Chapter 1

A script for processing DAFS data

This chapter will aim to focus on the details of the bespoke computational software that has been developed during the course of this thesis in order to process, analyse and extract useful DAFS data. In order to maximise user friendliness and applicability, the program is available as both a *.py* file and as a graphical user interface. I will primarily focus my explanations on the subject of the GUI on the basis that this will likely be preferred by the user, and accompany psuedo-code explanations with figures of code taken directly from the program wherever relevant or useful. The program is also available on GitHub and is executable on Windows, Mac and Linux systems alike. Due to the inherently complex nature of DAFS data analysis, it is also necessary for this chapter to include potential limitations associated with the analysis procedure, to present notes where users should take care, and to provide useful advice for places where users may run in to some common issues when troubleshooting.

1.1 Data extraction

1.1.1 Creating an Interface

The first step in processing DAFS data is to import the powder X-ray diffraction pattern output from the detector at each incremental energy step in the energy scan, as well as the concurrent transmission x-ray absorption spectrum. In the event that the data is collected in fluorescence mode, the program can be adapted to account for this simply by adjusting the column number that is imported, given that the detector outputs data in a similar way. In any case, tailoring the method of importing the

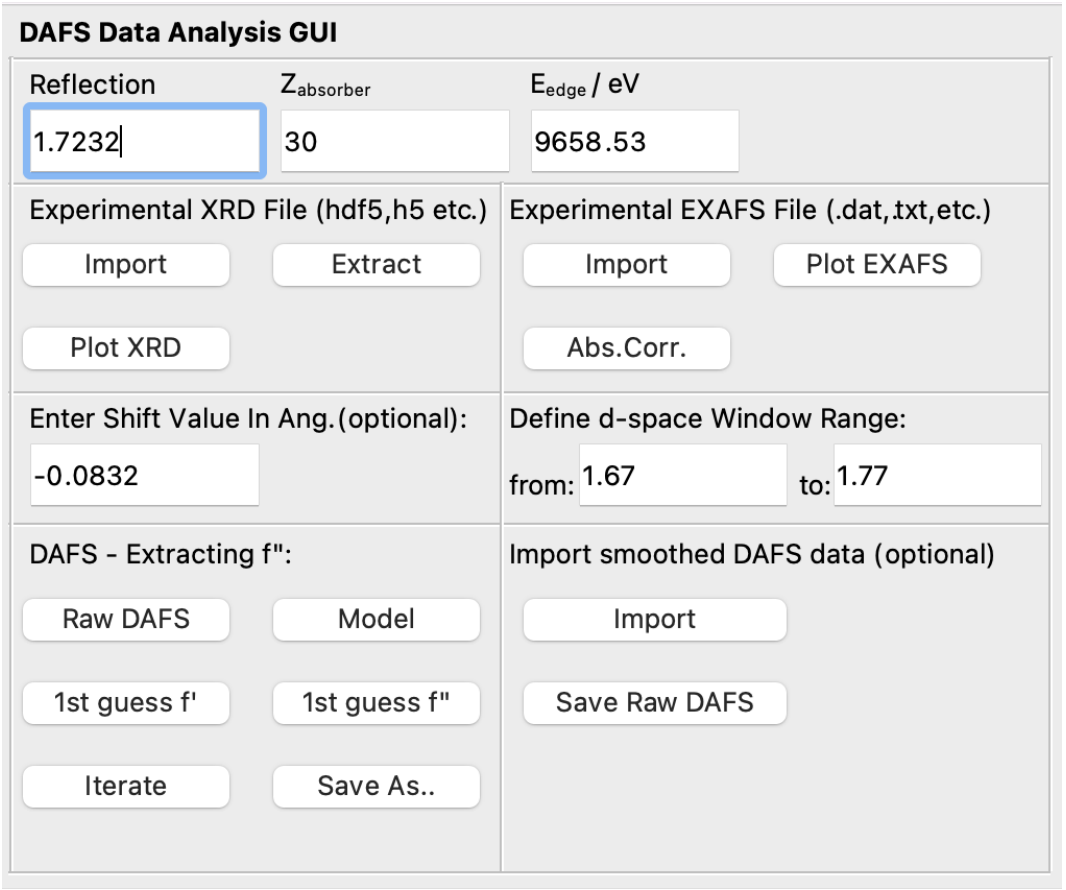

Figure 1.1: The graphical user interface built and used for DAFS data extraction, processing and analysis throughout this thesis.

data to a given detector should be a reasonably facile process. I have restricted the explanation to the context of the PILATUS detector on the basis that this detector was found to be superior for our DAFS data analyses, however the method of importing diffraction data for other detectors such as the Mythen and Vortex should be relatively simple to implement if desired.

To create this user interface, *PyQt5* [**?**] has been used, which functions by generating a main window class, within which we define a series of 'widgets' which generate the facade of the UI. Each widget is linked to a button, which is linked to a specific block of code that is executed when the corresponding button is pressed. The full source code in the context of an example dataset of tetrahedral $ZnFe_2O_4$ can be found in the appendix. This is provided in the context of the non-GUI format on the basis that the code is shorter and easier to follow, although the UI-based source

code is available on *GitHub*, accompanied by the relevant supporting documentation.

1.1.2 Importing diffraction data

The explanations presented in the subsection attempt to illustrate clearly how the program imports data for different detectors, and explain the first section of the user interface responsible for executing the first stage of the DAFS analysis procedure: the *Experimental XRD File* section.

