# CHEM0062 Report

# XPS Spectra Plotter with Custom Error Datapoint Exclusion

## Summary

The importance of X-ray photoelectron spectroscopy is described and the need for a spectra plotter which excludes error datapoints at a user-defined threshold is explained. The inputs that are needed for the script to run and the expected outputs are stated alongside a brief description of the script. The six iterations of the development of the program are discussed. This encompasses the challenges and solutions faced from the initial plotting and folder-selection pop up creation stage to the production of output plots. This is then followed by five tests which display how input errors were prevented and calculations produce valid results. Furthermore, the testing section showed how the standard deviation threshold function worked whilst the other proposed (percentage) threshold function did not. Also, this programs compatibility with other similar datasets was also proven. Documentation is provided including installation and examples of how the program should be used. This report concludes with a self-reflection regarding the progress of my programming understanding and suggestions for future improvement of this program.

## Background

Soft X-ray photoelectron spectroscopy (XPS) is a surface analysis technique. It can be used to determine the elemental composition of a material’s surface as well as the electronic state of surface atoms. When analysing samples which are extremely beam-sensitive, the B07 beamline at Diamond Light Source can be used.1

Due to the sensitive nature of the samples, errors in data acquisition are common. Furthermore, the instruments which are used for XPS commonly face calibration errors which cause further errors in the collected data. To ensure that errors do not corrupt the whole dataset, a sample is scanned multiple times, collecting multiple ‘sweeps’ of results. A single beamtime can produce files with hundreds of sweeps. An outputted .dat file contains all individual sweeps for one component type of a sample’s surface composition. For example, oxygen atoms may be one component type of a sample’s surface composition.

The proposed program aimed to plot average plots for each component on the surface of a sample and exclude the datapoints which are most likely to be errors at a user-specified threshold.

An XPS spectrum plots intensity against binding energy. For a plot to be produced, the photon energy at which a scan was run needs to be used to convert recorded kinetic energy to binding energy as shown in Equation 1.

(1)

Where BE = binding energy (eV), PE = photon energy (eV) and KE = kinetic energy (eV).

There are similar programs to this one. “XPS visualizer and converter” (XPSVC) allows for the plotting of a cumulative spectrum for XPS data in the .xy format.2 “LG4X” is another program for XPS spectra plotting which allows for average spectra plotting, optimised average spectra plotting and curve fitting.3 LG4X allows for .txt, .csv and .vms input formats.

These alternative programs do not allow for the exclusion of error datapoints at a user-specified threshold. A user may want to specify and adapt the error threshold themselves since the datapoints with high errors may, occasionally, be in a key range. Also, the user may want information on the error datapoints (e.g. standard deviations) so they know whether they need to recalibrate their XPS machine or take more scans for a particular surface component. Furthermore, the experimental data on which the proposed program is based is in the standard machine outputted .dat format. The alternative programs do not allow for .dat files to be inputted. Hence, if a user were to use one of these alternative programs, they would have to convert multiple files to the suitable format which may be time consuming.

## Program specifications

### Inputs

* ‘y’. Upon running the program, an introductory message will appear. This letter must be entered to continue.
* A folder containing .dat files will be chosen by the user via a pop-up folder selection window. These .dat files will contain the experimental data from the XPS machine. Specifically, each .dat file should contain kinetic energy in eV in its first column, intensity values for each sweep in the following columns and sum of the intensities in its last column (unless the file only has 1 sweep, in which case a sum column is not required).
* The photon energy in eV at which the scan was run. This must be a positive float or integer.
* A positive number (without units) which will serve as the “standard deviation threshold value”. Datapoints with standard deviations above this number will be excluded from the plots.
* A choice of the type of output plots desired. “i” is entered for individual plots, “f” is entered for final combined plots and “b” is entered for both types of plots.

### Outputs

* Either:
  + Individual plots for each of the individual .dat scan files

OR

* + A final combined plot which combines all the individual plots.

OR

* + Both plot types.
* A file containing the standard deviations of all datapoints for all the files in the folder. Data in this file are in the format: [filename, datapoint number, line in the respective file where the datapoint is, standard deviation].
* Files containing the excluded datapoints and included datapoints for each individual plot.

### Details

This program extracts values for kinetic energy, sweep intensities and sum of intensities from .dat files by reading in the file line-by-line (datapoint-by-datapoint). Calculations are carried out to determine the binding energy, mean intensity and standard deviation for each datapoint. With binding energies on the x-axis and mean intensities on the y-axis, plots are produced. These plots exclude error datapoints based on one of the standard deviation-based error exclusion thresholds described above. Files containing details of the included/excluded datapoints and a file containing the standard deviations of every datapoint are outputted.

## Development

### Iteration 1: basic plot creation

The first version of the script aimed to read a single file and create a plot with the kinetic energy on the x-axis and the first sweep intensity on the y-axis. This is initial script is shown below in Figure 1.

Text

Description automatically generated

Text

Description automatically generatedFigure 1: Code snippet of first version of the script. Initial code for plot creation.

This initial version did not work for a couple of reasons. Firstly, the delimiter set above was a space but the actual delimiter of the file was a tab. Hence, the " " was replaced with "\t" as shown in Figure 2.

Figure 2: Code snippet of updated delimiter line.

Secondly, the file path was written incorrectly and so the print(os.path.exist(filepath)) statement returned “False”. This is because the file path should have been defined by joining the folder path with the file name however what was done above meant that: file path = [folder path + file name + file name]. This mistake was made as the UV mini project was used as a template for this code but the file path definition code was initially misunderstood. This was fixed via removal of the hard coded file name as shown in Figure 3.

Figure 3: Code snippet of updated file path line.

Thirdly, to read the file and extract values, the first line needed to be skipped as this line contained column headers. The range “numhead -1” did not allow this line to be skipped. The logic behind this was that the first line had index 0 and so setting the range in this way would cause the script to read the first line of the code and then skip it, after which it operates on the remaining lines of the file. However, this did not work because, after the first line is skipped, an empty string is passed into the while loop which causes an error. This was fixed by changing the range to “numhead +1” as shown in Figure 4. This meant that the first line of the file was read and discarded and then the following lines are passed into the while loop.

Figure 4: Code snippet of updated for loop range line.

These edits allowed the code to run and an initial plot to be created. The x-axis in XPS plots should be binding energy not kinetic energy and so code for this conversion was inserted as shown in Figure 5. Furthermore, the y-axis was changed to mean intensity, the calculation for which is shown in Figure 5. The axis labels were adjusted accordingly.

Text

Description automatically generatedFigure 5: Code snippet of the updated while loop. Note: the num\_sweeps variable has been defined earlier in the script as “num\_sweeps = 10” and the photon energy has been defined as “photon\_energy = 2410” in the variable definition section of the script.

Chart, histogram

Description automatically generatedRunning this code resulted in the output plot shown in Figure 6.

Figure 6: Plot created by the script produced in Iteration 1.

### Iteration 2: key input and output code

A dialogue box pop-up window was created which asks the user to import a folder. This was done to prevent the user from making typos which could occur if the user were to manually write the file path into the terminal. A loop was created which retrieved the file paths for all the files in the selected folder. This was used to create plots for all .dat files in a folder.

