Mantid Documentation

Release 4.0.0

Mantid

May 30, 2019

CONTENTS

These tutorials and training courses are designed to give you a step-by-step guide to how to use some of the basic features of Mantid. The initial sections deal with some of the fundamental functions that are required for (almost) any use of of Mantid. We then build up to more bespoke functionality, providing the tools that you need to manipulate Mantid for your particular problem. We also include some more instrument specific training, provided by various sites.

Mantid basic course, runs through functions such as loading, plotting, exploring and saving data. At the end of this course you can start to use Mantid independently.

Python introduction is a crash course in the Python that you need for scripting in Mantid.

Python in Mantid introduces how to interact with Mantid through Python. Some knowledge of Python is assumed; if you have completed the Python introduction course you are well equipped for this course.

Extending Mantid with Python demonstrates how you can become a power user and really stretch the capabilities of Mantid, by developing your own scripts.

Sections:

CHAPTER

ONE

MANTID BASIC COURSE

Sections

The mantid basics course gives a hands-on introduction to many of the important features of Mantid. You will learn how to load, process, visualise and analyse data using Mantid.

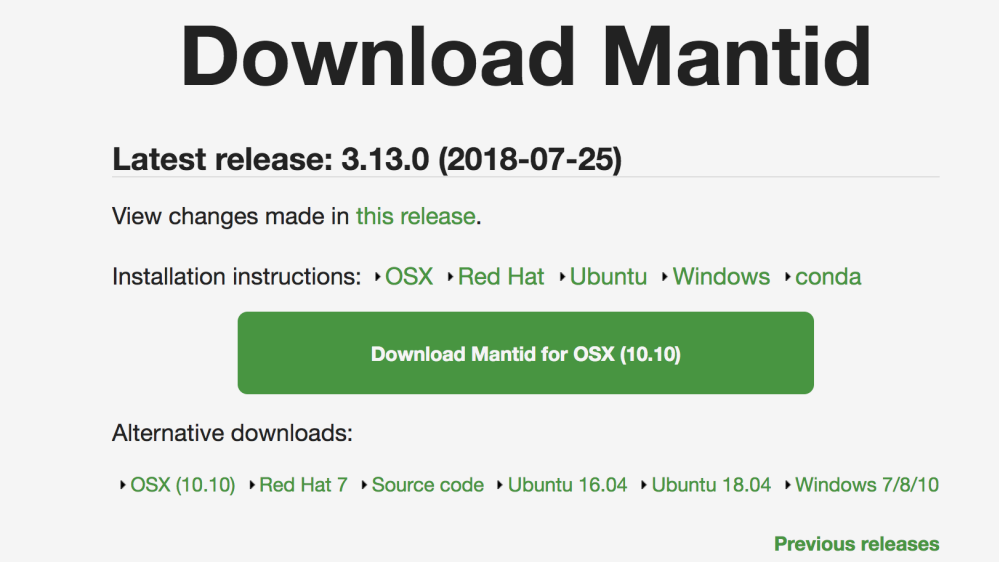
1.1 Setting Up and Getting Started

1.1.1 Installing Mantid

Mantid is available to download for Mac, Linux or Windows. Pre-compiled executable binaries are available to download from the Downloads section of the website.

On the downloads page you will be able to get the latest stable release of Mantid, which we recommend. There is also an option to download the latest nightly build, this is a more experimental version of the code. Whilst the nightly build may have more features, it is less tested and comes with much less guarantee.

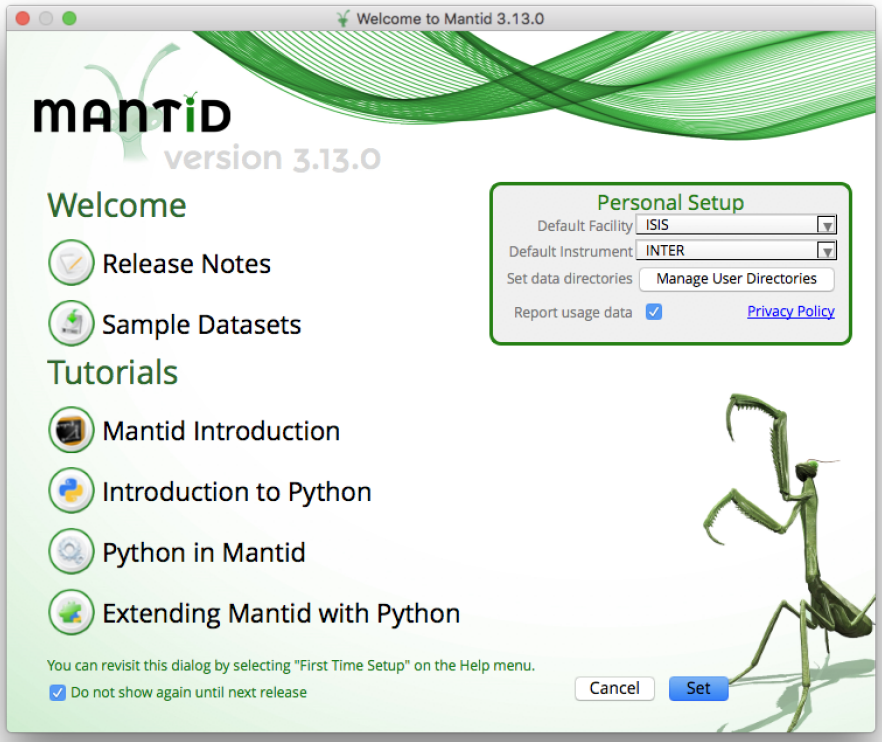
You will also find detailed instructions for installing Mantid on any of the above mentioned operating systems.



1.1.2 MantidPlot First-time Setup

The first time you start up MantidPlot you will get the "First- time Setup" screen, pictured below. This screen allows you to set default options related to the instrument from which your data was collected, and the locations to look for data by default. You can also access certain resources directly from this screen.

If you do not want to see this screen every time you start Mantid, you can check the "Do not show again until next release" box. You can re-access this screen from the main MantidPlot GUI at any time by going to "Help > First Time Setup".



- Select your favourite instrument, or default to ISIS and MARI.
- Click on the "Manage User Directories" button.
- Click "Browse To Directory" and navigate to the location of your data files (see section below for how to get some training data if needed).
- Do the same for the default save directory.
- Click "OK".
- Click "Set".

1.1.3 Getting Additional/training Data

All of the data you need to follow the introductory training in *Mantid Basic Course* can be downloaded from the Downloads section of the website.

- Download the file "TrainingCourseData.zip"
- Unzip this file in a location where MantidPlot will look by default, as specified in the section above on "First-time Setup".

1.2 Algorithms, Workspaces and Histories

Sections

1.2.1 Introduction

Introduction

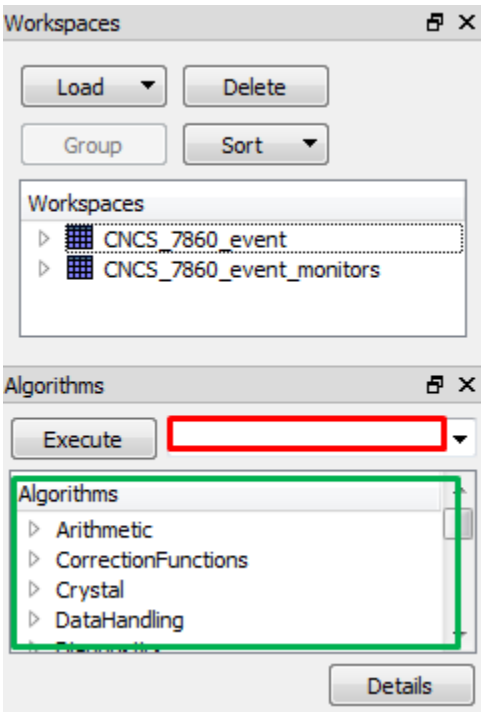
Workspaces are the data storage units of Mantid. If you want to create, save or manipulate in any way a workspace this is done using Mantid *Algorithms*.

For example an algorithm is used when loading data into a workspace. The current list of available algorithms is very large, but most algorithms interact with workspaces in the same way, generating a new output workspace as the result.

In this section we will introduce you to some basic types of workspaces, and some key algorithms to operate upon those workspaces.

1.2.2 Algorithms

Algorithms

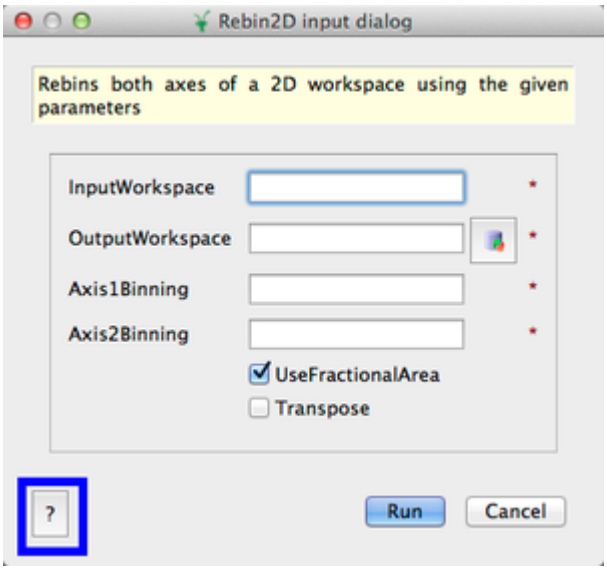


Algorithms are the verbs of Mantid - they are the "doing" objects. If you want to manipulate your data in any way it will be done through an algorithm. Algorithms operate primarily on data in workspaces. They will normally take

one or more workspaces as an input, perform some processing on them and provide an output as another workspace (although it is possible to have multiple outputs).

In MantidPlot, your primary source for Algorithms is the Algorithm panel shown on the right: The panel should be visible by default on the bottom right of the MantidPlot window. It can be shown/hidden by choosing **View -> Algorithms** option from the MantidPlot top menu. The panel may also be re-located and re-docked by dragging the top bar with the left mouse button held down.

The algorithms can either be navigated by category (see green highlighted section), or searched by name (red highlighted section). In the search box, start writing the algorithm name, and it will predict the name of the algorithm as you continue typing.



Algorithm Help

Once you have an algorithm selected, you can get additional information about what it does, what it takes as inputs, what it provides as outputs, and how to use it. This is achieved via the help button on the algorithm dialog box that appears once the algorithm is selected. The help button shows up as a **?** and will take you to the documentation page specific to that algorithm.

Validation of Inputs

Many algorithms have mandatory input properties. Others have constraints on the inputs, such as numbers that are bounded. The red asterisk next to some of the input properties indicate that they must be provided before the algorithm can run. Hovering over the asterisk will tell you what is wrong with the input.

Input workspaces are also validated against. If the workspace type does not match the validator of the input workspace for the algorithm it will not appear as a possible workspace to use as an input.

Overwriting InputWorkspace

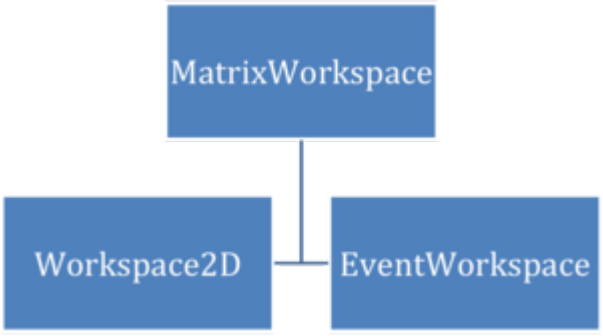
The button next to the OutputWorkspace will populate the OutputWorkspace property with the same name as that of the InputWorkspace. This will mean that the new OutputWorkspace will overwrite the original input workspace.

Running Algorithms

On the Algorithm dialog, the run button will execute the algorithm. If successful, new output workspaces will appear in the Workspace list panel. Algorithm outputs are sent to the Results Log panel. If the algorithm fails to complete, the reason should also appear in the same Results Log panel.

1.2.3 Workspaces

Workspaces come in several forms, but the most common by far is the MatrixWorkspace which represents XYE data for one or more spectra. The MatrixWorkspace itself can be sub-grouped into EventWorkspace and Workspace2D.



Workspace2D

A Workspace2D consists of a workspace with 1 or more spectra. Typically, each spectrum will be a histogram. For each spectrum X, Y (counts) and E (error) data is stored as a separate array.

Spectrum Number	X Data			
1	0	1	2	3
2	0	1	2	3

Spectrum Number	Y Data		
1	0	1	0
2	0	1	0

Spectrum Number	E Data		
1	0	0	0
2	0	0	0

Each workspace has two axes, the Spectrum axis and the X-axis. Where an axis holds a known unit type, it may be converted to another set of units.

Spectrum Number	X Axis	X Data		
1	0	1	2	3
2	0	1	2	3
3	0	1	2	3
4	0	1	2	3
5	0	1	2	3
6	0	1	2	3

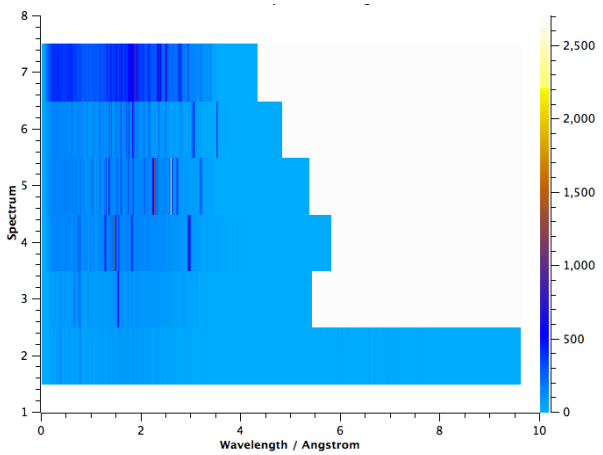
Spectrum Axis

Rebinning a Workspace2D is a one-way process when the rebinning leads to a coarser structure.

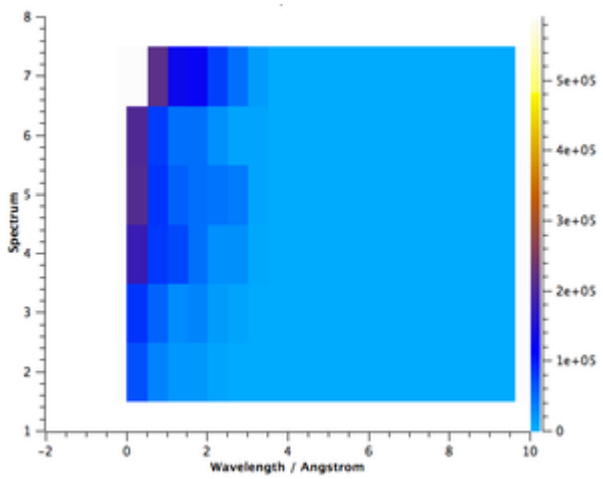
Ragged Workspaces

Converting x-axis units can lead to a ragged workspace, in which bin boundaries are not consistent across the spectra. Some algorithms will not accept input workspaces that are ragged. The fix to this is to apply Rebin to the Ragged workspace structure, as the example below shows.

1. run **Load** on *GEM38370_Focussed.nxs* setting the **OutputWorkspace** to be *ws*
2. run **ConvertUnits** on *ws* setting **OutputWorkspace** to *lambda*, **Target**=*Wavelength*, **EMode**=*Elastic*. Plotting this in the *Color Fill Plot* demonstrates the ragged X-bins.



3. run **Rebin** on *lambda* setting **Params** to *0.5* and **OutputWorkspace** to *Rebinned*. Plotting this in the *Color Fill Plot* demonstrates that uniform binning across all spectra has been achieved.



Event Workspaces

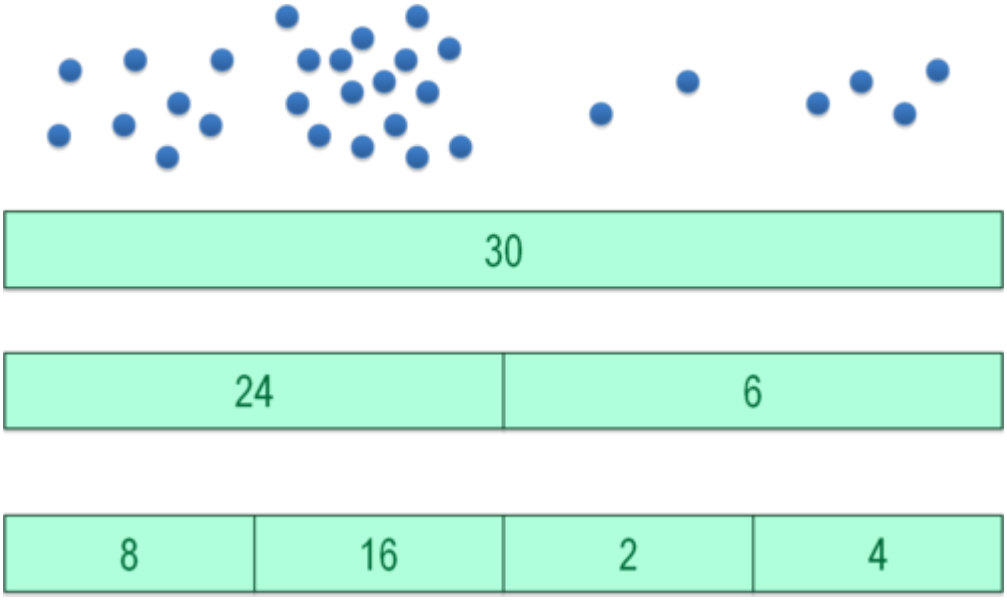
An EventWorkspace stores information about each individual event observation in detectors. More specifically, at a neutron spallation source, this means that the time of arrival and detector ID of each individual neutron is recorded. Only fairly recent advances in computer and acquisition hardware have made storing this detailed knowledge a practical solution. For example at the SNS facility all data, except for data collected in monitors, are stored in this way.

Event specifies "when" and "where"

Pulse time – when the proton pulse happened in absolute time

Time-of-flight – time for the neutron to travel from moderator to the detector

Basic Example



Rebinning

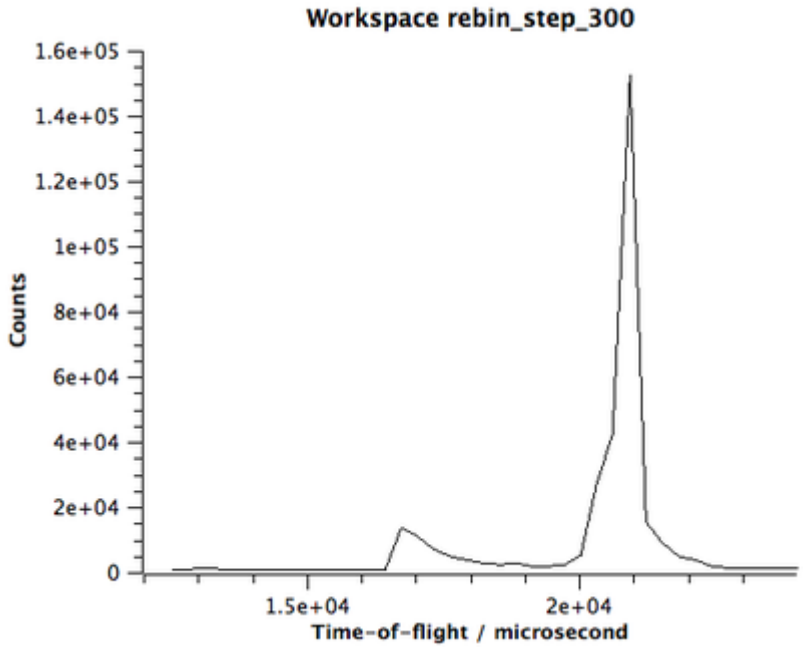
- Rebinning is essentially free and can be conducted in-place. This is because the data does not need to change, only the overlaying histogramming.

Performance

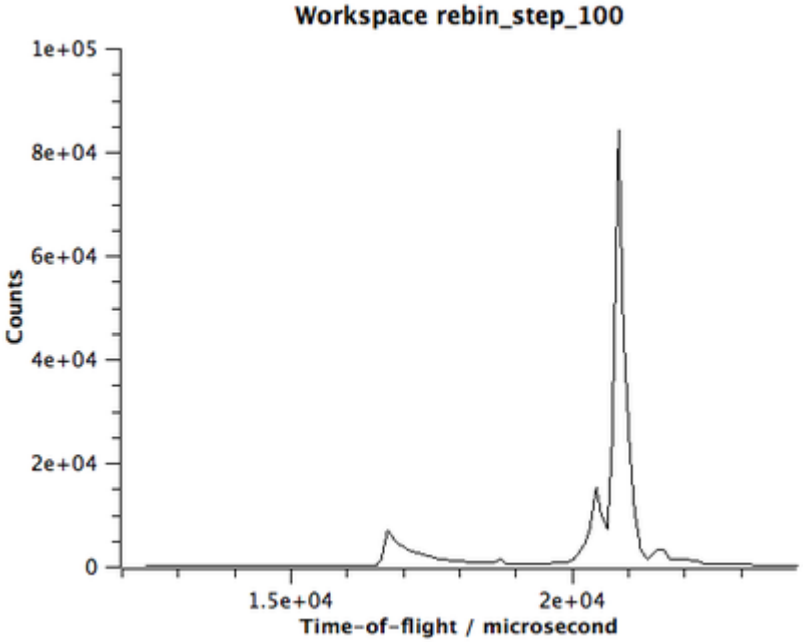
- Each event list is separate
- Sorting events is O(n) = n log(n)
- Histogramming is O(n) = n
- Only histogram as needed

Example of Workspace usage

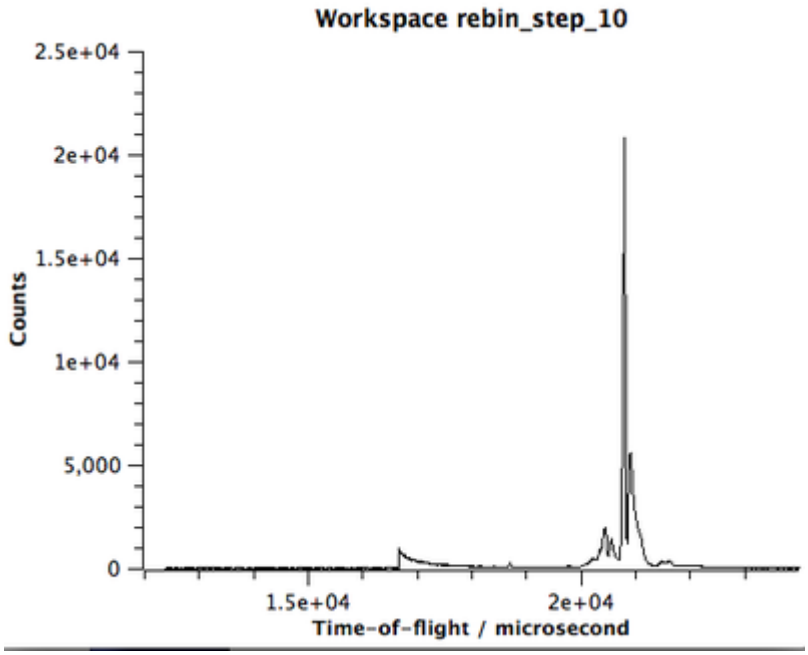
1. Load the event data HYS_11388_event.nxs
2. Execute the 'SumSpectra' algorithm
3. Rebin with Params=300 and plot, ensure PreserveEvents=True



4. Rebin with Params=100, the plot will automatically update, ensure PreserveEvents=True



5. Rebin with Params=10 the plot will automatically update, ensure PreserveEvents=True



Keep the workspace open for the next section.

Other Workspace Types

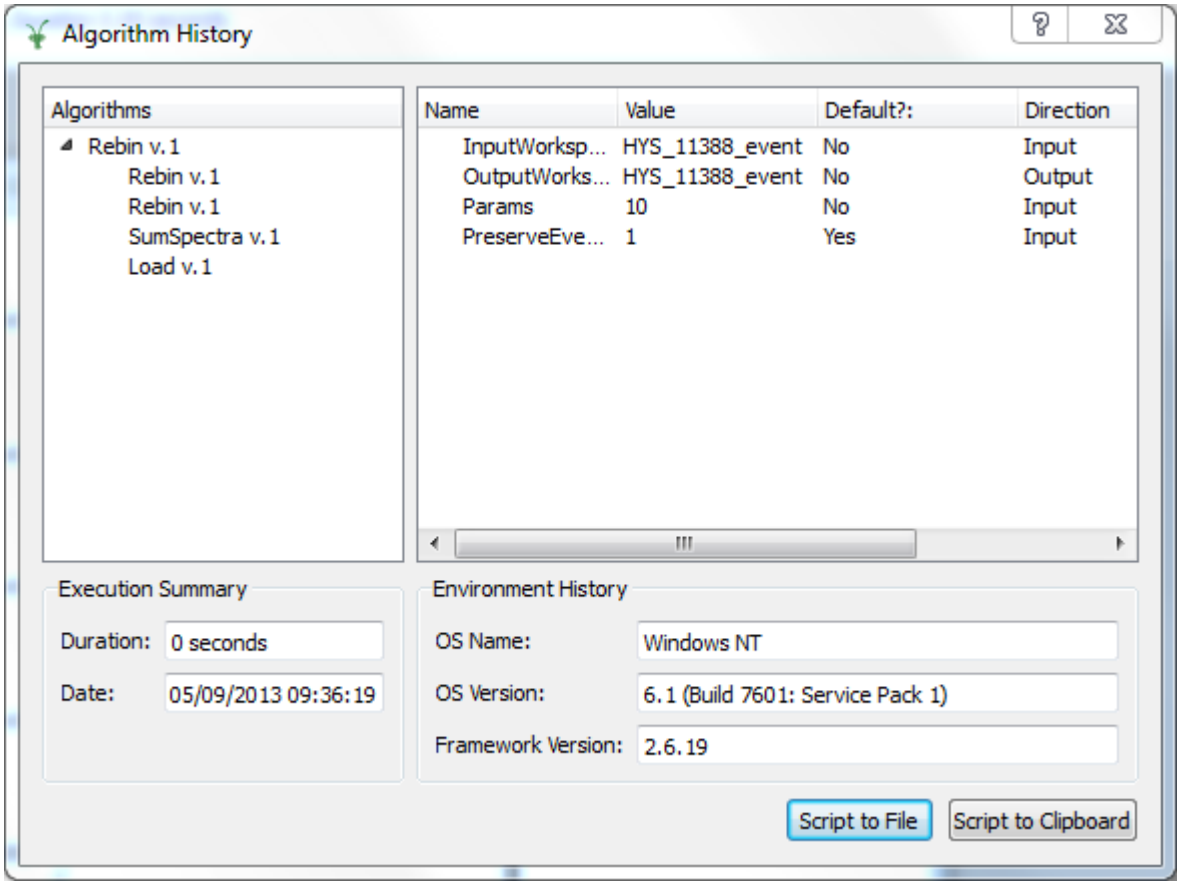
- GroupWorkspaces store a collection of other workspaces in a group, this can be created manually and is often used in multi-period data. Either the whole group or individual members can be processed using algorithms.
- TableWorkspaces stores data as cells. Columns determine the type of the data, for example double precision float, while each entry appears as a new row. This is analogous to a Microsoft Excel Spreadsheet.
- PeaksWorkspace is a special type of TableWorkspace with additional support for Single Crystal peaks.
- MDWorkspace will be covered later in this course

1.2.4 Algorithim Histories

Algorithm History

Mantid keeps the entire history of all algorithms applied to workspaces. Not only does this allow you to audit the data reduction and analysis, it also provides the means to extract a re-executable python script from the GUI.

1. Right click on HYS_11388_event and select 'Show History'. This will open up the Algorithm History window. In the left-hand side Algorithms panel click on the arrow in front of Rebin v.1 and should see the following:



This reveals the history done to this workspace, i.e. Load to load the workspace, SumSpectra, and three Rebins. To replay the history do:

2. From the main MantidPlot menu Select **View->Script Window**, this opens the 'MantidPlot: Python Window'
3. Go back to Algorithm History window and press the 'Script to Clipboard' button and close the Algorithm History window
4. Flip to the script window ('MantidPlot: Python Window') and paste what was copied to the clipboard into the script window
5. Close the HYS_11388_event-1 plot window
6. Delete the HYS_11388_event workspace from the Algorithms panel
7. To recreate the work you have just done script window execute the script via **Execute->Execute All** on the script window.
8. To verify that history has been replayed either look at history or plot spectrum again

1.2.5 Interfaces

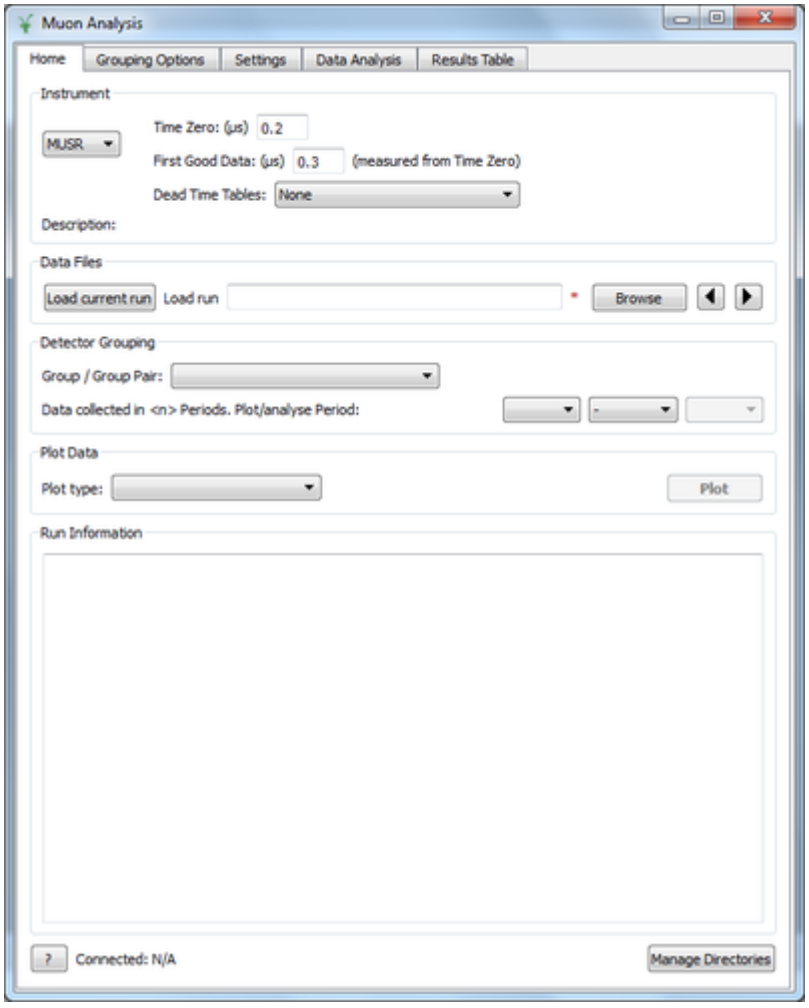
Custom Interfaces

Some analysis workflows are too complex, need more user input or need a better way to present the results than a single algorithm interface can provide. This is where custom interfaces come in. These are pluggable interfaces for Mantidplot that are loaded at run time to give a much richer interface to work with for your data analysis. Several

interfaces have now been developed to handle different aspects of the reduction and analysis workflow for data from the various scientific techniques.

An Example - The Muon Analysis Interface

The muon analysis interface is an example that uses both algorithmic analysis, plotting and curve fitting within one interface to provide a single interface that covers a large proportion of the analysis required for basic muon spin resonance.



