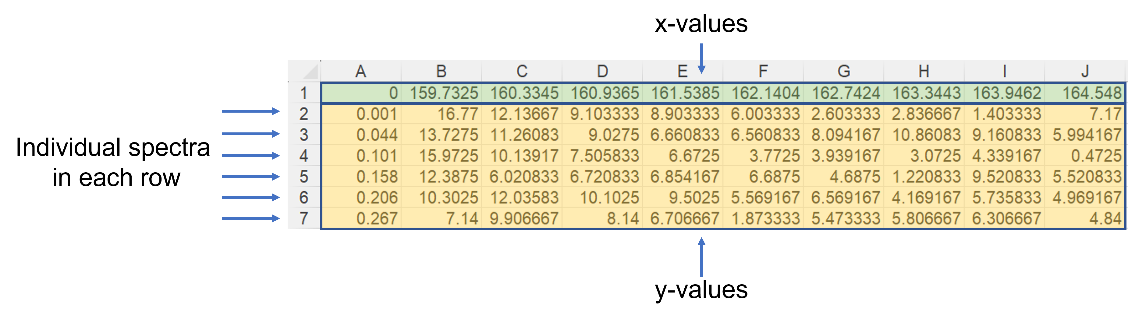
**General Information**

* Please install a dedicated IDE tool like PyCharm or Spyder or Visual Studio Code to run the python automated peak fitting module.
* Please ensure that the dataset is contained within a .csv file, and that the data format is correct, i.e. x-values as column headers, corresponding y-values as entries in each row, each row representing a single spectra/graph/plot as shown below:  
  
* For automated fitting of a single peak across multiple spectra, simply follow the instructions prompted within the GuidedSinglePeakFitting.py submodule. A more detailed explanation about the prompts will be given in this document.
* The GuidedSinglePeakFitting.py submodule should theoretically be able to handle all datasets containing a single peak as specified in the second bullet point above, regardless of the analytical method used to obtain the data. In other words, the submodule is agnostic to the type of data (GC/ Raman/ NMR/ UV-Vis, etc.) being fitted if the data structure is appropriate.
* For more complicated fittings and deconvolutions for regions with overlapping peaks or multiple peaks in a single region, a detailed example (without prompts) is given by the Raman vinyl and p-xylene peak fitting exercise. In the context of this peak fitting module, a region refers to a segment of the spectrum within which the peak to be fitted lies.
* The automated peak fitting library requires the following libraries:

1. Numpy
2. Scipy
3. Lmfit
4. Pandas
5. Matplotlib
6. Sys
7. Sklearn

* The peak fitting modules can be customized to fit other regions with different requirements by editing the following submodules:
  1. Residuals.py
     + This module is critical in defining the objective function to be minimized, using lineshapes from the library lmfit, depending on the requirements of the regions to be fitted.
  2. CurveFitting.py
     + This module contains a function which executes the curve fitting. The functions within the module can be modified based on the type of fitting parameter that the user wants stored and reported.
     + The function uses the Levenberg-Marquardt method to minimize the least-squares errors of non-linear functions. Non-linear least squares (NLLS) methods require initial guesses. The user can modify the fitting method to suit their needs, and can refer to the lmfit documentation for more information.
  3. Excel file containing initial guesses of parameters for the region to be read by Parameters.py.
     + Depending on the region requirements and the type of objective function defined by the use, the excel file containing the initial guesses of paraters for the region has to be modified to be appropriate for fitting. An example of the excel file is shown below:

A screenshot of a computer

Description automatically generated with low confidence

* 1. Excel file containing the column indices which defines the leftmost and right bounds of the region.
     + Depending on the region of interest, the user must define the left most and right most column indices to truncate the dataframe.
     + The column indices are different from the actual x-values in the dataframe. Column indices are integers which represents the column positions of the x-values.
     + If the user only knows the bounds of the region in terms of the x-values in which to truncate the dataframe, please use the find\_nearest function from the RegionDataFrame.py submodule to find the column index which corresponds with the x-value bounds.

