Least Squares Fitting using Python & Comparison with Excel

Jamie Lee Somers & Christian Saracut,
B.Sc in Applied Physics.

Thursday 22th February, 2021

1 Basic Tasks:

After graphing out the data provided in the xydata.csv file, we inferred that the slope of the graph was somewhere around 4 and the lines would intercept the y-axis somewhere around 0.1. We used these Initial guess values when running the code.

After running the code with our initial guess slope = 4 and initial guess intercept = 0.1, we got the following values:

```
Initial guess for slope, m0: 4

Initial guess for y-axis intercept, c0: 0.1

Optimization terminated successfully.
         Current function value: 0.009479
         Iterations: 32
         Function evaluations: 61
Best fit value of slope =  3.8525233676220463
Best fit value of intercept =  0.08087392102923352
```

Using the LINEST function in excel we were able to get similar values:

3.852517	0.080887
0.074931	0.048267
0.997735	0.039748
2643.398	6
4.176321	0.009479

We can see in both circumstance (the slope and the intercept), the two techniques give a value which is identical for the first 5 digits, this is quite close considering the LINEST function presents the uncertainty in the best fit for the slope and intercept as ±0.0749 and ±0.0483 respectively.

Since the two methods provided the same first 5 digits, we can be sure that the best fit value is:

$$\text{Slope} = 3.8525 \quad \text{y-intercept} = 0.0808$$

2 Intermediate Tasks:

The full version of the code used for this project, along with all of the comments added to explain each line of the code can be found at the end of this report in the appendix.

Meanwhile the flowchart is presented on the following page on its own, for readability purposes.

The next task required us to create our own custom data which had to be representative of a straight line with a little bit of "noise" added to the y-values. Our custom data was as follows:

Custom Data

	1	2	3	4	5	6	7	8	9	10
x-values	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y-values	0.0301	0.0853	0.1560	0.2506	0.3521	0.4409	0.5291	0.6211	0.6865	0.7601

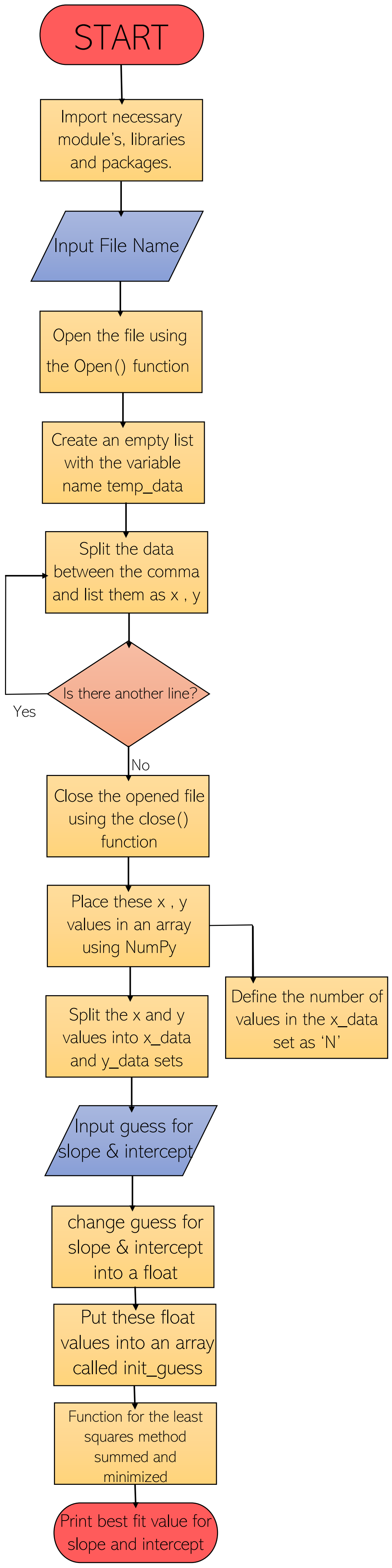
Now we have to carry out the best fit analysis like before on this set of data. When we graphed this function the slope appeared to be just under 1, we went with the value 0.85, and the intercept appeared to be slightly below 0, we guessed it was about -0.07. Inputting this data into python yields the following results:

```
Initial guess for slope, m0: 0.85

Initial guess for y-axis intercept, c0: -0.07

Optimization terminated successfully.
         Current function value: 0.001879
         Iterations: 21
         Function evaluations: 41
Best fit value of slope =  0.8501531744515523
Best fit value of intercept =  -0.07640553400246425
```

Using the LINEST method in Excel yields:

0.850189	− 0.07643
0.016874	0.01047
0.996859	0.015326
2538.643	8
0.596328	0.001879

Here our slope is equal to 5 digits and once again the y-intercept is equivalent to 5 digits. Line of best fit slope = 0.850, Line of best fit intercept = -0.0764.


