Science & Technology Facilities Council

Technical Report
RAL-TR-2014-010

An Overview of the Recent Development of Indirect Inelastic Data Analysis in Mantid

Jackson, S

September 2014

ISSN 1358-6254

An Overview of the Recent Development of Indirect Inelastic Data Analysis in Mantid

Samuel Jackson
ISIS Facility
STFC Rutherford Appleton Laboratory
samuel.jackson@stfc.ac.uk

September 1, 2014

Contents

1 Abstract

Mantid [1], the Manipulation and Analysis Toolkit for Instrument Data, is an open source and cross platform data analysis application specialising in neutron and muon scattering data. Recent development work has been undertaken to improve support for indirect geometry spectrometers, particularly in regard to those based at ISIS. This document aims to provide an overview of the current progress of development and describe from a technical perspective the procedures used in data reduction and analysis. The final section of the document then loosely outlines areas to focus on in further development in relation to the requirements described by instrument scientists.

2 Introduction

Mantid [1] is an open source, cross platform framework for the analysis of neutron and muon scattering data. The project is primarily written in C++ with a Python API. The project includes a GUI based off of the QtiPlot project and is primarily written using the Qt library.

Mantid aims to provide a single platform for neutron and muon scattering data analysis to all facilities across the world. It is developed by two teams of developers, one based at the ISIS facility at Rutherford Appleton Laboratory UK, the other at the Spallation Neutron Source (SNS) at Oak Ridge, Tennessee, USA. Recently additional partners from the Institut Laue-Langevin located in Grenoble and the Paul Scherrer Institute in Wrenlingen, Switzerland have also contributed.

The project is currently still under heavy development and provides regular incremental releases for evaluation and feedback from instrument scientists and scientific community in general and is therefore constantly evolving with each release. Until recently, Mantid operated on a three month release cycle. This has now been changed to so that the project releases every four months, with a longer emphasis on the testing and integration phase of development.

Time-of-flight neutron scattering instruments supported by Mantid, such as those based at ISIS and the SNS, can broadly be split into three categories based on the geometry of the instrument. In a direct instrument geometry the incident energy is fixed using choppers to select a specific incident wavelength directed towards the sample. Indirect or inverted geometry instruments use a polychromatic incident beam to scatter from the sample and select the final energy using a crystal analyser. Triple axis spectrometers use crystals both to moderate the incident beam and select the final energy of scattered neutrons [2]. This document focuses on the development progress of indirect geometry instruments within Mantid.

Up until now the indirect geometry section of Mantid has largely been based off of the the existing MODES application which in turn was based on OpenGenie [3] and the IRIS Data Analysis package [4]. The majority of the functionality has now been integrated into the Mantid framework in one form or another and we are now beginning to reach a point where we can expand the existing functionality further.

This document aims to give an overview of the current stage of development for the indirect geometry instruments section of the application as it currently exists in as much detail as possible to provide a historical snapshot of the progress so far. The remainder of the document then provides a weak outline of suggested directions for future development.

3 Indirect Geometry Instruments

Indirect geometry time-of-flight instruments, like the IRIS, OSIRIS, TOSCA, and VESUVIO spectrometers at ISIS are those in which the incident beam is polychromatic and crystal analysers are typically used to determine the final energy. Together these instruments cover a broad energy range. IRIS and OSIRIS operate in a low energy range of -0.4 - 0.4 meV (depending on the analyser used) and can be used to carry out Quasi-elastic neutron scattering around the elastic line [8, 9]. This is useful for studying the effects of diffusion and tunneling. TOSCA covers the 0 - 500 meV range for investigating the lattice and intra-molecular structure of matter [10]. VESUVIO operates at high energies of 5 - 150 eV and is used to perform deep inelastic neutron scattering [14]. Diagram 3.1 compares the energy range covered by the indirect geometry instruments at ISIS and the applicable techniques at each energy range.

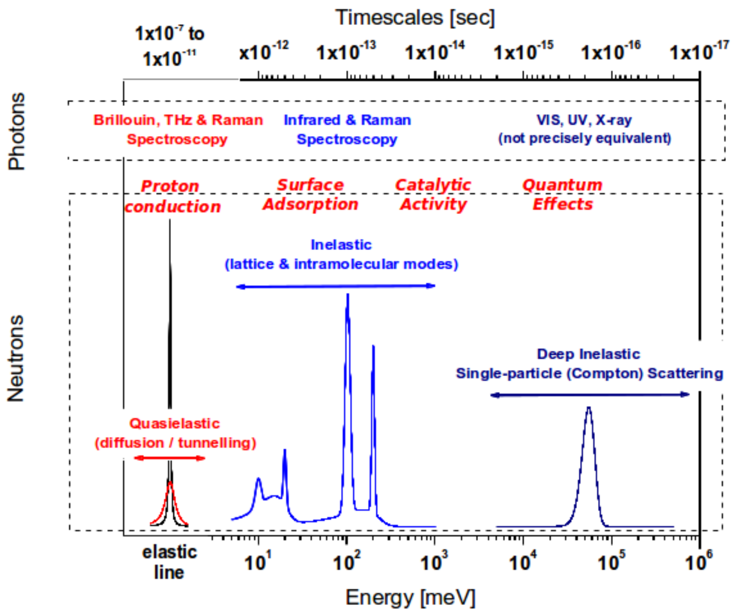

Timescales [sec]
1x10^-7 to 1x10^-11
x10^-12
1x10^-13
1x10^-14
1x10^-15
1x10^-16
1x10^-17
Photons
Neutrons
Brillouin, THz & Raman Spectroscopy
Infrared & Raman Spectroscopy
VIS, UV, X-ray (not precisely equivalent)
Proton conduction
Surface Adsorption
Catalytic Activity
Quantum Effects
Inelastic (lattice & intramolecular modes)
Deep Inelastic Single-particle (Compton) Scattering
Quasielastic (diffusion / tunnelling)
elastic line
10^1
10^2
10^3
10^4
10^5
10^6
Energy [meV]

Figure 3.1: *Energy ranges of various inelastic techniques. The ISIS instruments IRIS and OSIRIS cover the Quasi-elastic range, TOSCA covers the inelastic range and VESUVIO covers deep inelastic neutron scattering.*

IRIS and OSIRIS share the N6 beamline in target station 1 at the ISIS facility. Both instruments are fed via curved guides from the source moderator. Two choppers positioned at 10m and 6.3m from the instruments define the incident wavelength band and prevent frame overlap [5]. TOSCA is located on the N4 beamline in target station 1 with a incident flight path of 17m and also has a chopper on the incoming beamline to prevent overlap and remove fast neutrons [6]. VESUVIO is located on the S2 beamline and has an incident flight path of 11.005m [7].

The final energy for inelastic instruments such as IRIS, OSIRIS and TOSCA is determined by the reflections from analyser crystals by means of Bragg scattering. IRIS and OSIRIS both have pyrolytic graphite crystal analysers with reflection planes 002 and 004 delivering a final energy of E_f=1.845 and 7.38 meV respectively [8, 9]. This design leads to a resolution of 17.5 and 54.5 μeV for the 002 and 004 reflections on IRIS and 25 and 99 μeV on OSIRIS. IRIS additionally has a mica (muscovite or fluorinated) crystal analyser bank with reflections 002, 004, and 006, and E_f corresponding to 0.207, 0.826, and 1.86 meV respectively which provide better resolutions (at the cost of intensity) of 11μeV (006), 4.5μeV (004) and 1.2μeV (002) compared to graphite. Recently, OSIRIS has had an upgrade which expands the operational energy range to -0.25 to 25 meV with the pyrolytic graphite analyser with reflection 002. The primary setting for both instruments is graphite with reflection 002. TOSCA has just a pyrolytic graphite analyser with 002 reflection [6] and achieves a resolution of roughly 1.25% of the measured energy transfer [10].

VESUVIO operates on the principle of neutron Compton scattering and uses a pair of gold foils which can be moved into different positions for use with the foil cycling technique described in [11] which produces a neutron absorption resonance at 4.9 eV $\pm$ 0.15 eV, emitting a γ-ray cascade which is detected by the instrument's Yttrium Aluminium Perovskite γ-ray detectors [12].

4 Theory

Neutron scattering experiments are used to probe the structure of materials at a fundamental level. The results of such experiments can be used to learn a great deal about the internal structure of a sample and how it can behave under different conditions. In a neutron scattering experiment, a beam of neutrons is produced at a source (such as the one provided by ISIS) and directed to the various instruments attached to the beam line. These neutrons then scatter from the sample placed in the instrument. The angle and energy at which scattered neutrons are detected can then be used to deduce the structure and dynamics of the sample.

There are two main types of neutron scattering: elastic and inelastic. Elastic scattering is the special case when the final energy is equal to the incident energy and there is no energy transfer and the kinetic energy of the incident neutrons remain unchanged. Inelastic scattering is the more general case where a change in the kinetic energy of the incident neutron occurs [13].

4.1 Data Reduction Theory

Before data analysis can begin, the raw data collected from the instrument must be converted to an instrument independent function that describes the proportion of incident neutrons scattered with a given momentum and energy transfer known as $S(\mathbf{Q},\omega)$. Obviously, in order to calculate $S(\mathbf{Q},\omega)$ the quantities of $\mathbf{Q}$ (wavevector transfer) and ω (energy transfer) must first be derived.

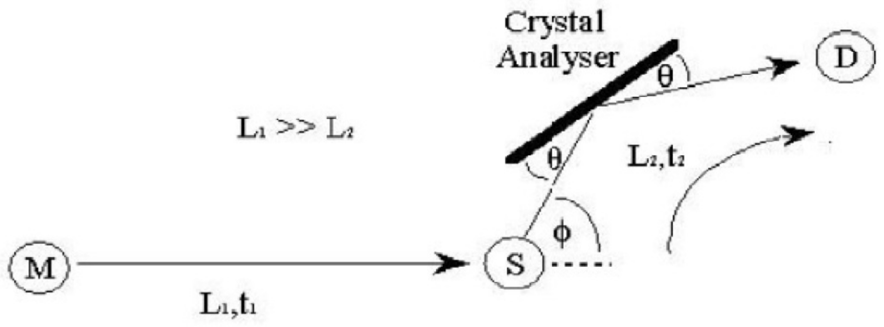


Figure 4.1: *Diagram of the basic principle of an indirect geometry neutron scattering experiment. M is the moderator, S is the sample, and D is the detector [5]*

In order to convert the raw data collected from the instrument from units of time-of-flight to energy transfer known parameters of the geometry of the instrument instrument are used, as illustrated in figure 4.1. The raw time-of-flight data is recorded in counts per microsecond and can be calculated using the parameters of the instrument from the time-of-flight equation:

$$t = \frac{L_1}{V_1} + \frac{L_2}{V_2} \tag{4.1}$$

Where L_1 is the distance between moderator and sample, L_2 is the distance between the sample and detector, and V_1 and V_2 are the incident and final velocities of a neutron. The final energy detected by a detector can be written in terms of a neutron's mass (m_n) and velocity (v) from the sample to the detector:

$$E_f = \frac{1}{2}m_n v^2 = \frac{1}{2}m_n (L_2/t_2)^2 \tag{4.2}$$

Which can also be written in terms of the distance between sample and detector (L_2) and the corresponding time-of-flight over this distance (t_2). To calculate the transfer of energy knowledge of the flight path between the moderator and sample (L_1) and the time of flight over this distance (t_1) can be used to calculate the incident energy using equation above. In indirect geometry instruments, the final energy is fixed by the crystal analysers which filters any neutrons with wavelengths not satisfying the Bragg condition:

$$\lambda = 2dsin(\theta) \tag{4.3}$$

Where λ is the neutrons wavelength, d is the spacing between planes in the crystal lattice, and θ is the scattering angle. The transfer of energy is then simply the difference between the incident and final energy:

$$\Delta E = E_i - E_f = \frac{1}{2}m_n[(L_1/t_1 - t_2)^2 - (L_1/t_2)^2] \tag{4.4}$$

This is the basic theory behind the convert to energy reduction routine described in section 5.1.1. Momentum transfer $\mathbf{Q}$, is defined as the difference between the final and incident wave vectors.

$$\mathbf{Q} = k_f - k_i \tag{4.5}$$

Where k_i and k_f are the incident and final wave vectors. The magnitude of $\mathbf{Q}$ is defined as:

$$Q^2 = k_i^2 + k_f^2 - 2k_ik_fcos2\theta \tag{4.6}$$

Where θ is the scattering angle of the sample [13]. In reality, the $S(\mathbf{Q},\omega)$ cannot be directly measured by experiment because of the resolution of the instrument. Instead, the quantity measured from the instrument (denoted $I(\mathbf{Q},\omega)$) is a convolution between the scattering function and the instrument resolution:

$$I(\mathbf{Q},\omega) = S(\mathbf{Q},\omega) \otimes R(\mathbf{Q},\omega) \tag{4.7}$$

Where $S(\mathbf{Q},\omega)$ is given by:

$$S(\mathbf{Q},\omega) = \Sigma_j \frac{b_j}{M_j}^2 Q^2 < u_j >^2 e^{-2W} \frac{Z(\omega)}{\hbar\omega} \tag{4.8}$$

Where b_j is the neutron scattering length of the atom, M_j is the mass of the atom, $< u_j >$ is the amplitude of vibration of each atom, $Z(\omega)$ density of states of a molecule, and e^{-2W} is the Debye-Waller factor [5] to correct for thermal energy in which W is equal to:

$$W = \frac{1}{2} < u_{MSD}^2 > Q^2 \tag{4.9}$$

And $< u_{MSD}^2 >$ is the mean squared displacement of the scattering atom. The mean squared displacement is the quantity that is analysed using the MSDFit routine described in section 6.1.2.

This theory is only relevant for IRIS, OSIRIS, and TOSCA spectrometers. The theory behind data reduction for the deep inelastic neutron scattering instrument VESUVIO is completely different and is based on the validity of the impulse approximation at sufficiently high energy and momentum transfer. A detailed description of the theory behind data analysis and reduction for VEUSVIO can be found in Ref. [14].

4.1.1 Diffraction Reduction

In the case of diffraction experiments, there is no need to convert to energy transfer or $S(\mathbf{Q},\omega)$. Instead, the results are converted from the raw time-of-flight data to d-spacing, the distance between the planes in a lattice. Results are calculated using known parameters of the instrument via the time-of-flight equation (equation 4.1) and Bragg's law (equation 4.3). Diffraction experiments may be run either on their own or in parallel with spectroscopy.

At pulsed time-of-flight sources such as ISIS a wavelength scanning method can be used to determine the scattering vector by varying the wavelength of the incident wavevector and keeping the scattering angle constant thereby satisfying the Bragg condition [2]:

$$\mathbf{Q} = 4\pi sin(\theta)/\lambda \tag{4.10}$$

Using equations 4.3 and 4.1, the d-spacing of a sample is given as a function of the scattering angle θ and distances between the source and sample L_0 and sample and detector L_1:

$$d = \frac{2t sin(\theta) M_n (L_0 + L_1)}{h} \tag{4.11}$$

Where t is the time-of-flight, M_n is the mass of a neutron and h is the Planck constant.

4.2 Data Analysis Theory

4.2.1 Convolution Theory

Data analysis of inelastic scattering data is primarily concerned with separating the scattering function from the instrument's resolution function in the relation defined by equation 4.7 . In theory, these can then be separated through the deconvolution of the two functions. There are several different functional forms that may occur depending on the sample used:

- The simplest case is when there is a simple diffusive motion within the sample $S(\mathbf{Q},\omega)$ and $R(\mathbf{Q},\omega)$ will both have the form of a Lorentzian and hence the convolution of the two will also be a Lorentzian.
- The second case is when both the scattering function and the resolution function have simple functional forms. For example, when $S(\mathbf{Q},\omega)$ is a Lorenztian and $R(\mathbf{Q},\omega)$ is a Gaussian.
- The final case is when $R(\mathbf{Q},\omega)$ does not have a simple functional form, and so must be convoluted numerically with the $S(\mathbf{Q},\omega)$.

The typical way to handle the latter case is through fitting the appropriate model to the convoluted data and is the purpose of the ConvFit interface described in section 6.1.5 which provides a least-squares fit to the convolution of the measured data and the resolution.

An alternative method of deconvolution of the scattering and resolution functions is through the intermediate scattering function, denoted $I(\mathbf{Q},t)$ which is proportional to the Fourier transform of the scattering and resolution functions:

$$I(\mathbf{Q},t) = S(\mathbf{Q},t) \times R(\mathbf{Q},t) \tag{4.12}$$

As the resolution of the instrument is known, the intermediate scattering function can be divided by the resolution and back transformed to obtain the scattering function. This method has the advantage in that the form of the function does not have to be assumed, but the back transformation from $S(\mathbf{Q},t)$ to the true scattering function is problematic due to the phase problem [13] and the propagation of statistical errors [15]. The Fury routine (section 6.1.3) provides an interface to perform a Fourier transform on the measured data and resolution, but does not provide an automated back transformation because of the aforementioned issues.

4.2.2 Bayesian Analysis

An alternative method to model fitting is the use of Bayesian methods to select the best model for the data based on a choice of possible functions. One method described by Sivia [16] selects the most likely choice of model from sum of a δ-function and zero or more Lorentzians convolved with the resolution function of the instrument. Using this method, the number of Lorentzians is iteratively varied and the optimum number selected as the final model. An extension of this method replaces the sum of Lorentzians with a single stretched exponential function. For each group of spectra, the best value for the β and Γ parameters is obtained [3]. This model can be used to fit the shape resulting from polymer samples [4, 17, 18], and is the basis behind the Stretch program described in section 6.2.3.