**GuidedSinglePeakFitting.py**

The primary aim of the GuidedSinglePeakFitting.py submodule is to allow non-programmers to execute an automated peak fitting process for a single peak via prompts that is highly user friendly and does not require a priori programming knowledge. The script takes in the answers from the user as instructions that will eventually allow for the automated peak fitting of a single peak across multiple spectra. When executed correctly, the script will save all relevant fitted parameters and plots. Please take note that the quality and accuracy of peak fit is highly dependent on the user’s input. The user must know the region of interest and the relevant guesses for initial fit parameters to obtain an appropriate fit. The user is wholly responsible for the results of the peak fitting exercise. If a peak fitting is unsuccessful or unsatisfactory, simply press CTRL+C or type in ‘n’ as input when prompted to terminate the script. The example dataset used is named: ‘FA 2-18 Br-I\_timed.csv’.

This submodule contains a total of 10 prompts, and the prompts will be detailed and with their subsequent actions depending on the inputs as follow:

1. *Is your data in the form of x-values (wavenumbers or wavelengths) as the column names, and y-values ' (intensity, or absorption) as the values of a table, with each row representing a single spectrum? [y/n]*
   1. If the answer is ‘y’, the program proceeds to ask prompt 2.
   2. If the answer is ‘n’, the program prints out an error comment and exits the script.
2. *Please fill in the file name (including extension) of the .csv file that contains your data:*
   1. Please include the file name that is contained within a .csv file. Please ensure that the file name is typed in correctly without errors. If the file name is input correctly, and the data is of the appropriate structure, the first spectrum will be plotted, i.e. plot the first row of y-values against the column headers which are the x-values. The plot is subsequently saved, and the program proceeds to ask prompt 3.
   2. If the file name is input wrongly, the program prints out an error comment and exits the script.
3. *The 1st spectrum from your dataset was plotted. Does the plot look correct? [y/n]*
   1. If the answer is ‘y’, the program proceeds to assert that the user deemed the 1st plot correct and continues with 2 sub-prompts:
      1. *What is the left-most x-axis value for the region of interest?*
         1. User must input an integer which correctly specifies the left-most x-value of the region of interest.
      2. *What is the right-most x-axis value for the region of interest?*
         1. User must input an integer which correctly specifies the right-most x-value of the region of interest.

Once the left-most and right-most x-values are input, the script identifies the actual boundary x-values contain within the dataset and plots the region of interest in the 1st spectrum along with the entire spectrum for comparison. The plot image is subsequently saved, and the script will begin prompt number 4.

* 1. If the answer is ‘n’, the program prints out an error comment and exits the script.

1. *The plots before and after dataframe slicing were plotted. Do the plots look correct? [y/n]*
   1. If the answer is ‘y’, the program proceeds to ask prompt 5.
   2. If the answer is ‘n’, the program prints out an error comment and exits the script.
2. *Do you need to do baseline subtraction for the region of interest? [y/n]*
   1. If the answer is ‘y’, the program proceeds to do baseline subtraction for the region of interest. A side-by-side comparison the region of interest before and after baseline subtraction is plotted. The plot image is subsequently saved, and the program proceeds to prompt the user to input the appropriate initial guesses for peak fitting. Once the initial guesses are entered, a comparison between the Lorentzian and the Gaussian fit will be plotted and saved. The program then proceeds to prompt 7.
   2. If the answer is ‘n’, the program proceeds without baseline subtraction to prompt the user to input the appropriate initial guesses for peak fitting. Once the initial guesses are entered, a comparison between the Lorentzian and the Gaussian fit will be plotted and saved. The program then proceeds to prompt 7.
3. *The spectra before and after baseline correction are plotted. Do the plots look correct? [y/n]*
   1. If the answer is ‘y’, the program proceeds to prompt the user to input the appropriate initial guesses for peak fitting. Once the initial guesses are entered, a comparison between the Lorentzian and the Gaussian fit will be plotted and saved. The program then proceeds to prompt 7.
   2. If the answer is ‘n’, the program prints out an error comment and exits the script.
4. *Do the peak fittings look correct? [y/n]*
   1. If the answer is ‘y’, the program proceeds to ask prompt 8.
   2. If the answer is ‘n’, the program prints out an error comment and exits the script.
5. *Which lineshape fits the peak the best? A higher R squared value indicates a better fit. If the lorentzian function form is a better fit, please key in 0. If the gaussian function form is a better fit, please key in 1.*
   1. If the answer is ‘0’, the program proceeds to use the Lorentzian functional form for all subsequent peak fittings. Script proceeds to prompt 9.
   2. If the answer is ‘1’, the program proceeds to use the Gaussian functional form for all subsequent peak fittings. Script proceeds to prompt 9.
6. *Proceed to do peak fitting for all spectra in the dataset? [y/n]*
   1. If the answer is ‘y’, the program proceeds to fit all spectra within a dataset, saves all fitted parameters for every spectrum in a .csv file, as well as the fitted parameter’s summary statistics in another .csv file. Program proceeds to prompt 10.
   2. If the answer is ‘n’, the program terminates.
7. *Proceed to plot fitted parameters against spectra index? [y/n]*
   1. If the answer is ‘y’, the program proceeds to plot all fitted parameters against summary statistics. Value of individual fitted parameters against spectrum index will be plotted in a scatter plot in the first row, and a box plot in the second row. The summary plot image will subsequently be saved. The script has run to the end and terminates.
   2. If the answer is ‘n’, the program terminates.