To do this I adapted code from: <https://stackoverflow.com/questions/18262293/how-to-open-every-file-in-a-folder>, see Figure 7 for my code adaptation.

Figure 7: Code snippet of the for loop which allows for reading of all .dat files in a folder. The main plot creation code block was entered into this for loop.

This worked for the first plot but did not for the other plots because the other files did not have the same number of sweeps (column headers). To fix this, “datapoint[11]”, as shown in Figure 5, was replaced with “datapoint[intensity\_sum\_index]” and intensity\_sum\_index was defined as “intensity\_sum\_index = int(len(datapoint)) -2” on the line above it. This meant that the value for the index of the last column was no longer hard coded but, instead, determined for the file using the len() function on the list containing the values in a line. There was some confusion as to why “2” was taken away from the length of the list rather than 1 (since the index was desired). This was because all the lines in the experimental files end in a tab which causes the script to recognise an empty string at the end of the line – this string was excluded.

This update allowed all 5 plots to appear as shown in Figures 8-12.

Chart, histogram

Description automatically generatedChart, line chart

Description automatically generatedChart, histogram

Description automatically generated

Figure 8: Plot called “figure 1”. Figure 9: Plot called “figure 2”. Figure 10: Plot called “figure 3”.

*Chart, histogram

Description automatically generatedChart, line chart, histogram

Description automatically generated*

Figure 11: Plot called “figure 4”. Figure 12: Plot called “figure 5”.

However, the first plot would pop-up in a window which would need to be closed before the next would appear. Since this is inconvenient for the user if they have many files, the “plt.show()” command was replaced with the line shown in Figure 13. This saved the figures to the folder that was selected earlier and uses the enumerate() function shown in Figure 7 to name the plot.

Figure 13: Code snippet of the line which saves the figures.

### Iteration 3: correction of plot errors

One issue with Figures 8-12 was that lines connected different sections within the plots in a way which is uncharacteristic of a XPS plot. After calculating the binding energies and mean intensities of some of the points for each file, it became clear that this was because each plot after the first included the plots for the previous files that were run through the loop. Since plt.plot() joins datapoints with lines, lines connected the separate sections together on the plots (although no datapoints were actually present on those lines).

To rectify this, “plt.plot(xaxis, yaxis)” was replaced with “plt.scatter(fp\_xaxis, fp\_yaxis, marker="o", s=0.5)”. This meant that the plots were now outputted as scatter plots and so the connecting lines disappeared as shown in Figures 14-18. This was fitting as experimental XPS plots are not usually presented with connecting lines.

Chart, line chart, histogram

Description automatically generatedChart, line chart, histogram

Description automatically generatedChart, histogram

Description automatically generated

Figure 14: Updated plot called “figure 1”. Figure 15: Updated plot called “figure 2”. Figure 16: Updated plot called “figure 3”.

Chart, histogram

Description automatically generatedChart, histogram

Description automatically generated

Figure 17: Updated plot called “figure 4”. Figure 18: Updated plot called “figure 5”.

The aim was to provide the user with 5 separate plots for the 5 files and 1 final combined plot which would include all the plots (as all plots represent a component on the surface of the scanned material). Individual plot creation was achieved by moving the x-axis and y-axis lists into the main loop which meant that they would be reset after every iteration. Furthermore, the final combined plot was created by introducing 2 new lists “fp\_xaxis” and “fp\_yaxis” which were placed above the main plot so that all datapoints from all iterations would be included. The code used to create this plot was placed after the main loop.

Furthermore, a mistake was noticed at this stage. The “num\_sweeps” variable was still set to 10 which was only true for the first file. This mistake meant that the other plots relied on an incorrectly calculated mean. Hence, “num\_sweeps” was defined as “num\_sweeps = intensity\_sum\_index – 1”. This altered the plots, but the difference is not easily visible, and so the plots have not been shown in this report.

### Iteration 4: standard deviation calculations and error exclusion threshold functions

A list called “intensities\_in\_a\_line\_list” was created to store the sweep intensities in each line. To do this, the “datapoint” list containing all values in the line was split within a range to exclude the energy and sum values. Furthermore, conditional statements were used because files with only a single sweep do not have a sum column and so the range used to split the “datapoint” list should only cut off the energy. At this point, it became evident that this was overlooked when calculating the mean. The num\_sweeps for a file with 1 sweep would be equal to zero. Since it is impossible to divide a number by zero, the equation for mean intensity calculation was adjusted to also involve a conditional statement. The standard deviation for each datapoint was calculated using the code from the “Introduction to Python Pt1” file from Moodle as a template. These edits are shown in Figure 19.

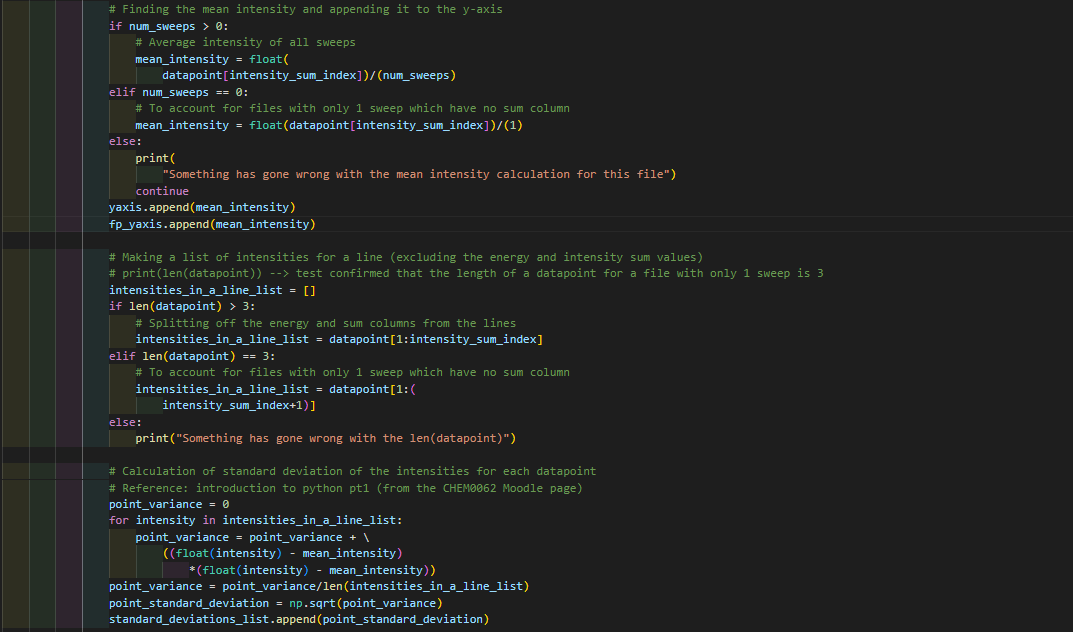


Figure 19: Mean intensity calculation, intensities\_in\_a\_list creation and standard deviation calculation.