From the width parameters of the selected model diffusion can be interpreted. This is typically done by modelling jump diffusion where it is assumed an atom remains at a given site for a length of time before moving rapidly to a new site in a time that is negligible to the time spent being stationary. There are several different models for diffusion depending on the application. The Chudley-Elliott form assumes the atom jumps from one point to its nearest neighbour site l distance away in a Bravis lattice [19] and has the form:

$$\Gamma(Q) = (\hbar/\pi t) \cdot (1 - sin(Ql)/Ql)/\tau \tag{4.13}$$

The Hall-Ross [20] form models the diffusive jump motion as several steps and does oscillatory motion at alternate sites.

$$\Gamma(Q) = (\hbar/\pi t) \cdot (1 - exp(-lQ^2))/\tau \tag{4.14}$$

Teixeria's model [21] is used to study the dynamics of super cooled water and has the form:

$$\Gamma(Q) = DQ^2/(1 + DQ^2\tau) \tag{4.15}$$

Where the diffusion constant D is equal to:

$$D =< l^2 > /6\tau \tag{4.16}$$

Finally the Fick diffusion model [22] has the form:

$$\Gamma(Q) = DQ^2 \tag{4.17}$$

Each of these functions are implemented as part of the JumpFit interface (section 6.2.4), but are also available through the generic fit wizard in Mantid.

4.2.3 Absorption Corrections

A sample typically must be corrected for absorption by the sample and by its container. Typically a sample will need to be corrected for flat plate geometry [23] or cylindrical geometry [24]. The methods for cylindrical geometry used in Mantid have been adapted from the methods described in the ATLAS manual [24].

Absorption corrections use the formalism specified by Paalman and Pings [25] to describe the attenuation factors of a sample/container. They are denoted by $A_{i,j}$ where i refers to scattering and j refers to attenuation. For example, $A_{s,sc}$ is the attenuation factor for scattering in the sample plus container. The scattering cross sections for the sample and container are denoted as Σ_s and Σ_c. The scattering from an empty container is then given by:

$$I_c = \Sigma_c A_{c,c} \tag{4.18}$$

Scattering from the sample plus the container is then given by:

$$I_{sc} = \Sigma_s A_{s,sc} + \Sigma_c A_{c,sc} \tag{4.19}$$

The scattering cross section for the sample can the be calculated as:

$$\Sigma_s = (I_{sc} I_c A_{c,sc}/A_{c,c})/A_{s,sc} \tag{4.20}$$

There are two routines in Mantid which perform these corrections. Calculate Corrections calculates the attenuation parameters $A_{i,j}$, while Apply Corrections uses them to calculate Σ_s based on the geometry as described here.

5 Data Reduction with Mantid

Mantid provides a collection of graphical user interfaces for the specialised indirect inelastic data reduction routines under the menu *Interfaces >Indirect*. The majority of these routines are based on or directly ported from the MODES 3 and IRIS Data Analysis (IDA) packages [3, 4]. As such, the core of the functionality of these routines has mostly remained the same, but the implementation has changed. This section outlines the current development of each of the interfaces and routines available.

The Convert to Energy interface provides the initial entry point for data into Mantid for indirect instruments. The functionality of this interface is currently also shared with the direct instruments, although a separation of the interfaces is planned in future releases.

The code for data reduction programs are primarily stored in the inelastic scripts folder of the Mantid installation (*<Installation Directory>/Mantid/scripts/Inelastic*). The energy conversion reducer is in a file called inelastic_indirect_reducer.py while the diffraction reducer is in IndirectDiffractionReduction.py. The reduction steps used by both are in inelastic_indirect_reduction_steps.py. The generic reducer class is stored in the reduction folder (*<Installation Directory>/Mantid/scripts/reduction*). Miscellaneous energy reduction routines are stored in IndirectEnergyConversion.py which is also in the inelastic scripts folder.

5.1 Convert To Energy

5.1.1 Energy Transfer

The Energy Transfer tab on the Convert To Energy interface is the starting point for data reduction. The energy transfer routine, as the name suggests, is used to convert a raw file representing a sample run from the time-of-flight measurement to units of energy transfer and also performs general preprocessing of the data to get it ready for analysis.

The structure of the underlying energy transfer reduction routine is based on the older Mantid concept of a Reducer class which has since been superseded by the concept of algorithms and work-flow algorithms. The Reducer class contains a list of ReductionSteps which represent a single logical operation to be carried out as part of the reduction. ReductionSteps just execute Mantid algorithms and contain the actual implementation details of the operation to be performed.

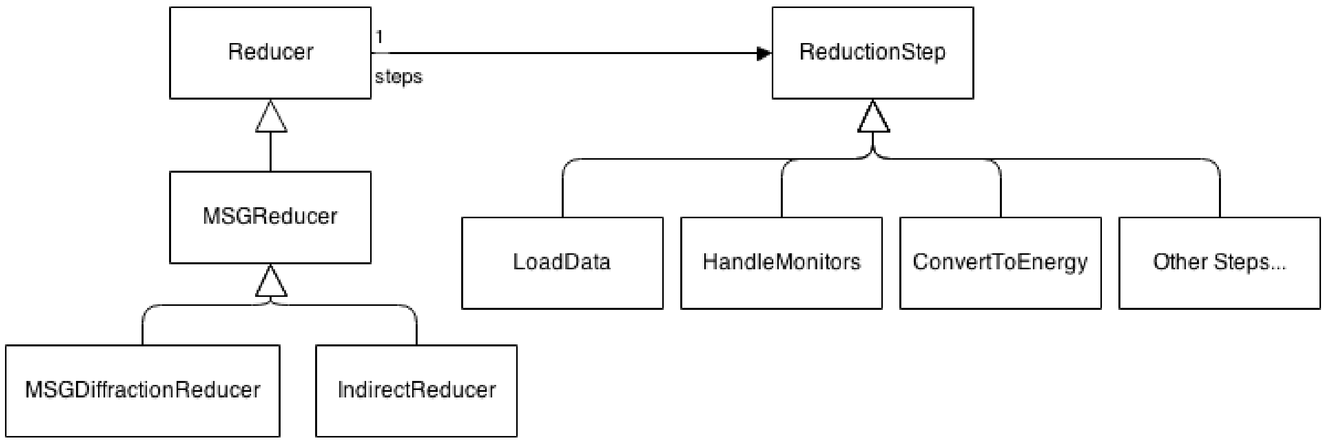


Figure 5.1: *Diagram showing the class structure for the inelastic reducers and accompanying reduction steps.*

Figure 5.1 shows the class structure of indirect reducers and a subset of the reduction steps that are defined for indirect energy reduction. In indirect data analysis there are two concrete reducer objects. One for energy conversion (IndirectReducer) and one for diffraction reduction (MSGDiffractionReduction). This section will just focus on the IndirectReducer. For discussion on diffraction reduction see section 5.2.

The IndirectReducer object defines what steps should be executed as part of a reduction work-flow and is the point at which the parameters gathered from the user interface are passed to the program. After setting up each reduction step with the relevant parameters, the reducer then iterates over the list of steps and executes each one in turn on the sample run to be reduced. Reducer objects also have the option to define a preprocessing function which is only executed once regardless of the number of runs being processed. The MSGReducer, which is the superclass of both the IndirectReducer and the MSGDiffractionReducer, uses this to execute the loading step to get the sample run(s) to be reduced into memory before continuing with the rest of the reduction on a per run basis.

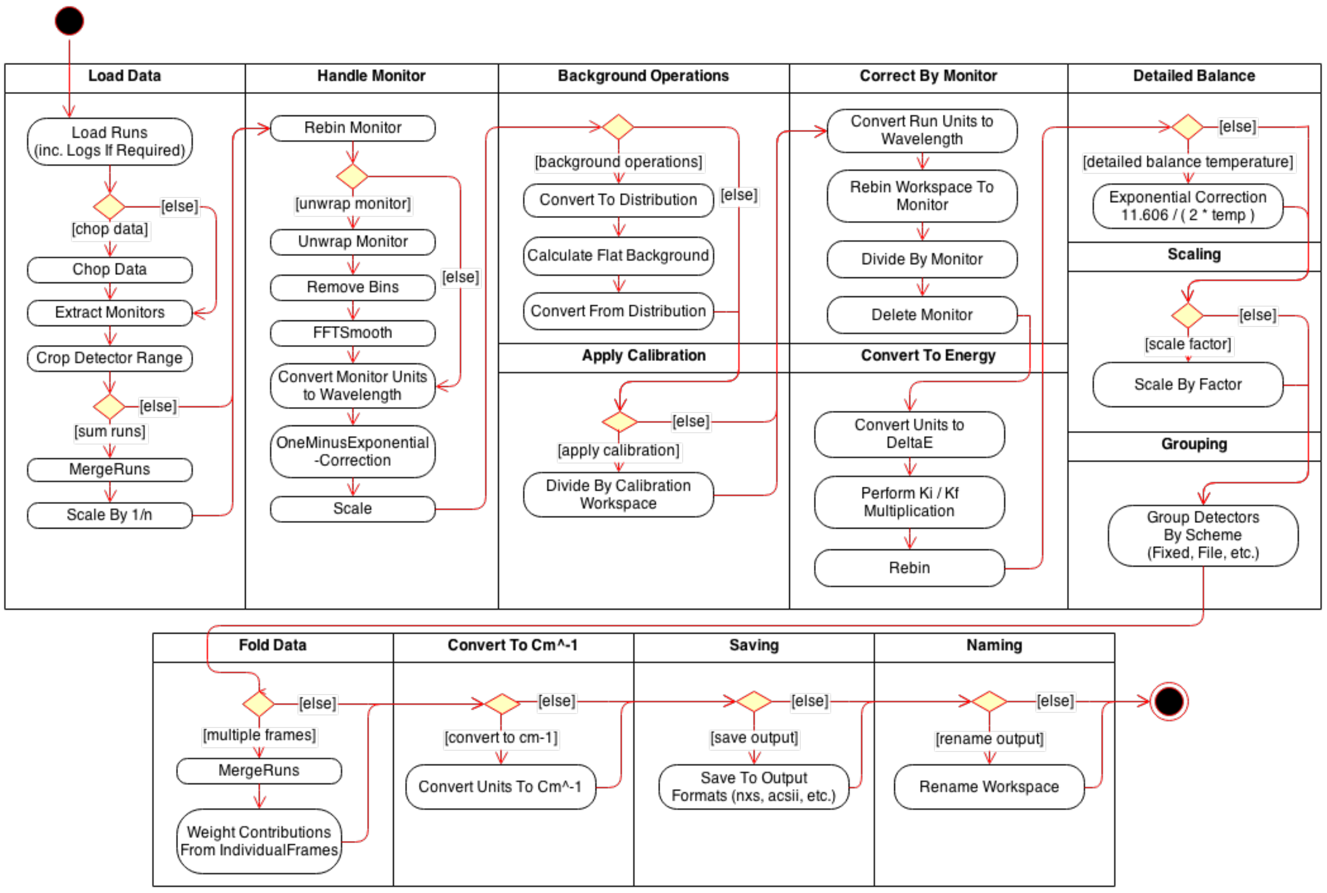


Figure 5.2: *Activity diagram showing the flow of execution for energy transfer reduction in release 3.2.*

The steps involved in energy transfer reduction are shown in activity diagram 5.2 along with the major operations performed within each step and the flow of execution for the reduction. The responsibility of each of the steps are as follows:

- **Load Data** - The load data step loads each individual sample run from file using the correct loader for the currently selected instrument. If the option is selected it will also attempt to load the logs for the runs as well. If the current instrument has the parameter Workflow.ChopDataIfGreaterThan defined in its instrument parameter file (IPF), the reduction step splits the data into multiple frames using the ChopData algorithm. The routine will then extract the monitor from the loaded workspace and crop the workspace to the desired detector range. If the sum option is checked the runs will be merged into a single workspace and averaged, otherwise subsequent steps are executed on each run individually.

- **Handle Monitor** - Next the reducer prepares the monitors for correction in a later step. This involves rebinning the monitor according to the step size defined in the IPF. Next it will unwrap the monitor, if required, using the UnwrapMonitors algorithm to convert the workspace to have common bins within the maximum wavelength range given a reference flightpath between source, sample and the first detector. It then tidies up the monitor by removing the bin at the joining wavelength and smooths it Using FFTSmooth. Finally it converts the units of the workspace to wavelength, performs an exponential correction for the thickness, attenuation, and area of the monitor and scales it if a scaling factor is defined in the IPF.

- **Background Operations** - This step simply calculates a flat background for the detector data if parameters are defined for the background range by calculating the mean of the bins in this range before dividing by the width of the x range.
- **Apply Calibration** - If a calibration file was supplied as a parameter to the reduction, the reducer will divide the workspace by the calibration run at this point. This just uses the standard Mantid Divide algorithm.
- **Correct By Monitor** - The reducer then corrects the workspace by the preprocessed monitor by converting the units to wavelength then rebinning it to and dividing it by the monitor.
- **Convert To Energy** - The conversion of the workspace to units of energy is then performed and multiplied by k_i/k_f to transform the differential scattering cross section into a dynamic structure factor. The workspace is then rebinned according to the user defined rebin string if one was supplied.
- **Detailed Balance** - The detailed balance step performs an exponential correction on the data if a temperature parameter was defined by the user. This uses the ExponentialCorrection algorithm.
- **Scaling** - This step will scale the workspace by an arbitrary factor supplied by the user.
- **Fold Data** - If the data was split into multiple frames during the loading step the fold data step is performed. The individual frames are merged back into a single workspace (using the MergeRuns algorithm) and the workspace is scaled by averaging the contributions of each individual frame across the x range.
- **Convert To** Cm^{-1} - If the option was selected, the workspace is converted to units of wavenumber.
- **Saving** - If any save options were selected, the reduced workspace is saved in the desired formats to the default save directory.
- **Naming** - Finally, the reduced workspace is renamed according to the instrument-analyser-refection convention used in the Mantid indirect framework.

Note that the instrument parameters used in energy transfer reduction are defined in files called <instrument_name>_Parameters.xml and <instrument_name>_<analyser>_<reflection>_Parameters.xml where the instrument name, analyser, and reflection are replaced by the actual values used (e.g. IRIS_graphite_002_Parameters.xml) and are kept in the instrument folder within the Mantid installation directory.

5.1.2 Calibration

The calibration section of the convert to energy interface can be used to create calibration and resolution files using vanadium sample runs which are used as part of data reduction and analysis. Calibration workspaces are generated by creating and executing a single reduction step object (without creating a reducer object) called CreateCalibrationWorkspace. The procedure for generating a single calibration file is to load the runs and if there is more then one of them merge them and scale the output workspace by a factor of $1/\,number\,of\,runs$. A flat background is then calculated using the background range supplied by the user. The routine then integrates over the peak range (again supplied by the user) and scales the resulting workspace by an arbitrary factor. An activity diagram for the routine is shown in figure 5.3.

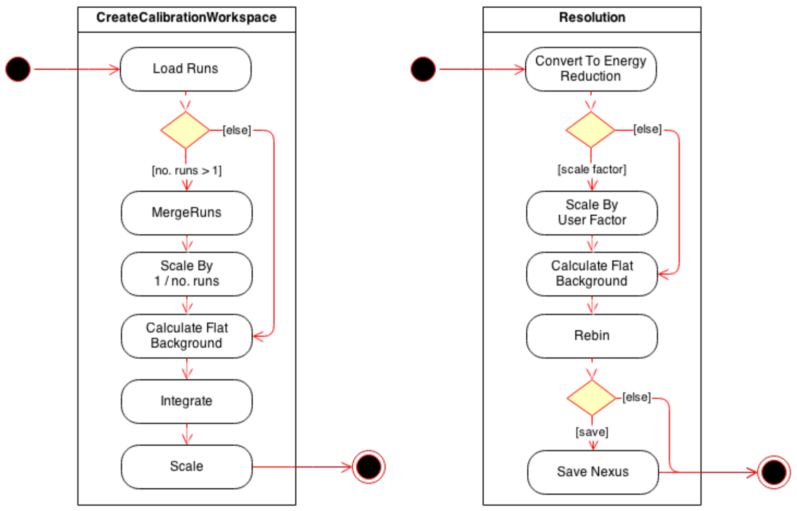


Figure 5.3: *Activity diagram showing the major steps of execution performed when creating a calibration and resolution workspace.*

As mentioned, the calibration interface can also create resolution files for later use in data analysis. This routine, like many within the indirect code base, is separate from the reducer-reduction step framework and is just a python function which is executed with parameters supplied from the interface. To create the resolution file, the routine first takes the vanadium run supplied on the calibration interface and converts it to energy using the energy transfer reducer described in section 5.1.1, but sets the detector mapping to use all detectors and will sum multiple runs if more than one run is used. It then, like calibration, calculates a flat background from a range supplied by the user, rebins it according to the parameters specified for the instrument, and optionally saves the workspace to file in the default save directory.

5.1.3 Diagnostics

The diagnostics routine (called Slice in IndirectEnergyConversion.py, named after the older routine from MODES [3]) provides an integration over a specified time-of-flight range. The slice routine begins by loading input files and a calibration file if supplied. If a calibration workspace is supplied, it is cropped to remove the monitors and match the range of the input raw file. Each sample workspace is cropped to the desired number of spectra and divided by the calibration workspace if it exists. The workspace is then integrated across the supplied time range. If two ranges are given the second range is used to remove a flat background prior to the integration using CalculateFlatBackground. Finally, the resulting workspace is transposed using the algorithm of the same name.