**Data Structure and Procedure for Raman vinyl and p-xylene peak fitting**

This is a more complicated example on the peak fitting of different regions of Raman spectral data from a previous work. The goal of the Raman spectra peak fitting is to convert Raman spectra data into conversion data, by monitoring the AUC of the vinyl peak over time. The procedure of the Raman vinyl and p-xylene peak fitting to reach this goal consists of the following:

1. The Raman spectra data structure contains the x-values as column headers and y-values as entries in each row. In addition, there are 2 adidtional columns to indicate the condition to which the spectrum belongs, and the original spectra index where the spectrum was extracted from. A single .csv file contains data from a single residence time, and there are a total of 5 residence times: 0 min, 30 min, 60 min, 90 min, 120 min, and a total of 6 .csv files, with 0 min being repeated:  
   A picture containing table

   Description automatically generated

Table

Description automatically generated

1. For every Raman spectra, identify the vinyl region and the p-xylene region:
2. For each region, conduct a baseline subtraction:

A picture containing chart

Description automatically generated

Chart, line chart, histogram

Description automatically generated

1. Conduct curve fittings for the respective regions, deconvoluting overlapping peaks in the process

Chart, histogram

Description automatically generated

Chart

Description automatically generated with medium confidence

1. Divide the AUC of the vinyl peak by the AUC of the p-xylene peak as a normalization step to obtain an AUC ratio.
2. Average the AUC ratio across all spectra belonging to a single condition to obtain the mean AUC ratio and the standard deviation of the AUC ratio:

Table

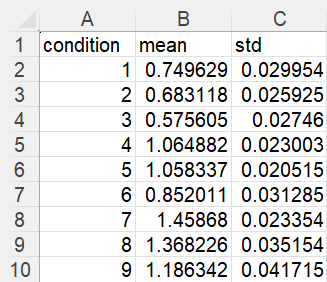
Description automatically generated

1. Calculate the conversion from the mean AUC ratio, and propagate the error using the standard deviation of the AUC ratio.
2. Plot the conversion of each condition over residence time:

A complete explanation of the study and the rationale for the procedure mentioned above is included in the manuscript at the DOI: 10.26434/chemrxiv-2022-tlz53

**Scripts and Submodules used for the Raman Example**

The first script to execute for this example is the AutomatedRatioExtraction.py script. The goal of this script is to convert raw spectra dataas shown in the previous section into a dataframe containing only the mean and standard deviation of the AUC ratios as shown below:



The script requires .xlsx files containing the initial guesses for the peak fitting of the vinyl and p-xylene regions like the example for the vinyl region shown below:

A screenshot of a computer

Description automatically generated with low confidence

The script iterates through all raman .csv files as shown in the previous section, and slices the dataframe into the vinyl and p-xyxlene regions using the region\_df\_slice function and the column indices specified by the excel file:

Text, table

Description automatically generated

The script then interative fits the vinyl and p-xylene regions of each spectrum using the appropriate residuals and curve fitting functions defined in Residuals.py and CurveFitting.py, respectively. After peak fitting, the fitted parameters will be stored in a dataframe. A module specific to this Raman example: Consolidate.py was written such that only peak fittings with for both regions in a spectrum is retained, and the mean and standard deviation of the AUC ratios are calculated. Another example specific module ConversionCalculation.py takes the mean and standard deviation of the mean AUC to calculate the conversion and conversion errors for each condition across all residence times and store them in the .csv files ‘df\_conversion.csv’ and ‘df\_error.csv’ respectively, as shown below:



Table

Description automatically generated



Table, Excel

Description automatically generated

Lastly, the example specific module ConversionPlot.py plots all conversion for the 9 conditions in a single page:

A screenshot of a computer

Description automatically generated with low confidence

**General Description of all Submodules**

Plotting.py is a submodule which contains functions for the plotting of graphs. The submodule contains the following functions:

1. simple\_line\_plot
2. subplot\_2\_by\_1
3. baseline\_subtraction\_plot
4. fitting\_comparison

Residuals.py is a submodule which contains functions which calculates the residuals between the fitted y-values and the actual y-values to be fitted. The submodule contains the following functions:

1. residuals\_lorentzian
2. residuals\_gaussian
3. residuals\_vinyl
4. residuals\_pxylene

CurveFitting.py is a submodule containing functions executes the curve fitting process. The submodule contains the following functions:

1. lorentzian\_curve\_fit
2. gaussian\_curve\_fit
3. curve\_fit

BaselineSubtractionFunction.py is a submodule containing the baseline subtraction function. The submodule contains the following function:

1. baseline\_subtraction\_function

RegionDataFrame.py is a submodule which contains functions to allow for easy slicing of dataframes. The submodule contains the following function:

1. find\_nearest
2. region\_df\_slice

Parameters.py is a submodule which contains a function to allow for easy definition of initial guesses for fitting parameters. The submodule contains the following function:

1. define\_region\_parameters

IterativeFitting.py is a submodule which contains a function to allow for easy iterative fitting of multiple spectra. The submodule contains the following function:

1. iterative\_fitting

Consolidate.py is a submodule which contains a function which calculates the mean and standard deviation of the AUC of multiple spectra associated with their respective conditions. The submodule contains the following function:

1. aggregate\_ratio

AutomatedRatioExtraction.py is a submodule which does not define any new function. The submodule is responsible for iterating through .csv files of extracted spectra (e.g. ‘df\_t0.csv’), fitting every spectrum contained within the files and saving the mean and the standard deviation of the ratio of the vinyl AUC to the p-xylene AUC into a .csv files with ‘\_ratio’ to the end of the original file name.

ConversionCalculation.py is a submodule which calculates the conversion and the error bars from all .csv files containing the mean and standard deviation, returning 2 .csv files as a result: 'df\_conversion.csv' and 'df\_error.csv'. The submodule contains the functions:

1. create\_mean\_df
2. create\_std\_df
3. create\_ufloat\_df
4. calc\_conv\_and\_error
5. conversion\_and\_error

ConversionPlot.py is a submodule that plots the conversion plot of 9 conditions in a 3 by 3 subplot. This submodule does not contain any functions.

**Plotting.py**

simple\_line\_plot(x, y, x\_label, y\_label, title, save\_name)

|  |  |
| --- | --- |
| **Parameters:** | x: List or numpy array for x values.  y: List or numpy array for y values.  x\_label: String which labels the x-axis.  y\_label: String which labels the y-axis.  title: String which represents the title.  save\_name: String which contains the file name with the file format. |
| **Returns:** | None. Plots the graph. |
| **Description:** | A simple function for graph plotting. x and y are arrays containing the x and y values, respectively. x and y must be of the same length for plotting purposes. |