1.1.2.1 PILATUS

To import the data using a PILATUS detector, the program first allows the user to locate the file in their system using the *'import'* button. This works by defining a file name using *PyQt5* module components. *QFileDialog* is first called to open the window for the user to search for their desired file, and then *QFileInfo* to relate the name of the selected file to the *'filename'* variable. The PILATUS detector outputs XRD data as a *.hdf* file, which is a 3D array of z number of XRD patterns, each of pixel size x by y. The program imports the data using the `hdf5_lib` module:

The prefix 'self' simply denotes the class that is responsible for containing and executing the GUI. We also define the number of energypoints, and the size of the rows and columns of the XRD patterns. While I have omitted some code for the sake of clarity, the full code is available in the appendices, or else on GitHub as referenced above. The for statement on line 16 of figure 1.2 serves to provide the user information on how best to select indices for the calculation of a circle to linearize and index the XRD patterns sequentially as explained in section 1.2.1 below.

1.2 Obtaining a DAFS spectrum

There are many steps in proceeding from the raw data, a set of 500-2000 diffraction patterns, to a final DAFS-extracted-XAS derived from the site-separated f". This program attempts to present an intuitive step-by-step guide for doing so, with an accompanied graphical user interface to improve user-friendliness. After importing the data, the next step is to index the raw diffraction data.

```
#importing modules
import os
from hdf5_lib import *
from PyQt5.QtCore import QFile, QFileInfo, QRect

filename = QFileDialog.getOpenFileName(MainWindow, 'Open File') #opens a window allowing the user to select their desired file.
self.filename = filename[0] #relates the first element of the tuple (i.e. the XRD file name) to the main window class via 'self'.

self.XRD=hdf_get_item(filename,'entry/data/data') #self.XRD is a 3D array containing an 'n_energypoints' number of images, each image being an XRD at 1 energy

self.n_energypoints= shape(self.XRD)[0] # number of incremental energy points
n_column = shape(self.XRD) [1] # The height of each XRD image, 195 pixels
n_row = shape(self.XRD) [2] # The length of each XRD image, 1475 pixels

# The following 'for' statement prints the row at which the pixel generates the highest intensity at each column
for i in range(n_column):
    buf=z[i,:]; index1=np.where(buf==max(buf));
    print(i, index1)

```

Figure 1.2: The first block of code responsible for executing the importing of XRD data.

1.2.1 Indexation

In order to extract the intensities, the circular XRD patterns must first be linearized. To do this, we define 3 points of a circle approximately at $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{3}{4}$ of the rows of pixels where the intensity is at a maximum. Since there are 195 columns, the 97th column is selected, for example, as the index at the middle column, as in figure 1.3, and the row at which its intensity is maximum is also printed.

By printing each maximum index at each column, we can decide which columns to select and avoid any discontinuities due to bright spots or similar maximum intensities that would compromise the linearization of the data, with the goal to reduce noise in the final product of site-separated f". Note that 33, 97, and 169 are example values and will be different for each dataset. We then define indices along the x axis for which the `buf` values for the y axis are at a maximum.

```
# The 3 columns for which the maximum intensity is taken to
    calculate the centre and radius of a circle for
    linearisation of the diffraction data
self.buf1 = 33
self.buf2 = 97
self.buf3 = 169

buf=z[self.buf1,:]; index1=np.where(buf==max(buf));
buf=z[self.buf2,:]; index2=np.where(buf==max(buf));
buf=z[self.buf3,:]; index3=np.where(buf==max(buf));

# The corresponding 3 rows at which the intensity is maximum
    for the specified rows.
print(index1,index2,index3)
```

Figure 1.3: The selection of 3 indices to be used for linearization.

```
self.Cen_x, self.Cen_y, rad = findCircle(int(index1[0]), self.
    buf1, int(index2[0]), self.buf2, int(index3[0]), self.buf3)
self.extracted_xrd = np.ones([len(self.x), np.shape(self.XRD)
    [0]+1])
```

Figure 1.4: The series of code used to find the centre and radius of the circle specified by the 3 indices, using the FindCircle function.

These indices are used to calculate a circle, find its centre, and then turn the rectangular image into a cone-like shape such that the rings of the diffraction data are linearized as opposed to curved. The method is based on 2 functions: `find_data_in_ROI` and `radial_profile`.

The `find_data_in_ROI` function takes 5 arguments. The first 3 are the x,y,z values for each dataset, where x and y are the axes for each image, and z is the energy point number for that given image. It then takes the centre of the circle as the other 2 arguments, given by `Cen_x` and `Cen_y`. It calculates the equation of the two lines using a polynomial fit, from the centre of the defined circle to the maximum and minimum values of y at the max x as in figure 1.5. The function then uses the coefficients generated from the polyfit to define `line_up` and `line_down` functions for each of the x values. This allows there to be an equation of 2 lines, generated by multiplying the coefficient by each value of x, which generates the x values of these lines and in turn has the effect of transforming the data from a 'strip' to a 'cone'. The full definitions of the functions used for circle calculation can be found in the

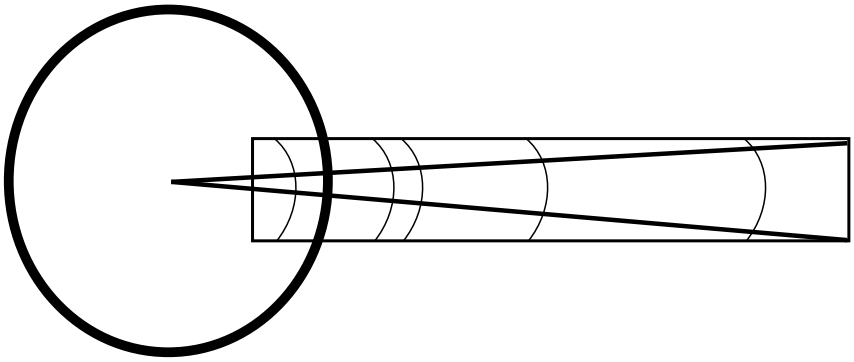

Figure 1.5: An illustration of the process for linearizing the diffraction data in the DAFS data analysis software.

```
def find_data_in_ROI(x,y,z, Cen_x,Cen_y):
    coeff_line_up = np.polyfit([Cen_x, max(x)],[Cen_y, max(y)
    ],1);
    coeff_line_down = np.polyfit([Cen_x, max(x)],[Cen_y, min(y)
    ],1);
    line_up=np.polyval(coeff_line_up,x); line_down=np.polyval(
    coeff_line_down,x);
    reduced_data = z
    for index1 in range(len(x)):
        for index2 in range(len(y)):
            if y[index2] > line_up[index1] :
                reduced_data[index2,index1]=0;
            if y[index2] < line_down[index1] :
                reduced_data[index2,index1]=0;
    return reduced_data
```

Figure 1.6: The function used for linearization of the Pilatus output.

full source code in the appendix.