A simple walkthrough

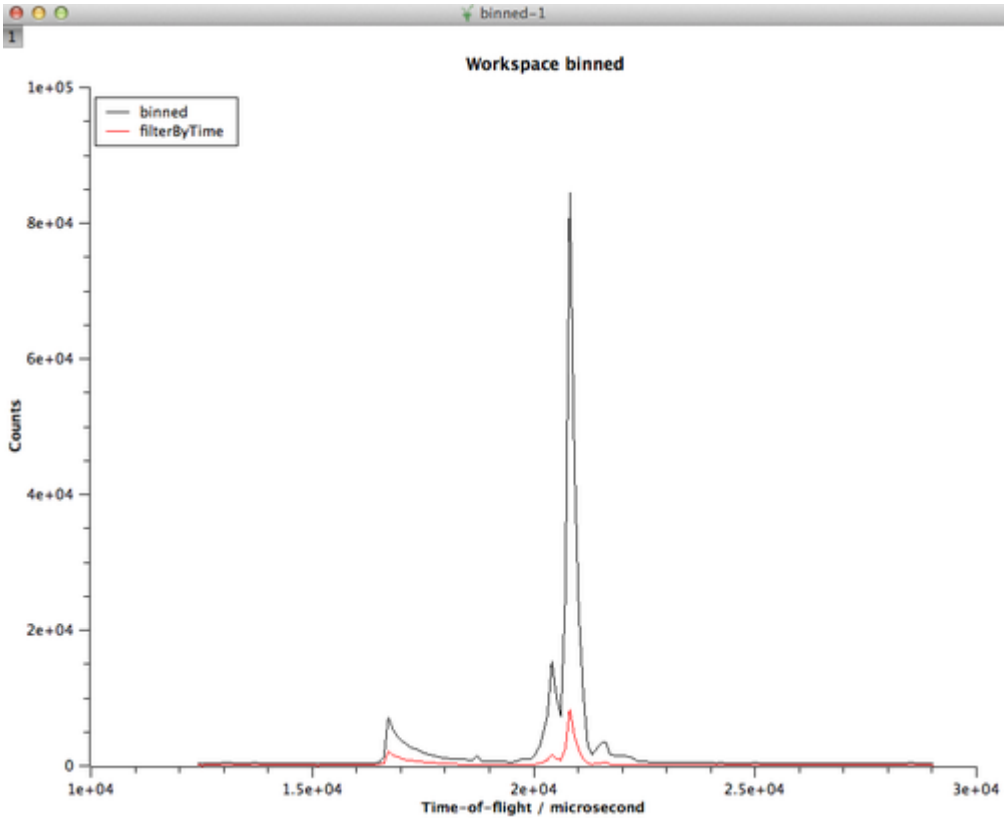
1. Start the interface with the Interfaces->Muon Analysis Menu
2. On the Settings tab set the end value to 10
3. On the Home tab Change the instrument to Emu
4. Enter 20884 in the Load Run box and hit enter
5. You will get a plot of the Asymetry of the data
6. You can change the plot data drop down box to see other plot types

7. With the Asymetry plot visible, go to the Data Analysis Tab
8. Right click the grey Functions bar and "Add Function"
9. Under Muon select ExpDecayOsc and click ok
10. Click Fit under the fit button, the fitted data is displayed on the graph
11. On the Results Table tab click the sample_magn_field and sample_temp logs and click create table
12. A table with all of the fit parameters and the selected log values is created

1.2.6 Exercises

Exercise 1

1. Load the EventWorkspace *HYS_11388_event.nxs*
2. Sum across each spectra in the workspace using the **SumSpectra** algorithm. Set the **OutputWorkspace** to be called *Sum*
3. **Rebin** this grouped workspace, specify **OutputWorkspace** to *binned* and that the bin width is *100* microseconds, and keep **PreserveEvents** ticked
4. Right-click the workspace called *binned* and choose the **Plot Spectrum** option. Once the graph is plotted, leave do not delete it
5. Events in a EventWorkspace may get filtered according to other recorded events during the experiments. At perhaps the simplest level you can filter out events between specific times. Use algm-FilterByTime for this. It has a parameter called **StartTime**, which is the start time, in seconds, since the start of the run. Events before this time are filtered out. Run **FilterByTime** with **StartTime**=*4000* and call the **OutputWorkspace** *FilteredByTime*
6. Drag the workspace FilteredByTime into the plot where workspace *binned* is plotted. What you should see now

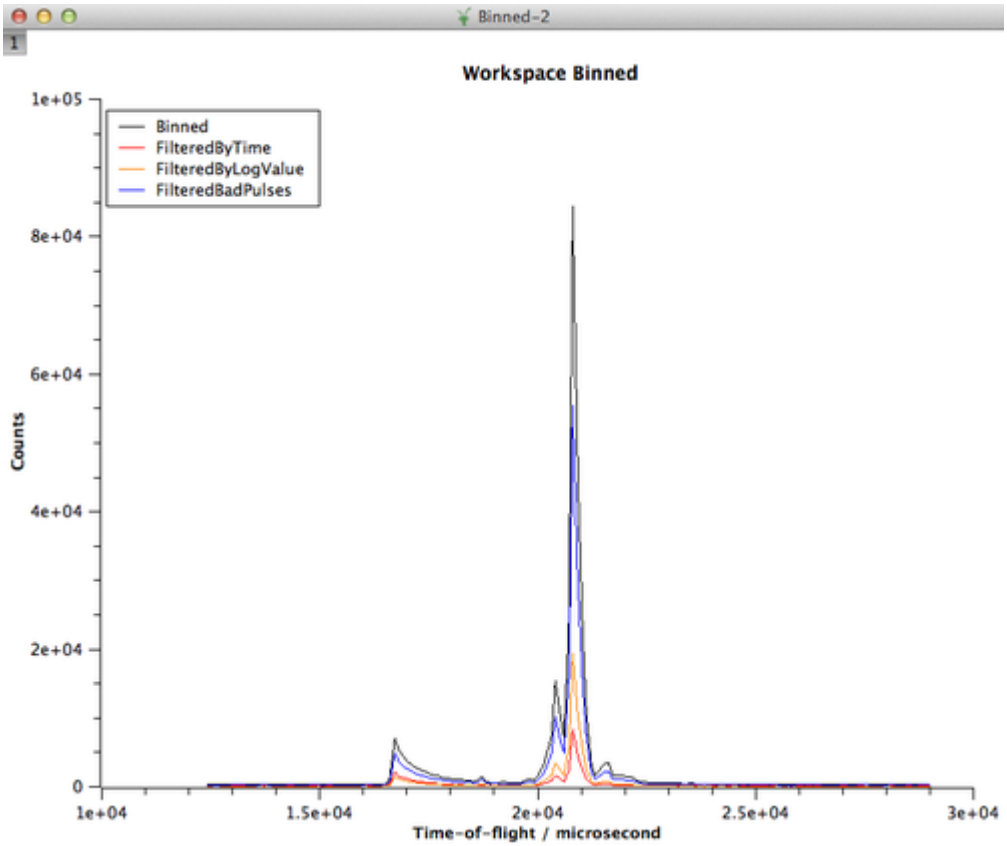


is

7. Replay by the steps used to create the *FilteredByTime* workspace from the history of this workspace

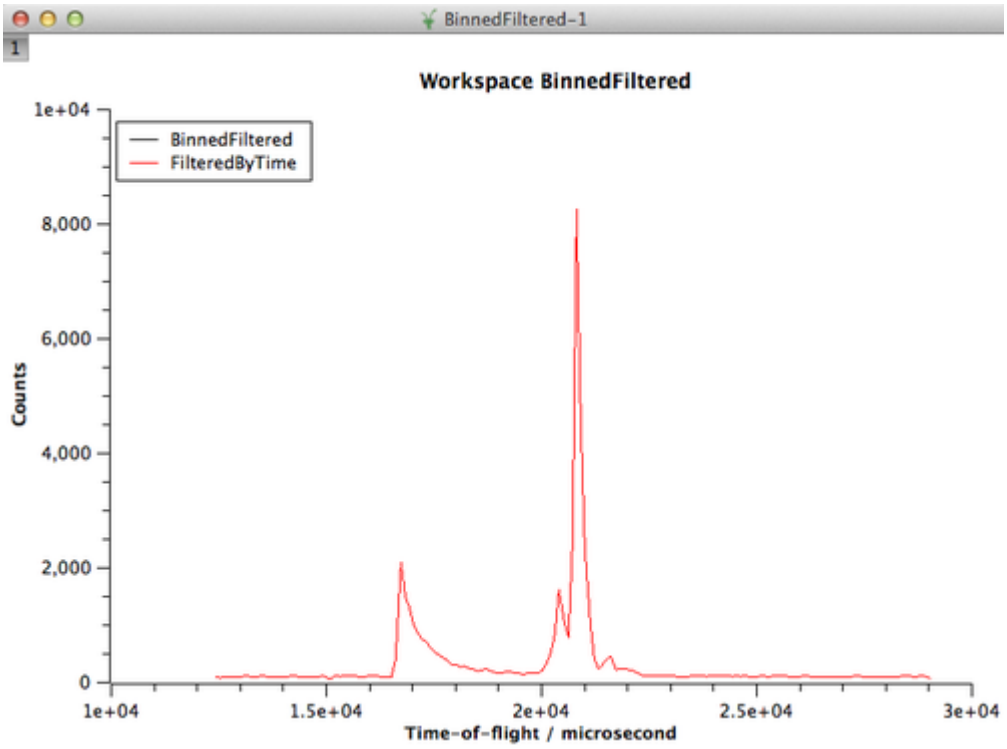
Exercise 2

1. Using the *Binned* workspace from the previous example as the InputWorkspace, use **FilterByLogValue** with **LogName**=*proton_charge*, **MinimumValue**=*17600000*, **MaximumValue**=*17890000*
2. Overplot the OutputWorkspace over your existing plots from the previous example
3. Run **FilterBadPulses** with **InputWorkspace**=*Binned* and **LowerCutoff**=*99.999*
4. Overplot the *OutputWorkspace* over your existing plots



Exercise 3

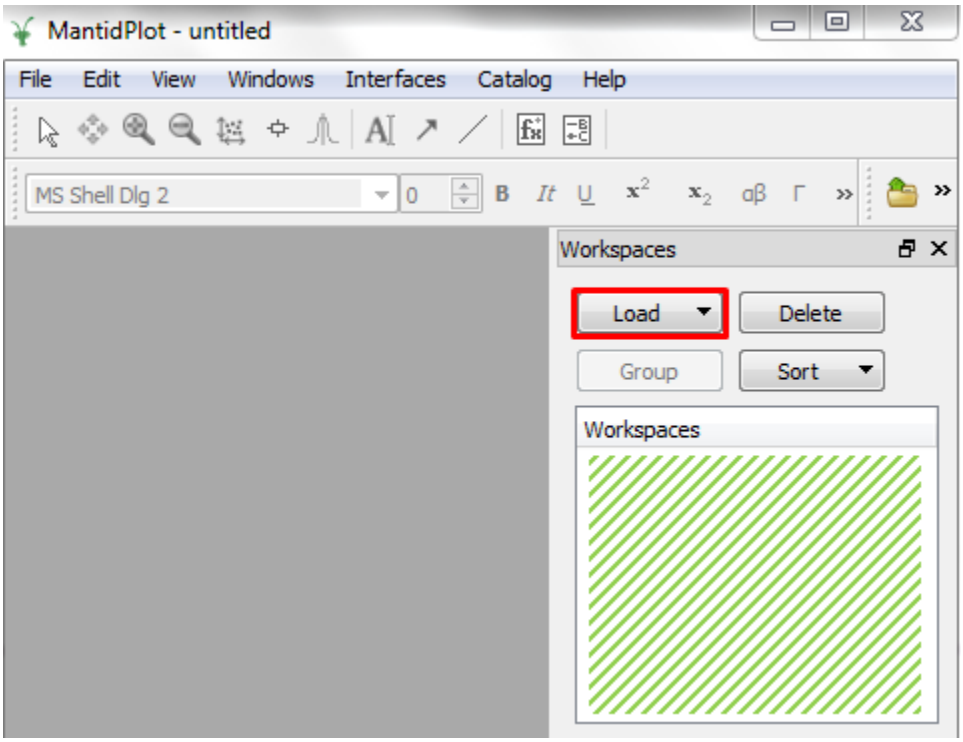
1. **Load** the same workspace used in Exercise 1, but this time perform the same filtering achieved in Exercise 1 as part of the Loading
2. **SumSpectra** on your new workspace
3. Use **RebinToWorkspace** to achieve the same binning as the existing *Binned* workspace
4. Plot both your newly rebinned workspace and *FilteredByTime* created in exercise 1 on a new plot.



1.3 Loading and Displaying Data

Sections

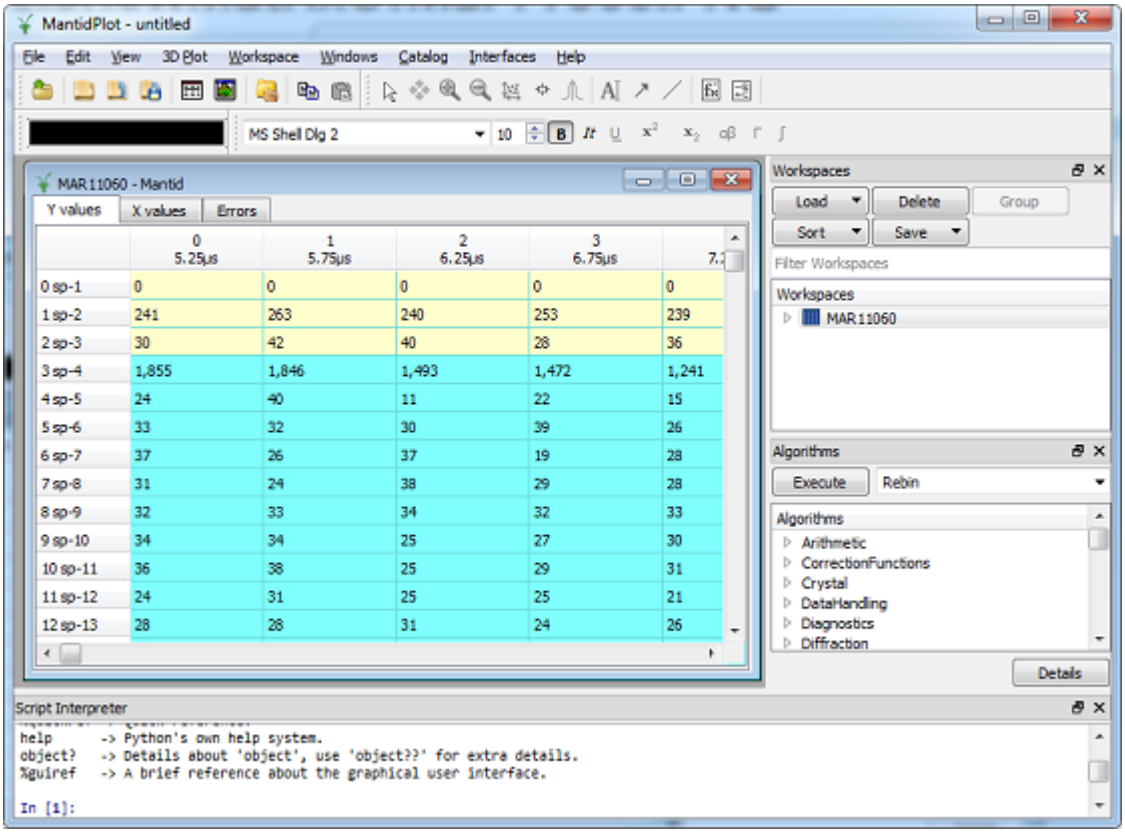
1.3.1 Loading Data



Loading a File

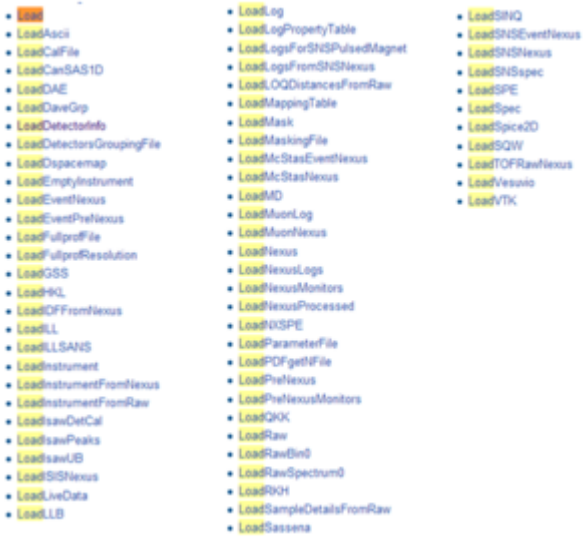
First, let's load the dataset MAR11060.raw collected on the ISIS MARI instrument:

1. In MantidPlot click on the the "Load" button (as shown highlighted in red), and select "File". This will open the Load Dialog window.
2. Browse to the location of the file MAR11060.raw.
   - If you have successfully added the directory containing this file to your data search directories (see *Setting Up and Getting Started*) then you can simply enter the filename in the File textbox.
   - Typing "MAR11060" would actually be enough for MantidPlot to find the data, but if you have set MARI as you default instrument typing just "11060" would work!
3. Mantid will suggest the OutputWorkspace name to be "MAR11060", but feel free to use whatever you like.
4. Leave the other properties alone and press the Run button. A Workspace will appear in the Workspaces list (as shown highlighted in green).
5. Click on this workspace entry and drag it into the main area to the left of the workspace. This will display the data in a matrix window named 'MAR11060 - Mantid', as shown below:



Other equivalent methods exist for loading files in MantidPlot:

- Navigating to the "File->Load->File" menu item.
- Clicking the Load File Button toolbar button.
- Using the Load Algorithm. (More on this later!)



Types of Data Files

Mantid can load many different data formats. A few examples are:

- ISIS, SNS, ILL, PSI .nexus data files.
- ISIS .raw data and log files.
- Simulated data formats.
- Ascii data, Table data, etc.
- Live data streams.

Fortunately, you don't have to learn how to use all of these Load algorithms. In fact, just one, "Load", which was used earlier when you loaded the workspace. Whenever you use "Load" Mantid takes care of the following:

- Expanding out run numbers to full file names.
- Finding the file in the data search directories, and optionally the facility archive.
- Determining the format of the file and using the correct algorithm to read it.
- Loading or summing multiple files.

Loading Lots of Data Files

You can load multiple files into mantid with a single Load command, either keeping each workspace separate, or summing the data into a single workspace:

Usage	Description	Example
Input	Result	
`,`	Load a list of runs.	`INST1,2,3`
`+`	Sum a list of runs together.	`INST1+2+3`
`:`	Load a range of runs.	`INST1:4`
`-`	Sum a range of runs.	`INST1-4`

1.3.2 The Matrix Workspace

The Workspace Toolbox

MantidPlot needs a way of storing the data it loads into something. These 'somethings' are called workspaces. You can see all of the Workspaces that Mantid has loaded in the Workspaces toolbox, together with buttons for:

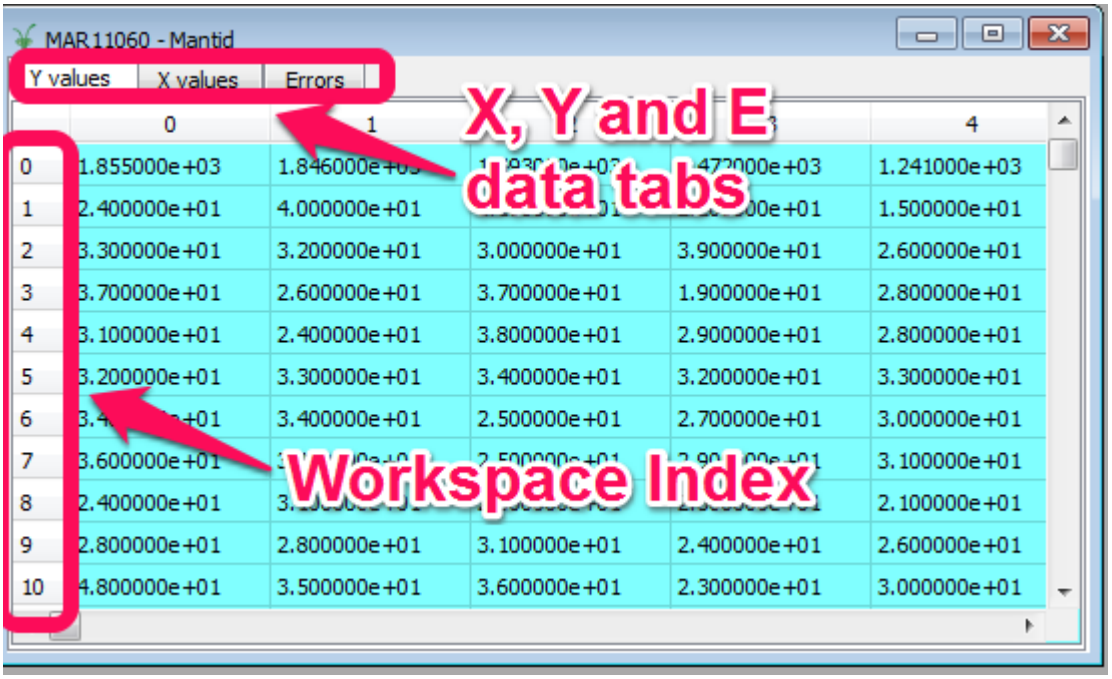
- Loading new workspaces.
- Deleting Workspaces.
- Grouping Workspaces. If a grouped workspace is already selected this will change to "Ungroup" the workspaces.
- Sorting the workspace list.
- Saving Workspaces.

Next to each workspace is a little triangle you can click to review a few more details about the workspace.

The Workspace Matrix View

The dialog with the blue background is the workspace matrix view, and can be used to look at histogram or event data. You can create one in a few ways:

- Click and drag the workspace name into the main window area (the grey bit in the middle).
- Right click the workspace name and select show data. The next dialog gives you options to limit the amount of data displayed; just click OK.
- Double click the workspace name
- Using python commands. There are more details in the Python in Mantid course.



Each row in the matrix shown shows data values of a single spectra. By flipping between the tabs you can see the X, Y and E values.

- X contains the Bin boundaries or bin centre values of the X axis. If they are bin Boundaries you will have one more column in the X tab than you do in the Y or E tabs.
- Y contains the counts found in each bin of the spectrum.
- E contains the errors associated with the counts. For most raw data this will initially be the square root of Y on loading.

The number at the left of each row we call the "Workspace Index", or WI for short, and is simply the row number as data is read into the workspace; in the same way as a spreadsheet program like Excel uses row numbers. This always starts from zero and is important as it is used quite a bit in displaying and running Algorithms on your data.

Spectra that correspond to monitor detectors appear as the first rows and are marked with light yellow background. The rows corresponding to the masked detectors (if present) will be highlighted with light grey background. If a monitor spectrum itself is masked, it will be marked with light grey just as other masked spectra. Masked bins will also be marked with light grey, except for EventWorkspaces.

Linking Workspace Index to Spectra Number

Many instruments use a unique number to refer to each of the spectra in a data file. We call this the "Spectrum Number". If you load the whole file you will often find that the Workspace Index and Spectrum Number match each other closely, just out by one. However if you only load part of a file, for example spectra 100 - 200, then the Workspace Indices will still be 0-100, but Mantid will remember the original Spectra Number of each spectrum.

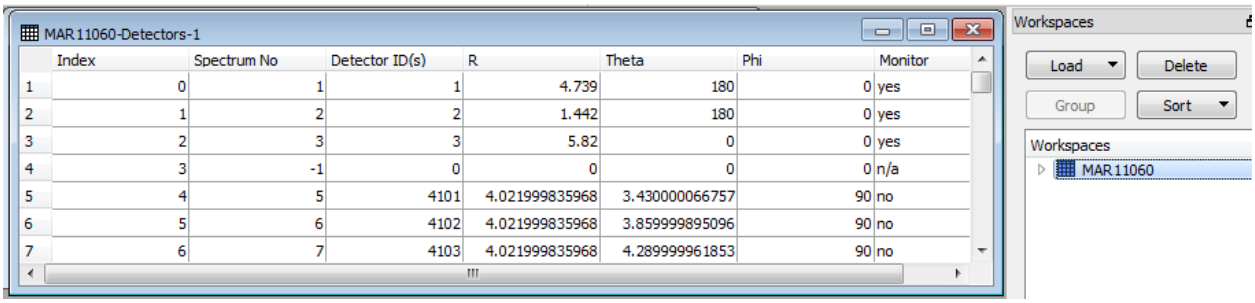
Linking Workspace Index to Detector IDs

The data in each spectrum can come from one or more detectors, and sometimes it is useful to know exactly which detectors. You can get spectra linked to multiple detectors either by hardware detector grouping or by grouping the detectors in Mantid.

There are two methods you can use to see the detector table of a workspace within MantidPlot:

- Right-click on the workspace name and select "Show Detectors".
- Right-click within the workspace matrix and select "View Detectors Table".

Either of these methods will display a table containing the Workspace Indices, Spectra Numbers, Detector IDs and locations of the detectors, together with a flag showing which are monitors. If a spectrum contains a group of detectors then the position shown will be the average position of those detectors.



	Index	Spectrum No	Detector ID(s)	R	Theta	Phi	Monitor
1	0	1	1	4.739	180	0	yes
2	1	2	2	1.442	180	0	yes
3	2	3	3	5.82	0	0	yes
4	3	-1	0	0	0	0	n/a
5	4	5	4101	4.021999835968	3.430000066757	90	no
6	5	6	4102	4.021999835968	3.859999895096	90	no
7	6	7	4103	4.021999835968	4.289999961853	90	no

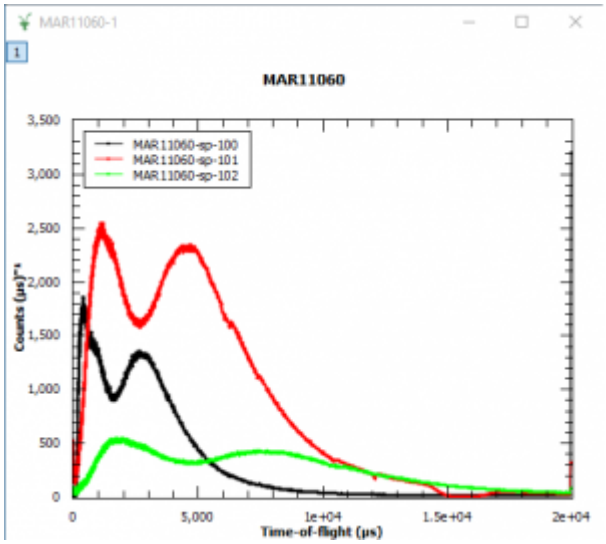
In the example above, the first three spectra correspond to data from monitors. The fourth spectrum (with a Spectrum Number of -1) loaded from the MAR11060 run is for some reason not present in the Instrument definition, and the remaining rows that are visible show histograms wired up to detectors with spectrum number 5-7 or equivalently detectors with IDs of 4101-3.

1.3.3 Displaying 1D Data

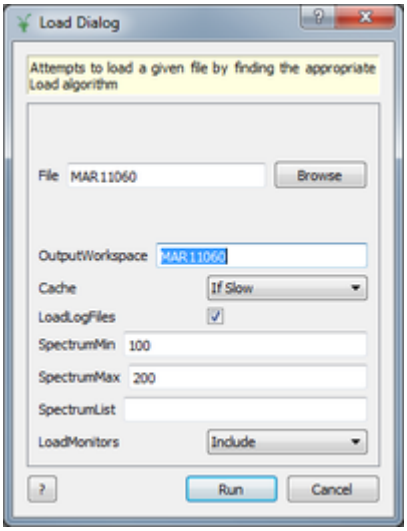
The Simplest Plot

1. Load "MAR11060".
2. Right click the workspace in the workspace list.
3. Select "Plot Spectrum . . . ".
4. In the dialog that appears enter "1-3" in the spectra number box.

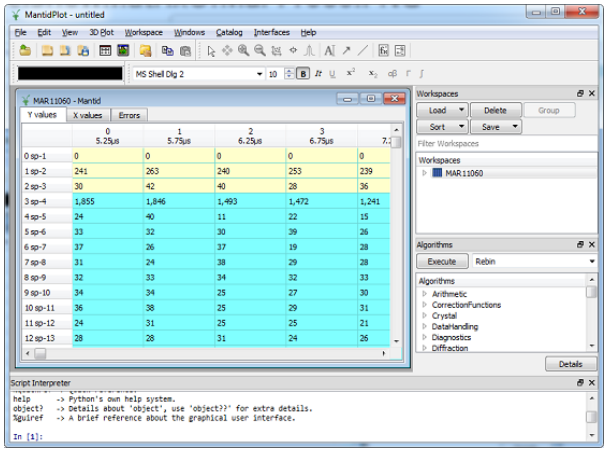
You should get a plot like this with three spectra corresponding to spectra 1-3 (which in this case these are the monitors on MARI):



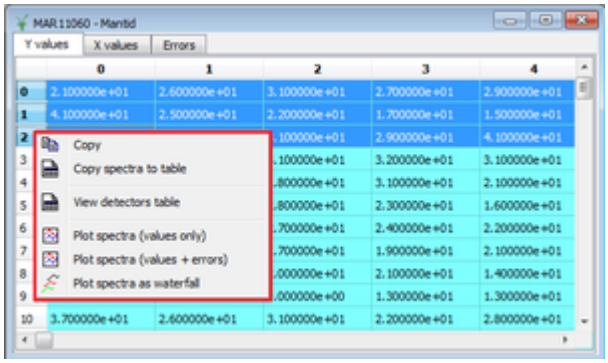
Another way to plot



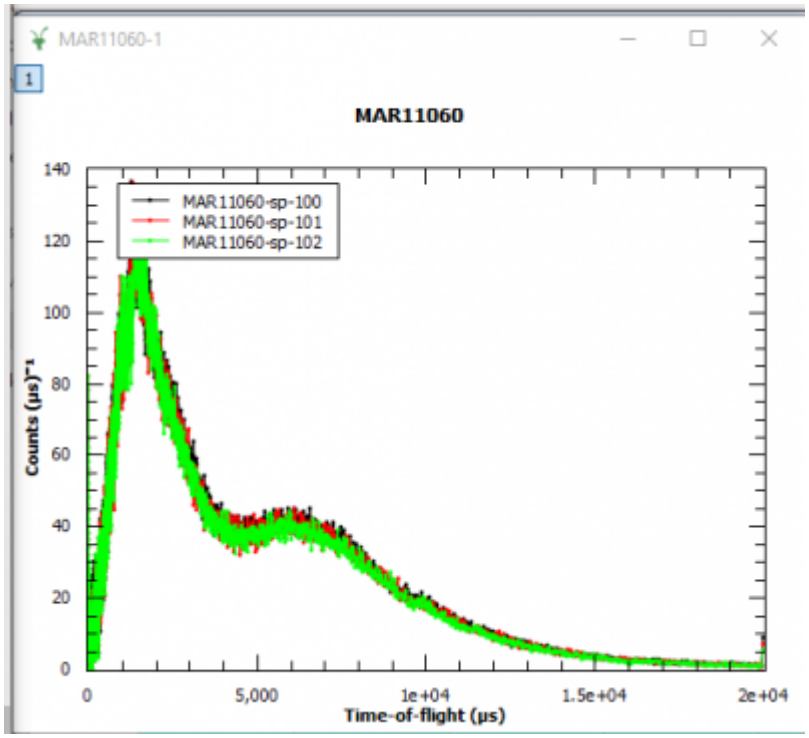
1. Load MAR11060.raw, but this time set SpectrumMin to be “100” and SpectrumMax to be “200”. Because the values are inclusive, we are actually loading 101 spectra, starting at 100.
2. Display the matrix window to view the data:



1. In the matrix window highlight rows 0-2 and bring up the right-click menu:



1. Select multiple rows using one of the following methods:
   - Click the first index label, and then hold shift while selecting the last.
   - Click and drag down the index labels to select as many as you want.
   - Click while holding Ctrl to select / deselect individual rows.
2. You will presented with 3 options for plotting the selected spectra, which are plotting with and without errors or plotting the spectra as a waterfall. The latter option is only available if more than one spectra has been selected. Select “Plot Spectra” (showing the values only):



Notice the legend entries “MAR11060-sp-100”, . . . , “MAR11060-sp-102”, where “sp” is shorthand for spectrum.

Adding a curve to an existing plot

There are two ways to add spectra to an already existing graph, either from the same workspace of from another.

Drag and Drop

1. Drag the Workspace from the Workspace List and drop it onto the graph you want to add the curve to.
2. If the Workspace contains more than one spectra you will be asked which you wish to add to the plot.

From Another Plot

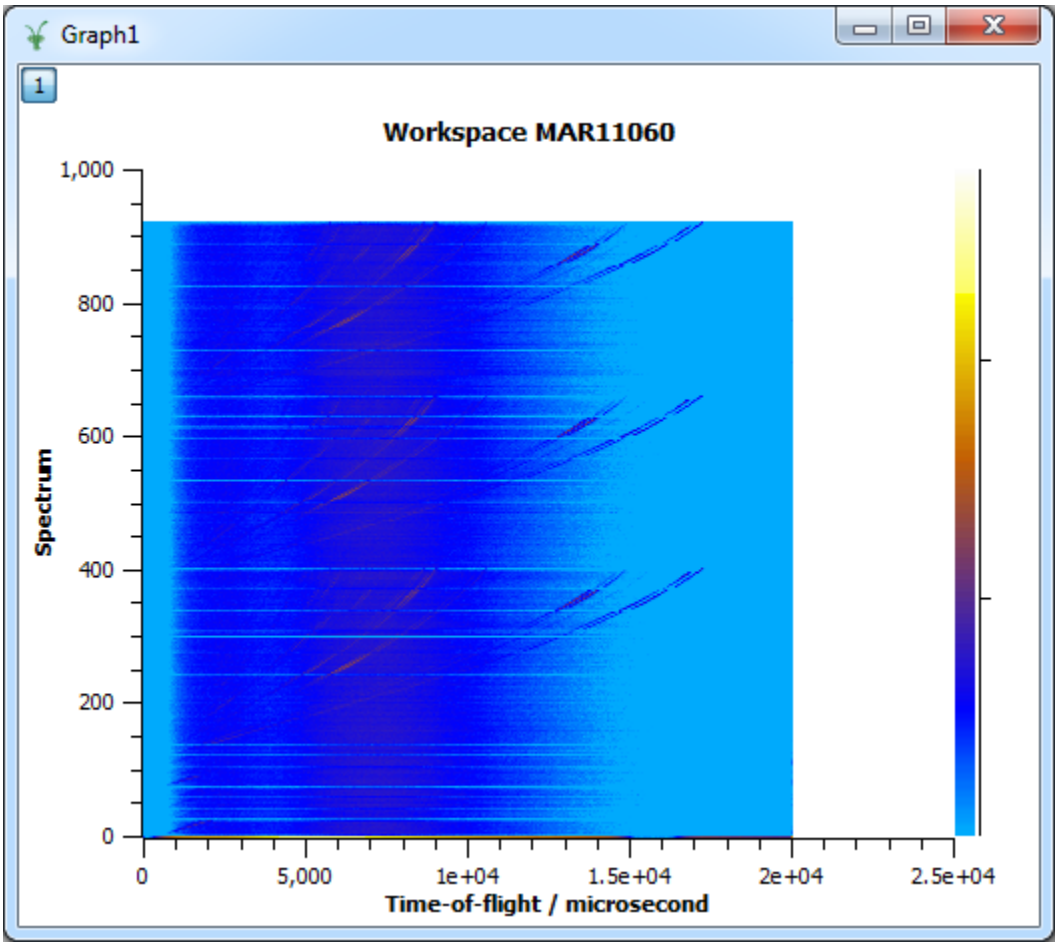
1. Create a plot containing the spectra or curve you want to import into your destination plot.
2. In the destination plot double click the “1” in the top left.
3. In the “Add/Remove Curves Dialog” that appears, select the data you want from the available data, and click “->” to add it to your plot.

1.3.4 Displaying 2D Data

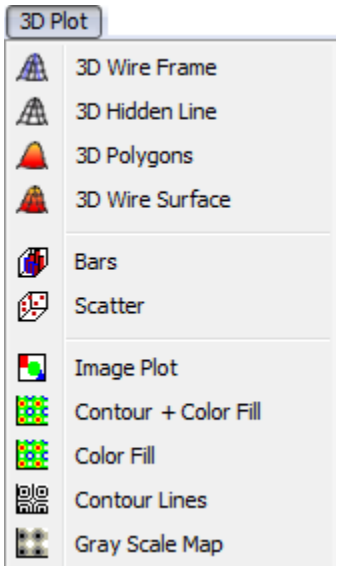
Plotting All Spectra

We have previously seen how to plot one or more rows from a dataset. Here we will show how to visually inspect entire datasets.

1. Load the MAR11060 dataset.
2. Right click the workspace in the workspace list and select "Colour Fill Plot". This will create a fairly boring blue display of the data, because one spectrum contains much higher counts than any of the other spectra, and the color scale has adjusted accordingly.
3. To make the colour fill plot more sensitive to its smaller features, right-click on the plot and select "Colour Bar->Log Scale".
4. Finally, right-clicking on the colour bar and selecting 'Rescale to show all' will give a more meaningful result:



Plotting from the Matrix View



As long as this matrix is in focus, the MantidPlot menubar will contain a menu called "3D Plot". This contains many options for plotting full datasets. The "Color Fill" option is one of the most useful among them.

