
GDA Analysis Guide

Release 8.2

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

March 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 | DataSet | 5 |
| 2.1 | Jython | 6 |
| 3 | Input/Output | 7 |
| 4 | Plot server | 9 |
| 4.1 | Jython | 9 |
| 5 | Plot View | 11 |
| 5.1 | Jython | 12 |
| 6 | 2D Image Plot Profiles | 17 |
| 6.1 | ROI objects | 19 |
| 6.2 | Jython | 19 |
| 7 | NeXus Tree View | 21 |
| 7.1 | Introduction | 21 |
| 7.2 | Interaction | 21 |
| 7.3 | Jython | 21 |
| 8 | 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 DLS's 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 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.

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>

DATASET

The DataSet class aims to emulate some of the functionality of NumPy's ndarray class. This section documents version 1.0 of the Scisoft analysis plugin or version 8.1 of GDA.

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 views
- 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, min, max, argmin, argmax, sum, all, any
- Array creation: array, zeros, ones, linspace, arange
- Array manipulation: repeat, tile
- Array modification: fill, append
- Maths: add, subtract, multiply, divide, negative, power, absolute, exp, exp2, log, log2, log10, expm1, log1p, sqrt, square, reciprocal
- Trig: sin, cos, tan, arcsin, arccos, arctan, arctan2, hypot, sinh, cosh, tanh, arcsinh, arccosh, arctanh, deg2rad, rad2deg
- Rounding: rint, ceil, floor
- Stats: mean, std, var
- Random: rand, randint, random_integer, randn, exponential, poisson, seed

Non-NumPy

- Constructors

- Dataset methods: get, set, getAbs, setAbs, getFirstValue, getLastValue, range
- Maths: dividez, cbrt
- Stats: getAverageDeviation, setChiSquared, skew, kurtosis, centroid
- min, max, getMinPos, getMaxPos
- containsNans, containsInfs
- Import/export JAMA
- exec functions: CentroidND, Histogram, Integrate2D, LineSample, MapToPolar, MapToRotated-Cartesian, Sum, MakeMask

2.1 Jython

This is an example of DataSet usage:

```
from gda.analysis import *

# create a 1D dataset from 0 to 9
a = DataSet.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 = DataSetMaths.pow(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 dataset of 2 rows and 3 columns
a = DataSet.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 = DataSet.array([[-2, 2.3], [1.2, 9.3]])
# modify dataset and set a slice to the values in another dataset
a[1:,1:] = b

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

INPUT/OUTPUT

The primary class is ScanFileHolder.

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

```
from gda.analysis.io import *
shf = ScanFileHolder()
shf.load(ADSCImageLoader("dataFile"))
```

Where the 'ADSCImageLoader' can be interchanged with one of the following:

- CrystallisLoader()
- MACLoader()
- MARloader()
- PilatusTiffLoader()
- CBFLoader()

Various images can be read

- JPEGLoader()
- PNGloader()
- TIFFImageLoader()

There is a facility to save the output from a ScanFileHolder as either a .png or .jpeg file. The following example assumes that there is a ScanFileHolder called shf which contains a dataset that is to be output as an image:

```
from gda.analysis.io import *
shf.save(JPEGSaver("/your/directory/image.jpeg"))
shf.save(PNGSaver("/your/directory/image.png"))
```

This will save a .jpeg image and a .png image containing the data held within the ScanFileHolder. If there are multiple datasets held within the ScanFileHolder then multiple images will be saved, suffixed with a number representing the number of the dataset within the ScanFileHolder. There are also an issue where the image types can only save an image to a given bit depth. To save in image that has a greater bit depth then is it recommended that the scaled savers are used:

```
from gda.analysis.io import *
shf.save(JPEGScaledSaver("/your/directory/image.jpeg"))
shf.save(PNGScaledSaver("/your/directory/image.png"))
```

These methods will scale the data held within the ScanFileHolder such that the pixel values will be scaled within 0 and the maximum bit depth of the image format.

PLOT SERVER

The plot server sits on the command server and acts an intermediary between the command server and the GUI clients. GUI information and plot data can be passed back and forth. It also passes NeXus trees.

4.1 Jython

The plot client regularly updates the plot server with GUI information. This can be obtained from the server using the RCPPlotter class:

```
from uk.ac.diamond.scisoft.analysis.plotserver import *  
  
# grab bean  
gb = RCPPlotter.getGuiBean("Plot 1")
```

The GuiBean is a dictionary object with a set of possible keys listed in the GuiParameters 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 the server:

```
RCPPlotter.setGuiBean("Plot 1", gb)
```

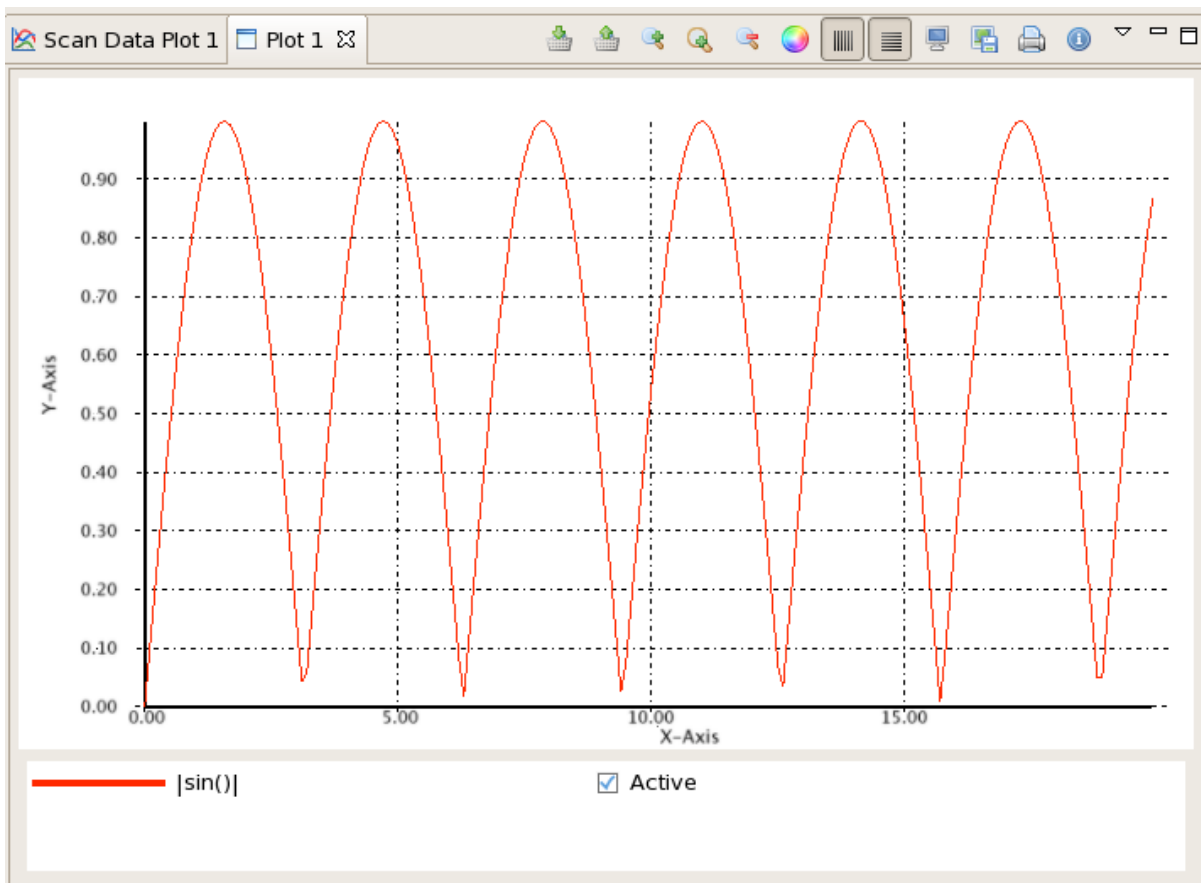
and the client will respond appropriately to the updated GUI information. The keys for the dictionary are listed as strings in the GuiParameters class:

```
dir(GuiParameters)
```

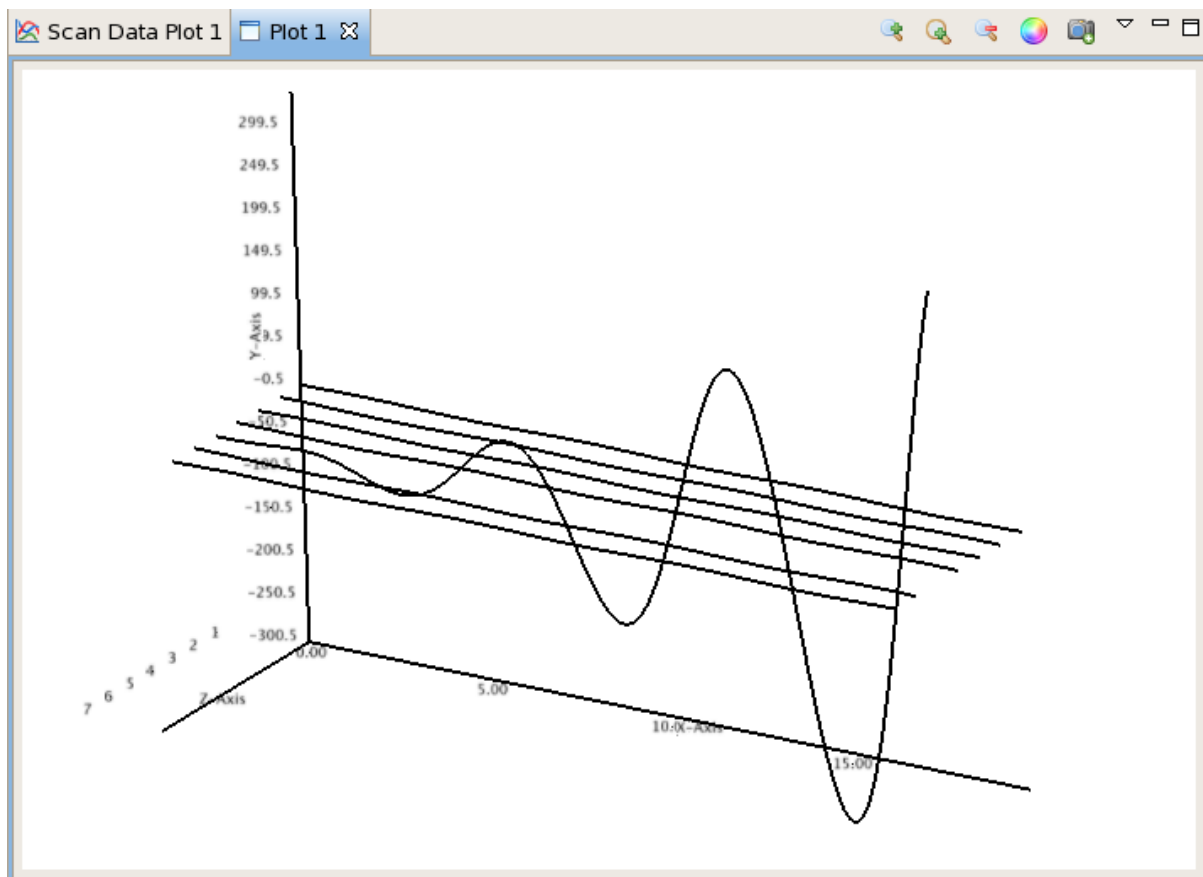

PLOT VIEW

The plot view is the main window where all graphical plotting is displayed. Plot view is a generic plotting UI, that allows graphical plotting of different scalar dataset types. Currently supported scalar type sets 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 instance of the Plot view open and plot to 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.

5.1 Jython

Plotting any data in any form to one of the Plot Views can be done from the server using the RCPPlotter class:

- 1D scalar plots:

```
RCPPlotter.plot("Plot 1", xAxisDataSet, yAxisDataSet)
```

- multiple 1D scalar plots as 3D series:

```
RCPPlotter.stackPlot("Plot 1", xAxisDataSet, [yAxisDataSet1, yAxisDataSet2, ..., yAxisDataSetn])
```

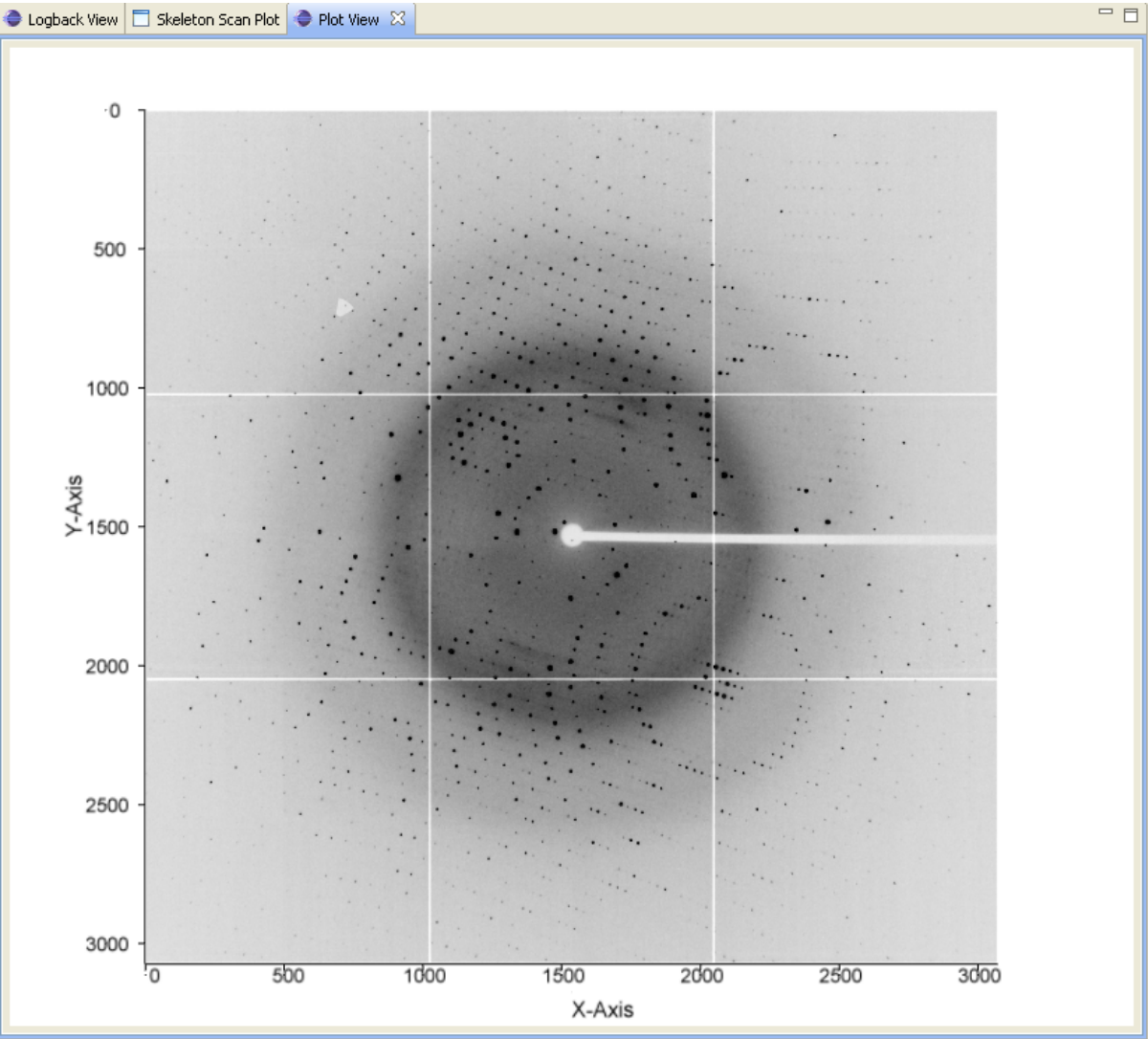
or in case of multiple x-axis:

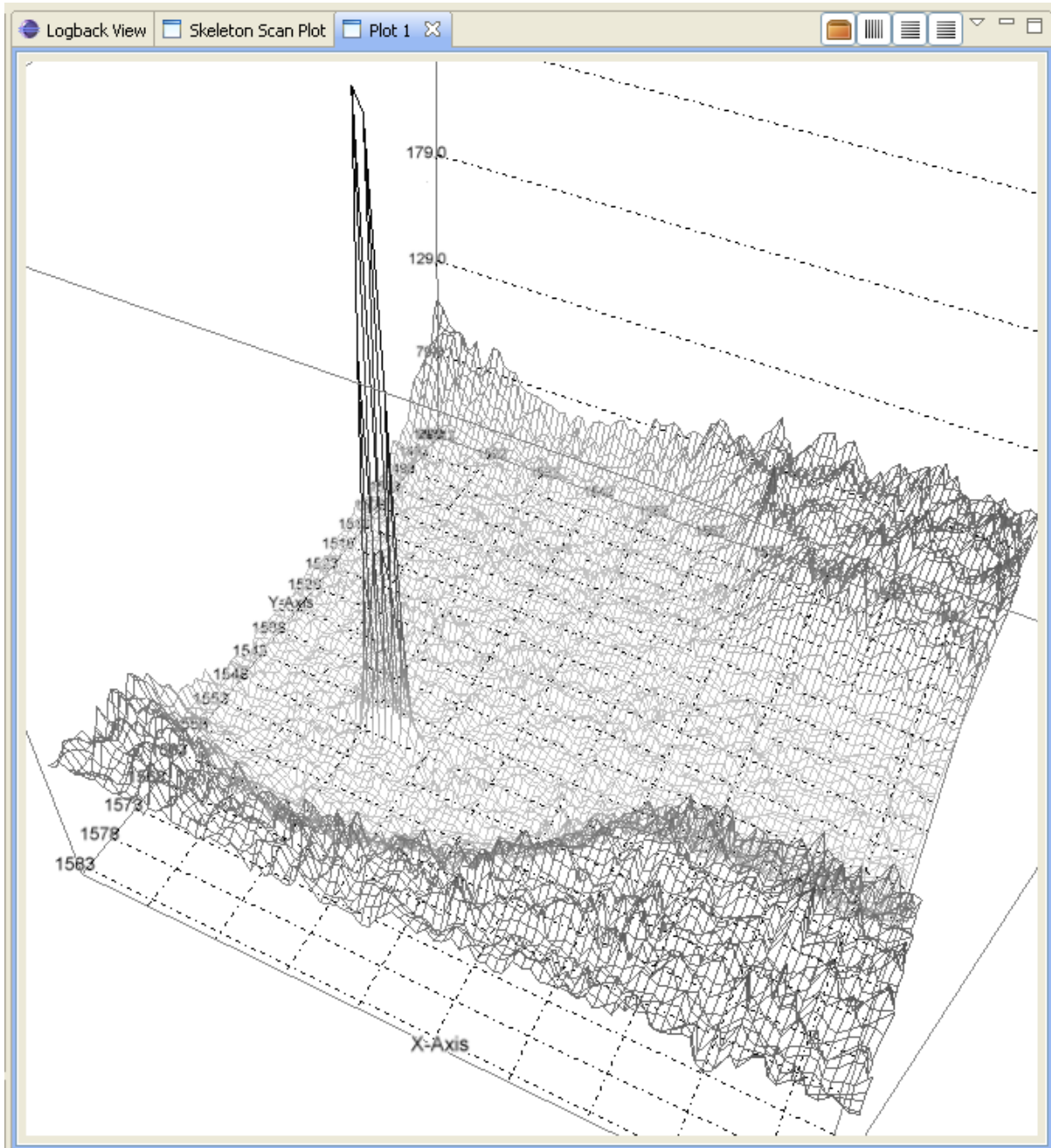
```
RCPPlotter.stackPlot("Plot 1", [xAxisDataSet1, xAxisDataSet2, ..., xAxisDataSetn],
[yAxisDataSet1, yAxisDataSet2, ..., yAxisDataSetn])
```

or if the z axis should be specified as well:

```
RCPPlotter.stackPlot("Plot 1", xAxisDataSet,
[yAxisDataSet1, yAxisDataSet2, ..., yAxisDataSetn],
zAxisDataSet)
```

combination of the previous two:





```
RCPPlotter.stackPlot("Plot 1", [xAxisDataSet1, xAxisDataSet2, ..., xAxisDataSetn],  
                        [yAxisDataSet1, yAxisDataSet2, ..., yAxisDataSetn],  
                        zAxisDataSet)
```

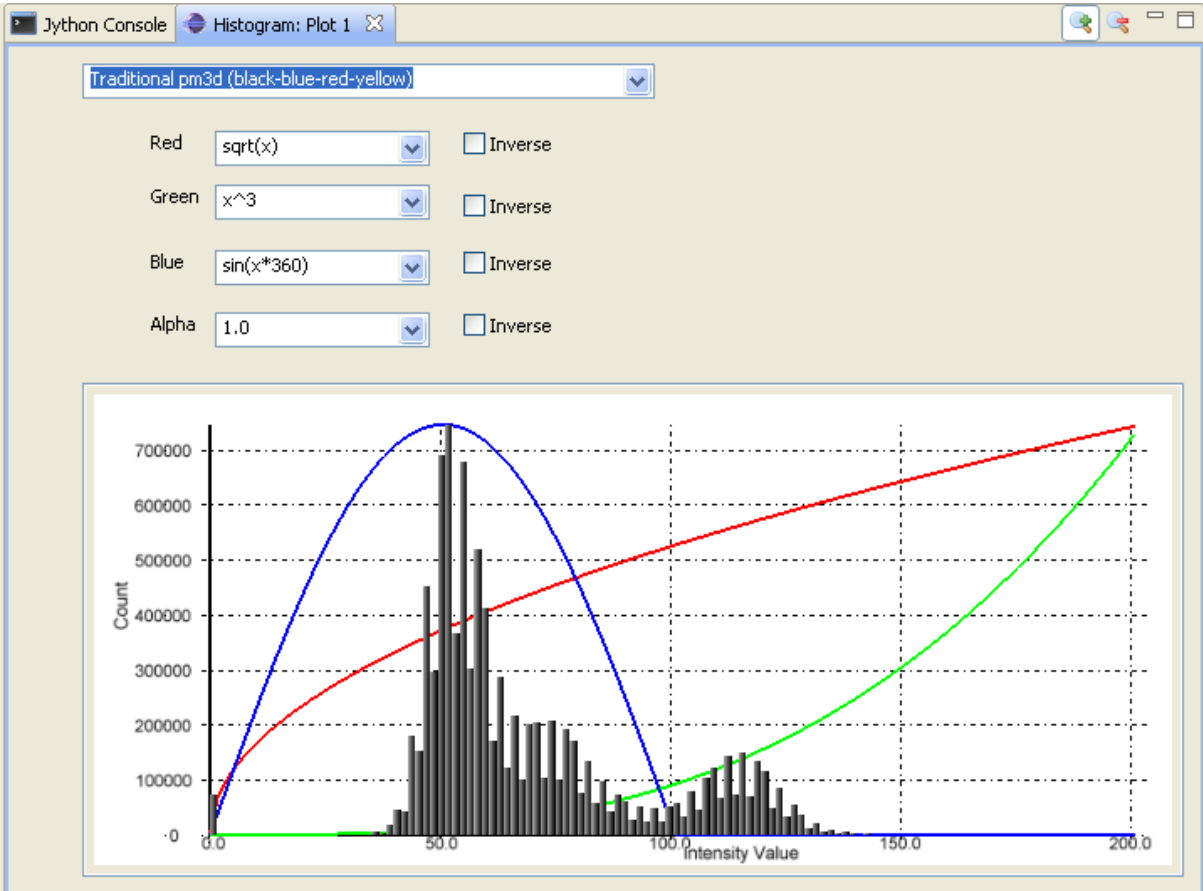
- 2D scalar image plots:

```
RCPPlotter.imagePlot("Plot 1", imageDataSet)
```

- 2D scalar 3D surface plots:

```
RCPPlotter.surfacePlot("Plot 1", imageDataSet)
```

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 to the different colour channels.



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.

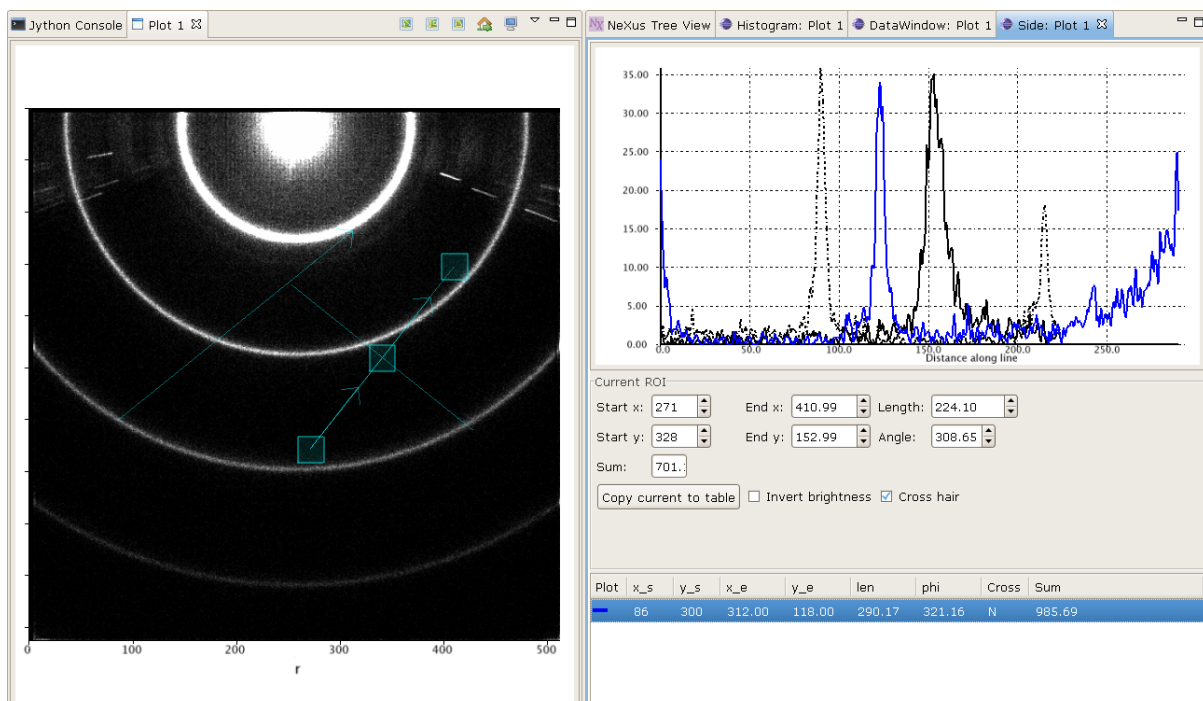


Figure 6.1: Line profile tool

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.

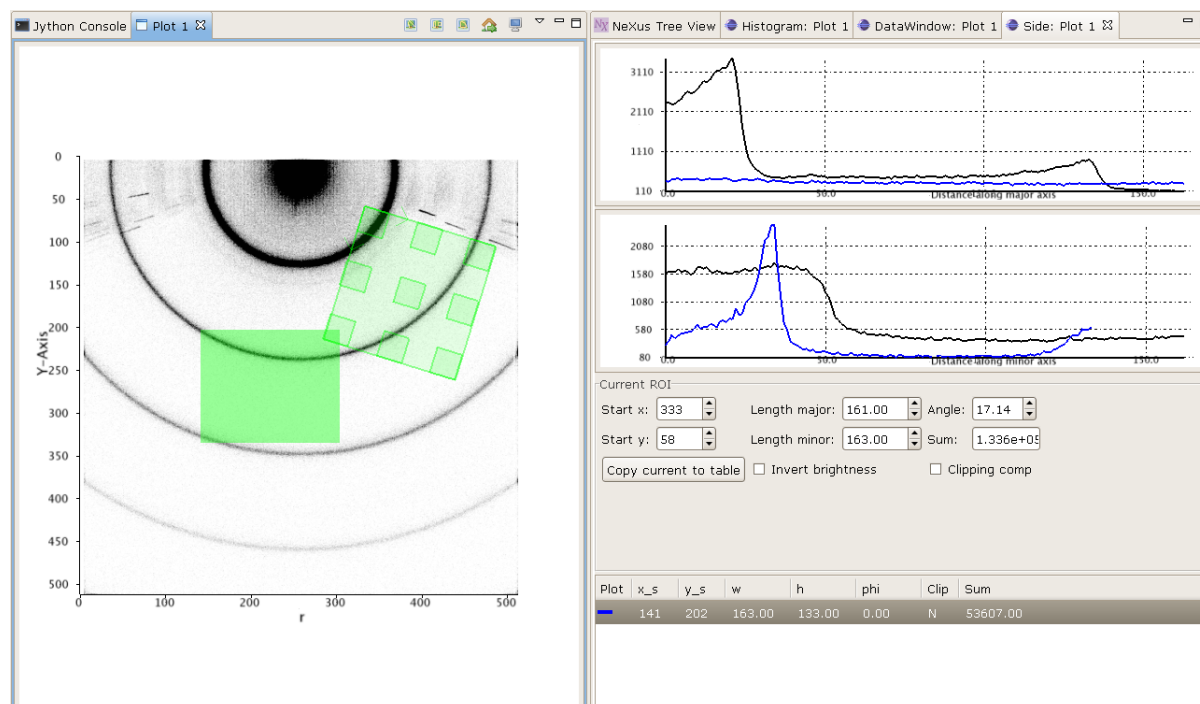


Figure 6.2: Box profile tool

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 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.

