Title Page

NOTE: Future changes are marked with “!!” for later editing.

**Table of Contents**

[1. Intro to Spyder 1](#_Toc45354171)

[2. Basic Example 3](#_Toc45354172)

[3. Working with Raw Data Files 14](#_Toc45354173)

[3.1. Searching for Multiple Files 14](#_Toc45354174)

[3.2. Data Import Options 18](#_Toc45354175)

[4. Editing the Program 23](#_Toc45354176)

[4.1. Adding or Editing Characterization Techniques 23](#_Toc45354177)

[4.2. Adding Functions/Calculations 31](#_Toc45354178)

[4.3. Changing Excel Formatting 32](#_Toc45354179)

[5. Peak fitting 33](#_Toc45354180)

[5.1. Some text explaining peak fitting 33](#_Toc45354181)

[5.2. Examples 34](#_Toc45354182)

[5.2.1. Basic Example 34](#_Toc45354183)

[5.2.2. Intermediate Example 51](#_Toc45354184)

[5.2.3. Multiples peaks with different line shapes and using BIC to find best model 51](#_Toc45354185)

[5.2.4. Finding and fitting residual peaks 55](#_Toc45354186)

[5.2.5. Using a manual background and only fitting part of a spectrum 55](#_Toc45354187)

[5.3. Things to consider 55](#_Toc45354188)

[5.3.1. Minimization method 55](#_Toc45354189)

[5.3.2. Picking a maximum Sigma 59](#_Toc45354190)

[5.3.3. Getting additional information 61](#_Toc45354191)

[6. Plotting 61](#_Toc45354192)

# Intro to Spyder

This will be a quick intro to using the program Spyder to familiarize the user. Skip if you are not using Spyder to run these programs, or if you are already familiar with Spyder. The only necessary bits of information are how to start and stop programs in Spyder.

Spyder is one tool used to edit and run python scripts. It does not seem to be very popular compared to other editors from what I have read, but I like using it for one reason: the variable explorer (upper right window in Figure 1). The variable explorer keeps track of all variables, and makes it very easy to see what’s going on within the program; you can just double click on any of the variables to see what their value was after running the program.

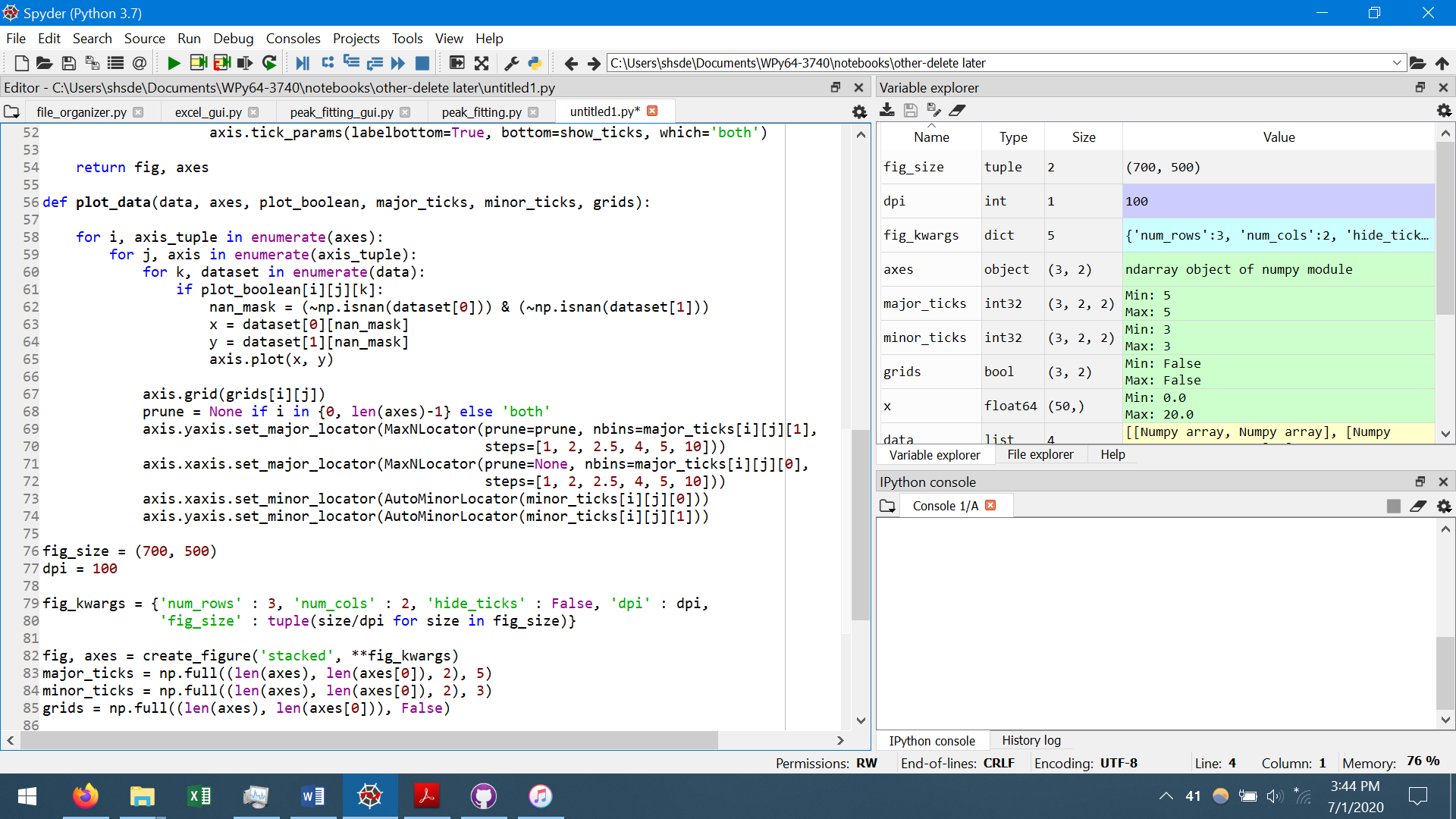


Figure : Layout of Spyder; note: the layout of Spyder can be changed, so it may not look like this for you.

To start a program, all you have to do is press the green arrow in the upper left of Figure 1, under the word “Run”. Alternative, you can press “Run” and then “Run Program”. To stop a program while it is running, all you have to do is press the red rectangle in the lower right of Figure 1 (enlarged picture in Figure 2a). Sometimes, especially when dealing with graphical user interfaces (GUIs), there can be problems when stopping the program early. A common issue you may see is that the GUI will not close after stopping the program. To deal with this issue, or others you may come across, all you have to do is restart the kernel (shown in Figure 2b). Restarting the kernel will clear all variables, get rid of all GUIs, and kind of restart python.

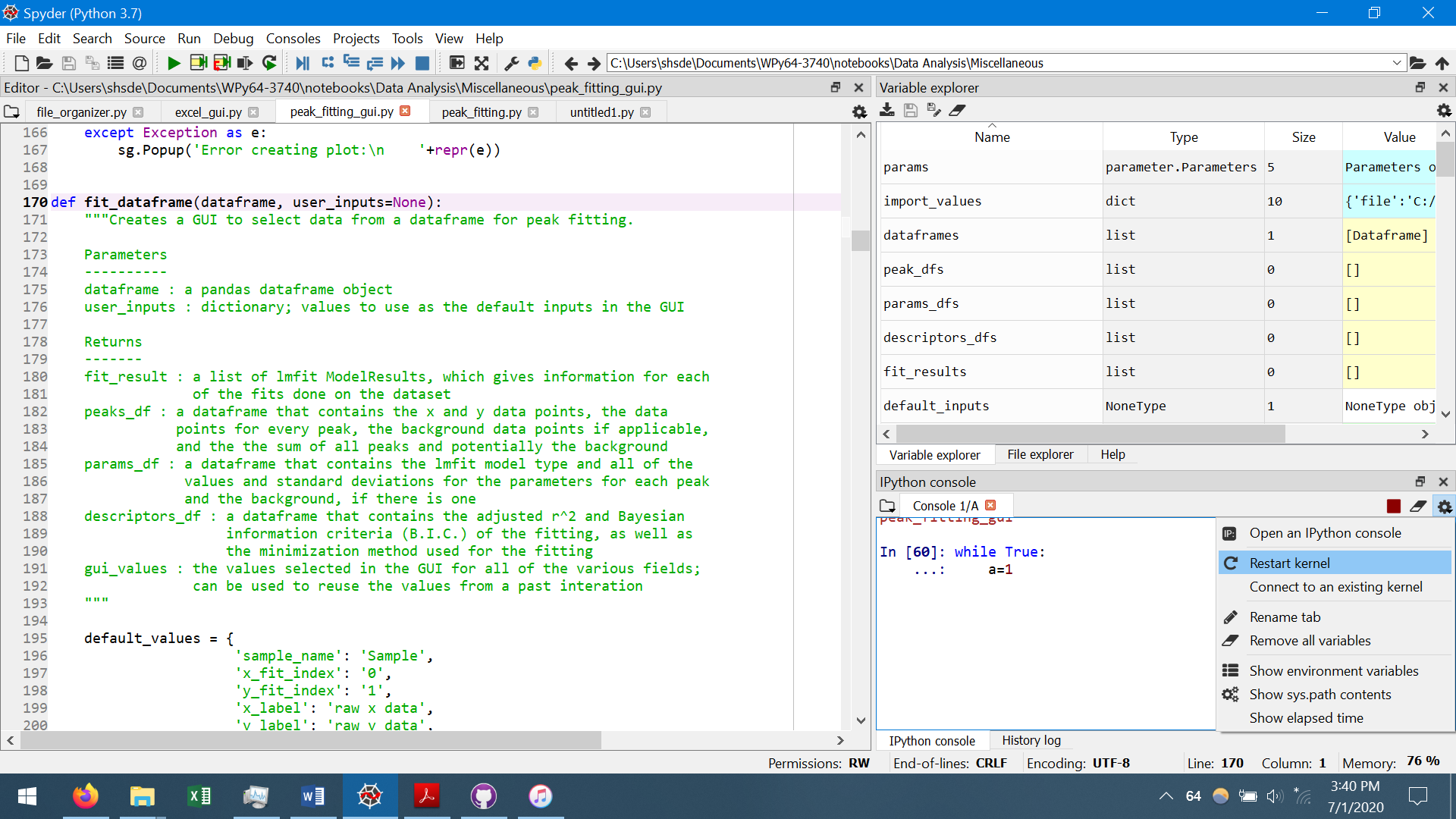
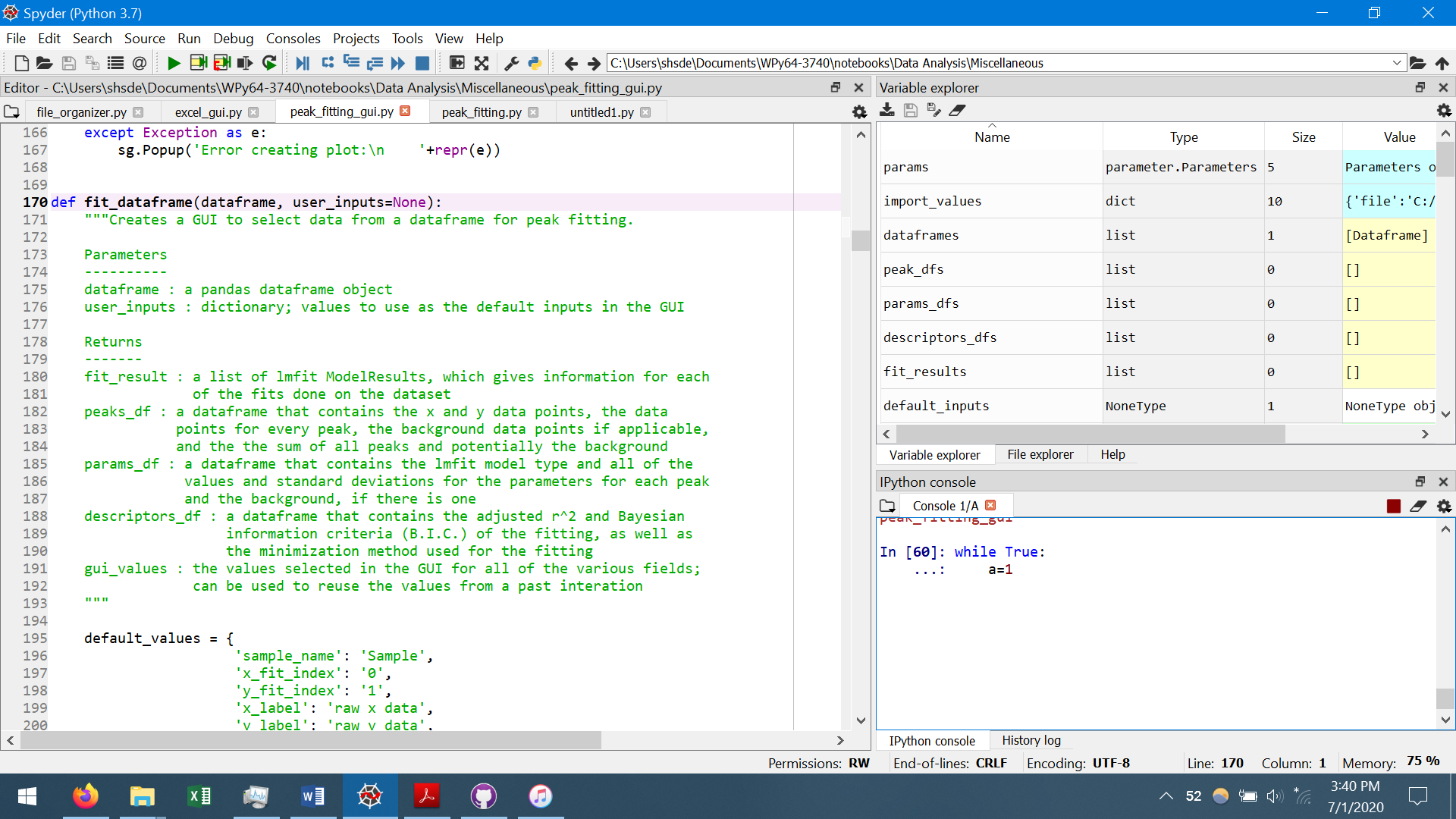
**

Figure : (a) Red rectangle to stop a program; (b) how to restart the kernel.

The bottom right of Figure 1 is the output of the current program. If something goes wrong with the program, text may pop up in that section to let you know what went wrong. Most of the error output is useless for beginners, but the last line can be helpful when trying to figure out what to Google search to figure out the error (Figure 3). If an error occurs, usually just rerunning the program will work, but I just wanted to give you an idea of what is being printed out.

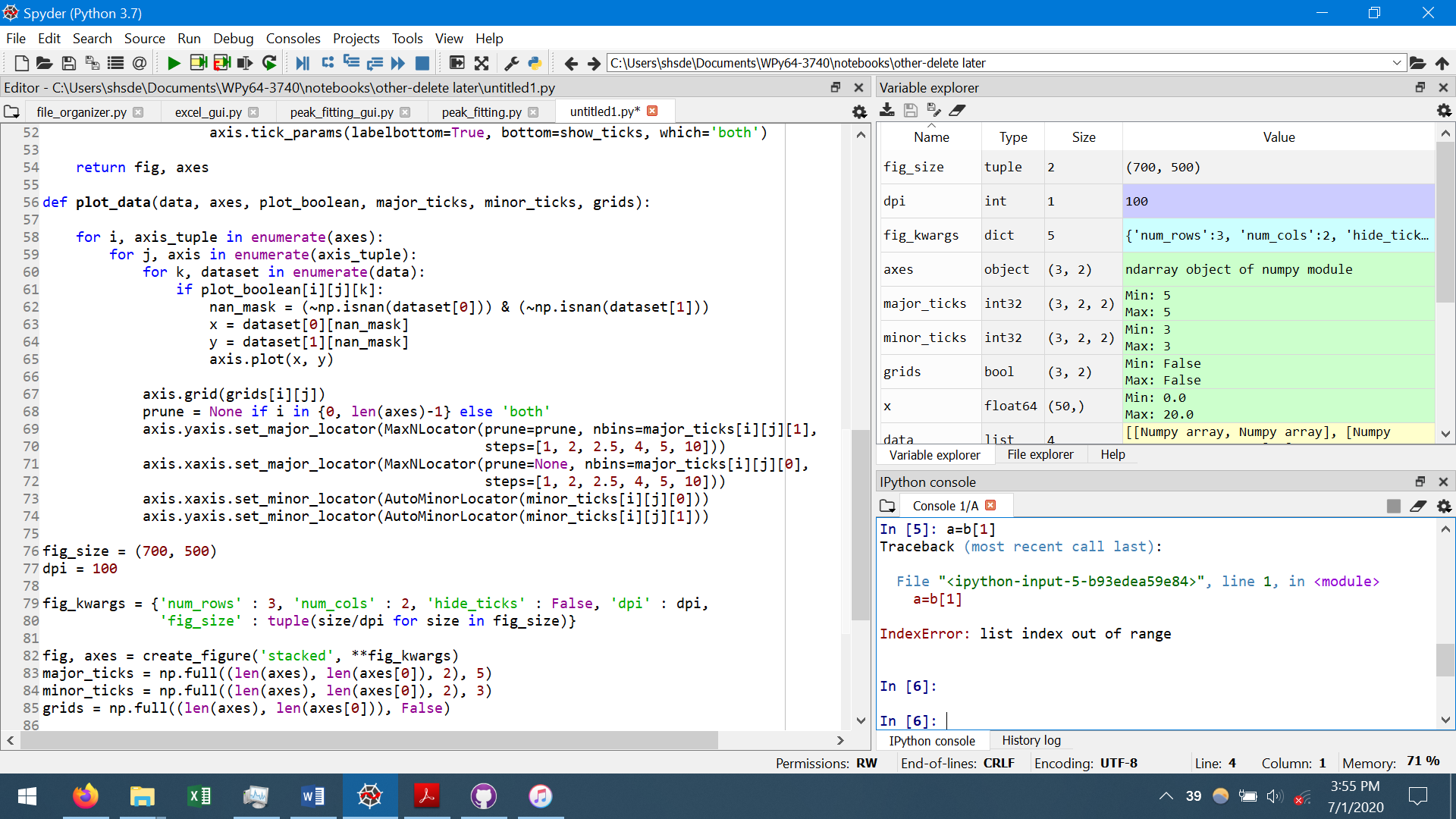


Figure : Output when there is an error

Another helpful piece of information for Spyder is that right clicking a function or class and then pressing “Go to definition” will usually go to the part of the code where the function or class is defined (Figure 4). This makes it easier to look up the documentation for functions and classes.