Changing the Colour Map

There are several colour maps to choose from that are already installed with Mantid:

1. Double-click within the data of the plot (or right-click and select "Properties").
2. In the "Plot Details" dialog that appears select "Layer Details" in the left-hand pane.
3. Select "Custom Color Map" in "Colors" tab, click the "Select ColourMap" button, and select any of the colour map files.

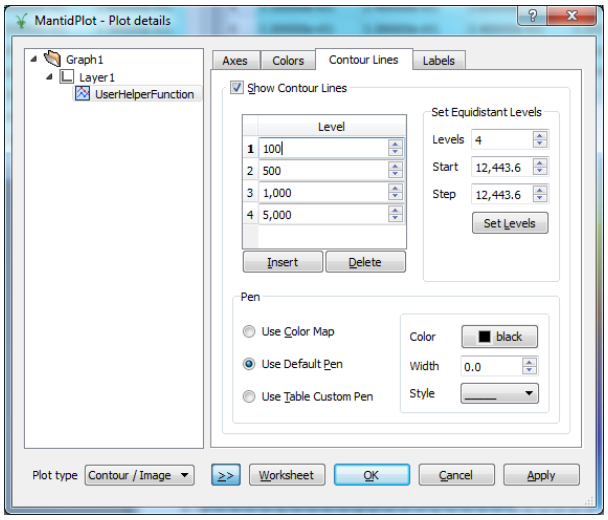
Creating Your Own Colour Map

If you don't like any of the colour maps you can create your own. The files are just 256 entries of Red, Green and Blue values (0-255 for each).

```
...
120 136 260
124 140 260
128 144 260
128 144 260
132 148 260
136 152 260
140 152 260
140 156 211
144 160 211
```

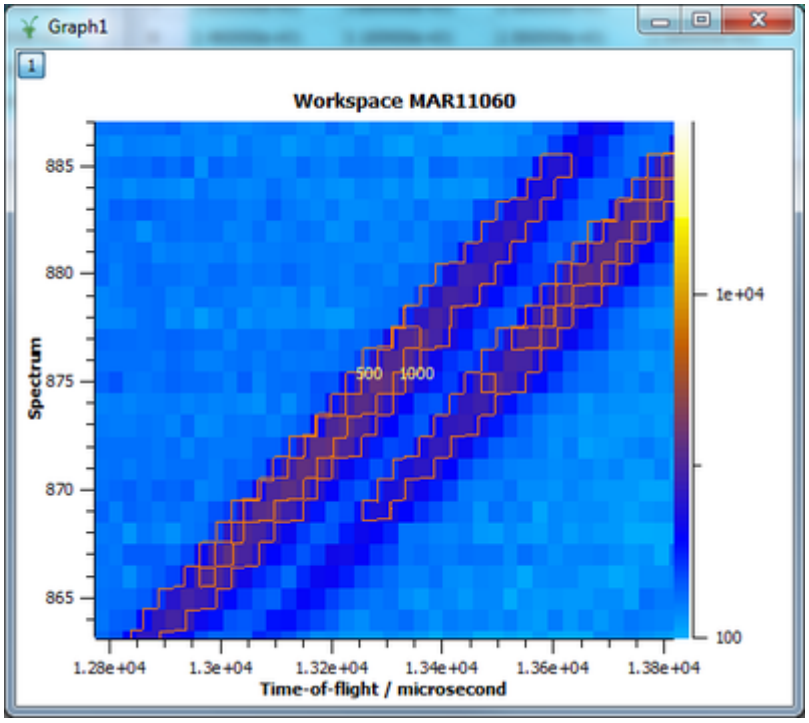
```
148 160 211
148 164 211
152 168 211
156 168 211
160 172 211
...
```

Contour Lines



You can plot contour lines onto your colour maps.

1. Double-click within the data of the plot (or right-click and select “Properties”).
2. In the “Plot Details” dialog that appears select “Layer details” in the left hand pane.
3. Select the “Contour Lines” tab.
4. Here you can set the values for the contours; the pen contour, line thickness, etc. according to the “Plot Details” dialog example shown
5. Click “OK” and the contour lines will appear on your plot:



Spectrum Viewer

The Spectrum viewer is a useful way to investigate 2D image data. You can rapidly look through the spectrum and bin data for any point in the 2D map. To launch the Spectrum Viewer just right-click on the MAR11060 Workspace and select 'Show Spectrum Viewer':

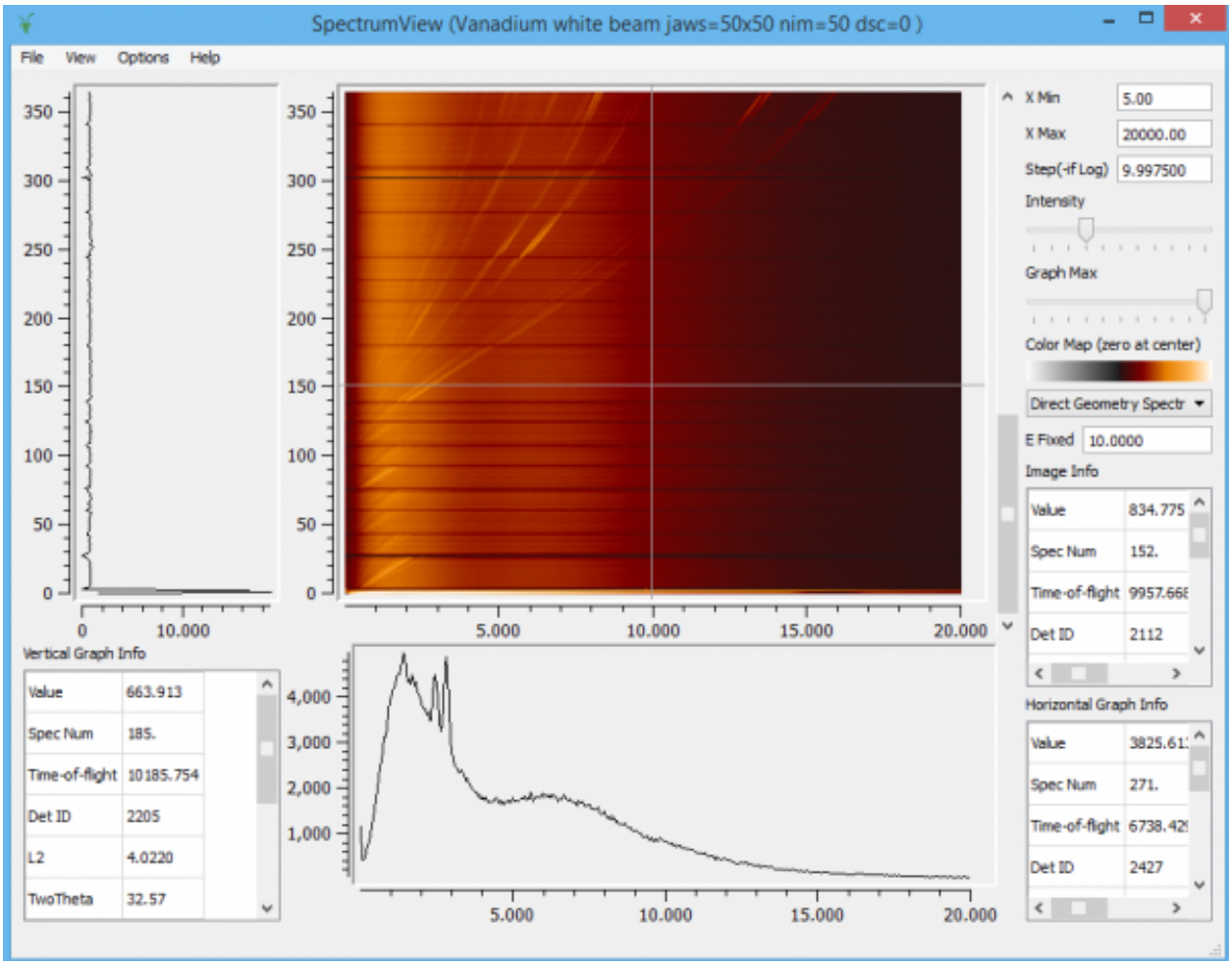


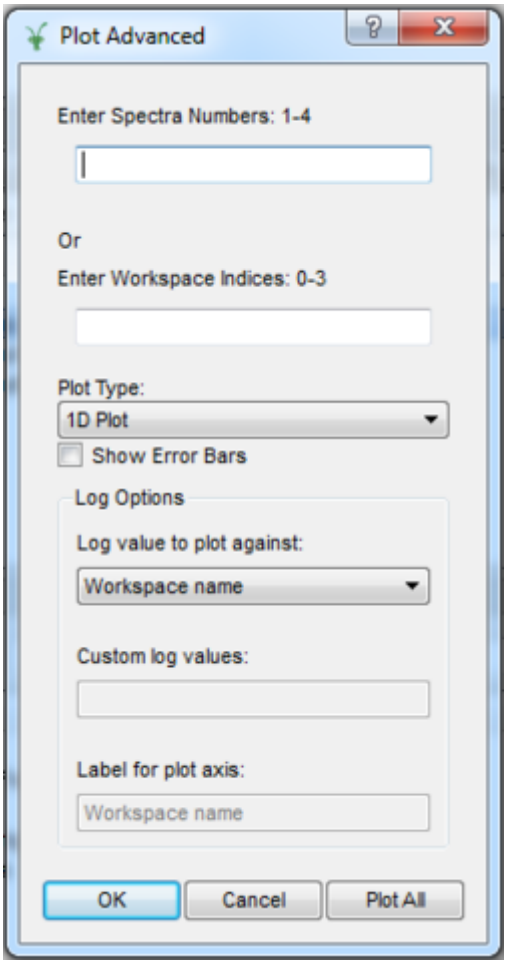
Fig. 1: centre | 600px

1.3.5 Plotting Multiple Workspaces

Plot Advanced

1. Run the Python script "TempInGroup.py" found in the data directory, by opening it in the Mantid Script Window and selecting Execute->Execute All
2. Right click on the workspace "TestGroup"
3. Select "Plot Advanced . . . "

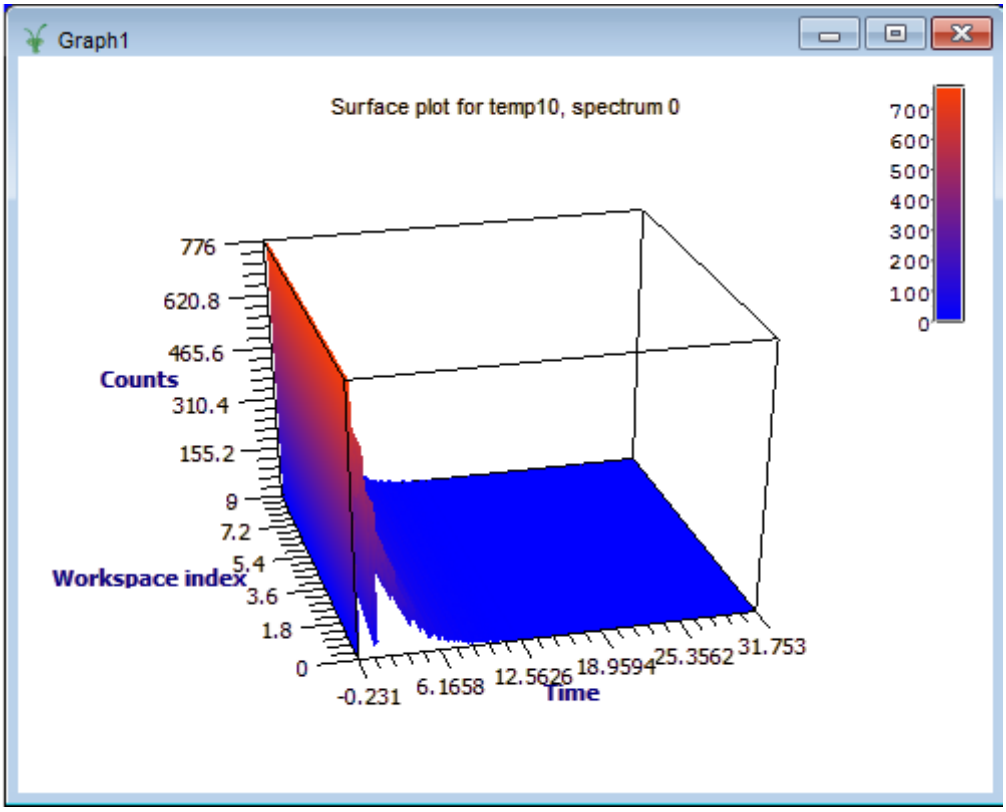
Then we get the following dialog box:



Surface Plot

1. Select “Plot Type” of “Surface Plot of Group”
2. Press “Plot All”

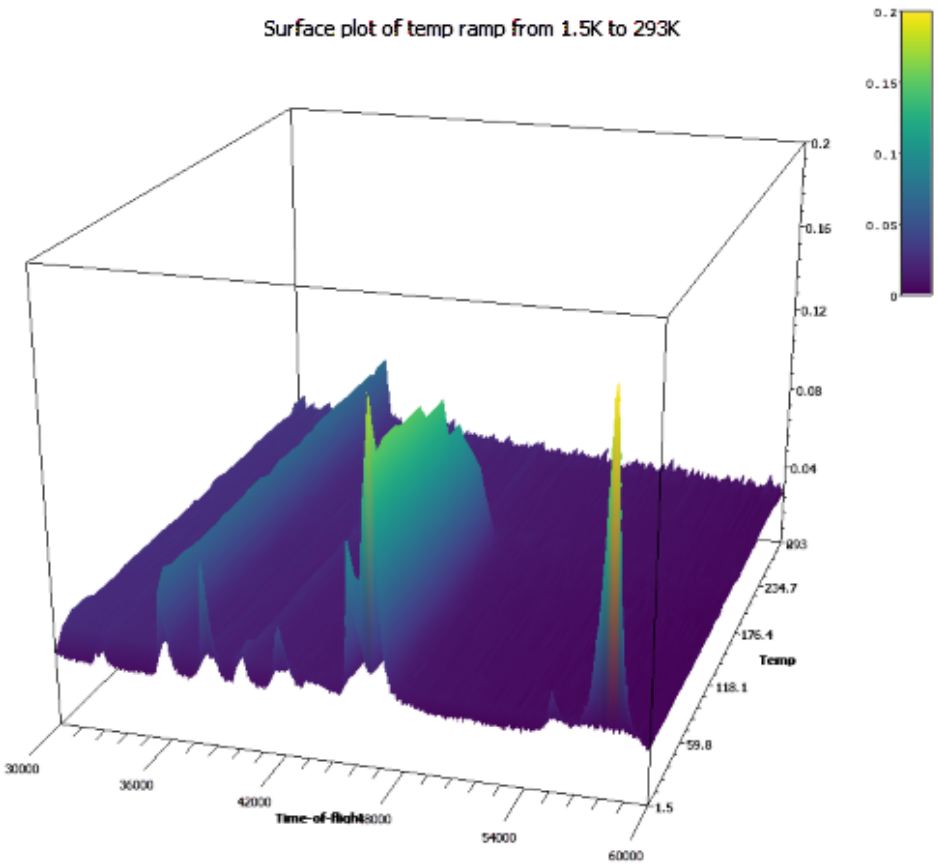
Then we get the following surface plot:



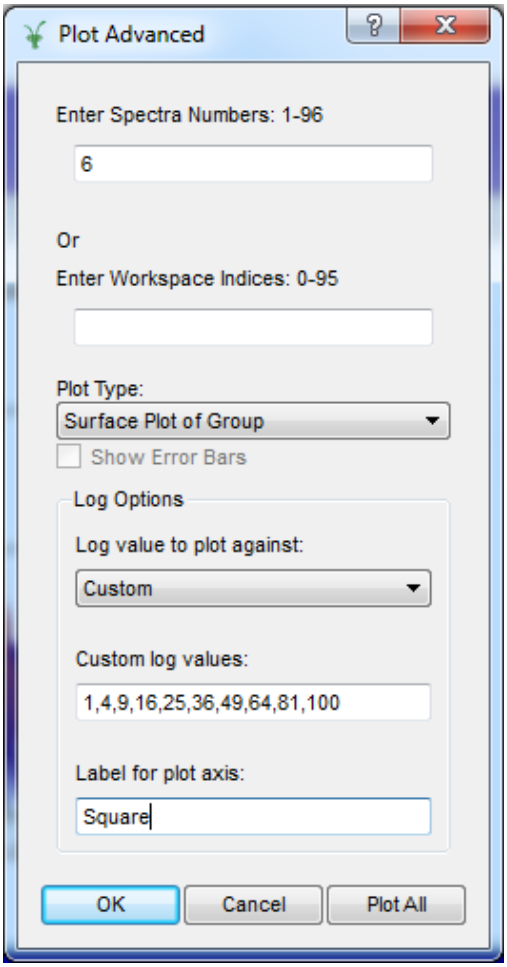
Plot with Value of Chosen Log

1. Do another advanced plot on “TestGroup”.
2. Select surface plot again.
3. In “Log value to plot against” select “Temp”.
4. Press “Plot All”

and the “Temp” log appears on an axis in place of workspace index:

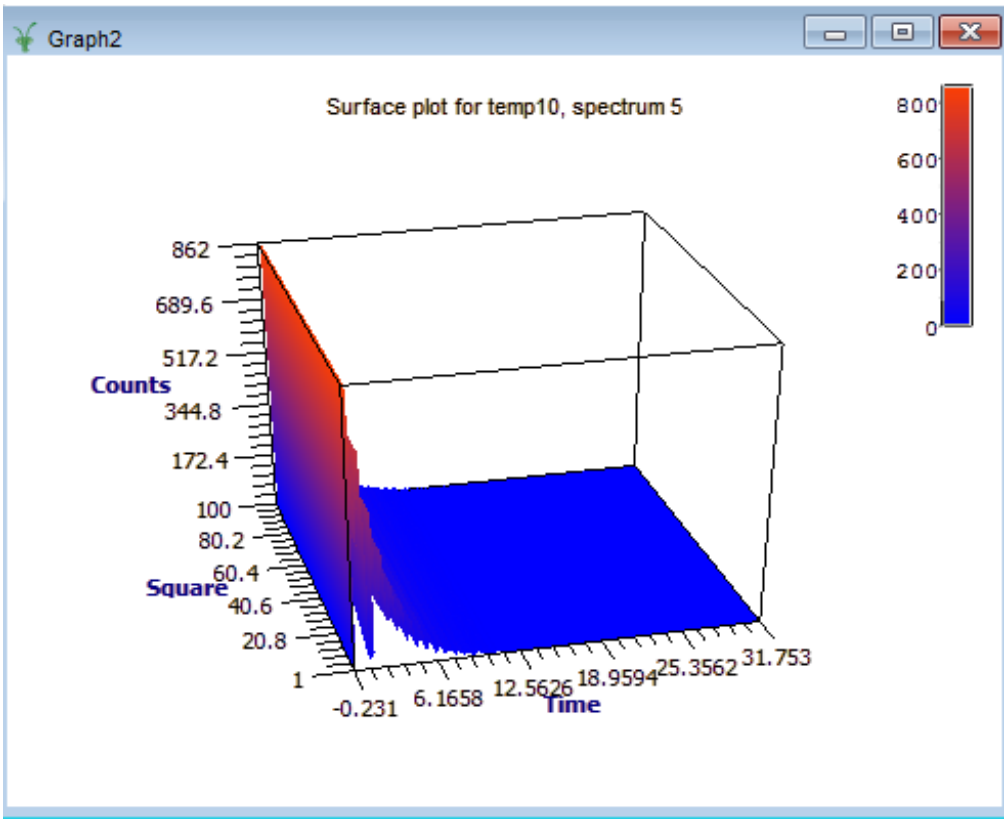

Surface plot of temp ramp from 1.5K to 293K
0.2
0.15
0.1
0.05
0
0.2
0.16
0.12
0.08
0.04
293
234.7
176.4
118.1
59.8
1.5
Temp
30000
36000
42000
Time-of-flight
54000
60000

Plot with Custom Log



1. Select Plot advanced and fill in as shown above
2. Press “OK”

Then one gets the following with custom log “Square” shown in an axis in place of Workspace index:



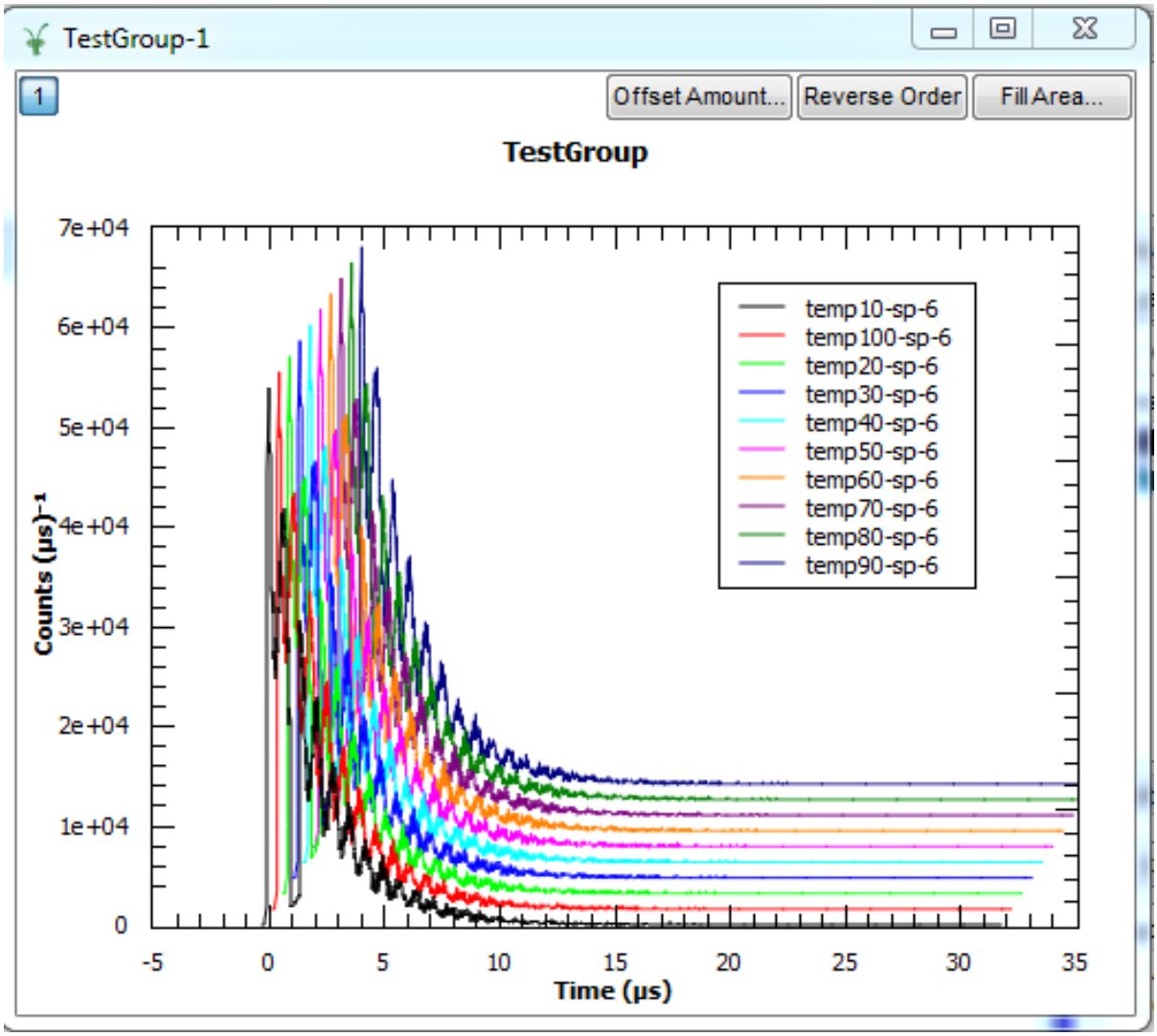
Contour Plot

A contour plot can be generated in the same manner as surface plot, but with "Contour Plot of Group" as the "Plot type" instead of "Surface Plot of Group".

Waterfall Plot

1. Select "Plot Advanced" on the group "TestGroup"
2. Enter spectra number 6
3. Select "Plot Type" of "Waterfall Plot"
4. Press OK

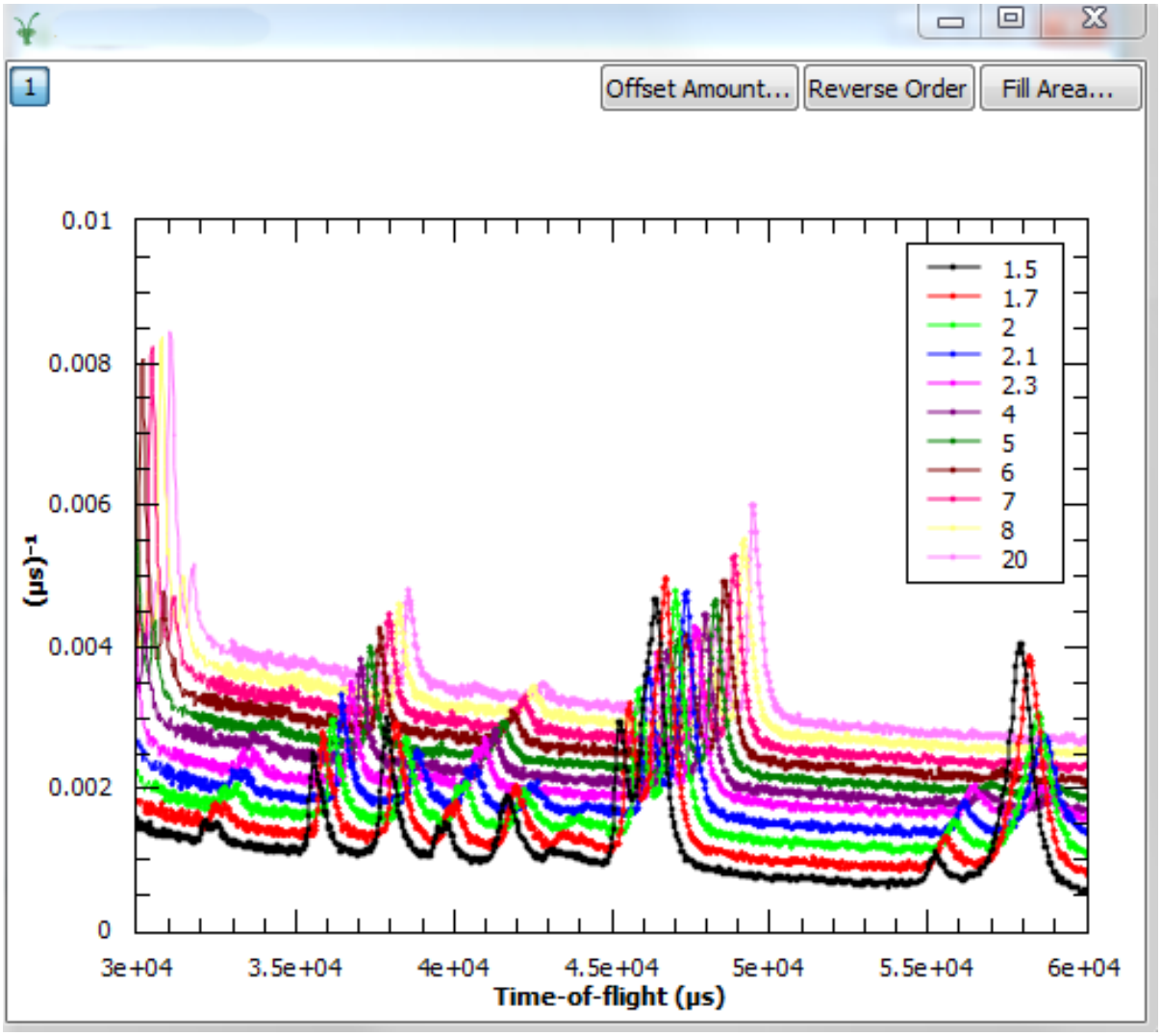
Then one gets (after moving the legend to a convenient place):



Plot with Value of Chosen Log

1. Select “Plot Advanced” on the group “TestGroup”
2. Enter spectra number 6
3. Select “Plot Type” of “Waterfall Plot”
4. In “Log value to plot against” select “Temp”.
5. Press OK

Then one gets (after moving the legend to a convenient place):

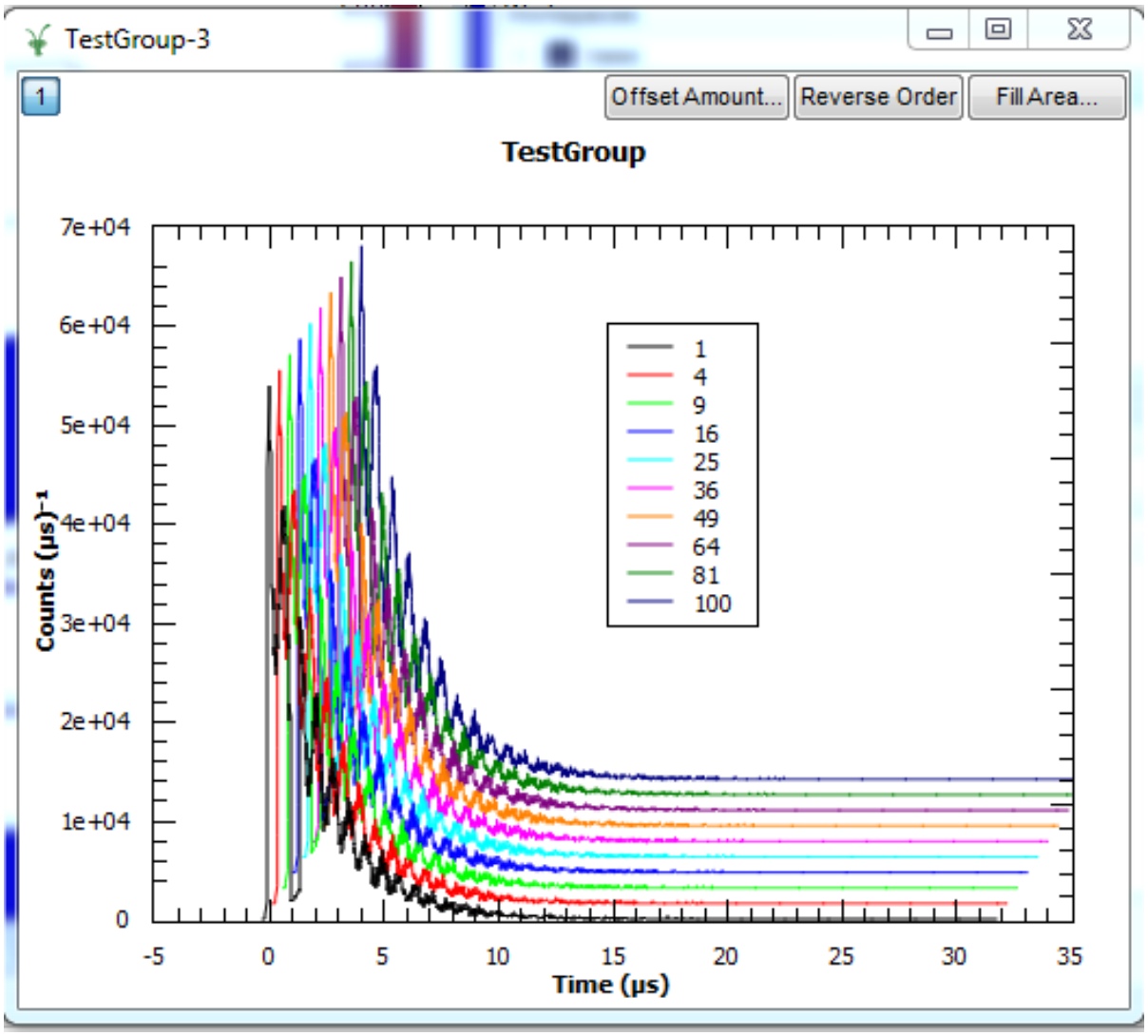


The values of the selected log "temp" are shown in the legend.