With these 2 lines, we define 2 `for` loops for each index of the maximum intensity value in y, and iterate through x and y. So, if the value of y where the index of the maximum is, is greater than the `line_up` value of the other index, then the reduced data becomes 0. Then, the `radial_profile` function essentially calculates the point-by-point radius and populates a 1-D array with the intensity at each distance from the centre. The result is shown in figure 1.7. We then use a polynomial fit to align the x axis to 2θ using a reference XRD pattern, and this can be converted to d to ensure there is no shifting of the data.

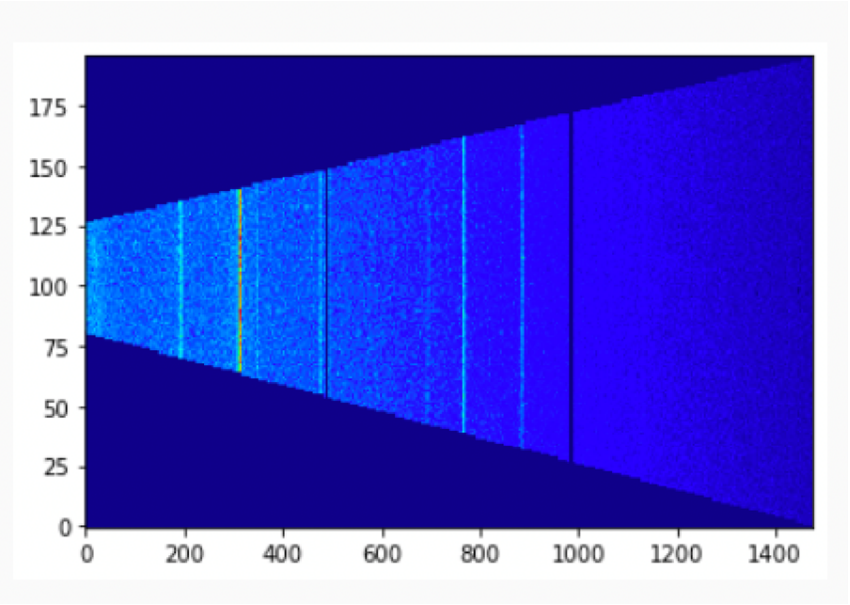

Figure 1.7: The result of linearizing output from the diffraction experiment for a $ZnFe_2O_4$ sample. Note that I have shown the high energy limit of the scan here to demonstrate the manifestation of fluorescence on the intensity of the pattern as a whole.

1.2.2 Peak selection

Once we have linearized the data, and have an understanding of our peaks of interest, we can specify a d-spacing range for the program to focus on, as shown in figures 1.8 and 1.9. We can also plot the entire set of XRD patterns in the energy scan using the `plot XRD` button. This is an interactive window where the user may zoom in and gain a better insight into how the data is presented. This overcomes potential limitations to do with inaccuracies in x-axis calibration and further increases the robustness of the program.

1.2.3 Background Removal

While the data presented in figure 1.8 above has a primitive method of background removal by subtracting the minimum value of intensity from each data point, it is important to increase the sophistication of this method in order to minimise any extraneous noise in the data arising from the program's peak integration method.

In order to do this, the program utilises a composite fitting model based on a psuedo-Voigt fitting profile for the XRD peak, and a linear model for the background,

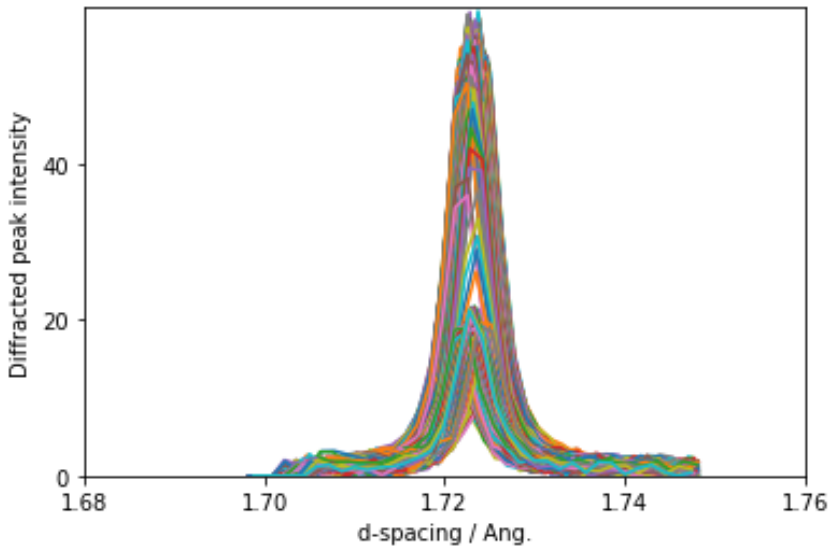

Figure 1.8: The data analysis program focuses on a user specified region of d-spacing where 1 peak exists, shown here, and extracts the background subtracted area under this peak by fitting a composite model based on a pseudo-Voigt fitting profile.