START
Import necessary module's, libraries and packages.
Input File Name
Open the file using the Open() function
Create an empty list with the variable name temp_data
Split the data between the comma and list them as x , y
Is there another line?
Yes
No
Close the opened file using the close() function
Place these x , y values in an array using NumPy
Define the number of values in the x_data set as 'N'
Split the x and y values into x_data and y_data sets
Input guess for slope & intercept
change guess for slope & intercept into a float
Put these float values into an array called init_guess
Function for the least squares method summed and minimized
Print best fit value for slope and intercept

Week 4 - Using Least Squares Fitting for Non-Linear Data

In order to adapt our code for use with our non-linear equation:

$$y = A + B \cdot cos((0.26 \cdot x) + C)^2 \tag{1}$$

We had to change our guess values from being the slope m0 and intercept c0 and instead have variables for components A, B and C in the equation above as follows:

```
inpA0 = input('Initial␣guess␣for␣A:') #Defines a new variable inpm0 and asks the
    user to input an initial guess for A, which changes the shift of the cosine
    function up and down.
inpB0 = input('Initial␣guess␣for␣B:␣') #Defines a new variable inpc0 and asks the
    user to input an initial guess for B, which changes the amplitude of the cosine
    function.
inpC0 = input('Initial␣guess␣for␣C:') #Defines a new variable inpC0 and asks the
    user to input an initial guess for C, this affects the shift of the cosine
    function left and right.
A0=float(inpA0) #Changes the value inputed for A into a floating point number. This
     changes intergers into usable values with a decimal.
B0=float(inpB0) #Changes the value inputed for B into a floating point number. This
     changes intergers into usable values with a decimal.
C0=float(inpC0) #Changes the value inputed for C into a float point number. This
    changes intergers into usable values with a decimal.
```

Like the linear code these values can then be placed into an array and used to in the squared difference equation, we must also adapt the equation from the linear equation provided to the equation our group was assigned, which is typed out as follows:

```
 squared_diff = ((A + (B*np.cos((0.26*x_data)+C))**2)-y_data)**2 #This equation is
     given the name 'squared_diff' which involves adding A to B times cos(0.26x_data
     )+C) where x_data is the list of x values found using slice notation squared -
     y_data and the whole answer is then squared.
```

Our initial guess for the values of A, B and C were 6.9, 2.5 and 20.344 respectively. Using these values we extracted a best fit value as follows:

```
Initial guess for A:6.9

Initial guess for B: 2.5

Initial guess for C:20.344

Optimization terminated successfully.
         Current function value: 13.807796
         Iterations: 52
         Function evaluations: 99
Best fit value of A =  6.909815062934323 Best fit value of B =  2.53957032035156
    Best fit value of C =  20.343670789884357
```

In order to plot the graph of our data using Python, we changed the graphed equation to be a scatterplot that used the x-values found in the data file imported as well as the y-values in the data file. Meanwhile the best fit line was produced using a set of 300 x-values we defined as being in the range from the first x data value to the last x data value for a smoother line of best fit and the best fit slope and best fit intercept value produced by the python programme earlier.

The graph we produced using our above estimates is as follows:

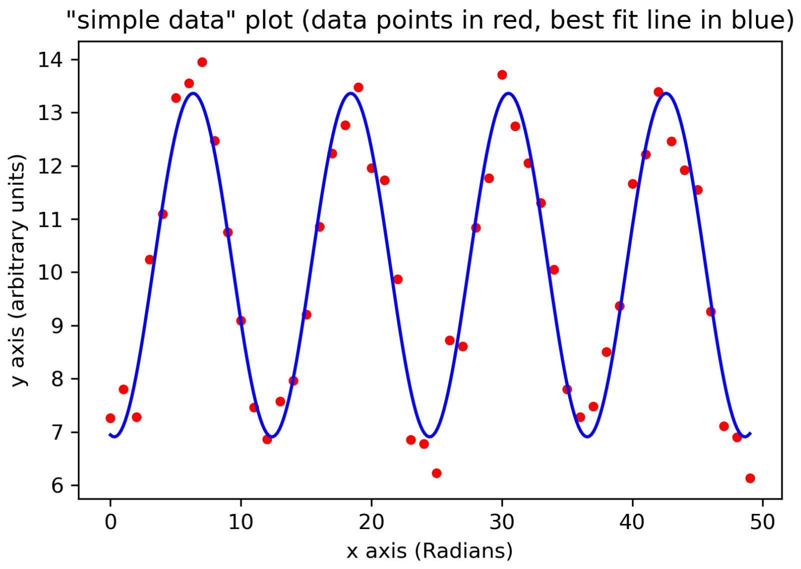


Figure 1: Graph of linear function with 50 data points and 300 line of best fit values

At this point it is important to note that although our value for C is around 20.344, it is not the only possible value for this variable.

Since the C value shifts the function along the horizontal plane and the cosine function is repeated in an identical pattern, it is possible to shift the C value exactly 1π radian and still end up with the exact same pattern output.

Other possible values for C were found to be: 1.494, 4.635, 7.777, 10.919 and 14.060

These values closely correlate to $\frac{\pi}{2}$, $\frac{3\pi}{2}$, $\frac{5\pi}{2}$, $\frac{7\pi}{2}$ and $\frac{9\pi}{2}$ respectively

We can see that our graph fits the data perfectly except for the peaks where the line of best fit fall slightly short of the data points, this implies that our amplitude value B, might have been slightly off.

The best fit function verifies this analysis by claiming that the value of B is actually closer to 2.54 than 2.5.

3 Advanced Tasks:

The produced graphs appear as follows:

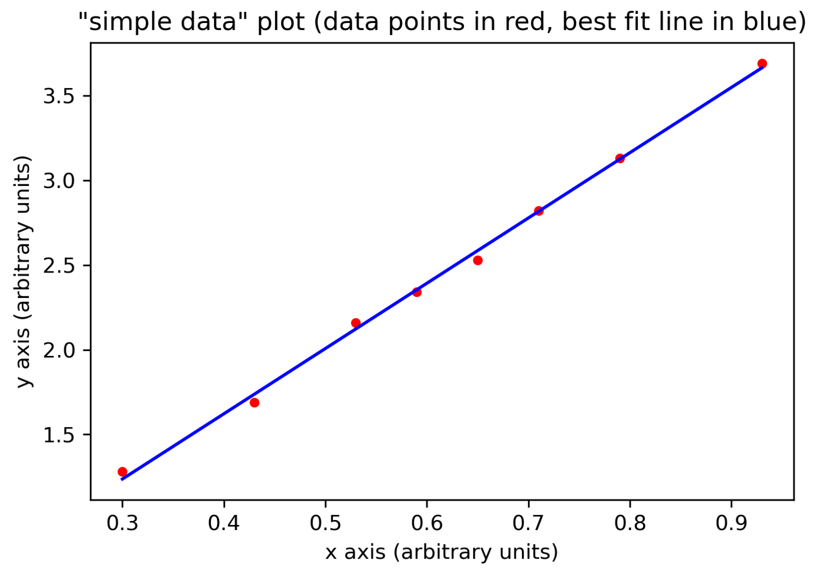


Figure 2: Graph of supplied sample data with 8 data points

We can see that the graph axis scales appropriately based on what the outputted x and y values are.

As previously mentioned our guess for the slope of our custom data was 0.85 and our y-intercept was -0.07, when we use these values to plot the graph we get:

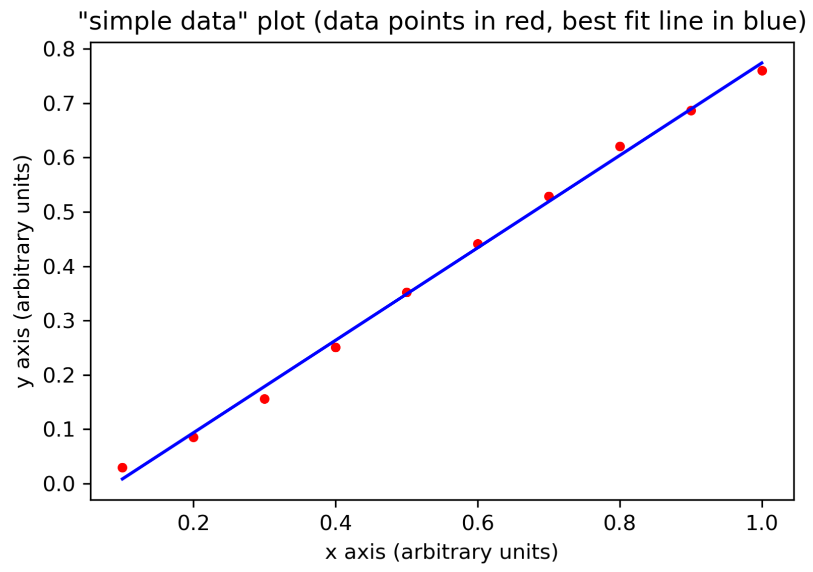


Figure 3: Graph of custom created data with 10 data points

We can see from the graph that our slope appears to be correct. To the eye this line of best fit looks like it passes evenly through each of the points, which is what we look for in a line of best fit.

Fit parameters with uncertainties

The final section of code we were to use, allows us to output not only the best fit value for our variables in the equation but also their uncertainties. These quantities can be compared with those outputted by excel using the LINEST function.

The outputted values for the Linear xydata.csv was as follows:

```
Initial guess for slope, m0: 4

Initial guess for y-axis intercept, c0: 0.1

Best fit value of slope using curve_fit =  3.852516547559557 +/-
   0.07493131005162702
Best fit value of intercept using curve_fit =  0.0808866764901232 +/-
   0.04826747813963919
```

If we compare the slope value 3.852516547559557 and uncertainty 0.07493131005162702, to that outputted earlier by the LINEST function, slope value 3.852517 and uncertainty 0.074931.

We can clearly see that the values perfectly match however the Excel values only have 7 significant figures and round the final value meanwhile the python values are correct to 16 significant figures.

Meanwhile the intercept value 0.0808866764901232 and uncertainty 0.04826747813963919, compared to the LINEST function intercept value 0.080887 and uncertainty 0.048267 is once again a perfect match with the python values just being accurate to way more significant figures.

We repeated this process for the non-linear once we changed the code to allow 3 variables for our function. Our output was as follows:

```
Initial guess for A:6.9

Initial guess for B: 2.5

Initial guess for C:20.344

Best fit value of vertical shift using curve_fit =  6.909845022018075 +/-
   0.12871069578506864
Best fit value of amplitude using curve_fit =  2.5395540153590725 +/-
   0.042089821333540126
Best fit value of horizontal shift using curve_fit =  20.34367687534512 +/-
   0.017072326524096078
```

We can now compare the vertical shift value 6.909845022018075 to the one produced earlier 6.909815062934323, the values are identical for the first 5 significant figures, after which they differ. Its important to note that the uncertainty value given above is ± 0.1 which the discrepancy between these two values is certainly within.

The amplitude value 2.5395540153590725 compared with the 2.53957032035156 shows a similar story, where the values are identical for the first 5 significant figures, after which they differ. Once again the uncertainty value given is ± 0.04 which the discrepancy falls within.

The horizontal shift value 20.34367687534512 compared with the previous 20.343670789884357 is accurate to 7 significant figures, which is certainly in the range of the uncertainty value ± 0.02.

Final Values for Linear Data

Our final values with their uncertainty are as follows:

Slope Value:

$$m = 3.85 \pm 0.07$$

Y-intercept Value:

$$c = 0.08 \pm 0.05$$

Final Values for Non-Linear Data

Our final values with their uncertainty are as follows:

Vertical Shift Value:

$$A = 6.9 \pm 0.1$$

Amplitude Value:

$$B = 2.54 \pm 0.04$$

Horizontal Shift Value:

$$C = 20.34 \pm 0.02$$

Appendix:

Linear Code (with comments)