Two error exclusion thresholds were created. The standard deviation threshold removes datapoints above the user specified standard deviation which is assigned to a variable called “point\_standard\_deviation” – this variable uses the input() function. This function uses a simple inequality condition. The percentage threshold defines a range with the top X% datapoints with the lowest standard deviations and removes any datapoints outside this range – where X is the user specified percentage stored in the variable “percentage\_input”. (Note Figure 20 shows a hard coded percentage and standard deviation. These were later replaced by user inputs as shown in Figure 22.) This function uses an equation which finds the maximum standard deviation in the accepted range based on the maximum standard deviation in the set and the percentage. It then removes all datapoints with standard deviations above this value using an inequality condition. Two lists were also created called included\_point\_numbers and excluded\_point\_numbers which stored the included and excluded datapoints in two separate lists. This is all shown in Figure 20.

Text

Description automatically generated

Figure 20: Code snippet of the error exclusion threshold functions. The first line is cut off but the statement printed is not important as it is later completely changed – this line takes an input of either ‘s’ or ‘p’.

To let the user choose between the two error thresholds, a conditional statement was introduced which used the user input variable “std\_or\_percentage” in Figure 20. This can be seen in Figure 21. This code was placed after the threshold functions.

Text

Description automatically generated

Figure 21: Code snippet showing assignment of user input choice to the corresponding threshold type using conditional statements.

Furthermore, the hard coded percentage of “60” in figure 20 was replaced with the variable percentage\_input and the hard coded “40” in Figure 20 is also replaced with an input variable as shown in Figure 22. This code was placed after the definition of “std\_or\_percentage” but before the threshold functions.

Text

Description automatically generated

Figure 22: Code snippet showing the variables containing the user-specified standard deviation/percentage threshold value.

### Iteration 5: text file creation

Three text files were created: one contained standard deviations for each datapoint, one contained excluded datapoints and one contained included datapoints. Information was printed to text files using the code from the following link as a template: <https://howtodoinjava.com/python-examples/python-print-to-file/>.

Initially, I tried to write the files by writing “w” within the open() function to write the standard deviation text file as shown in Figure 23. This did not work as it only printed the standard deviation of the last datapoint of the last file. This was because the “w” meant that the text file would overwite itself for every line. To resolve this problem, “w” was replaced with “a+” as this appended each standard deviation rather than overwriting them. This worked and so it was also used to write the other 2 text files as shown in Figure 24. Concatenation was used to print statements alongside values for the excluded/included points files so that the data is self explanatory for the user. The standard deviation file does not have this but the “Specification: Outputs” section of this report as well as the script itself outline how the data is in the format: [filename, datapoint number, line in the respective file where the datapoint is, standard deviation].



Figure 23: Code snippet showing the broken code which tried to write the standard\_deviation\_file.txt file.

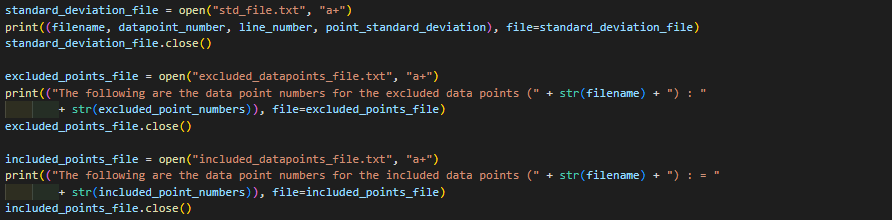


Figure 24: Code snippet showing the creation of the 3 text files.

Although this worked, it only worked when running the program once. When the program was run for the second time, it appended the second set of values from the second run onto those from the first run. This problem did not occur if the text files were deleted before running the program again. To ensure that the user does not need to deal with this, the code in Figure 25 was added at near the top of the program (above the main loop) to delete these files at the start of every run if they are already present in the folder.

Text

Description automatically generated

Figure 25: Code snippet of a loop which deletes the output text files on the condition that they already exist.

### Iteration 6: refinement of the script

In this iteration, finishing touches were applied to the code. Besides Figures 26 and 27, no additional figures are shown here as all the discussed edits (other than that relating to Figure 27) are present in the final “Code Listing” section at the end of this report.

Firstly, the photon\_energy variable was assigned to “float(input(“Enter photon energy in eV”))” as this variable was hard coded up to this point. Also, a loop was created with a conditional statement and try/except statements to ensure that the inputs for photon energy and standard deviation were both positive numbers. An error message is printed if not. A try/except statement was also used to print an error message for when the wrong folder is used. This was done by redirecting the NameError which appears when running the standard deviation function to print an error message. I learnt how to do this using: <https://www.oreilly.com/library/view/python-cookbook/0596001673/ch17s02.html>.

Secondly, loops and functions were created to increase efficiency.

A function called “datapoint\_removal()” containing a loop was created to decrease the repetition evident is Figure 20. This loop uses a range on a list of the axes lists of the format [x::y] to remove binding energies from the x-axes and mean intensity from the y-axes. The [x::2] format allows for the selection of every other list in the axes list, starting from the list with index x.

The datapoint\_removal() function was embedded into the threshold functions to minimise repetition. See Figure 26 for the three functions of the program. (Note: the testing section will justify why the code for this section differs in the final code listing).

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

Figure 26 Code snippet of the three functions. The percentage threshold function is removed and the datapoint removal function is embedded into the standard deviation threshold function in the final version. However, these functions are still visible in the “Discussed in testing section folder” within the “testing\_section\_code.py” file.

Common delimiter alternatives which the user may accidentally use instead of a tab were replaced with the tab delimiter. The code for this was replaced by a loop as shown in Figure 27.

Text

Description automatically generated

Figure 27: Code snippet showing the delimiter replacement code. The first attempt at writing this is shown in the docstring. The final loop is shown after the docstring.

This code was tested by running the program with the original .dat input files with the standard deviation threshold at 100 and then running it again with the same conditions but after altering the input file. The input file was altered by replacing the first 4 tabs in the first file in the folder with a comma, semicolon, space and double-tab. This does not work however it works when excluding the double-tab. This does not work for the double-tab because the script believes that there is an empty list between the 2 tabs. If I had more time, I would find a way around this or a suitable way to tackle this. For now, the user is told to input files with tabs as delimiters. Note that, in the files provided, I have left these alterations (comma, semi-colon and space) in the first line of the file scan\_019873scan\_VB\_SCL.dat within the GdH2\_pSi\_2410 folder to display that this code works. (The original experimental files did not have these alterations.)

Thirdly, the input for the plt.savefig() function was changed so that the output figures are saved as “Figure for [filename]”. Also, some extra print statements were added to the script to add clarity of instruction to the terminal prompts. Also, sections were added at the start and end of the code which lets the user choose whether they want to output individual plots, final plots or both. This was done using conditional statements and logical operators (“and” and “or”).

Lastly, the formatting of the code was refined. The “Format on Save” option with the “autopep8” option in the settings of VSCode was used to format the script so that it fit the PEP 8 guidelines. Docstrings were added and comments were edited.

## Tests

### Test 1: verification of wrong input errors

The user inputs were tested and the errors which appear when an unsuitable input is entered are shown below in Table 1.