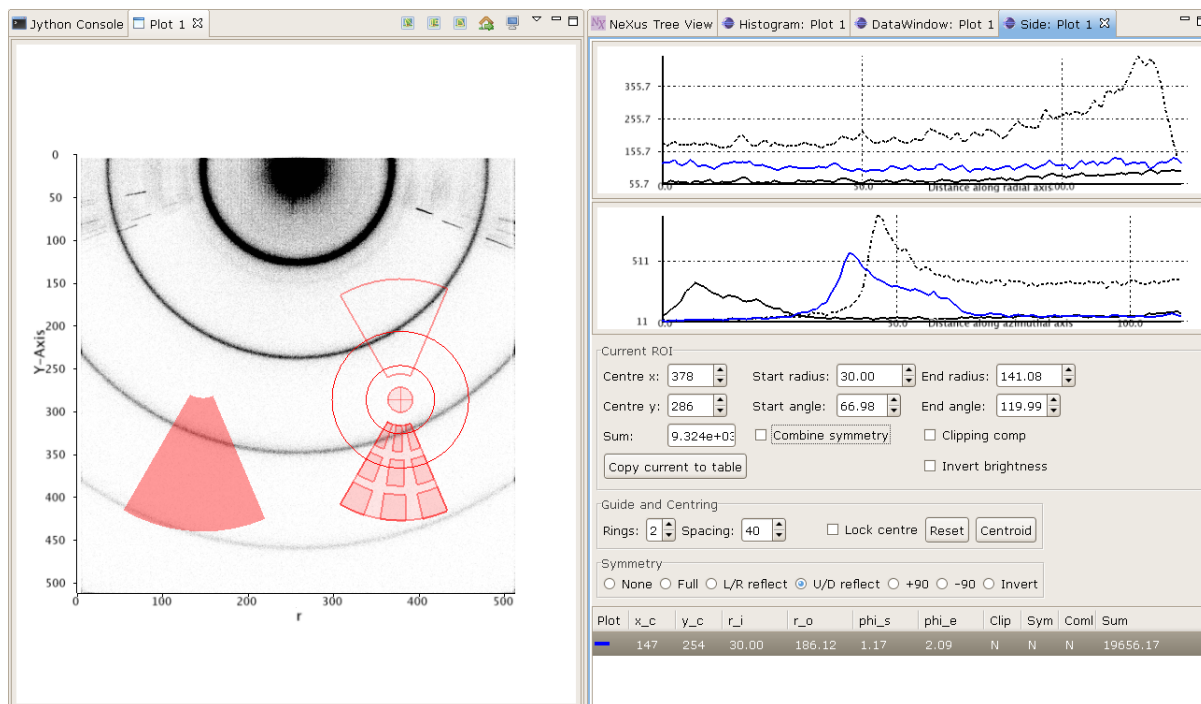


Figure 6.3: Sector profile tool

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.

6.1 ROI objects

The regions of interest defined are:

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

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

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

6.2 Jython

The current ROI and any ROIs stored in the table are sent via a GUI bean back to the plot server. A GUIBean is a Jython dictionary object and is obtained as follows:

```
from uk.ac.diamond.scisoft.analysis.dataset.roi import *
```

```
gb = RCPPlotter.getGuiBean("Plot 1")
```

The current ROI is held in the GUIBean object under the key “ROIData” and the table of ROIs under the key “ROIDataList”. The values of the keys depend on which plot profile tool is active.

When the line profile tool is being used, the ROIData item is a LinearROI object and any stored ROIs are held in a Jython list of LinearROIs:

```
cr = gb["ROIData"] # or gb[GuiParameters.ROIDATA]

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

lr = gb["ROIDataList"]

# get first item
ra = lr[0]

print ra.length, ra.angleDegrees

# copy ROI from list
roi = gb["ROIDataList"][0].copy()

# modify ROI
roi.setPoint(100,50)

# create new bean and add ROI
gbb = GuiBean()
list = LinearROIList()
list.add(roi)
gbb["ROIDataList"] = list

# push bean back
RCPPlotter.setGuiBean("Plot 1", gbb)
```

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

```
# 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 = ROIProfile.line(image, lroi, 0.5)
```

NEXUS TREE VIEW

7.1 Introduction

The NeXus 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.

7.2 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.

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.

7.3 Jython

To load a NeXus file:

```
from gda.analysis import *  
  
s = ScanFileHolder()  
  
s.load(NexusLoader("/path/to/file.nxs", True))  
  
RCPNexusTreeViewer.viewNexusTree("nexusTreeView", s.getNexusTree())
```

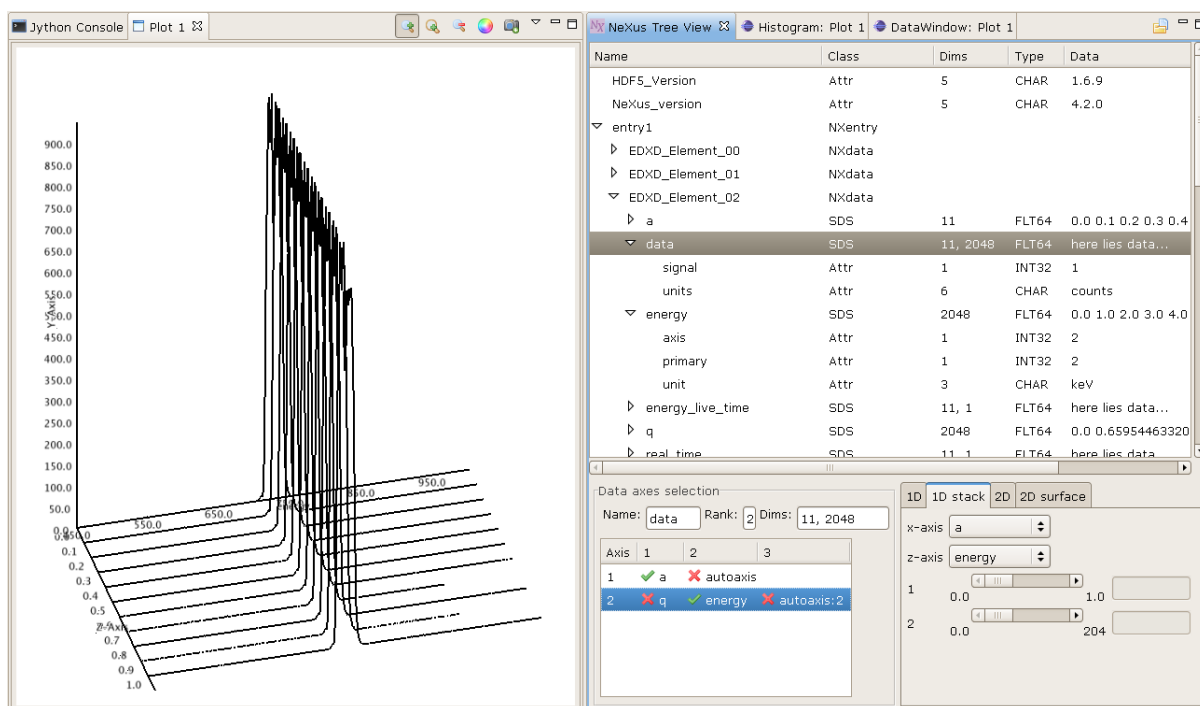


Figure 7.1: NeXus tree viewer

INDICES AND TABLES

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