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 21 14:38:51 2021

@authors: JamieSomers & ChristianSaracut
"""

import math as math #Imports the module known as math (this supplies mathematical
    functions on floating-point numbers) and changes its name to math
import numpy as np #Imports the library known as numpy and changes its name to np
import cmath as cm #Imports the module known as cmath (this supplies equivalent
    functions on complex numbers) and changes its name to cm
import scipy as sp  #Imports the package known as SciPy (this provides more utility
     functions for optimization, stats and signal processing) and changes its name
    to sp
from scipy import optimize #Uses SciPy to import omptimize (this provides functions
     for minimizing (or maximizing) objective functions)
import matplotlib.pyplot as plt #Imports MatPlotLib which is a comprehensive
    library for creating static, animated and interactive visualizations in Python
    like creating plots, and changes its name to plt

'''---------------------------------------------------------------------'''
filename = input('Data␣file␣name␣(including␣extension):␣') #The first line of code
    asks the user to input the filename which has the data they want analzed (
    typically a .csv file) and changes this file to the variable filename

my_file = open(filename) #The Second line of code opens the data file using the
    open() function and makes the open file a variable called my_file

temp_data=[] #This makes an empty list and gives it the variable name temp_data

for line in my_file: #This is the beginning of the function 'for' which tells the
    program what to do FOR every line in the opened file my_file created above.
    x, y = line.split(',') #This tells the program to seperate the file on each
        line into a stringe with two components an x components, they will be split
        wherever there is a comma in the string.
    temp_data += [float(x), float(y)] #This line tells the program to add each of
        the x,y strings created above to the empty temp_data we created earlier,
        this will fill the temp_data list with all x and y values in the opened file
        .

my_file.close() #This line tells the program that we can now close the opened file
    using the close() function once all the data has been placed in the list.

data_set = np.array(temp_data) #This changes the filled temp_data list into an
    array using the NumPy library and makes this a new variable called data_set.
```

```python
x_data = data_set[0::2] #This is using whats known as 'Slice Notation',
    specifically this line of code tells the program to take the first element in
    the list before stepping 1 and taking what would be the third element, the
    reason its skipping one is to ignore the y-values as this is the variable for
    x_data.
y_data = data_set[1::2] #This is more Slice notation telling the program to start
    at the second element in the list (1, since python starts counting at 0) and
    step 1 element every time as this is the variable for y_data.
N=len(x_data) #This is defining a varialbe 'N' to be however many values are within
     the x_data string. Realistically we could also use y_data here as the number of
     values in the string should be identical. (We shouldn't have more x-values than
     y-values.)

'''-----------------------------------------------------------------------'''

'''-----------------------------------------------------------------------'''
inpm0 = input('Initial␣guess␣for␣slope,␣m0:␣') #Defines a new variable inpm0 and
    asks the user to input an initial guess for the slope m0.
inpc0 = input('Initial␣guess␣for␣y-axis␣intercept,␣c0:␣') #Defines a new variable
    inpc0 and asks the user to input an initial guess for the y-intercept c0.
m0=float(inpm0) #Changes the value inputed for slope m0 into a floating point
    number. This changes intergers into usable values with a decimal.
c0=float(inpc0) #Changes the value inputed for y-intercept c0 into a floating point
     number. This changes intergers into usable values with a decimal.

init_guess = np.array([m0,c0]) #Takes the two new floating point numbers m0 and c0
    and uses NumPy to produce an array which then gets the variable name init_guess.

def lin_fit(par): #This line is the beginning of a new function denoted by the def
    keyword, and the function name which is defined as lin_fit with the argument (
    par).
    m=par[0] #This line sets up the variable 'm' which defines the slope to be hard
         coded as 0 in the function parameters.
    c=par[1] #This line sets up the variable 'c' which defines the y-intercept to
        be hard coded as 1 in the function parameters.
    squared_diff = (((m*x_data)+c)-y_data)**2 #This equation is given the name '
        squared_diff' which involves multiplying the each value in the x_data list
        found using slice notation by the hard coded slope 0, and adding the hard
        coded y-intercept 1 after, then that whole answer has each y_data value in
        the list subtracted from it before being squared.
    sum_squares = np.sum(squared_diff) #Since the calculation above uses many
        different values for the x_data and y_data, this line uses the NumPy SUM
        function to add up all of the answers and gives this answer the variable
        name sum_squares
    return sum_squares #This is the function return statement, it returns the value
         for the sum of all the values gotten by the function.