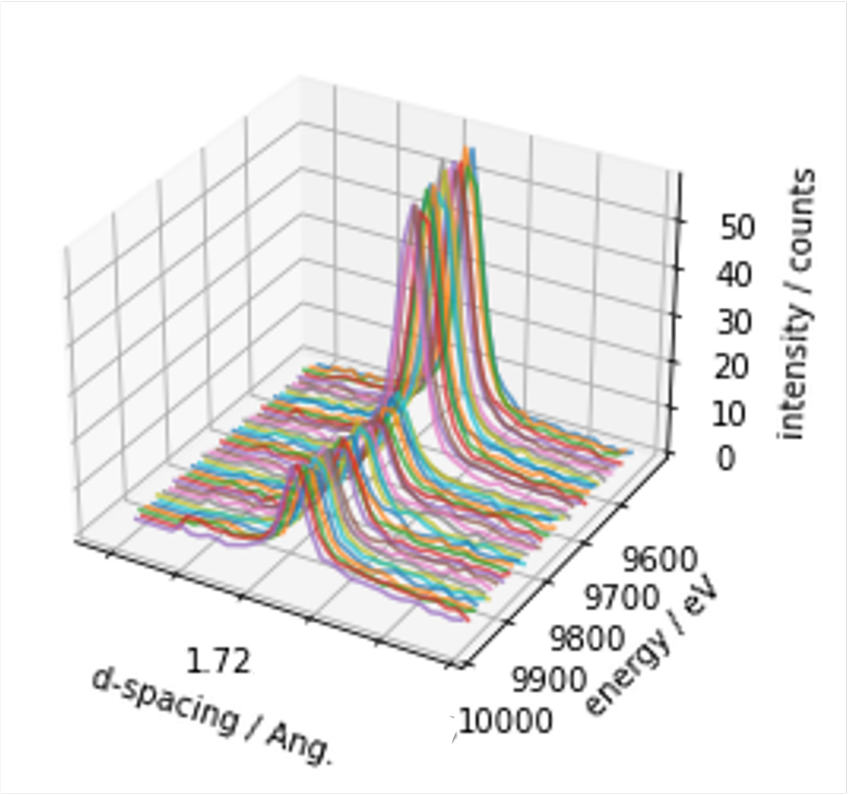

Figure 1.9: A 3D representation of the specified peak window as shown in Figure 1.8. For clarity, only 1 in 20 peaks are shown.

```
    # define a d specific range for the np.where command to use
    as its range
    dmin = float(self.d_min.text())
    dmax = float(self.d_max.text())

    # store the number of values of d, that is, the index of d,
    between the specified range, i.e. the 12th, 13th, 14th
value
    index_d = np.where((self.d >= dmin) & (self.d <= dmax))
    d_select = self.d[index_d[0]]

    # repeat this for the y-axis values
    y_select = I[index_d[0]]

    #defining parameters for the first fit only
    if i == 0:
        n = len(d_select)
        mean = sum(d_select) / n
        bkg = min(j for j in y_select if j > 0) #A nested loop
to only take bkg values above 0 to avoid and account for any
    glitches
        sigma = np.sqrt(sum((y_select-bkg)*(d_select-mean)**2)/
sum(y_select))
        amp = max(y_select)
        area_guess = 0.5

    #defining parameters for the rest of the fits so that they
start with the parameters calculated for the previous
iteration
    elif i != 0:
        amp = out.params.get('pv_height').value
        area_guess = out.params.get('pv_amplitude').value
        mean = out.params.get('pv_center').value
        sigma = out.params.get('pv_sigma').value
```

Figure 1.10: First part of the code responsible for fitting the composite model to extract the intensity under the peak at each energy. In this figure, the window is defined and the parameters for the fit are created.

as seen in figure 1.12. The software then extracts only the single background-subtracted pseudo-Voigt component of this model, and extracts the area under this peak by trapezoidal integration using the `trapz` function. The program also uses the shape of the previous peak as the starting point for the next point, which reduces the likelihood of errors arising due to fit failure. Once we extract the area, we normalise with respect to I_0 to produce a raw DAFS spectrum.

It should also be noted that, in this fit, the user specifies a window of d-spacing based on inspection of the plot of extracted powder-XRD patterns. With this window,

```
    # Lmfit built-in PseudoVoight Model, given a prefix to
distinguish from the background model components
    mod = PseudoVoigtModel(prefix = 'pv_')
    params = mod.make_params(pv_amplitude = area_guess,
pv_height = amp, pv_center=mean, pv_sigma=sigma)

    # Lmfit built-in Linear Model, given a prefix to
distinguish from the background model components
    mod_lin = LinearModel(prefix= 'lin_')
    params.update( mod_lin.guess(y_select, x=d_select))
    model  =  mod + mod_lin

    # Fitting the model function based on the desired x, y data
    and using the specified parameters
    init = model.eval(params, x=d_select)
    out = model.fit(y_select, params, x=d_select)

    # 'comps' separates the two components of the model
    comps = out.eval_components(x=d_select)
    res = (out.best_fit) # overall 'total' best-fit

    # Trapezoidal integration for the extraction of area under
the peak of the Pseudo-Voigt component of the model
    area = (trapz(comps['pv_']))

    # appends the value to form an iterative list for each
energy and presents this as an array.
    area_arr.append(area)
    area_arr = np.array(area_arr)
```

Figure 1.11: The second part of code responsible for fitting the composite model to extract the intensity under the peak at each energy. Here, the composite model is generated and fit to the experimental data specified by the user-controlled window.

the program selects all values of d and their corresponding I, and generates the composite model. In order to do this, the parameters for the first guess composite fit to the peak and background are generated based on some parameters, such as the maximum and minimum values of intensity, the number of data points in the user-specified d-spacing window, to generate initial values for the amplitude, mean, and standard deviation for the fit to begin from. Once the first peak is fit, the parameters calculated by the model are then used as input values for the next energy point. This drastically improves the robustness of the fitting process.