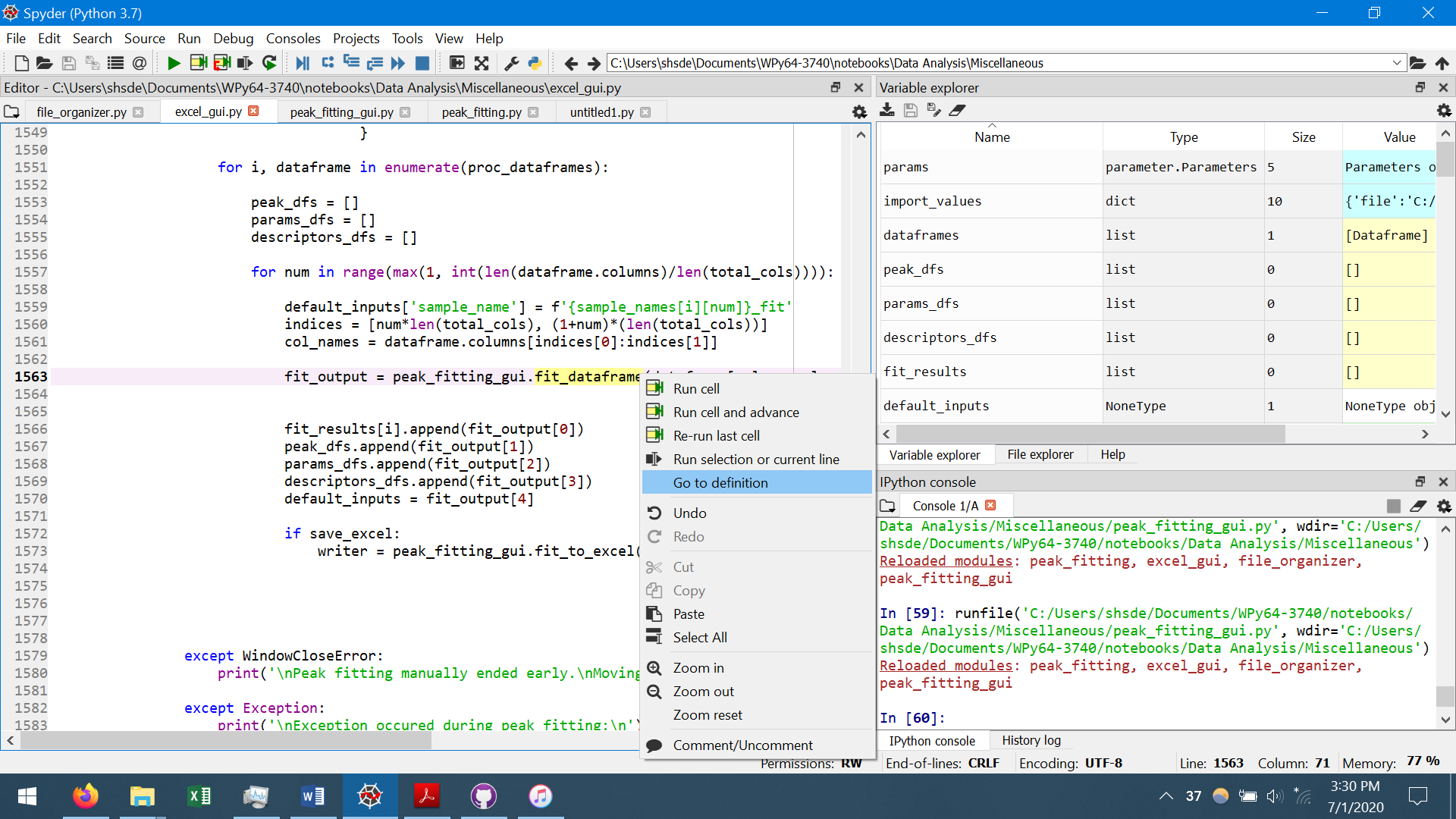
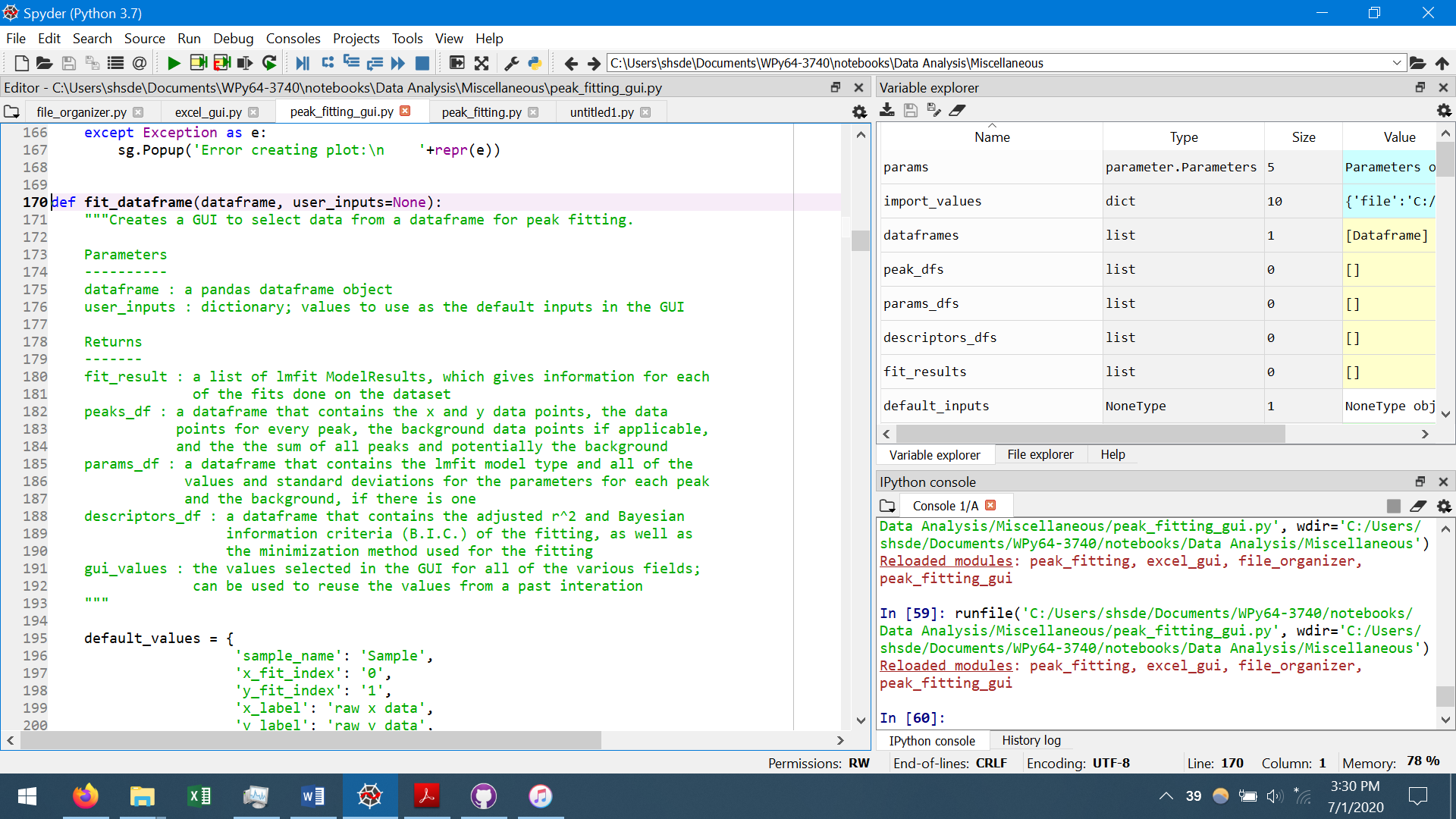
 

Figure : Looking up the definition of a function by using “Go to definition”.

# Basic Example

My intent with all of my GUIs are that they are easy to understand and that all of the options are apparent simply by reading. If I did my job correctly, you won’t need to read most of this guide, but I wanted to give in depth explanations for each window because I know things are probably not as clear as I wished. My best suggestion would be to just play around with the various options in my program and see what happens and then read in this guide about anything that is unclear. Note: in most of the GUIs, either clicking “Next” or “Submit” will move onto the next window, or you can simply hit the Enter key.

This section will show a basic example of using the program to read multiple raw data files, process and write the data to an Excel file, and move the raw data files to a different folder. Minimal explanation will be given in this section; further explanations are given for searching and importing raw data (Section 3), adding/editing characterization techniques and editing the Excel formatting (Section 4), peak fitting (Section 5), and generating high quality plots (Section 6). NOTE: All sections throughout this manual use raw data that is created by running the generate\_raw\_data.py program. This program creates a Raw Data folder containing example data for various characterization techniques meant to showcase the uses of this program. It should be stressed that the raw data and the sample names generated by the generate\_raw\_data.py program are not associated, meaning the sample names were just randomly selected.

When running excel\_gui.py, the first window to pop up will look like Figure 5. The fields in Figure 5a are:

1. Multiple Files: if this option is selected, the program will use multiple files which are found by using a keyword search (discussed in detail in Section 3.1)
2. Use previous search: not available the first time the program is run. After searching for multiple files, the program will save all of the file locations to a file called “previous\_search.json” in the same folder as where the program is located. If those files are to be used again, this option can be selected to avoid having to search for the same files again.
3. Single File: will use a single file as the data source
4. Fit Peaks: will open a GUI for peak fitting the data, discussed in Section 5; will save the results of peak fitting if Save Excel File is also selected.
5. Plot in Python: opens a GUI for plotting the data to make nice looking plots (not currently implemented)
6. Move File(s): will move copies of all of the files to the desired folder. It first copies the files before moving, so the files are also kept in their original folder.
7. Save Excel File: saves the data, any applied calculations, and any peak fitting results to Excel; can either select “Append to existing file” (default) to add the sheets generated by the programs to an existing Excel file (or create a new file if no file exists with the selected filename) or “Create new file” which will either create a new file or overwrite an existing file with the same name.
8. Plot data in Excel: will create simple plots of the data in Excel
9. Plot fit results in Excel: will create simple plots of the peak fitting results in Excel
10. The filename for the output Excel file.

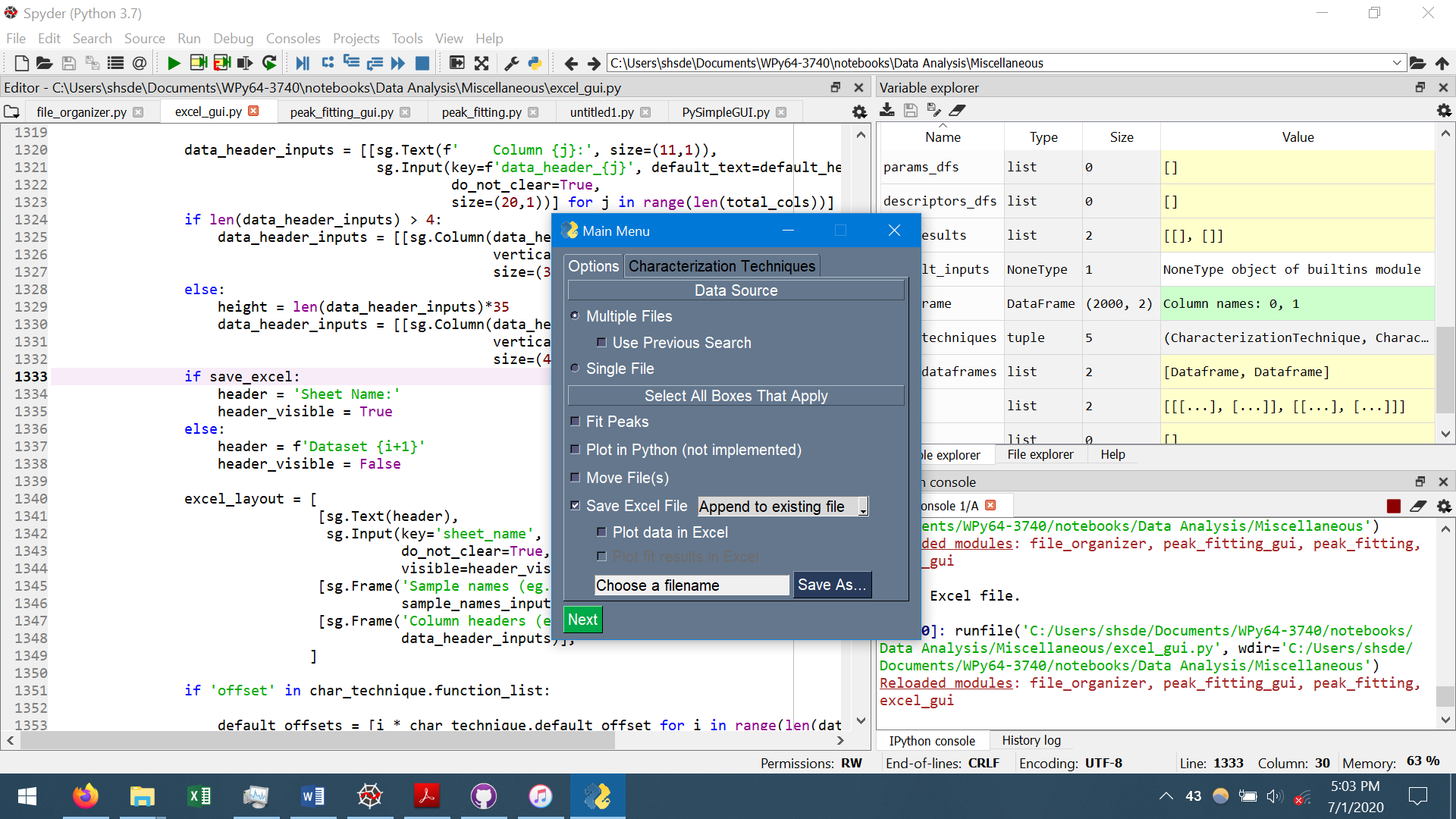
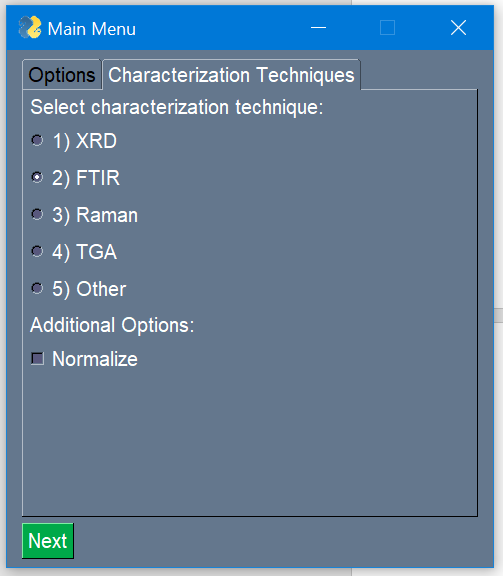
 

Figure : First window; (a) options for selecting files and processing data; (b) available characterization techniques.

The fields in b are:

1. All available characterization techniques; the fields available here depend on the CharacterizationTechnique objects input into the launch\_main\_gui function (Section 4.1).
2. Normalize: will normalize the raw data between 0 and 1; only available if the selected characterization technique has “normalize” in its function list (Section 4.1).

There are two reasons for selecting characterization techniques within the program. First, the program adds calculations with the raw data when putting it into Excel, such as applying offsets for plotting, normalizing the data, or calculating the derivative of the data (a full list of functions that can be computed is given in Section 4.1). The second reason for selecting characterization techniques is that each technique has a default set of options for reading the data files and the file type to use for file searching. That way, the program can be run through quickly if the default settings were set correctly. See Section 4.1 for how to set these default fields.

For this basic tutorial, select “Multiple Files” as the data source, and select “Move File(s)”, “Save Excel File (Create new file)”, and “Plot data in Excel”. The filename for the Excel file will be “output”. For the characterization technique, select “XRD”, and then hit “Next”.

The next window is the file searching window shown in . The meaning of the keywords will be explained in Section 3.1. For now, select the Raw Data folder that was generated by the generate\_raw\_data.py program as the topmost folder for searching, enter (without quotation marks) “2” for the number of main keywords and “3” for the number of secondary keywords. The file extension was automatically set to “csv” for the XRD characterization technique, and can be left alone. Pressing the “Help” button will give a basic example of choosing keywords to find certain files. Now, press “Next”.

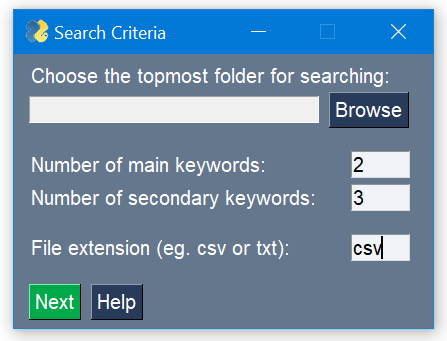


Figure : The first file searching window.

The second file searching window is shown in Figure 7. For this example, enter “700” and “800” as the main keywords, “0W”, “1W”, and “2W” as the secondary keywords, and then press “Submit”. In this example, the program will search for files with the following names: 700+0W, 700+1W, 700+2W, 800+0W, 800+1W, 800+2W, where “+” just denotes a pairing of keywords. More discussion of search keywords is given in Section 3.1.

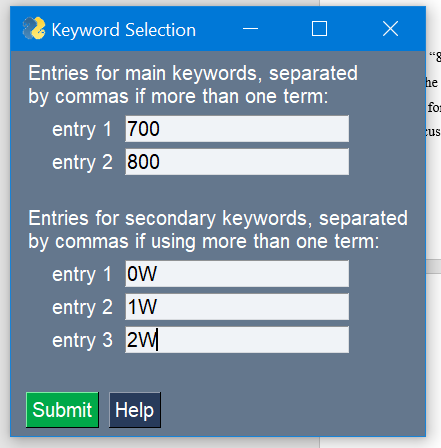


Figure : Second window for file searching.

If the search terms only produce one file that matches the keyword pairing, then that file will automatically be selected; otherwise, will appear to allow selection of the correct file. For each file in , the listed file contains both the file name, and the names of all folders underneath the selected topmost folder from , in order to tell where the file is located. For example, the first listed file is named “Fe-0W-700.csv” and it is located in the “Fe” folder, which itself located in the “XRD” folder within the “Raw Data” folder selected as the topmost folder. For this example, select only the files with “Fe” in the file name for each of the search terms (although both Fe and Ti files are actually the same data). Note: it is possible to use better search terms so that the program would only find one file per search term, to be discussed in Section 3.1.

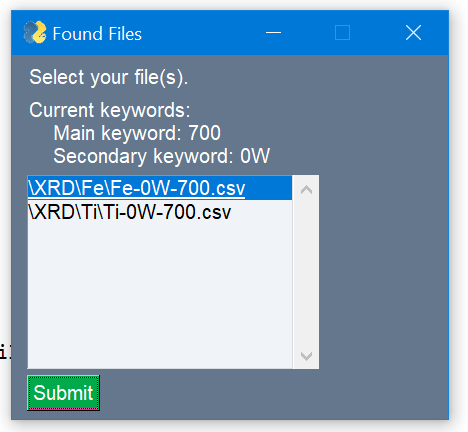


Figure : Window that appears if more than one file is found for a search term.

The next window is the Data Import window in Figure 9. If “Single File” was selected in Figure 5, then a window very similar to Figure 5 would be shown, except there would also be a field to select the data file (see Figure 16). The relevant fields for the XRD characterization technique are automatically filled in, so little needs to be done for this window. For this example, the only change is to set the “Number of empty columns to put between datasets” to 1. Hitting the “Test Import” button will show an example of how the columns and rows will be imported from the raw data files using the current values in the window. Note that only numbers should be in the columns designated as the x- and y-columns, otherwise issues may arise during calculations with the data. More details on the meanings of all the fields in the Data Import window will be discussed in Section 3.2. Press “Next”.