Table 1: The error messages which are printed to the terminal for unsuitable user-inputs for each input type are shown.

|  |  |  |
| --- | --- | --- |
| Input type | Incorrect input | Output message |
| Initial message | Anything other than “y” | "Re-run the code and make sure to just enter the letter 'y' with no spaces or apostrophes - just the letter." |
| Folder selection | Any folder other than the folder in which the script is saved in | "You selected the wrong folder. It must be the folder that this script is saved in." |
| Photon energy | Negative numerical input | "The number you entered for photon energy or standard deviation is negative. Re-run the program and try again." |
| Photon energy | Non-numerical input | "You did not enter a number for either photon energy or standard deviation. Re-run the program and try again." |
| Standard deviation choice | Negative numerical input | "The number you entered for photon energy or standard deviation is negative. Re-run the program and try again." |
| Standard deviation choice | Non-numerical input | "You did not enter a number for either photon energy or standard deviation. Re-run the program and try again." |
| Output plot type choice | Anything other than “b”, “B”, “i”, “I”, “f” or “F”. | "You did not enter an appropriate letter when choosing output plot types. Re-run the program and try again.") |

### Test 2: verification of key calculations

The calculation of binding energy, mean intensity and standard deviation were key to the production of the outputs. I tested these calculations for datapoint 1 of the scan\_019873scan\_VB\_SCL.dat file in the GdH2\_pSi\_2410 folder and presented the outcomes in Table 2.

Table 2: Tests for calculations of the key calculations involved in the program. Expected values are hand calculated.

|  |  |  |
| --- | --- | --- |
| **Calculation tested** | **Expected** | **Output** |
| Binding energy (eV) | 54.99951172 | 54.9995117200001 |
| Mean intensity (a.u.) | 609.88625489 | 609.886254883 |
| Standard deviation (a.u.) | 24.63787121 | 24.637871209340187 |

As shown, the expected values match the output values up to 7 decimal places. The binding energy value calculated by the program ends in a “1” which it should not in reality. This is caused by a “floating point error” which causes the program to round the infinite number of zeros up to 1. This does not affect the accuracy of this program too much as it occurs at the 13th decimal place. The mean values are only correct up to 7 decimal places since the program calculates the mean by dividing the sum column by the number of sweeps as opposed to adding all intensities up manually and dividing the sum by the number of sweeps. This causes it to give a slightly different mean than the expected because the sums provided by the experimental data is only accurate up to 7 decimal places. If the expected mean is calculated by dividing the experimentally provided sum by the number of sweeps, then the expected value would equal the output. The standard deviations are accurate up to a higher number of decimal places than I was able to calculate by hand.

### Test 3: verification of plotting with standard deviation threshold function

This program was created using the files in the folder “GdH2\_pSi\_2410”. To test the exclusion of datapoints using this threshold function, the program was run twice. Firstly, it was run with the standard deviation threshold set at 218 because 217.72607419999986 is the highest standard deviation of the dataset and so this would include all datapoints. Secondly, it was run with the threshold set at 30 so that some of the points would be excluded. The plots produced by the first and second run are shown in Figures 28-33.

Chart, line chart, histogram

Description automatically generated Chart, histogram

Description automatically generated

Figure 28: Output plots for scan\_019873scan\_VB\_SCL in folder GdH2\_pSi\_2410 with standard deviation threshold at 218 (on the left) and 30 (on the right).

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Figure 29: Output plots for scan\_019874scan\_Gd4d in folder GdH2\_pSi\_2410 with standard deviation threshold at 218 (on the left) and 30 (on the right).

Chart, histogram

Description automatically generatedChart, histogram

Description automatically generated

Figure 30: Output plots for scan\_019875scan\_Gd4p in folder GdH2\_pSi\_2410 with standard deviation threshold at 218 (on the left) and 30 (on the right).

Chart

Description automatically generatedChart

Description automatically generated

Figure 31: Output plots for scan\_019876scan\_O1s in folder GdH2\_pSi\_2410 with standard deviation threshold at 218 (on the left) and 30 (on the right).

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Figure 32: Output plots for scan\_019877scan\_Gd3d in folder GdH2\_pSi\_2410 with standard deviation threshold at 218 (on the left) and 30 (on the right).

Chart

Description automatically generatedChart

Description automatically generated

Figure 33: Output plots for the final combined plots in folder GdH2\_pSi\_2410 with standard deviation threshold at 218 (on the left) and 30 (on the right).

As shown in Figures 34 and 35 and evident from the plots above, the plots created in the first run included all datapoints whilst those from the second run excluded some datapoints.

A picture containing text

Description automatically generated

Figure 34: The excluded\_datapoints\_file.txt file for the first run of test 3.

A picture containing text

Description automatically generated

Figure 35: The excluded\_datapoints\_file.txt file for the second run of test 3.

Note that the 3rd file does not have any point exclusions since it only has 1 sweep and so the threshold functions do not work for this file. To make this file more accurate, the user should collect more sweeps (as they are told in the terminal).

To check that all excluded datapoints had standard deviations of above 30, a list called “std\_test\_stds” was created which contained all the standard deviations of all excluded datapoints. This was done by defining the list at the top of the main loop and the code in Figure 36. Manual checking of this file confirmed that none of the excluded datapoints had standard deviations above 30.

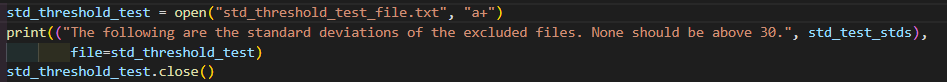


Figure 36: Code snippet of the standard deviation function threshold test file creation. This code was then removed as it is not needed in the final version of the program. However, the output file has been saved within the “Discussed in testing section” folder in the zip file along with a version of this python file with the testing edits (edits from this test and test 3).

### Test 4: verification of plotting with percentage threshold function

Testing of the percentage threshold function was similar to the testing of the standard deviation threshold function. Figures 37-42 show the output plots when the program was run with the threshold set at 100% and at 10%. Evidently, the function removed datapoints.

Chart, line chart, histogram

Description automatically generatedChart, scatter chart

Description automatically generated

Figure 37: Output plots for scan\_019873scan\_VB\_SCL in folder GdH2\_pSi\_2410 with percentage threshold at 100% (on the left) and 10% (on the right).

Chart, line chart

Description automatically generatedChart

Description automatically generated

Figure 38: Output plots for scan\_019874scan\_Gd4d in folder GdH2\_pSi\_2410 with percentage threshold at 100% (on the left) and 10% (on the right).

Chart, histogram

Description automatically generatedChart, histogram

Description automatically generated

Figure 39: Output plots for scan\_019875scan\_Gd4p in folder GdH2\_pSi\_2410 with percentage threshold at 100% (on the left) and 10% (on the right).

Chart

Description automatically generatedChart, line chart

Description automatically generated

Figure 40: Output plots for scan\_019876scan\_O1s in folder GdH2\_pSi\_2410 with percentage threshold at 100% (on the left) and 10% (on the right).