best_fit = sp.optimize.fmin(lin_fit, init_guess) #This line produces a variable
    called best_fit which is created using the SciPy package 'optimize' which we
    imported earlier. This finds the best fit by minimizing the function using a
    downhill simplex algorithm using both the inputed guesses for the slope and y-
    intercept.
```

```
print("Best␣fit␣value␣of␣slope␣=␣",best_fit[0], "Best␣fit␣value␣of␣intercept␣=␣",
    best_fit[1]) #Finally the program prints the string of text "Best fit value of
    slope = " followed by the best_fit value for 0 which denoted the slope and the
    string of text "Best fit value of intercept = " followed by the best_fit value
    for 1 which denoted the intercept
'''-----------------------------------------------------------------------'''

'''-----------------------------------------------------------------------'''
x = x_data #Set the x values to be the number of x values in the inputted file
y1=m0*x+c0 #y-values using guess for slope and y-axis intercept values
y2=best_fit[0]*x+best_fit[1]#y-values using best fit value of slope and best fit
    value of intercept


plt.scatter(x, y_data, s=13, color="red") #Scatterplot using y-values from guess
    for slope and guess for y-axis intercept, the s=13 changes the diameter of the
    dots, and changes the color to red.
plt.plot(x, y2, color="blue", linewidth=1.5)#line of best fit using y-values from
    best fit value of slope and best fit value of intercept,changes the color to
    blue, the linewidth changes the thickness of the line.

plt.xlabel('x␣axis␣(arbitrary␣units)') #label the x-axis with the word "x axis (
    arbitrary units)"
plt.ylabel('y␣axis␣(arbitrary␣units)') #label the y-axis with the word "y axis (
    arbitrary units)"
plt.title('"simple␣data"␣plot␣(data␣points␣in␣red,␣best␣fit␣line␣in␣blue)') #label
    the plot title "simple data" plot (data points in red, best fit line in blue)
plt.savefig('filename2.png', dpi=300)#This line of code saves the plot as a .png in
     higher resolution than Spyder will allow

'''-----------------------------------------------------------------------'''

'''-----------------------------------------------------------------------'''

def line(xdata,m,c): #This line is the beginning of a new function denoted by the
    def keyword, and the function name which is defined as line with the arguments (
    xdata,m and c).
    return ((m*xdata)+c) #This is the return statement, it returns the value
        produced by the equation of a line when the values for xdata,m and c are all
         inputted.

best_fit, cov = sp.optimize.curve_fit(line, x_data, y_data,[m0,c0]) #Creates two
    variables best_fit and cov, which both equate to an optimized curve fit function
     where our equation of a line is f, our x values are x_data, our y values are
    y_data and uses our initial guesses m0 and c0. these variables both produce
    arrays
fit_err = np.sqrt(np.diag(cov)) #Creates a variable named fit_err which equates to
    the square root of the diagonal array produced by the covarience variable
    defined above.
```

```
print("Best␣fit␣value␣of␣slope␣using␣curve_fit␣=␣",best_fit[0], "+/-", fit_err[0])
    #Prints the best fit value of the slope using the curve fit function which is
    the first value of our best_fit array and its error is the square root of the
    diagonal first value in our cov array
print("Best␣fit␣value␣of␣intercept␣using␣curve_fit␣=␣",best_fit[1], "+/-", fit_err
    [1]) #Prints the best fit value of the intercept using the curve fit function
    which is the second value of our best_fit array and its error is the square root
     of the diagonal second value in our cov array

'''_______________________________________________________________________'''
```

Non-Linear Code (with comments)

```
# -*- coding: utf-8 -*-
"""
Created on Fri Feb 12 20:12:11 2021

@authors: JamieSomers & ChristianSaracut
"""

import math as math #Imports the module known as math (this supplies mathematical
    functions on floating-point numbers) and changes its name to math
import numpy as np #Imports the library known as numpy and changes its name to np
import cmath as cm #Imports the module known as cmath (this supplies equivalent
    functions on complex numbers) and changes its name to cm
import scipy as sp  #Imports the package known as SciPy (this provides more utility
     functions for optimization, stats and signal processing) and changes its name
    to sp
from scipy import optimize #Uses SciPy to import omptimize (this provides functions
     for minimizing (or maximizing) objective functions)
import matplotlib.pyplot as plt #Imports MatPlotLib which is a comprehensive
    library for creating static, animated and interactive visualizations in Python
    like creating plots, and changes its name to plt