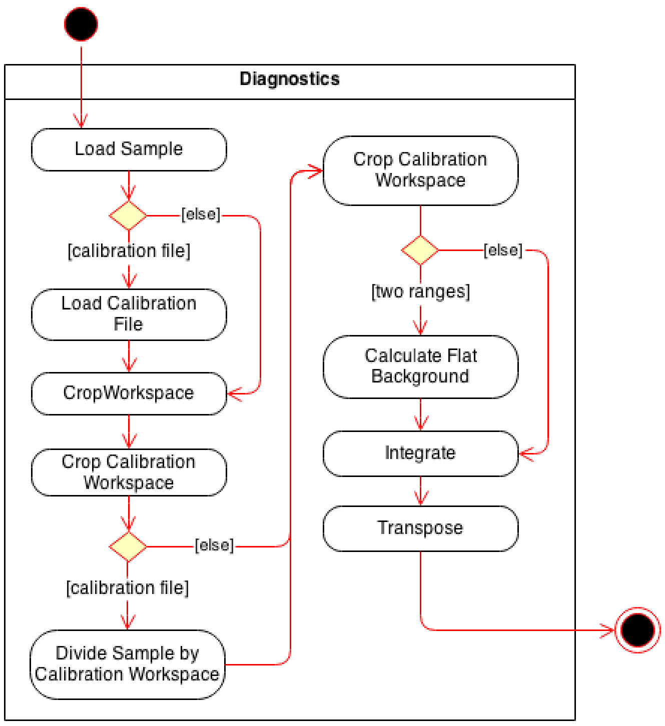


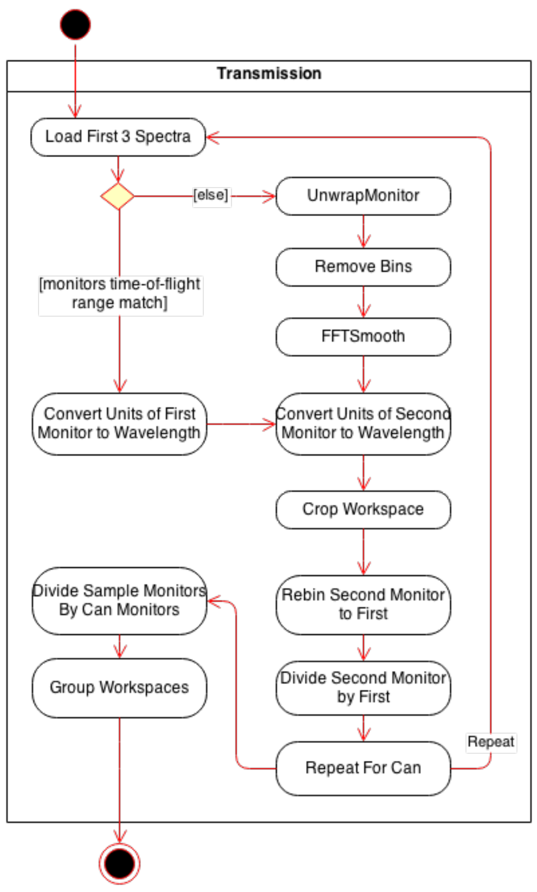
Figure 5.4: *Activity diagram showing an activity diagram for the diagnostics routine.*

5.1.4 Transmission

The transmission routine is a simple program used to calculate the measured transmission of a sample using the incident and transmission monitors. Using a run of both the sample and the can, monitors from each raw file are extracted and converted to units of wavelength. The transmission monitor for each run is then normalised by the incident monitor over the common wavelength range. Finally, the sample is divided by the can to give the sample transmission as a function of wavelength.

The procedure for doing this using the Mantid API is shown in the activity diagram 5.5. The routine assumes that the first two spectra in a raw file are the monitors (as is generally the case for ISIS indirect instruments) where the first is the incident monitor and the second is the transmission monitor. It also loads the third spectrum in the raw file (which should be actual detector data) to use as a reference time range.

If the first value on the detector's x-axis matches the corresponding value on the incident monitor it simply converts the monitor workspace to units of wavelength. If it does not match the monitor must first be unwrapped using the UnwrapMonitor algorithm with a hard coded parameter for the flight path of 38.76. Then it corrects the join wavelength for the UnwrapMonitor algorithm by removing the bins at the join with a linear interpolation method that uses the FFTSmooth algorithm for smoothing. It then crops the incident monitor to the shortest wavelength of the two monitors and rebins the second monitor to match. Finally the transmission monitor is divided by the incident monitor.



Transmission
Load First 3 Spectra
[else]
UnwrapMonitor
[monitors time-of-flight range match]
Remove Bins
FFTSmooth
Convert Units of First Monitor to Wavelength
Convert Units of Second Monitor to Wavelength
Crop Workspace
Divide Sample Monitors By Can Monitors
Rebin Second Monitor to First
Group Workspaces
Divide Second Monitor by First
Repeat
Repeat For Can

Figure 5.5: *Activity diagram showing the flow of execution for the Transmission routine.*

This procedure is repeated for both the sample and the can file provided by the user. After the monitors for both the sample and can have been converted the sample transmission monitor is divided by the can transmission monitor. Finally, a group workspace containing the sample transmission monitor, can transmission monitor, and sample transmission monitor with can removed is created.

5.1.5 $S(Q, w)$

In this interface, reduced files created as part of an energy transfer reduction can be converted to an $S(Q, w)$ workspace. The Mantid framework has a collection of three algorithms that are used to convert a workspace from energy transfer to $S(Q, w)$ depending on the choice of rebinning to use. The mapping between $S(Q, w)$ algorithms and the rebin types are given in table 5.1. The code for running an $S(Q, w)$ conversion is entirely contained within the Indirect interface where it dynamically builds a string of python code that is then executed. Optionally, the user many rebin the workspace in energy before hand. If this is case the routine simply calls the rebin algorithm with supplied parameters.

Rebin Type	Algorithm Used
Centre	SofQW
Parallelepiped	SofQW2
Parallelepiped / Fractional Area	SofQW3

Table 5.1: *Table showing the relationship of rebin types to SofQW algorithms*

The approach to rebinning taken by each of the algorithms listed in table 5.1 is as follows:

- **SofQW** - The simplest version of the algorithm iterates over each individual detector in a workspace with units of energy transfer and calculates the value of Q using parameters from the instrument and detector and rebins by using the centre point of the bin between two points in the workspace.

- **SofQW2** - This version calculates the width of theta over all detectors and uses the maximum and minimum as the offset for the value of theta in combination with two points in energy transfer to define a quadrilateral for rebinning. This algorithm inherits from the Rebin2D algorithm and uses its methods to rebin both axes without the use of the fractional area option.
- **SofQW3** - The third version of the algorithm is an extension of the second. Again, this algorithm inherits from Rebin2D but uses the fractional area option to rebin. This algorithm also takes into account the angular width based on each individual detector and in the case of PSDs also accounts for the value of phi.

5.1.6 $S(Q, w)$ Moments

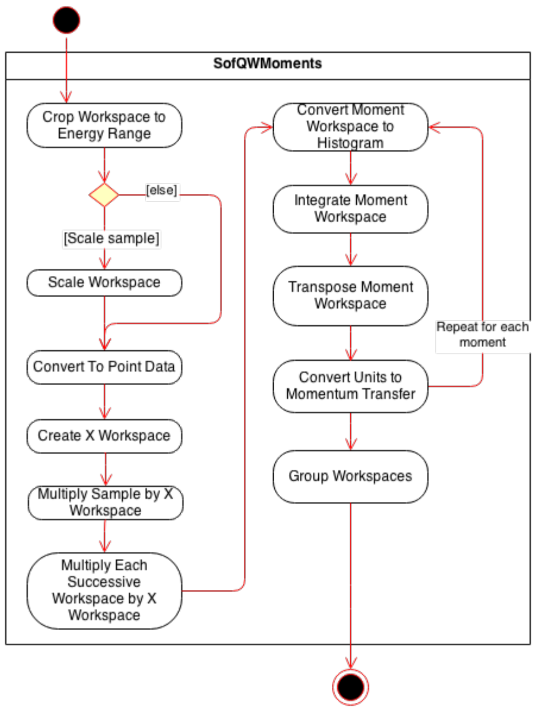

SofQWMoments
Crop Workspace to Energy Range
[else]
[Scale sample]
Scale Workspace
Convert To Point Data
Create X Workspace
Multiply Sample by X Workspace
Multiply Each Successive Workspace by X Workspace
Convert Moment Workspace to Histogram
Integrate Moment Workspace
Transpose Moment Workspace
Repeat for each moment
Convert Units to Momentum Transfer
Group Workspaces

Figure 5.6: *Activity diagram showing the flow of execution for the SofQWMoments algorithm.*

This is a new addition in Mantid 3.2 and provides a simple routine for calculating the first four statistical moments of an $S(Q, w)$. This is implemented as a standard Mantid algorithm (SofQWMoments) which is created and called from the interface though the algorithm manager and produces a group workspace with one workspace for each moment for every detector and which has been transposed in order to make visualisation of the workspace easier.

This algorithm proceeds by first cropping the sample to the desired energy range and optionally scaling it by an arbitrary factor. The sample is then converted from histogram data to point data. A workspace containing the x values of the sample workspace on both axes is then created using the CreateWorkspace algorithm. The sample workspace is then multiplied by this workspace using the Multiply algorithm. The result of this operation is then multiplied by the X workspace. This process repeats on each successive resulting workspace for every moment. These resulting workspaces are then converted back to histogram data, integrated, transposed, and converted to units of momentum transfer. Finally each of the moment workspaces is placed in a single group workspace.

5.1.7 Support for the ILL

The convert to energy reduction currently supports instrument at ISIS and the SNS. In release 3.2 a lot of work has been done in order to increase the level of support for ILL instruments. Prior releases of Mantid added a loader for ASCII data files from the ILL to the Indirect Load ASCII interface and are stored in the file IndirectNeutron.py. In 3.2 loaders for various ILL instruments (including the very useful LoadILLIndirect) have been added. Additionally, a new reduction algorithm for IN16B, called IndirectILLReduction, has been created to give support for energy transfer reduction within Mantid. Future releases aim to integrate this algorithm into the current reduction chain and therefore completely remove Indirect Load ASCII from the system.

5.2 Diffraction Reduction

Another key part of data reduction is the ability to reduce diffraction data. The process of preparing diffraction runs for analysis is much simpler than an energy transfer reduction. Indirect diffraction reduction uses the same reducer system described in section 5.1.1 but uses the MSGDiffractionReducer object (see figure 5.1) and a different selection of reduction steps.

The instrument parameters used in the reduction are defined in files called <instrument_name>_diffraction_diff_spec_Parameters.xml and <instrument_name>_diffraction_diff_only_Parameters.xml (if the instrument supports diff only mode).

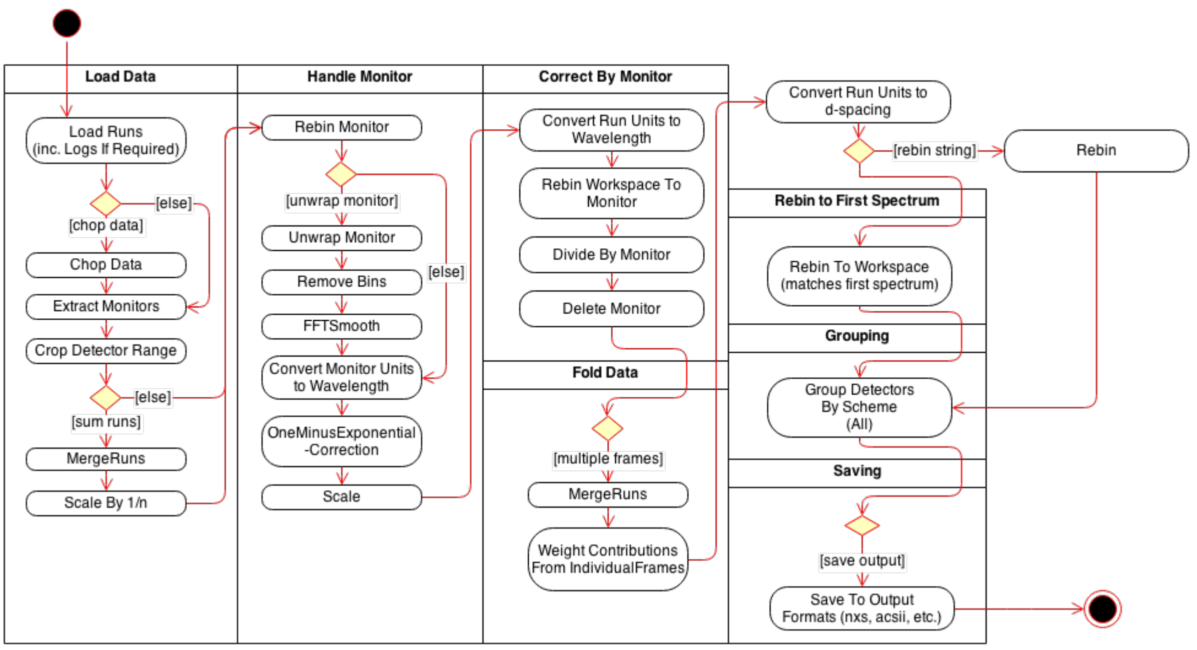


Figure 5.7: *Activity diagram showing the flow of the Diffraction Reduction in release 3.2. Steps outside of boxes are individual algorithms that are not part of a reduction step.*

- **Load Data** - The first step in diffraction reduction is to load all of the data files to be reduced. This reduction step is exactly the same as the one described in section 5.1.1.
- **Handle Monitor** - Next the reducer prepares the monitors for correction. Again the procedure for this step is exactly the same as the one described in section 5.1.1.
- **Correct By Monitor** - Again this is the same as described in section 5.1.1, but when running the convert units algorithm to convert the workspace to wavelength the reduction step uses the elastic energy mode option instead of Indirect energy mode option.
- **Fold Data** - If the data was split into multiple frames during the loading step, the fold data step is performed. This is identical to the step described in section 5.1.1.
- **Convert Units To *d*-spacing** - This is not a reduction step, but just a call to the Mantid ConvertUnits algorithm to covert the workspace to units of *d*-spacing with the energy mode used being elastic.

- **Rebin** - If the a rebin string was supplied to the reducer, the workspace is rebinned according to the supplied parameters. Again this is not a reduction step but just a plain call to the Rebin algorithm. If no rebin string was supplied, the workspace is rebinned to the first spectrum using the reduction step of the same name. This step just calls the RebinToWorkspace algorithm with all arguments set to the same workspace.
- **Grouping** - Next the grouping reduction step is performed, which is the same as in section 5.1.1 but with the grouping type hard coded to simply show all spectra.
- **SaveItem** - Finally, the reduced workspace is past through the SaveItem reduction step where the output options past to the reducer are processed. This step is identical to the one described in section 5.1.1.

The OSIRIS instrument can double as a diffractometer with a much larger diffraction detector range compared to the other indirect instruments at ISIS. It therefore uses a specialised algorithm called OSIRISDiffractionReduction for use in diff only mode and does not use the reducer chain. OSIRISDiffractionReduction is based off of the ARIEL program [26].

This algorithm takes a list of samples and a list of vanadium runs and an old style calibration file (these are not the ones generated by the routine in section 5.1.2, but the type supplied to the AlignDetectors and DiffractionFocussing algorithms). Each of the sample/vanadium run pairs can cover a different d-range which the algorithm then merges using the MergeRuns algorithm. This algorithm proceeds as follows:

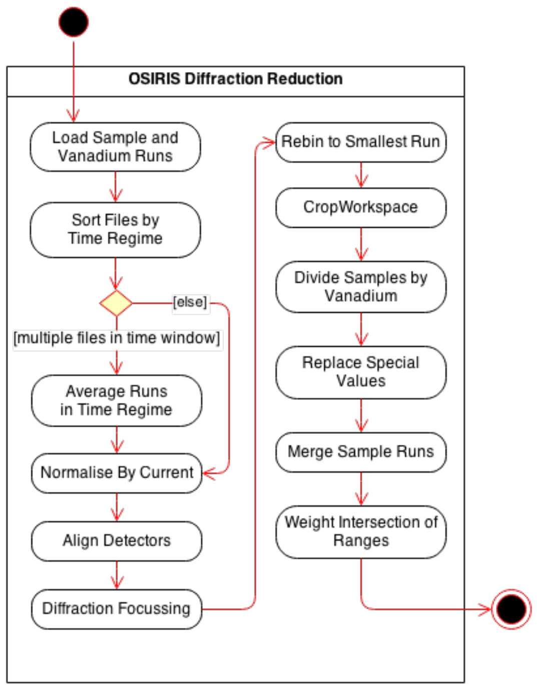

OSIRIS Diffraction Reduction
Load Sample and Vanadium Runs
Sort Files by Time Regime
[else]
[multiple files in time window]
Average Runs in Time Regime
Normalise By Current
Align Detectors
Diffraction Focussing
Rebin to Smallest Run
CropWorkspace
Divide Samples by Vanadium
Replace Special Values
Merge Sample Runs
Weight Intersection of Ranges

Figure 5.8: *Activity diagram showing the flow of the OSIRISDiffractionReduction algorithm in release 3.2.*

- The algorithm begins by loading all of the samples and vanadium runs into workspaces. This just uses the standard Load algorithm.
- The algorithm then maps each of the supplied samples to the corresponding d-range and averages any workspaces that are in the same d-range. The same procedure is repeated for the vanadium runs.
- At this point the algorithm has a processed list of all workspaces and their background for each d-range and can begin the reduction. The algorithm now normalises by the beam current, aligns detectors, and performs diffraction focussing using the calibration file provided. It then crops the workspace to match the corresponding d-range. These operations are performed for each sample and vanadium run.

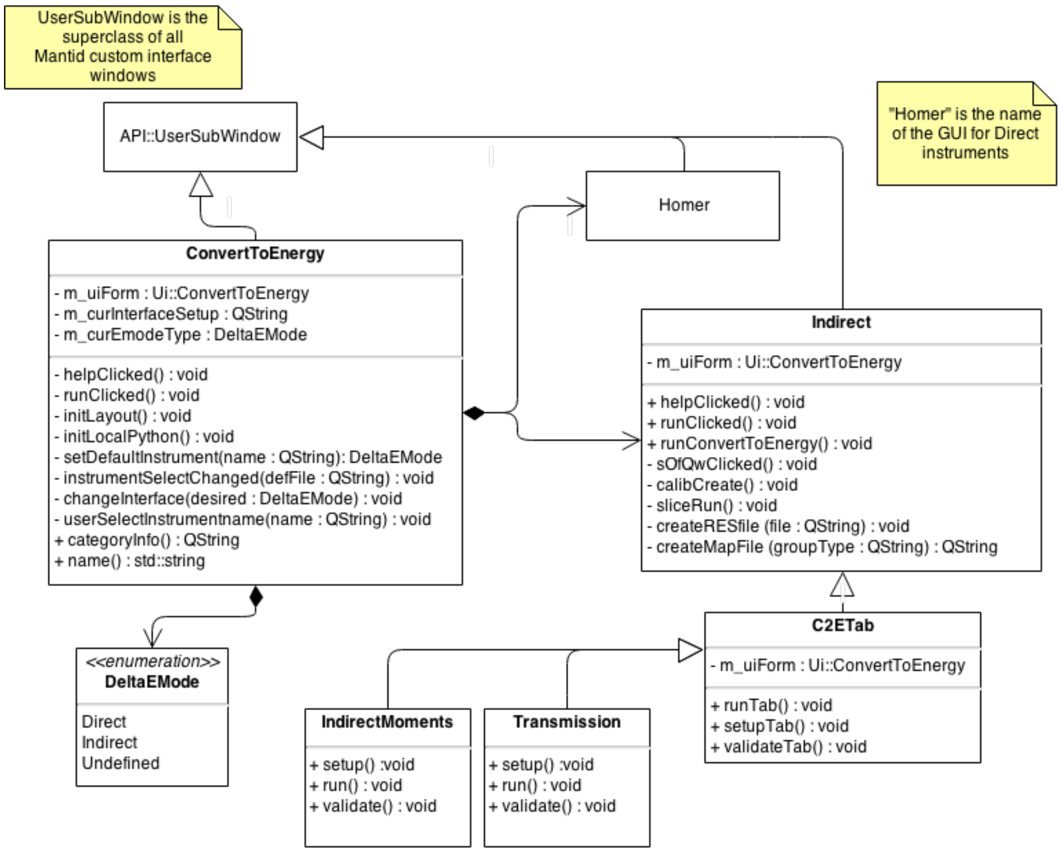
- The background for each sample is then processed by rebinning each sample/vanadium pair to match the smallest of the pair (using the RebinToWorkspace algorithm) and then divides the sample by the vanadium. Any NaN or infinity values after this operation are replaced with zeros using the ReplaceSpecialValues algorithm.
- Finally all of the samples are merged into a single workspace using the MergeRuns algorithm and the workspace is weighted at intersection where the ranges overlap by scaling by the number of overlapping contributions at a given point.