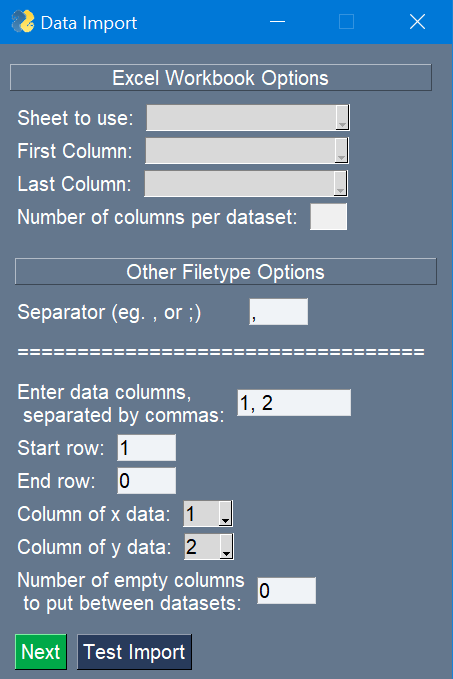


Figure : Window for data importing options.

The next window that will show is Figure 10, which determines various labels in the created Excel file. When first appearing, Figure 10 will have several fields already filled in, using the defaults for the selected characterization technique. Figure 10 will appear once for every main keyword used in the file search (corresponding to the number of sheets in the Excel file). For example, if there were two main keywords (two Excel sheets to create), Figure 10 would pop up twice before a different window is shown. The fields in the Formatting tab in Figure 10a are:

1. Sheet Name: the name of the sheet in the generated Excel file. In this case, “700\u00B0C” was input, which is 700, the Unicode for the degree symbol \u00B0, and C, resulting in a sheet name of “700°C” in the Excel file.
2. Sample names: the label for the data from each raw data file; the number of samples per sheet is equal to the number of secondary keywords used in the file search (3 in this example).
3. Column headers: the headers for each column of data. The column headers will be put with the data for each Sample name (see Figure 11). In this example, the column headers are “2\u00B8 (u\00B0)” which is 2θ (°) after converting from Unicode, Intensity (Counts), Offset Intensity (a.u.), and a blank column. Although there were only two data columns imported from the raw data files (data columns in Figure 9), there are four total columns in Figure 10. This is because the third column is added by the program to contain a calculation on the raw data (in the case of XRD, this calculation is simply an offset to the intensity data column), and the fourth blank column corresponds to the 1 blank column specified when importing the data in the previous paragraph, (although the column header itself does not have to be blank). Also note that the column numbering starts at 0 rather than 1; this is simply to be consistent with numbering throughout the program, in which rows and columns start numbering at 0 just to ease their use in the actual programming (Python starts counting at 0 rather than 1).
4. Offsets: only available if “offset” or “normalize” is in the characterization technique’s function list (Section 4.1). This value will be added to the y-values of the raw data to offset the data for plotting.

Another way to explain these fields is simply by looking at Figure 11, which shows the Excel sheet created by the entries in Figure 10. The first row contains the sample names, the second row contains the column headers repeated for each sample, and the rest of the rows are the data. As already discussed, Unicode can be input into the fields to make various symbols. Pressing the “Unicode Help” button will give the Unicode for the common symbols of °, 2, and θ.

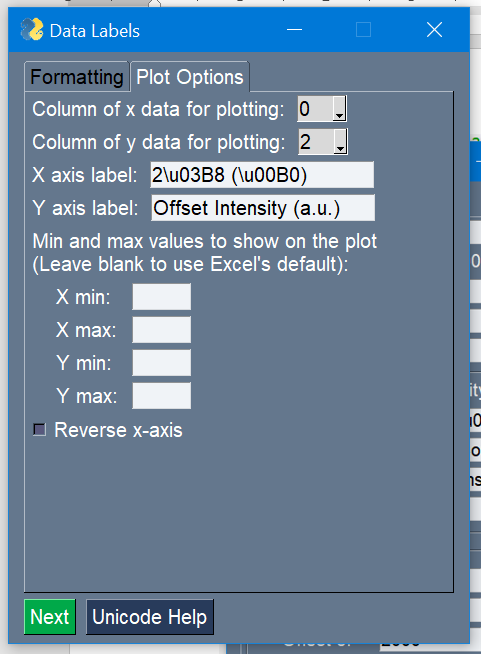
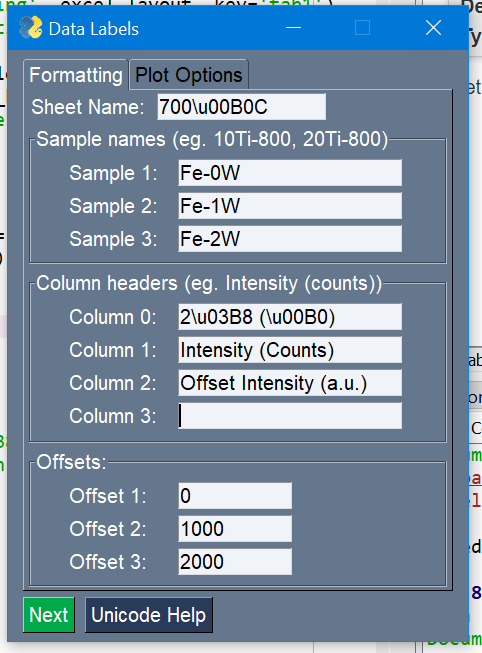


Figure : Various inputs that determine (a) labeling in the output Excel file and (b) options for the plot made in Excel.

The fields in the Plot Options tab in b, which is only shown if “Plot in Python” or “Plot in Excel” are selected in , are:

1. Column of x data for plotting: which column number to plot on the x-axis; the numbers correspond to the columns in a, so in this case column 0 is the 2θ values.
2. Column of y data for plotting: which column number to plot on the y-axis; in this example, column 2 (the offset intensity values) will be plotted on the y-axis.
3. X axis label and Y axis label: the labels that will be put onto the plot for the x and y axes, respectively.
4. Min and max values for the x and y axes: self-explanatory; usually best to leave blank so that Excel uses its default values for the axes bounds.
5. Reverse x-axis: if checked, the x-axis will be flipped so that it goes from max to min; for characterization techniques such as FTIR. Note: x\_min should still be less than x\_max if using “reverse x-axis”, due to how Excel handles its bounds.

After filling out the Data Labels window to look like Figure 10, press “Next”, fill out the window again for the second set of data, and then press “Next” again.

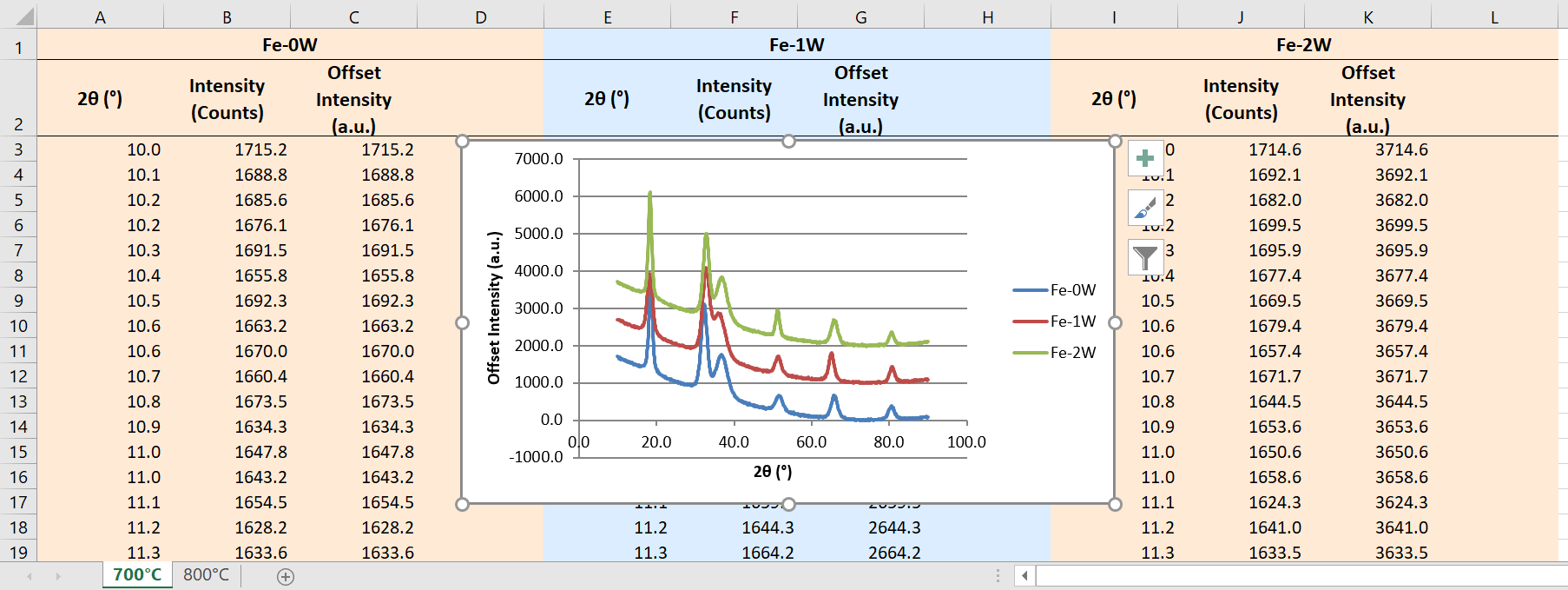


Figure : The Excel file created by the entries given in .

If the option to move raw data files was selected in Figure 5a, then the window in Figure 12 is shown, and you simply have to select or create a folder to move the files to. Each grouping of files can be moved into separate folders, or “All Same Folder” can be selected, which will make all files be moved to the folder specified for Dataset 1. For this example, the folder “700” is created for Dataset 1, and the folder “800” is created for Dataset 2. Hitting “Submit” will be the last input required by the user. The program will then save the Excel file (to be discussed) and move the files to the specified folders. Figure 12b shows the files for Dataset 1 after they have been moved by the program. Note: the raw data files are not actually moved by the program. Rather, they are copied, and then their copies are moved to the desired folders, thus preserving the file location of the original files. If a file already exists in the output folder, then the file name is simply appended with “\_COPY\_1” (or “\_COPY\_2”, if “COPY\_1” already exists, etc).

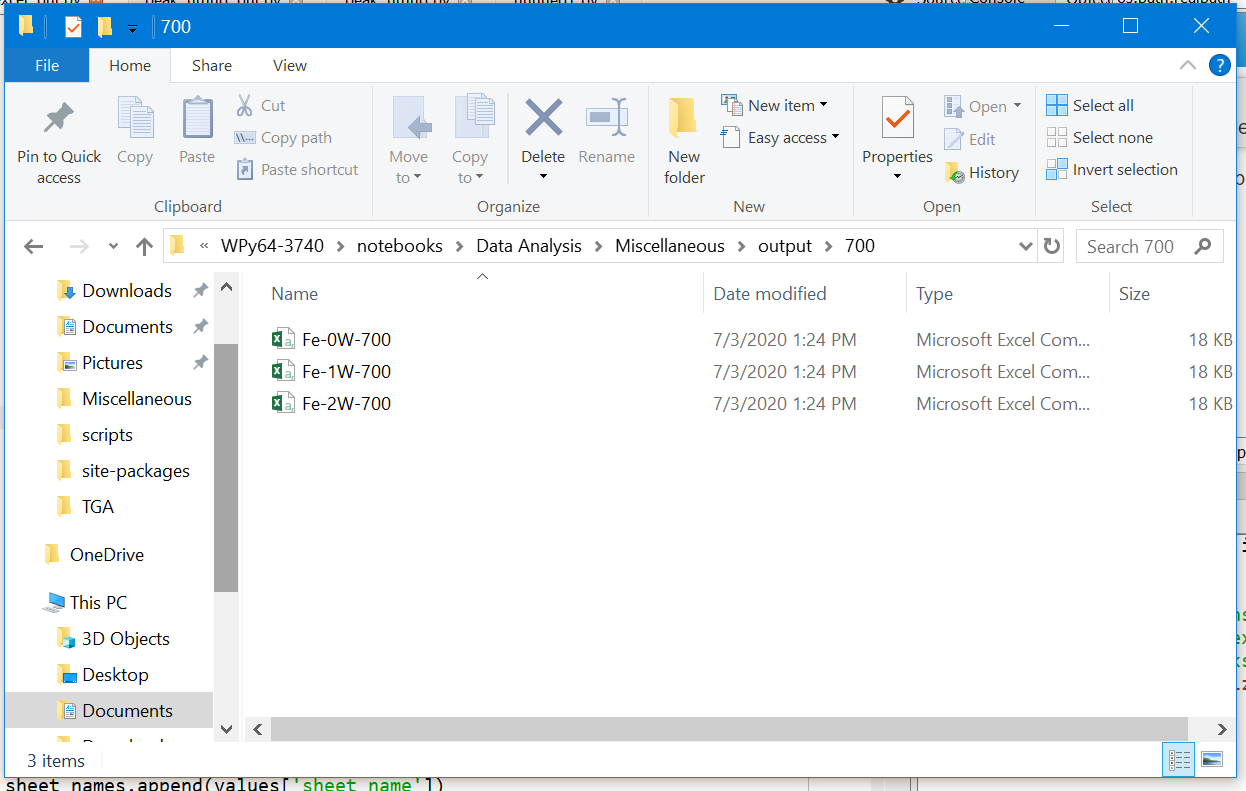
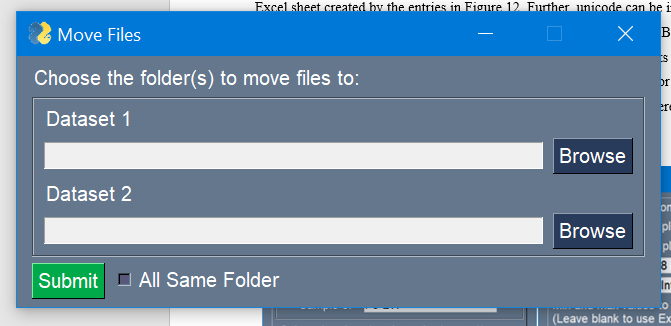


Figure : (a) Window to select the folders to move files to; (b) the moved files from Dataset 1.

If everything went correctly, then the program will read all of the data files, process the data, save everything to an Excel file, move the raw data files, and print out the confirmation message in Figure 13a. If trying to overwrite an existing Excel file (by selecting “Create new file” in Figure 5) while the file is open, then the program will print the message in Figure 13b. The program will try ten times to save the file, once every six seconds. If it has still been unable to save the file, then the program will simply stop without saving the Excel file. If “Append to existing file” was selected in Figure 5, then the program will first save a temporary Excel file called “temporary\_file\_to\_be\_deleted.xlsx” in the current working directory, copy all the sheets from the temporary file to the specified Excel file, delete the temporary file, and output the messages in Figure 13c. Appending to an existing file can be done regardless of whether the file is open or not.

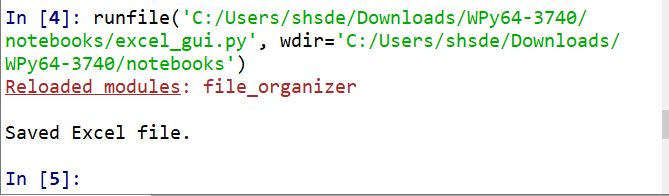
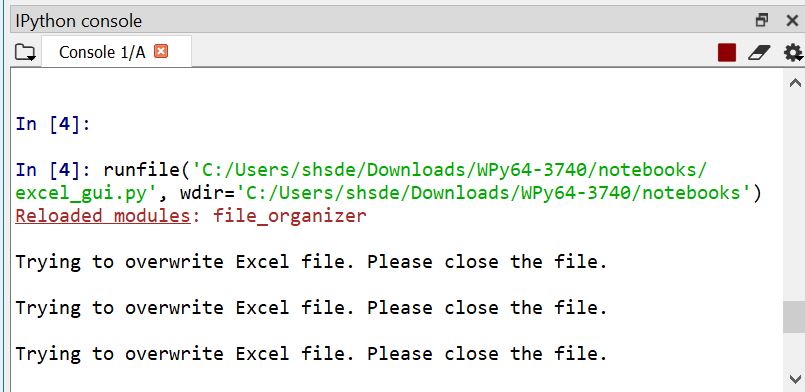
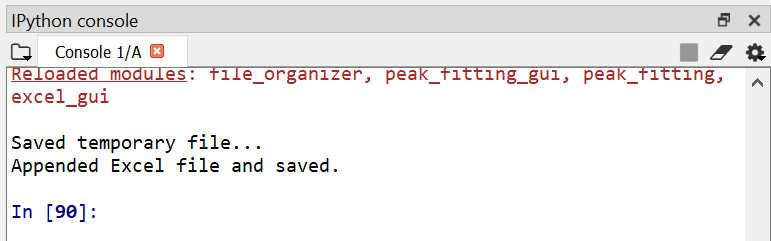
1. 
2. 
3. 

Figure : (a) Output when a new Excel file is saved. (b) Output when trying to overwrite an Excel file that is currently open. (c) Output when appending to an existing Excel file.

# Working with Raw Data Files

# Searching for Multiple Files

The way the file searching works is that it searches all files and folders underneath a topmost folder, which is selected in Figure 6. Say your folder structure was as follows:

1) Documents (folder)

* 1. Papers (folder)
     1. paper\_for\_journal\_a.docx
     2. paper\_for\_journal\_b.docx
     3. journal\_count.csv
  2. Experiments (folder)
     1. TGA (folder)
        1. Monday (folder)
           1. Test1.txt
           2. Test2.txt
        2. Tuesday (folder)
           1. Test3.txt
           2. Test4.txt
     2. XRD (folder)
        1. Test1.csv
        2. Test2.csv

If “Documents” was selected as the topmost folder, then any of files could be found, whereas if “Monday” was selected as the topmost folder, then only Test3.txt and Test4.txt could be found. It is best to make the topmost folder the folder which is ensured to encompass any data that you would want in your search. For example, if you wanted to ensure all TGA data was included in the file search, then the “TGA” folder would be the best folder to select as the topmost folder.

The keywords in the file searching work as follows: every main keyword corresponds to one grouping of data, and the number of secondary keywords dictates how many raw data files is associated with each set of data. If outputting to Excel, this would correspond to one sheet per main keyword, with the number of data entries per sheet equal to the number of secondary keywords. Likewise, if plotting, each main keyword would be associated with one plot, and the number of series on the plot would equal the number of secondary keywords.

While searching for files, each main keyword is paired with one secondary keyword at a time. For example, suppose that there were two main keywords (Ti and Cu) and three secondary keywords (700, 800, 900). Then the keyword pairings, and likewise the file searches, would look like this (the “+” just denotes a pairing, it is not actually put into the search term):

1. Ti+700
2. Ti+800
3. Ti+900
4. Cu+700
5. Cu+800
6. Cu+900

The resulting Excel file would have two sheets: one with the three Ti samples and one with the three Cu samples.