Once we have done this for each incremental energy, we are able to use the `raw DAFS` button in the user interface to plot the raw DAFS spectrum, which is a

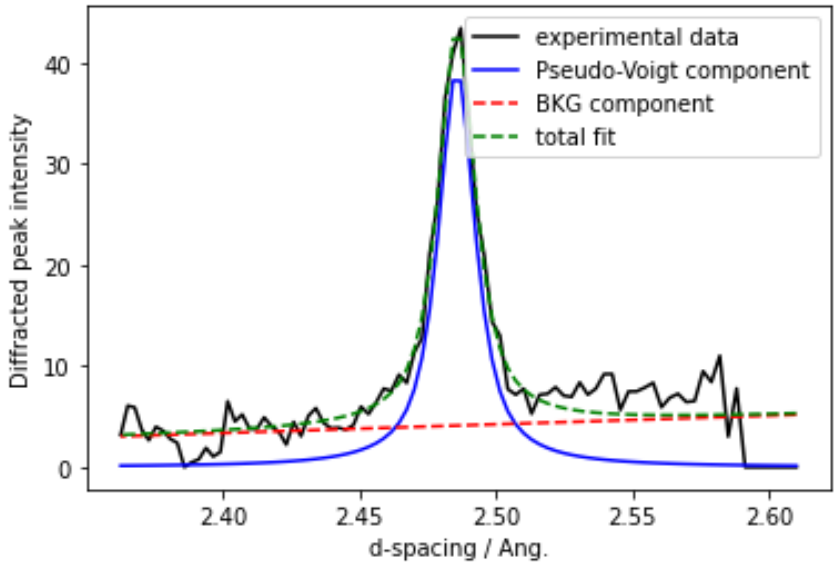

Figure 1.12: The process of composite fitting to a particular peak of interest in the diffraction data, for a single energy point in the scan only. Here, only the pseudo-Voigt component (blue) is extracted.

representation of the variation of intensity of the pseudo-Voigt integrated area as a function of energy, and which resembles figure **??**. It is from here that we can begin to model the data and extract the anomalous contributions.

Another important aspect of the program is the processing of the x-ray absorption spectrum that is run concurrent to the diffraction. The GUI has a section marked *Experimental XAS File* which typically is of the format *.dat*, *.txt* or some similar extension. This section allows us to import the energies, the I_0, and the transmitted intensity, as well as ultimately allowing us to calculate the absorption correction according to equation **??**. It is important that the user imports the absorption data prior to generating the raw DAFS spectrum to ensure the necessary parameters are imported for calculation of the raw DAFS. In the `Experimental XAS File` section responsible for the calculation and plotting of XAS, the data is imported as described for the XRD section above, and the user is able to manipulate the plot window to zoom into the relevant features, as in the XRD section. The program also saves all relevant data in the form of a .txt file at each step of the analysis procedure, so that the user can view and manipulate the data to match their objectives, in ATHENA or Origin for example.

1.3 Resonant term separation and extraction

Now the raw DAFS spectrum has been extracted, the next step is to separate and extract the resonant terms - the real and imaginary components of the scattering factor f' and f" - and ultimately isolate the f" via an iterative model. Once we have isolated a stable imaginary contribution, we can obtain a reflection-specific absorption spectrum, which we can use to generate information about the structure and electronics of single crystallographic sites. The first step in the resonant term separation and extraction process is to generate a model fit to the raw DAFS intensity.

1.3.1 Programming an iterative model

here...

The model can be defined by equation **??**, and is input into the `model` button of the user interface. Here I have replaced some of the variable names in the script with their corresponding parameters in equation **??**, to improve clarity. The starting values for atomic f' and f" values are obtained from the `diffkk` package [**?**], which takes the conventional XAS as an input to calculate better guess starting points for f' and f" that are based on the local environment of the central absorber, as compared with the Cromer-Liberman starting values. It does so by first performing a differential Kramers-Krönig transform on the difference between the $\mu(E)$ and the Cromer-Liberman value of the atomic f", I.E. the fine structure function χ" as cited in the literature [**?**], [**?**]. The result is then added to the atomic f'. The goal is to obtain anomalous scattering values that contain fine structure, to subsequently allow the iterative model a better likelihood of converging. This `diffkk` package has been recently integrated into Larch [**?**], a data analysis tool for XAS similar to the Demeter package already described. Contrary to some literature published in 2012 on the *Evolution of charge order through the magnetic phase transition of* $LuFe_2O_4$, `diffkk` does not compute the site-separated f' and f", instead it merely acts as a starting point for these values from which the iterative Kramers-Krönig based algorithm that we have developed during this work may proceed. The `slope` parameter of the I_{model} equation is an optional energy-dependent fitting parameter that may improve the model's fit to experimental data, but is typically set to 0. Once

```
 # Define a model function based on the Imodel equation and the absorption correction:
 def intensity(en, phi, beta, t, exafs, sin_theta, fprime, fsec, I0=1, slope=0, Ioff=0):
     costerm = (np.cos(phi) + (beta*fprime))
     sinterm = (np.sin(phi) + (beta*fsec))
     return I0 * (costerm**2 + sinterm**2) * ((1 - e**((-2*exafs*t)/sin_theta)) / (2*exafs)) + Ioff

 # Importing modules
 from larch.xafs import diffkk
 from lmfit.models import PseudoVoigtModel, LinearModel, Model

 # Generating both atomic and molecular starting points for f' and f"
 self.dkk=diffkk(E, $\mu(E)$, z= $z_{absorber}$, edge='K', mback_kws={'e0':$E_0$})
 self.dkk.kk()

 # Generate a model fit based on the starting guess parameters set by 'make_params'.
 imodel = Model(intensity, independent_vars=['en', 'fprime', 'fsec', 'exafs', 'sin_theta'])
 params = imodel.make_params(scale=0.2, offset=0.75, slope=0, beta= 0.1, phi= 6.28, t = 1)

 # Fitting a first guess and best fit model to the experimental data
 init_value = imodel.eval(params, en=ene, fprime=f1, fsec=f2, exafs = exafs, sin_theta = sin_theta)
 result = imodel.fit(experimental_raw_DAFS, params, en=ene, fprime=f1, fsec=f2, exafs = exafs, sin_theta = sin_theta)
 print(result.fit_report())

 # Plotting the data, model and first guess fit
 plt.plot(ene, (result.best_fit), '--', label='best fit (lmfit)', color = 'blue')
 plt.plot(ene, (init_value), '--', label='init value (lmfit)', color = 'black')
 plt.plot(ene, experimental_raw_DAFS, label='exp. data', color = 'red')
```

Figure 1.13: The generation of a model based on the I_{model} equation to fit to the experimental raw DAFS data.

