This allows regions to be selected from the plot and peaks to fitted within this region. To do this right click and drag across the region containing the peak. When the button is released then the a menu allowing the user to choose the number of peaks that are to be fitted to the region selected. Using the advanced settings, the type of peak, fitting method (including accuracy) and the smoothing can be specified. If a peak is found then the table is updated and the manually found peaks are coloured green



7.4 Peak Manipulation

Once peaks have been found, the location of these peaks can be shown on the main plot by selecting 'show all peaks'. This will highlight the region of data where a peaks has been found in the main plotter. The box is defined as the full width at half maximum of the fitted peak. This allows the performance of the peak fitting to be assessed by examining the plot after the routing is finished. By right clicking on a row in the table the data and the fitted peak is shown in the lower plot. Overlaid onto this sub plot is the full width at half maximum and the mean of the fitted peak.

If the peak is found to be incorrectly fitted then the peak can be deleted from the list. This is particularly useful if the automatic peak fitting has found an erroneous peak. This is achieved by left clicking on the row in the table and selecting delete. A peak can also be edited using the edit option. Using this a different probability density function can be fitted to the region where the existing peak was found, for example. If no peak is found the table remains unchanged.

7.5 Preferences

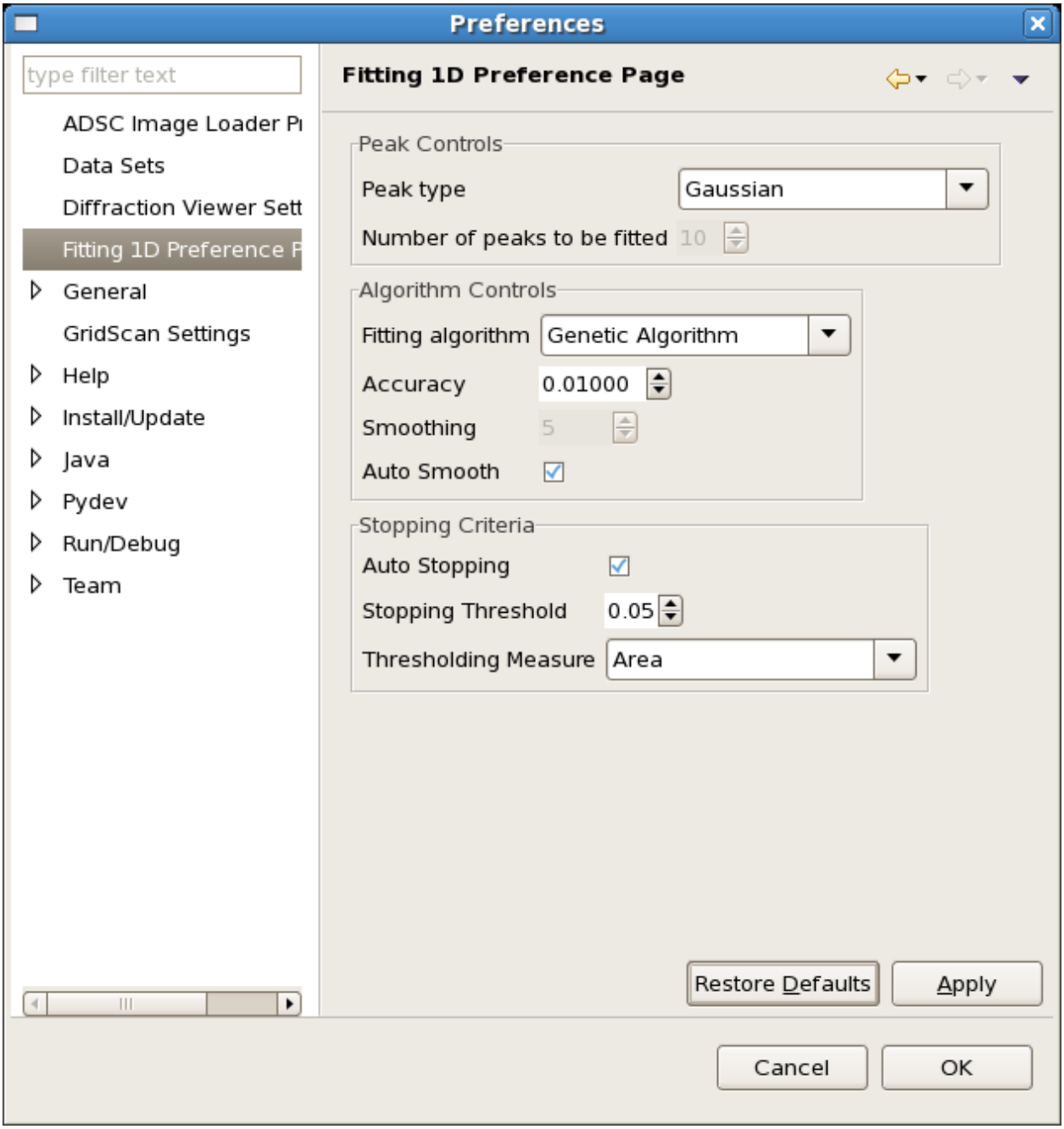
The preferences interface is used to control various features in the fitting routine and interface. The preferences can be found in Windows > Preferences.