As another example, suppose those same Ti and Cu samples had data for both Ar and Ne processing atmospheres, and all of the data was desired. When inputting keywords, multiple terms can be combined simply by adding a comma between the terms. In this case, one way to do this search would be to have four main keywords (“Ti, Ar”, “Cu, Ar”, “Ti, Ne”, “Cu, Ne”) and three secondary keywords (700, 800, 900), giving the following search terms

1. Ti+Ar+700
2. Ti+Ar+800
3. Ti+Ar+900
4. Cu+Ar+700
5. Cu+Ar+800
6. Cu+Ar+900
7. Ti+Ne+700
8. Ti+Ne+800
9. Ti+Ne+900
10. Cu+Ne+700
11. Cu+Ne+800
12. Cu+Ne+900

Note that the actual name of the file does not have to be in the order listed above, since every permutation of the keywords is searched for (so if the keyword pairing was Ti+Ar+700, it would search for Ti+Ar+700, Ti+700+Ar, 700+Ti+Ar, 700+Ar+Ti, Ar+Ti+700, and Ar+700+Ar). Likewise, the file search looks for the keywords within the filename, but they do not have to match exactly or be the correct case. For example, if the search term was Ti+700+Ar, and the actual filename was **ti**tanium\_**700**0\_**AR**GON, then the file would be selected because it contains the keywords (bolded).

If no file exists that matches the search terms, then the program will search for files that match just the main keyword terms and just the secondary keyword terms. For example, if Ti+Ar+700 didn’t give any files, then the program would search for any files with Ti+Ar or 700. If still no files were found, then the program would search for any of the keywords, so Ti, Ar, or 700. If still no files were found, the program simply searches for all files with the specified file extension, and if no files were still found, the program will simply find all files within the specified folder. Further, if you left both the main keyword and the secondary keyword blank in , then the program would find all files with the given file extension.

Going back to the search used in Section 2, the original keywords were “700” and “800” as main keywords, and “0W”, “1W”, and “2W” as the secondary keywords. These search terms produced two possible files that matched (Figure 8). However, by adding additional terms to the main keyword as discussed above, then the search can be refined so that only one file is found.



Figure : Searching with more than one term per keyword.

After all of the files are selected, then a file called “previous\_search.json” will be saved in the same folder as the excel\_gui program, and the option to select “Use previous search” will be available in Figure 5 when rerunning the program. A json file is somewhat similar to a text file, but it stores data differently; the data structure that contains all of the file locations is complicated and would have been difficult to write to and read from a text file, but with json the data structure can be written to and read from directly, just FYI.

# Data Import Options

For all types of data files, the bottom of (the part below “====”) will need to be filled in. The meanings of the fields are as follows:

1. Data columns: which columns in the raw data file to import
2. Start row: the first row to use from the raw data file
3. End row: the last row to use from the raw data file
4. Column of x data: which column (of the column numbers input in the “Data columns” field) to use as the x variable, or the independent variable. See Section 4.1 for an explanation of how the x variable is used by the program
5. Column of y data: which column to use as the y variable, or the dependent variable. See Section 4.1 for an explanation of how the y variable is used by the program.
6. Number of empty data columns to put between datasets: will insert this many columns between each set of data in the Excel file; can be used to space out the Excel file so it is not as cramped or to provide additional columns for calculations that are not done by this program.

shows examples of different start rows, end rows, data columns, and separators to help show the meanings of these terms. In python, counting starts at 0, so if you wished to use the first row, for example, you would put 0 as the start row. For the end row, it is counting up from the bottom, so the bottom most row is row 0, the row above it is row 1, etc.

C:\Users\shsde\AppData\Local\Microsoft\Windows\INetCache\Content.Word\a.tif

Figure : Examples of inputs for row start, row end, data columns, and separator.

For files types other than xlsx files (for example, csv or txt files), the only other unique item that needs filled out is the separator. Common separators for files are commas (,), semicolons (;), and tabs (\t). Regular expressions can also be used as separators (see <https://docs.python.org/3/howto/regex.html>).

For importing data from existing xlsx files (Excel files), there are additional options to choose. The assumption when importing data from an Excel file is that the data has a common, repeating layout. The options are as follows:

1. Sheet to use: which sheet in the Excel file to use; if you want to use more than one sheet from the same Excel file, you can select “Multiple Files” in as the data source, do the file search in such a way that the same Excel file is found multiple times, and then select a different sheet each time the Data Import window is shown
2. First Column: the first column in the sheet to use
3. Last Column: the last column in the sheet to use
4. Number of columns per dataset: how many columns are associated with one set of data in the selected sheet; the total number of sets of data that will be extracted from the Excel file will be equal to (Last Column +1 – First Column) / Number of columns per dataset. For example, if the First Column is column 0, the Last Column is column 11, and there are 4 columns per dataset, then the total number of sets of data is (11 + 1 – 0)/4 = 3.

It is recommended to start from the top of the Data Import window, and work down because various fields are filled out automatically when selecting the sheet to use, number of columns per dataset, etc. In the following example, the file “output.xlsx” created in Section 2 is used as the input file. When first selecting the file, the number of columns per dataset will be set equal to the total number of columns in the sheet (so that there is only one set of data). For this file, there were actually only 4 columns per dataset (see or ), so that needs updated. Likewise, the first two rows in the Excel file contain the data headers, so the start row needs to be changed to 2, resulting in the filled out window in . (Recall that only the first and second columns contained the raw data, so actually should have only 0, 1 as the data columns, although it will not actually affect any calculations as long as the x and y data columns are set correctly). Clicking “Next” on and filling out the rest of the windows again would give an Excel file containing the three sets of data that were originally on the 700°C sheet in the “output.xlsx” file.

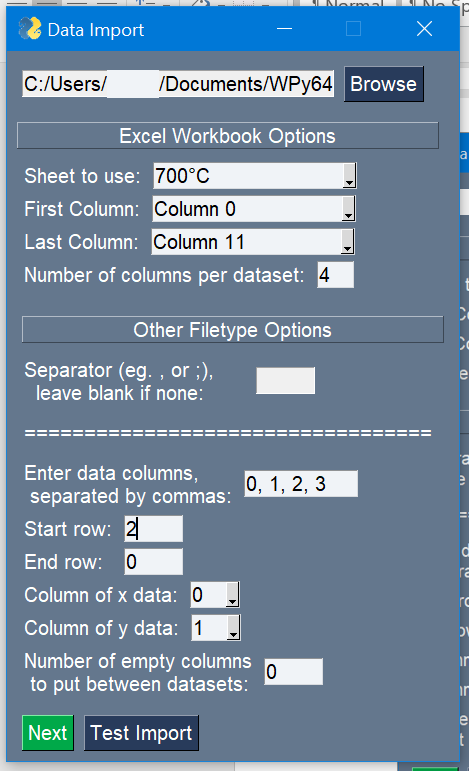


Figure : Import options when selecting the “output.xlsx” file created in Section 2.

At any time, the “Test Import” button can be pressed to look at how the imported data will look (). When the data source is an xlsx file, two windows will show up, with the first window corresponding to the total set of data starting at the first column and ending at the last column, and the second window corresponding to the first set of data. Any errors that would have occurred during data importing will simply give a pop-up warning window when using “Test Import”, rather than causing any issues, so it is best to use “Test Import” if you are unsure if the import options are correct. Further, note that only numbers should be in the columns designated as the x- and y-columns, otherwise issues may arise during calculations with the data.

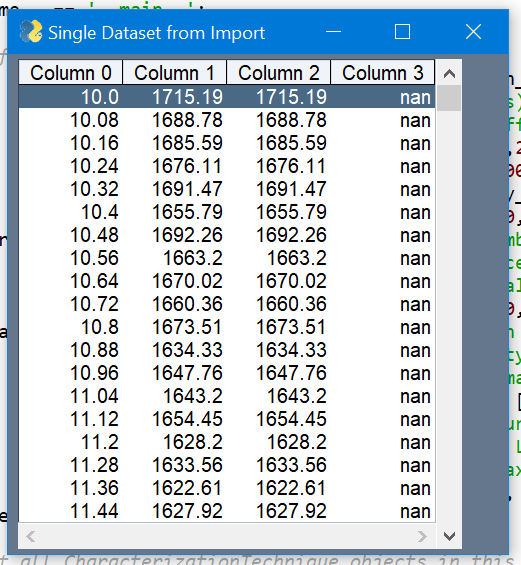
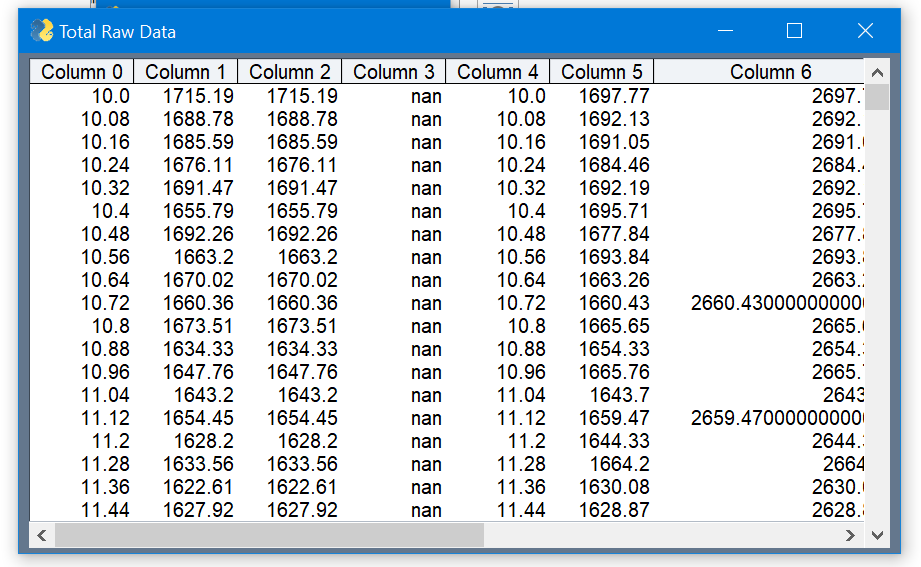


Figure : (a) the total raw data and (b) one dataset that is shown when pressing “Test Import” with the inputs in Figure 16. “nan” just denotes nothing is in that cell. !!Change (b) to reflect the new code that will show all individual datasets.

# Editing the Program !!Change this section later once the function use has been implemented !!Further, make the excel writer layouts an attribute of CharacterizationTechnique objects.

# Adding or Editing Characterization Techniques

Adding a characterization technique so that it shows up in Figure 5 is quite easy. All you have to do is first create a CharacterizationTechnique object (see Figure 18) and then put the corresponding variable into the char\_techniques tuple (could also be a list) shown in Figure 18. The char\_techniques tuple is then input into the function “launch\_main\_gui”, which handles all of the processes in this guide. The window in Figure 5 will automatically take all of the characterization techniques in the char\_techniques variable and create an option to select them. To find the part of the code where Figure 18 is located, all you have to do is locate the line that says “if \_\_name\_\_ == ‘\_\_main\_\_’:”, and the creation of characterization techniques will be directly below it. Adding characterization techniques will not affect what Excel calculations are available since those need to be added in separate places, but what it does do is allow you to input default settings for reading raw data files for those techniques, as discussed for Figure 9.

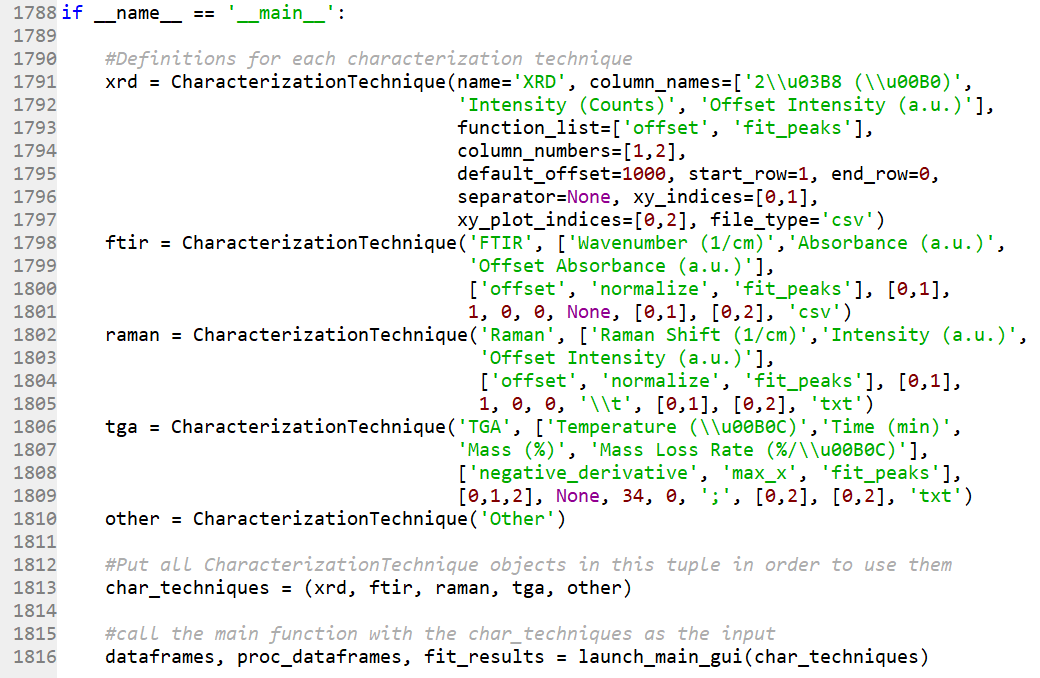


Figure : Picture showing the creation of characterization techniques.

There are a couple of ways to create a CharacterizationTechnique object (abbreviated as CT object from now on), as shown in Figure 18. When creating a new CT object, you can either put “variable\_name=value” for each variable, as shown for xrd, or you can simply put the values in the correct order, as shown for ftir, raman, and tga. You can also just put the name, as done for ‘other’, and the default values for the CT will be used instead. To see the default values for a CT object, just look near the beginning of excel\_gui.py, where the CharacterizationTechnique class is defined. Note: if you are inputting Unicode for the column names while creating a CT object, you must use two ‘\’ marks rather than one, as shown in Figure 18 for xrd.

The input parameters for a CharacterizationTechnique object ares as follows:

1. name: the string representation of the technique that will be shown in Figure 5a.
2. column\_names: a list of strings to use as default column headers. When adding column names for calculations (current calculations are “offset”, “derivative”, “negative\_derivative”, and “fractional\_change”), be sure that the column headers are in the correct order if doing multiple calculations. Calculations are added in the following order: “offset”, “derivative”, “negative\_derivative”, “fractional\_change”.

For example, if using both “offset” and “derivative”, put the column header for the offset column before the column header for the derivative column (this will be explained below when discussing the input terms for the tga CharacterizationTechnique in Figure 18).

1. function\_list: a list of strings that are associated with various functions or features within this program (explained below).
2. column\_numbers: a list of integers corresponding to which columns in the raw data file will be imported; starts at 0
3. default\_offset: the default number to offset the y values for each dataset; only used if 'offset' is in the function\_list
4. start\_row: an integer designating the first row in the raw data file to import data from; starts at 0
5. end\_row: an integer designating the last row in the raw data file to import data from; counts up from the bottom row of the raw data file (so the bottom row is end\_row = 0)
6. separator: a string designating the separator used in the raw data file; examples include ',' (comma), ';' (semicolon), '\\t' (tab), ‘\\s’ or ‘ ’ (space), ‘\\s+’ (all whitespace). The separator term can also be a regular expression (see <https://docs.python.org/3/howto/regex.html>). Again, note that two forward slashes “\\” are used here, while only one forward slash “\” would be used when manually entering the separator in Figure 9. If the separator is a comma, then the separator is optional and can be set to None since the “pandas” python module used to read csv files uses a comma as the default separator. However, it is best to put ‘,’ as the separator to be explicit.
7. xy\_indices: a list of two integers designating which of the imported data columns correspond to the x and y indices, which are used for calculations from the function\_list; for example, [0, 1] would mean the first column is the x data and the second column is the y data; the indices refer to the data after importing, so even if the column\_numbers used to import from the raw data file were [1, 2], the xy\_indices would still start at 0 and the available columns would be 0 and 1.
8. xy\_plot\_indices: a list of two integers designating which data columns correspond to the x and y indices when plotting (or for fitting data); similar rules as for xy\_indices, i.e. starts at 0 regardless of the column\_numbers, but the indices can be higher than the number of columns imported; for example, if the imported data has columns 0 and 1 (x and y columns), and an offset calculation is done on column 1, then the xy\_plot\_indices could be [0, 2] to use the raw x data and the calculated offset y data.
9. file\_type: the file extension (csv, txt, etc...) of the raw data files that will be used during file searching (Figure 6).

The list of available functions to put in the function\_list of the CharacterizationTechnique object are as follows:

1. offset: adds an offset to the y values of the raw data
2. normalize: will normalize the y data, based on height, between 0 and 1; allows selection of 'Normalize' in Figure 5b
3. derivative: will compute d(y)/d(x) using the central difference formula, where the derivative at point i is equal to (y­i+1 – yi-1) / ( xi+1 – xi-1) which smooths the data a little compared to a direct derivative.
4. negative\_derivative: will compute -d(y)/d(x) using the central difference formula
5. max\_x: will only take data up to the maximum x value; to use when x increases and then decreases, like for TGA data.
6. fractional\_change: computes (yi – y0) / (yf – y0) where y0 and yf are the initial and final y values, respectively, and yi is the y value at any point. Can be used for kinetics studies, such as for decomposition or diffusion. Note that the term will equal 0 when yi = y0 and 1 when yi = yf, regardless of whether y is an increasing or decreasing function. Also note, the given expression is equivalent to (y0 – yi) / (y0 – yf)
7. fit\_peaks: allows peak fitting.

Graphical representations for all of these functions except for fit\_peaks are shown below in Figure 19 through Figure 24.

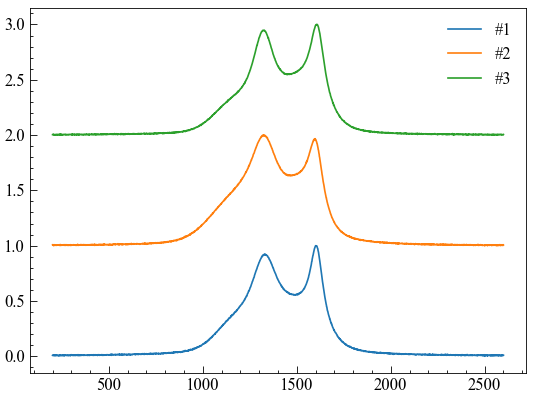
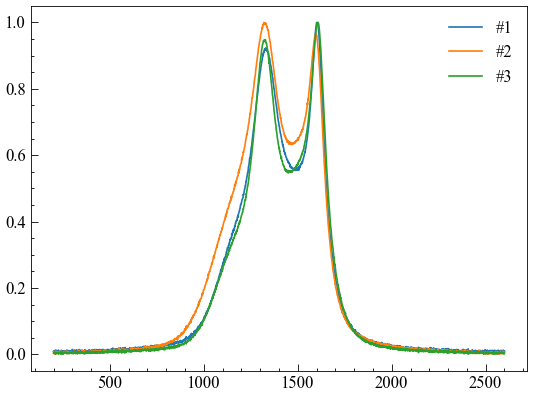


Figure : (a) Data without offsets and (b) data with offsets.