5.3 Data Reduction GUI

5.3.1 Energy Transfer GUI Overview

Figure 5.9 shows the class structure of the indirect convert to energy GUI. Note that for clarity only a selection of the more fundamental methods and attributes are shown as part of the class diagram. The Convert to Energy window (for both direct and indirect instruments) inherits from Mantid's generic UserSubWindow class which provides the basic functionality required to render a sub window within the application and also provides some useful helper methods such as the ability to execute a string as a python script.

The ConvertToEnergy class contains the code which is shared between both direct and indirect versions of the interface. It is responsible for loading the appropriate instrument definition files as the user selects an instrument, swapping between the indirect and direct interfaces depending on the geometry of the selected instrument, and handling the major user actions such as clicking the run energy transfer and help buttons.



UserSubWindow is the superclass of all Mantid custom interface windows
"Homer" is the name of the GUI for Direct instruments
API::UserSubWindow
Homer
ConvertToEnergy
- m_uiForm : Ui::ConvertToEnergy
- m_curInterfaceSetup : QString
- m_curEmodeType : DeltaEMode
- helpClicked() : void
- runClicked() : void
- initLayout() : void
- initLocalPython() : void
- setDefaultInstrument(name : QString): DeltaEMode
- instrumentSelectChanged(defFile : QString) : void
- changeInterface(desired : DeltaEMode) : void
- userSelectInstrumentname(name : QString) : void
+ categoryInfo() : QString
+ name() : std::string
Indirect
- m_uiForm : Ui::ConvertToEnergy
+ helpClicked() : void
+ runClicked() : void
+ runConvertToEnergy() : void
- sOfQwClicked() : void
- calibCreate() : void
- sliceRun() : void
- createRESfile (file : QString) : void
- createMapFile (groupType : QString) : QString
C2ETab
- m_uiForm : Ui::ConvertToEnergy
+ runTab() : void
+ setupTab() : void
+ validateTab() : void
<<enumeration>>
DeltaEMode
Direct
Indirect
Undefined
IndirectMoments
+ setup() :void
+ run() : void
+ validate() : void
Transmission
+ setup() :void
+ run() : void
+ validate() : void

Figure 5.9: *Class diagram showing the structure of the Indirect Convert To Energy GUI in release 3.2.*

ConvertToEnergy is composed with two classes called Homer and Indirect that handle the functionality of direct and indirect geometry instruments respectively. In previous releases of Mantid the Indirect class contained all of the GUI code for every tab on the Indirect Convert To Energy interface. This is no longer the case in more recent versions of Mantid which have moved towards a more modular class structure.

As of release 3.2, the Indirect class contains the GUI code for the Energy Transfer, Calibration, Diagnostics, $S(\mathbf{Q}, w)$, and Moments tabs. For each of these tabs there are corresponding methods (which are inconsistently named) to validate the user's input for that tab and run the appropriate routine. The specifics of each of these routines is outlined in the previous sections. In order the execute the required routine the GUI code for a particular tab builds a string which represents an executable python script complete with the options collected from the interface that can be executed using the runPythonCode() method inherited from UserSubWindow.

The actual design of the GUI is stored in a separate XML file that was created using Qt Designer and is used to produce a dynamically generated class for the interface at compile time which includes all the interface widgets for ConvertToEnergy. The ConvertToEnergy, Homer, and Indirect classes each have a reference to a copy of this interface object which they can query at runtime. The ConvertToEnergy class is responsible for adding and removing the appropriate tabs as the user switches between instrument geometries. The Indirect class uses this reference to query user input and update the interface appropriately in response to user actions.

In the past year there have been two additional tabs added to the Convert To Energy interface which do not follow the monolithic class structure currently used by the majority of tabs in the indirect energy conversion interface. Instead of adding the code for the additional tabs IndirectMoments and Transmission directly into the Indirect class, a new class was created called C2ETab which is composed with the Indirect class (see diagram 5.9). The specific implementations of the two new tabs are then handled by in a separate class derived from the abstract C2ETab.

This approach is similar to the one already place in the Indirect Data Analysis interface (see 6.5.1) and provides a number of benefits over the initial design. Structuring the code in this way allows for greater modularity because each subclass of C2ETab is only concerned with the functionality relating to a single tab on the interface.

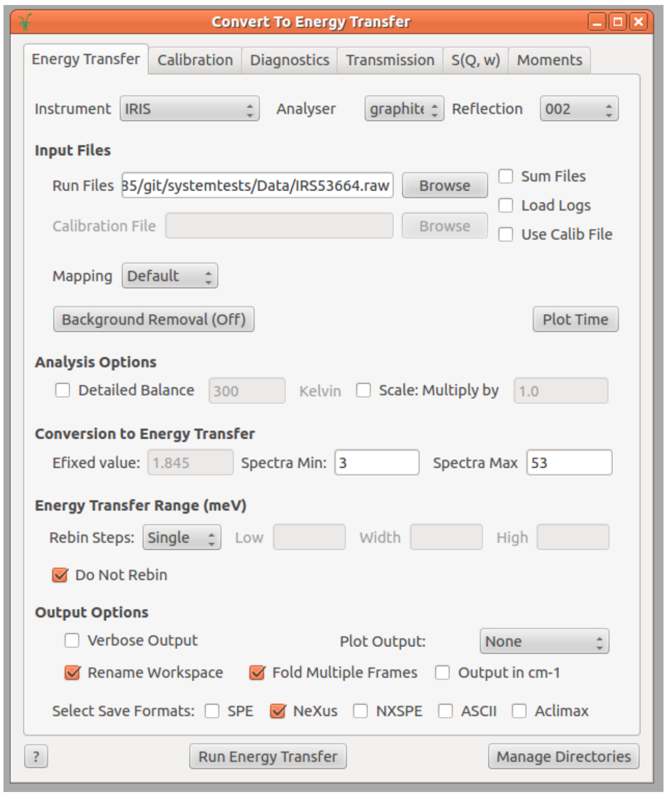
Furthermore, this design still allows code that is common between multiple tabs (such as code for plotting a histogram in a mini-plot) to be written once and shared across all classes through inheritance. Note that C2ETab also has a reference to the UI object so that it can interact with the widget objects defined on the interface, however this could easily be changed to use tab orientated interface objects (like the approach used by IndirectBayes, see 6.5.2 and further discussion in 8.3) instead of having unrestricted access to the whole interface.

5.3.2 Indirect Diffraction GUI Overview

The corresponding interface for diffraction reduction is called Indirect Diffraction. Unlike the other interfaces in the Indirect collection the structure of this interface is just a single class with a single UI file defining the layout. Each instrument has either the option of running in diff spec or diff only mode (TOSCA and VESUVIO only have diff spec mode) and, as described in the previous section, OSIRIS has a different reduction routine in the case of diff only mode. This is reflected in the interface when the user selects the appropriate option. The GUI changes the way the interface looks by checking the instrument parameter files for the currently selected instrument and reflection.

5.4 An Example Reduction

In this final subsection, we illustrate an example reduction of some data from the IRIS instrument using the Indirect Convert To Energy GUI in Mantid. The screen shot in figure 5.10 shows the Indirect Convert to Energy interface in release 3.2. The example shows the instrument selected as being IRIS and using the graphite analyser with reflection 002. One or more runs can be selected using the file browser in the input files section of the interface.



Convert To Energy Transfer

Energy Transfer | Calibration | Diagnostics | Transmission | S(Q, w) | Moments

Instrument: IRIS   Analyser: graphite   Reflection: 002

Input Files

Run Files: 35/git/systemtests/Data/IRS53664.raw   Browse   ☐ Sum Files   ☐ Load Logs   ☐ Use Calib File

Calibration File: Browse

Mapping: Default

Background Removal (Off)   Plot Time

Analysis Options

☐ Detailed Balance 300 Kelvin   ☐ Scale: Multiply by 1.0

Conversion to Energy Transfer

Efixed value: 1.845   Spectra Min: 3   Spectra Max: 53

Energy Transfer Range (meV)

Rebin Steps: Single   Low   Width   High

[✓ Do Not Rebin]

Output Options

☐ Verbose Output   Plot Output: None

[✓ Rename Workspace]   [✓ Fold Multiple Frames]   ☐ Output in cm-1

Select Save Formats: ☐ SPE   [✓ NeXus]   ☐ NXSPE   ☐ ASCII   ☐ Aclimax

?   Run Energy Transfer   Manage Directories

Figure 5.10: *Indirect Convert to Energy interface reducing some data from the IRIS instrument in release 3.2*

There are lots of additional options that can be configured depending on the user's requirements for the output of the reduction. This includes the rebinning, spectrum mapping, and detector ranges to be used, whether to sum the files and what formats to save the resulting workspace as (in the user's default save directory). All of these options are set to a sensible default for each instrument as it is selected, so in the simplest case all the user needs to do is select the instrument and the appropriate run file(s) and click run energy transfer.

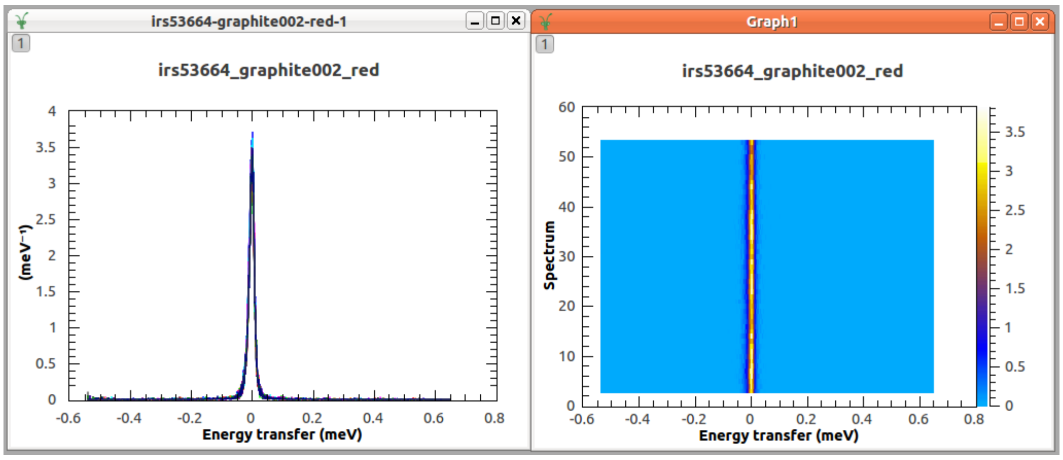
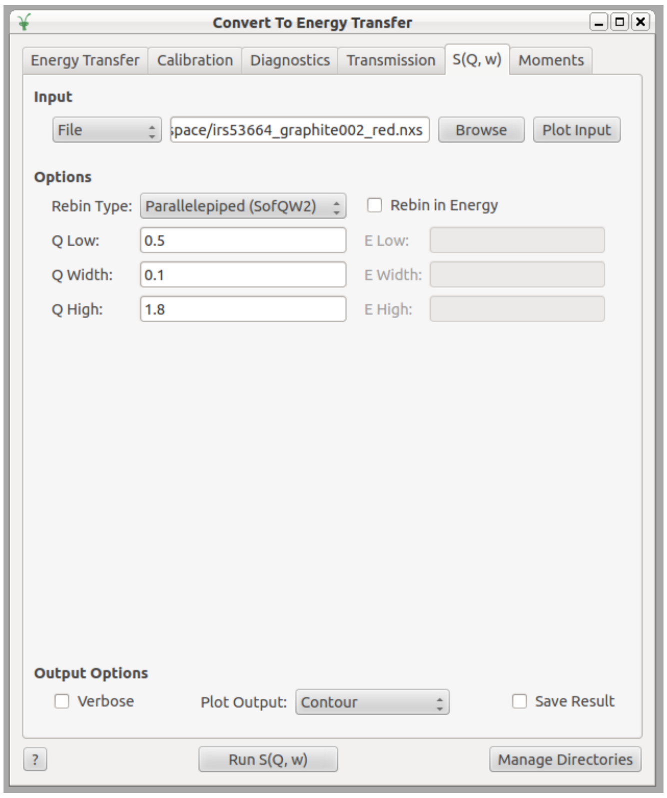


Figure 5.11: *Reduced IRIS data plotted as both a contour and spectrum plot*

Doing this with the settings shown in figure 5.10 produces a single workspace for run 553664 on the IRIS instrument with units of energy transfer. This workspace can then be used with the Indirect Data Analysis and Indirect Bayes interfaces. We can also run this through the SofQW algorithm to calculate the $S(\mathbf{Q}, \omega)$ and convert the workspace to units of momentum transfer using the SofQW tab on same interface.



Convert To Energy Transfer

Energy Transfer | Calibration | Diagnostics | Transmission | S(Q, w) | Moments

Input

File: space/irs53664_graphite002_red.nxs

Browse | Plot Input

Options

Rebin Type: Parallelepiped (SofQW2)

☐ Rebin in Energy

Q Low: 0.5

Q Width: 0.1

Q High: 1.8

E Low:

E Width:

E High:

Output Options

☐ Verbose

Plot Output: Contour

☐ Save Result

? | Run S(Q, w) | Manage Directories

Figure 5.12: *Indirect Convert to Energy interface converting some reduced data output from the previous screen shot using the SofQW algorithm in release 3.2*

This interface allows us to specify the binning we use in Q and in energy and also the type of rebinning to be used in the conversion (see section 5.1.5). Plotting the resulting _sqw workspace as a contour plot shows energy transfer vs. Q.

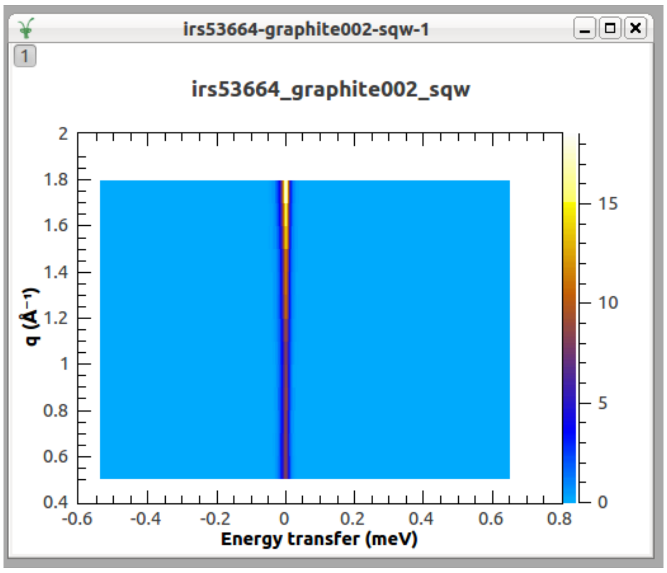


Figure 5.13: *Contour plot of a _sqw workspace in release 3.2*

6 Data Analysis with Mantid

After data has been reduced using the convert to energy or diffraction reduction routines described in the previous section the reduced data needs to be analysed. In the indirect section of Mantid there are two major collections of data analysis scripts that are currently in place. The first and most important are those included in the Indirect Data Analysis interface which provide a variety of different fit functions and transforms that can be performed on reduced data. The second collection of routines are accessible via the Indirect Bayes interface. This provides a collection of routines that can be used to infer fitting parameters using Bayesian model selection techniques which are complementary to the routines in Indirect Data Analysis.

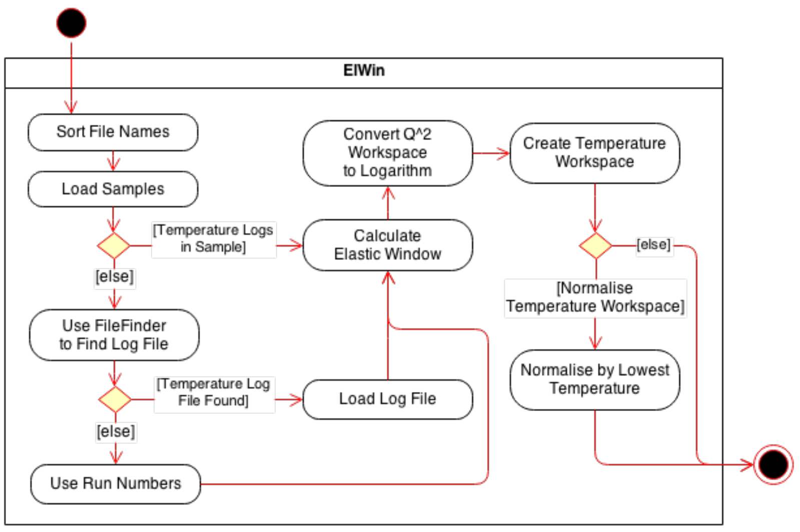
The majority of the code for data analysis scripts can be found in the file IndirectDataAnalysis.py with the Inelastic script folder of the Mantid installation, with the exception of Calculate Corrections which is in a file called IndirectAbsCor.py. For the Bayesian fitting programs, the code is stored in the file IndirectBayes.py, with the exception of JumpFit which is stored in IndirectJumpFit.py both of which are in the Inelastic scripts folder.