Plot with Custom Log

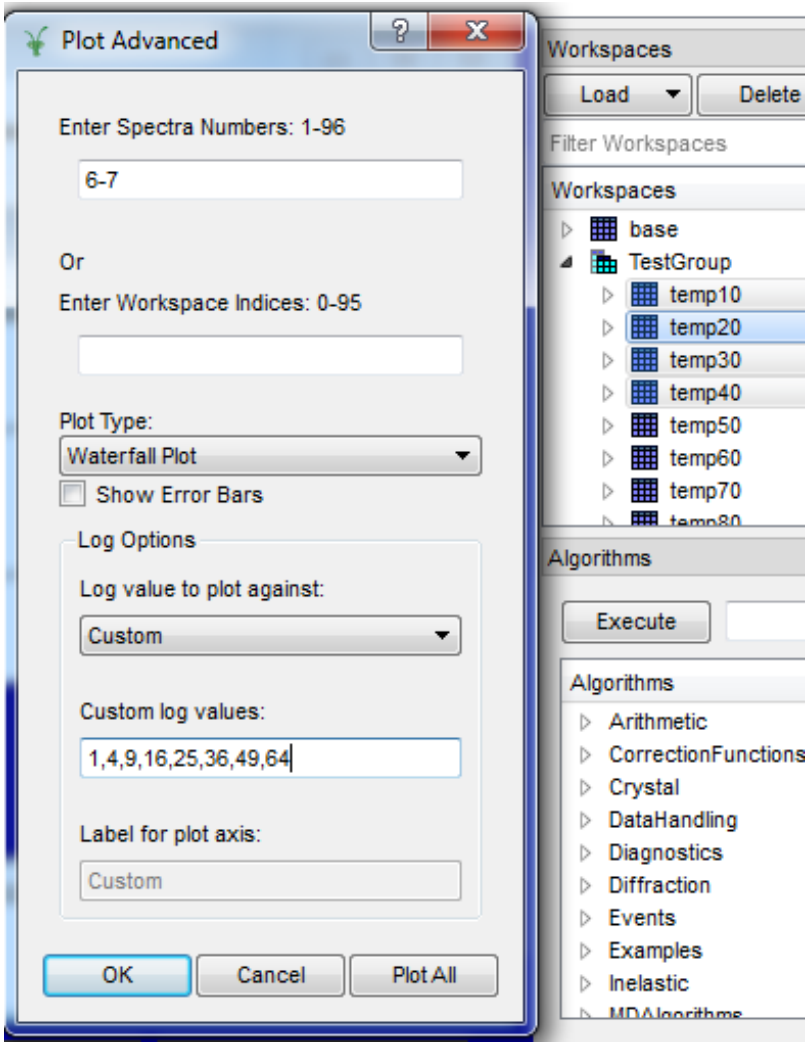
1. Select "Plot Advanced" on the group "TestGroup"
2. Enter spectra number 6
3. Select "Plot Type" of "Waterfall Plot"
4. In "Log value to plot against" select "Custom".
5. Add custom log values "1,4,9,16,25,36,49,64,81,100".
6. Press OK

Then one gets (after moving the legend to a convenient place):



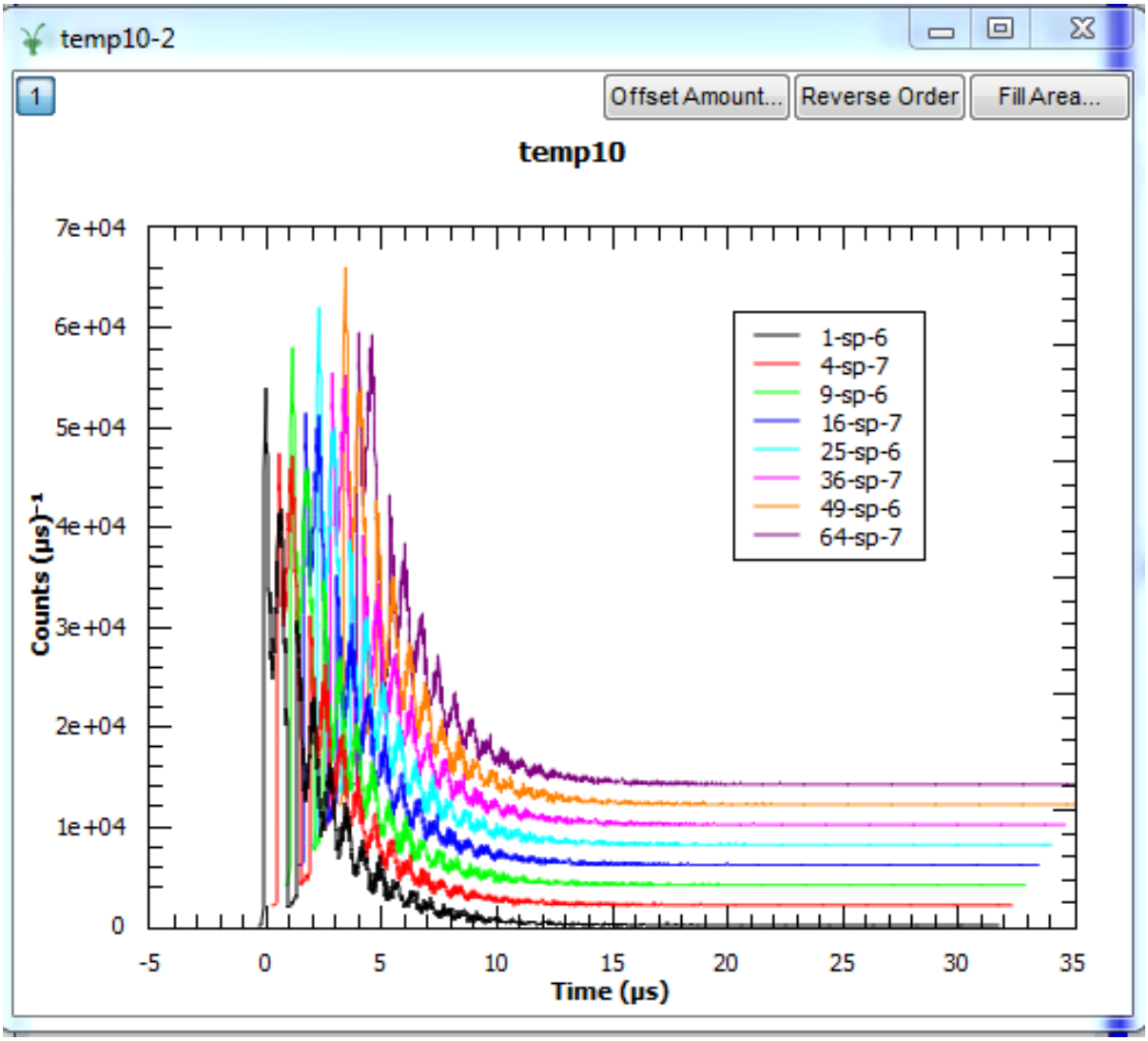
The custom log values are shown in the legend.

Plot of 2 spectra with Custom Log



1. Select 4 workspaces and fill in Plot Advanced as shown above
2. Press “OK”

Then one gets (after moving the legend to a convenient place):



Here the legend shows both the log value and the spectrum number.

1D and Tiled Plots

Simple 1D plots and Tile Plots can be done in a similar manner as Waterfall plots, but with the appropriate choice of "Plot Type".

1.3.6 Formatting Plots

Floating and Docking a Window

By default on Windows a plot window will be floating, meaning it can be dragged outside the main MantidPlot work area. For Linux versions, windows do not float by default as they have a tendency to get lost behind the main application window. The state of the selected window can be controlled using the "Windows Menu".

1. Use "Windows->Change to docked". This will dock this plot window within MantidPlot.
2. Use "Windows->Change to floating" to float it again.

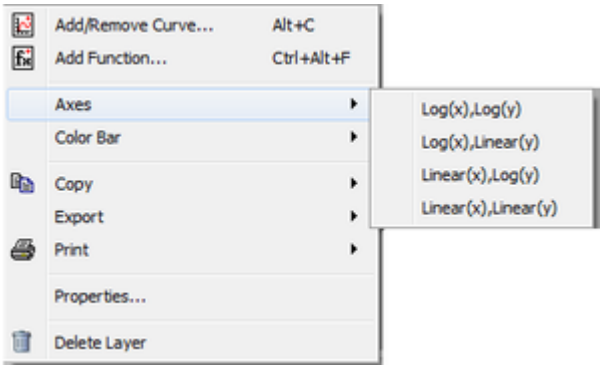
The Plot Toolbar

This toolbar is only active when a plot window is selected.

Adjusting the Properties of Plot Windows

Many aspects of these graphs can be adjusted to fit how you want to display the data. Just double-click on the item you want to change. For example, the following are editable:

- The graph title.
- The legend text. You may also move the legend by selecting and dragging it.
- The axes labels.
- The axes themselves. You can change the range, scaling and gridlines, or even introduce axis breaks.
- A curve allows you to set the line colour and styles.
- Double-clicking the "1" in the corner allows you to remove curves or add them from other graphs.



Quickly changing to Log axes

We have a fast way to change the axes between log an linear as this is such a common operation.

1. Right-click within the plot.
2. Select "Axes" to quickly switch the X and Y axes.
3. Select "Color Bar" to quickly switch the colour bar scaling.

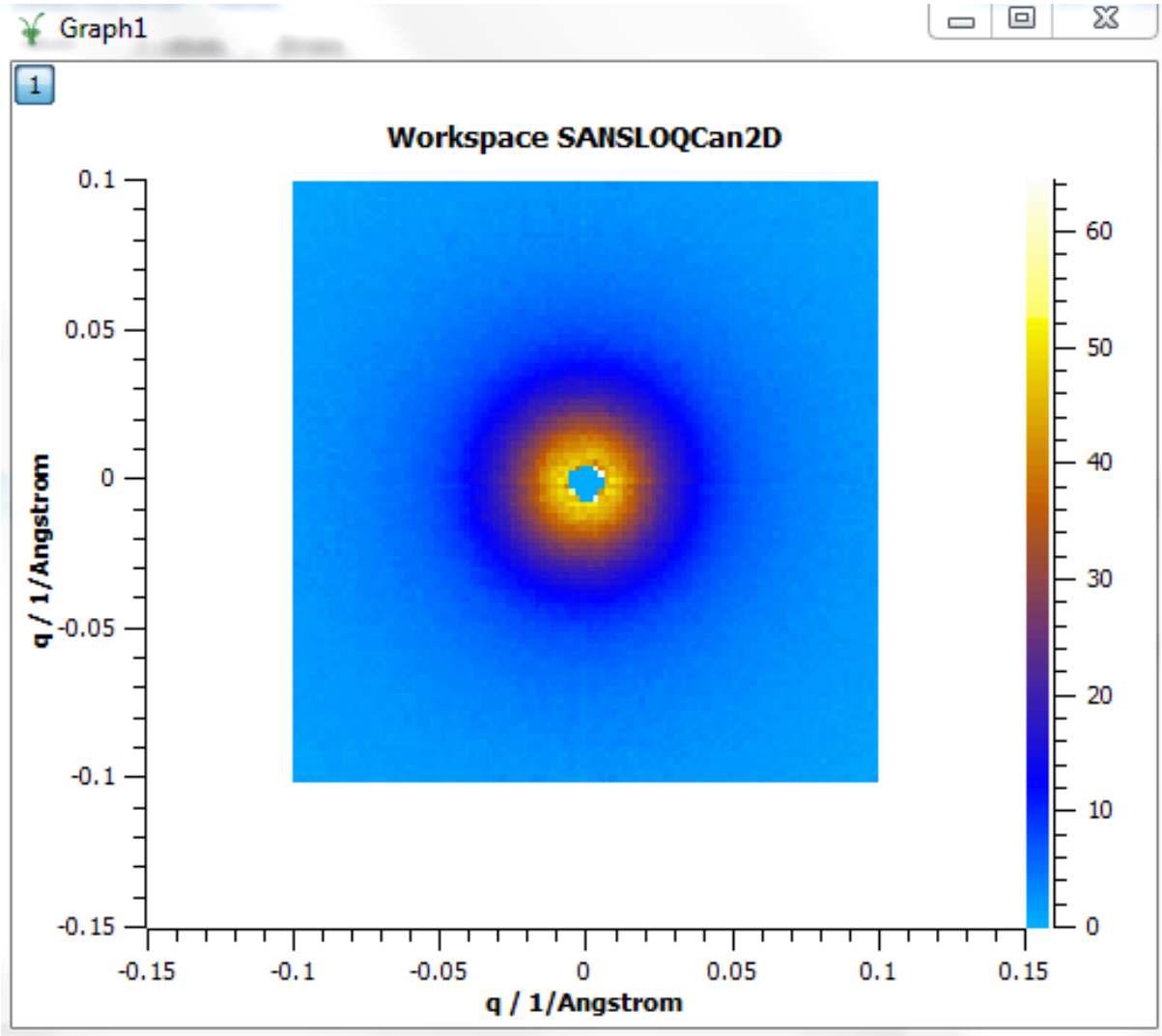
1.3.7 Exercises

Part 1

1. Load the File "GEM38370_Focussed_Legacy.nxs".
2. Plot spectra 2-7 (all of them).
3. Edit the d-spacing axis range, or zoom into the range of 0-10 angstroms.
4. Try changing the X-Axis to log scaling.

Part 2

1. Load the SANSLOQCan2D.nxs data. This is the output of a 2D SANS data reduction. Although a Workspace2D is mainly designed to store spectra, this is just the default, and in this data the loaded axes are momentum transfer Qx and the scattering cross section, and the 'spectrum' axis is momentum transfer Qy.
2. Plot this data as a colour fill plot to get the following result:

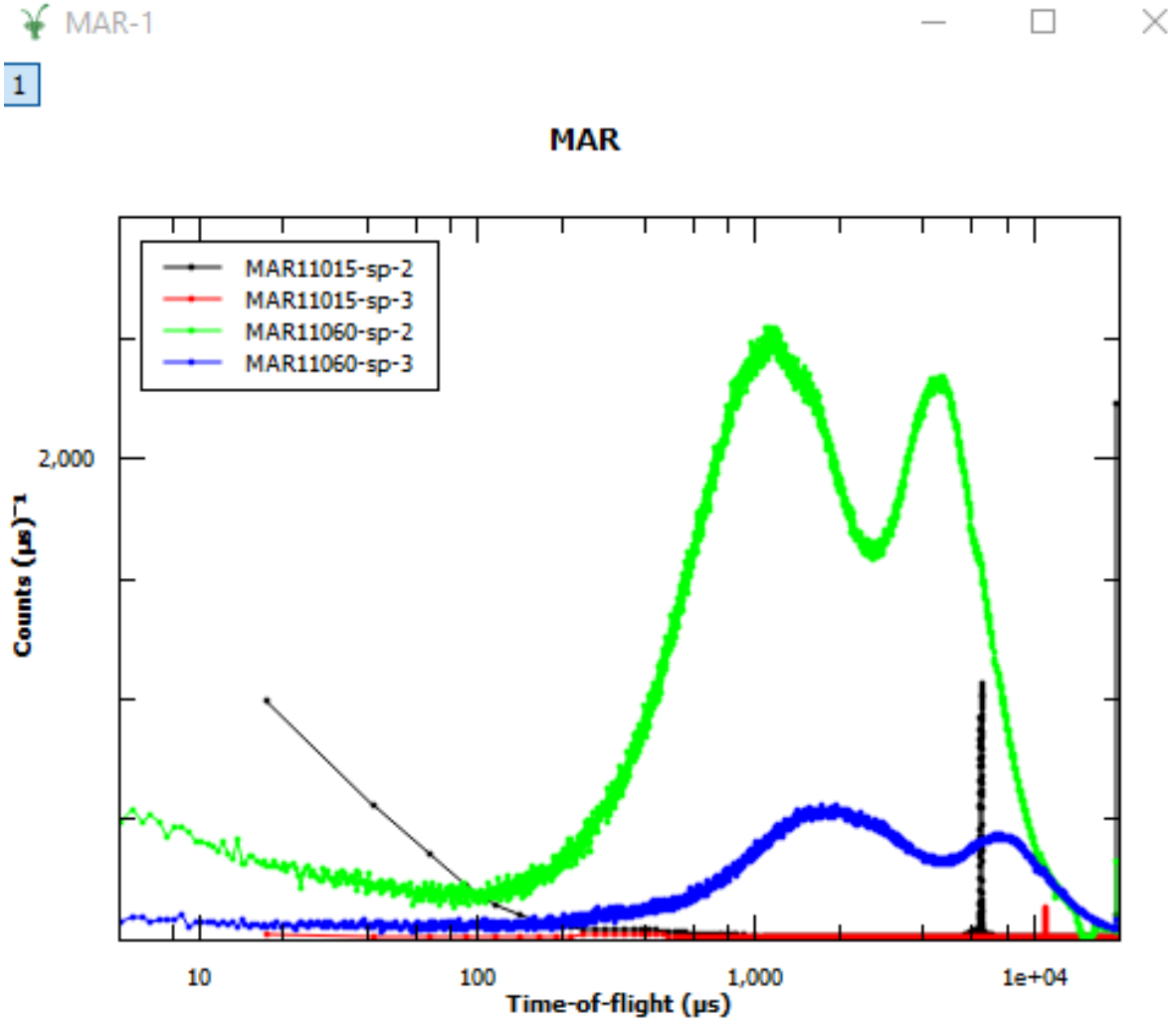


1. Change the colour bar axis to logarithm.
2. Change the Colour Map to Jet.
3. Create Contour Lines and labels in White at 20, 30 and 40.
4. Zoom in to see the results of your work.

Part 3

1. Using the workspaces MAR11015 and MAR11060 try to reproduce the plot below.
   - You will need to group the two workspaces together and then right-click on the group to plot spectra 2-3.

- Adjust the axes to use log(x), linear(y) scaling



1.4 Connecting Data To Instruments

Sections

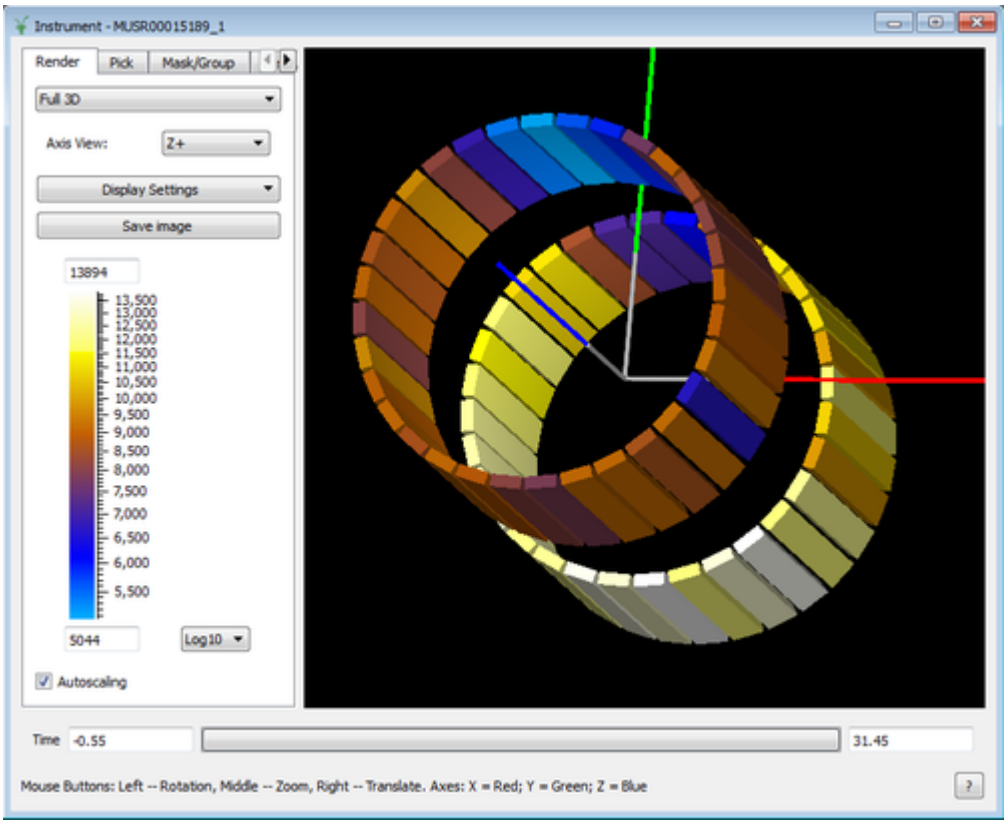
- *Introduction*
- *Displaying And Navigating*
- *Investigating Data*
- *Masking And Grouping*
- *Instrument Tree*
- *Exercises*

1.4.1 Introduction

Introduction

The Instrument View is a widget for displaying instrument geometry. It consists of three elements:

- the controls panel
- the graphical display
- the data integration slider



The controls in the controls panel are placed into four tabs according to their functions

- Render
- Pick
- Mask/Group
- Instrument Tree

The display window show the instrument detector coloured according to the integrated counts in a workspace. The integration range is adjusted using the data integration slider.

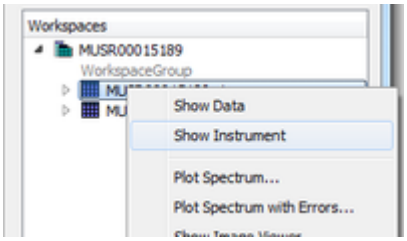
1.4.2 Displaying And Navigating

Opening the Instrument View

To see the Instrument View

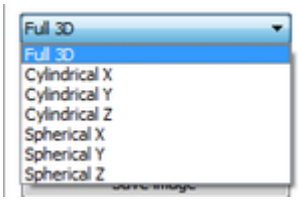
1. Load a data set

2. Right-click on the workspace and select Show Instrument

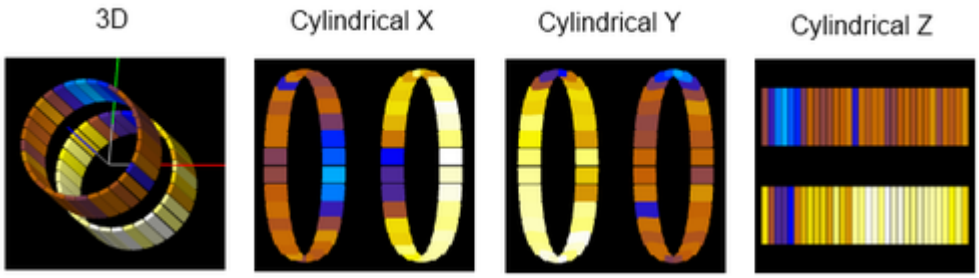


Different projections

There are two ways to display the instrument geometry: the 3D view and an "unwrapped" view. An "unwrapped" view shows a projection of the instrument onto a surface (a cylinder or a sphere) unfolded onto the screen. Use this drop-down menu to select the type of the projection

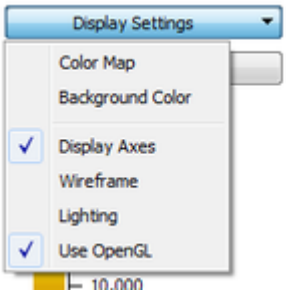


Here is an example of how an instrument may look in different views:



Settings

Some aspects of the instrument appearance in the view can be set from the Display Settings drop-down menu.



Navigating in 3D

In the 3D view:

- The X axis is red.
- The Y axis is green.
- The Z axis is blue.

Rotate the instrument by left-clicking and dragging.

Pan the view by right-clicking and dragging.

Zoom in and out by rotating the wheel or holding down the middle button and dragging up and down.

Zooming in 2D

In an unwrapped view to zoom in select an area by left-clicking and dragging.

Right click to zoom out.

1.4.3 Investigating Data

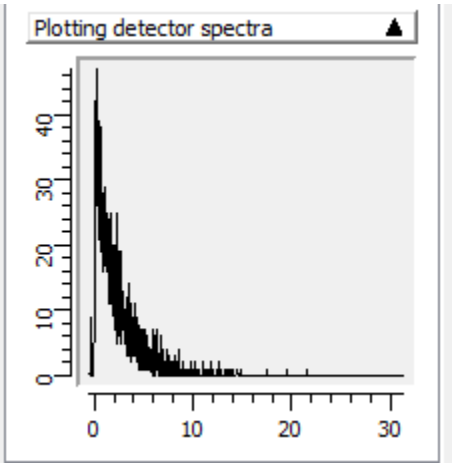
The Pick tab on the controls panel allows you to see the data in the workspace.

The Toolbar

-  - Navigate in the instrument display window.
-  - Edit a shape.
-  - Draw an ellipse.
-  - Draw a rectangle.
-  - Draw an elliptical ring.
-  - Draw a rectangular ring.
-  - Select a single detector.
-  - Select a tube/bank.
-  - Add a single crystal peak.
-  - Erase a peak.

Picking a Single Detector

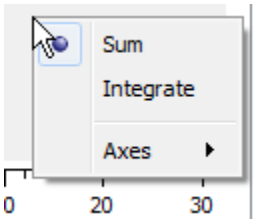
The Single Pixel tool  displays the detector data in the mini-plot at the bottom of the tab.



Hover the mouse over a detector and see the mini-plot update.

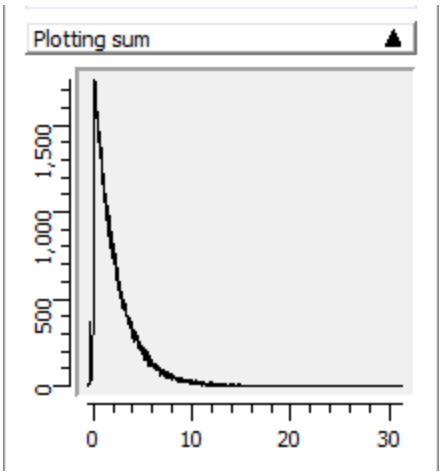
Picking a Tube

The Tube selection tool  is useful for tube instruments. When it's on the mini-plot displays the integrated data in the whole tube. The integration is done either over the detectors in the tube (Sum option) or over time (Integrate). To switch between the option use the context menu of the mini-plot:



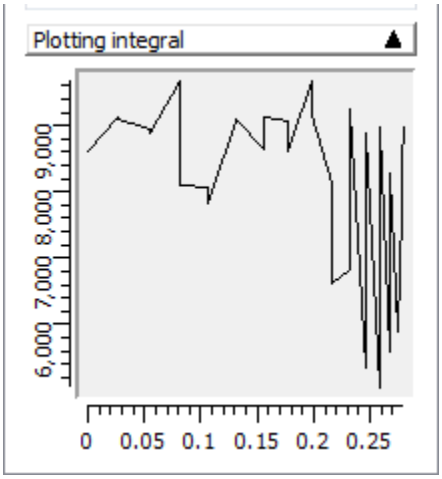
Summing over the detectors

With the Sum option the mini-plot displays a sum of the counts in all detectors in a tube vs time of flight.

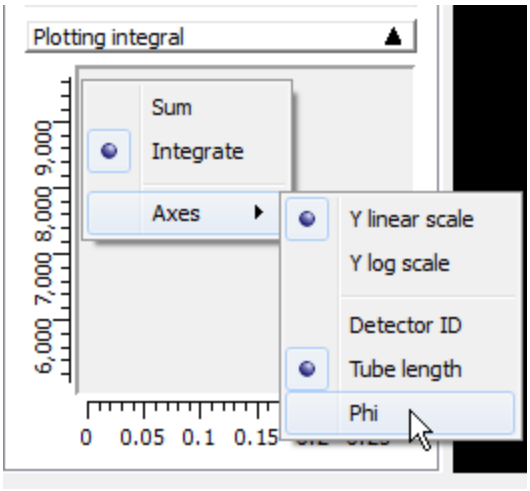


Integrating over the time of flight

With the Integrate option the mini-plot displays the counts integrated over time of flight vs detector position in the tube.



Detector positions can be shown as detector IDs, or distance form a tube's end, or the φ angle. Switch between the units using the mini-plot's context menu.



Navigate

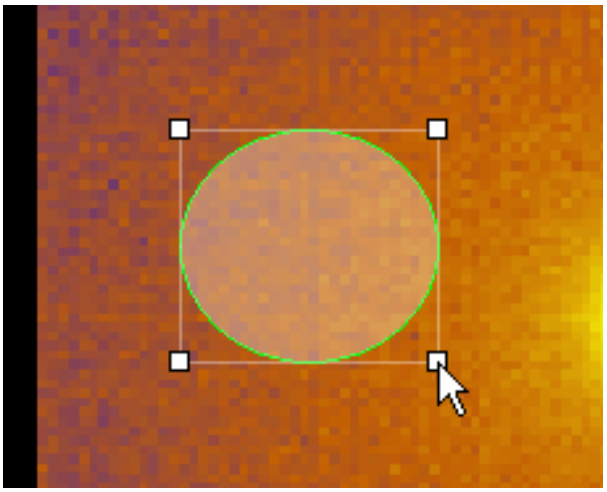
The  tool button switches on the navigation mode which is the same as in Render Tab.

Selecting Arbitrary Sets of Detectors

The rest of the buttons in the top row are for making complex selections. Buttons  are for drawing shapes,  is for editing them.

Draw an ellipse

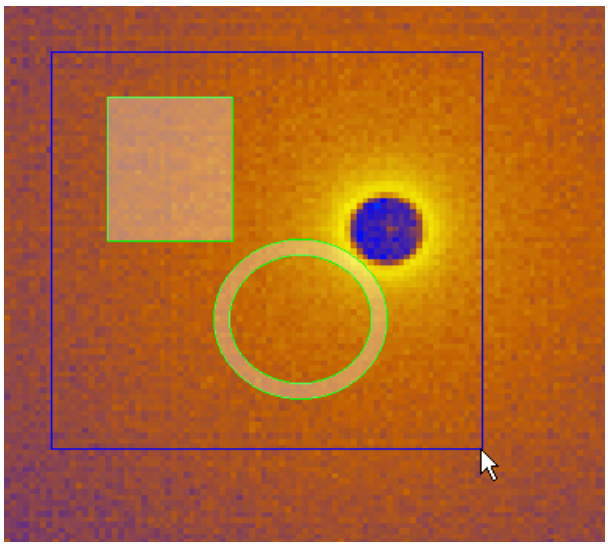
1. Click the  button.
2. Click and hold the mouse button down to start drawing.
3. Drag to resize.



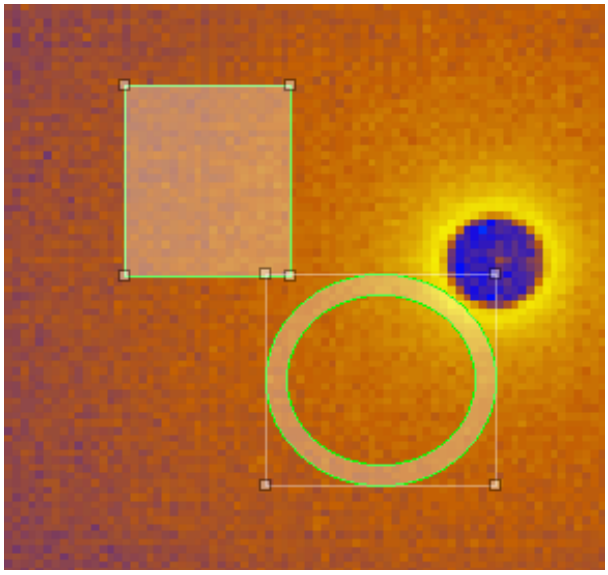
Edit a shape

1. Switch on the editing tool  .
2. Click on a shape you would like to edit. The selected shape displays control points as small white rectangles.
3. Drag the control points to resize the shape.
4. To translate the shape click inside its shaded area and drag.

To select multiple shapes draw a rubber band around them.



The selected shapes are indicated by drawing a bounding box around each of them.



Only translation is possible for a multiple selection.

Sum selected detectors

The mini-plot automatically sums the counts in the detectors covered by the shapes and plots them vs time of flight.

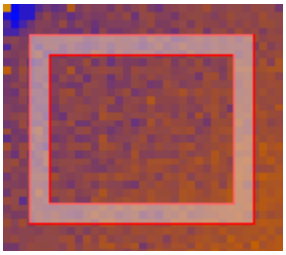
1.4.4 Masking And Grouping

The Draw tab is for grouping or masking detectors.

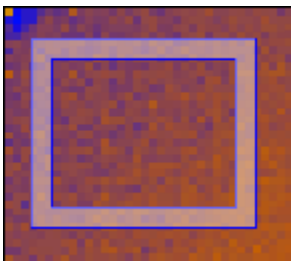
The toolbar on this tab consists of the same tools as the top row of tools in the Pick tab. Their functions and behaviours are also the same but the shapes are used for masking and grouping instead of displaying the data.

Use these radio-buttons to switch between masking and grouping functions.

The shapes ready for masking have red border

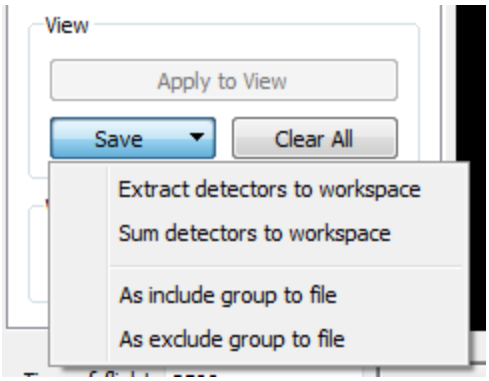


The shapes ready for grouping have blue border



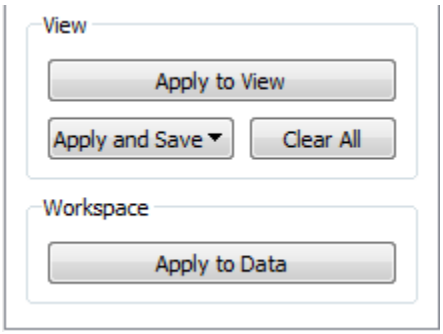
Grouping

In the grouping mode you can extract the data or save grouping to a file:

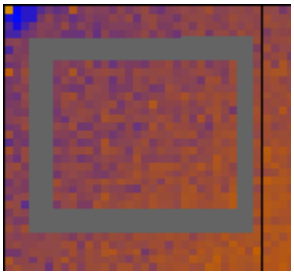


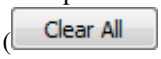
Masking

In the masking mode operations can either be applied to the view only or to the underlying workspace.

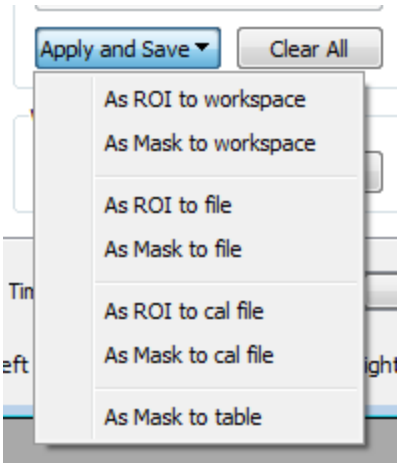


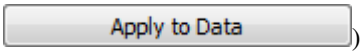
When the "Apply to View" button is clicked the detectors covered by the shapes are greyed out indicating that they are masked.



The purpose of applying to view only is to make creation of masking workspaces/files easier. You can create a mask, apply it to view, save it in a workspace or a file but then clear the view (

button) and create another mask without closing the Instrument View or need to reload the data. The masking operations are available from the "Apply and Save" menu.



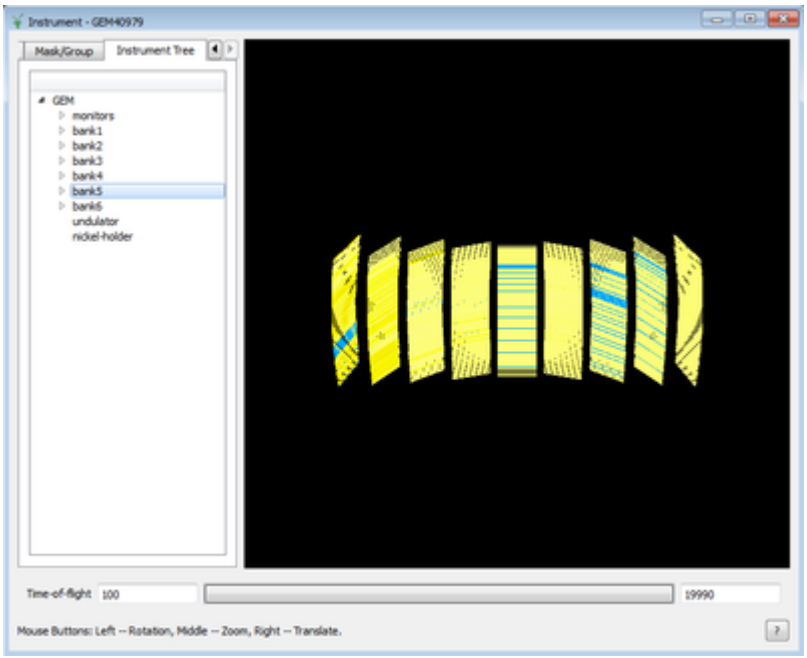
If masking is applied to the data (by clicking 

) it cannot be undone.

1.4.5 Instrument Tree

Instrument Tree

The Instrument Tree tab allows you to see parts of the instrument in isolation. Select a part in the tree widget and the display will show only the selected part.