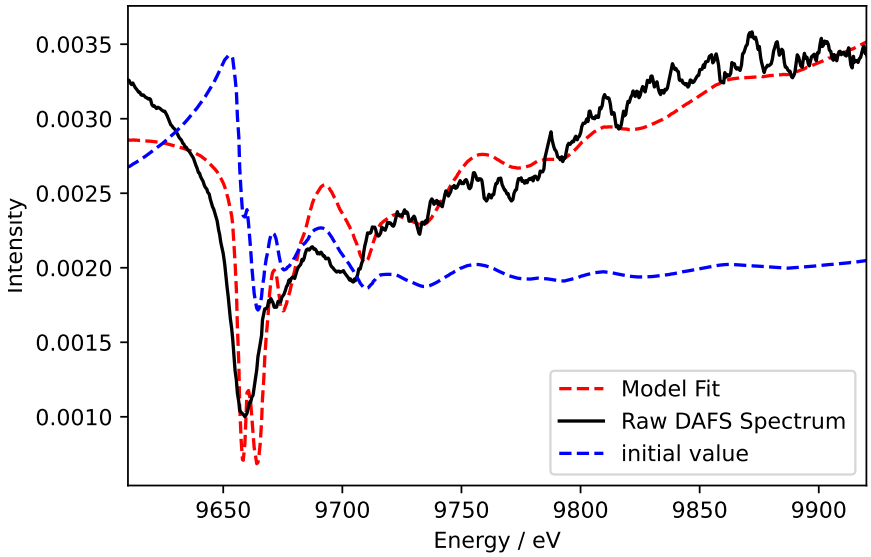

Figure 1.14: The experimental data of ZnO for which the first guess and model fit are calculated using lmfit.

we have obtained our initial values for f' and f", we fit a model to the experimental 'raw DAFS' data that resulted from fitting a composite Psuedo-Voigt model and integrating the area of the given Bragg peak at each incremental energy, we use the lmfit `Model` package to define our own parameters and function, and fit it to this experimental data.

Care must be taken by the user to ensure that the starting point value calculated by the `init_value` command is sufficiently close to the experimental data such that a reasonable minimum is reached, with parameters existing within sensible ranges. It is possible to constrain parameters using lmfit's sophisticated modelling function [**?**]. Plotting the initial first guess allows the user to inspect how changing the starting point of parameters affects the fit, and means that we can be sure the fit is converging toward sensible parameter values, meaning that the resultant calculated f' and f" will be accurate. An example of a first guess fit can be seen in figure 1.14

The model outputs parameters for the fit, and an example of these can be shown in table 1.1. These parameters are used to readily obtain f' and f" first guesses according to the corresponding buttons in the user interface. The f' is easily obtained by defining a function as described in eq. **??** and adjusting the $\pm$ sign as necessary

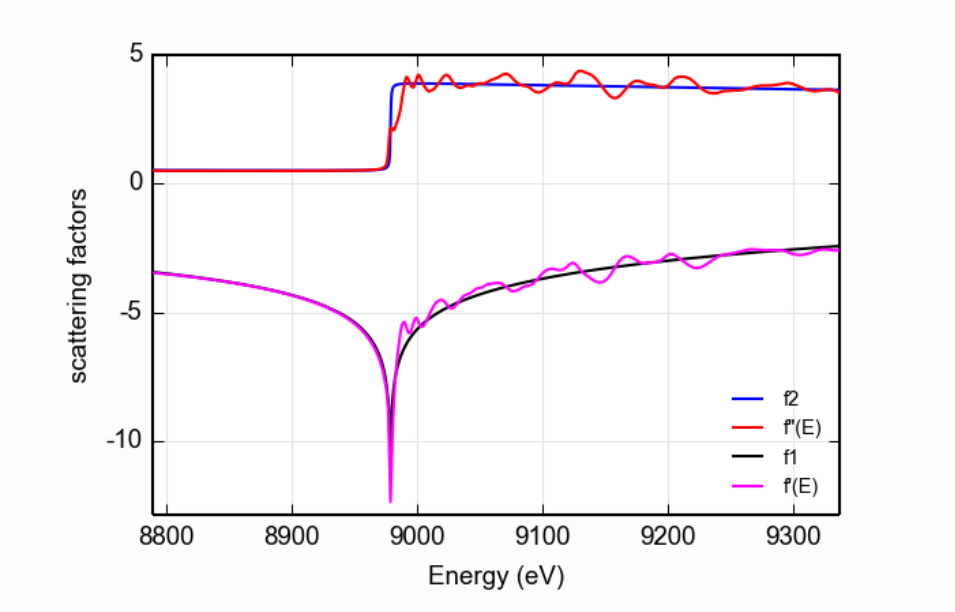

Figure 1.15: The anomalous scattering factors determined for copper metal from a copper foil, compared with the bare-atom, Cromer-Liberman values. Figure obtained from the Larch diffkk webpage [**?**].

Table 1.1: An example of the resulting parameters obtained for the fit in figure 1.14.