6.1 Basic Analysis

6.1.1 ElWin

The ElWin (elastic window) program is used to perform elastic window scans by integrating over the energy range of a reduced file and is based on the routine of the same name that was originally provided as part of the MODES package [3]. As with almost all routines in the Indirect Data Analysis interface ElWin is implemented as a collection of free standing functions in IndirectDataAnalysis.py. Internally this routine uses Mantid's ElasticWindow algorithm. This is a small C++ side algorithm that integrates over the input workspace. The script provided in IndirectInelastic.py pre and post processes the data.

The elastic window algorithm converts the spectrum axis of the input workspace to two output workspaces with units of Q and Q^2. Both of these workspaces are then transposed so they are easier to visualise. Optionally, this routine can also remove a background using a second range. The Q^2 output for ElWin is used as the input to the MSDFit routine described in the following section.



ElWin
Sort File Names
Load Samples
[Temperature Logs in Sample]
[else]
Use FileFinder to Find Log File
[Temperature Log File Found]
[else]
Use Run Numbers
Load Log File
Calculate Elastic Window
Convert Q^2 Workspace to Logarithm
Create Temperature Workspace
[else]
[Normalise Temperature Workspace]
Normalise by Lowest Temperature

Figure 6.1: *Activity diagram for the ElWin routine.*

The ElWin routine proceeds by first sorting the list of workspace names supplied as input. As there is an unofficial naming convention in the indirect inelastic section of Mantid where the names include the run number this should produce a list with the smallest run number first.

The program then iterates over each workspace and attempts to extract temperature information from the sample logs. The logs will only be present if the load logs option was checked during the energy transfer reduction. Otherwise, the routine searches the users managed directories for a log file matching the log number and name of the instrument for the current run and loads it using the LoadLog algorithm. If no logs are found then the run numbers for each file are used instead of temperature.

The ElasticWindow algorithm is then executed on the sample. Optionally two ranges can be supplied which will remove a flat background from the workspace first. The algorithm produces the two workspaces in units of Q and Q^2. The Q^2 workspace is then converted to be the Logarithm of the data.

Finally, a third workspace is created from the two output from the ElasticWindow algorithm with units of Kelvin on the x axis vs. momentum transfer on the y axis. This uses the value for temperature loaded from the sample logs at the start of the routine. If the option is selected the temperature workspace is normalised by the lowest temperature in the workspace.

6.1.2 MSDFit

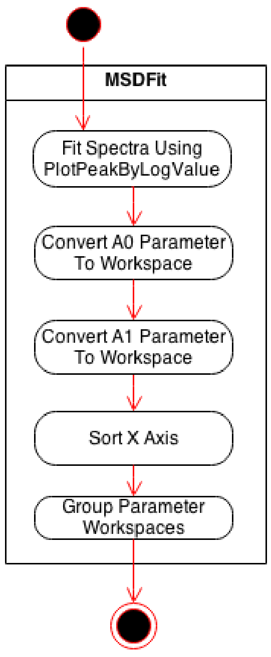

MSDFit
Fit Spectra Using PlotPeakByLogValue
Convert A0 Parameter To Workspace
Convert A1 Parameter To Workspace
Sort X Axis
Group Parameter Workspaces

Figure 6.2: *Activity diagram for the MSDFit routine.*

The MSD fit routine is used to calculate the mean squared displacement from the Q^2 workspace output from ElWin. In essence this routine performs a linear fit to every spectrum in the workspace using the algorithm PlotPeakByLogValue. However the routine also performs some additional operations to transform the fit parameters from a table workspace to a matrix workspace using the ConvertTableToMatrixWorkspace algorithm. The final parameter workspaces are then grouped.

This routine produces a group workspace containing the linear fits to each spectrum and a workspace for each of the parameters of the fit to each spectrum transposed into a matrix workspace.

6.1.3 Fury

The Fury program is used to perform a fast Fourier transform on a reduced workspace in order to calculate the intermediate scattering function $I(Q, t)$. The routine can handle converting multiple workspaces at once, but is currently only set-up on the interface to convert one reduced sample at a time.

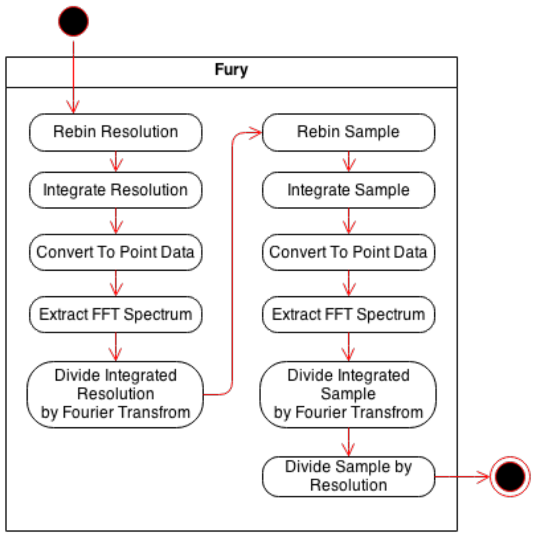
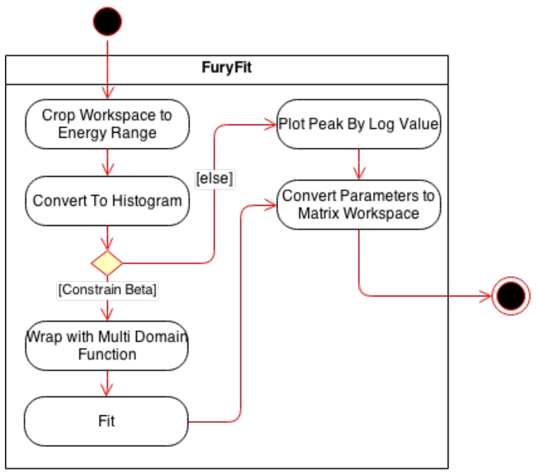


Figure 6.3: *Activity diagram for the Fury routine.*

The routine first Fourier transforms the resolution workspace then transforms each of the samples using the exact same method, then divides each of them by the resolution. In order to convert to the intermediate scattering function all of the workspaces must have the same x range and the same regular binning in order for the conversion to work. Therefore the rebin parameters provided by the user must satisfy these conditions. The workspace is then integrated to a separate workspace. The original workspace is then converted to point data in preparation for running the fast Fourier transform algorithm.

The algorithm used is called ExtractFTTSpectrum which runs Mantid's FFT algorithm on each spectrum in the workspace to output the modulus positive frequencies of the transform in the corresponding spectrum. The transformed sample workspaces are then divided by the transformed resolution workspace. The upper half of the transformed workspace is then cropped to get rid of excess noise from the transform and then optionally saved to a nexus file.

6.1.4 FuryFit



FuryFit
Crop Workspace to Energy Range
Convert To Histogram
[else]
[Constrain Beta]
Wrap with Multi Domain Function
Fit
Plot Peak By Log Value
Convert Parameters to Matrix Workspace

Figure 6.4: *Activity diagram for the Fury Fit routine.*

FuryFit provides a fit routine for the output of Fury offering a variety of decaying exponential functions. Similar to MSDFit, the routine uses the Fit and PlotPeakByLogValue algorithms to sequentially fit each spectrum with the parameters.

The FuryFit code actually consists of two routines which are very similar and which could be merged in future release of Mantid called FuryFitSeq and FuryFitMult which perform a sequential fit and a multi domain fit respectively. Both routines works by executing a function string which is dynamically created in the GUI code for FuryFit, but FuryFitMult first wraps the input function with a MultiDomain composite function. FuryFitMult is only executed when the user has chosen to constrain the β parameter for a stretched exponential fit across all Q.

Both of these routines convert the table of fitting parameters created from Fit (or PlotPeakByLogValue) into a matrix workspace using the same code as used by MSDFit.

6.1.5 ConvFit

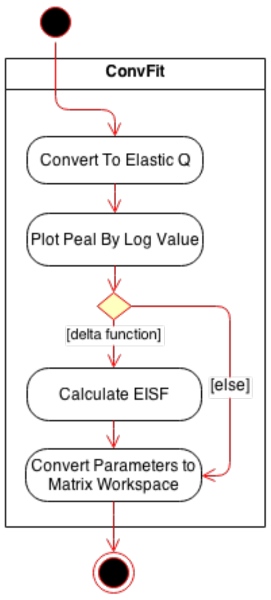


Figure 6.5: *Activity diagram for the Conv Fit routine.*

The ConvFit routine is used to perform a convolution fit to a workspace. Again, this routine follows a similar procedure to both the MSDFit and FuryFit routines. A fit function with the options the user has selected are gathered from the user interface and a string representing the function to be fitted is passed to the PlotPeakByLogValue algorithm to perform a sequential fit.

The functions fitted are a δ-function, one or two Lorentzians or a combination of both which are then convolved with the user supplied resolution workspace. The type of background can also be selected from a choice of Fixed Flat (flat background without the intercept tied to the user supplied value), Fit Flat (flat background with the intercept unconstrained), and Fit Linear (unconstrained linear background). Optionally a temperature correction can be applied to the Lorentzians. The temperature correction is defined as:

$$f(x) = (\frac{x \cdot 11.606}{t})/[1 - exp(-\frac{x \cdot 11.606}{t})] \tag{6.1}$$

Where t is the user supplied value for the temperature in kelvin, x is the value in energy transfer at a given point in the workspace and 11.606 is the conversion factor from units of meV to K. Each Lorentzian is multiplied by this factor then convolved with the resolution.

The final composite function is represented as a string that the Mantid fit algorithm can interpret. This function string is generated within the GUI code for ConvFit and then passed to the underlying python routine.

As with FuryFit, ConvFit will convert the input workspace to elastic Q first before performing the fit and converts the output table of fit parameters to a matrix workspace. If a δ-function was used the elastic incoherent structure factor (EISF) is calculated from the resulting table of parameters.

6.1.6 Calculate Corrections

Calculate Corrections is used to create a group workspace of absorption corrections from known parameters of the sample and can used in an experiment which can be applied to a reduced workspace using the Apply Corrections routine (see section 6.1.7). This routine supports both flat plate and cylindrical geometries.

The code for calculating corrections for a cylindrical geometry is has been ported into Mantid using the F2Py package from it's previous incarnation as the Acorn routine in the MODES package. F2Py allows a python script run the Fortran code without it being converted to python code. Unfortunately the downside to this is that F2Py cannot be packaged into the Mantid installer for any platform outside of Windows. The code for flat plate corrections was ported use Mantid's python API as part of release 3.1.

In depth discussion on the exact methods of calculating the absorption factors for flat plate and cylindrical geometries can be found in Carlile [23] and the ATLAS manual [24] respectively.

The python code underlying the interface of calculate corrections and responsible for liaising between the Fortran and C++ GUI code is (unlike the other routines described in this section) stored in a separate file called IndirectAbsCor.py. This is to keep F2Py import call separate from the rest of the data analysis code so that the other analysis routines can be used on platforms other than Windows.

Regardless of the geometry used the Python/Fortran code calculates four arrays for each spectrum in the sample workspace each of which are collected into four resulting output workspaces with the suffixes *ass*, *assc*, *acsc*, and *acc* which loosely correspond to the formalism described in Paalman and Pings [25]. These stand for:

- **Ass** - Attenuation factor for scattering in the sample and attenuation in the sample.
- **Assc** - Attenuation factor for scattering in the sample and attenuation in the sample plus container.
- **Acsc** - Attenuation factor for scattering in the container and attenuation in the sample plus container.
- **Acc** - Attenuation factor for scattering in the container and attenuation in the container.

If a workspace representing the container is not supplied by the user then all workspaces except the *_ass workspace will just contain zeros.

6.1.7 Apply Corrections

Apply Corrections provides a complementary routine to the calculate corrections routine described in the preceding section. Apply corrections has two modes of applying corrections to a sample. The first and simplest method is to just take a run of the container for the sample and subtract it from the run of sample plus container. The second method uses the corrections workspace generated by the calculate corrections routine and applies them to the sample & can workspaces (if a can is supplied). A detailed activity diagram of the two modes of execution are shown in diagram 6.6

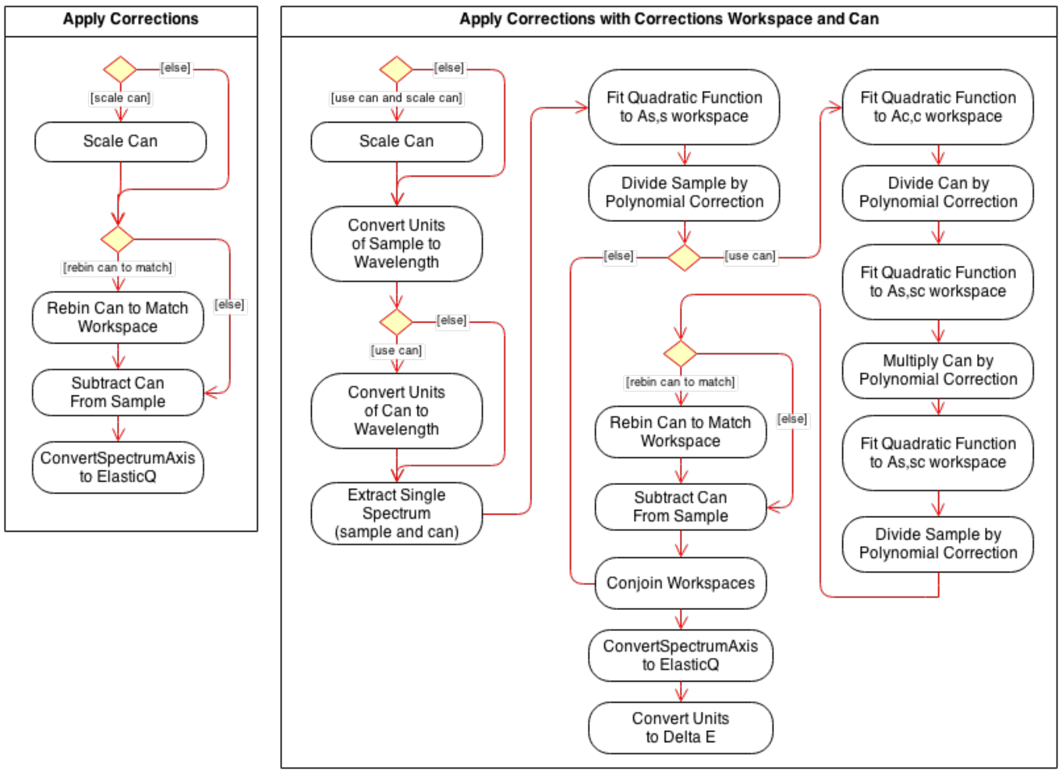

Apply Corrections
[else]
[scale can]
Scale Can
[rebin can to match]
Rebin Can to Match Workspace
[else]
Subtract Can From Sample
ConvertSpectrumAxis to ElasticQ
Apply Corrections with Corrections Workspace and Can
[use can and scale can]
Convert Units of Sample to Wavelength
[use can]
Convert Units of Can to Wavelength
Extract Single Spectrum (sample and can)
Fit Quadratic Function to As,s workspace
Divide Sample by Polynomial Correction
[use can]
Conjoin Workspaces
Convert Units to Delta E
Fit Quadratic Function to Ac,c workspace
Divide Can by Polynomial Correction
Fit Quadratic Function to As,sc workspace
Multiply Can by Polynomial Correction
Divide Sample by Polynomial Correction

Figure 6.6: *Activity diagram showing the flow of the apply corrections routine in release 3.2.*

In the former mode of operation (left activity diagram) the scale is first optionally scaled by an arbitrary factor. If the energy ranges of the sample and container do not match the can is rebinned if the user has specified the rebin can option, otherwise an error is thrown. If rebinning the can to match the sample is requested the routine uses the RebinToWorkspace algorithm. After the sample has been corrected, it will use the ConvertSpectrumAxis algorithm to convert the resulting corrected sample to units elastic Q.

In the latter mode of operation, the routine first converts both the sample and container to units of wavelength. Then for each spectrum in the sample and can (if supplied) performs a quadratic fit (using the standard Mantid Fit algorithm) to the corresponding spectrum in each of the members of the group workspace of absorption corrections. The coefficients from this fit are then supplied as parameters to the PolynomialCorrection algorithm. The resulting single spectrum workspace for the can is subtracted from the corresponding single spectrum sample workspace and the result is stitched back together into a single workspace using the ConjoinWorkspaces algorithm. The resulting workspace is output both in units of energy transfer and elastic Q.

6.2 Advanced Analysis

The Indirect Bayes interface provides a collection of routines for performing model and parameter selection using Bayesian methods. These routines are, like those in proceeding sections, based on previous implementations in the MODES package. The ResNorm, Quasi, and Stretch routines described in the following sections are all imported into Mantid using the F2Py package directly from the original Fortran code.

In future releases of Mantid it is hoped that these routines will be converted into proper C++ or Python algorithms as is the case with the rest of Mantid. The routines defined here are included in Mantid in the file IndirectBayes.py and the accompanying Fortran modules are included in the same folder. The majority of the code in this file does not currently use standard Mantid algorithms, largely because the the Fortran code obviously doesn't work with workspaces and so we are reduced to working with numpy arrays and manually building workspaces. Worse than this, in the Quasi fitting routine the fit parameters get written to file, which is then immediately read back into the program which is both unnecessary and hugely inefficient.

6.2.1 ResNorm

ResNorm is used to create a group normalisation workspace which can be input to the Quasi routine (described in the next section). It normalises the input data using the instrument resolution by fitting. A group normalisation may also be done by grouping Q values where the resolution is stretched using a stretch factor. The routine produces two group workspaces with the suffixes *_ResNorm and *_ResNorm_Fit respectively. The ResNorm group workspace contains two workspaces with the suffixes intensity and stretch. The intensity workspace contains the intensity normalisation factor and stretch contains a stretch factor for the width of the supplied resolution file. The ResNorm_Fit group workspace contains the fits to each spectrum in the workspace.