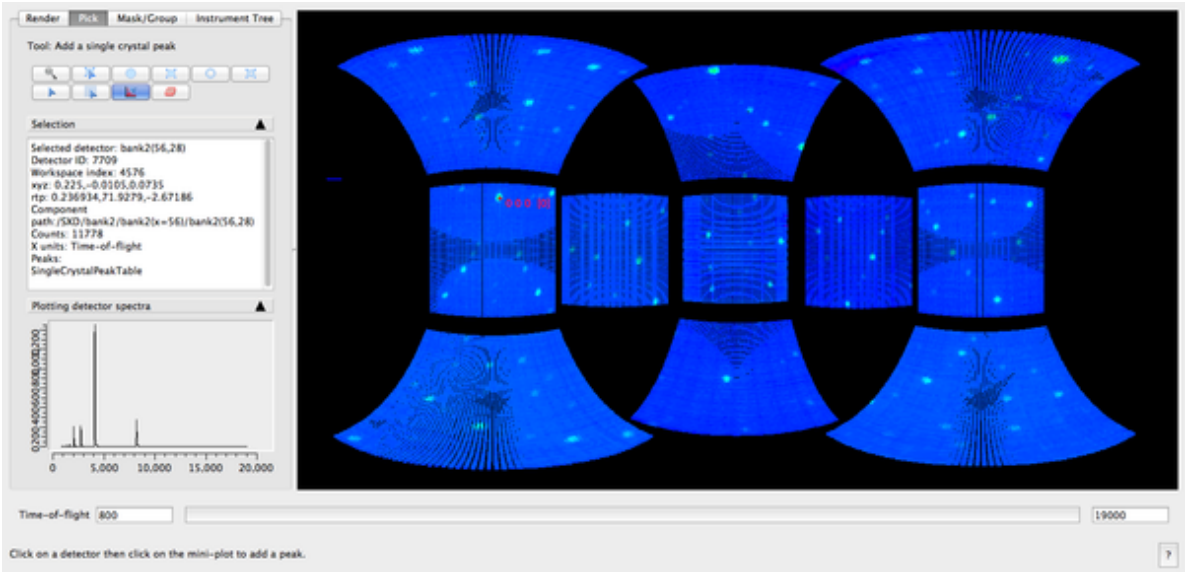


1.4.6 Exercises

Exercise 1

- **Load** *SXD23767.raw* into MantidPlot

- Open the Instrument View
- Change colour map to IDL-rainbow
- Change the colour scale to *Log10* set the scale range to be between 1 and 1e9
- Go to the *Pick* tab. choose *Add single crystal peak*  go to a Bragg peak and left-click. The mini-plot should now be showing a plot of counts vs TOF for that detector.
- Move the mouse over the mini-plot, **right-click** the mouse and select *Save plot to workspace*, plot the workspace called *Curves* which should now be in the MantidPlot workspace panel.
- Go back to the Instrument View, and the same mini-plot. This time select a peak in the plot and **left-click** the mouse. A new peak should be shown in the instrument view. A workspace (PeaksWorkspace) called *SingleCrystalPeakTable* should also exist now in the MantidPlot workspace panel.

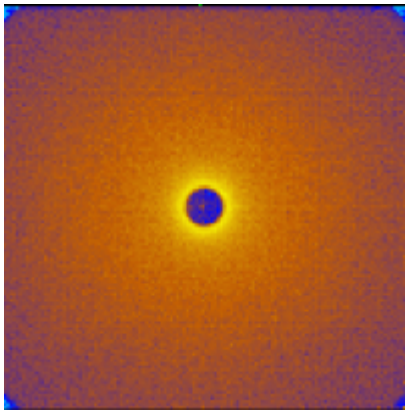


- Double click the new PeaksWorkspace to open it as a table and verify that the TOF value is the same as shown in the mini-plot and the detector ID is the same as shown in the text area above the mini plot
- Now in the Instrument View window use the *Erase single crystal peak(s)*  tool to remove the Peak just added

Exercise 2

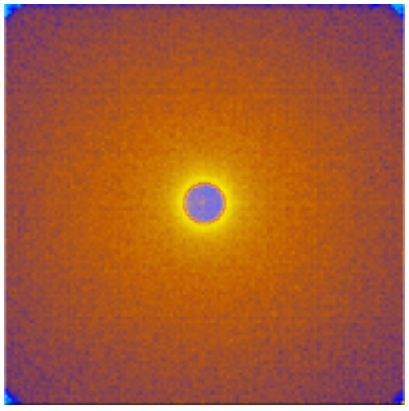
Here we are going to mask out the beam stop and the edge of the main detector for a small angle dataset.

1. Load LOQ48097 data set.
2. Open the Instrument View by right-clicking on the workspace and selecting Show Instrument.
3. If the Instrument View shows one of the "unwrapped" projections switch to Full 3D on Render tab.
4. Select the Axis view to Z+ (to avoid having a view with the front LOQ detector in front of the main detector)
5. Make sure that the colour map axis has the Log10 scaling. The instrument display should look like this:

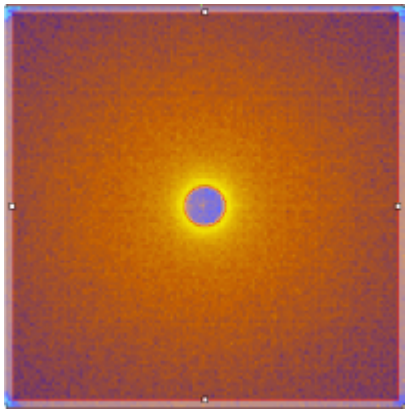


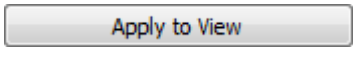
1. Switch to the Draw tab.

6. Select the ellipse drawing button  .
7. Draw an ellipse in the middle of the panel to cover completely the red disk there.

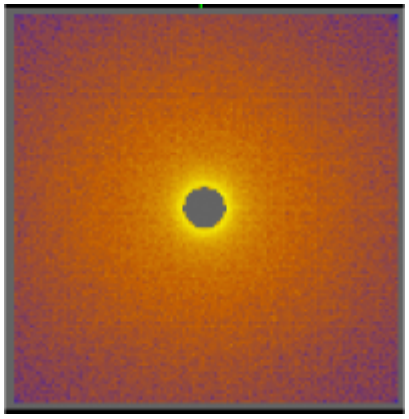


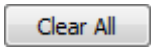
8. Select the button for drawing a rectangular ring 
9. Draw a ring masking the edge detectors of the panel. Use its control points to adjust it to the right sizes.



10. Click 

button.



11. Click Apply and Save button and select As Mask to Workspace. A workspace named "MaskWorkspace" is created.
12. Click  button. The instrument image returns to the original, all masking is removed.
13. Right-click on "MaskWorkspace" and select Show Instrument.
14. Change to Full 3D. The image should look like this:

1.5 Fitting Data

Sections

1.5.1 Fitting Models To Data

Fitting is the modelling of data where parameters of a model are allowed to vary during a fitting process until the agreement between model and data has seen an improvement according to some cost function.

In summary the Mantid fitting provides

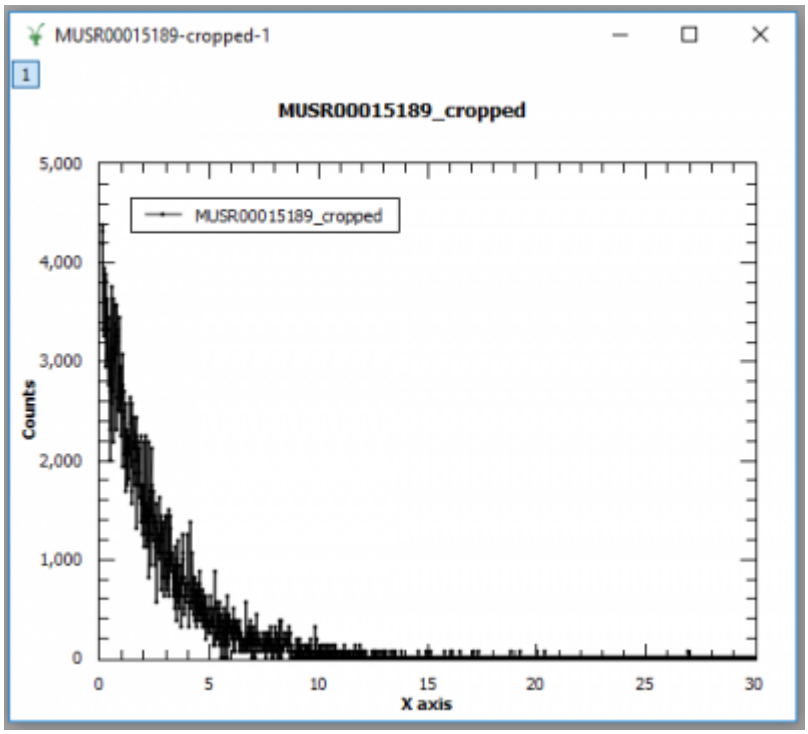
- General fitting capabilities

- Fitting extras, that make use of data log file and instrument geometry information to enhance the user fitting experience
- Easily expandable

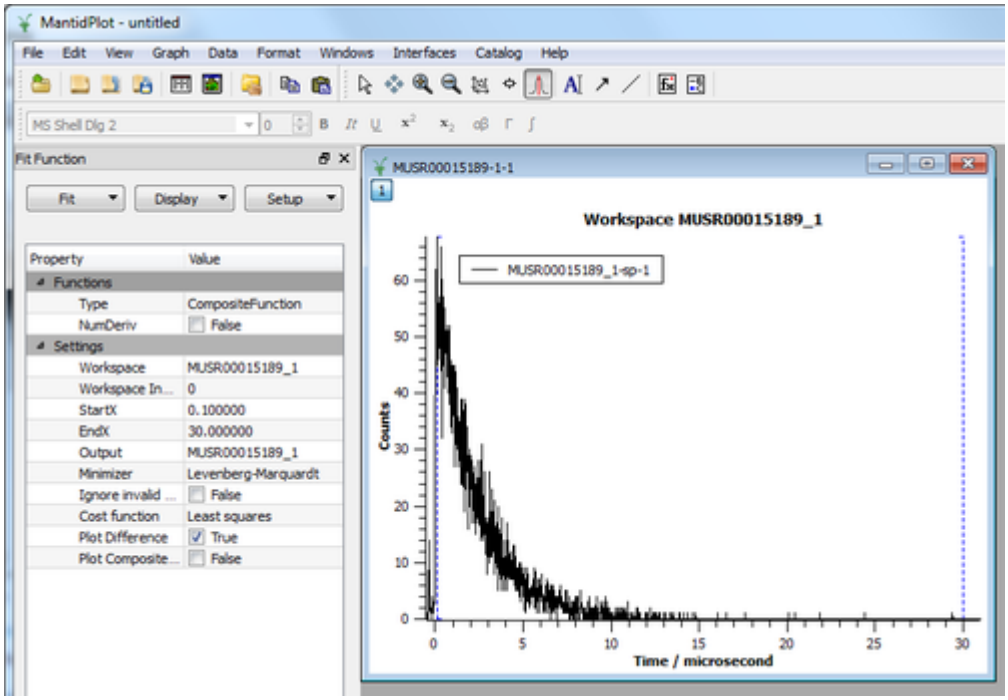
The main focus of this basic course is to cover the basic mantid fitting capabilities.

Simple fitting

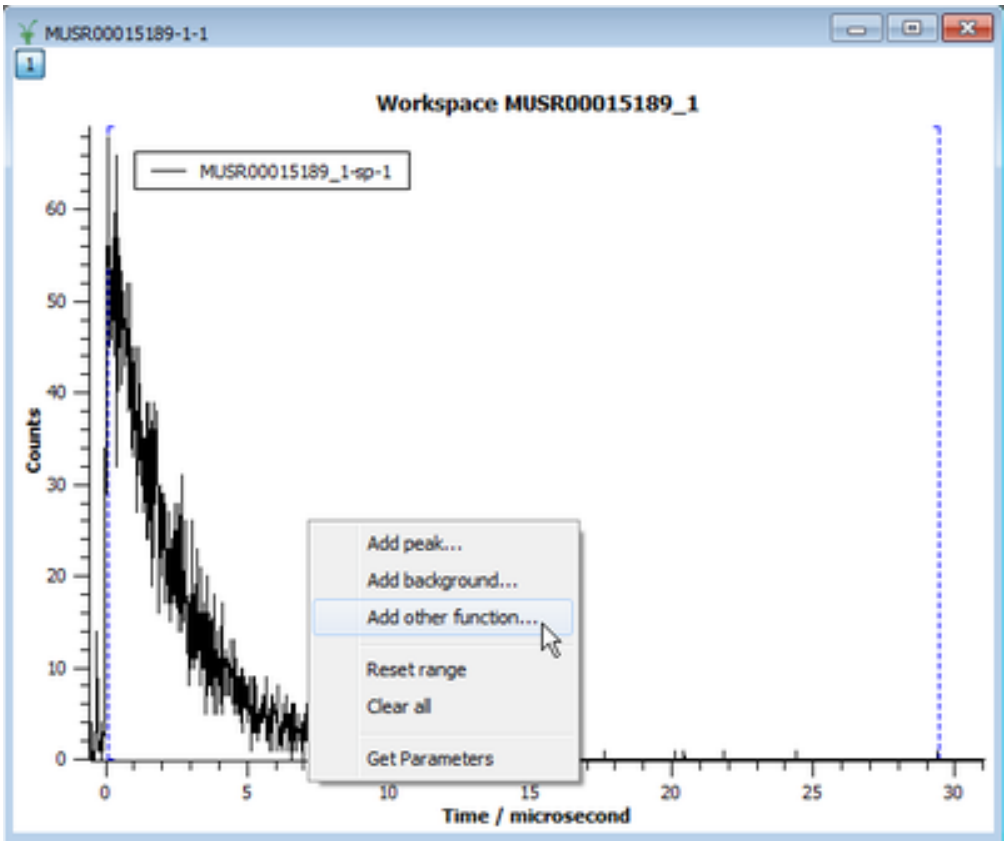
1. Plot a data set. We will use data from MUSR00015189_cropped.nxs file.



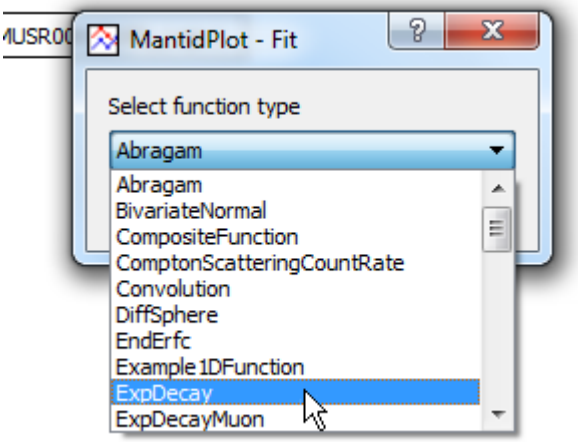
2. Select the fitting tool button: 



3. Right click on the plot to select a fitting function.

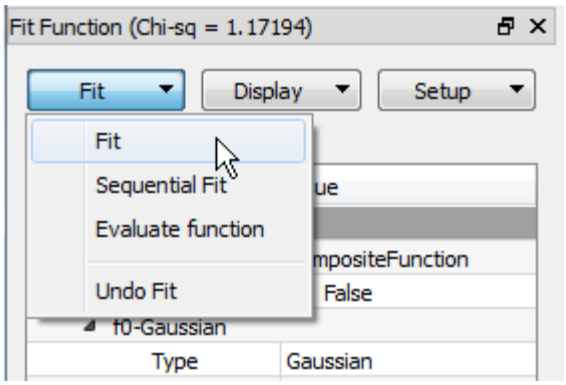


: 4. Choose ExpDecay.



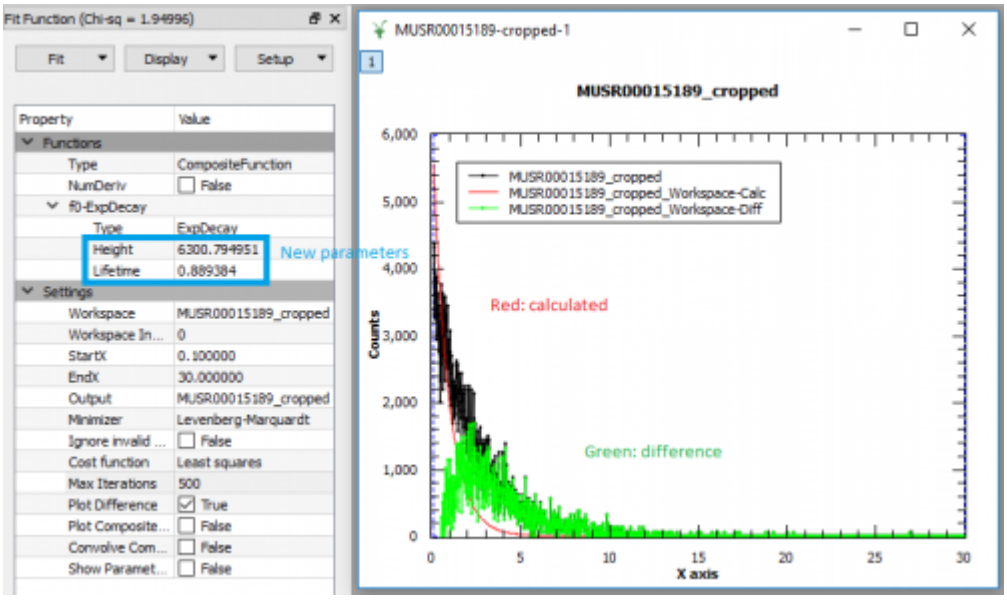
:

: 5. Run Fit.



:

: 6. Examine the results.



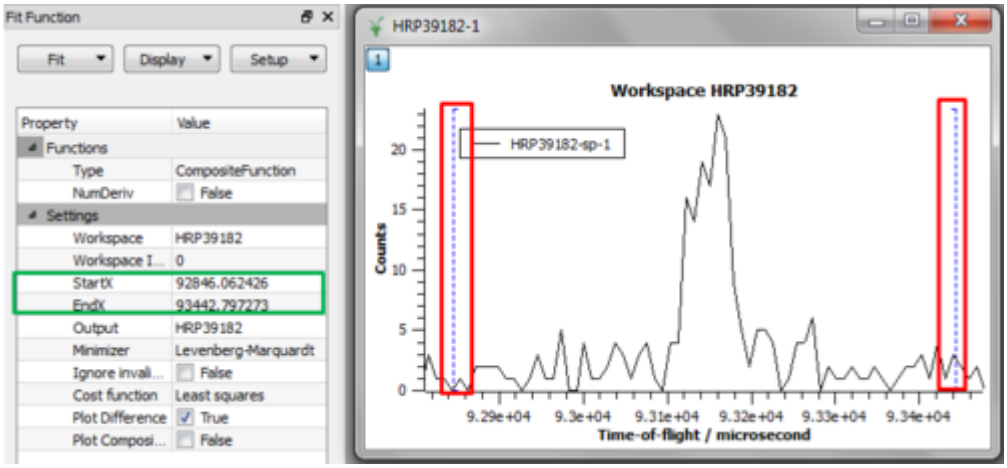
:

A more complex case

Activating the fitting tools

1. Open any data, here HRP39182.raw and plot first spectrum
2. Zoom in on any peak, for example the peak near 93150 microseconds
3. Close to the zoom toolbar button, click on the Fit Function toolbar button:  . If this toolbar is not visible ensure that View->Toolbars...->Plot is ticked.

What you should now see is something similar to



Three things happens when you click on the Fit Function toolbar button

- The Fit Function toolbox, by default, will position itself on the left side of MantidPlot (if not already docked somewhere else)
- fit function items are adding to the plot, initially the two vertical dashed lines highlighted in red in the image above
- and additional right click menu options becomes available on the plot window

Selecting the fitting range

The fitting range is the region of the data where you will attempt to do a fit.

This vertical dashed lines shows the fitting region of the data. These are in sync with the Fit Function setting properties: StartX and EndX values highlighted in green in the image above.

1. Use the mouse to adjust the vertical dashed lines and therefore the fitting range. Notice the StartX and EndX get updated in the Fit Function property browser.
2. Alternative adjust StartX and EndX in the Fit Function property browser which will adjust the positions of the vertical dashed lines

Other Fit Function Settings properties

In addition to StartX and EndX there are a number of other essential fit setting properties including:

- Workspace and Workspace Index: what data to fit. Note typically you should not need to update these manually, these gets updated automatically as the Fit Function button is used to associate the Fit Function toolbox with different plots
- Minimizer: Here you chose between a number of minimizer to search for better fit to a model
- Cost function: The function used to specify the quality of a fit.
- Plot Difference: When the result of a fit is displayed optionally the difference between the model and the fit can also be displayed

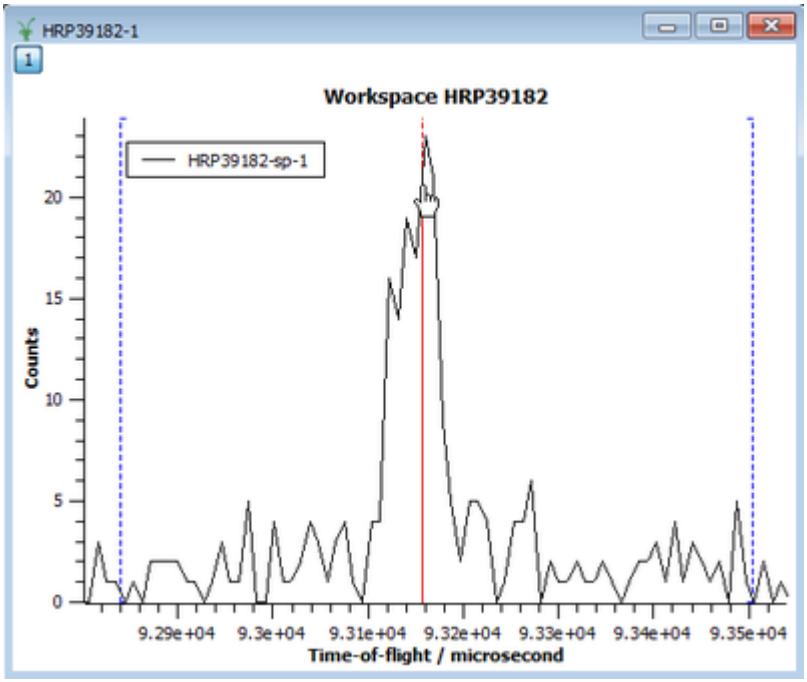
More documentation of these is available from Fitting.

Setting up a fit model

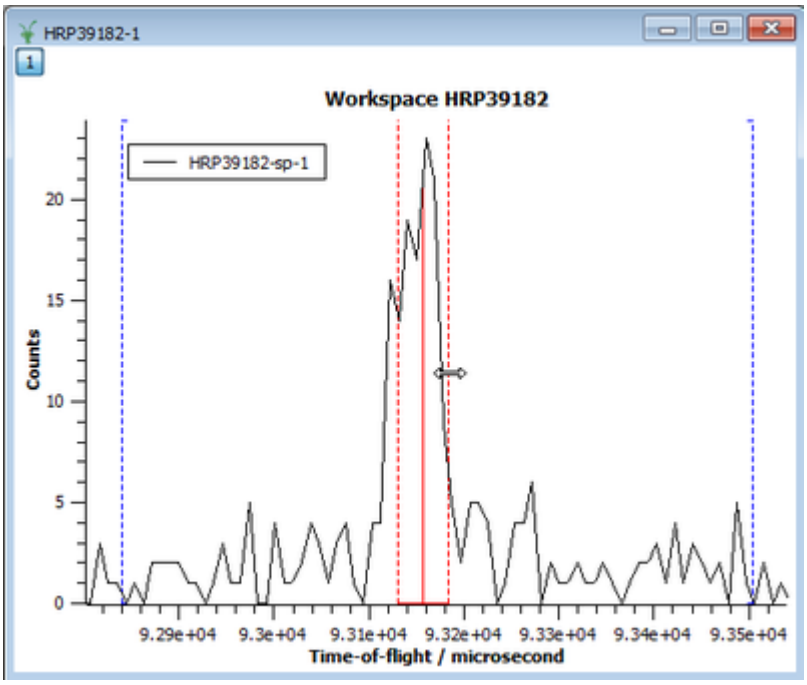
Here start up with building up a model consisting of one peak and one background function.

This can in fact be done using just the Fit Function toolbox. However in practice it is advantegeous to also use the fitting tools available on the plot and from the plot right click menu.

1. Select the right click plot menu option 'Add peak...'. This pops up a new window and in the combobox, select Gaussian. The mouse cursor then changes to a cross. Move this cross near the top of the peak and click any mouse button



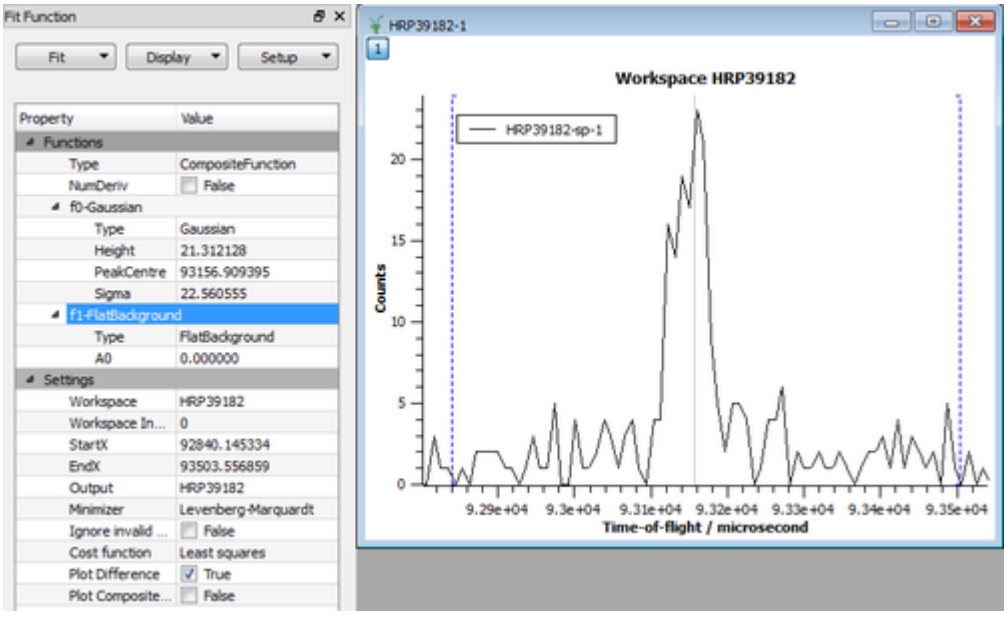
2. Click on the red line and drag it sideways to set the initial width (FWHM) of the peak.



Now you can see 3 red lines - one solid line indicating peak's position and height, and two dashed ones representing the width of the peak. The dashed lines usually show the points of half peak's maximum. All three lines can be dragged within the plot to modify its centre, height or with parameters which is instantly reflected in the Fit Function browser.

3. Select the right click plot menu option 'Add background. . . '. This pops up a new window and in the combobox, select FlatBackground

The result of this is



Note that a vertical line has appeared in the plot where the peak was positioned and two fit function entries have appeared in the Fit Function property browser called 'f0-Gaussian' and 'f1-FlatBackground'.

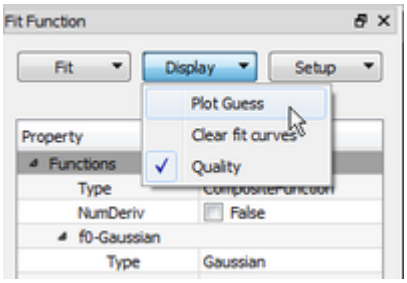
In summary from the Fit Function property browser you have created model consisting of a CompositeFunction which contains a Gaussian and a flat-background. The CompositeFunction part means that the model is sum of its parts, i.e. here the model is

‘‘ ‘f0-Gaussian’ + ‘f1-FlatBackground’‘‘

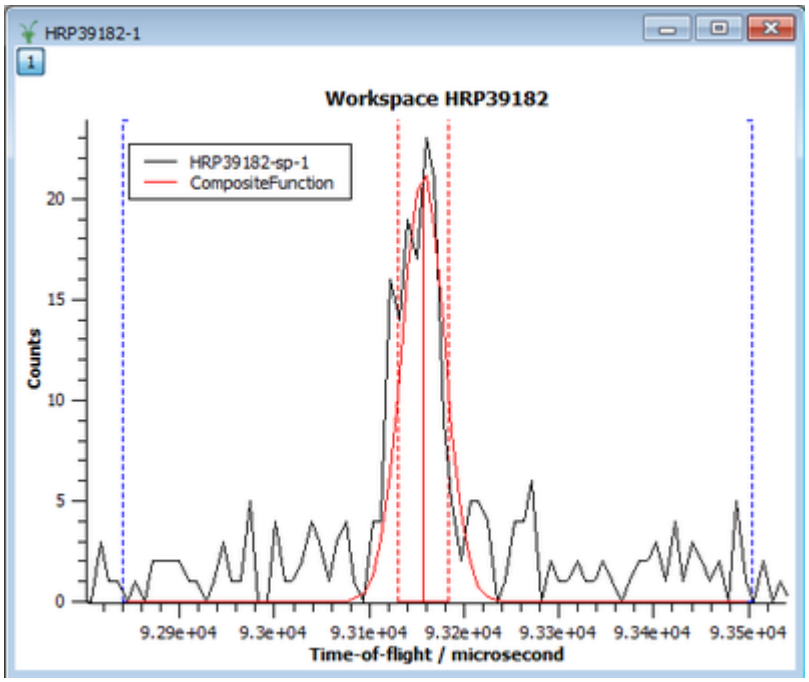
Adjusting fit function parameter

When you do fitting the starting fit function parameter values can greatly affect both the speed and the result you obtain from fitting. In general you want to use staring parameter that are a close as possible to the result you want to obtain.

Fit tools are available on the plot to help you with doing this. In doing this it can be helpful to turn on Plot Guess,



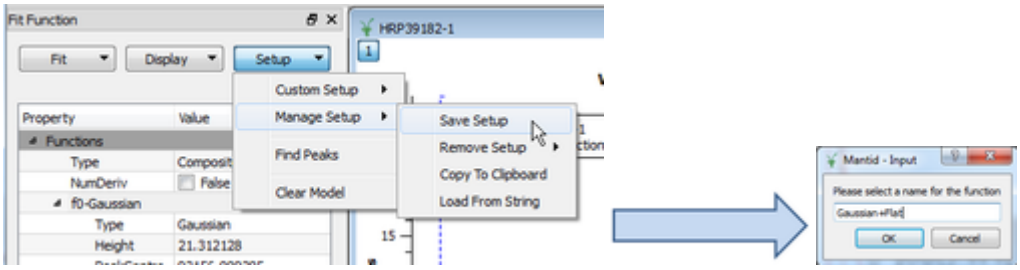
what this means does is to plot how the model you have created looks like compared to your data.



The aim is to have this plot and your data overlap reasonable well before you do the actual fit. Note this is not an absolute requirement but your chances of a successful fit increase this way.

Saving a model

If you have spend a considerable amount of type setting up a model you can save it and then load it later.



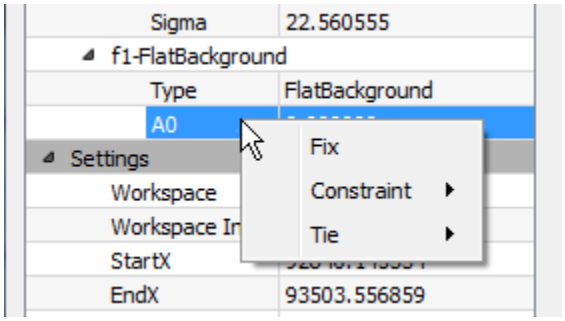
After saving the model it can be imported into Fit Function browser at any time using Custom Setup option from the same menu. Importing (loading) a model replaces all functions in the browser.

Tying and constraining fit parameters

Mantid fitting framework allows you to tie or constrain parameters during the fit. We define tying as setting a parameter equal to a result of an expression involving other parameters of the same function. The expression can be a constant (doesn't involve any parameters), in this case we call the tied parameter fixed.

By constraining we understand setting conditions on acceptable values of a parameter. For example limiting its value from below or above.

To set a constraint or a tie right click on a parameter name:



The menu offers three options:

- Fix: tie this parameter to its current value.
- Constraint: set a bounding constraining condition - define a lower or upper bound, or both.
- Tie: tie this parameter to an arbitrary expression.

The tying expression can be as simple as a parameter name:

```
f0.Height
```

Note that parameter names of a model include prefixes such as "f0." which indicate the particular function they belong to.

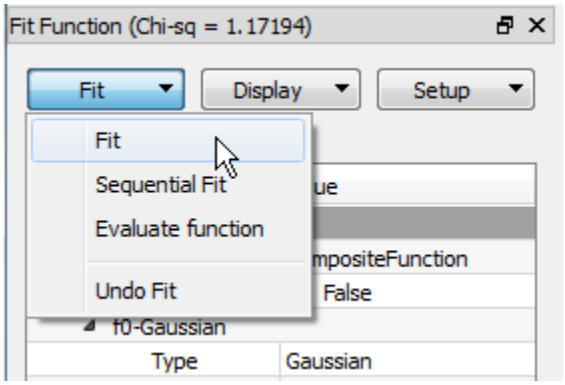
A more interesting example is setting parameter f1.A0 from our earlier model to formula

```
20 - f0.Height
```

This will ensure that the maximum point is exactly 20 units above 0.