'''_______________________________________________________________________'''
filename = input('Data␣file␣name␣(including␣extension):␣') #The first line of code
    asks the user to input the filename which has the data they want analzed (
    typically a .csv file) and changes this file to the variable filename

my_file = open(filename) #The Second line of code opens the data file using the
    open() function and makes the open file a variable called my_file

temp_data=[] #This makes an empty list and gives it the variable name temp_data

for line in my_file: #This is the beginning of the function 'for' which tells the
    program what to do FOR every line in the opened file my_file created above.
    x, y = line.split(',') #This tells the program to seperate the file on each
        line into a stringe with two components an x components, they will be split
        wherever there is a comma in the string.
    temp_data += [float(x), float(y)] #This line tells the program to add each of
        the x,y strings created above to the empty temp_data we created earlier,
        this will fill the temp_data list with all x and y values in the opened file
        .

my_file.close() #This line tells the program that we can now close the opened file
    using the close() function once all the data has been placed in the list.

data_set = np.array(temp_data) #This changes the filled temp_data list into an
    array using the NumPy library and makes this a new variable called data_set.

x_data = data_set[0::2] #This is using whats known as 'Slice Notation',
    specifically this line of code tells the program to take the first element in
    the list before stepping 1 and taking what would be the third element, the
    reason its skipping one is to ignore the y-values as this is the variable for
    x_data.
y_data = data_set[1::2] #This is more Slice notation telling the program to start
    at the second element in the list (1, since python starts counting at 0) and
    step 1 element every time as this is the variable for y_data.
N=len(x_data) #This is defining a varialbe 'N' to be however many values are within
     the x_data string. Realistically we could also use y_data here as the number of
     values in the string should be identical. (We shouldn't have more x-values than
     y-values.)
x_best = np.linspace(x_data[0],x_data[N - 1],num=300) #By using np.linspace we can
    increase the number of values our line of best fit has which will give it
    overall a smoother appearance than using just the 50 datapoints we were given.
'''-----------------------------------------------------------------------'''

'''-----------------------------------------------------------------------'''
inpA0 = input('Initial␣guess␣for␣A:') #Defines a new variable inpm0 and asks the
    user to input an initial guess for A, which changes the shift of the cosine
    function up and down.
inpB0 = input('Initial␣guess␣for␣B:␣') #Defines a new variable inpc0 and asks the
    user to input an initial guess for B, which changes the amplitude of the cosine
    function.
inpC0 = input('Initial␣guess␣for␣C:') #Defines a new variable inpC0 and asks the
    user to input an initial guess for C, this affects the shift of the cosine
    function left and right.
A0=float(inpA0) #Changes the value inputed for A into a floating point number. This
     changes intergers into usable values with a decimal.
B0=float(inpB0) #Changes the value inputed for B into a floating point number. This
     changes intergers into usable values with a decimal.
C0=float(inpC0) #Changes the value inputed for C into a float point number. This
    changes intergers into usable values with a decimal.

init_guess = np.array([A0,B0,C0]) #Takes the three new floating point numbers A0,
    B0 and C0 and uses NumPy to produce an array which then gets the variable name
    init_guess.

def lin_fit(par): #This line is the beginning of a new function denoted by the def
    keyword, and the function name which is defined as lin_fit with the argument (
    par).
    A=par[0] #This line sets up the variable 'A' to be hard coded as 0 in the
        function parameters.
    B=par[1] #This line sets up the variable 'B' to be hard coded as 1 in the
        function parameters.
    C=par[2] #This line sets up the variable 'C' to be hard coded as 2 in the
        function parameters.
    squared_diff = ((A + (B*np.cos((0.26*x_data)+C))**2)-y_data)**2 #This equation
        is given the name 'squared_diff' which involves adding A to B times cos(0.26
        x_data)+C) where x_data is the list of x values found using slice notation
        squared -y_data and the whole answer is then squared.
    sum_squares = np.sum(squared_diff) #Since the calculation above uses many
        different values for the x_data and y_data, this line uses the NumPy SUM
        function to add up all of the answers and gives this answer the variable
        name sum_squares
    return sum_squares #This is the function return statement, it returns the value
         for the sum of all the values gotten by the function.