6.2.2 Quasi

The Quasi routine provides a model selection algorithm using Bayesian methods to choose the number of lorentzian components present in the input data [16] and was originally known as Quasi Lines in the old MODES package [3]. This routine takes a sample to fit to and a resolution file created in the convert to energy interface and optionally a resolution normalisation file created in ResNorm (see the previous section). This program can run either by fitting a δ-function plus a sum of Lorentzians or a single stretched exponential.

There are three Fortran modules used depending on the input. The QLres module fits a sum of Lorentzians. If the resolution file contains more than one spectrum the QLdata module is used and any ResNorm file supplied is ignored. Finally if the choice of fitting a stretched exponential is used the QLse module is used. Each of these modules share very similar code which is duplicated and modified slightly to suit the specifics of the implementation.

Inside the fitting routine it first refines the parameters for the δ-function, linear background, energy range offset and thereby estimates the probability of the absence of quasielastic components in data. It then refines the parameters for one, two and three Lorentzian components successively [16].

Internally the program loads both the ResNorm and width files into 1D arrays. It then extracts from the sample and resolution workspaces the x, y, and e arrays for each spectrum in the workspace and passes this data to the Fortran routine with any accompanying fit parameters used. The corresponding fits to the data are returned in the form of x, y, and e arrays for the spectrum and its log probability for each Lorentzian (from 0 to 3). Additionally, the program also reads the program parameters from the generated file (saved in the user's default save directory) and creates a workspace of fit parameters similar to the one output by ConvFit. Likewise, it will also calculate the EISF from the fit parameters.

6.2.3 Stretch

Stretch is a variation on the stretched exponential option provided by the Quasi program described in the previous section and was originally called Quest in the old MODES package [3]. The Fortran module used is called Que. The operation of the two programs are essentially the same, except that a grid of β and σ parameters are fitted to each spectrum instead.

This fit routine produces one group workspace containing the fit to the data for each spectrum and two additional workspaces containing the parameters β and σ parameters of the fit.

6.2.4 JumpFit

JumpFit provides very simple routine for fitting the width parameters for each spectrum fitted using either the Quasi or ConvFit routines. It offers a collection of diffusion fit functions (defined as Mantid fit functions). The desired component width and function type is selected by the user and a fit function string is built and passed to Mantid's Fit function. The fit functions used in JumpFit are currently hard coded to use the default values shown in table 6.1.

Function	Function Definition	Default Parameter Values
Chudley-Elliot	$\Gamma(Q) = (1 - sin(Ql)/Ql)/\tau$	$\tau = 1.0/QMax, L = 1.5$
Hall-Ross	$\Gamma(Q) = (1 - exp(-lQ^2))/\tau$	$\tau = 1.0/QMax, L = 1.5$
Fick	$\Gamma(Q) = DQ^2$ where $D =< l^2 > /6\tau$	$D = \Delta y/\Delta x^2$
Teixeira Water	$\Gamma(Q) = DQ^2/(1 + DQ^2\tau)$ where $D =< l^2 > /6\tau$	$\tau = 1.0/QMax, L = 1.5$

Table 6.1: *JumpFit function names, definitions, and default parameter values.*

6.3 TOSCA Data Analysis

All of the data analysis routines mentioned in the previous section are orientated towards the analysis of quasi-elastic neutron scattering (QENS) data. The TOSCA spectrometer is used to perform inelastic neutron scattering and does not make use of the QENS routines outlined in previous sections. Their data analysis procedures are far simpler. The general process for analysing TOSCA data is to perform an energy transfer reduction as described in section 5.1.1 and then directly compare the resulting workspace with one produced from theory using the visualisation tools available through the Mantid framework.

Currently the simulated data used in comparison is generated using tools such as Gaussian 03 and a-CLIMAX and then loaded into Mantid. In the future we aim to add more support in Mantid for comparing experimental results with theory (see section 7). As an example, the plot in figure 6.7 shows a comparison of the experimental result from a sample of toluene with the spectrum generated from theory by simulation programs.

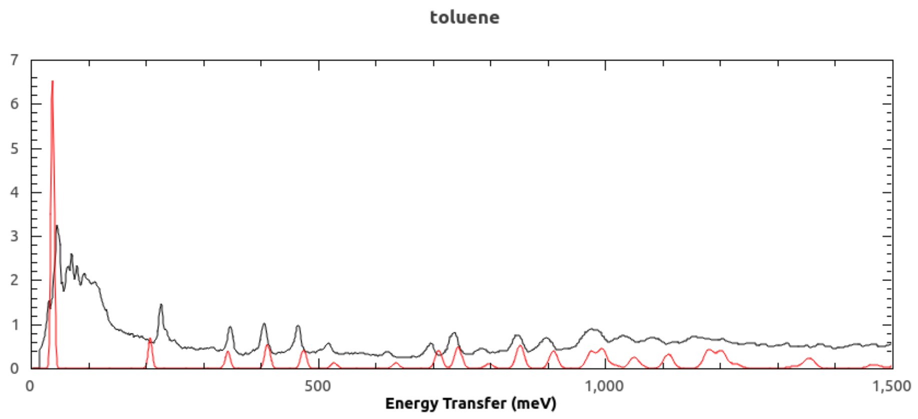


Figure 6.7: *Plot of a toluene sample (run 13930) cooled to 10K in black with theoretical simulation of the spectrum in red.*

6.4 VESUVIO Data Analysis

The VESUVIO spectrometer is currently not officially supported by the Mantid framework. Analysis with the spectrometer is primarily centred on the determination of atomic momentum distributions in condensed matter systems [14]. Development work has already been undertaken to add support for the spectrometer to Mantid. A collection of data analysis routines are available, but are currently under heavy testing and may be unstable. Information on using these routines, how to obtain them, and the progress of development are outlined in Ref. [27].

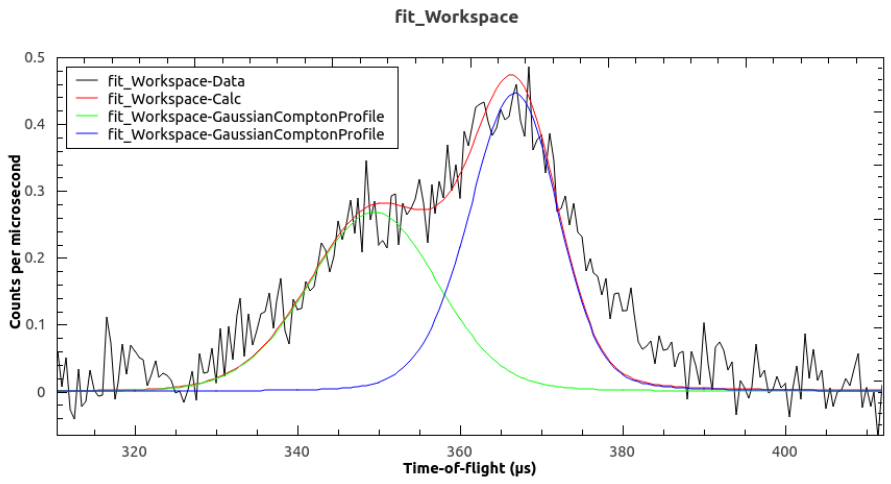


Figure 6.8: *Plot of some VESUVIO data for a sample of ZrH_2 using the routines described in Ref. [27] and based on the methods described in Ref. [14]. The black line shows the original data, the red line shows the fit to the data is in red, and the individual contributing peaks are in green (aluminium) and blue (zirconium)*

6.5 Data Analysis GUI

6.5.1 Indirect Data Analysis GUI Overview

The Indirect Data Analysis user interface follows a similar design structure to the Convert To Energy interface but is more modular. The interface is composed of a single parent class for the window which sets up and manages all tabs on the interface and is simply called IndirectDataAnalysis. A collection of classes for each tab on the interface which inherit from a common base class, similar to the C2ETab class in the convert to energy GUI, are composed with the window class to handle the specific implementation details of each the corresponding tabs.

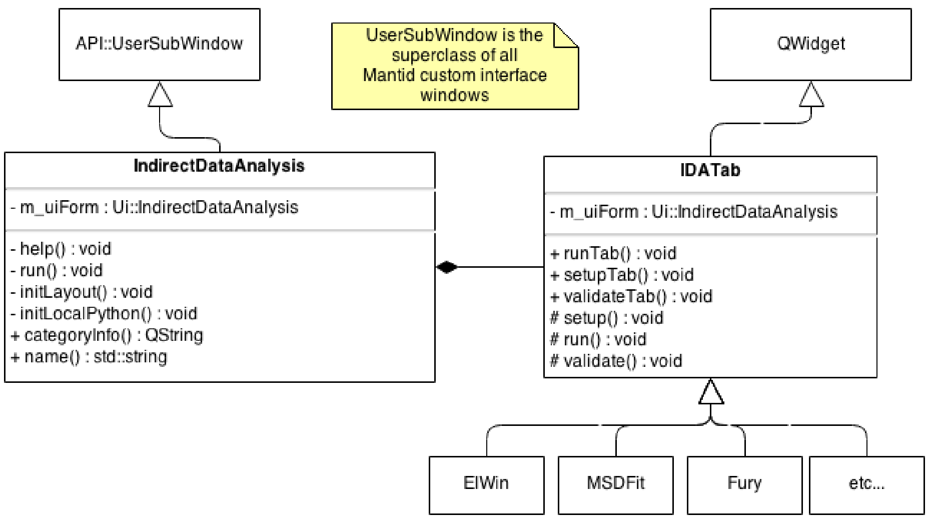


Figure 6.9: *Class diagram for the Indirect Data Analysis GUI.*

As with the convert to energy interface, the layout of the entire GUI is defined in a single XML file that each of the tab objects in the GUI system have a reference to. As of release 3.2 the indirect data analysis interface consists of seven tabs; namely: ElWin, MSDFit, Fury, FuryFit, ConvFit, Calculate Corrections, and Apply Corrections. Each of these have a single class which inherits from a base class called IDATab where the common logic shared between tabs for setting up the tabs, drawing to the mini plot, and loading a nexus file, among other things, is stored.

This approach is much better compared to the one used in the convert to energy interface. The class structure used in the IDA GUI avoids the monolithic class design in convert to energy and separates the implementation logic specific to each routine from the rest of the system while still allowing common code to be shared. However, this structure could still be improved further to remove the reliance on a single GUI file for the defining the interface.

6.5.2 Indirect Bayes GUI Overview

The GUI structure for the Bayes interface again follows a similar structure to the convert to energy and indirect data analysis interfaces, but takes the abstraction a step further. Like the others, this interface has a single parent class called IndirectBayes which defines the window and a collection of concrete classes paired with each of the tabs which inherit from an abstract base class called IndirectBayesTab which implements the common functionality for each tab.

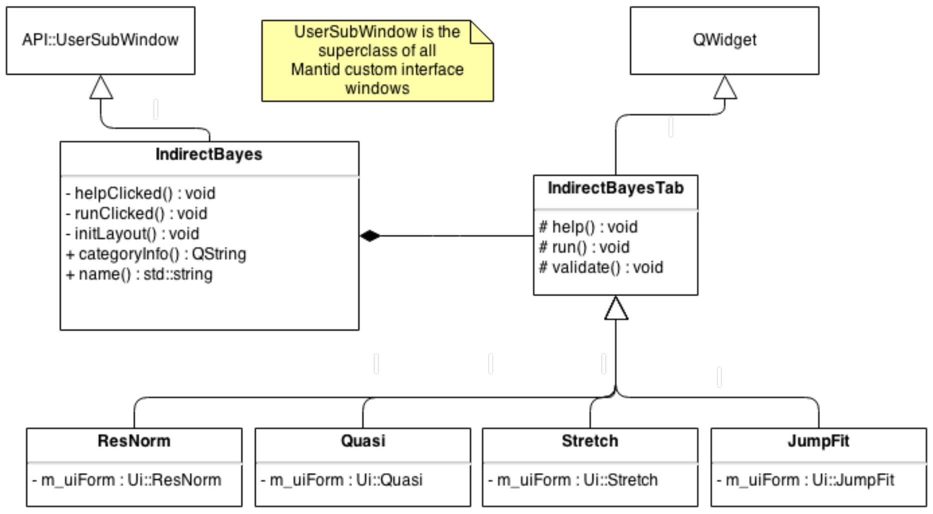
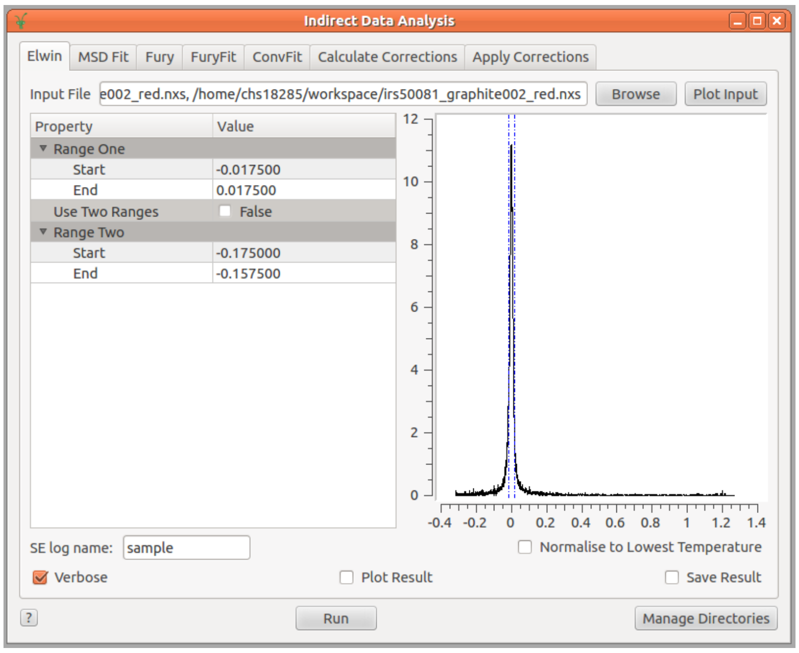


Figure 6.10: *Class diagram for the Indirect Bayes GUI.*

The key difference between the two is that instead of the interface being defined in a single UI file, the main window and each of the tabs are defined in separate UI files. This is a subtle difference, but one which provides an enormous benefit to the development process. Splitting the UI into multiple files makes the interface more maintainable as a single tab may be worked on individually without affecting the others. If, for example, a tab needs to be removed or moved to a different part of the system it can be easily moved by removing its inclusion in the main window (defined by IndirectBayes).

6.6 An Example of Basic Data Analysis

Once a file has been reduced using the convert to energy interface as described in section 5.1.1 it can be used as input to the various data analysis routines. This section shows a couple of examples of using the Indirect Data Analysis interface with a reduced workspace. Figure 6.11 shows an example of running ElWin to examine the elastic window of an IRIS reduced workspace created from the convert to energy reduction routine. This interface includes a miniplot which shows the first spectrum of the workspace to help pick sensible values for the ranges.



Input File e002_red.nxs, /home/chs18285/workspace/irs50081_graphite002_red.nxs

Property	Value
Range One	
Start	-0.017500
End	0.017500
Use Two Ranges	False
Range Two	
Start	-0.175000
End	-0.157500

SE log name: sample

Normalise to Lowest Temperature

Verbose [✓]

Plot Result

Save Result

Figure 6.11: *Using a reduced workspace with ElWin in release 3.2.*

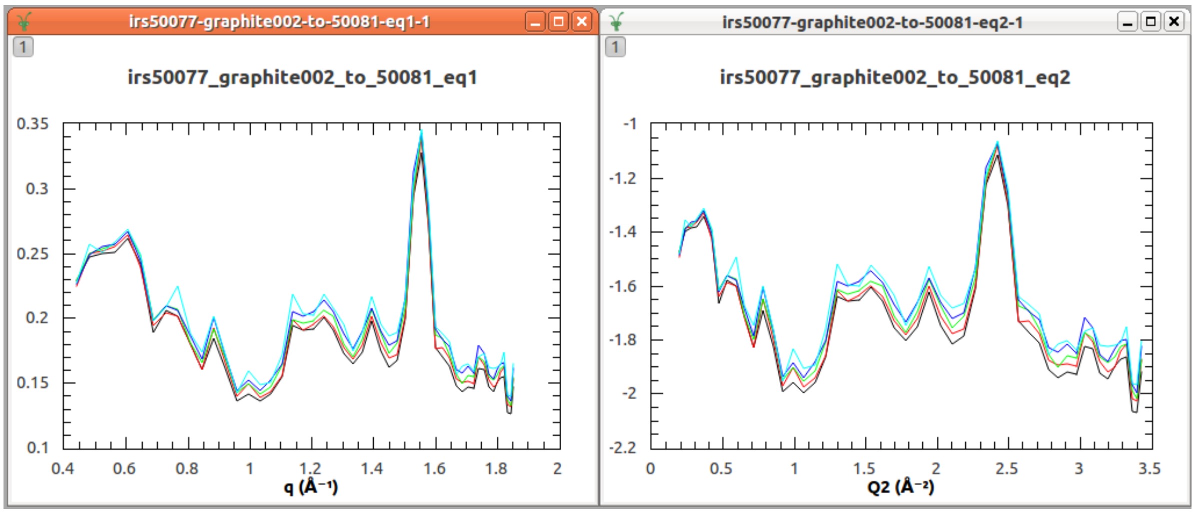
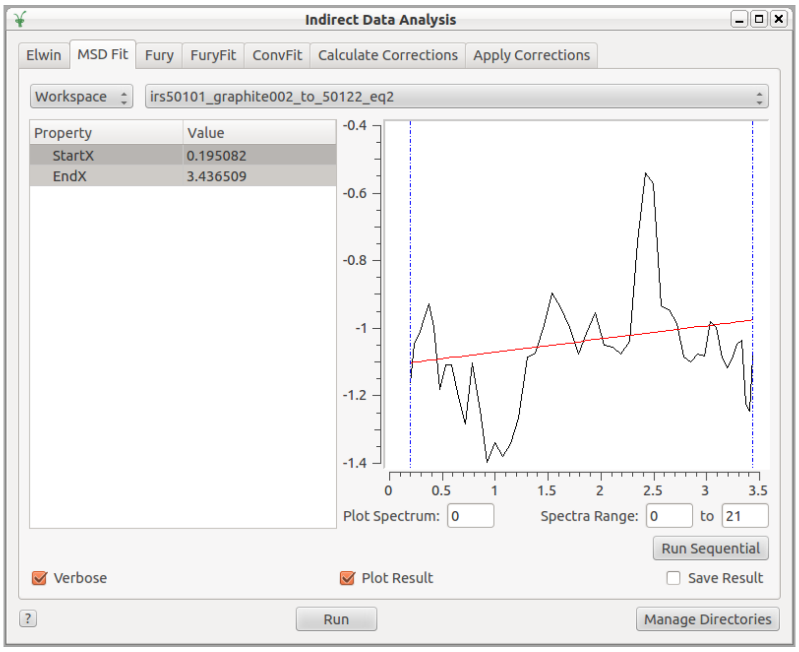


Figure 6.12: *Plots of the output workspaces from ElWin.*

The _eq2 file created by the routine can be passed to the MSDFit routine to perform a linear fit to the data over all spectra. Again, the miniplot can be used to select the range used with the data. Clicking the run button will fit to just the first spectrum, allowing the user to "get a feel" for the range to be used with the data. Running a sequential fit will use the same range to perform a sequential fit across all spectra in the workspace.