**Plotting.py**

subplot\_2\_by\_1(x1, y1, x2, y2, x\_label, y\_label, title1, title2, save\_name)

|  |  |
| --- | --- |
| **Parameters:** | x1: List or numpy array for x values for the first plot.  y1: List or numpy array for y values for the first plot.  x2: List or numpy array for x values for the second plot.  y2: List or numpy array for y values for the second plot.  x\_label: String which labels the x-axis.  y\_label: String which labels the y-axis.  title1: String which represents the title for the first plot.  title2: String which represents the title for the second plot.  save\_name: String which contains the file name with the file format. |
| **Returns:** | None. Plots the graph. |
| **Description:** | A simple function for plotting a graph side-by-side for comparison purposes. x1 and y1 are x and y-values for the first plot, while x2 and y2 are x and y-values for the second plot. The first plot is plotted on the left, while the second plot is plotted on the right.  x1 and y1 are arrays containing the x and y values, respectively.  x1 and y1 must be of the same length for plotting purposes.  The same applies for x2 and y2. |

**Plotting.py**

baseline\_subtraction\_plot(x, y\_original, y\_baseline, y\_subtracted, x\_label, y\_label, title1, title2, save\_name)

|  |  |
| --- | --- |
| **Parameters:** | x: List or numpy array for x values for the region of interest.  y\_original: List or numpy array for y values for the original y-values.  y\_baseline: List or numpy array for y values for the baseline.  y\_subtracted: List or numpy array for y values after baseline subtraction.  x\_label: String which labels the x-axis.  y\_label: String which labels the y-axis.  title1: String which represents the title for the first plot.  title2: String which represents the title for the second plot.  save\_name: String which contains the file name with the file format. |
| **Returns:** | None. Plots the graph. |
| **Description:** | A simple function for plotting a graph side-by-side for comparison purposes. x is the array containing x-values for the region of interest.  The first plot contains the plot of the region of interest before baseline correction, and a straight line representing the baseline.  The second plot contains the plot of the region of interest after baseline correction. |

**Plotting.py**

fitting\_comparison(x, y, best\_fit1, best\_fit2, fit\_params1, fit\_params2, x\_label, y\_label, title1, title2, save\_name)

|  |  |
| --- | --- |
| **Parameters:** | x: List or numpy array for x values for the region of interest.  y: List or numpy array for y values for the region of interest.  best\_fit1: List or numpy array for best-fit y-values using the 1st functional form.  best\_fit2: List or numpy array for best-fit y-values using the 2nd functional form.  fit\_params1: Dictionary containing the fitted parameters using the 1st functional form.  fit\_params2: Dictionary containing the fitted parameters using the 2nd functional form.  x\_label: String which labels the x-axis.  y\_label: String which labels the y-axis.  title1: String which represents the title for the first plot.  title2: String which represents the title for the second plot.  save\_name: String which contains the file name with the file format. |
| **Returns:** | None. Plots the graph. |
| **Description:** | A function to plot fitted peaks side by side for a comparison. |

**Residuals.py**

residuals\_lorentzian(parameters, x, y)

|  |  |
| --- | --- |
| **Parameters:** | parameters: An lmfit Parameters object defined by the user.  x: Numpy array containing the wavenumber values.  y: Numpy array containing the y-values (intensities) after baseline subtraction. |
| **Returns:** | residuals - Numpy array containing the residuals between the fitted model’s y-values and the baseline subtracted y-values. |
| **Description:** | A function which calculates the residuals between a set of fitted data points and the original data points. This function will be called using the lmfit Minimizer object as an objective function to be minimized.  This residual function calculates the residuals based on fitting a Lorentizan lineshape to the single peak.  Parameter object must first be initialized and the parameters within the object must be pre-defined using a best guess. |

**Residuals.py**

residuals\_gaussian(parameters, x, y)

|  |  |
| --- | --- |
| **Parameters:** | parameters: An lmfit Parameters object defined by the user.  x: Numpy array containing the x-values.  y: Numpy array containing the y-values. |
| **Returns:** | residuals - Numpy array containing the residuals between the fitted model’s y-values and the actual y-values. |
| **Description:** | A function which calculates the residuals between a set of fitted data points and the original data points. This function will be called using the lmfit Minimizer object as an objective function to be minimized.  This residual function calculates the residuals based on fitting a Gaussian lineshape to the single peak.  Parameter object must first be initialized and the parameters within the object must be pre-defined using a best guess. |