Parameter	Value
ϕ	5.72
β	0.022
I_{off}	-0.27
I_0	3.96
t	0.39

to produce an f' that contains fine structure as in figure 1.16. It is also possible to calculate a starting guess to f", and transform to f' using the KK transform. This may be useful if the f' produced is not as the user desires. At the time of writing, this functionality is only compatible with the raw `.py` file, and not yet with the GUI. Using this method, although not preferred by *Cross*, did allow for the use of the `diffkk` algorithm, applying it to the calculated f" to calculate the corresponding f' via it's built-in Kramers-Krönig transform.

When coding the iterative Kramers-Krönig algorithm to transform from the real to imaginary components of the anomalous scattering factor, we apply a lambda function as in figure 1.17. This f' and f" are then input back into the Imodel equation,

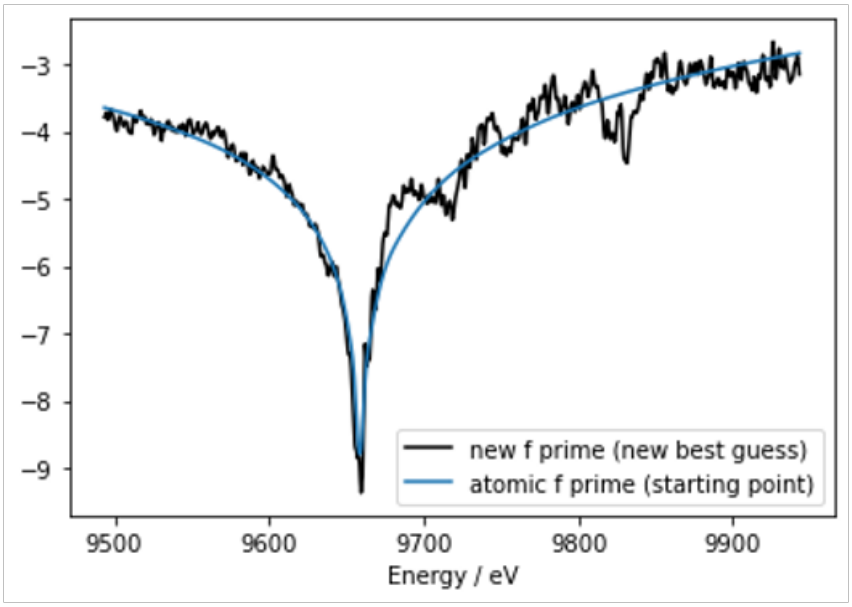

Figure 1.16: The effect of fitting f' according to equation **??** for the experimental raw DAFS data of a $ZnFe_2O_4$ system.The bare-atom lineshape obtained from Cromer-Liberman values is shown for comparison.

```
1     ene = np.array(ene)
2     ene_dash = np.array(ene)
3     f2KK = lambda ene_dash : (f1_minus - dkk.f1) / ((ene_dash
     -1)**2 - (ene[i])**2)
4     f2KK_arr = []
5     f2KK_arr, err = (quad_vec(f2KK,(ene_dash[0]),(ene_dash[-1])
     ))
6     fsecond_KK = (dkk.f2 - ((2*ene/math.pi) * f2KK_arr))
```

Figure 1.17: The Kramers-Krönig transform for the forward transform from f' to f".

and the process can be iterated as many times as desired using the `iterate` button. This is the final result of the fit, once the parameters converge and the model fit matches the experimental data, at which point we have extracted the f" for the given reflection.

There is also an option to take advantage of a convolution in the fit, and one option is by using ATHENA's built in smoothing function to eliminate some noise from the resultant raw DAFS spectrum, as the program automatically saves the lineshape in a format that can be imported readily. I have therefore included a section for importing the resultant smoothed data, which can be treated analogously to above. Alternatively, I have included a convolution using `np.convolve` and this can be

found in the source code, contained within the appendix.

1.3.2 User considerations for DAFS data analysis

The inherent difficulty of DAFS data collection, analysis and extraction mean that, although this chapter presents a bespoke data analysis software, it comes with its limitations and considerations. The end of this chapter aims to present some of these inevitable limitations in certain aspects of the program, that the user may wish to explore to ensure accurate and facile extraction of f".

1. **Sample thickness**

   One experimental parameter to which particular attention must be paid is the thickness of the sample, as this affects the magnitude of the absorption correction. The intensity of diffracted X-rays are attenuated by absorption in the vicinity of the absorption edge. This attenuation is therefore dependent on sample thickness, and is modelled by the absorption correction. It was found after some experimentation revolving around the variation of sample thickness that the optimal thickness was where a sufficient edge jump of classical $\mu(E)$ was achieved, and with a concentration of sample thin enough to achieve a pellet thickness returning a sufficient diffraction peak intensity, but not so thick that the sample was dominated by absorption, so as to minimise the correction applied, as the attenuation can cause discrepancies in the diffraction peak intensity variation and therefore the DAFS if not correctly accounted for.

   To remove a consideration for the user, thickness has been included as a variable parameter in the I_{model} fitting procedure, to ensure that the absorption correction value is correctly applied. In the unlikely event that the user wishes to control this parameter, they can simply constrain it as discussed below in the model considerations section.

2. **Window selection and other user definitions**

   Within the GUI, there are user specified parameters, such as the edge energy or the $Z_{absorber}$. Another optional parameter is the d-spacing window. The user must ensure when selecting this window that the peak is reasonably central,

and there are no extraneous features within the window. This can be done by simply inspecting the region of interest in the generated d-spacing XRD patterns, and making sure there are no nearby peaks or glitches in the window that will have an effect on the calculated background. In the regular `.py` file I have also included a plot for each 20^{th} iteration, to qualitatively inspect the reliability of the fit and subsequent intensity extraction. In any case, the window selected will have an effect on the speed and accuracy of the fit, and so must be given adequate thought by the user.