Workspace: irs50101_graphite002_to_50122_eq2

Property	Value
StartX	0.195082
EndX	3.436509

Plot Spectrum: 0 Spectra Range: 0 to 21

[✓] Verbose [✓] Plot Result [ ] Save Result

Figure 6.13: *Using an _eq2 workspace output from ElWin as input to MSDFit.*

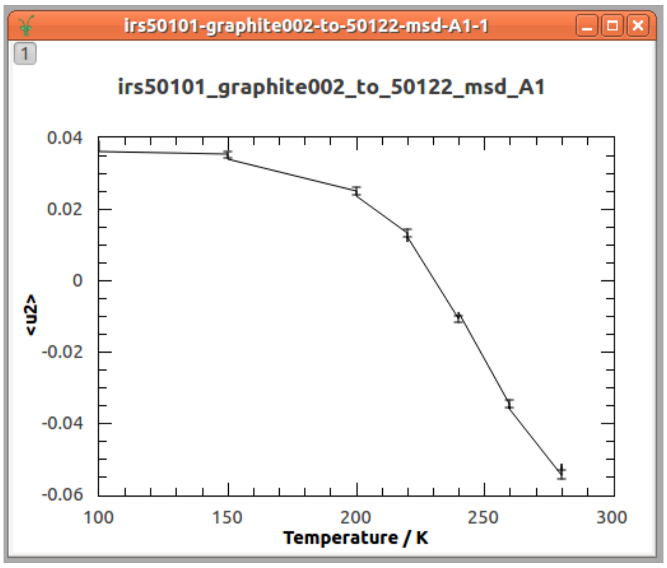
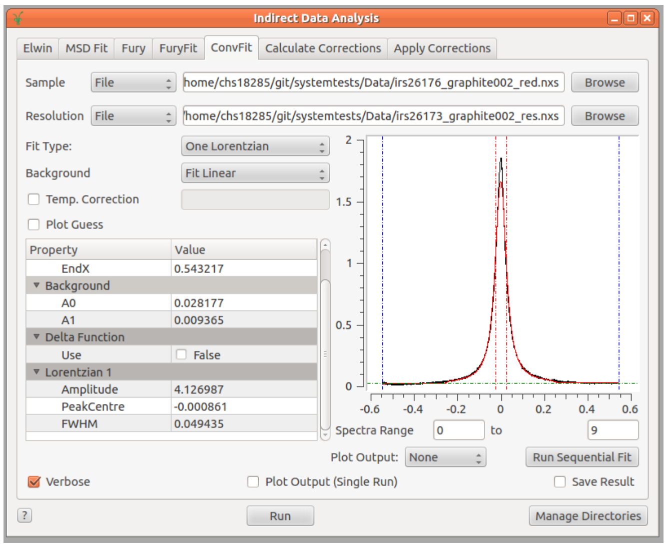


Figure 6.14: *Resulting parameter workspace from MSDFit.*

The ConvFit interface (section 6.1.5) can be used to separate the instrument resolution from the $S(\mathbf{Q}, \omega)$ function by fitting a convolution of the resolution and the appropriate model to the data. A screen shot of the ConvFit interface is shown in figure 6.15. This interface has lots of configurable options for the model that is to be convolved with the resolution, but works on similar principle to the MSDFit interface.

The run button at the bottom of the screen is used to perform a single fit to the data which also updates the parameters for the model in the properties browser. This is useful because it allows the user to choose parameters close to the appropriate value, then tighten them up before fitting all spectra. Once the user is happy with the fit parameters for a single spectrum (shown in the mini plot) the sequential fit button can be used to fit all of the spectra.



Sample: File — home/chs18285/git/systemtests/Data/irs26176_graphite002_red.nxs

Resolution: File — home/chs18285/git/systemtests/Data/irs26173_graphite002_res.nxs

Fit Type: One Lorentzian

Background: Fit Linear

☐ Temp. Correction

☐ Plot Guess

Property	Value
EndX	0.543217
▾ Background	
A0	0.028177
A1	0.009365
▾ Delta Function	
Use	☐ False
▾ Lorentzian 1	
Amplitude	4.126987
PeakCentre	-0.000861
FWHM	0.049435

Spectra Range 0 to 9

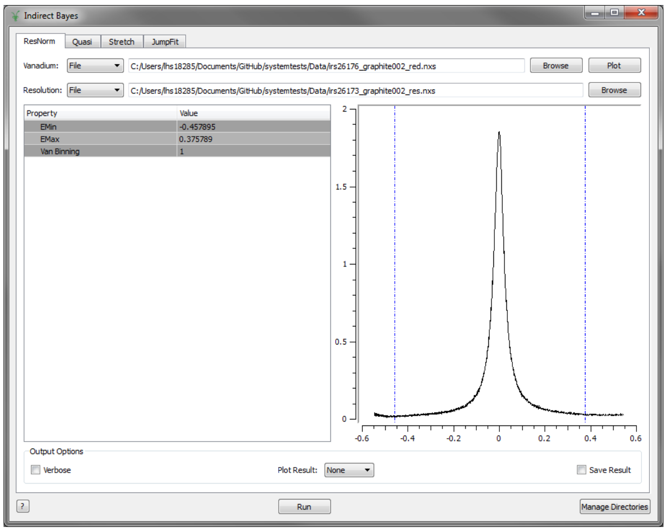
Plot Output: None

✓ Verbose ☐ Plot Output (Single Run) ☐ Save Result

Figure 6.15: *Screen shot of using the ConvFit interface to fit a single Lorentzian to the data.*

6.7 An Example of Advanced Data Analysis

The Bayesian fitting routines also operate on the reduced data created from Indirect Convert to Energy interface. Using an example reduced workspace from the IRIS instrument we can run the ResNorm program to generate a normalisation group workspace as shown in screen shot 6.16. As with other interfaces, the mini plot shows the first spectrum in the workspace and can be used to select the range used for binning. The resolution workspace is one that has been generated from the Calibration tab on the Convert to Energy interface (see section 5.1.2).



Vanadium: File — C:/Users/lhs18285/Documents/GitHub/systemtests/Data/irs26176_graphite002_red.nxs

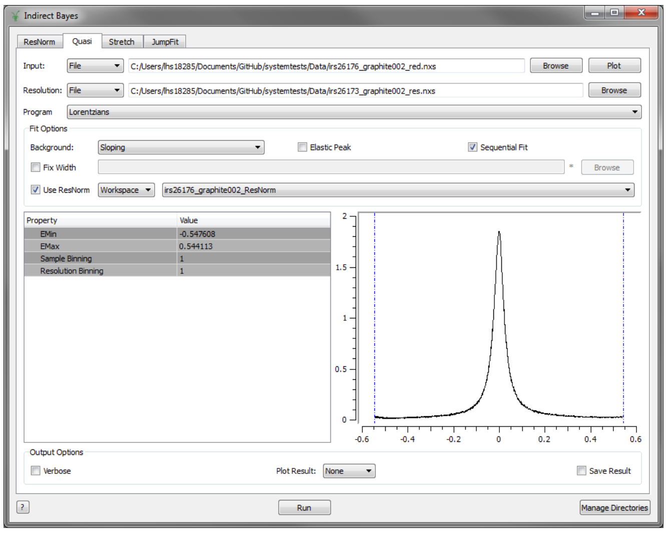
Resolution: File — C:/Users/lhs18285/Documents/GitHub/systemtests/Data/irs26173_graphite002_res.nxs

Property	Value
EMin	-0.457895
EMax	0.375789
Van Binning	1

☐ Verbose Plot Result: None ☐ Save Result

Figure 6.16: *Screen shot of using the ResNorm interface in release 3.2.*

The Quasi routine offers a complementary alternative to the ConvFit routine. The workspace output from ResNorm can be used as input to the Quasi routine. Like the ConvFit routine this takes both a reduced sample and resolution file. The model to be fitted can be configured using the various options for the background and type of peak before running. Figure 6.17 shows an example of using the interface in release 3.2.



Input: File C:/Users/lhs18285/Documents/GitHub/systemtests/Data/irs26176_graphite002_red.nxs

Resolution: File C:/Users/lhs18285/Documents/GitHub/systemtests/Data/irs26173_graphite002_res.nxs

Program: Lorentzians

Fit Options

Background: Sloping

[ ] Elastic Peak

[✓] Sequential Fit

[ ] Fix Width

[✓] Use ResNorm Workspace irs26176_graphite002_ResNorm

Property	Value
EMin	-0.547608
EMax	0.544113
Sample Binning	1
Resolution Binning	1

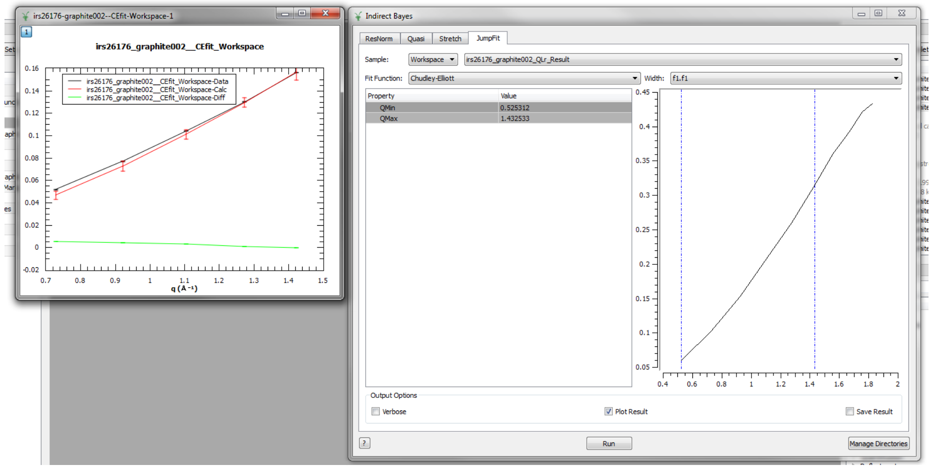
[ ] Verbose

Plot Result: None

[ ] Save Result

Figure 6.17: *Fitting a model to some reduced data from IRIS using the Quasi interface.*

As a final example, shown in figure 6.18, the workspace of fit parameters output from Quasi can be used as direct input to the JumpFit routine. This automatically finds the width parameters within the workspace and allows the user to select which one to fit. The user can choose between any of the models defined for diffusion. Future releases hope to add support for using a custom function string as the model for fitting.


irs26176-graphite002--CEfit-Workspace-1
irs26176_graphite002__CEfit_Workspace
irs26176_graphite002__CEfit_Workspace-Data
irs26176_graphite002__CEfit_Workspace-Calc
irs26176_graphite002__CEfit_Workspace-Diff
q (Å⁻¹)
Indirect Bayes
ResNorm
Quasi
Stretch
JumpFit
Output Options
Run
Manage Directories

Sample: Workspace irs26176_graphite002_QLr_Result

Fit Function: Chudley-Elliott

Width: f1.f1

Property	Value
QMin	0.525312
QMax	1.432533

[ ] Verbose

[✓] Plot Result

[ ] Save Result

Figure 6.18: *Using JumpFit on the output from the Quasi interface.*

7 Materials Modelling

Another important part of experimental analysis involves the comparison of experimental results against theoretical predictions. This is an area of development that has only just begun with only a very small amount of support implemented so far.

7.1 Indirect Simulation

Indirect Simulation is a new interface for release 3.2 that splits the basic nMolDyn loader that is currently in Mantid off from its original location in the Indirect Load ASCII interface. This interface is planned to be a place to expose simulation routines to the GUI level, such as adding interface support for the Sassena routines already in Mantid in future releases.

7.2 nMolDyn

The nMolDyn interface that currently exists is a very simple routine to load data from nMolDyn simulation files into a Mantid workspace. In future releases of Mantid it is proposed to attempt to properly integrate support for the nMolDyn python libraries into Mantid, providing a much richer selection of tools for the user. The code for this can be found in the IndirectMolDyn.py file in the Inelastic scripts folder.

7.3 Density Of States

The DensityofStates algorithm was a new addition in the 3.1 release of Mantid and does not currently have GUI support. It provides a loading algorithm for .castep and .phonon files generated by the CASTEP simulation program and creates workspaces with both total and partial density of states data. Further support for CASTEP is planned in future releases. The code for this algorithm can be found in the DensityOfStates.py file under the python algorithm plugins section of the Mantid install.

8 Planning For the Future

This section outlines outstanding work required to fully support indirect geometry instruments within Mantid. This section is not meant as a "set in stone" plan for further development but is meant as a guide for future development work in the area and a loose record of user and project requirements all of which are subject to change.

8.1 Conversion of Existing Routines to Algorithms and Work-Flow Algorithms

Now that a large proportion of the code base required for data reduction and analysis of data from indirect geometry spectrometers has been incorporated into Mantid, it is time to move forward with consolidating the existing code base we already have. As already mentioned in the earlier sections of this document most of the code base is either using the old style reducer-reduction step framework or is in the form of simple python scripts which are both not sufficiently documented and (largely) untested.

Ideally, the indirect geometry framework should take advantage of the core concept of the Mantid framework: algorithms. An algorithm in Mantid is defined as being a set of code that performs some operation on a workspace. Algorithms are well documented (both for the user and the development team) and are the preferred common interface for performing actions in Mantid.

There are two types of algorithms in Mantid. Normal algorithms are simply ones which perform some operation on a workspace, such as ConvertUnits or Fit or Load. The second type is called a work-flow algorithm. This is an algorithm that performs no direct manipulation of a workspace itself, but executes a collection of other algorithms in order to perform some larger task. An example of such an algorithm would be DgsReduction or ReflectometryReductionAutoOne. The benefits of the latter are that:

- Large operations get broken do into smaller, more easily understood operations which already have algorithms defined and are used by multiple research groups and facilities and have become tried and tested.
- Algorithms and work-flow algorithms still support inheritance like the reducer objects
- They are easily unit testable, meaning that automated checking of the integrity of each part of the reduction is handled automatically by the build servers, which helps prevent bugs.
- They are by default exposed through the Mantid simple python API like any other algorithm which means they can easily be chained together with other algorithms and can be run even if the interface is broken.
- As of release 3.2 the ability to track the history of a work-flow algorithm and its child algorithms has been added, allowing scientists to see exactly what operations have been performed on a workspace since it entered Mantid.

Energy transfer reduction and diffraction reduction are both prime candidates for work-flow algorithms. Neither of these routines actually perform any real direct manipulation of the workspaces they reduce themselves, but delegate all of the operations to existing Mantid algorithms within the reduction steps. Converting this would be a delicate and large job, but not necessarily difficult. The basic outline of both reductions is already documented in diagrams 5.2 and 5.7 respectively. This, along with the actual code (which is reasonably well documented) should be enough for an easy transition to a work flow algorithm for each type of reduction.

Some of the large reduction steps would also benefit from being converted to work-flow algorithms in there own right, particularly the reduction steps which are larger and shared between both reducer classes. Prime examples are the LoadData and HandleMonitor steps which are both complex enough that they could be converted to work-flow algorithms.

To convert the existing reduction routines to work-flow algorithms, I would suggest first writing detailed unit tests to cover all of the reduction steps. The full energy transfer reduction itself is already covered by automated system testing to check the integrity of the results. The next step would be to covert the reducer classes to be algorithms one at a time and hook them up with the relevant GUI. The final step would be to convert the reduction steps to either be algorithms or work-flow algorithms in there own right or to simply incorporate the code into the reduction work-flows themselves. Work-flow algorithms are just a class and therefore support inheritance which means common code between both reducers should end up being stored in a common base class.

Outside of the main reducers, the other scripts used throughout the Indirect framework should be converted to be proper Mantid algorithms for the reasons listed above. However, many of these routines are considerably simpler than the reducer work-flows and for many of them it may be best just to convert them to plain algorithms. Many routines, such as the the data analysis fitting routines MSDFit, FuryFit and ConvFit share very similar functionality with only a few slight differences between implementations. In the case of these routines, they all essentially pass a function string to PlotPeakByLogValue or Fit and convert the table workspace output from the algorithm to a matrix workspace of parameters. These could probably be refactored to be a single algorithm which handles the fitting for all three routines. Consolidating the code base in this way could also be done with the Bayesian fitting routines Quasi and Stretch but should only be done after porting the Fortran modules (see section 8.5).

8.2 Better Automated Test Coverage

One key development area in which the indirect framework is lacking is a good level of test coverage. At the time of writing the Mantid development system carries out two types of automated testing: unit tests and system tests. Unit tests are designed to be quick to execute and run frequently. System tests are typically longer and check the the integrity of the system as a whole.

Currently the only automated testing that is carried out by the system is through the system tests. These tests run one or more of the routines and compares the results against a reference file to check that the results still match after changes have been made to the framework. This is essential for checking that nothing gets broken in the core parts of the system, such as the energy transfer and diffraction reduction but in no way covers every aspect of the system. There are currently no unit tests covering any of the scripts mentioned in the previous sections, except for algorithms such as SofQW, SofQWMoments, and DensityOfStates.