Chart, line chart

Description automatically generatedChart

Description automatically generated

Figure 41: Output plots for scan\_019877scan\_Gd3d in folder GdH2\_pSi\_2410 with percentage threshold at 100% (on the left) and 10% (on the right).

Chart

Description automatically generatedChart

Description automatically generated

Figure 42: Output plots for the final combined plot in folder GdH2\_pSi\_2410 with percentage threshold at 100% (on the left) and 10% (on the right).

To check whether datapoints were removed in a way which was in-line with the specified condition, the line shown in Figure 43 was inserted into the program which printed the maximum standard deviation that an included datapoint could have for each file. These printed values are shown in figure 44.



Figure 43: Code snippet which prints the maximum standard deviation for datapoints in one file in the folder. This code was inserted at the bottom of the file opening “with” loop so that a value was produced for each file. This is not shown in the final code as it is not needed but will be included in “testing\_section\_code.py” in the “Discussed in testing section folder”.

A picture containing text

Description automatically generated

Figure 44: The terminal output resultant from the code in Figure 43.

Since there were less included values than excluded values, the included values text file was checked. This confirmed that only datapoints with standard deviations below the maximum standard deviations were included. For conciseness, only the first file has been used as an example to show this. Included datapoints for scan\_019873scan\_VB\_SCL.dat and their standard deviations are shown in Table 3.

Table 3: Standard deviations of the datapoints included in the plot produced for scan\_019873scan\_VB\_SCL in test 4. All these values are below the threshold determined in Figure 44.

|  |  |  |
| --- | --- | --- |
| Datapoint number | Standard deviation | Below the threshold (6.31)? |
| 1061 | 5.798893771405422 | YES |
| 1064 | 5. 557537935747617 | YES |
| 1071 | 6. 269799540526251 | YES |
| 1116 | 5. 4346785462784 | YES |
| 1123 | 6. 112715874168888 | YES |
| 1129 | 4. 284345891116219 | YES |

This test shows datapoint exclusion however, the condition met was not the one that was initially intended. The initial intention was to create a function which only includes the (specified)% of most accurate datapoints however the equation used in this function uses the highest standard deviation for each file to output plots which only include values with standard deviations that are (specified)% of the maximum standard deviation.

For example, the dataset considered in the example above has a maximum standard deviation of 63.07536812980691. 10% of this maximum standard deviation is 6.307536812980691 which is the threshold value used to exclude datapoints above. This is not what was intended. For the intended outcome to be achieved, the datapoints should have been sorted in ascending order by standard deviation and the last 90% of datapoints should have been included.

Having noticed this mistake, an attempt was made to rectify this as shown in Figure 45 using dictionaries and another for loop. However, it did not work as it showed the error in Figure 46. In the future, I would fix this error.

Text

Description automatically generated

Figure 45: Code snippet of attempt at fixing the broken threshold function. This does not work so has not been included in the final code. It is available (commented-out) in testing\_section\_code.py in the “Discussed in testing section” folder.

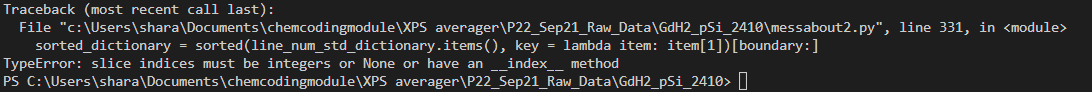


Figure 46: Terminal error message caused by the code in Figure 45.

Since the percentage threshold function does not work the way it was intended to, I have not included it in the final code. I have edited some other bits of the script (e.g. the code which makes the user choose between threshold functions as one is no longer an option). Notably, the datapoint\_removal function has been integrated into the standard\_deviation\_threshold function since there is no longer a reason to have it separate (as it is no longer repeated in 2 functions as the percentage threshold function has been removed). Since I would make some changes to fix this in the future, I have included the percentage threshold function that was created during the development stage (and this trial commented out) in the testing\_section\_code.py file in the “Test 4” folder in the “Discussed in testing section” folder.

### Test 5: verification of plotting for a range of experimental datasets

To check that the script works for other datasets, this program was also run on 5 other experimental datasets. The plots were run using the standard deviation threshold set at 100. All plots were successfully created for all 5 datasets, however, for the purpose of conciseness, only the final combined plots for all the 5 datasets are shown in Figures 47-51. The full collection of the figures produced have been saved in a folder in the zip folder provided.

The validity of error exclusion and calculations have already been shown so the purpose of this test is just to verify that plots (that look like they could be XPS data) can be successfully produced for other datasets with other photon energies and other numbers of files with different numbers of sweeps. This test was successful.

Chart

Description automatically generatedChart

Description automatically generated

Figure 47: Final combined plot for GdH2\_pSi\_7230 Figure 48: Final combined plot for TiH2\_pSi\_2410

Chart

Description automatically generatedChart

Description automatically generated

Figure 49: Final combined plot for TiH2\_pSi\_7230 Figure 50: Final combined plot for YH2\_pSi\_2410

Chart

Description automatically generated

Figure 51: Final combined plot for YH2\_pSi\_7230.

## Documentation

### Installation

xps\_plotter.py needs to be saved in the same folder as the .dat files for which plots will be created. The user may have downloaded this script into a folder but then want to run this script on a folder which it is not present within. In this case, the user must copy and paste the script file into this other folder and run this new copy of the script.

The user should install an IDE. For example, Visual Studio Code which can be installed using the “x64 User Installer” available at: [https://code.visualstudio.com/download#](https://code.visualstudio.com/download).

The user should install Ananconda (version 2022.05). This can be done using the following link: <https://www.anaconda.com/products/distribution>.

This script works with Python 3.9.12. This can be installed using the following link: <https://www.python.org/downloads/release/python-3912/>. Either the “Windows installer (64-bit)” or the “macOS 64-bit universal2 installer” should be downloaded depending on the user’s computer system and installed using all the default settings.

NumPy (1.21.5) must be installed in the command prompt using the following statement: “pip install numpy==1.21.5”. Matplotlib (3.5.1) must also be installed via the command prompt using the following statement “pip install matplotlib==3.5.1”.

The glob2 (0.7) module does not need to be used if the same version of python is being used. If an earlier version of python is used, then the command “pip install glob2==0.7” would be needed to install it via the command prompt.

### Usage

To run this code, open a new window in an IDE such as Visual Studio Code and open the folder containing the .dat files and the python script. Run the python script within this IDE and respond to the prompts that appear within the terminal. Input instructions are printed in the terminal. (Refer back to the specifications section for further detail on inputs and outputs).

Note: the standard deviation threshold is recommended since the percentage threshold could exclude low error datapoints in high precision datasets which have a highest error datapoint with a very low standard deviation. However, both can be useful depending on the user’s use case.

**Usage Example 1 – production of both output file types:**

Terminal prompt:

****

Figure 52: Terminal prompt stating initial instruction message to user.

Input: y

Terminal prompt: (The pop up window) Graphical user interface, application

Description automatically generated

Figure 53: Pop up window that appears.

Input: the folder selected in the image above.