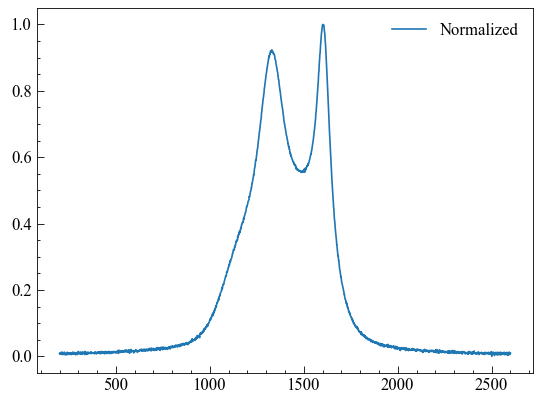
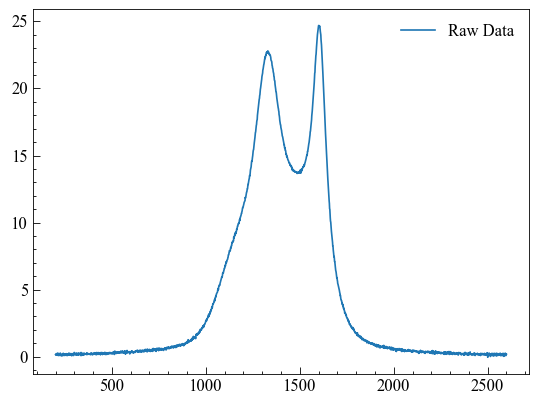


Figure : (a) Raw data vs (b) normalized data.

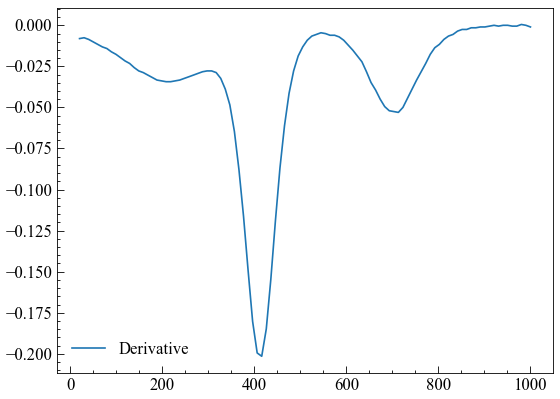
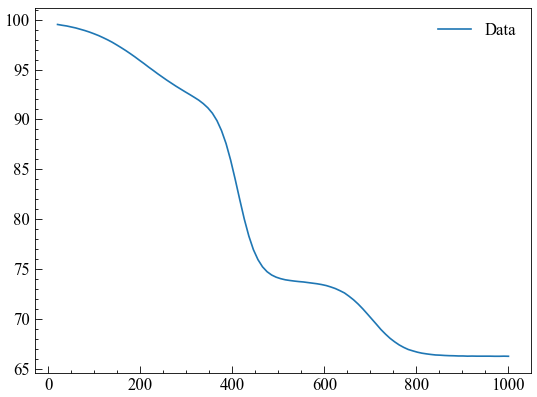


Figure : (a) Raw data and (b) derivative of the data.

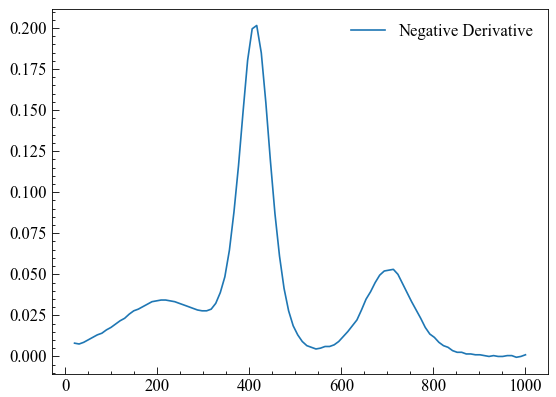
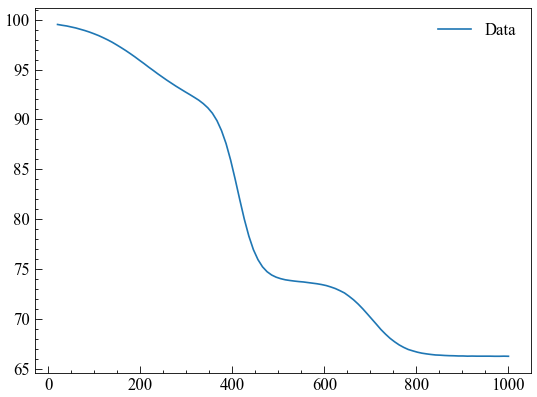


Figure : (a) Raw data and (b) negative derivative of the data.

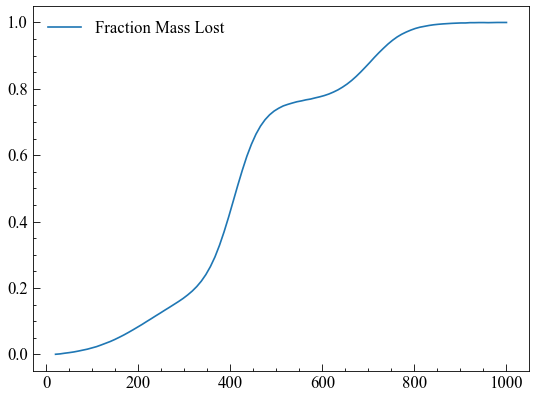
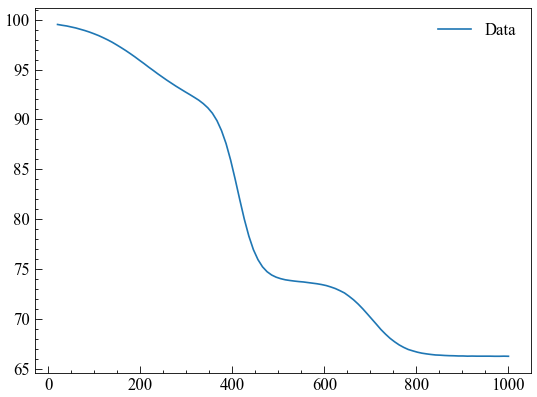


Figure : (a) Raw data and (b) fraction mass lost as a function of temperature, using the “frational\_change” function.

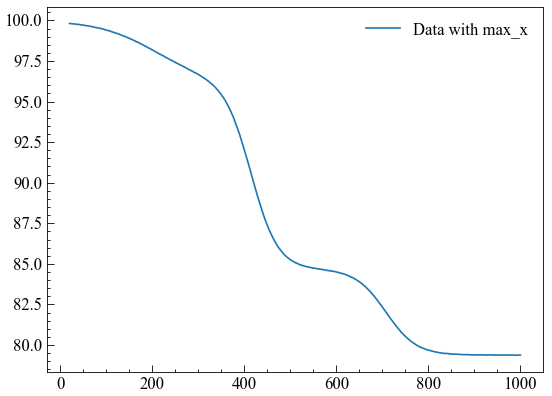
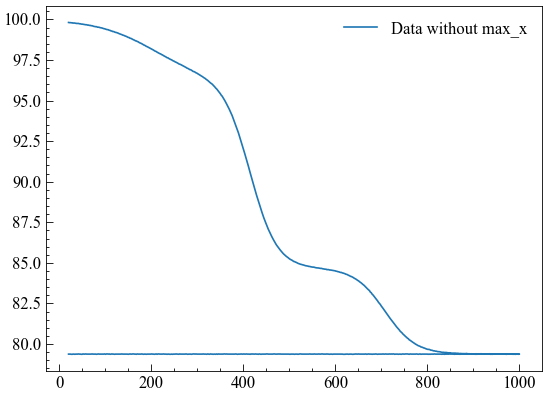


Figure : (a) Data without max\_x in the CharacterizationTechnique’s function\_list and (b) data with max\_x.

Put explanation here for the creation of TGA data since it is probably the most complicated and can have multiple calculations.

As an example, a new CT object will be created to handle DSC data generated by the generate\_raw\_data program. The raw data for the DSC data is shown in Figure 25. Its parameters are as follows (with explanations in parentheses):

1. name: ‘DSC’ (Note: single quotes ‘’ and double quotes “” both designate strings in python)
2. column\_names: [‘Temperature (\u00B0C)’, ‘Time (minutes)’, ‘Heat Flow, exo up (mW/mg)’]
3. function\_list: [‘fit\_peaks’] (Will allow peak fitting for DSC data. Don’t want to put “max\_x” in the function\_list because both heating and cooling are important for DSC data)
4. column\_numbers: [0, 1, 2] (uses the first three columns)
5. default\_offset: None (does not matter since “offset” is not in the function\_list)
6. start\_row: 34 (the data starts on the 35th line, but python starts at 0, so start at row 34)
7. end\_row: 0 (data continues to the last row of the file)
8. separator: ‘;’
9. xy\_indices: [0, 2] (so temperature is the x data, and heat flow is the y data)
10. xy\_plot\_indices: [0, 2] (if plotting, would want to plot temperature on the x axis and heat flow on the y axis)
11. file\_type: ‘txt’

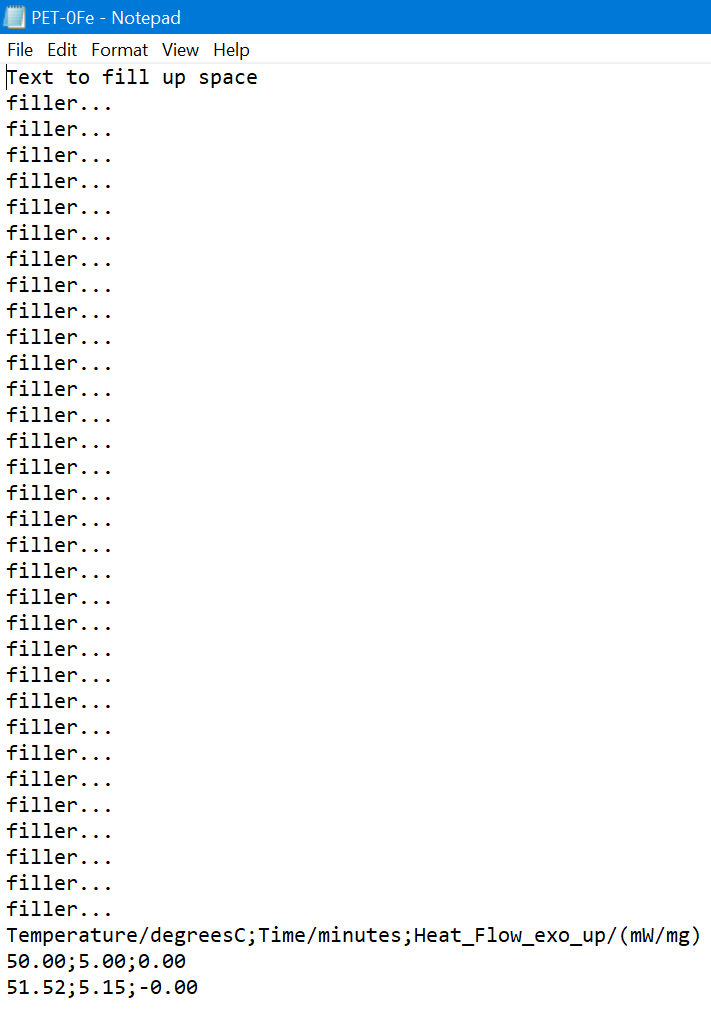


Figure : The beginning of a file for the DSC data.

Figure 26a below shows the resulting python code to create a new CT object to use for DSC data, and then adding the variable representing the new CT object (dsc) into the char\_techniques tuple. Now when running the program, DSC can be selected as a characterization technique, as seen in Figure 26b.

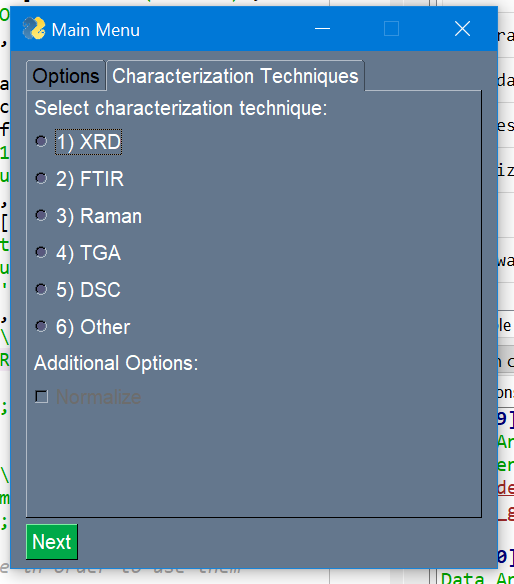
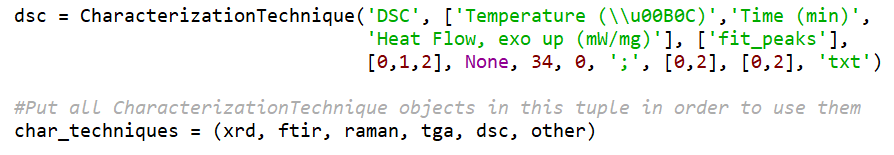


Figure : (a) The code to create a CT object for DSC data, and adding it to the char\_techniques tuple. (b) b, but now with the option to select DSC as the characterization technique.

# Adding Functions/Calculations

Add text here about the steps needed to add a calculation.

1. Add a string for the calculation to the supported\_functions tuple in the CharacterizationTechnique initialization
2. If the formula does a calculation on the data, also add the string to the calc\_functions tuple, so that the program ensures there are enough columns to perform the calculation
3. If the formula does a calculation on the data, add the calculation to excel\_formulas and python\_formulas functions. In the excel\_formulas function, can either provide a string that will execute the formula in Excel (which is nice because its value can be changed if the data in the Excel file changes), or provide the actual values. Be sure that the row number is correct, since the data actually starts on row 3 (sample names and data headers are on the first two rows).
4. If using a string formula in excel\_formulas, be sure to add the appropriate formula to carry out the actual calculation in python\_formulas.
5. If not using a string formula in excel\_formulas, be sure to increase calc\_index (calc\_index +=1) so that the columns are still referenced correctly.
6. If the formula does not do a calculation (like ‘max\_x’ or ‘fit\_peaks’) be sure to document what the function does.

Give an example of adding a function.

# Changing Excel Formatting

An example of the code for formatting the generated Excel files can be seen in . By default, alternating sets of data will have different background colors so that distinguishing between data sets is easier. If you wish to change the colors used, or various other formatting options, all you have to do is edit the code shown in . The two functions that write to Excel are the generate\_excel function in excel\_gui.py and the fit\_to\_excel function in peak\_fitting\_gui.py. To specifically change the colors used, you would edit the values after the term “bg\_color”; the colors need to be in hexadecimal format. The formatting is done using the python module “xlsxwriter”, so further details on controlling the formatting of the generated Excel files can be found by looking at the documentation for xlsxwriter.

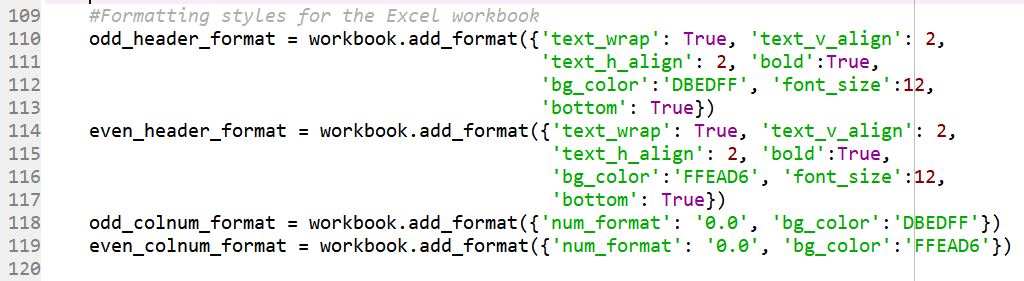


Figure : Code that changes Excel formatting.

# Peak fitting

# Some text explaining peak fitting

Recommend using dedicated software for each characterization technique, since the background and peak fitting parameters will be specialized for that particular technique.

Mention that both positive and negative peaks can be fit, but need to make sure that the y values are positive/negative after subtracting the background -> can check using the manual peak selector, which does the initial background subtraction in the same way as the peak fitting process

For fitting the background, the included functions are quite simple. More complicated methods can be used, including asymmetric least squares (ALS), penalized asymmetric least squares (eg. airPLS, arPLS), polynomial/spline fittings, etc. Look at the python module Rampy for ideas on implementing these background types in python. If using these methods, should subtract the background from the data before fitting, and do fitting without a background specified.

Talk about lmfit, peak finding parameters (prominence and height)

Discuss how to determine a good fit (R2, Bayesian info criteria, etc)

Explain that each peak is individually fit and then added to the total model that consists of all of the peaks and the background, if present, added together

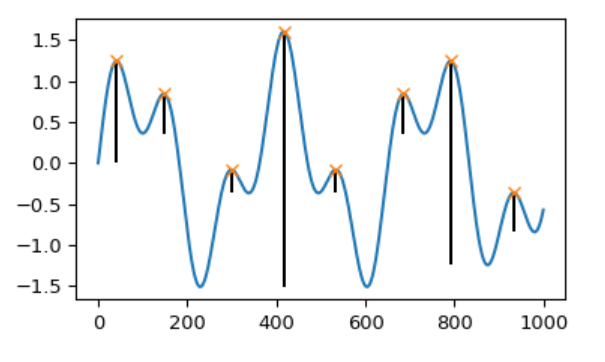


Figure x: plot showing the prominence of each peak (obtained from <https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.peak_prominences.html#scipy.signal.peak_prominences>).

# Examples

# Basic Example

Change the file to refer to gauss2.dat from lmfit’s github, and then change the section below as needed. Update gui pictures. Add a section on how to manually select peaks, or else create another example using the xrd data from section 2. -> probably best to do another example to show off more features and make the basic example very easy and quick to read. In the intermediate example, also discuss the debug button, and the plots that it shows (minus the residual plots, which will be discussed in the residual section).

This example will walk through using the GUI to fit a dataset containing two Gaussian peaks, which is also used as a tutorial by lmfit: <https://lmfit.github.io/lmfit-py/builtin_models.html#example-3-fitting-multiple-peaks-and-using-prefixes>. This example will use the same inputs as those used in the lmfit tutorial, where applicable. The data for the example can be obtained from NIST’s website: <https://itl.nist.gov/div898/strd/nls/data/LINKS/DATA/Gauss2.dat> or from lmfit’s github: <https://github.com/lmfit/lmfit-py/blob/master/examples/NIST_Gauss2.dat>.

If directly running peak\_fitting\_gui.py, the first window to appear is , but with none of the fields filled in. If peak fitting is being done through excel\_gui.py, will appear after file searching or directly after if “Single File” was the selected data source. For the Gauss2.dat file, to get the correct import options, set the separator to “\s+” (all whitespace between text), set the start row to 60, and set the data columns to “0, 1”. Hit the “Test Import” button to confirm that the data will be imported correctly from the file.