**Residuals.py**

residuals\_vinyl(parameters, x, y)

|  |  |
| --- | --- |
| **Parameters:** | parameters: An lmfit Parameters object defined by the user.  x: Numpy array containing the x-values.  y: Numpy array containing the y-values. |
| **Returns:** | residuals - Numpy array containing the residuals between the fitted model’s y-values and the actual y-values. |
| **Description:** | A function which produces the residuals between a set of fitted data points and the original data points. This function will be called using the lmfit Minimizer object as an objective function to be minimized.  This objective function is appropriate when there are two convoluted peaks which can be fitted to a linear combination of 2 Lorentizan lineshapes. This objective function was specifically used to fit the vinyl region.  Parameter object must first be initialized and the parameters within the object must be pre-defined using a best guess. |

**Residuals.py**

residuals\_pxylene(parameters, x, y)

|  |  |
| --- | --- |
| **Parameters:** | parameters: An lmfit Parameters object defined by the user.  x: Numpy array containing the x-values.  y: Numpy array containing the y-values. |
| **Returns:** | residuals - Numpy array containing the residuals between the fitted model’s y-values and the actual y-values. |
| **Description:** | A function which produces the residuals between a set of fitted data points and the original data points. This function will be called using the lmfit Minimizer object as an objective function to be minimized.  This objective function is appropriate when there is a peak to the left and two convoluted peaks to the right, which can be fitted to a linear combination of a Lorentizan, a second Lorentzian and a Split-Lorentzian lineshape. This objective function was specifically used to fit the p-xylene region.  Parameter object must first be initialized and the parameters within the object must be pre-defined using a best guess. |

**CurveFitting.py**

lorentzian\_curve\_fit(residuals, parameters, x, y)

|  |  |
| --- | --- |
| **Parameters:** | residuals: Function imported from Residuals module which is the objective function to be minimized.  parameters: Parameter Object which contains all the relevant parameters for curve fitting.  x: Numpy array of x-values  y: Numpy array of y-values |
| **Returns:** | best\_fit - Numpy array containing the y-values of the best fit lineshape for the peak  fit\_params - Ordered dictionary of best fit parameters that can best fit the data, including the following:  1. Amplitude  2. Center  3. Half-Width at Half-Maximum  4. R-squared  5. FWHM  6. Height  7. AUC |
| **Description:** | Fit the peak via given a specified objective function (the residual which takes into account the lineshape), a set of parameters that is required to fit the lineshape, and the peak containing region's x-values and y-values. |

**CurveFitting.py**

gaussian\_curve\_fit(residuals, parameters, x, y)

|  |  |
| --- | --- |
| **Parameters:** | residuals: Function imported from Residuals module which is the objective function to be minimized.  parameters: Parameter Object which contains all the relevant parameters for curve fitting.  x: Numpy array of x-values  y: Numpy array of y-values |
| **Returns:** | best\_fit - Numpy array containing the y-values of the best fit lineshape for the peak  fit\_params - Ordered dictionary of best fit parameters that can best fit the data, including the following:  1. Amplitude  2. Center  3. Half-Width at Half-Maximum  4. R-squared  5. FWHM  6. Height  7. AUC |
| **Description:** | Fit the peak via given a specified objective function (the residual which takes into account the lineshape), a set of parameters that is required to fit the lineshape, and the peak containing region's x-values and y-values. |

**CurveFitting.py**

curve\_fit(residuals, parameters, x, y, region)

|  |  |
| --- | --- |
| **Parameters:** | residuals: Function imported from Residuals module which is the objective function to be minimized.  parameters: Parameter Object which contains all the relevant parameters for curve fitting.  x: Numpy array of x-values  y: Numpy array of y-values  region: String indicating either 'vinyl' or 'pxylene'. This parameter is crucial because it will trigger different fitting functions for calculating the AUC. |
| **Returns:** | fit\_params - Ordered dictionary of best fit parameters that can best fit the data  r2score - Float of the calculated r2 score between fitted curve and actual data  area - Float of the calculated AUC of the selected peak |
| **Description:** | Fit a curve to the region of interest. This curve fitting function was specifically written for the vinyl and p-xylene regions of a Raman spectra. Therefore, the region of interest must be clearly stated in the region parameter.    In general, the curve\_fit function can be modified to suit the requirements of a different region of interest. The programmer must ensure that the residual is well-defined in the Residuals.py submodule and that the initial guesses in the parameters file is appropriate.    The function can also be modified to return different fit parameters like the AUC, FWHM, height, etc., depending on the lineshape chosen. |