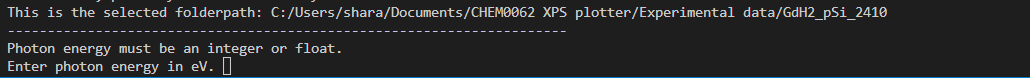
Terminal prompt: 

Figure 54: Terminal prompt asking for the photon energy.

Input: 2410

(Note: the names of all the folders provided in the zip file end in either: “2410” or “7230”. These numbers are the photon energies (eV) at which the scans were run.)

Terminal prompt:

A picture containing graphical user interface

Description automatically generated

Figure 55: Terminal prompt asking for the standard deviation threshold value.

Input: 100

Terminal prompt:

Text

Description automatically generated

Figure 56: Terminal prompt asking for output plot type choice.

Input: b

Outputs:

* The following terminal message. Text

  Description automatically generated

Figure 57: Terminal prompt with the final messages for the user which includes suggestions on how the user may want to adapt their input choices in the future.

* The following plots. They were saved to the folder which was selected earlier. All these figures are shown in one combined Figure 58 below:

A picture containing shape

Description automatically generated

Figure 58: An image showing all the plots produced in Usage Example 1.

* The standard deviation text file and the excluded/included datapoints text files.

Graphical user interface, text

Description automatically generated

Figure 59: The std\_file.txt output. It prints: (file name, datapoint number, line in the respective .dat file where the datapoint is, standard deviation).

Graphical user interface, application, table

Description automatically generated with medium confidence

Text

Description automatically generated with medium confidence

Figure 60: Snapshots of excluded\_datapoints\_file.txt and included\_datapoints\_file.txt.

(To demonstrate how all scans are included in the included\_datapoint\_file I have also provided an IDE version of this text file below since it is not clear in the image above.)

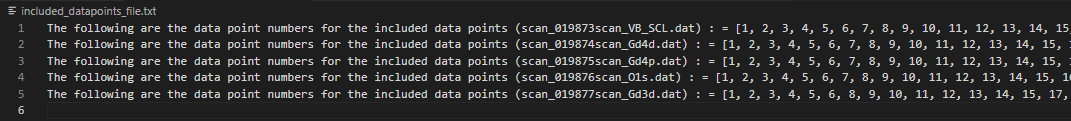


Figure 61: IDE version of the included\_datapoints\_file.txt snapshot

After running the program, all these files appeared in the inputted folder as seen below.

Graphical user interface, application

Description automatically generated

Figure 62: What the input folder should look like after these operations have been carried out.

**Usage example 2 – production of only individual plots OR only final plots:**

To reduce repetition, the process is not shown again. The only things that differ would be the input letter “i” or “f” into the terminal prompt in Figure 56. If “i” was selected, then all the plots except the final one in Figure 58 would be produced. If “f” was selected, then only the final combined plot in Figure 58 would be produced.

## Self-reflection

Before starting this module, I had some familiarity with Python syntax as I previously started a Codecademy Python 2 course whilst I was at school. However, I did not complete it and had never completed a project of my own before. I decided to do this project because I struggled with the UV mini project in term 1. I was not able to successfully produce the plots without NumPy and struggled to read/write files in Python. I wanted to work on my understanding of these things as they are important for data analytics which I am interested in.

I feel as though my programming abilities and understanding have improved significantly over the last two terms. Specifically, I became more comfortable with writing loops and functions as well as writing docstrings and comments throughout my code. Furthermore, I think my new understanding of error trapping using try/except statements could be particularly useful if I were to do another programming project in the future.

One thing I found unexpected about my project was the steep learning curve I faced when figuring out how to create a folder selection pop up window and writing file paths in my main loop in a way which worked. I overcame this difficulty by researching around the glob2 module and tkinter and then adapting what I found on Stack Overflow so that it fit my project. Furthermore, I learnt the importance of variable locations in a script as I had to experiment with them to get my script to run many times. I feel as though I now have an improved understanding of scripts need to be indented for them to work.

If I had more time, I would have found a way to detect files with 1 sweep and tell the user which only have one sweep. I did mention in the terminal prompts that more data should be collected for files with 1 sweep. But it would be useful to detect which files only have 1 sweep for a user since they may have many files and so it could be time consuming to check. I tried to do this however the nesting of loops within my main loops makes this difficult. If it is printed inside the line reading loop, then the message is printed [number of lines in the file] times which is too many and repetitive. If it is printed outside this loop but inside the open and read file path loop, then a NameError appears. If it is printed at the bottom of the program, then a message is only printed for the last file with 1 sweep in the folder – as the variable is overwritten with each iteration of the file reading loop. I am currently unsure of how I would fix this but I would research further if I had more time.

Furthermore, I would also find a way of colour coding each section of the final plot and adding a legend which states the corresponding filename. On top of this, I would find a way of fixing the percentage threshold function and reintroducing it into the program so that the user has a greater variety of thresholding options.

Lastly, with more time, I would adapt this program so that it could be useful for other types of spectra such as NMR spectra. If adapted to NMR, the standard deviation threshold could be replaced by and intensity threshold which could remove very small trace peaks as they are often impurities. This could allow integration values to be interpreted faster and more easily by users, given that these are also printed by the script.

## Code Listing

'''

XPS Spectra Plotter with Custom Error Datapoint Exclusion.

This script allows the user to create average plots from experimental X-ray photoelectron spectroscopy (XPS) data.

The user can import a folder containing (.dat) files for the different surface component scans.

The imported folder must contain this script.

The user will define a standard deviation-based error threshold.

The standard deviation threshold value must be a positive value (float or integer).

Datapoints above the user-specified standard deviation value will be excluded from plots.

This program can produce individual plots for each of the surface component scan files.

This program can produce a final combined plot which combines all the surface component plots.

The user can choose whether they would like to produce individual plots, combined plots or both.

A file containing the standard deviations of all datapoints for all the files in the folder will be generated.

Files containing the excluded datapoints and included datapoints for each individual plot will be created.

It is required that:

    \* the experimental files are in .dat file format.

    \* this script is saved in the folder which contains the .dat files for which plots will be created.

    \* The first column in the files correspond to kinetic energy in eV.

    \* The delimiter must be a tab.

    \* Each scan file contains 1 or more sweeps and a "sum of all sweeps" column if there are multiple sweeps.

    \* Each line of each file has a tab at the end (this is the standard nature of the experimental .dat files)

It is required that the user installs:

    \* Anaconda (version 2022.05)

    \* Python (version 3.9.12)

    \* numpy (version 1.21.5)

    \* matplotlib (version 3.5.1)

This should be done within the python environment where this script is currently being run.

This file contains the following functions:

    \* standard\_deviation\_threshold - removes datapoints which are above the specified standard deviation from plots

'''

# General imports

import os  # lets vscode interact with the operating system

import matplotlib.pyplot as plt  # for plot creation

import numpy as np  # for mathematical calulations

# The following imports will be used to allow the user to import files

# This has been adapted from the following references:

# Stack Overflow, https://stackoverflow.com/questions/18262293/how-to-open-every-file-in-a-folder, (accessed 15/01/23).

# Stack Overflow, https://stackoverflow.com/questions/73003417/input-folder-path-and-return-a-list-of-files-on-tkinter, (accessed 18/03/23).