best_fit = sp.optimize.fmin(lin_fit, init_guess) #This line produces a variable
    called best_fit which is created using the SciPy package 'optimize' which we
    imported earlier. This finds the best fit by minimizing the function using a
    downhill simplex algorithm using both the inputed guesses for the slope and y-
    intercept.

print("Best␣fit␣value␣of␣A␣=␣",best_fit[0], "Best␣fit␣value␣of␣B␣=␣",best_fit[1], "
    Best␣fit␣value␣of␣C␣=␣",best_fit[2]) #Finally the program prints the string of
    text "Best fit value of A = " followed by the best_fit value for 0 which denotes
     A, the string of text "Best fit value of B = " followed by the best_fit value
    for 1 which denotes B and the string of text "Best fit value of C = " followed
    by the best_fit value for 2 which denotes C
'''-----------------------------------------------------------------------'''

'''-----------------------------------------------------------------------'''
x = x_data #Set the x values to be the number of x values in the inputted file

y1=A0 + (B0*np.cos((0.26*x_data)+C0))**2 #y-values using guess for A, B and C
    values inputed.
y2=best_fit[0] + (best_fit[1]*np.cos((0.26*x_best)+best_fit[2]))**2#y-values using
    best fit value of A, B and C values and using x_best to give us 300 data points
    instead of just 50.


plt.scatter(x, y_data, s=13, color="red") #Scatterplot using y-values from guess
    for A, B and C values, the np.radians() function was used to denote the x-values
     are in radians, the s=13 changes the diameter of the dots, and changes the
    color to red.
plt.plot(x_best, y2, color="blue", linewidth=1.5)#line of best fit using y-values
    from best fit value of A, B and C,changes the color to blue, the linewidth
    changes the thickness of the line.

plt.xlabel('x␣axis␣(Radians)') #label the x-axis with the word "x axis (Radians)"
plt.ylabel('y␣axis␣(arbitrary␣units)') #label the y-axis with the word "y axis (
    arbitrary units)"
plt.title('"simple␣data"␣plot␣(data␣points␣in␣red,␣best␣fit␣line␣in␣blue)') #label
    the plot title "simple data" plot (data points in red, best fit line in blue)
plt.savefig('filename2.png', dpi=300)#This line of code saves the plot as a .png in
     higher resolution than Spyder will allow

'''-----------------------------------------------------------------------'''

'''-----------------------------------------------------------------------'''

def line(xdata,A,B,C): #This line is the beginning of a new function denoted by the
     def keyword, and the function name which is defined as line with the arguments
    (xdata,A,B and C).
    return A + (B*np.cos((0.26*xdata)+C))**2 #This is the return statement, it
        returns the value produced by the equation when the values for xdata,A,B and
        C are all inputted.

best_fit, cov = sp.optimize.curve_fit(line, x_data, y_data,[A0,B0,C0]) #Creates two
     variables best_fit and cov, which both equate to an optimized curve fit
    function where our equation is f, our x values are x_data, our y values are
    y_data and uses our initial guesses A0, B0 and C0. these variables both produce
    arrays
fit_err = np.sqrt(np.diag(cov)) #Creates a variable named fit_err which equates to
    the square root of the diagonal array produced by the covariance variable
    defined above.

print("Best␣fit␣value␣of␣vertical␣shift␣using␣curve_fit␣=␣",best_fit[0], "+/-",
    fit_err[0]) #Prints the best fit value of the vertical shift using the curve fit
     function which is the first value of our best_fit array and its error is the
    square root of the diagonal first value in our cov array
print("Best␣fit␣value␣of␣amplitude␣using␣curve_fit␣=␣",best_fit[1], "+/-", fit_err
    [1]) #Prints the best fit value of the amplitude using the curve fit function
    which is the second value of our best_fit array and its error is the square root
     of the diagonal second value in our cov array
print("Best␣fit␣value␣of␣horizontal␣shift␣using␣curve_fit␣=␣",best_fit[2], "+/-",
    fit_err[2]) #Prints the best fit value of the horizontal shift using the curve
    fit function which is the third value of our best_fit array and its error is the
     square root of the diagonal third value in our cov array

'''-----------------------------------------------------------------------'''
```