**BaselineSubtractionFunction.py**

baseline\_subtraction\_function(region)

|  |  |
| --- | --- |
| **Parameters:** | region: Pandas series of defined region of interest, extracted from dataset containing extracted spectra. |
| **Returns:** | linear\_fit - Numpy array containing the linear fit of the left-most 5 and right-most 5 x and y values, representing the intensities of the baseline.  y\_subtracted - Numpy array containing the y-values after baseline subtraction |
| **Description:** | Use linear least-squares to fit a linear baseline to the left-most 5 and right-most 5 x and y values of the region using the Polynomial module from the NumPy Library. The output of the fitting gives the coefficients a and b of the linear equation y = ax + b.  From the fitting coefficients a and b, construct linear\_fit, which is a NumPy array with the same length as array y. This linear\_fit array is the baseline, which contains all intensities of the baseline.  Subtract the baseline array linear\_fit from the original y-values to obtain a baseline-corrected array named y\_subtracted. |

**RegionDataFrame.py**

find\_nearest(array, value)

|  |  |
| --- | --- |
| **Parameters:** | array: Numpy array  value: Integer or float |
| **Returns:** | index - Index of element in the array  element - Element of the index. |
| **Description:** | Find the nearest element in an array to the user defined value, as well the index of the element in the array. Methodology: Use numpy broadcasting and subtract the array by the value, to give a new difference array. Then use the .argmin() method to find the index of the lowest difference in the difference array. Original element can be easily extracted using array[index]. |

**RegionDataFrame.py**

region\_df\_slice(col\_indices\_filename, raw\_data\_filename)

|  |  |
| --- | --- |
| **Parameters:** | col\_indices\_filename: String containing the filename with extension of .xlsx containing column indices set by the user.  raw\_data\_filename: String containing the filename with extension of .xlsx containing all extracted Raman spectra |
| **Returns:** | df\_vinyl: DataFrame truncated to be between the column indices for vinyl region set by the user.  df\_pxylene: DataFrame truncated to be between the column indices for pxylene region set by the user. |
| **Description:** | Truncates dataframe by pandas index slicing, based on the indices provided by the user in the excel file.  This function was specifically written for spitting the dataframe into vinyl and p-xylene regions but can be modified to suit the needs of the user. |

More details:

An Excel file with the name and extension: “column\_indices.xlsx” was created to be read by region\_df\_slice to partition the raw dataset appropriately into separate regions of interest:



The Excel file contains the leftmost and rightmost column indices of the vinyl and p-xylene regions.

Text, table

Description automatically generated

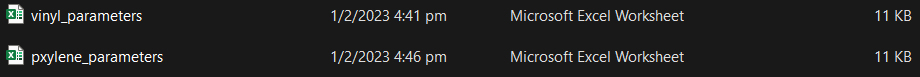
**Parameters.py**

define\_region\_parameters(filename)

|  |  |
| --- | --- |
| **Parameters:** | filename: String containing the excel filename with the file extension |
| **Returns:** | parameters - Parameters object which contains all relevant use defined parameters |
| **Description:** | Reads an Excel file and imports the user defined parameter values and bounds within the excel file into a Parameter object, amenable for usage during curve fitting using the lmfit library. |

More details:

Excel files with the names and extensions: “vinyl\_parameters.xlsx” and “pxylene\_parameters” were created to be read by the define\_region\_parameters function create lmfit Parameters objects for peak fitting:



The Excel file “vinyl\_parameters.xlsx” contains the initial guesses of the parameters to be fitted for the region:

A screenshot of a computer

Description automatically generated with low confidence

The Excel file “pxylene\_parameters.xlsx” contains the initial guesses of the parameters to be fitted for the region:

Table

Description automatically generated

The entries within the Excel files can be customized along with the define\_region\_parameters function, Please define parameter values according to the guidelines in the lmfit documentation. Leave all other optional entries blank.