# glob is a module that returns all filepaths that follow a specific pattern

# In this program, glob will be used to return files of the .dat format

# tkinter is Python's standard Graphical User Interface (GUI) creation library

# In this program, tkinter will be used to create the folder selection pop-up

# tkinter has a prebuilt dialogue window to access files within its filedialog module

# The askdirectory() function within the filedialog module allows for the selection of one folder or file

# In this program, this function will be used to select a folder containing the input files

from tkinter import Tk

import tkinter

from tkinter.filedialog import askdirectory

import glob

# Initial message to the user explaining this program

print("You will be prompted to select the folder which contains the .dat files you would like to make plots for.")

print("(Note: files in subfolder will not be selected.)")

initial\_message = input("To continue, press 'y' and then enter ")

# Creation of folder selection pop-up

if initial\_message == "y":

    # If the user continues after the inital message: dialogue box appears, asks user to select folder and returns the path

    folderpath = askdirectory(title='Select Folder')

else:

    # Message to the user in case they enter the wrong input

    print("Re-run the code and make sure to just enter the letter 'y' with no spaces or apostrophes - just the letter.")

    exit()  # Ends the execution of the program so the user will re-run it

# Statement lets the user check that they imported the right folder

print("This is the selected folderpath: " + folderpath)

# Set up of x-axis and y-axis lists for the final combined plot (of all files in the selected folder)

fp\_xaxis = []

fp\_yaxis = []

# Definition of variables used to read lines of code in the loop

# A tab is the standard delimiter of experimental XPS (.dat) files

delimit = "\t"

numhead = 1  # Number of header-lines in the file which will not be read as data as they contain column headers

# Section divider printed in terminal for clarity

print("----------------------------------------------------------------------")

# The user inputs the photon energy at which the scan was run

print("Photon energy must be an integer or float.")

photon\_energy = input("Enter photon energy in eV. ")

# Section divider printed in terminal for clarity

print("----------------------------------------------------------------------")

# The user chooses the threshold standard deviation

print("What standard deviation would you like to specify?")

std\_input = input("Just enter the number without any units. It must be positive. ")

# Error trap

# Creation of an error message if a number is not entered for photon energy or standard deviation value

# Creation of an error message if the number entered is negative

user\_input\_numbers = [photon\_energy, std\_input]

for entered\_value in user\_input\_numbers:

    try:

        entered\_value = float(entered\_value) # Ensuring that the input is a number

        if entered\_value >= 0: # Ensuring the number is positive or zero

            pass

        else:

            print("The number you entered for photon energy or standard deviation is negative.")

            print("Re-run the program and try again.")

            exit()

    except ValueError:

        print ("You did not enter a number for either photon energy or standard deviation.")

        print("Re-run the program and try again.")

        exit()

# Section divider printed in terminal for clarity

print("----------------------------------------------------------------------")

# The user chooses what kind of output they want

print("You can choose the type of output plots produced.")

print("If you would like to produce only individual plots for each file, enter 'i'.")

print("If you would like to produce only the final combined plot which uses all files in the folder, enter 'f'.")

output\_plot\_choice = input("If you would like to produce both, enter 'b'.")

if output\_plot\_choice == "i" or output\_plot\_choice == "b" or output\_plot\_choice == "f":

    pass

elif output\_plot\_choice == "I" or output\_plot\_choice == "B" or output\_plot\_choice == "F":

    pass

else:

    print("You did not enter an appropriate letter when choosing output plot types.")

    print("Re-run the program and try again.")

    exit()  # Ends the execution of the program so the user will re-run it

# Section divider printed in terminal for clarity

print("----------------------------------------------------------------------")

# Standard deviation, excluded datapoint and included datapoint files which are produced must be reset at the start of every run

# This is in case the the user tries to run the program twice without deleting the files produced by the first run

# This prevents the program from appending the information from the second run onto that of the first

output\_file = ["std\_file.txt", "excluded\_datapoints\_file.txt",

                "included\_datapoints\_file.txt"]

for txt\_file in output\_file:

    if os.path.exists(txt\_file):  # Existance of the files from previous runs is checked

        os.remove(txt\_file)  # Old files are deleted

    else:

        pass

# Defining a function

def standard\_deviation\_threshold():

    '''

    This function removes datapoints with standard deviations above the user-chosen one from the plots.

    Parameters

    ----------

    point\_standard\_deviation : float

        Standard deviation of the datapoint

    std\_input : string

        Standard deviation value chosen by the user

    axes: list[list]

        List of names of all axes

    xaxis: list

        x-axis values for individual file plots

    yaxis: list

        y-axis values for individual file plots

    fp\_xaxis: list

        x-axis values for final combined plot

    fp\_yaxis: list

        y-axis values for final combined plot

    binding\_energy: float

        Binding energy of the datapoint

    mean\_intensity: float

        Mean intensity of the datapoint

    excluded\_point\_numbers: list

        List of excluded datapoint numbers

    datapoint\_number: int

        Datapoint number

    included\_point\_numbers : list

        List of included datapoint numbers

    Returns

    ----------

    The function does not return a value but, instead, updates the axes list

    '''

    # Reference: https://www.oreilly.com/library/view/python-cookbook/0596001673/ch17s02.html

    # This reference helped me redirect the NameError message when user selects the wrong folder

    # try/except statements allow common errors to be redirected to a custom backup action

    try:

        if point\_standard\_deviation > float(std\_input):

            axes = [xaxis, yaxis, fp\_xaxis, fp\_yaxis]

            for xax in axes[0::2]:

                xax.remove(binding\_energy)  # Removal of x-axis values from plots

            for yax in axes[1::2]:

                yax.remove(mean\_intensity)  # Removal of y-axis values from plots

            # Excluded datapoint number stored in a list

            excluded\_point\_numbers.append(datapoint\_number)

        else:

            # Included datapoint number stored in a list

            included\_point\_numbers.append(datapoint\_number)

    except NameError:

        # Section divider printed in terminal for clarity

        print("----------------------------------------------------------------------")

        # Substitute for the NameError message

        print("You selected the wrong folder. It must be the folder that this script is saved in.")

        exit() # Ends the execution of the program so the user will re-run it

# Creation of the major loop which gets the filepaths for all the files in the selected folder

# glob is used to return files ending with '.dat'

for count, filename in enumerate(glob.glob("\*.dat")):

    with open(os.path.join(os.getcwd(), filename), 'r') as f:

        # Message to the user so they know the program is running

        if output\_plot\_choice != "f" and output\_plot\_choice != "F":

            print("The plot for " + filename + " is currently being created.")

        else: print("Loading...") # If only the final combined plot output is chosen.

    filepath = os.path.join(folderpath, filename)

    # Reference: the UV mini project from the CHEM0062 Moodle page

    # Creation of an average xps plot for all .dat files in the folder

    # Individual plots for each .dat file will be plotted as well as a plot which combines all the individual plots

    # Plots will exclude datapoints - exclusion is based on the threshold set by the user

    if os.path.exists(filepath):

        with open(filepath, 'r') as infile:

            # Definition of list variables

            xaxis = []

            yaxis = []

            standard\_deviations\_list = []

            excluded\_point\_numbers = []

            included\_point\_numbers = []

            intensities\_in\_a\_line\_list = []

            # The document is read line-by-line

            for x in range(numhead+1):

                line = infile.readline()

            line\_number = 1  # Setting up the line counter

            while line:

                # Saving the current line number

                line\_number += 1

                # Save current datapoint number

                datapoint\_number = (line\_number - 1)

                # Creating a list containing all the values in a line

                # Error trap: common alternative delimiters that may be present are replaced with the standard delimiter

                alternative\_delimiter = [",", ";", " "]

                for alternate in alternative\_delimiter:

                    line = line.replace(alternate, delimit)

                datapoint = line.split(delimit)

                # Appending energies to the x-axis

                kinetic\_energy = float(datapoint[0])

                binding\_energy = float(photon\_energy) - kinetic\_energy

                xaxis.append(binding\_energy)

                fp\_xaxis.append(binding\_energy)

                # Setting a variable for the index of the last column

                # '-2' on the first line is because we want the index and there is a tab at the end of each line that needs to be excluded

                intensity\_sum\_index = (int(len(datapoint)) - 2)

                num\_sweeps = intensity\_sum\_index - 1

                # Finding the mean intensity and appending it to the y-axis

                if num\_sweeps > 0:

                    # Average intensity of all sweeps

                    mean\_intensity = float(

                        datapoint[intensity\_sum\_index])/(num\_sweeps)

                elif num\_sweeps == 0:

                    # To account for files with only 1 sweep which have no sum column

                    mean\_intensity = float(datapoint[intensity\_sum\_index])/(1)

                else:

                    print(

                        "Something has gone wrong with the mean intensity calculation for this file")

                    continue

                yaxis.append(mean\_intensity)

                fp\_yaxis.append(mean\_intensity)

                # Making a list of intensities for a line (excluding the energy and intensity sum values)

                # print(len(datapoint)) --> test confirmed that the length of a datapoint for a file with only 1 sweep is 3

                intensities\_in\_a\_line\_list = []

                if len(datapoint) > 3:

                    # Splitting off the energy and sum columns from the lines

                    intensities\_in\_a\_line\_list = datapoint[1:intensity\_sum\_index]

                elif len(datapoint) == 3:

                    # To account for files with only 1 sweep which have no sum column

                    intensities\_in\_a\_line\_list = datapoint[1:(

                        intensity\_sum\_index+1)]

                else:

                    print("Something has gone wrong with the len(datapoint)")

                # Calculation of standard deviation of the intensities for each datapoint

                # Reference: introduction to python pt1 (from the CHEM0062 Moodle page)

                point\_variance = 0

                for intensity in intensities\_in\_a\_line\_list:

                    point\_variance = point\_variance + \

                        ((float(intensity) - mean\_intensity)

                            \*(float(intensity) - mean\_intensity))

                point\_variance = point\_variance/len(intensities\_in\_a\_line\_list)

                point\_standard\_deviation = np.sqrt(point\_variance)

                standard\_deviations\_list.append(point\_standard\_deviation)

                # Printing standard deviations to a separate file

                # This file prints: (filename, datapoint number, line in the respective file where the datapoint is, standard deviation)

                # https://howtodoinjava.com/python-examples/python-print-to-file/

                # 'a+' appends each std to the file line by line (rather than 'w' which overwrites each datapoint std after every line)

                standard\_deviation\_file = open("std\_file.txt", "a+")

                print((filename, datapoint\_number, line\_number,

                        point\_standard\_deviation), file=standard\_deviation\_file)

                standard\_deviation\_file.close()

                # Error threshold application based on user choice of threshold type

                standard\_deviation\_threshold()

                line = infile.readline()  # Closing the loop

            # Creation of individual scatter plots for each .dat file

            # Dot markers are used and marker size are set

            plt.scatter(xaxis, yaxis, marker="o", s=0.5)

            plt.xlabel("Binding energy / eV")

            plt.ylabel("Average sweep intensity / a.u.")

            # Saving the figure to the folder where the .dat files are located

            # This means the user does not need to keep closing pop-up figure windows as with "plt.show()"

            # Reference: https://stackoverflow.com/questions/17788685/python-saving-multiple-figures-into-one-pdf-file

            # Removal of .dat filename suffix - figure can only save as a .png file if suffix is removed

            filename\_stripped = filename.replace(".dat", "")

            if output\_plot\_choice != "f" and output\_plot\_choice != "F":

                plt.savefig("Figure for " + str(filename\_stripped))

                plt.close()

            # Creation of the excluded and included datapoint files

            # Note: a nested loop was not created for this section as I had trouble with the TextIOWrapper format (see the report)

            excluded\_points\_file = open("excluded\_datapoints\_file.txt", "a+")

            print(("The following are the data point numbers for the excluded data points (" +

                    str(filename) + ") : " + str(excluded\_point\_numbers)), file=excluded\_points\_file)

            excluded\_points\_file.close()

            included\_points\_file = open("included\_datapoints\_file.txt", "a+")

            print(("The following are the data point numbers for the included data points (" +

                    str(filename) + ") : = " + str(included\_point\_numbers)), file=included\_points\_file)

            included\_points\_file.close()

# Error threshold application based on user choice of threshold type

standard\_deviation\_threshold()

# Creation of the final combined plot

# Dot markers and marker size are set

plt.scatter(fp\_xaxis, fp\_yaxis, marker="o", s=0.5)

plt.xlabel("Binding energy / eV")

plt.ylabel("Average sweep intensity / a.u.")

plt.savefig("Final combined plot")

plt.close()

# Removing final combined plot if it was not asked for by the user

if output\_plot\_choice == "i" or output\_file == "I":

    os.remove("Final combined plot.png") # Deletes the final plot if only individual plots are wanted.

elif output\_plot\_choice == "b" or output\_plot\_choice == "B":

    pass

elif output\_plot\_choice == "f" or output\_plot\_choice == "F":

    pass

# Section divider printed in terminal for clarity

print("----------------------------------------------------------------------")

# Final statements to aid the users' understanding of their results and aid future uses of this script.

print("Try out different standard deviations to see what suits your needs.")

print("Note that the highest standard deviation in the set was " + str(max(standard\_deviations\_list)))

print("Also note that plots of files with only 1 sweep cannot be improved using this program.")

print("This is because this program requires more than 1 sweep to calculate error values.")

print("Improve such plots by collecting more experimental data for corresponding files.")

print("Note: it is recommended you output both plots when experimenting with standard deviation values.")

## References

1. Diamond, <https://www.diamond.ac.uk/Instruments/Techniques/Spectroscopy/XPS.html>, (accessed March 2023).
2. Github, <https://github.com/NikolasEnt/XPS-visualizer-and-converter>, (accessed March 2023).
3. LG4X, <https://github.com/hidecode221b/LG4X>, (accessed March 2023).