3. **Diffraction indexation**

   In the indexation procedure, the circular 'rings' generated by powder diffraction are transformed to vertical lines, so that the intensity of the given row can be accurately extracted. To do so, 3 indices are selected by the user, which are used to calculate the centre and radius of a circle. It is important to visually inspect the full list of indices, and select appropriate values that are roughly 1/4, 1/2, and 3/4 of the way up the rows, and that are not more than 5 indices apart from one another. In addition, the calibration for transformation to 2θ, and therefore to d, may not be completely accurate. It is possible to modify this calibration if required, however it is usually close enough that the user can visually inspect the spectrum and select an appropriate d-spacing window for the extraction of the desired peak intensity variation.

4. **Misalignment**

   In the event that the user is facing stubborn adversity when attempting to process their data using this program, this may be resolved by considering interpolation. If the diffraction detector and ionisation chambers are not temporally aligned, then the energy point of diffraction peak intensity may not match the corresponding energy point of absorption. This may result in a poor model fit, or anomalous terms that do not converge to finite values at every energy, or else exist with inaccuracies or artefacts, as a result of this misalignment.

As a result of this, I have included an optional interpolation function to give the user the additional capability of controlling the shifting of the raw DAFS, or the $\mu(E)$. As this is variable on a case-by-case basis, it is likely best to consult the issues section of the *Github* page for the program, and post a specific query for which the issue can be troubleshooted accordingly. However, one common solution is for the user to alter the energy of the DAFS spectrum and $\mu(E)$ such that the inflexion point of the $\mu(E)$ is close to the minimum of the raw DAFS spectrum. In addition, it may also be helpful to ensure the energy of the edge is within reasonable distance from the actual edge energy, as the bare-atom atomic anomalous terms used in the I_{model} fit will be defined based on this edge energy.

5. **Model considerations**

   In the generation of a model, the data analysis program uses the `lmfit` package for the custom curve-fitting procedure. In doing so, we must ensure the generation of a curve that fits to the raw DAFS spectrum with reasonable parameters, such that the subsequent f' and f" calculation generates values with the correct sign and magnitude relative to the bare-atom lineshapes. In order to do so, we can take advantage of the sophisticated `lmfit` package, firstly to gain a better understanding of the initial value fit. Of course, parameters are different on a case-by-case basis, and so plotting this initial value relative to the experimental data and fit, and modifying the individual starting parameters should allow the user to generate an initial guess that is not far from the calculated first guess fit.

   Secondly, it is also possible to constrain certain parameters to ensure that they remain within what the user dictates to be a reasonable range. The figure 1.18 demonstrates an example of fitting the initial value to get an idea of reasonable initial guess parameters, and constraining them to ensure an accurate fit to the experimental data is obtained.

6. **Generating f' and f"**

```
1
2 # Defining the model fit equation
3 def intensity(en, phi, beta, t, exafs, sin_theta, fprime, fsec,
      scale=1, slope=0, offset=0):
4     costerm = (np.cos(phi) + (beta*fprime))
5     sinterm = (np.sin(phi) + (beta*fsec))
6     return scale * (costerm**2 + sinterm**2) * ((1 - e**((-2*
     exafs*t)/sin_theta)) / (2*exafs)) + offset
7
8 # Creating parameters for the model fit
9 imodel = Model(intensity, independent_vars=['en', 'fprime', '
     fsec', 'exafs', 'sin_theta'])
10 params = imodel.make_params(scale=1, offset=0.5, slope=0, beta=
      0.135, phi= 2.258, t = 1)
11
12 # Optionally constrain parameters, for example to force I0 to
     be positive, to fix the value of t, or for phi to exist only
      within a specified range.
13 params['t'].vary = False
14 params['scale'].min = 0
15 params['phi'].max = 15
16 params['phi'].min = 1
17
18 # Calculating the initial value and the fit
19 init_value = imodel.eval(params, en=ene, fprime=f1, fsec=f2,
     exafs = exafs, sin_theta = sin_theta)
20 result = imodel.fit(y_new_shift, params, en=ene, fprime=f1,
     fsec=f2, exafs = exafs, sin_theta = sin_theta)
21
22 # Plotting the experimental data
23 plt.plot(ene, (result.best_fit), '--', label='best fit (lmfit)'
     , color = 'blue')
24 plt.plot(ene, (init_value), '--', label='init value (lmfit)',
     color = 'black')
25 plt.plot(ene, y_new_shift, label='exp. data', color = 'red')
26 plt.legend()
27 plt.show()
```

Figure 1.18: Highlighting the model considerations, including initial value plotting and parameter constraining, in order to ensure the generation of accurate parameters during the fit.

The raw DAFS spectrum is generated as a result of the anomalous scattering that occurs when observing diffraction in the vicinity of the absorption edge. This spectrum is a combination of real and imaginary components, denoted by f' and f'' respectively. It is therefore possible that this spectrum is dominated by one anomalous contribution over the other. The implications of this mean that the parameters generated by the fit may be much more well-suited to calculate an f' over an f'', or vice versa. For this reason, in some contexts it may be advantageous to first calculate f'' over f'. The option to do so has therefore been included in the `.py` file, although this ability is not compatible with the GUI.

In addition, the magnitude and signs of the parameters generated from the fit mean that sometimes the generated anomalous contributions have an inverted shape. In this event, the $\pm$ signs can be adjusted as in eq. **??** and **??** such that the molecular f' or f'' now matches the magnitude and sign of the atomic bare-atom lineshapes.

Applying the above considerations, a user should be able to analyse their DAFS data and return accurate site- or spatially-specific $\mu(E)$ with ease, equipped with only the XRD and XAS data, and just by modifying a few simple experiment-specific parameters: the central absorber and it's edge, the d-spacing of the concerned diffraction peak, the indices for diffraction linearisation and indexation, and the starting point parameters for the I_{model} fit.