Hitting Next will bring up the Peak Fitting window shown in . Most fields in the Peak Fitting window already have default inputs, to ease the use of the program. Note that “inf” and “-inf” denote infinity and negative infinity, respectively. The first tab in the window, Raw Data, contains the following fields:

1. Sample Name: name of the sample, will be the sheet name in the Excel file containing the output of the peak fitting (to be discussed)
2. Column of x data for fitting: allows selection of which column to use as the x data
3. Column of y data for fitting: allows selection of which column to use as the y data
4. x data label: label for the x data in the output Excel file
5. y data label: label for the y data in the output Excel file
6. x min: minimum x value to include in the peak fitting
7. x max: maximum x value to include in the peak fitting
8. Show plots after fitting: whether or not to show a plot of the fit and a plot of each individual peak after fitting (the plots can be seen at the end of this example)
9. Batch Fit: if selected, the same options selected after doing one fit will be used for all subsequent fits without showing the Peak Fitting window again; good for analyzing multiple datasets from an Excel file all with the same peak positions.
10. Peak x values: the user can specify the peak locations here rather than relying on peak finding.
11. Prominence: the minimum height a peak must have for it to be considered a peak by the peak finding algorithm (a figure showing prominences of various peaks is shown in section 5.1). If prominence if infinite (as it is by default), then no peak finding will occur and the program will rely on user input peaks from the Peak x values field.
12. Minimum height: the minimum y value a peak must have for it to be considered a peak by the peak finding algorithm.

On the bottom of the Peak Fitting window are four button: Fit, Test Plot, Show Data, and Reset to Default. Pressing Fit will commence the peak fitting algorithm if all of the necessary user inputs are filled in. Pressing Test Plot will show a plot of the data along with the range of data to be used for fitting (changed by changing x min and x max), the range of data to use for fitting the background, if there is a background (to be discussed), and any peaks either input by the user in the Peak x values field or found using the peak finding algorithm with the input prominence and minimum height values. Pressing Show Data will show a table of the data, similar to Figure x(b). Finally, pressing Reset to Default will set all of the values back to their original default values.

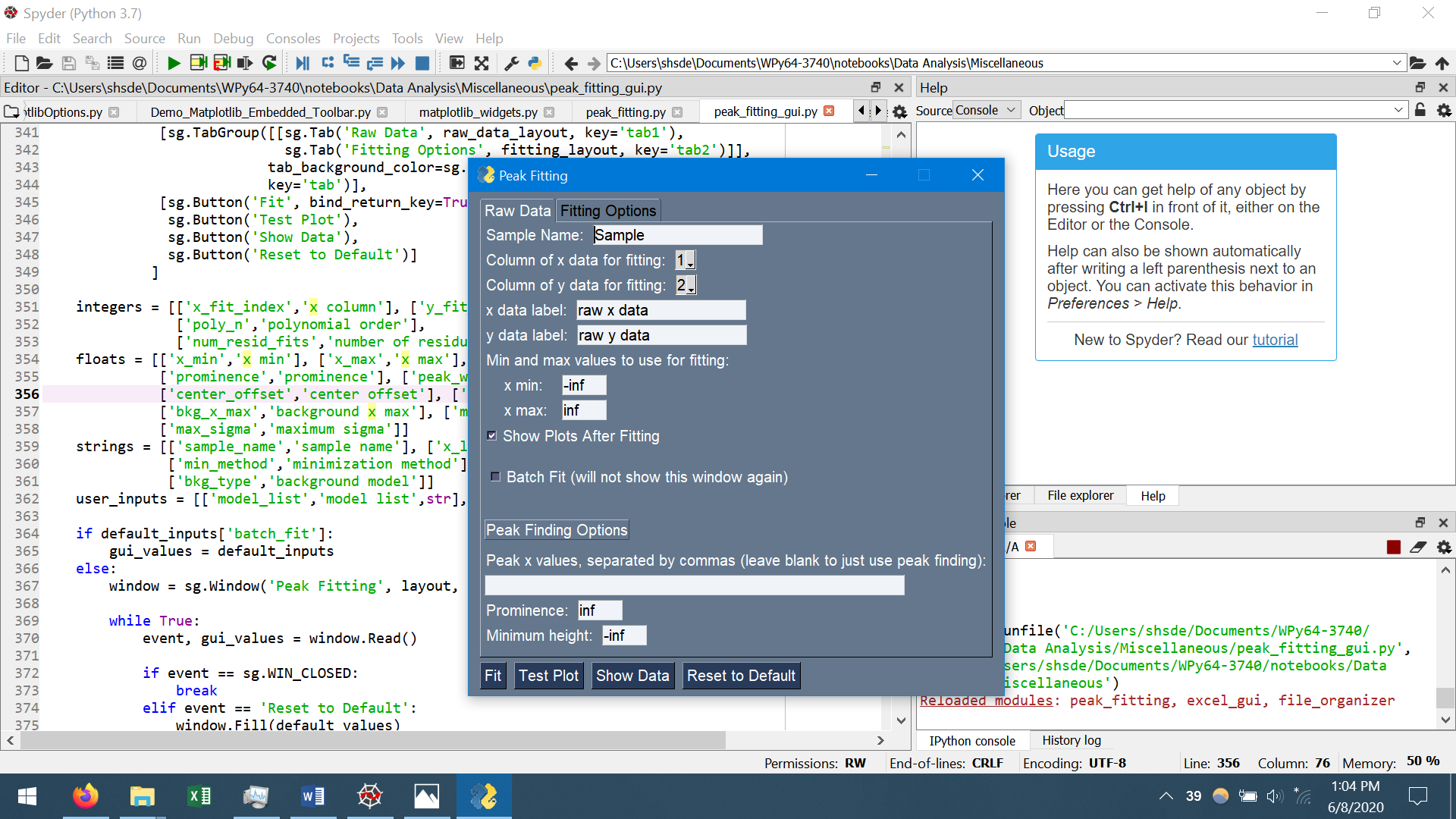


Figure : The Peak Fitting window with default options filled in.

The first step in peak fitting is to look at the data. Pressing “Test Plot” without changing any of the default values will show the picture in a, which does not look correct. Looking at the Gauss2.dat file, the y data is in the first column and the x data is in the second column, so they need to be switched in the program. Do this by setting “Column of x data for fitting” to 1 and “Column of y data for fitting” to 0, resulting in the correct orientation shown in b. It is now apparent from b that there are two peaks, centered at around 105 and 155.

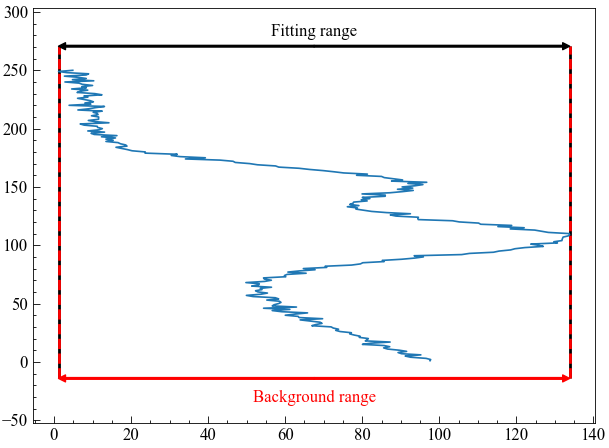
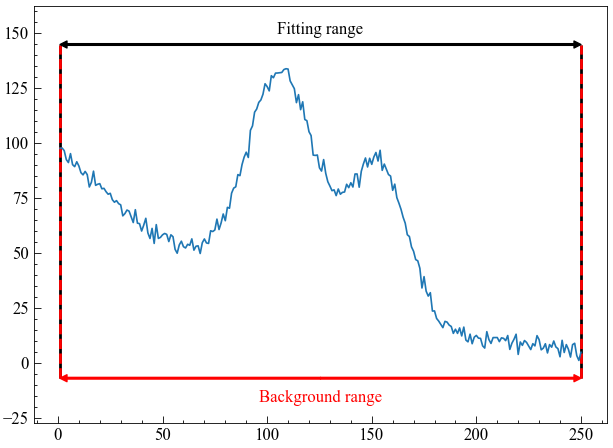
 

Figure : The data from gauss2.dat after pressing Test Plot (a) with the default values and (b) after changing the x column to 1 and y column to 0.

For this example, the only change that needs to be made on the Raw Data tab is to change prominence to 20. The sample name and x and y labels can also be changed if desired. Figure x below shows the Peak Fitting window after making these few changes.

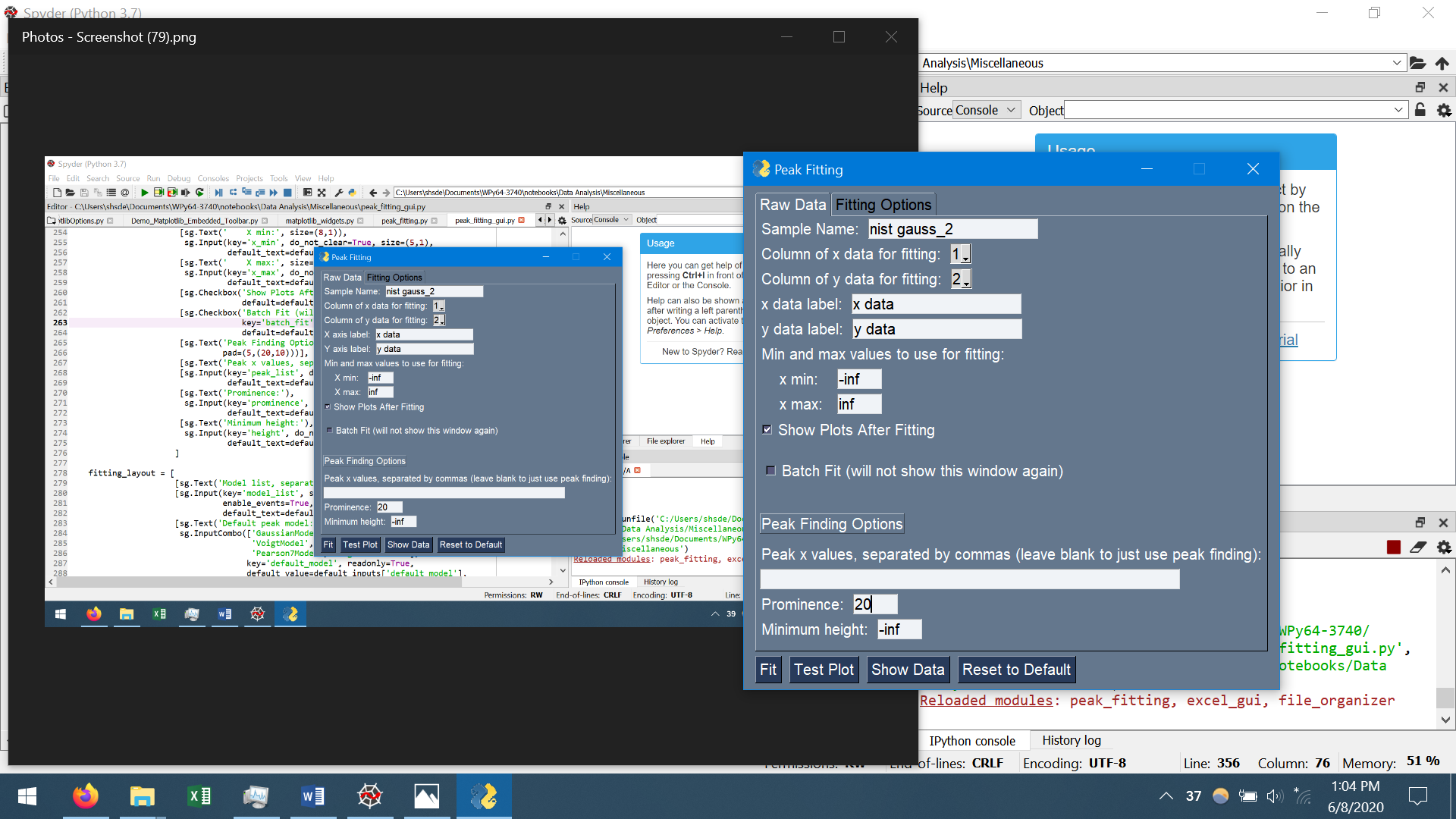


Figure x: The Peak Fitting window after applying some changes.

Pressing Test Plot after changing the prominence to 20 will produce the plot shown in Figure x(a). Alternatively, if the user kept prominence as “inf” and input “105, 155” as the peak x values, the plot shown in Figure x(b) would appear. Note that the user can both manually input peak positions and use prominence and height to find peaks at the same time if desired.

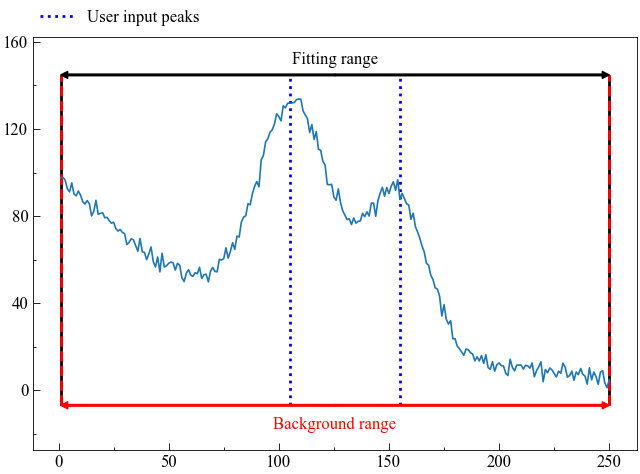
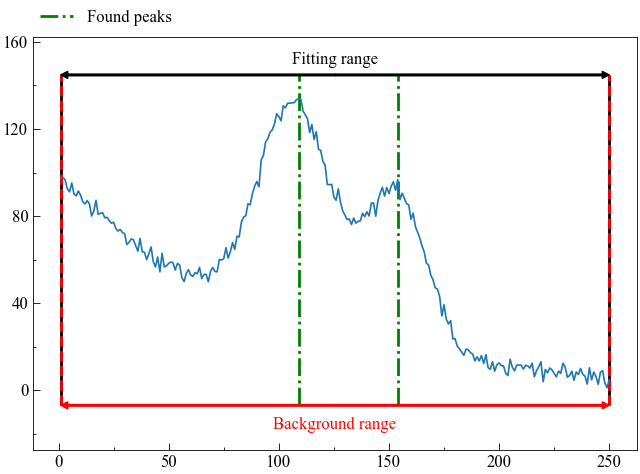
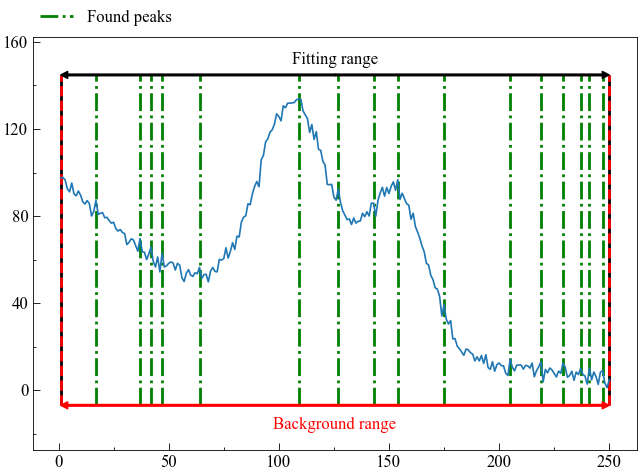


Figure x: (a) peaks found using height = -inf, prominence = 20; (b) peaks manually input at 105 and 155

A prominence value of 20 was chosen because with that value, the peak finding algorithm finds only the peaks near 105 and 155. If a larger prominence was chosen, the smaller peak at 155 could potentially be excluded. Alternatively, a lower prominence value could be input to “find” other smaller peaks. Figure x(a) shows all of the peaks found using a prominence of 5. In this case, the minimum height can be increased so that all peaks other than the main two are no longer considered peaks. Figure x(b) shows the result of setting the minimum height to 95 while keeping the prominence at 5.



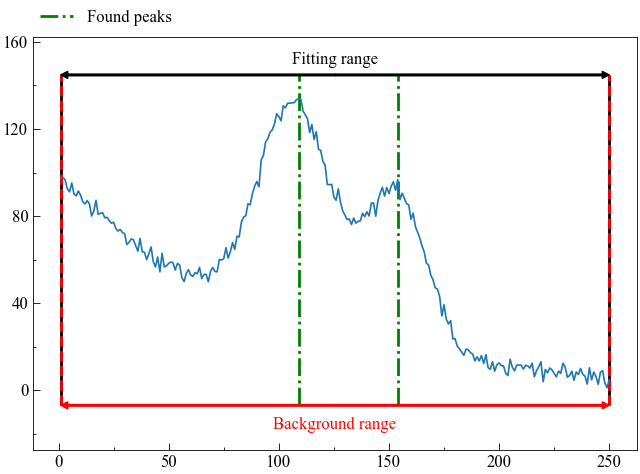


Figure x: (a) Peaks found using height = -inf, prominence = 5, (b) peaks found using height = 95, prominence = 5.