Execute your fit

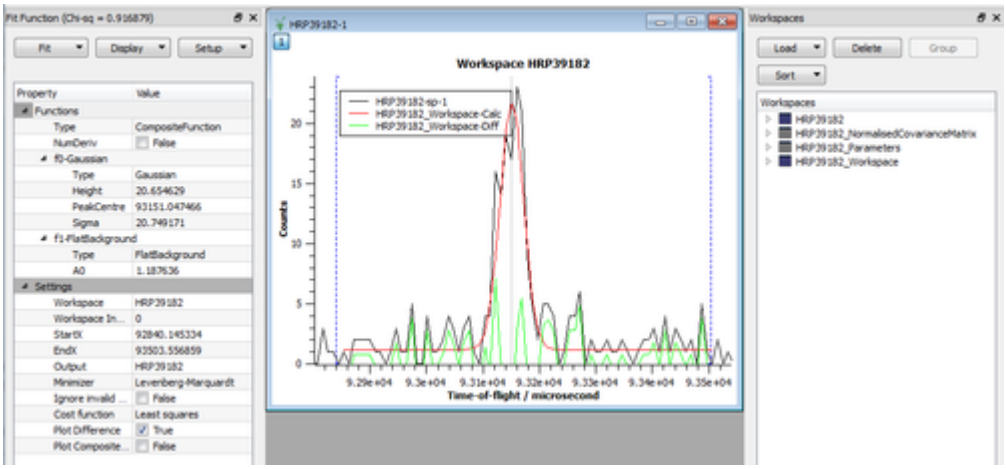
After the model has been defined, its initial values are set, any ties and constraints defined we are ready to run a fit. It is done by selecting the Fit option from the Fit menu:



Inspecting a fit result

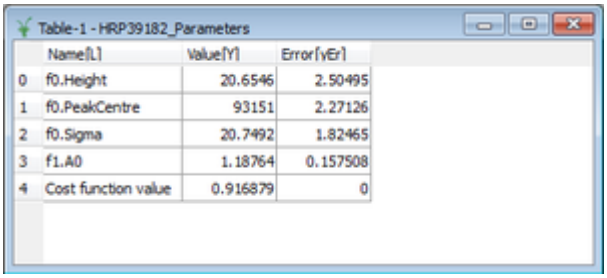
After a successful fit the results can be examined in three ways.

1. The Fit Function property browser will show the fitted parameters instead of their initial values.
2. A plot of the fitted model will be added to the graph along with the difference with the original data.
3. Output workspaces will be created and available via the workspace dock.

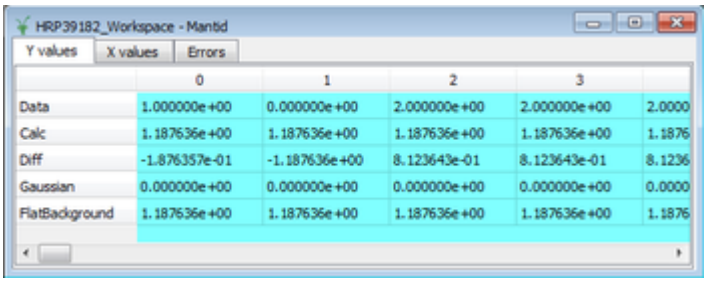


There are three output workspaces:

1. A TableWorkspace with the name suffixed with "_Parameters". It contains the fitting parameters and their corresponding errors.



2. A MatrixWorkspace with the name suffixed with "_Workspace". Its first three spectra are: the original data, the calculated model, and the difference.



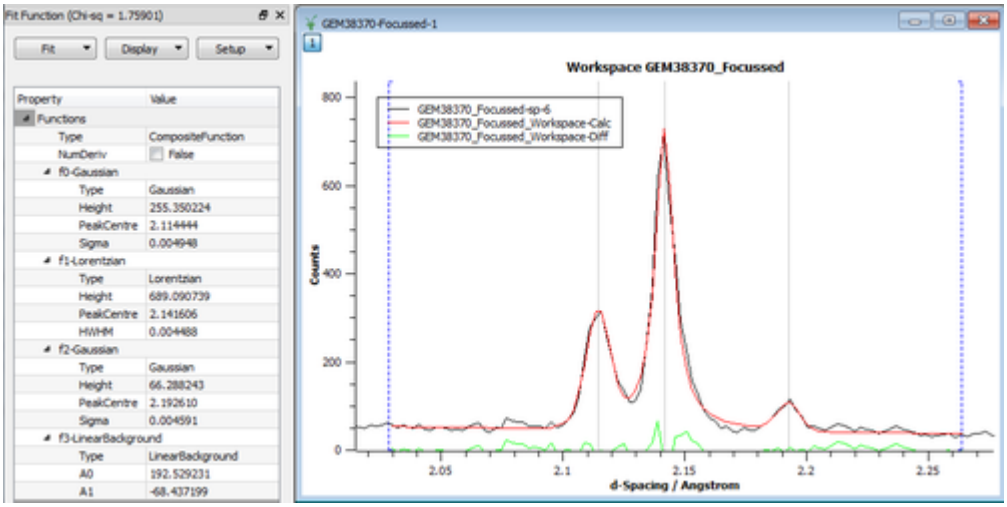
3. A TableWorkspace with the name suffixed with "_NormalisedCovarianceMatrix". It contains the variance-covariance matrix normalized to 100.

1.5.2 Fit Model Choices

Mantid containing a constant increasing catalog of predefined fitting function which can be used to create a model. Some of these well be demonstrated here. In addition create new fitting function using the Fit Function toolbox or by extending Mantid using either a plugin mechanism. The former will be covered here.

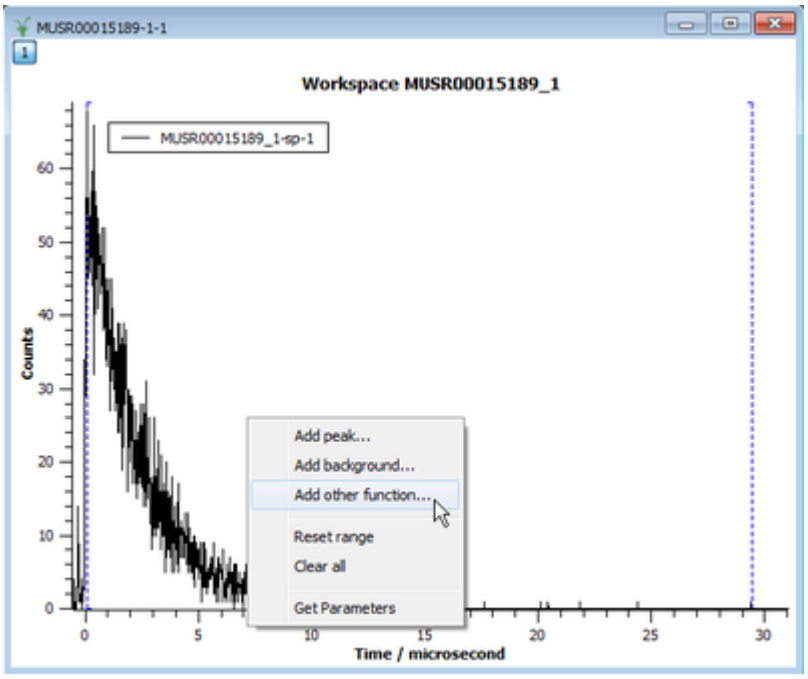
Multiple peaks + background

Mantid allows one to fit overlapping peaks on a common background. Just follow the steps described earlier to add more peaks to the model. The peaks don't have to be of the same type.



Non peak model + background

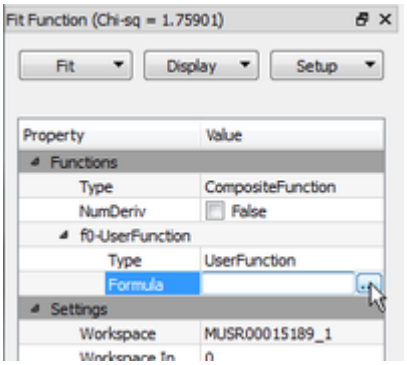
Mantid fitting tool isn't limited to peaks and backgrounds. You can select any other function from a list offered by the "Add other function.." option.



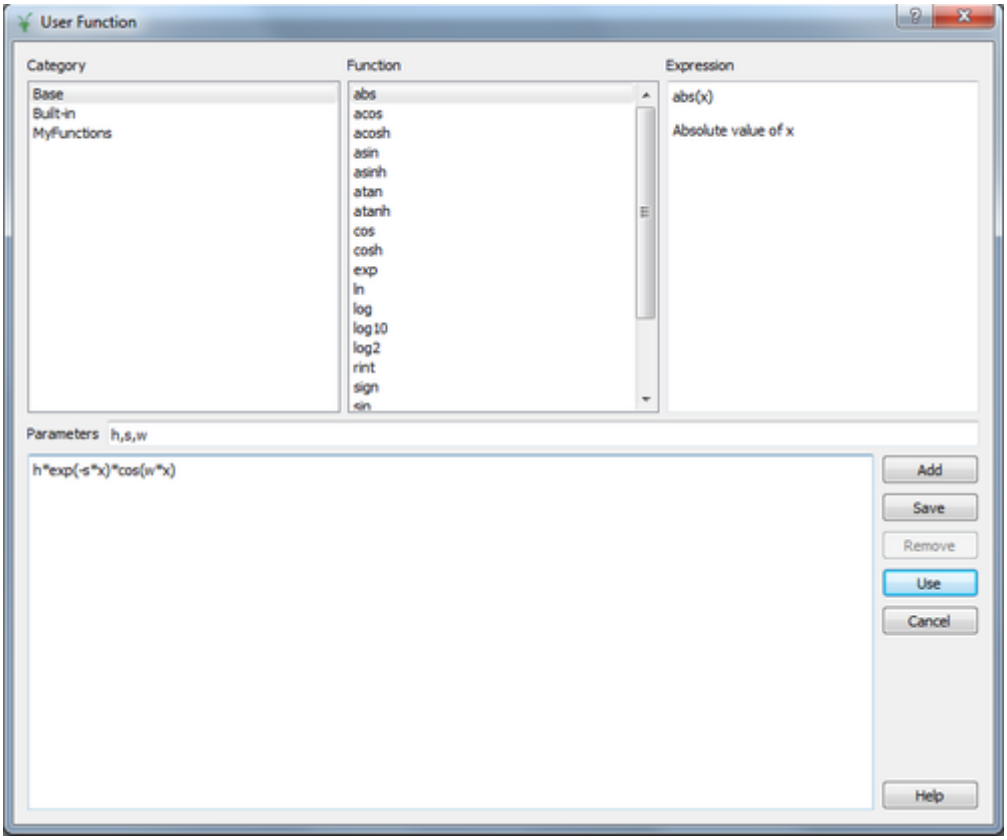
Use custom fitting function

User defined function

Mantid provides a user defined function called UserFunction. It has an attribute called "Formula" which accepts a text string with a mathematical formula. All variables in the formula are treated as parameters except for "x" which is the argument.

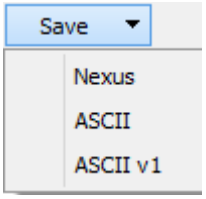


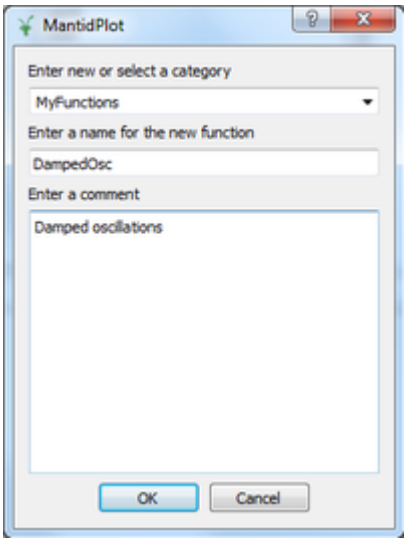
The formula can either be entered in the text editor in the Fit Function browser or constructed with the help of the User Function Dialog.



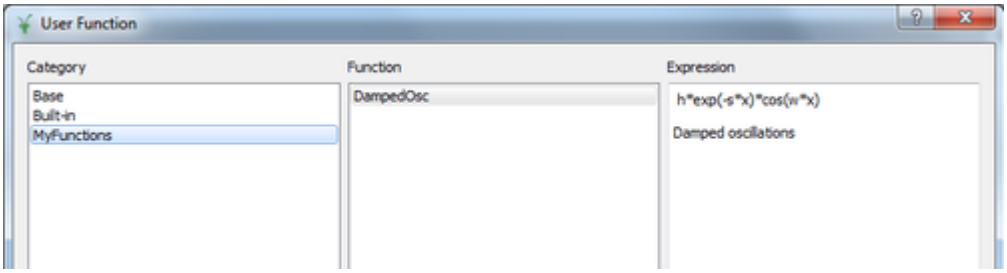
Edit your function in the text field, browse and add () to your formula any built-in or saved function. The fitting parameters are extracted automatically and displayed in the Parameters read-only field. If the field is empty then your formula contains errors.

When finished click the Use button  to insert the formula into the Fit Function browser.

The constructed formula can be saved permanently for future use. Click the Save button  to see the dialog:



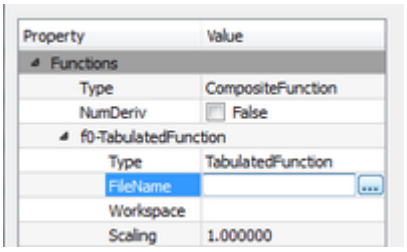
Now your function appears in the list of available functions:



Any unwanted function can be removed from the list using  button.

Tabulated function

A TabulatedFunction takes its values from a file or a workspace



1.5.3 Intelligent Fitting

Intelligent Fitting

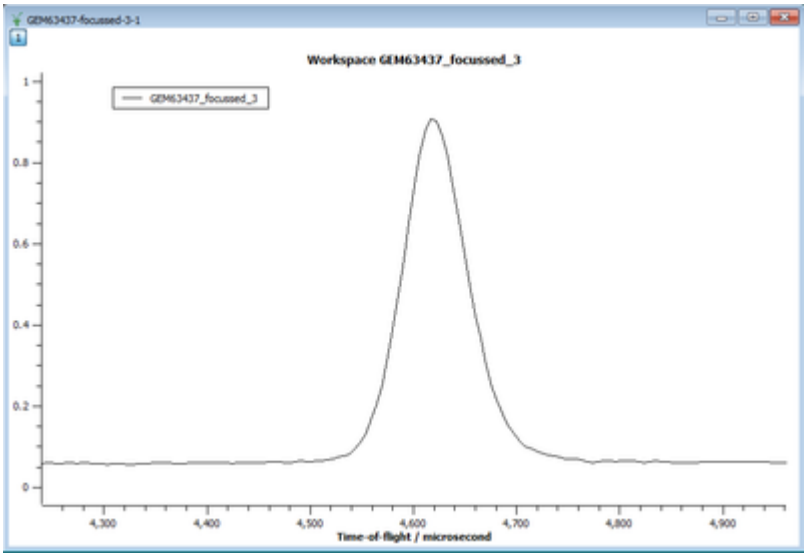
Fitting is the modelling of data where parameters of the model are allowed to vary during a fitting process until the model and data agree better according to some cost function.

In summary the Mantid fitting provides

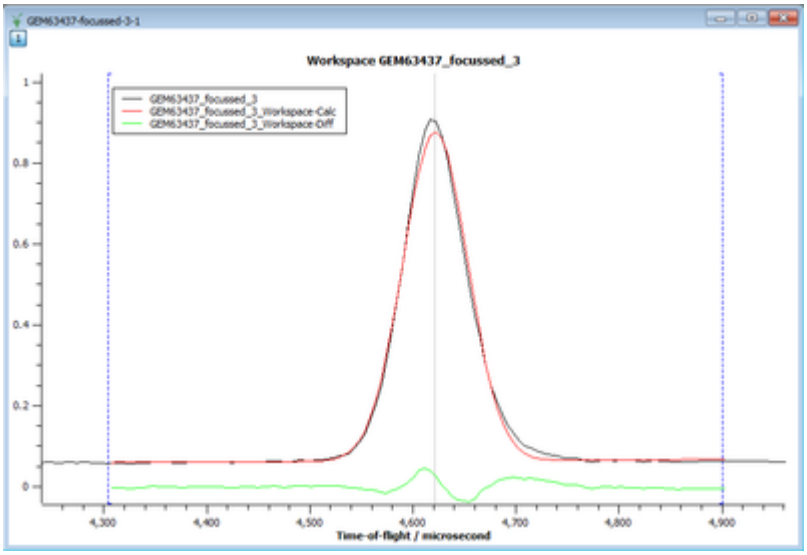
- General fitting capabilities
- Neutron and Muon intelligent fitting tools, which makes use of additional information about the data, such as instrument geometry and log value information
- Easy expandability

Here a more advanced aspect of Mantid fitting is presented. We will fit an asymmetric peak from a GEM data set with the Ikeda-Carpenter function on a LinearBackground.

Let's load GEM63437_focussed.nxs file and plot GEM63437_focussed_3 workspace. Zoom into the region 4300 - 4900 microseconds.

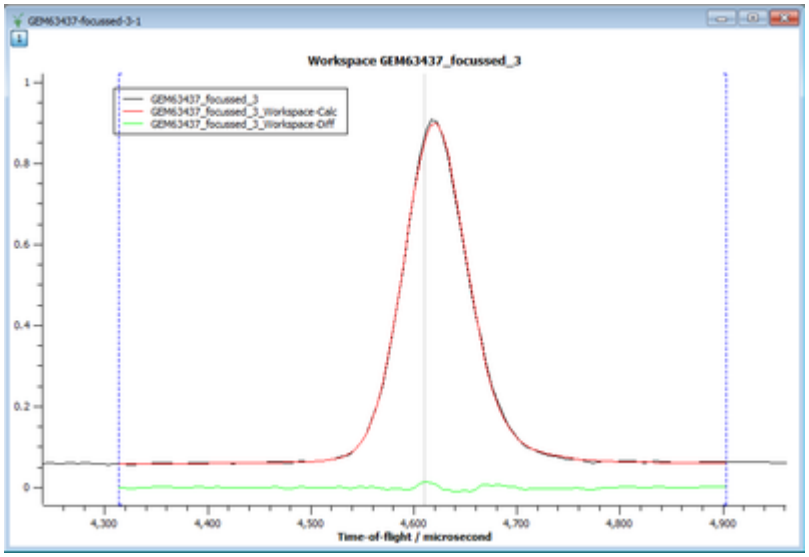


Try to fit it with a Gaussian (plus LinearBackground):



Not a very good job. The Ikeda-Carpenter function is a better choice here. But this is a very difficult function to work

with. It requires very good initial parameter values for the fit to converge. The Mantid approach is to use the pre-set values which are defined on a per-instrument basis. For example, when the Ikeda-Carpenter is used with GEM data the fitting tools automatically find and set the appropriate initial values. This results in a good fit.

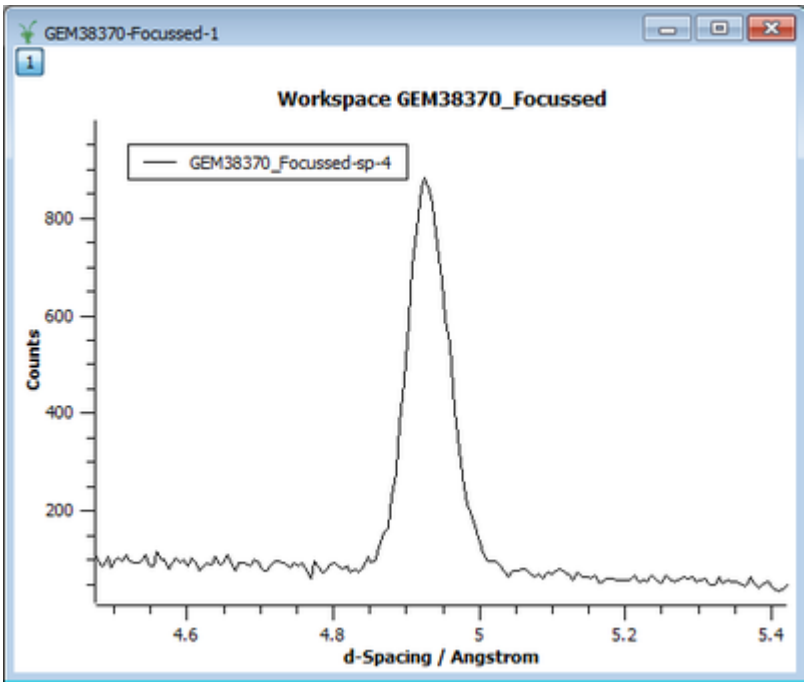


1.5.4 Exercises

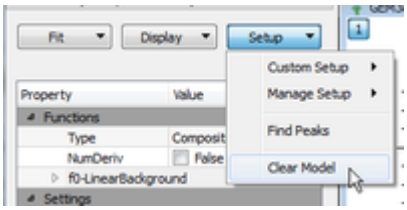
Exercise 1

In this exercise we will fit a simple Gaussian on a linear background.

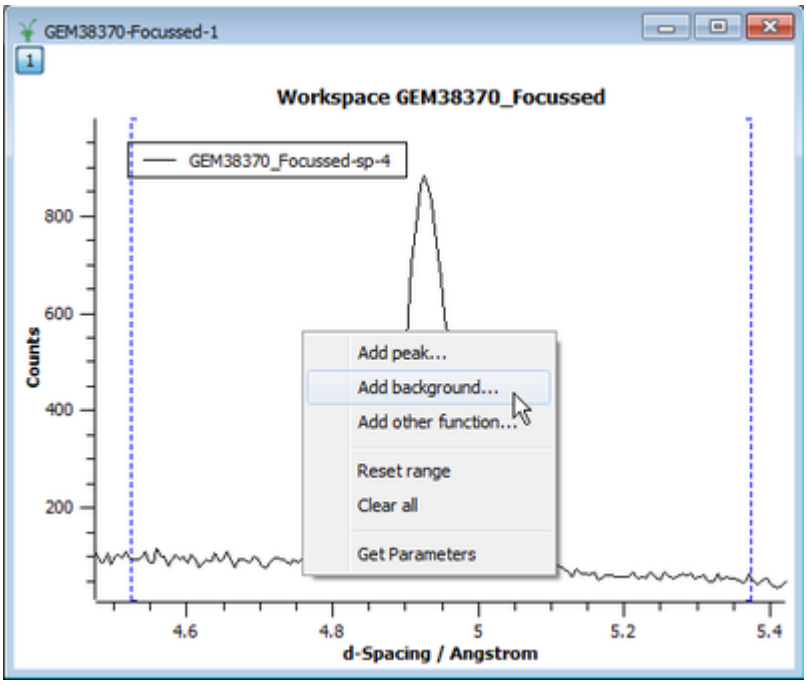
1. Start with loading the data set (GEM38370_Focussed).
2. Plot spectrum 4.
3. Zoom into the peak around 5 angstroms.



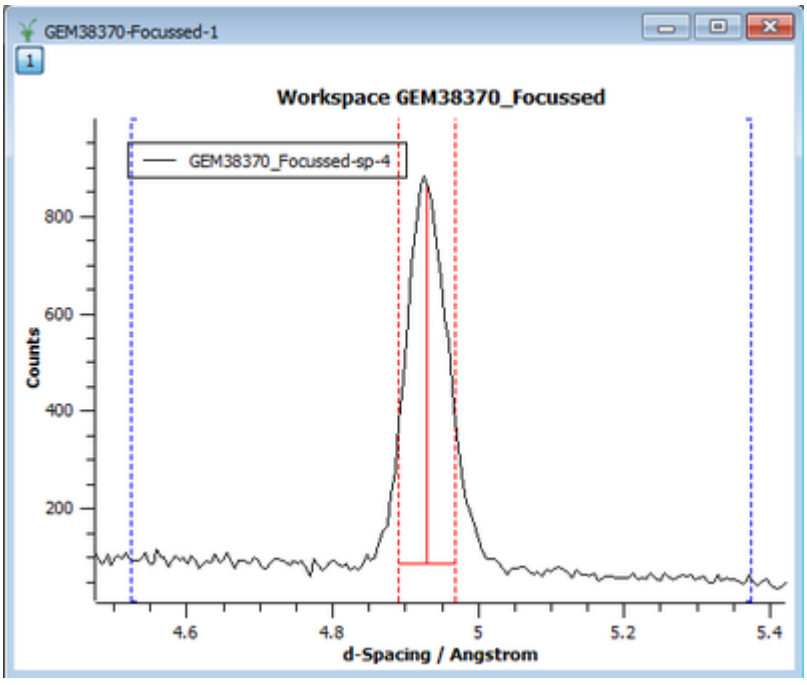
4. Start the fitting tool.
5. Adjust the fitting range if needed.
6. Make sure the fitting model is empty in the Fit Function browser. If necessary clear it.



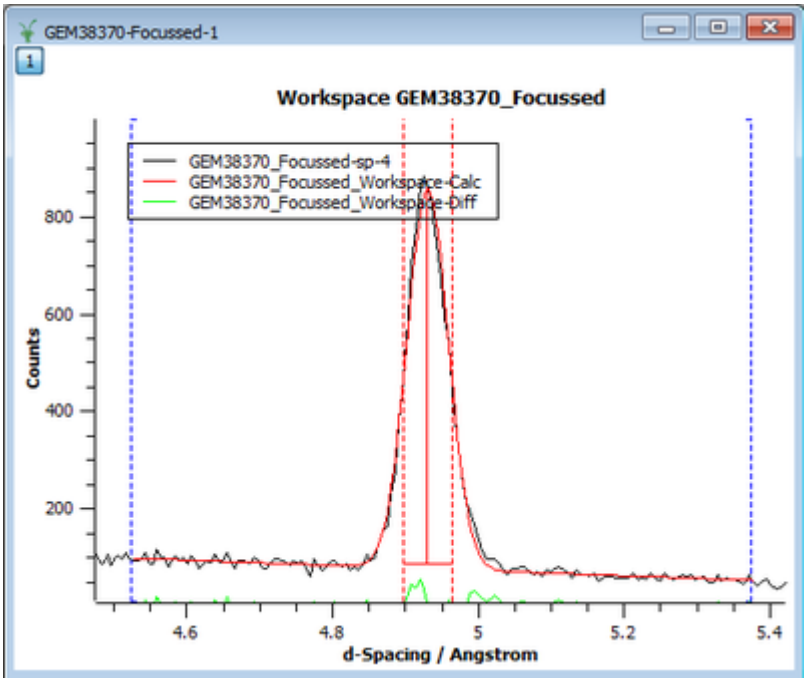
7. Add a background function. Select LinearBackground.



8. Add a peak. Select Gaussian.
9. Click at peak's maximum point to set initial values for the centre and the height.
10. Adjust the width.



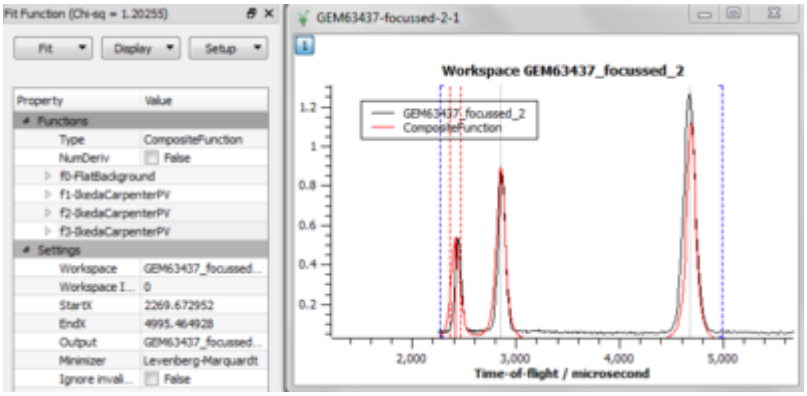
11. Run Fit.



Exercise 2

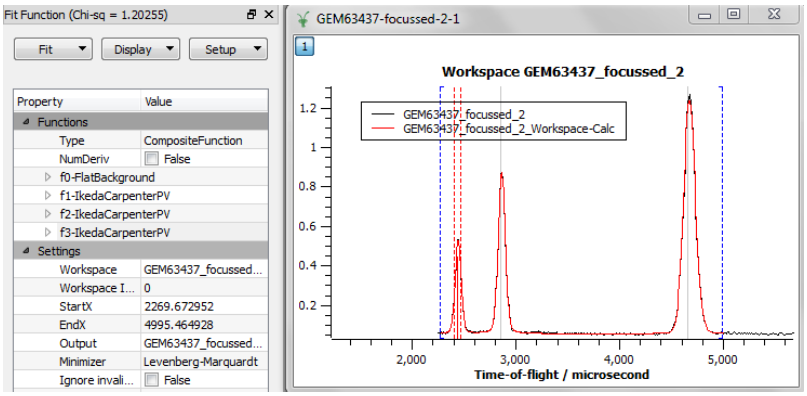
This exercise will mainly quiz about fitting.

1. Ensure that the Fit Function panel is clear using panel option Setup->'Clear model'
2. Load the GEM63437_focussed.nxs data. Note the workspace created is a WorkspaceGroup. This is simply a containing workspace grouping one of more other workspaces. This dataset has been proceed already with Mantid. How many algorithms have been applied to this processed dataset?
3. Plot the spectrum in GEM63437_focussed_2, and zoom in on the area of the three peaks
4. Associate the plot with the Fit Function panel and set fitting range to be between approximately 2270 and 5000 microseconds
5. Right click on plot and select Add background... then FlatBackground
6. Right click on plot and add peak IkedaCarpenterPV. This is peak function where some parameters of the peak function may be related to instrument geometry. As of this writing scientist(s) of the GEM instrument modified the file MantidInstall/instrument/GEM_Parameters.xml such that when you added the IkedaCarpenterPV peak it automatically selected some sensible starting parameter values. This is evident from the starting guess of the peak width but also by inspecting this function in the Fit Function panel
7. Hold down shift key and click on top of the two other peaks
8. Plot (fitting) guess and what you should see is something similar to



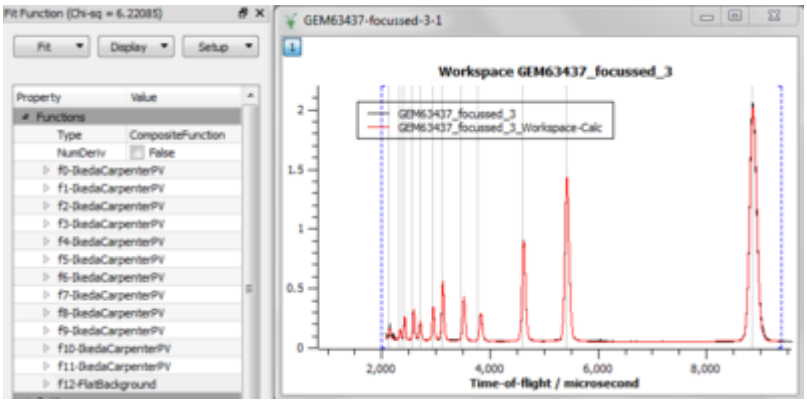
where the red line is the guess

9. Remove plot guess
10. Fit the data with the model, where the output should be something similar to



where the red line is here the fit

11. clear the model
12. Plot the spectrum in GEM63437_focussed_3
13. Using the same background and peak fitting function as above fit the region between about 2000 and 9400 microseconds, and obtain a result similar to



Note this will take a while as it is a fit with almost 40 independent parameters to optimise.

14. Optionally using a similar approach try to fit the spectrum in for example GEM63437_focussed_5

Exercise 3

1. Load the MUSR00015189 data set.
2. Plot a spectrum.
3. Start the fitting interface.
4. Follow the steps described earlier to add and set up a user defined function (UserFunction).
5. Define a function with Formula = h*exp(-a*x)
6. Fit the data.

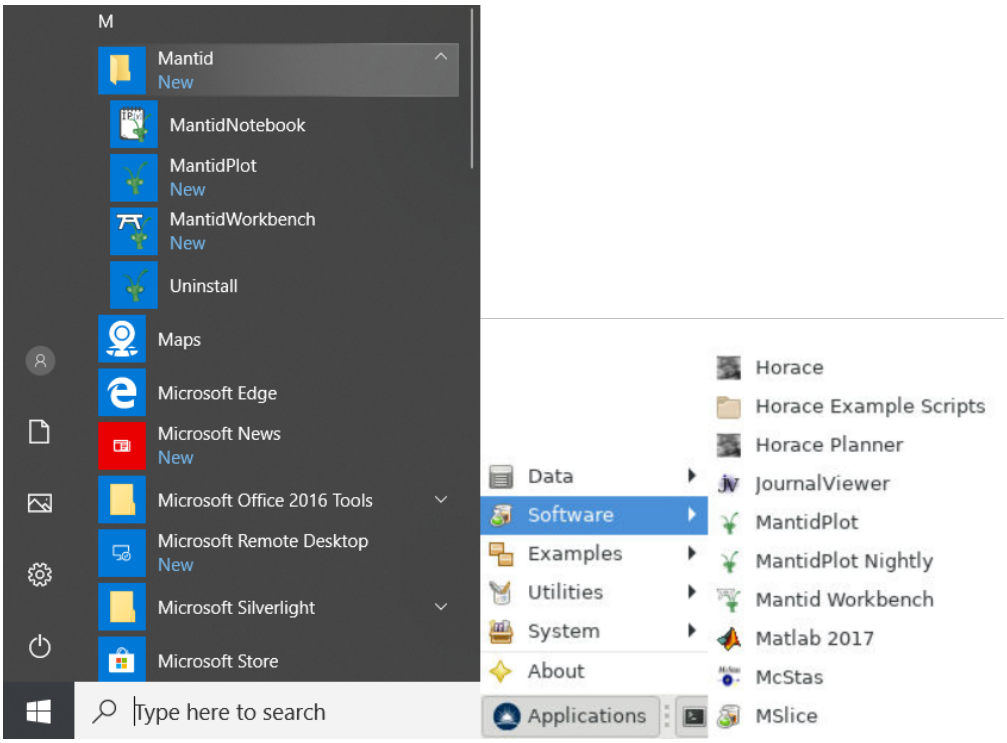
1.6 Mantid Workbench

Sections

1.6.1 Opening Mantid Workbench

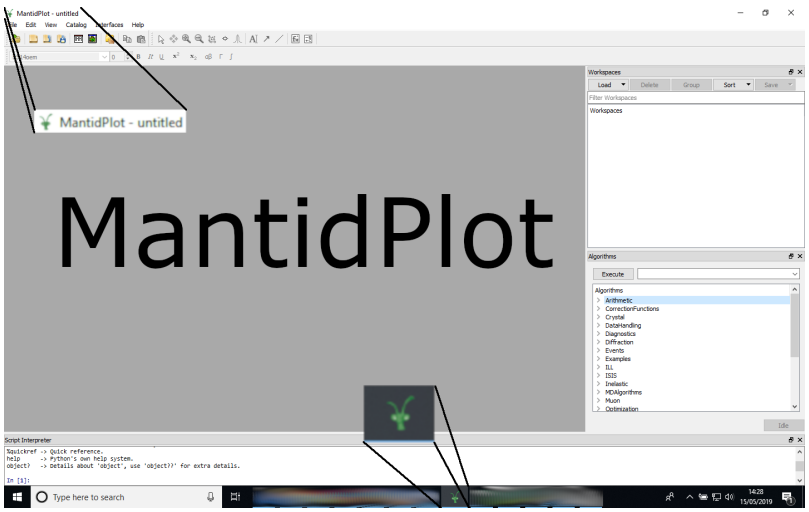
If you are familiar with older versions of Mantid you will be used to the user interface MantidPlot. MantidPlot was built on Qt4 but as Qt4 is no longer being supported the Mantid Workbench was made using Matplotlib for it plotting functionality. Currently the Mantid Workbench does not have all the functionality of MantidPlot but once it does Workbench will fully replace MantidPlot. In this chapter we will cover how to recognize and use some of the features of the Mantid Workbench, as well as the features from MantidPlot that are not yet integrated into Workbench.