The 'Peak control' section allows which type of peak is fitted to the data in the plot. This will then be the type of peak fitted for auto fitting and is preserved between sessions. The number of peaks is the maximum number of peaks the routine will fit if automatic stopping is not selected. If automatic stopping is selected then this will be disabled.

Below this is the algorithm controls. This allows the the type of fitting algorithm employed and the accuracy of that algorithm to be specified. The lower the number the more accurate, but more expensive and time consuming, the fitting will be. The smoothing that is applied to calculate the differential of the data being fitted can also be specified. The units associated with this smoothing are the number of data points in the data. When auto smoothing is selected the smoothing is set to 1% of then data length.

The stopping criteria can also be customised. When selected the routine will continue until the specified threshold is met. This threshold is the proportion of the largest peak. The measure can be either the height of the largest peak or the area of the largest peak.

For example, if the threshold is set to 0.10 and the largest peak is has an area of 100 then the routing will continue to run until the next peak being fitted as an area of less that 10. At this point the routine will stop and the results from the peaks found will be displayed.


Preferences
type filter text
ADSC Image Loader Pr
Data Sets
Diffraction Viewer Sett
Fitting 1D Preference P
General
GridScan Settings
Help
Install/Update
Java
Pydev
Run/Debug
Team
Fitting 1D Preference Page
Peak Controls
Peak type
Gaussian
Number of peaks to be fitted
10
Algorithm Controls
Fitting algorithm
Genetic Algorithm
Accuracy
0.01000
Smoothing
5
Auto Smooth
Stopping Criteria
Auto Stopping
Stopping Threshold
0.05
Thresholding Measure
Area
Restore Defaults
Apply
Cancel
OK

CHAPTER

EIGHT

NEXUS TREE VIEW

8.1 Introduction

The NeXus [1] 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.

8.2 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 used as plot axes.

[1] NeXus: http://www.nexusformat.org

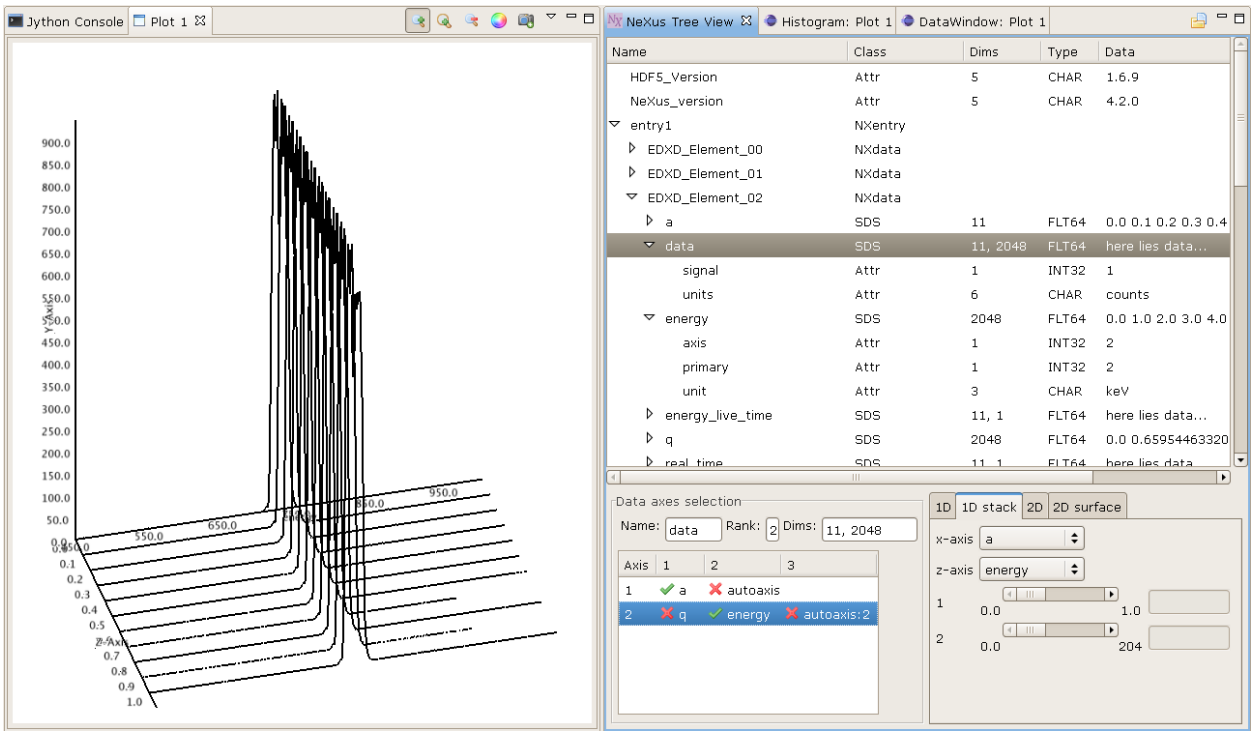


Figure 8.1: NeXus tree viewer

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

8.3 References

CHAPTER

NINE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*