Text

Description automatically generated with low confidence

<https://lmfit.github.io/lmfit-py/parameters.html#lmfit.parameter.Parameters.add>

**IterativeFitting.py**

iterative\_fitting(df\_region, parameter\_filename, region, residuals)

|  |  |
| --- | --- |
| **Parameters:** | df\_region: pandas DataFrame already truncated to contain the region of interest  parameter\_filename: String of filename with file extension  region: String indicating the region of interest  residuals: Function which acts as the objective function to be minimised. |
| **Returns:** | bestfit\_params\_list - List of Ordered Dictionary of Best fit parameters that can best fit the curve  r2\_score\_list - List of R2 scores of the fit  area\_list - List of AUC of Vinyl Peak |
| **Description:** | Iterate through every row of the region of interest and execute the curve fitting. |

**Consolidate.py**

aggregate\_ratio(df, vinyl\_area, vinyl\_r2\_score, pxylene\_area, pxylene\_r2\_score, filename

|  |  |
| --- | --- |
| **Parameters:** | df: DataFrame of the excel file containing the original  vinyl\_area: List of all vinyl peak AUC.  vinyl\_r2\_score: List of all vinyl region R2 scores.  pxylene\_area: List of all pxylene peak AUC.  pxylene\_r2\_score: List of all pxylene region R2 scores.  filename: String of the filename WITHOUT the extension. |
| **Returns:** | df\_ratio - DataFrame consisting only of the condition label, the mean ratio and the standard deviation of the ratio. |
| **Description:** | A function which calculates the aggregate mean AUC ratio and the standard deviation of the AUC ratio of multiple Raman spectra associated to their respective conditions. |

**ConversionCalculation.py**

create\_mean\_df(dataframes)

|  |  |
| --- | --- |
| **Parameters:** | dataframes: List of DataFrames containing the mean AUC ratio of conditions in the different residence times. |
| **Returns:** | df\_mean: DataFrame containing the mean ratio of the AUCs associated with each condition across  residence times. |
| **Description:** | Create a dataframe to store the mean AUC ratio values of each condition across residence times. |

**ConversionCalculation.py**

create\_std\_df(dataframes)

|  |  |
| --- | --- |
| **Parameters:** | dataframes: List of DataFrames containing the standard deviation of the AUC ratio of conditions in the different residence times. |
| **Returns:** | df\_std: DataFrame containing the standard deviation of the AUC ratios associated with each condition across residence times. |
| **Description:** | Create a dataframe to store the standard deviation of the AUC ratio values of each condition across residence times. |

**ConversionCalculation.py**

create\_ufloat\_df(df\_mean, df\_std)

|  |  |
| --- | --- |
| **Parameters:** | df\_mean: DataFrame containing the mean AUC ratios.  df\_std: DataFrame containing the standard deviation of the AUC ratios. |
| **Returns:** | df\_ufloat - DataFrame containing the mean with standard deviation as a single object in each DataFrame entry. |
| **Description:** | A function that takes in both df\_mean and df\_std and creates a single df\_ufloat containing the mean with standard deviation as a single object in each DataFrame entry. In this ufloat format, conversion calculation and error propagation would be immensely simpler.  df\_mean, df\_std and df\_ufloat are of the same shape. |

**ConversionCalculation.py**

calc\_conv\_and\_error(df\_ufloat)

|  |  |
| --- | --- |
| **Parameters:** | df\_ufloat: DataFrame containing the mean and std object in each entry. |
| **Returns:** | df\_conv\_and\_error - DataFrame containing the conversion and propagated error object in each entry. |
| **Description:** | Calculate the conversion and propagate the error leveraging upon the ufloat package. pandas broadcasting is used for conversion calculation and error propagation. |

**ConversionCalculation.py**

conversion\_and\_error(df\_conv\_and\_error)

|  |  |
| --- | --- |
| **Parameters:** | df\_conv\_and\_error: DataFrame containing the conversion and propagated error object in each entry. |
| **Returns:** | conversion\_df - DataFrame containing only conversion floats.  error\_df - DataFrame containing only error floats. |
| **Description:** | Create separate conversion and error DataFrames from df\_conv\_and\_error for plotting purposes. |