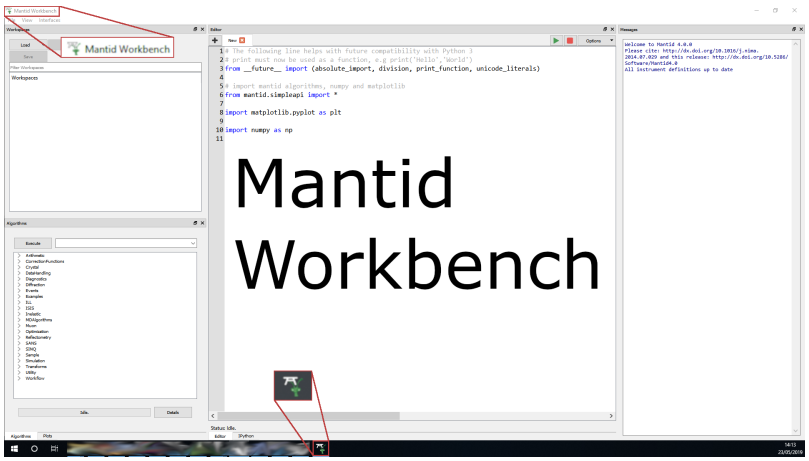
Launching the Mantid Workbench



The Mantid Workbench is available in any install of Mantid 4.0 or higher, available to download form the Downloads section of the website. This will install both MantidPlot and the Mantid Workbench which will be available to launch from the same directory.

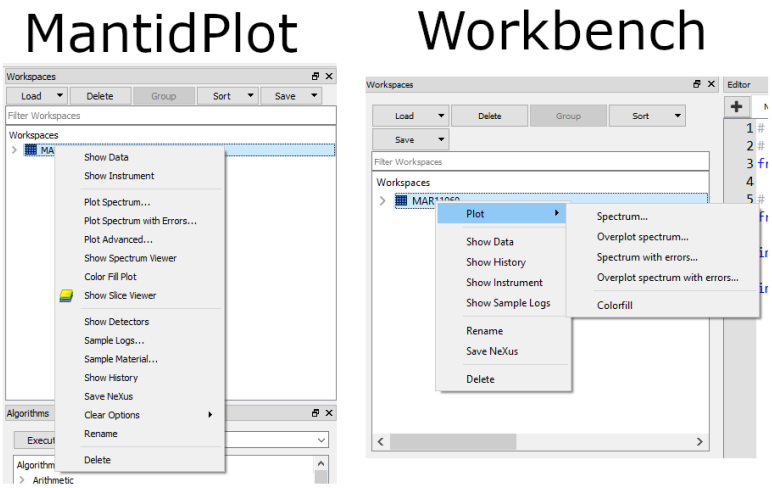
MantidPlot and the Mantid Workbench are visually distinct. If the default layout has the workspace and algorithm toolboxs on the right with a lot of empty space then you have opened MantidPlot. If it has the workspace and algorithm toolboxs on the left with a script editor in the centre then you have opened the Mantid Workbench.





The default layout of both can be changed so you can't always rely on that if you are opening a copy of Mantid that has already been altered. In that case you can also tell the difference by the name in the top left corner, and the icon on the taskbar.

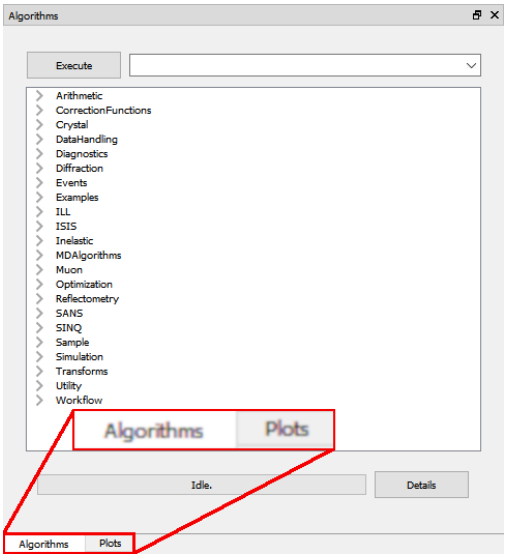
Most of the basic functionality you have covered in MantidPlot is available to you in Workbench. You are still able to load workspaces as you did before but the context menu you get when right clicking on a workspace is different in Workbench.



When starting in Workbench it is good to get used to the differences in layout between the two and knowing which is best suited for for your task.

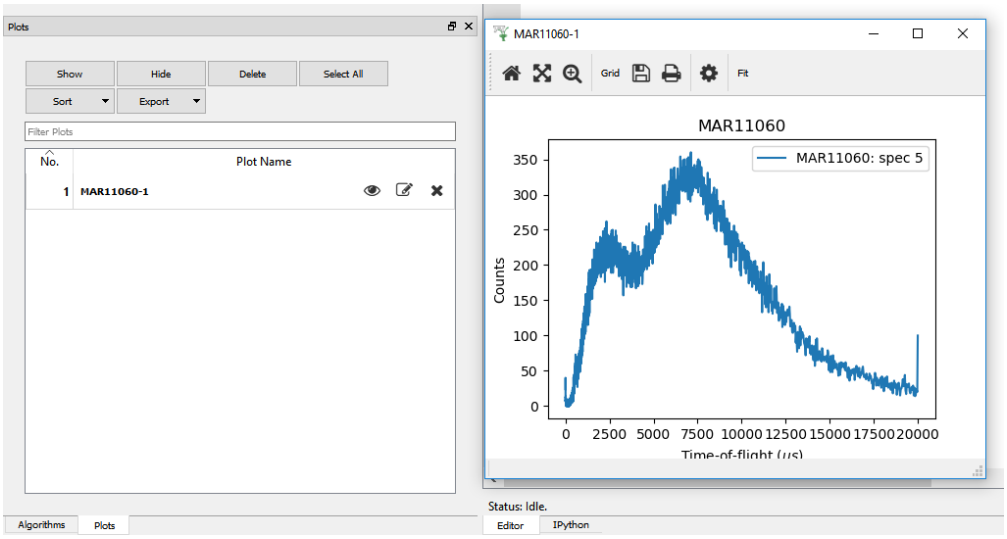
1.6.2 Plotting in Workbench

A new feature of Workbench is the Plots Toolbox, with this you can manage several plots at once from one window. to switch to the Plots Toolbox click on the Plots tab in the bottom left had corner, this will switch out the Algorithms Toolbox.



The Plots Toolbox has a number of functions to help manage your plots:

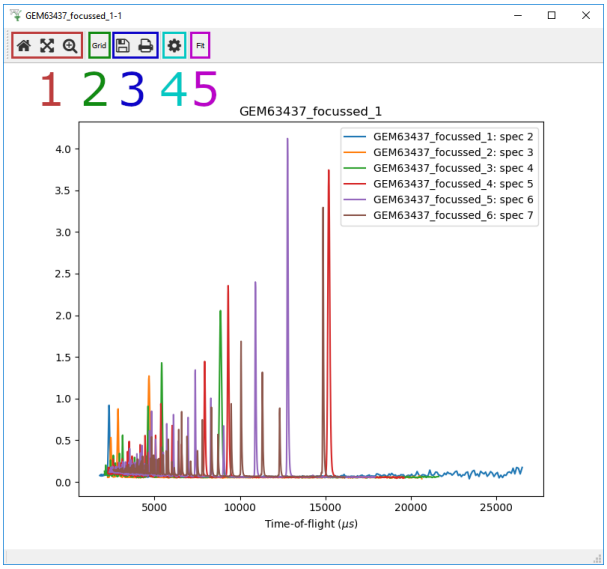
- Show/Hide: Toggles the visibility of the selected plot, the same function can accessed by clicking the eye button next to the plot name.
- Delete: Closes the selected plot from the Toolbox, this is the same as closing the plot window or clicking the X on a plot in the Toolbox list.
- Select All: Selects all plots currently in the Toolbox
- Sort: Allows for plots in the Toolbox to be sorted in order of name, number or last opened.
- Export: Exports the selected plot as an .EPS, .PDF, .PNG or .SVG fromat.
- Rename: By clicking on the symbol of a square and a pencil (between the eye and the X) you can rename the selected plot, names default to "workspace"-"number"



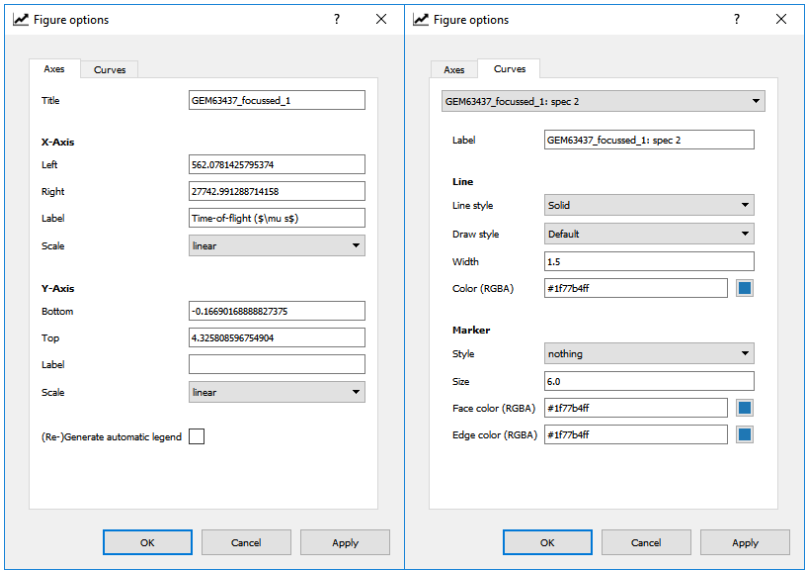
Matplotlib

Workbench uses the Python Package *Matplotlib* to build it's graphs instad of *Qt4* which was used in MantidPlot. As such the plot window itself looks somewhat different. Along with your plot you now have a taskbar with a number of options:

1. Reset view/Pan and zoom with mouse/Zoom to rectangle.
2. Toggle grid lines on and off.
3. Export to file or Print figure. (Print not yet implemented)
4. Configure plot options.
5. Open fit browser tool



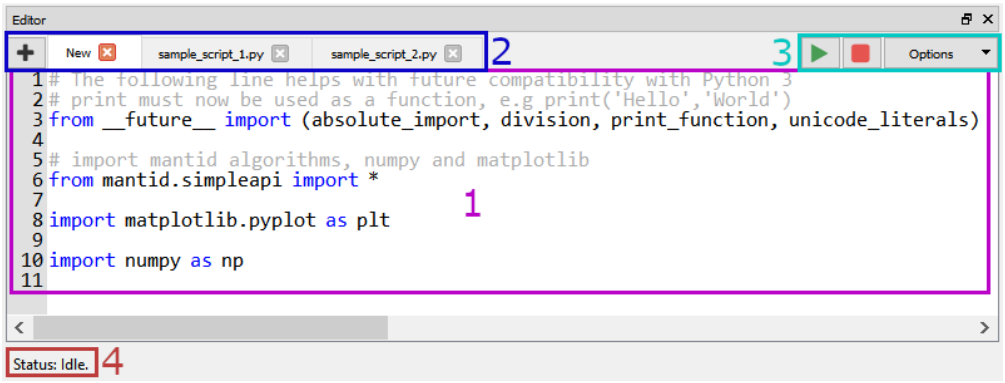
The plot can be modified from the plot options menu, from here the axis labels and titles can be changed along with the scale and range of the plot axes. You can also change the line specs and labels of indevidual curves including the colour by giving an 8 digit RGBA hex code (red, green, blue, alpha) or use the built in widget by clicking on the square button next to the text box.



Currently the Workbench user interface is unable to do everything MantidPlot was able to, as it does not support 2D or 3D plots and slacks the spectrum viewer among others. To get the full use out of Matplotlib you can work from the IPython prompt or by writing script in the Workbench main window. If you have not used Matplotlib before it might help to consult the online Matplotlib tutorial.

1.6.3 Working with Python in Workbench

Within the Mantid Workbench you can make use of the script window or IPython prompt to manipulate data and build plots exactly to your specifications in ways that you would not be able to with the interface alone.



You can also use the IPython tab below the script editor to change to an inline Python interpreter. Both the python interpreter and scripts run thorugh Mantid are able to interact with Mantid workspaces, plot and algorithms.

1. Default imports

By default any new script opened in Workbench will come with some default imports:

```
from __future__ import (absolute_import, division, print_function, unicode_literals)
```

This provideds compatability between scripts written in Python2 and Python3, notably all print statements must not use brackets.

```
from mantid.simpleapi import *
```

This imports all the algorithems from Mantid to be used within your Python script.

```
import matplotlib.pyplot as plt
import numpy as np
```

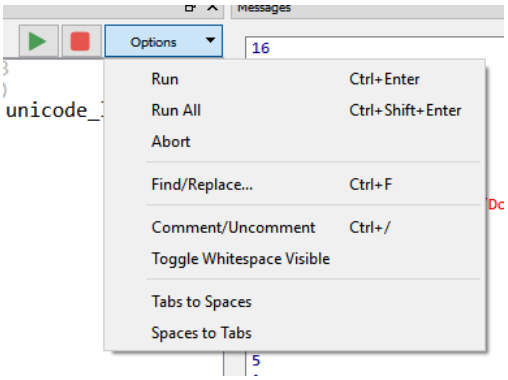
This imports Matplotlib for use as well the the excellent NumPy package.

2. Tabs and multiple scripts

To open a new script you can click on the **+** button, or to open an existing script in a new tab go to "File" -> "Open Script" in the Workbench taskbar or use the hotkey *Ctrl* + *o*. From the script window you can also tab different scripts working on each independanty and multiple scripts can be run simultainiusly. Workbench will restore any tabs that were open the last time Mantid was shut down.

3. Running, aborting, and options

From here you can run or abort your script, toggle options for accesability, and use the find and replace tool.



- Run: This executes the currently selected text in your script, if nothing is selected it will run the entire script. The green play button serves as a shortcut for this option.
- Run All: This will run the entire script regardless of whether any text is selected.
- Abort: This imediatly aborts the currently running script. The red stop button serves as a shortcut for this option.
- Find/Replace: this opens the find and replace toolbar which can be used to make quick mass changes to your script or find sections of code.
- Comment/Uncomment: This tool comments out or removes commenting on highlighted lines by adding or removing # at the begining of the line.
- Toggle Whitespace visible: Turning this on will make all spaces apear as faint dots and all tabs appear as arrows.
- Tabs to spaces/Spaces to tabs: This will convert any tabs highlighted into groups of 4 white spaces or vice versa.

The toggle whitespace visible option is global and will effect the apperance of all your tabs.

4. Status

The status bar tells you if the code in the currently open tab is running or not. If the code has been run previously the status bar will give deatils of the previous run including runtime, and whether the code ran without errors.

1.6.4 What is missing from Workbench

While much of MantidPlot's functionality can be replicated through the python scripts the interface on Workbench does lack some aspects that you have used in previous parts of this tutorial.

Plotting tools

In the loading_and_displaying_data chapter you learnet how to create 2D colour plots and 3D surface plots. Colour plots in Workbench cannot have their colour bar scale changed between linear and logarithmic removing the usefulness of colourfill plots. Workbench also lacks an interface for producing 3D plots at all, lacking the advanced plot tool that was presant in MantidPlot. Access to these plots must be done through Matplotlib directly, it may help to consult the Matplotlib in Mantid tutorial.

Interfaces

MantidPlot provides a range of different interfaces that are not yet available in Workbench. Currently the interfaces available through Workbench are:

- Defraction
  - HFIR 4 Circle Reduction
  - Powder Defraction Reduction
- Direct
  - DGSPlanner
  - DGS Reduction
  - MSlice
  - PyChop
- SANS
  - ISIS SANS
  - ORNL SANS
- Utility
  - FilterEvents
  - QECoverage
  - TofConverter

1.6.5 Exercises

It is possible to replicate plots you've made previously in MantidPlot entirly using python script in Workbench. For all these exercises do not make any changes to the default imports given by the Workbench script editor. You can execute the script after each step to make sure that there are no errors raised.

Part 1:

1. Load the File "GEM38370_Focussed_Legacy.nxs" using the algorithm: `GEM38370_Focussed_Legacy = Load("GEM38370_Focussed_Legacy")`
2. Before plotting from workspaces you will need to set your subplot up with the Mantid projection so it can interpret them. To do this include:

   `fig,ax = plt.subplots(subplot_kw = {"projection": "mantid"})`

   This will let you plot spectra from workspaces by including a spectrum number `specNum` or workspace index `wkspIndex` in the plot command.

   `ax.plot(RawData, specNum = 2)` or `ax.plot(RawData, wkspIndex = 3)`
3. Plot the other spectra 3-7.

   Now that the subplot is set up to plot spectra you can plot other curves in the same window using the same command with a different `specNum` or `wkspIndex`, by modifying the script you have written already you can use a `for` loop to plot all spectra in sequence.

```
for i in range(6):
    ax.plot(GEM38370_Focussed_Legacy, wkspIndex = i)
```

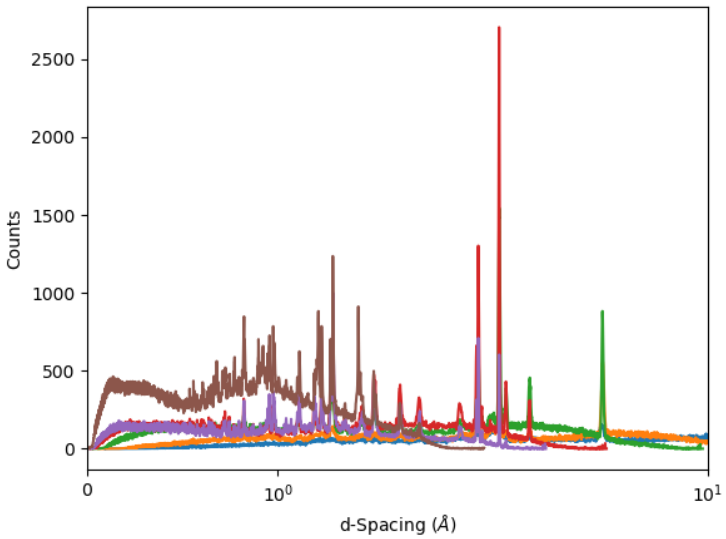
4. change the d-spacing axis range to 0-10 angstroms and set the x-axis to a logarithmic scale

   Since `ax` is a matplotlib subplot we can us the built-in matplotlib attributes to change the the axis range `ax.set_xlim([0.0,10.0])` and scale `ax.set_xscale('symlog')`. You can then use the `fig.show()` function to show your plot

By following these steps you should end up with code that looks something like this:

```
# The following line helps with future compatibility with Python 3
# print must now be used as a function, e.g print('Hello','World')
from __future__ import (absolute_import, division, print_function, unicode_literals)

# import mantid algorithms, numpy and matplotlib
from mantid.simpleapi import *

import matplotlib.pyplot as plt

import numpy as np

GEM38370_Focussed_Legacy = Load("GEM38370_Focussed_Legacy")
fig,ax = plt.subplots(subplot_kw = {"projection": "mantid"})
for i in range(6):
    ax.plot(GEM38370_Focussed_Legacy, wkspIndex = i)
ax.set_xlim([0.0,10.0])
ax.set_xscale('symlog')
fig.show()
```

That will produce a figure looking like this:



Part 2

1. Load the EventWorkspace HYS_11388_event.nxs
2. using the `SumSpectra()` algorithm sum across each spectra in the workspace, assigning the result to a new workspace `Sum`
3. using the `Rebin()` algorithm rebin your new `Sum` workspace with bin width of 100 microseconds and events being preserved and assign it to another new workspace `binned`.

   Hint: your `Params` *should be set to 100 and* `PreserveEvents` *should equal* `True`
4. plot the spectrum of `binned` in the same was the previous part.
5. Filter out events recored before 4000 microseconds using the `FilterByTime()` algorithm into a workspace called `FilteredByTime`, and plot `FilteredByTime` to the same figure.
6. Using the Matplotlib function add a legend to your plot `ax.legend()` and then show your figure.

By following these steps you should end up with code that looks something like this:

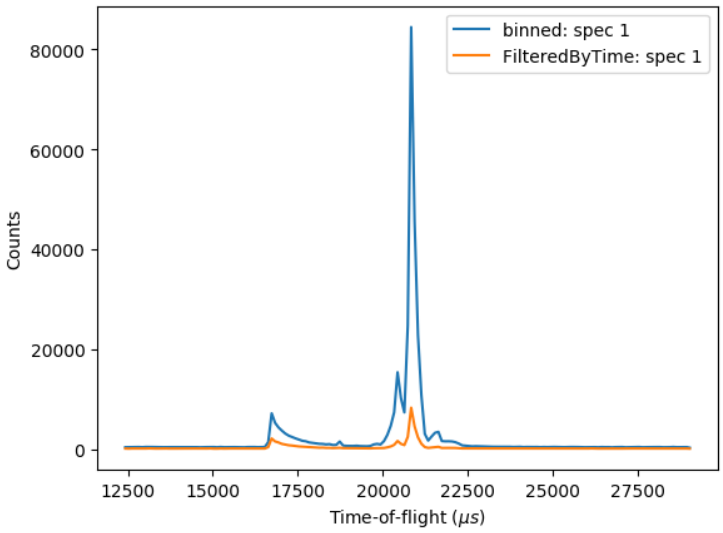
```
# The following line helps with future compatibility with Python 3
# print must now be used as a function, e.g print('Hello','World')
from __future__ import (absolute_import, division, print_function, unicode_literals)

# import mantid algorithms, numpy and matplotlib
from mantid.simpleapi import *

import matplotlib.pyplot as plt

import numpy as np

HYS_11388_event = Load("HYS_11388_event")
Sum = SumSpectra(HYS_11388_event)
binned = Rebin(Sum,Params = 100,PreserveEvents=True)
fig,ax = plt.subplots(subplot_kw = {"projection": "mantid"})
ax.plot(binned, wkspIndex = 0)
FilteredByTime = FilterByTime(binned, StartTime = 4000)
```

(continues on next page)

(continued from previous page)

```
ax.plot(FilteredByTime, wkspIndex = 0)
ax.legend()
fig.show()
```

That will produce a figure looking like this:



CHAPTER

TWO

INTRODUCTION TO PYTHON

The following set of slides provides an introduction to the Python language. It is designed for those with some programming experience and does not cover advanced material such as object oriented programming. For information regarding these topics see other references, e.g. "Core Python Programming", W. J. Chun, Prentice-Hall 2008; "Learning Python", M. Lutz, O'Reilly, 2007;.

Sections

2.1 Basic Language Principles

- Python is an interpreted language meaning there is no explicit compilation step.
- The code is simply executed "as-is". This, coupled with the fact that Python has a simple and easy-to-read syntax means it is an excellent choice for a scripting language.
- It includes all of the features that one expects from a programming language such as basic numerical types, a boolean, a string type and support for various operations upon them. The table below summarises how to use them in Python:

Type	Example	Python code
integer	5	x = 5
float	5.0	x = 5.0
boolean	True/False	x = True
string	'python'	x = 'python' or x = "python"

- The operations are supported as long as it makes sense for that type, e.g. there is no string division but + just means join the two strings together.
- Variable assignment is simple than in other languages as you do not have to declare the type and moreover it can be changed during execution, e.g.

```
import numpy
# Here x is initialized to 5 and Python then treats this as an integer
x = 5
# It can be incremented and have all of the expected operations applied to it
x += 1

# Later on it can be used for something else
# x is a string and adding a number produces an error
x = "a string"
x + 5

#will give you an error
#Traceback (most recent call last):
# File "<stdin>", line 1, in <module>
#TypeError: cannot concatenate 'str' and 'int' objects
```

- Comments are signified by the # symbol.
- Errors are signified by things known as **Exceptions**. In the above example a typical error message is shown which says that an exception of type `TypeError` occurred and the program needed to terminate (more on handling errors later).

Category:Tested Examples

2.2 Type Conversions

- In many circumstances we want to be able to perform operations on differing types, as in the earlier example where we tried to add the number '5' to a string variable. In this situation what we really meant was take a string representation of the number 5 and join that together with our first string. This is called a type conversion and for the string case it is achieved with the `str()`, e.g.

```
x = 'The meaning of life is ... '
answer = 42
y = x + str(answer)  # This converts the number 42 to a string and joins
                     # it with the first string and then assigns y to a
                     # new string containing the concatenated string
```

2.2.1 Printing

- The most useful situation for this function is when printing output to a screen. Printing is achieved with the `print` command that expects a string, e.g.
- in Python 2, you do not need to provide the brackets around the arguments of the print function, but we have included them here, so these examples will work with either python 2 or 3.

```
print('The meaning of life is ... ')
```

- By default the `print` command outputs a new line to the screen. To suppress this behaviour add a comma after the string:

```
# this now means the next print statement will continue from where this
# left the cursor
# In python 2 (in this case you cannot include the brackets)
print 'The meaning of life is ... ',
# the equivalent in python 3 is
#print('The meaning of life is ... ', end=' ')
```

- The comma can also be used to print several things to the screen on one line separated by a space:

```
x = 5
y = 6
# Python 2
print "X,Y:",str(x),str(y)    # prints 'X,Y: 5 6' with a newline
#print("X,Y:",str(x),str(y))    # prints '('X,Y:', '5', '6')' with a newline
# Python 3
#print ("X,Y:",str(x),str(y))  # prints 'X,Y: 5 6' with a newline
```

- As above printing other types is then simple with the `str()` function:

```
x = 'The meaning of life is ... '
answer = 42
print(x + str(answer))
```

Gives the output:

```
The meaning of life is ... 42
```

- If you want to avoid wrapping variables in str, you can use the format statement to insert values into a template string:

```
answer = 42
print('The meaning of life is ... {}'.format(answer))
```

Gives the output:

```
The meaning of life is ... 42
```

2.2.2 Converting Between Types

- Type conversions are not only important for converting to strings but are sometimes necessary to achieve expected answers, e.g.

```
x = 1/2
print(x)     # Prints 0!!! in Python 2 and 0.5 in Python 3
```

- In this case we have asked Python to take two integers (1,2) and then divide them and assign the result to `x`. The result is another integer which in this case is the integer part of the real number 0.5. If, as in this case, the real number is required then we must ask Python to use floating point numbers instead of integers. This can be achieved in two ways:

```
x = 1.0/2.0
print(x)

# or using the float function float()
x = 1
y = 2
print(float(x)/float(y))
```

Gives the output:

```
0.5
0.5
```

- The type conversion functions for the 4 basic types are:

Type	Function	Example
integer	int()	int(3.14159) => 3
float	float()	float(5) => 5.0
bool	bool()	bool(5) => True
string	str()	str(5) => '5'

- If a type cannot be converted then a 'ValueError' occurs (see error handling section).

Category:Tested Examples Category:Tested Examples

2.3 Sequence Data Types

- Python supports a range of types to store sequences. Here we will explore lists, sets, tuples and dictionaries. The string type is also considered a sequence but for our purposes here we shall consider it as a simple list of characters.
- A list is essentially an ordered collection of elements where the ordering is defined by the creator.
- Lists are created using square brackets to enclose the sequence of elements
- Elements can be added using the `append()` function
- Access to a specific element is done again by the square bracket operator by providing the required index of the element. **Note that in Python the first index is 0**
- Removing an element can be done by the `del` command or the `.remove()` function
- Replacement is also done using the square-bracket operator

```
lottery_numbers = [1,2,3,4,5,6]

bonus = 7
```

(continues on next page)

(continued from previous page)

```
lottery_numbers.append(bonus)

# print the first element
print(lottery_numbers[0])
# print the last element
print(lottery_numbers[6])
print(lottery_numbers)
lottery_numbers.remove(5) # Removes first occurrence of value 5 in list
del lottery_numbers[0]  # Removes the 0th element of the list
print(lottery_numbers)

lottery_numbers[3] = 42
print(lottery_numbers)
```

Givest the output:

```
1
7
[1, 2, 3, 4, 5, 6, 7]
[2, 3, 4, 6, 7]
[2, 3, 4, 42, 7]
```

- The square bracket operator also provides an operation known as *slicing*.
- Slicing allows contiguous portions of lists to be sectioned out by using `[i:j]` syntax where i and j are indexes.
- In this case it is helpful to think of the indices of the sequence slightly differently. Instead of thinking of each index as being assigned to a specific element within the list, think of them as being assigned to the boundaries of the elements, e.g.

- The output of a slice operation is then much more obvious as it simple slices out the items within the boxes contained by the given range.
- This also works with strings

```
my_list = ['M','A', 'N', 'T', 'I', 'D']
print(my_list[1:4])

my_string = 'MANTID'
print(my_string[1:4])
```

Gives the output:

```
['A', 'N', 'T']
ANT
```

- Lists can be sorted using the `sort()` function which modifies the list in place.

```
my_list = [5,4,3,2,7]
print(my_list)
my_list.sort()
print(my_list)
```

Gives the output:

```
[5, 4, 3, 2, 7]
[2, 3, 4, 5, 7]
```

- The default sorting criterion is less-than where items lower in the list are "less-than" items higher in the list. You can reverse this with,

```
l = [5,4,3,2,7]
l.sort(reverse=True)
print(l)  #prints list in descending order
```

Gives the output:

```
[7, 5, 4, 3, 2]
```

Category:Tested Examples

2.4 Control Structures

- As in any programming language there are mechanisms to control program flow: `if ... else`, `for ...`, `while`

2.4.1 Comparisons testing

- Control flow requires knowing how to compare values, for instance does one value equal another. In Python there are 6 operators that deal with comparisons:
  - == Tests for equality of two values, e.g. `x == 2`
  - != Tests for inequality of two values, e.g. `x != 2`
  - < Tests if lhs is less than rhs, e.g. `x < 2`
  - > Tests if lhs is greater than rhs, e.g. `x > 2`
  - <= Tests if lhs is less than or equal to rhs, e.g. `x <= 2`
  - >= Tests if lhs is greater than or equal rhs, e.g. `x >= 2`

2.4.2 Control blocks

- In Python bodies within control blocks are defined by indentation: spaces or tabs. Each level of indentation defines a separate control block. Tabs and spaces should never be mixed and each block must have the same indentation level.

2.4.3 If else

- The simplest control structure runs one of two different blocks of code depending on the value of a test,

```
x = 5
if x == 5:
    print('x has the value 5')
else:
    print('x does not equal 5')
x = 4
if x == 5:
    print('x has the value 5')
else:
    print('x does not equal 5')
```

Gives the output:

```
x has the value 5
x does not equal 5
```

- To test for ranges combine test with the `and` keyword

```
x = 2
if x > 0 and x < 5:
    print('x is between 0 and 5 (not inclusive)')
else:
    print('x is outside the range 0->5')
x = 7
if x > 0 and x < 5:
    print('x is between 0 and 5 (not inclusive)')
else:
    print('x is outside the range 0->5')
```

Gives the output:

```
x is between 0 and 5 (not inclusive)
x is outside the range 0->5
```

- Here we show an example of incorrect indentation and the subsequent error,

```
if x == 5:
    print('In x = 5 routine')
   print ('Doing correct thing')  # Results in error "IndentationError:
                                 # unindent does not match any outer
                                 # indentation level"
else:
    print('Everything else')
```

2.4.4 If ... elif ... else

- For situations with more than 2 possible outcomes there is an enhanced version of `if ... else` using the keyword `elif` to add additional blocks, e.g.

```
x = 3
if x == 1:
    print('Running scenario 1')
elif x == 2:
    print('Running scenario 2')
elif x == 3:
```

(continues on next page)

(continued from previous page)

```
    print('Running scenario 3')
else:
    print('Unrecognized option')
```

Gives the output:

```
Running scenario 3
```

- Tests can also be combined with the **not** to negate the test or with the **or** keyword to test one of two values.

```
x = 2
if x == 1 or x == 2:
    print('Running scenario first range')
```

Gives the output:

```
Running scenario first range
```

Category:Tested Examples

2.5 Looping

- Executing a given block of code a number of times is achieved using either the `for` or `while` statements

2.5.1 For . . . else

- The syntax to loop with a `for` statement is

```
for ''target'' in sequence:
    ''block-of-code-to-execute''
```

where *target* gets the value of the current number in *sequence*.

- A sequence from the loop can be a predefined sequence that has been constructed previously within the code or the `range(start, end, step)` function can be used to generate a sequence of numbers from *start* to *end-1* in steps of *step*. The default step value is *1*. E.g.

```
for i in range(0,10):
    print(i)
```

gives the output

```
0
1
2
3
4
5
6
7
8
9
```

```
for i in range(0,10,2):
    print(i)
```

gives the output

```
0
2
4
6
8
```

- Extra loop control is provided by the `break` and `continue` statements. `break` causes the loop to terminate at that point without executing any other code in the block,

```
nums = [1,2,3,-1,5,6]
list_ok = True
for i in nums:
    if i < 0:
        list_ok = False
        break

if list_ok == False:
    print('The list contains a negative number')
```

gives the output

```
The list contains a negative number
```

- `continue` causes execution to immediately jump to the next iteration of the loop,

```
nums =  [1,2,3,-1,5,6]
pos_sum = 0
for i in nums:
    if i < 0:
        continue
    pos_sum += i      # compound assignment means pos_sum = pos_sum + i

print('Sum of positive numbers is ' + str(pos_sum))
```

gives the output

```
Sum of positive numbers is 17
```

- An optional `else` clause can be added after the loop that will only get executed if the whole loop executes successfully,

```
for i in range(0,10):
    print(i)
else:
    print('done')      # Prints numbers 0-9 and the 'done'
```

gives the output

```
0
1
2
3
```

(continues on next page)

(continued from previous page)

```
4
5
6
7
8
9
done
```

```
for i in range(0,10):
    if i == 5:
        break
    print(i)
else:
    print('done')      # Prints numbers 0-4
```

gives the output

```
0
1
2
3
4
```

2.5.2 While

- While is another looping statement that simple executes until a given statement is False,

```
sum = 0
while sum < 10:
    sum += 1    # ALWAYS remember to update the loop test or it will
                # run forever!!

print(sum)       # Gives value 10
```

gives the output

```
10
```

- The `while` loop also supports the else syntax in the same manner as the `for` loop

Category:Tested Examples

2.6 Basic Python Exercises

- Here we will try some exercises to get to grips with the basics of Python

2.6.1 Exercise 1

- Write a program that prints out the square of the first 20 integers in a block such that the block is a rectangle of size 4x5. (Hint: One method uses the `%` (modulo) operator to test if a number is a multiple of another number. The answer is the remainder of the division of the lhs into whole parts, e.g. 4 % 1 = 0, 4 % 2 = 0, 4 % 3 = 1, 4 % 4 = 0)

2.6.2 Exercise 2

- Write a program that prints out the first 25 Fibonacci numbers. (The Fibonacci sequence starts as with `0,1` and next number is the sum of the two previous numbers)
  - Extend the program to also print the ratio of successive numbers of the sequence.