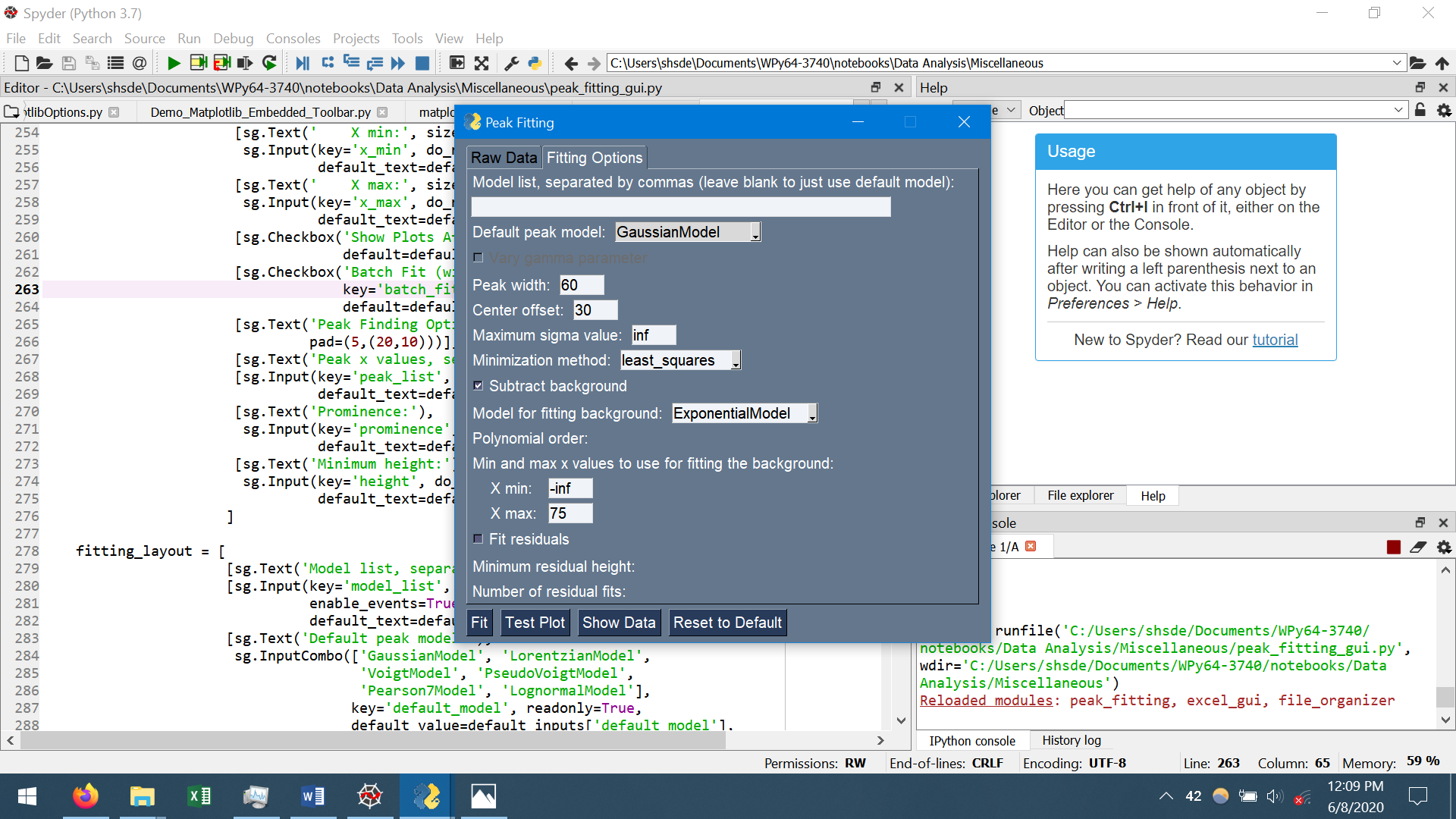
The second tab of the Peak Fitting window is Fitting Options, shown in Figure x. The Fitting Options tab consists of the following fields:

1. Model list: allows user to input models in a specific order if more than one type of peak can be found within the data. Model names can be found on lmfit’s website: <https://lmfit.github.io/lmfit-py/builtin_models.html>, and are typically the peak name followed by “Model”, eg. GaussianModel.
2. Default peak model: the peak model that will be used if there is no model list, or if more peaks are present than models input in the Model list field.
3. Vary gamma parameter: if using a VoigtModel or a SkewedVoigtModel, the gamma parameter will be equal to the sigma parameter if this checkbox is not checked; if checked, the gamma parameter will be allowed to vary independently of the sigma parameter.
4. Peak width: the approximate width of peaks in the data; this value is used to initialize the peaks before doing the fitting on the entire dataset, but it does not have to be very exact.
5. Center offset: determines the maximum and minimum allowable values for the peak center in the fitting (min = peak center – center offset, max = peak center + center offset)
6. Maximum sigma value: the maximum value for the sigma parameter in the peak models; typically, it can remain as infinite, but it can be reduced to control the maximum width that peaks can have (the full-width-at-half-max for most peak types is ~ 2\*sigma). Section 5.5.2 will discuss changing the maximum sigma value.
7. Minimization method: the minimization method that will be used in the peak fitting; the only two options in the drop-down box are “least\_squares” and “leastsq”, although the user can also manually input a minimization method instead. Section 5.5.1 discusses the minimization methods in more detail.
8. Subtract background: if checked, the background will also be fit.
9. Model for fitting background: the model to use for the background; the only two options in the drop-down box are PolynomialModel and ExponentialModel, although the user can manually input a different model, if desired (will probably also include ConstantModel at a later time).
10. Polynomial order: only available if PolynomialModel is selected as the background model; determines the maximum order of the polynomial used for fitting the background. For example, the polynomial c4\*x4 + c3\*x3 + c2\*x2 + c1\*x + c0 has a polynomial order of 4.
11. x min: minimum x value to use for fitting the background.
12. x max: maximum x value to use for fitting the background. The background is initially fit independently of the peaks, so choosing a min and max x where only the background is present will give a better initial fit
13. Fit residuals: if checked, the program will search for peaks in the residual data (raw y data – fitted y data) and then redo the peak fitting with any additional peaks found.
14. Minimum residual height: a value between 0 and 1; the height relative to the value max\_y – min\_y above which the peak finding algorithm will consider a peak to be a peak (I plan on changing the residual fitting a bit to make this a bit clearer).
15. Number of residual fits: the maximum number of times the program will loop through the fitting and residual peak finding functions; the program can also exit the loop if the adjusted r^2 value of the fitting (describes how close the fitting is to the data) does not change, which usually indicates no more residual peaks were found.



Figure x: The Fitting Options tab with the default values.

For this basic example, most of the fields in the Fitting Options tab will not be changed. The only inputs are 60 in the Peak Width field, 30 in the Center Offset field, changing the background model to ExponentialModel, and setting the max x value for the background to 75. These changes are shown in Figure x(a). Pressing the Test Plot button after making these changes will display the plot shown in Figure x(b).



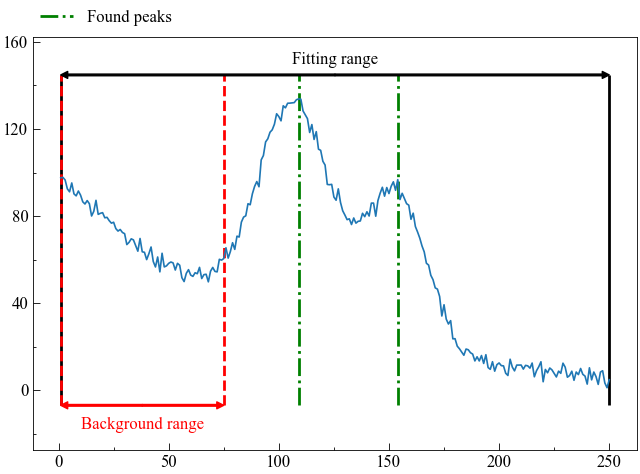
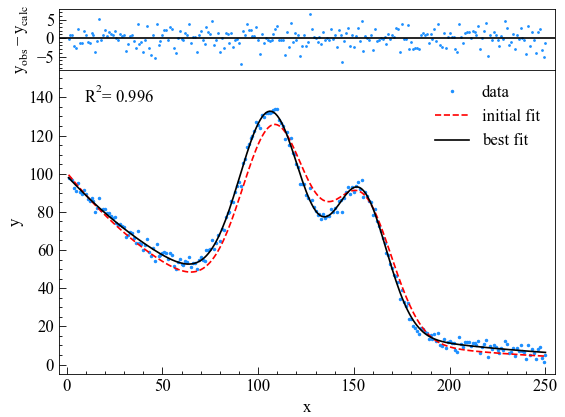


Figure x: (a) the changes made to the Fitting Options tab; (b) the plot shown by clicking the Test Plot button with the values in (a).

All of the necessary changes have been made in the Peak Fitting window, so next press the Fit button. After a short amount of time, the fitting will complete. If “Show plots after fitting” was selected in the Raw Data tab, then the two plots shown in Figure x(a) and (b) will appear. Figure x(a) shows the raw data, the initial fit, and the fitting after optimizing. The top portion of Figure x(a) shows the residuals of the fitting, where yobs is the raw y data and ycalc is the fitted data. Figure x(b) shows each individual peak, the background, and the sum of the peaks and background.



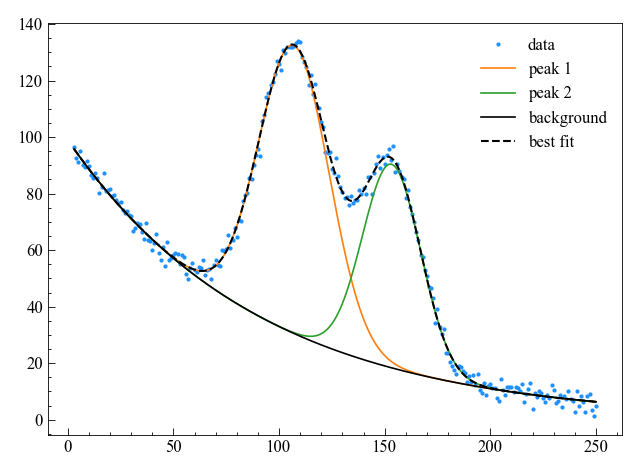


Figure x: (a) plot showing initial fit and best fit after peak fitting; (b) plot showing each individual peak, the background, and the best fit.

If peak fitting was done by running peak\_fitting\_gui.py, then the program will output an Excel file called “temporary file from peak fitting.xlsx” in the same folder as the program. The Excel file has three sections. The first section contains the values for the raw x and y data, and the data for all of the individual peaks, the background, and the total fit (peaks+background).

The second section contains the value and standard deviation (st. dev.) for all of the parameters for the peaks and background. An explanation of the parameters for the peaks can be found from <https://lmfit.github.io/lmfit-py/builtin_models.html> (note that for peaks, the name of the parameter ‘amplitude’ is changed to ‘area’ in this program to reduce confusion about its meaning. The way lmfit defines its models and parameters, amplitude is equivalent to area). For the parameters, if the peak does not have that parameter, a “-” will appear in the column. If a standard deviation could not be calculated for the parameter, usually resulting from a very incorrect fitting, then it could be listed as either “N/A” or “-”, depending on where the calculation failed. Likewise, if the standard deviation has roughly the same or greater magnitude as the parameter’s value, then it most likely indicates an incorrect fitting.

The third section of the Excel file is the “Fit Descriptors” (I need to think of a better name), which gives some information about the fitting. The four descriptors are adjusted r2, Akaike information criteria (AIC) Bayesian information criteria (BIC), and the minimization method used for the fitting. The adjusted r2 simply tells how well the fitting matches with the raw data, and differs from regular r2 by taking into account the number of variables in the fitting. AIC and BIC are values that can be used to compare fittings of the same raw data with different fit parameters, with the lower value being the more likely fit. The difference between AIC and BIC is that BIC is more conservative than AIC. For example, if fitting data with Gaussian peaks produced a BIC value of 10, and fitting the same data with Lorentzian peaks produced a BIC value of 11, then the Gaussian peaks are more likely the correct peak type to use. Refer to:

Fulton G, Lunev A. Probing The Correlation Between Phase Evolution and Growth Kinetics in the Oxide Layers Of Tungsten Using Raman Spectroscopy and EBSD. *Corros Sci* **2020**, *162*, 108221.

for seeing an example of how AIC and BIC parameters are used to determine the most likely number of peaks to use for fitting data (Note: I am not associated with the group that published that paper, it was simply an open-access paper I found when Google searching lmfit and Bayesian criteria).

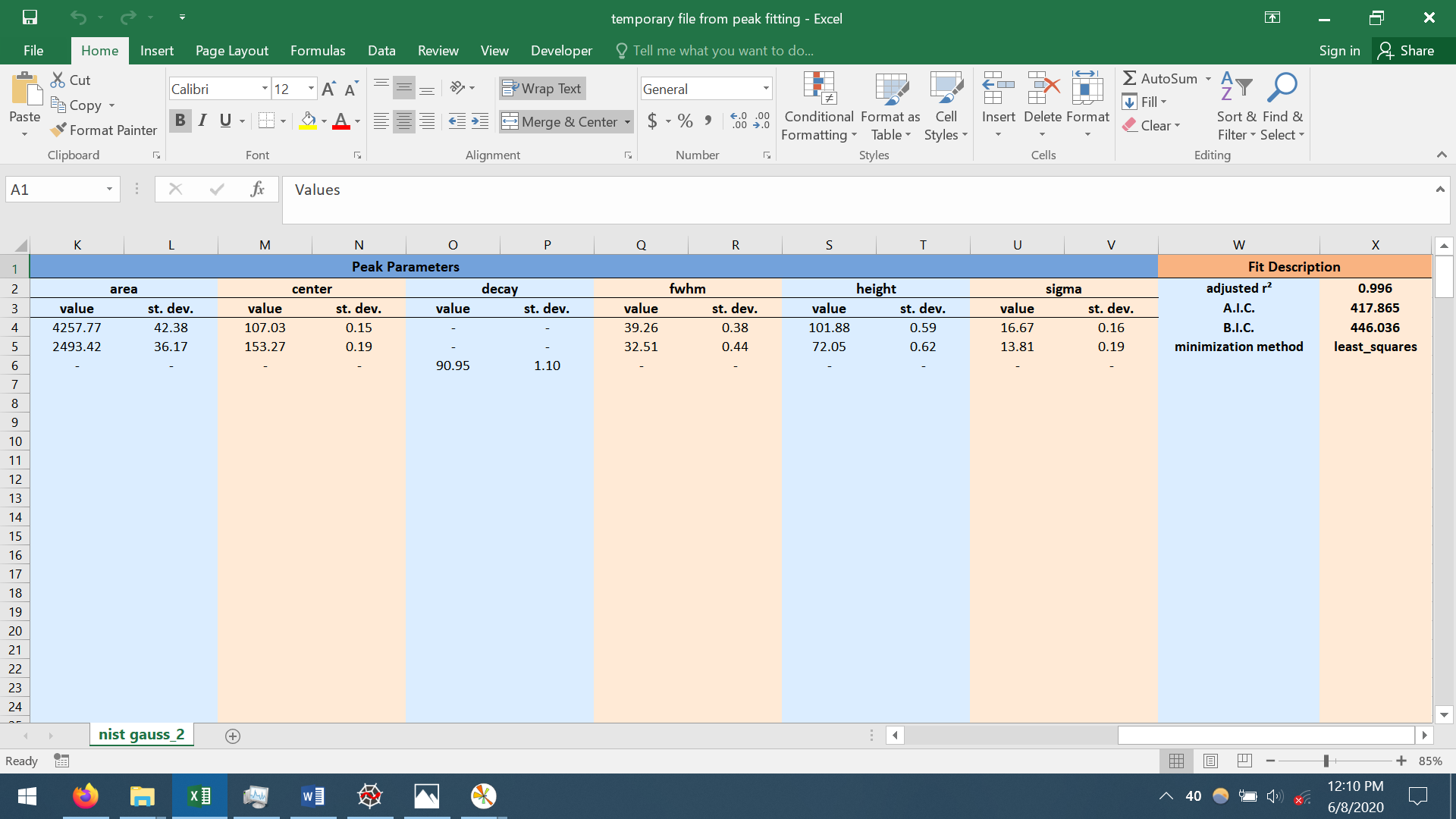
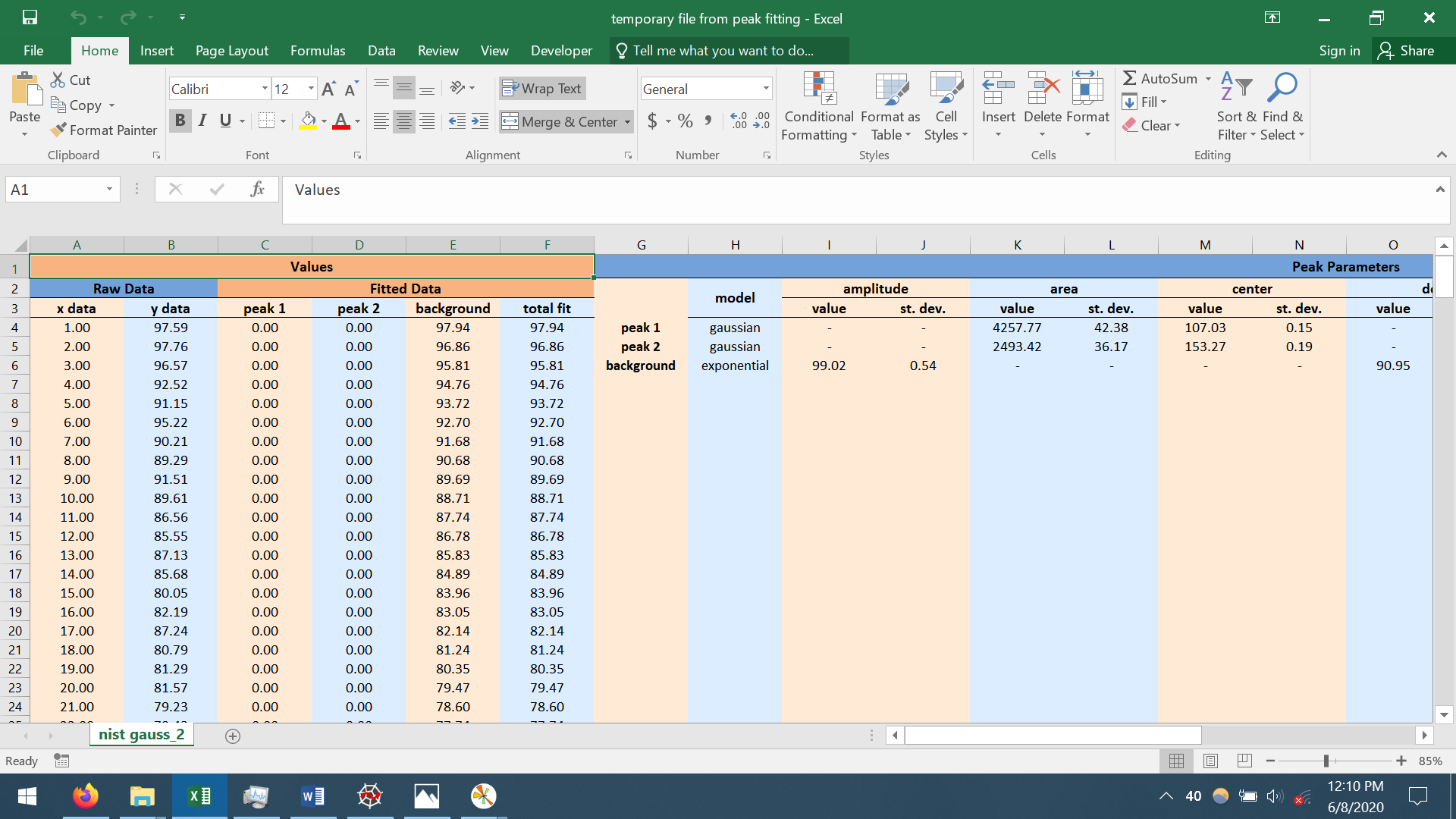


Figure x: Excel file output by the fitting program.

# Intermediate Example

This example will show how to process multiple sets of data by only setting the fitting parameters once, and will discuss some of the fitting parameters in more detail. This example will use the “output.xlsx” file created in Section 2 as the data source.

Discuss how peaks are initialized, using the data only between the middle of each peak center, and that the background is re-evaluated across the entire dataset once the peaks are created, which avoids getting stuck in a local minimum since the peaks were all initialized with the initial background.

# Multiples peaks with different line shapes and using BIC to find best model

Maybe put this section last since it seems the most advanced.

Discuss how for disordered carbon, literature cites multiple peaks can be present besides the main D and G peaks. Further, their lineshape is not always consistent in literature. In this case study, will first decide the best number of peaks to fit the example Raman data, and then decide the best lineshape for each of the peaks.

To make the analysis faster, it is possible to select “Multiple Files” and just do a file search that will give the same file six times (). That way, all of the fits will be put into a single Excel file. Be sure to set the y column to column 1 (the raw data) rather than column 2 (the offset data).