A small maintenance project needs to be undertaken to increase the automated test coverage across all areas of the indirect framework. Ideally every function, routine, reducer, and reduction step should have a unit test or suite of tests covering it. Improving the test coverage offers little direct benefit to the users of the system, but will make it easier to identify and prevent bugs within the system ultimately leading to more time spent on productive development and less time fixing bugs.

Additionally, more system tests need to be added to more extensively cover some of the common use cases between different routines. For example, there is currently a test for running Fury and passing it to FuryFit, but no such test for running Quasi and passing the result to JumpFit.

8.3 Improved GUI structure and design

All of the interfaces for indirect data reduction and analysis share common themes in GUI design (except diffraction, largely due to its simplicity). They all share the same basic format: one tab for each routine, input files at the top of the window, parameters in the middle and a mini plot if required, output options at the bottom followed by the run button.

Having a consistent feel across all interfaces is extremely important in making it easy for users to learn to use the software and there is plenty that could be done to improve this within the existing GUIs. However, a more pertinent issue, and one that should be tackled first, is the repetition of common code both between interfaces and between individual tabs. Notice that the class diagrams in the GUI overviews in sections 5.3.1, 6.5.2, and 6.5.1 share a very similar class structure.

Future development should aim to unify this repeatable structure into a using a single common class hierarchy with abstract base classes sharing functionality that is shared between all of the interfaces. This would include, for example:

- Mini plot set-up, drawing, updating, and management routines.
- Common file loading and pre-processing calls.
- Algorithm instantiation, execution, and error handling code.

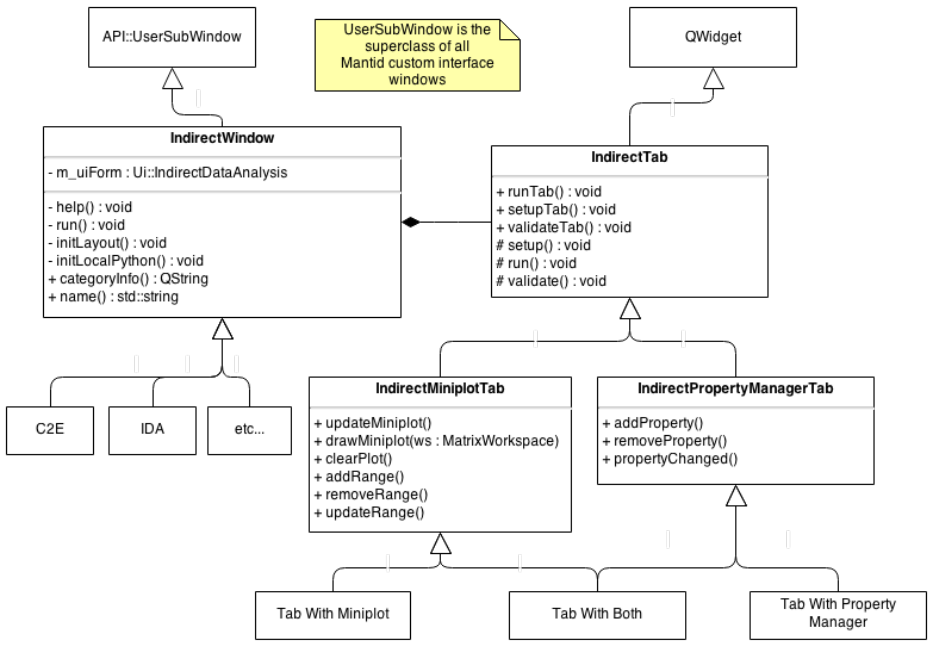


Figure 8.1: *Class diagram for what a refactored GUI hierarchy might look like.*

A refactored GUI class hierarchy could look something like diagram 8.1. This shows that all of the interface classes share a common abstract base class. More specific abstract base classes then implement functionality which is dependant upon what is contained within the interface (e.g. a mini plot) until reaching the concrete class which contains the exact details of setting up, validating and running a specific routine. This allows us to share common code that has already been written multiple times throughout the various interfaces, but have it defined in only one place.

The GUI window itself is a single abstract base class with a single interface file. All of the indirect GUIs follow the same pattern: tabs for each routine with run, help, and manage directories buttons at the bottom of the window. The name and specific settings of the window can be then be implemented using a concrete derived class for each individual interface.

Furthermore, to expand on the last bullet point above, the interface currently executes python routines by creating a string of python code with the calls to each routine and all of the parameters it requires. This approach is both fiddly and bug prone. It is easy to build a string which is not formatted correctly in some subtle way which leads to broken options and code paths.

A much better approach would be to convert the existing routines to Mantid algorithms (as discussed in the previous section) and then use Mantid's algorithm runner class to execute the desired algorithm on the separate thread. This approach makes it much harder to break because any errors in preparing the algorithm are likely to be raised at compile time as opposed to run time (as with python scripts). It also means that the errors occurring during runtime can be caught by the interface (which cannot be done using the current method) and handled appropriately.

The code to handle setting up and executing algorithms would not be overly complicated, Mantid already has an algorithm runner class, but it could be adapted further if required and could be written into the base class of the GUI hierarchy. This would allow it to be written once and used everywhere.

One final thing to note is that this should wait until the energy conversion for direct geometry instruments is split from the indirect geometry interface so that restructuring the interface does not effect development in that area.

8.4 Simulation Support in Mantid

Support for simulation is an area in which there has been little development so far for indirect, but would be a useful addition to our existing collection. As mentioned in section 7 there is a small amount of existing support for some simulation programs within Mantid. There are currently loaders for the output of CASTEP, Sassena, and nMOLDYN simulations. In the most recent release, a new interface was created as a place to integrate simulation programs into Mantid's GUI.

One current goal in the development of Mantid is to attempt to integrate the open source nMOLDYN library into Mantid and package it with the distribution of the project. This would not only further enrich the existing Python API available in Mantid, but would also provide us with an extensive, reliable molecular dynamics simulation package with which to build on.

Approval has already been given to include the library in Mantid as the functionality it provides has been deemed generic enough to be beneficial to a significant number of Mantid users. The first step in this project would be to separate the the nMOLDYN library from its dependencies and examine what libraries it requires that Mantid does not already have support for. Assuming that there are then no dependencies that cannot be shipped directly with Mantid, further development could then focus on building useful simulation routines for comparison with experimental data in conjunction with the requirements of instrument scientists.

8.5 Conversion of Remaining Fortran Routines

As mentioned in several sections through out the documentation of existing development, are several routines that still rely on Fortran routines that have been directly assimilated into Mantid using the F2Py package. This has allowed quick integration of routines needed for analysis, but it is not a cross platform solution, is not well documented, and the current code is incredibly difficult to decipher.

A long term goal would be to convert these existing, tried and tested routines into well documented, thoroughly unit tested C++ or python code which utilises the some of the existing fitting routines that are already present inside Mantid. As mentioned in the previous section, some of the functionality is repeated inside the code for Quasi and Stretch and it may be the case that common code could be merged during the conversion.

The biggest difficulty with this is that it will require a sizeable chunk of a developers time and a good understanding of Fortran and what the underlying code is doing in order to convert the routines and write thorough tests for the converted code to ensure the fitting is good enough to match the existing Fortran, while still matching the process described in the references.

There are currently four routines that are still in Fortran code: the three Bayes programs (ResNorm, Quasi, and Stretch) and the cylindrical absorption corrections program used in the calculate corrections routine. There are additional Fortran based routines in the for multiple scattering (see section 8.9) which will also need to be ported to Mantid in future releases.

8.6 Support for VESUVIO

Due to the uniqueness of the instrument, VESUVIO is currently not officially supported by the Mantid framework. Significant development work has already been undertaken add the data reduction and analysis routines that VESUVIO requires into the Mantid framework. The majority of the modifications to the framework have already been implemented, but some areas such as multiple scattering corrections [28] and global fitting are still incomplete. Besides this there is currently no GUI support for VESUVIO within Mantid. Once all parties involved are in happy with the state of the prototype reduction and analysis routines in Mantid work can begin on developing a GUI. Broadly, this would require a separate interface under the indirect section of Mantid, loosely separated in to the following sections:

- **Loading:** This would provide a user interface for using the LoadVesuvio algorithm to get raw data into Mantid as well as providing plotting functions for examining the captured time-of-flight data.
- **Corrections:** The corrections interface would provide a user interface for calculating both multiple scattering [28] and gamma background [29] corrections and applying them to data loaded in the first tab.
- **Fitting:** The fitting interface is the most complex in the series. This will provide support for setting up a fit using the same procedures used in the Ref. [12], but in a more user friendly manner than the current implementation achieves. This would provide support for both the Gaussian and Gram-Charlier functions with the ability to easily set the intensity constraints and hermite polynomial expansion coefficients.

Additional sections may also be required, such as interfaces for the global fitting and fitting data directly in y-space, but a full design will wait until after the framework behind VESUVIO analysis is finalised.

8.7 Additional Support for TOSCA

TOSCA has had several previous incarnations before its current set up, and has been known as TFXA in the past [30, 31, 32, 6]. Mantid currently only has the most recent set up for the instrument defined. The older incarnations should be able to be supported fairly easily using tools already in the Mantid framework. The geometry of each instrument is defined using a instrument definition XML file. This XML file has a date valid from and to attribute that allows Mantid to figure out at load time what instrument definition to use with a particular run. Each older incarnation can be defined using this attribute by setting it to be valid within the appropriate date range, meaning the correct definition for TOSCA will be loaded at runtime for each sample reduced based on the age of the sample run.

A prototype instrument definition file has already been tested for TFXA, but additional work will be required to properly support the full range of TOSCA's history. In particular a full description of the geometry of the instrument in 3D space is required for each prior incarnation for an accurate definition.

Another requirement by the TOSCA team is to add support for the Renishaw Ramen spectroscopy instrument which is used to perform Ramen spectroscopy and neutron spectroscopy simultaneously. The data from the Renishaw instrument can currently be imported into Mantid through the ASCII loading algorithms already in place, but this is an awkward process. A better approach would be to provide a way to store the runs on the TOSCA data archive and then provide procedures for loading the run directly from the archive as is currently done for regular neutron sample runs on TOSCA.

8.8 Improved Support for Other Facilities

Officially the framework we currently have supports both the SNS and ISIS. However, there has been a lot of development during the last release to further support for the ILL within Mantid. One of the goals of future development should be to integrate the existing loading and reduction routines with the existing GUI. This would eventually lead to the removal of the Indirect Load ASCII interface which would no longer be required.

8.9 Multiple Scattering Support

The multiple scattering routines from the MODES package are the last major parts that have not been ported from the MODES application in one form or another. There are two different multiple scattering routines called MuscatData and MuscatFunc which were originally part of the MODES package as the program MINUS which in turn was adapted from the DISCUS program [3, 33]. Both routines work with both flat and cylindrical geometries. The difference between the two is MuscatData uses takes $S(Q, w)$ workspace as input, while MuscatFunc just uses a set of specified parameters from the user.

Ideally these routines should be converted from Fortran to C++ code first before including them in Mantid, but if demand is high they can be included in the GUI in the same way Bayes and cylindrical corrections works using the F2Py package. There has already been some existing work to import them into Mantid, but the current versions do not work. Recently, there has also been a goal outlined by the scientific steering committee to provide a general framework for multiple scattering corrections within Mantid. At the time of writing it is uncertain whether the planned implementation of this will be compatible with the existing methods used by the MODES routines.

Acknowledgements

The author would like to thank S. Mukhopadhyay for her assistance in the writing of this report and in particular for her guidance and support with the theory section of the document. Additional thanks W.S. Howells for his enumerable contributions to the development of the indirect geometry section, particularly in regards to QENS analysis, and F. Fernandez-Alonso for his constructive comments and criticisms.

References

[1] Mantid Project. *Mantid: Manipulation and Analysis Toolkit for Instrument Data.* 2013. doi: 10.5286/SOFTWARE/MANTID. URL `http://dx.doi.org/10.5286/SOFTWARE/MANTID`.

[2] F. Fernandez-Alonso and D. L. Price. *Neutron Scattering Fundermentals.* Academic Press, 1st edition, 2013.

[3] W.S. Howells, V. Garcia-Sakai, F. Demmel, M.T.F. Telling, and F. Fernandez-Alonso. *MODES User Guide, Version 3.* Technical Report RAL-TR-2010-006, Rutherford Appleton Laboratory, 2010.

[4] W.S. Howells. *IDA - IRIS Data Analysis.* Technical Report RAL-TR-96-006, Rutherford Appleton Laboratory, 1996.

[5] S. Mukhopadhyay. *How to use Mantid for low energy inelastic neutron scattering data analysis on indirect geometry instruments.* Technical Report RAL-TR-2014-005, Rutherford Appleton Laboratory, 2014.

[6] D. Colognesi, M. Celli, F. Cilloco, R.J. Newport, S.F. Parker, V. Rossi-Albertini, F. Sacchetti, J. Tomkinson, and M. Zoppi. *TOSCA neutron spectrometer: the final configuration.* Applied Physics A, 74(1):s64–s66, 2002.

[7] J. Mayers and M.A. Adams. *Calibration of an electron volt neutron spectrometer.* Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 625(1):47–56, 2011.

[8] V. Garcia-Sakai, M.A. Adams, W.S. Howells, and M.T.F. Telling. *The IRIS user guide.* Technical Report RAL-TR-2011-004, Rutherford Appleton Laboratory UK, 2011.

[9] M.T.F. Telling and K.H. Andersen. *The OSIRIS user guide.* Technical Report RAL-TR-2003-016, Rutherford Appleton Laboratory UK, 2003.

[10] S.F. Parker, J. Tomkinson, A.J. Ramirez-Cuesta, and D. Colognesi. *The TOSCA user manual.* Technical Report RAL-TR-2003-015, Rutherford Appleton Laboratory UK, 2003.

[11] E.M. Schooneveld, J. Mayers, N.J. Rhodes, A. Pietropaolo, C. Andreani, R. Senesi, G. Gorini, E. Perelli-Cippo, and M. Tardocchi. *Foil cycling technique for the VESUVIO spectrometer operating in the resonance detector configuration.* Review of scientific instruments, 77(9):095103, 2006.

[12] J. Mayers. *User guide to VESUVIO data analysis.* Technical Report RAL-TR-2011-003, Rutherford Appleton Laboratory UK, 2010.

[13] D. S. Sivia. *Elementary Scattering Theory.* OUP Oxford, 2011.

[14] J Mayers and G Reiter. *The VESUVIO electron volt neutron spectrometer.* Measurement Science and Technology, 23(4):045902, 2012.

[15] U.P. Wild, A.R. Holzwarth, and H.P. Good. *Measurement and analysis of fluorescence decay curves.* Review of Scientific Instruments, 48(12):1621–1627, 1977.

[16] D.S. Sivia, C.J. Carlile, W.S. Howells, and S. König. *Bayesian analysis of quasielastic neutron scattering data.* Physica B: Condensed Matter, 182(4):341–348, 1992.

[17] J.S. Higgins, G. Allen, R.E. Ghosh, W.S. Howells, and B. Farnoux. *Observation of quasi-elastic broadening in neutron scattering from polymer solutions.* Chemical Physics Letters, 49(2):197–202, 1977.

[18] J.S. Higgins, R.E. Ghosh, W.S. Howells, and G. Allen. *Q 4-dependent broadening in quasielastic incoherent neutron scattering from a polymer melt.* Journal of the Chemical Society, Faraday Transactions 2: Molecular and Chemical Physics, 73(1):40–47, 1977.

[19] C.T. Chudley and R.J. Elliott. *Neutron scattering from a liquid on a jump diffusion model.* Proceedings of the Physical Society, 77(2):353, 1961.

[20] P.L. Hall and D.K. Ross. *Incoherent neutron scattering functions for random jump diffusion in bounded and infinite media.* Molecular Physics, 42(3):673–682, 1981.

[21] J. Teixeira, M.C. Bellissent-Funel, S. Chen, and A. Dianoux. *Experimental determination of the nature of diffusive motions of water molecules at low temperatures.* Physical Review A, 31(3):1913, 1985.

[22] A. Fick. *V. On liquid diffusion.* The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 10(63):30–39, 1855.

[23] C. J. Carlile. *Spectrum correction factors for sample holder and self-shielding effects for planar samples in thermal neutron scattering studies.* Technical Report RL-74-103, Rutherford Appleton Laboratory, 1974.

[24] A. K. Soper, W. S. Howells, and A. C. Hannon. *Analysis Of Time-of-flight Diffraction Data from Liquid And Amorphous Samples.* Technical Report RAL-89-046, Rutherford Appleton Laboratory, 1989.

[25] H. H. Paalman and C. J. Pings. *Numerical Evaluation of XRay Absorption Factors for Cylindrical Samples and Annular Sample Cells.* Journal of Applied Physics, 1962.

[26] P. G. Radaelli. *ARIEL Manual*, 2000.

[27] S. Jackson. *VESUVIO Data Reduction and Analysis in Mantid.* Unpublished report., 2014.

[28] J Mayers, AL Fielding, and R Senesi. *Multiple scattering in deep inelastic neutron scattering: Monte Carlo simulations and experiments at the ISIS eVS inverse geometry spectrometer.* Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment, 481(1):454–463, 2002.

[29] J Mayers. *Calculation of background effects on the VESUVIO eV neutron spectrometer.* Measurement Science and Technology, 22(1):015903, 2011.

[30] J Penfold and J Tomkinson. *The ISIS time focussed crystal analyser spectrometer, TFXA.* Technical report, Rutherford Appleton Lab., Chilton (UK), 1986.

[31] SF Parker, CJ Carlile, T Pike, J Tomkinson, RJ Newport, C Andreani, FP Ricci, F Sacchetti, and M Zoppi. *TOSCA: a world class inelastic neutron spectrometer.* Physica B: Condensed Matter, 241: 154–156, 1997.

[32] ZA Bowden, M Celli, F Cilloco, D Colognesi, Robert J Newport, SF Parker, FP Ricci, V Rossi-Albertini, F Sacchetti, J Tomkinson, et al. *The TOSCA incoherent inelastic neutron spectrometer: progress and results.* Physica B: Condensed Matter, 276:98–99, 2000.

[33] M. W. Johnson. *Discus: A computer program for the calculation of multiple scattering effects in inelastic neutron scattering experiments.* Technical Report R7682, Atomic Energy Research Establishment, 1974.