2.7 More Sequence Types

2.7.1 Tuples

- A tuple is similar to a list except that once a tuple has been created its value cannot be changed, only a new tuple created in its place. In this manner it can be thought of as a read-only list.
- They are created by enclosing the required sequence by parentheses `()`,

```
lottery_numbers = (1,2,3,4,5,6)
```

- Most list operations are still supported; index lookup, slicing, etc. However, these operations can only be used in a read-only sense, e.g.

```
print(lottery_numbers[0])      # prints 1
print(lottery_numbers[1:3])    # prints (2,3)
# Assignment not allowed
lottery_numbers[3] = 42      # gives error "TypeError: 'tuple' object does not␣
↪support item assignment"
```

- Tuples are most useful when returning values from functions as they allow returns of more than 1 item as will be shown later.

2.7.2 Dictionaries

- Python contains a mapping type known as a *dictionary*.
- The order of items in a dictionary is not user-defined (out of the box), i.e you cannot access a dictionary by index.
- Instead a dictionary maps a key to a value so that look up is by key, which may be a number but it is not limited to this, and not an index position.
- A dictionary is created using braces `{}` and can be created empty or initialized with elements. If initial elements are required then each key/value pair should be specified using `key:value` syntax and then each separated with a comma, e.g.
- Accessing a value is done by using square brackets where the argument is the key, e.g.

```
empty_dict = {}       # Empty dictionary
my_lookup = {'a' : 1, 'b' : 2} # A dictionary with two keys, each
                               # mapped to the respective value
print(my_lookup['b'])
```

Gives the output:

```
2
```

- Trying to retrieve a key that does not exist results in a *KeyError* and the program will halt,

```
empty_dict['a']    # Results in "KeyError: 'a'"
```

- Unlike tuples, dictionaries can be updated with new values, simply use the square brackets on the left-hand side of a assignment
- This syntax can also be used to replace a value that is already in the dictionary since every key has to be unique,

```
empty_dict = {}       # Empty dictionary
my_lookup = {'a' : 1, 'b' : 2} # A dictionary with two keys
empty_dict['a'] = 1
my_lookup['b'] = 3    # Replaces the value that was referenced by the key 'b' with the␣
↪new value 3
print(empty_dict['a'], my_lookup['b'])
```

- To remove a key/value from the dictionary, use the `del` command

```
del my_lookup['b']    # Removes the key/value pair with the specified key
my_lookup.clear()    # Empties the dictionary
```

- As a dictionary's order is undefined it is not possible to use slicing syntax as with lists and tuples.

2.7.3 Sets

- Sets are another unordered sequence of elements but unlike dictionaries, sets do not map keys to values instead they simply store a unique group of values.
- Unlike the other sequence types there is no special syntax for creating a set, there is instead the `set()` or `frozenset()` function. The difference simply corresponding to whether the structure is marked read-only after creation, where the `frozenset` is the read-only structure.
- To create a set simply pass a list or tuple to the `set()` function,
- Changing elements in a set is accomplished with the `add()` or `remove()` functions,

```
values = set([1,1,3])
print(values)
values.add(4)
values.remove(1)
print(values)
```

Gives the output:

```
set([1, 3])
set([3, 4])
```

- As with dictionaries, sets are unordered so it is not possible to access a set with a square bracket operators and they do not support slicing

2.7.4 Common Operations

- All sequence types support a number of common operations: `len()`, `x in s` and `x not in s`.
- `len()` gives the length of the sequence passed as its argument.
- `x in s` returns `True` if x is a member of the sequence s.
- `x not in s` returns `True` if x is not a member of the sequence s.

- Examples:

```
s = [1,2,3,4,5,6]       # Also works with all other sequence types
print(len(s))

test = 3 in s
print(test)
test = 7 not in s
print(test)
```

Gives the output:

```
6
True
True
```

2.8 Error Handling

- As we saw earlier Python can give error messages under certain conditions. This is known as raising an exception and Python's default behaviour is to halt execution. Exceptions are raised using the `raise` keyword:

```
# ...

# Execution stopped:
raise RuntimeError
```

- In some cases we may be able to manage the error and still continue. This is known as exception handling and is achieved using `try ... except` clauses,

```
arr = [1,2,3,4,5]

for i in range(6):
    try:
        val = arr[i]
        print(str(val))
    except IndexError:
        print('Error: List index out of range, leaving loop')
        break
```

- If an exception is raised then the code immediately jumps to the nearest `except` block. The output of the above code block is:

```
1
2
3
4
5
Error: List index out of range, leaving loop
```

- As with other control structures there is an extra `else` clause that can be added which will only be executed if no error was raised,

```
arr = [1,2,3,4,5]
value = 0
try:
```

(continues on next page)

(continued from previous page)

```
   value = arr[5]
except IndexError:
   print('5 is not a valid array index')
else:
   print('6th element is ' + str(value))
```

gives the output

```
5 is not a valid array index
```

- With a `try...except...else` structure only one of the except/else clauses will ever be executed. In some circumstances however it is necessary to perform some operation, maybe a clean up, regardless of whether an exception was raised. This is done with a `try...except...finally` structure,

```
value = 0
arr = [1,2,3,4]
element = 6
try:
    value = arr[element]
except IndexError:
    print(str(element) + ' is not a valid array index')
else:
    print(str(element + 1 ) + 'th element is ' + str(value))
finally:
    print('Entered finally clause, do cleanup ...')
```

gives the output

```
6 is not a valid array index
Entered finally clause, do cleanup ...
```

- Changing the value of the element variable between valid/invalid values will show that one of the except/else clauses gets executed and then the finally clause always gets executed.
- It is also possible to catch exceptions of any type by leaving off the specific error that is to be caught. This is however not recommended as then it is not possible to say exactly what error occurred

```
value = 0
arr = [1,2,3,4]
element = 6
try:
    value = arr[element]
except:      # Catch everything
    print("Something went wrong but I don't know what")
```

gives the output

```
Something went wrong but I don't know what
```

Category:Tested Examples

2.9 Working With Functions

- Functions are useful for splitting out common code that can be reused to perform repetitive tasks. In Python the keyword `def` is used to mark a function and as with the other control structures the block of code that defines

the function is indented

2.9.1 Simple Functions

- The simplest function takes no parameters and returns nothing, e.g.

```
def sayHello():
    print(' ----- HELLO !!! ----- ')
```

where now every call to `sayHello()` will produce the same message format and if the code requires updates then there is only one place to change.

2.9.2 Function arguments

- More useful functions will accept one or multiple arguments and perform some action based upon their value(s). Arguments are specified within the braces after the function name,

```
def printSquare(n, verbose):
    if verbose == True:
        print( 'The square of ' + str(n) + ' is: ' + str(n*n))
    elif verbose == False:
        print(str(n*n))
    else:
        print('Invalid verbose argument passed')

printSquare(2, True)  # Produces long string
printSquare(3, False) # Produces short string
printSquare(3,5)      # Produces error message
```

Gives the output:

```
The square of 2 is: 4
9
Invalid verbose argument passed
```

where we have combined functions and control structures to do something more useful.

- In the above function calls we specified the parameters in order but Python allows argument names to be specified when calling so that the order does not matter, e.g.

```
printSquare(verbose = True, n = 2)  # produces the same as
                                    # printSquare(2, True)
```

- Note that explicitly specifying an argument name forces all arguments that follow it to also have their name specified, e.g.

```
def foo(A, B, C, D, E):
    # ... Do something
    return

foo(1, 2, 3, 4, 5)      # Correct, no names given
foo(1, 2, 3, D=4, E=5)  # Correct as the first 3 get assigned to the first
                        # 3 of the function and then the last two are
                        # specified by name
foo(C=3, 1, 2, 4, 5)    # Incorrect and will fail as a name has been
```

(continues on next page)

(continued from previous page)

```
                    # specified first but then Python doesn't know
                    # where to assign the rest
```

This kind of calling can be useful when there is a function with many arguments and some at the end have default values (see below). The one that the user wishes to pick out can simple be given by name

2.9.3 Default Arguments

- In some situations extra function parameters maybe required for extra functionality but a user may want a certain default value to be specified so that the majority of the time the function call can be executed without specifying the parameter,where the second argument is now optional and will be assigned the given value if the function is called without it.

```
def printSquare(n, verbose = False):

    if verbose == True:
        print( 'The square of ' + str(n) + ' is: ' + str(n*n))
    elif verbose == False:
        print(str(n*n))
    else:
        print('Invalid verbose argument passed')
    return

printSquare(2)                              # Produces short message
printSquare(2, verbose = True)   # Produces long message
```

Gives the output:

```
4
The square of 2 is: 4
```

2.9.4 Return Values

- Most functions take in arguments, perform some processing and then return a value to the caller. In Python this is achieved with the `return` statement.

```
def square(n):
    return n*n

two_squared = square(2)
# or print it as before
print(square(2))
```

Gives the output:

```
4
```

- Python also has the ability to return multiple values from a function call, something missing from many other languages. In this case the return values should be a comma-separated list of values and Python then constructs a *tuple* and returns this to the caller, e.g.

```
def square(x,y):
    return x*x, y*y

t = square(2,3)
print(t)
# Now access the tuple with usual operations
```

Gives the output:

```
(4, 9)
```

- An alternate syntax when dealing with multiple return values is to have Python "unwrap" the tuple into the variables directly by specifying the same number of variables on the left-hand side of the assignment as there are returned from the function, e.g.

```
def square(x,y):
    return x*x, y*y

xsq, ysq = square(2,3)
print(xsq)  # Prints 4
print(ysq)  # Prints 9
# Tuple has vanished!
```

Gives the output:

```
4
9
```

Category:Tested Examples

2.10 Basic Python Exercises

2.10.1 Exercise 3

- Starting with your solution to exercise 2, rewrite your code so that you have a function called `fib` that accepts a number. The function should compute a fibonacci sequence containing this many elements and return them as a list.

2.10.2 Exercise 4

- Write a program that creates a dictionary and initializes it with 5 names/ID pairs.
  - Create a function that prints out the dictionary as a 2 columns: the first being the key and the second the value
  - Update the dictionary with another 5 name/values and reprint the table, making sure you understand the ordering within the map

2.11 Working With Files

- Reading/writing files in Python is made quite simple and both are controlled through strings.

2.11.1 Reading

- To be able to read a file we must first open a "handle" to it through the open command,

```
file = open('filename.txt', 'r')
```

- where the first argument is the full path to the file and the second argument is one of mode flags:
  - read - `'r'`
  - write - `'w'`
  - append - `'a'`
  - and binary - `'b'`.
- In our example we have opened the file in read mode and so only reading can take place. We will not discuss the binary mode flag, for further information please see one of the external references.
- Once opened the file can be accessed in a variety of ways. Firstly, the entire file can be read at once into a string using the `read()` command,

```
contents = file.read()
```

but this is only advised for very small files as it is an inefficient way of processing a file.

- Secondly, the `readline()` can be used to read a single line up to a new line character ('n'). Note that the newline is left at the end of the string and the returned string is only empty if the end of the file has been reached, e.g.

```
while True:
    line = file.readline()
    if line == "":
        break
    print(line)
file.close()
```

- The final way to read data from a file is to loop in a similar manner to looping over a sequence. This is by far the most efficient and has the cleanest syntax.

```
for line in file:
    print(line)
file.close()
```

- Note that you should always call the `close()` command on the file after it has been dealt with to ensure that its resources are freed.
- In the above examples the lines that have been read from the file still contain a newline character as the last character on the line. To remove this character use the `rstrip()` function with no arguments which strips whitespace characters from the right-hand side of the string

```
# First we will write a file to read in
import os

file = open('MyFile.txt', 'w')
file.write('ID WIDTH THICK HEIGHT\n')
file.write('a  1.0   2.0   3.0 \n')
file.write('b  2.0   3.6   1.2 \n')
file.close()
```

(continues on next page)

(continued from previous page)

```
# Now read it in
file = open('MyFile.txt')
for line in file:
    print(line)
file.close()

#Second try

#Reading agiain, but with rstrip
file = open('MyFile.txt')
for line in file:
    line = line.rstrip()
    print(line)
file.close()
```

This should give:

```
ID WIDTH THICK HEIGHT
a  1.0   2.0   3.0
b  2.0   3.6   1.2

ID WIDTH THICK HEIGHT
a  1.0   2.0   3.0
b  2.0   3.6   1.2
```

2.11.2 Writing

- A string is written to a file using the `write()` command once a file has been opened in write mode, 'w'. Note that the user controls the line formatting and using write does not automatically include a new line,

```
import os
file = open('NewFile.txt', 'w')
file.write('1 2 3 4 5 6\n')
file.write('7 8 9 10 11\n')
file.close()
file = open('NewFile.txt', 'r')
print(file.read())
```

Produces a file with the numbers on 2 separate lines

```
 1 2 3 4 5 6
 7 8 9 10 11
```

Category:Tested Examples

2.12 Using Modules

- Functions allow blocks of code to be separated into reusable parts that can be called from a user's script. Modules take this idea one step further and allow the grouping of functions and code that all perform similar tasks into a library, or in Python speak, *module*.
- A module has its definitions, the functions and code that make it up, within a separate file ending with a '.py' extension and this file is then *imported* into a user's current script. This is most easily demonstrated through an example. We will define a mathematics module called *mymath* and use this from a separate script.

```
## File: mymath.py ##
def square(n):
    return n*n

def cube(n):
    return n*n*n

def average(values):
    nvals = len(values)
    sum = 0.0
    for v in values:
        sum += v
    return float(sum)/nvals
```

- To use this module from a user script we need to tell the script about the module using the `import` statement. Once the definitions are imported we can then use the functions by calling them as `module.functionname`. This syntax is to prevent name clashes.

```
## My script using the math module ##
import mymath  # Note no .py

values = [2,4,6,8,10]
print('Squares:')
for v in values:
    print(mymath.square(v))
print('Cubes:')
for v in values:
    print(mymath.cube(v))

print('Average: ' + str(mymath.average(values)))
```

- Another variant of the import statement, `import` *''module''* `as` *''new-name''* can be used to have the module referred to under a different name, which can be useful for modules with long names

```
import mymath as mt

print(mt.square(2))
print(mt.square(3))
```

- It is possible to avoid the `module.functionname` syntax by using an alternate version of the import statement, `from` *''module''* `import *`. The functions can then be used as if they were defined in the current file. This is **dangerous** however as you do not have any protection against name conflicts when using multiple modules.

2.12.1 Importing Modules from Other Locations

If the module lives within the same directory as the script that is using it, then there is no problem. For system modules, these are pre-loaded into pythons sys.path list.

For example the numpy module location is here

```
import numpy
print(numpy.__file__)
```

There are several ways to make modules available for import.

Modifying the PYTHONPATH

This environmental variable is used by python on startup to determine the locations of any additional modules. You can extend it before launching your python console. For example on linux:

```
export PYTHONPATH=$PYTONPATH:{Path to mymodule directory}
python
```

On windows, it would look like this

```
SET PYTHONPATH=%PYTHONPATH%;{Path to mymodule directory}
python
```

Appending to sys.path

Another way to make modules available for import is to append their directory paths onto *sys.path* within your python session.

```
python
>>> import sys
>>> sys.path.append({Path to mymodule directory})
>>> import mymodule
```

Python's Standarad Library

- Python comes with a large number of standard modules that offer a wealth of functionality. The documentation for these modules can be found at http://www.python.org.
- They are used in exactly the same manner as user-defined modules using the `import` statement. Some examples of two useful modules are shown below.

datetime

- This module provides processing for date/time formats, reference: http://docs.python.org/2/library/datetime.html
- Example usage:

```
import datetime as dt
format = '%Y-%m-%dT%H:%M:%S'
t1 = dt.datetime.strptime('2008-10-12T14:45:52', format)
print('Day ' + str(t1.day))
print('Month ' + str(t1.month))
print('Minute ' + str(t1.minute))
print('Second ' + str(t1.second))

# Define todays date and time
t2 = dt.datetime.now()
diff = t2 - t1
```

Gives the output:

```
Day 12
Month 10
Minute 45
Second 52
```

os.path

- The os.path module provides facilities for path manipulation in an OS independent manner, reference http://docs.python.org/2/library/os.path.html
- Example usage: Open some files in a common directory

```
import os.path

directory = 'C:/Users/Files'
file1 = 'run1.txt'
fullpath = os.path.join(directory, file1)  # Join the paths together in
                                           # the correct manner

# print stuff about the path
print(os.path.basename(fullpath))  # prints 'run1.txt'
print(os.path.dirname(fullpath))  # prints 'C:\Users\Files'

# A userful function is expanduser which can expand the '~' token to a
# user's directory (Documents and Settings\username on WinXP  and
# /home/username on Linux/OSX)
print(os.path.expanduser('~/test')) # prints /home/[MYUSERNAME]/test on
                                   # this machine where [MYUSERNAME] is
                                   # replaced with the login
```

Numpy Introduction

- Python extension designed for fast numerical computation: http://numpy.scipy.org/
- Numpy provides multidimensional array objects, including masked arrays and matrices
- Numpy uses c-style arrays, which provide locality of reference for fast access
- Numpy comes with a vast assortment of inbuilt mathematical functions which can operate on the ndarrays. These functions are implemented in c, and optimised to give good performance
- Now available as standard within Mantid on all three platforms. Full tutorial: http://www.scipy.org/Tentative_NumPy_Tutorial.
- To use Numpy in a script you must first import the module at the top of your script

```
import numpy
```

Numpy Arrays

- Python lists are flexible as they can store any type.
- Iteration can be slow though as they are not designed for efficiency in this area.

- Numpy arrays only store a single type and provide optimized operations on these arrays.
- Arrays can be created from standard python lists

```
import numpy
x = numpy.array([1.3, 4.5, 6.8, 9.0])
```

- There is also a function, `arange`, which is numpy's counterpart to `range`, i.e. it creates an array from a start to and end with a given increment

```
x = numpy.arange(start=0.0, stop=10.0, step=1.0)
```

Numpy Functions

- Numpy arrays carry attributes around with them. The most important ones are:
  - ndim: The number of axes or rank of the array
  - shape: A tuple containing the length in each dimension
  - size: The total number of elements

```
import numpy

x = numpy.array([[1,2,3], [4,5,6], [7,8,9]]) # 3x3 matrix
print(x.ndim) # Prints 2
print(x.shape) # Prints (3L, 3L)
print(x.size) # Prints 9
```

Gives the output:

```
2
(3, 3)
9
```

- Can be used just like Python lists
  - x[1] will access the second element
  - x[-1] will access the last element
- Arithmetic operations apply element wise

```
import numpy
a = numpy.array( [20, 30, 40, 50] )
b = numpy.arange( 4 )
c = a-b
print(c)
```

Gives the output:

```
[20 29 38 47]
```

Built-in Methods

- Many standard numerical functions are available as methods out of the box:

```
x = numpy.array([1,2,3,4,5])
avg = x.mean()
sum = x.sum()
sx = numpy.sin(x)
```

- A complete list is available at: http://docs.scipy.org/doc/numpy/reference/

2.13 Pattern Matching With Regular Expressions

- A common file processing requirement is to match strings within the file to a standard form, for example a file may contain list of names, numbers and email addresses. A email extraction would need to extract only those entries that matched which look like an email address.
- Regular expressions, commonly called regexes, are ideally suited for this task and although they can become very complex it is also possible to perform many tasks with some relatively simple expressions.
- At their simplest, a regular expression is simply a string of characters and this string would then match with only that exact string, e.g.

```
string in file: 'email'
regex: 'email' # This is a regular expression but albeit not a very
               # useful one as only matches with one word!
```

- In reality regexes are used to search for a string that "has the form" of the regular expression, as in the above email example. For this to be possible we need to define some syntax that lets us specify things such as 'a number is in a range', 'a letter is one of a set', 'a certain number of characters' etc. For this to work some characters are considered special and when used in conjunction with each other they let the user specify the correct search criteria.
- Here we will examine a few special characters, for a complete reference see http://www.regular-expressions.info/reference.html or search for "regular expression reference" online.

2.13.1 Special Characters

- An asterisk $\star$ specifies that the character preceding it can appear zero or more times, e.g,

```
regex: 'a*b'
test: 'b'         # Matches as there are no occurrences of 'a'
test: 'ab'        # Matches as there is a single 'a'
test: 'aaaaaaaab' # Matches as there are multiple occurrences of 'a'
test: 'aaaabab'   # Matches as there is an occurrence of a string of
                  # a's followed by a b
```

- A range of characters, or a "character class" is defined using square brackets `[]`, e.g.

```
regex: '[a-z]'
test: 'm' # Matches as it is a lower case letter
test: 'M' # Fails as it is an upper case letter
test: '4' # Fails as it is a number
```

- Several ranges can be specified such that they are all checked, e.g.

```
regex: '[a-z,A-Z,0-9]'
test: 'm'  # Matches!
test: 'M'  # Matches!
test: '4'  # Matches!
test: 'mm' #Fails as there are two characters
```

- Combining ranges and the asterisk allows us to specify any number of alphanumeric characters!, e.g.

```
regex: '[a-z,A-Z,0-9]*'
test: 'mm'    # Matches
test: 'a0123' # Matches
```

- To specify an exact number of characters use braces `{}`, e.g.

```
regex: 'a{2}'
test: 'abab'  # Fails as there is not two consecutive a's in the string
test: 'aaaab' # Matches
```

- For more complicated regular expressions it is not obvious whether you have written the expression correctly so it can be useful to check that it matches as you expect. For such tests there are online tools available such as the regex tester at http://www.regular-expressions.info/javascriptexample.html. Simply type in your regex and a test string and it will tell you a match can be found within your string.

2.13.2 Regular Expressions in Python

- Python contains a regular expression module, called `re` that allows strings to be tested against regular expressions with a few lines of code. Reference: http://docs.python.org/2/library/re.html
- The `compile` function also takes another optional argument controlling the matching process, all of which are documented at the above location. Here we pass the `RE.IGNORECASE` option meaning that a case-insensitive match is performed.
- Example:

```
import re

def checkForMatch(checker, test):
    if checker.match(test) != None:
        print('String matches!')
    else:
        print('String does not contain a match')
# End of function definition

checker = re.compile('[a-z]')
checkForMatch(checker, 'a')
checkForMatch(checker, '9')

checker = re.compile('[a-z]', re.IGNORECASE)
checkForMatch(checker, 'a')
checkForMatch(checker, 'A')
```

Gives the output:

```
String matches!
String does not contain a match
String matches!
String matches!
```

- Below we provide a more complex example of using regular expressions and a place where they would actually be used in a practical sense. The scenarios concern parsing a file with multiple lines of the form

```
Running 13 tests.............OK!
```

where the line has to start with the word 'Running' and end with the word 'OK!' or the test is considered a failure.

- Regular expressions make parsing such a file a relatively simple matter once the regular expression is known. Here is the full example:

```
import re

filetestsRun = 'testResults.log'
f = open(filetestsRun,'r')
reTestCount = re.compile("Running\\s*(\\d+)\\s*test", re.IGNORECASE)
reCrashCount = re.compile("OK!")
reFailCount = re.compile("Failed\\s*(\\d+)\\s*of\\s*(\\d+)\\s*tests", re.IGNORECASE)
testCount = 0
failCount = 0
testsPass = True
for line in f.readlines():
    m=reTestCount.search(line)
    if m:
        testCount += int(m.group(1))
        m=reCrashCount.search(line)
        if not m:
            failCount += 1
            testsPass = False
    m=reFailCount.match(line)
    if m:
        # Need to decrement failCount because crashCount will
        # have incremented it above
        failCount -= 1
        failCount += int(m.group(1))
        testsPass = False

f.close()

print("Tests Passed: {}".format(testsPass))
print("Tests Failed: {}".format(failCount))
print("Total Tests: {}".format(testCount))
```

- The loop keeps track of test crashes and failures by using regular expressions to match the required text within each line of the file

Category:Tested Examples

2.14 Basic Python Exercises

- This exercise aims to perform a moderately complicated set of operations using most of the topics covered over the course.
- For this exercise you will require 5 text files, each containing 2 columns of data: a timestamp and a value.

2.14.1 Exercise

1. Build a list containing the 5 filenames of the text files that are going to be used. (Hint: Can be done by hand or using the os.listdir('dirpath') function in the os module)
2. Add a bogus file name that doesn't exist to the list (so that we have to do some error handling)
3. Loop over the list and for each file (Remember here that we have a non existent file in the list and calling open on this will result in an IOError exception that needs to be dealt with)
    1. Open the file;
    2. Loop over each line;
    3. Split the line up into sections (Hint: The string has a `.split()` function that splits the string on whitespace and gives back a list with each section as an element of the list)
    4. Convert the second column value into an float
    5. Keep track of the values for each line and compute an average for the file.
4. Finally, print out a list of file,average-value pairs

2.15 Solutions To Exercises

2.15.1 Exercise 1

- Write a program that prints out the square of the first 20 integers in a block such that the block is a rectangle of size 4x5. (Hint: One method uses the `%` (modulo) operator to test if a number is a multiple of another number. The answer is the remainder of the division of the lhs into whole parts, e.g. 4 % 1 = 0, 4 % 2 = 0, 4 % 3 = 1, 4 % 4 = 0)

```
# Write a program that prints out the square of the first
# 20 integers a block such that the block has a dimension of 4x5.

# range(i,j) produces a list of numbers from i -> j-1
#python 2
for i in range(1,21):
    print str(i*i).center(3), # center is a function for strings that centers the␣
↪contents to the given width
    if i % 4 == 0:
        print

#Python 3
#for i in range(1,21):
#    print(str(i*i).center(3),end=' ') # center is a function for strings that␣
↪centers the contents to the given width
#    if i % 4 == 0:
#        print()

# ------------ Produces -------------
# 1   4   9   16
# 25  36  49  64
# 81 100 121 144
#169 196 225 256
#289 324 361 400
```

2.15.2 Exercise 2

- Write a program that prints out the first 25 Fibonacci numbers. (The Fibonacci sequence starts as with `0,1` and next number is the sum of the two previous numbers)
  - Extend the program to also print the ratio of successive numbers of the sequence.

```
# prev_2 - One before previous number
# prev_1 - Previous number
prev_2, prev_1 = 0, 1

# Already have first 2 terms above
print(prev_2)
print(prev_1)

# Now the next 23 terms (range(0,23) will run the loop 23 times)
for i in range(0,23):
    current = prev_2 + prev_1
    # Ratio to previous
    ratio = float(current)/prev_1
    print(str(current) + " ratio to previous= " +str(ratio))
    # Move the previous markers along one for the next time around
    prev_2 = prev_1
    prev_1 = current
```

2.15.3 Exercise 3

- Starting with your solution to exercise 2, rewrite your code so that you have a function called `fib` that accepts a number. The function should compute a fibonacci sequence containing this many elements and return them as a list.

```
# Write a program that builds a list of the first 20 Fibonacci numbers, then
## Use the list to print out the value of the ratio of successive numbers of the␣
↪sequence,
##   printing out the final value.
## Extend the program so that the Fibonacci list is calculated in a function that␣
↪takes the
##   number of required values as a parameter and returns the list.

# Function to calculate the first n fibonacci numbers
# and return them as a list
def fib(nfibs):
    if nfibs == 0:
        return []
    elif nfibs == 1:
        return [0]
    else:
        pass
    # First two numbers
    fibs = [0,1]
    if nfibs == 2:
        return fibs
    for i in range(2, nfibs):
        fibs.append(fibs[i-2] + fibs[i-1])
    return fibs
```

(continues on next page)

(continued from previous page)

```
#### fib ends here ####

# Print out successive ratio remembering that the first number is a zero
nfibs = 20
fib_nums = fib(nfibs)
for i in range(1,nfibs):
    try:
        numerator = fib_nums[i]
        denominator = fib_nums[i-1]
        ratio = float(numerator)/denominator
    except ZeroDivisionError:
        print('Warning: Invalid ratio: ' + str(numerator) + '/' + str(denominator))
    else:
        print('Ratio ' + str(numerator) + '/' + str(denominator) + ': ' + str(ratio))

##### Produces #####
#Warning: Invalid ratio: 1/0
#Ratio 1/1:  1.0
#Ratio 2/1:  2.0
#Ratio 3/2:  1.5
#Ratio 5/3:  1.66666666667
#Ratio 8/5:  1.6
#Ratio 13/8:  1.625
#Ratio 21/13:  1.61538461538
#Ratio 34/21:  1.61904761905
#Ratio 55/34:  1.61764705882
#Ratio 89/55:  1.61818181818
#Ratio 144/89:  1.61797752809
#Ratio 233/144:  1.61805555556
#Ratio 377/233:  1.61802575107
#Ratio 610/377:  1.61803713528
#Ratio 987/610:  1.61803278689
#Ratio 1597/987:  1.61803444782
#Ratio 2584/1597:  1.6180338134
#Ratio 4181/2584:  1.61803405573
```

2.15.4 Exercise 4

- Write a program that creates a dictionary and initializes it with 5 names/ID pairs.
  - Create a function that prints out the dictionary as a 2 columns: the first being the key and the second the value
  - Update the dictionary with another 5 name/values and reprint the table, making sure you understand the ordering within the map

```
# Write a program that creates a dictionary and initializes it with 5 names/ID pairs.
## Create a function that prints out the dictionary in a nicely formatted table;
## Update the dictionary with another 5 name/values and reprint the table,
##   making sure you understand the ordering within the map.

def formatLine(cola, colb, width):
    return cola.center(width) + '|' + colb.center(width)

# A simple two cloumn print out
```

(continues on next page)

(continued from previous page)

```
def outputStore(store):
    print('Phonebook contains {} entries:'.format(len(store)))

    # Do a quick sweep to find out the longest name
    col_width = 0
    for k in store:
        if len(k) > col_width:
            col_width = len(k)
    col_width += 5

    # Header
    print('-'*col_width*2)
    print(formatLine('Name', 'Ext.', col_width))
    print('-'*col_width*2)
    for k ,v in store.items():
        print(formatLine(k, str(v), col_width))

phone_book = {'Martyn Gigg' : 1234, 'Joe Bloggs' : 1233, 'Guido Van Rossum' : 4321,
↪'Bob' : 2314, 'Linus Torvalds' : 4132 }
outputStore(phone_book)

# Update Dictionary (replacing one person's phone number
new_entries = {'Bjarne Strousoup' : 9876, 'Bill Gates' : 9898, 'Steve Jobs' : 7898, \
              'Bob' : 9871, 'Dave' : 7098 }

phone_book.update(new_entries)
outputStore(phone_book)

#------------ Produces --------------------

#Phonebook contains 5 entries:
#----------------------------------------
#         Name         |         Ext.
#----------------------------------------
#         Bob          |         2314
#       Joe Bloggs     |         1233
#     Linus Torvalds   |         4132
#    Guido Van Rossum  |         4321
#      Martyn Gigg     |         1234
#Phonebook contains 9 entries:
#----------------------------------------
#         Name         |         Ext.
#----------------------------------------
#    Guido Van Rossum  |         4321
#      Martyn Gigg     |         1234
#       Steve Jobs     |         7898
#    Bjarne Strousoup  |         9876
#       Joe Bloggs     |         1233
#     Linus Torvalds   |         4132
#          Dave        |         7098
#       Bill Gates     |         9898
#          Bob         |         9871
```

2.15.5 Exercise 5

1. Build a list containing the 5 filenames of the text files that are going to be used. (Hint: Can be done by hand or using the os.listdir('dirpath') function in the os module)
2. Add a bogus file name that doesn't exist to the list (so that we have to do some error handling)
3. Loop over the list and for each file (Remember here that we have a non existent file in the list and calling open on this will result in an IOError exception that needs to be dealt with)
    1. Open the file;
    2. Loop over each line;
    3. Split the line up into sections (Hint: The string has a `.split()` function that splits the string on whitespace and gives back a list with each section as an element of the list)
    4. Convert the second column value into an float
    5. Keep track of the values for each line and compute an average for the file.
4. Finally, print out a list of file,average-value pairs

```
## Exercise 5
#   1.  Build a list containing the 5 filenames of the text files that are going to␣
↪be used.
#       (Hint: Can be done by hand or using the os.listdir() function in the os␣
↪module)
#   2. Add a bogus file name that doesn't exist to the list (so that we have to do␣
↪some error handling)
#   3. Loop over the list and for each file (Remember here that we have a non␣
↪existent file in the list and calling open
#      on this will result in an IOError exception that needs to be dealt with)
#         1. Open the file;
#         2. Loop over each line;
#         3. Split the line up into sections (Hint: The string has a .split()␣
↪function that splits the string on
#            whitespace and gives back a list with each section as an element of the␣
↪list)
#         4. Convert the second column value into an integer
#         5. Keep track of the values for each line and compute an average for the␣
↪file.
#   4. Finally, print out a list of file,average-value pairs
import os

file_dir = "C:\\MantidInstall\\data\\"
file_names = os.listdir(file_dir)
file_names.append('nonexistant.txt')
average_store = {}
print('Computing average for log files in directory "' + file_dir + '"')
for name in file_names:
    # Skip all no text files
    if name.endswith('.txt') == False:
        continue
    try:
        file_handle = open(os.path.join(file_dir,name), 'r')
    except IOError:
        print('\tError: No such file: "' + name + '". Skipping file')
        continue
    print('\tReading file',name)
```

(continues on next page)

(continued from previous page)

```
    # At this point we have an open file
    average = 0.0
    nvalues = 0
    line_counter = 1
    for line in file_handle:
        columns = line.split()
        if len(columns) == 2:
            average += float(columns[1])
            nvalues += 1
        else:
            print('\tWarning: Unexpected file format encountered in file {0}  on line
↪{1}'.format(name,line_counter))
        line_counter += 1

    average /= nvalues
    average_store[name] = average
    file_handle.close()

# Print out file averages
column_width = 30
print('')
print('-'*column_width*2)
print('File'.center(column_width) + '|' + 'Average'.center(column_width))
print('-'*column_width*2)
for key, value in average_store.iteritems():
    print(key.center(column_width) + '|' + str(value).center(column_width))
```