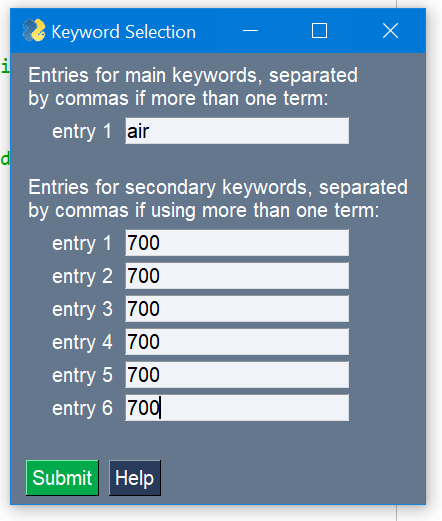


Figure : File search to get the same file 6 times for peak fitting with different options.

Be sure to set maximum sigma value to 300 (max fwhm ~600), just so that peaks are not allowed to have a very large width and small height. When using peaks at both 1590 and 1615, reduce the center offset to 10 so that the peaks are not allowed to merge. Mention that this first fitting will use pseudovoigt models, which are an additive combination of a Gaussian and lorentizian peak with the same fwhm, so that the peak type does not matter in this section. Also mention to use a 1st order polynomial to fit the background, setting min bkg x to 2400.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Peak locations | Center offset | Resulting AIC | Resulting BIC | Resulting reduced χ2 |
| Two peaks | 1350, 1590 | 20 | **-1675** | **-1626** | **0.185** |
| Three peaks | 1180, 1350, 1590 | 20 | **-2437** | **-2368** | **0.086** |
| Three peaks\_2 | 1350, 1500, 1590 | 20 | **-1931** | **-1862** | **0.143** |
| Three peaks\_3 | 1350, 1590, 1615 | 10 | **-1783** | **-1715** | **0.166** |
| Four peaks | 1180, 1350, 1500, 1590 | 20 | **-4495** | **-4406** | **0.011** |
| Five peaks | 1180, 1350, 1500, 1590, 1615 | 10 | **-4489** | **-4381** | **0.011** |

Discuss the larger error associated with the five peak model for the peak at 1615, and how it shifted to 1605 (the maximum it could shift since center offset=10) for both the Three peaks\_3 model and the Five peaks model, indicating that there was no peak for it to fit.

Discuss that for pseudovoigt models, if fraction = 1, it is lorentzian, if 0, it is Gaussian.

After deciding to use the four peak model, now need to choose the peak types for each peak. Could fit all gaussians, all lorentzians, or a mix of Gaussian and lorentzian peaks. Discuss how the results for the four peak model showed that the pseudovoigt’s fraction parameter was close to 0 for the peaks at 1180 and 1500, indicating Gaussian peaks, and close to 1 for the peaks at 1350 and 1590, indicating lorentzian peaks. Also mention that reduced χ2 should usually be close to 1, but since there is very little noise in the simulated spectra, and a large number of data points, it results in an overfitting with a χ2 much less than 1.

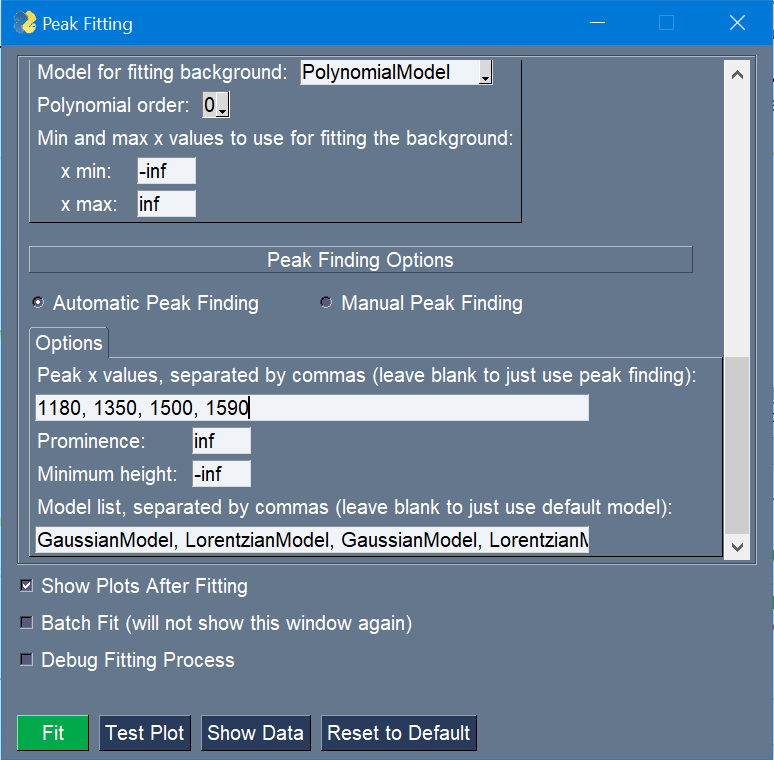


Figure : The Peak Finding section of the Peak Fitting window, showing how to input different models.

For this fitting, can set the center offset to a larger number (~50) since no peaks are close to each other. Can use the same file search as previously used, and just do four fittings out of the six files (can either exit after the fourth fitting, which will still save the Excel file, or can just press batch fit to fit the data again).

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Resulting AIC | Resulting BIC | Resulting reduced χ2 |
| All gaussian | **-1595** | **-1526** | **0.200** |
| All lorentzian | **-3986** | **-3917** | **0.018** |
| G for 1180 & 1500, L for 1350 and 1590 | **-4502** | **-4434** | **0.011** |
| All pseudovoigt (previous calc) | **-4495** | **-4406** | **0.011** |

Mention that this is only an overview of how to apply these methods. Actual Raman data for carbon may have significantly different spectra and lineshapes than the ones created by the generate\_raw\_data program.

# Finding and fitting residual peaks

Raman

Fit the same data as section 5.3, but just specify the peaks at 1380 and 1590. Then find the other two peaks by fitting the residuals.

Discuss how the residuals are initialized using data within center +- avg fwhm/2, rather than the typical process discussed in Section 5.2.2.

It is not recommended to rely on finding residuals, due to the difficulty in knowing what value to use for min\_resid. Instead, it is better to get possible peak positions from a literature search, and then compare the fit statistics (AIC, BIC, etc.) for fittings using increasing numbers of peaks, as demonstrated in Section 5.3, which is a more robust and statistics-based method. Further, residual peaks with negative heights cannot be found by the program, due to how it processes the residual data.

# Using a manual background and only fitting part of a spectrum

ftir

# Things to consider

# Minimization method

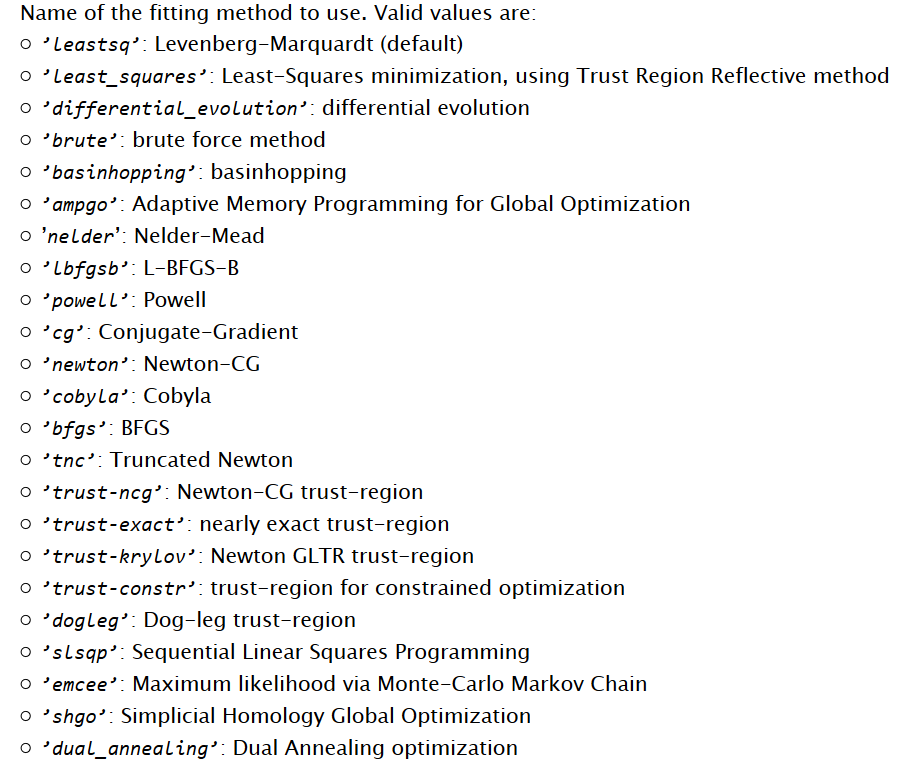


Figure x: valid inputs for the minimization method (obtained from <https://lmfit.github.io/lmfit-py/fitting.html>).

More information about the minimization methods can be found on scipy’s website: <https://docs.scipy.org/doc/scipy/reference/optimize.html>.

Discuss local minima vs global minima and what it is minimizing (squared error)

The “least\_squares” and “leastsq” methods are the two minimization methods that consistently produce good fit results in a timely manner. They are both local minimization techniques, so the final fitting result may not actually be the most optimal, but they usually are able to converge on a reasonable fit. The “least\_squares” is typically the faster of the two methods, and is the default for this program; the “leastsq” method occasionally does better at not getting trapped in local minima (based on my experience using these two methods for fitting). The “ampgo” global minimizing method works okay with simple models, but fails for more complex models; most other global optimizers take too long or do not produce good results.

for fitting the ‘basic example’ data (2 gaussian peaks and an exponential decay background)

least\_squares (local): ~0.1 s

leastsq (local): ~0.2 s

ampgo (global): ~2.8 s

basinhopping (global): ~36 s

for fitting four Lorentzian peaks from XRD data plus a 2th order polynomial background

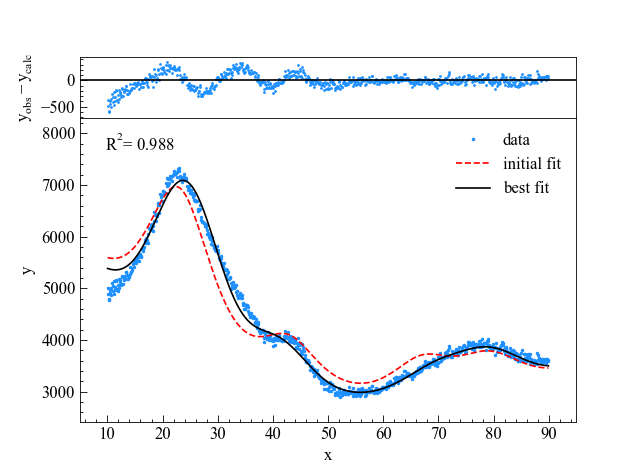
least\_squares: ~0.9 s

leastsq: ~0.4 s

ampgo: ~ 312.7 s

basinhopping: not tested

Considering that the residuals in Fig x(b) do not reflect the actual differences between the calculated fit and the raw data, I think the reason these fittings occur is that the function that handles the fitting produces an error which makes all the calculations fail.



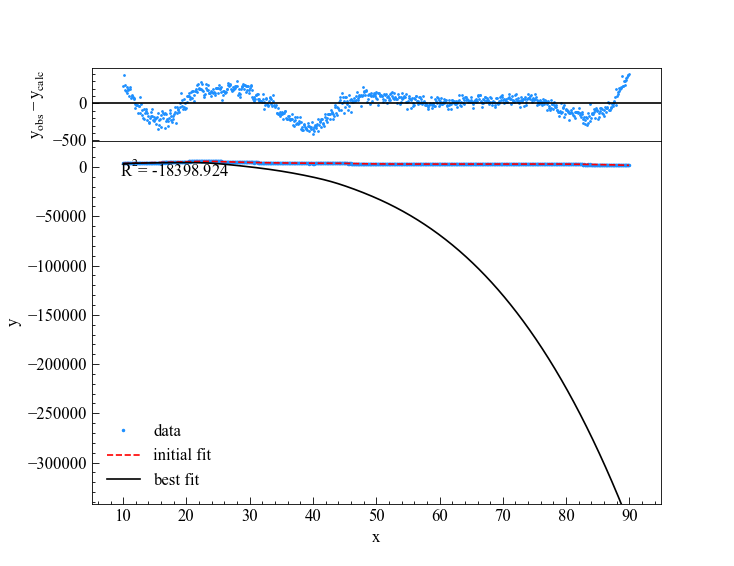
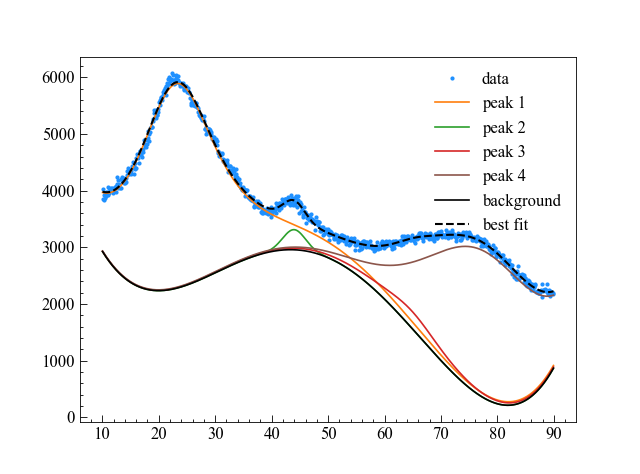


Figure x: (a) fitting using ‘ampgo’ minimization method using a 2nd order polynomial background; (b) ‘ampgo’ fitting using a 4th order polynomial.

# Picking a maximum Sigma

Sigma is approximately ½ \* fwhm, but varies slightly between peak models.

Usually do not need to specify a maximum sigma value, but when using higher order polynomials for the background (> 2nd order), it is usually best to specify a maximum sigma or else peaks will become very wide. Note that it is still guesswork to match a maximum sigma and a high order polynomial background.



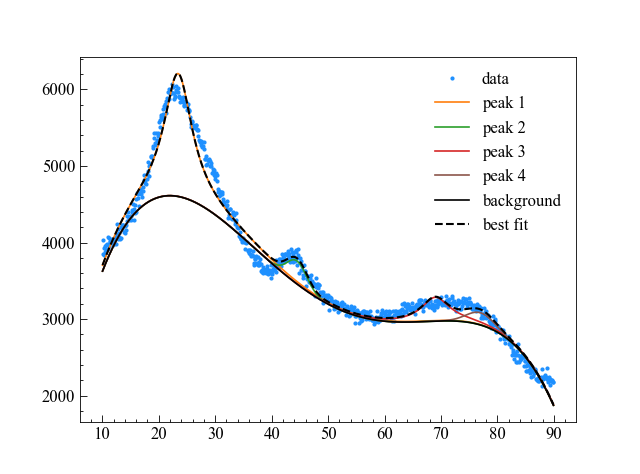
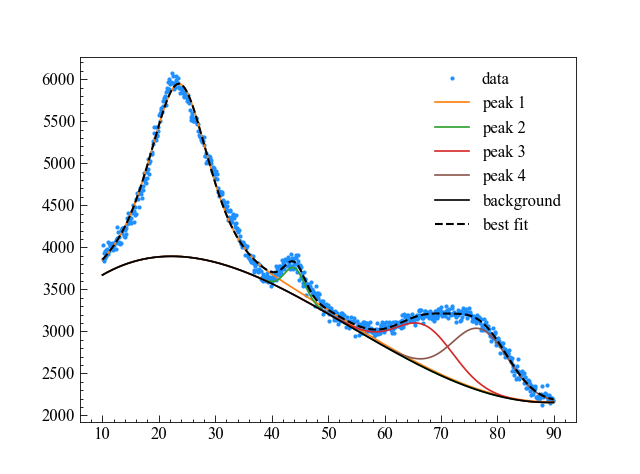


Figure x: (a) fitting with no maximum sigma value; (b) fitting with a maximum sigma value that is approximately correct (note how less chaotic the background is); (c) fitting with a maximum sigma value that is too low.

# Getting additional information

Look at lmfit’s documentation

print(fit\_results[0][-1].fit\_report())

show what ^ prints out.

# Plotting

NOTE: if using Times New Roman as the font for plotting in the python module “matplotlib” (by using matplotlib.pyplot.rcParams['font.serif'] = "Times New Roman"; matplotlib.pyplot.rcParams['font.family'] = "serif"), the font may become bold due to a mistake in how matplotlib reads system fonts. To correct this, use the following code (gotten from the issues tracker on matplotlib’s github: <https://github.com/matplotlib/matplotlib/issues/5574>), but be sure to copy the matplotlib folder first in case an error occurs:

import matplotlib

del matplotlib.font\_manager.weight\_dict['roman']

matplotlib.font\_manager.\_rebuild()

Plotting is done through the module matplotlib. There are numerous options to configure plots using matplotlib, too many to put into a GUI. Luckily, matplotlib gives a file named “matplotlibrc” (referred to as the rcParams file) which gives options to change many defaults for plots, such as text size, tick locations, line size, etc. More details on the rcParams file can be found at <https://matplotlib.org/3.3.0/tutorials/introductory/customizing.html#customizing-with-matplotlibrc-files>, and the method to find where the rcParams file can be found at <https://matplotlib.org/3.3.0/faq/troubleshooting_faq.html#locating-matplotlib-config-dir>. Changes to the rcParams can also be done during the running of code rather than editing the file, which is the method that this program uses. In fact, each DataSource can be given its own set of rcParam changes, to specify the plot layout for each type of DataSource.

The figures that are created by the program are returned by the plotting function, so any further changes can be made using the return figure objects. To view any of the figures, use figure.show() rather than plt.show() since plt.show() does not work once the figures have been closed.

mention symbols can be made either through using mathtext (<https://matplotlib.org/3.3.0/tutorials/text/mathtext.html>) or with unicode. For example, if the user wants to display the theta symbol (θ), in mathtext it would be “$\theta$”, while in Unicode it is “\u03B8”. Using mathtext may produce italicized, bold, or other formatted text, depending on the parameter “mathtext.default” in the rcParams file. Use “regular” as the term for the mathtext.default” term to use the same format as for regular text.

Plotting uses matplotlib’s tight\_layout option to ensure that all axes and labels fit within the figure. The plotting\_gui program disables constrained\_layout, so that tight\_layout is used instead, just so that the user knows. From testing, tight\_layout seems more stable at the moment for more complicated layouts, and is more easily configured, so it was selected rather than constrained\_layout. In order to change the padding between the edge of the picture and the figure, change the constant plotting\_gui.TIGHT\_LAYOUT\_PAD before running the functions (the default is 0.3 \* the text size). Further, invisible axes can be created in the plot layout window, allowing the user to put invisible axes around the axis they want to show, and allowing each invisible axis to have its own length and width to further modify the spacing. To change the horizontal or vertical spacing between plots if plotting multiple plots, change the constants plotting\_gui.TIGHT\_LAYOUT\_H\_PAD and plotting\_gui.TIGHT\_LAYOUT\_W\_PAD before calling the plotting functions (the default for both is 0.6 \* the text size).