1

**Photoelectric Effect Python Experiment**

Go to https://applets.kcvs.ca/photoelectricEffect/PhotoElectric.html# to start your simulation experiment to col lect data to use in your Python code. In the options menu on the upper left-hand corner, select a metal to analyze. You can choose from Cs, Ca, K, or Na. Adjust the wavelength to 200 nm. Record the frequency. Adjust the voltage by 0.25 and click record the data point. You can stop recording data once you have reached 5 V, click show data and download the CSV. Save the CSV file according to the name of the metal you are analyzing and the wavelength. Use an underscore instead of a space to save the file (Example Cs 200nm.csv). Go to the upper left-hand corner under options and clear the data. Start the process again at 250, 300, 350, 400, 450, and 500 nm. Put these files in your Python folder in the Photoelectric effect experiment on your desktop.

First, import the necessary libraries to work with your data.

1 import pandas as pd

2 import numpy as np

3 import matplotlib.pyplot as plt

4 import os #for manipulating files

5 import re #regular expressions for manipulating text

**Part 1**

First, we will write our plotting function for this data set. It will accept a dataframe object with our data points, the labels of the x and y columns, a title to display, the coefficients of the trendline to plot, and the filename to save a png image to.

1 def plot\_data(dataframe,x\_column,y\_column,title,trendline\_coeffs=None,output\_file=None): 2 x = dataframe[x\_column]

3 y = dataframe[y\_column]

4

5 plt.scatter(x,y, label = ’Data Points’, color=’blue’)

6

7 plt.xlabel(x\_column)

8 plt.ylabel(y\_column)

9 plt.title(title)

10

11 if trendline\_coeffs is not None:

12 poly\_coeff = trendline\_coeffs

13 poly\_degree = len(trendline\_coeffs) -1

14 poly = np.poly1d(poly\_coeff)

15 x\_fit = np.linspace(x.min(),x.max(), 500)

16 y\_fit = poly(x\_fit)

17

18 plt.plot(x\_fit, y\_fit, label=f’Polynomial Fit (Degree {poly\_degree})’,color=’red’) 19

20 equation\_text = "$y = " + " + ".join([f"{coef:.3e}xˆ{poly\_degree - i}" if i != poly\_degree else f"{round(coef, 3)}"

21 for i, coef in enumerate(poly\_coeff)]) + "$" 22 plt.text(0.2,0.8, equation\_text, transform=plt.gca().transAxes, fontsize=8, 23 verticalalignment=’bottom’, horizontalalignment=’left’) 24 plt.legend()

25

26 if output\_file:

27 plt.savefig(output\_file,format=’png’,dpi=300, bbox\_inches=’tight’) 28

29 plt.show()

2

Now, we will use this newly created plot function to display the data we collected. Load your data from your csv files into a pd.DataFrame object.

1 file\_path = ’Ca\_200nm.csv’

2 data = pd.read\_csv(file\_path)

Define the x column and y column and format your title.

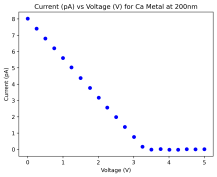
1 xcol = ’Voltage (V)’

2 ycol = ’Current (pA)’

3 title = f’{ycol} vs {xcol} for Ca Metal at 200nm’

Finally, we are ready to call our plot function.

1 plot\_data(data,xcol,ycol,title,output\_file=’Ca\_200nm.png’)

Your output should look something like this. If it doesn’t, you should get a TA to help before proceeding. 

**Part 2**

Now we will perform a polynomial fit on our data, and plot the best-fit curve. First, we will use NumPy’s polyfit function to obtain a tuple of polynomial coefficients.

1 poly\_degree = 4

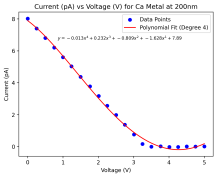
2 poly\_coeff = np.polyfit(data[xcol],data[ycol],poly\_degree)

Plotting our data with the fit is as simple as re-using the line of code we used to plot it without, except that we add the polynomial coefficients as a parameter.

1 plot\_data(data,xcol,ycol,title,poly\_coeff,output\_file=’Ca\_200nm.png’)

3

Your output should look something like this. Again, if it doesn’t, get a TA to help.



Calculate the y intercept and smallest positive root of the polynomial fit. We will use the smallest positive root as the cutoff voltage, *V*0.

1 def find\_root\_and\_yint(data,x\_column,y\_column,poly\_degree=2,interval=[0,np.inf]): 2 x = data[x\_column]

3 y = data[y\_column]

4 poly\_coeff = np.polyfit(x,y,poly\_degree)

5 root = min([float(root) for root in np.roots(poly\_coeff) if interval[0] < root and root < interval[1]])

6 yint = float(poly\_coeff[-1])

7 return root,yint

Here is an example of using this function to obtain the cutoff voltage for our data set:

1 root\_yint = find\_root\_and\_yint(data,’Voltage (V)’, ’Current (pA)’, poly\_degree) 2 poly\_V0 = root\_yint[0]

3 print(f"for degree {poly\_degree} fit : root: {root\_yint[0]:.3f} ; y-intercept: {root\_yint [1]:.3f}")

In your answer to Part 2, show your code, the plot with polynomial fit, and the calculated y intercept and *V*0.

4

**Part 3**

Now, we will repeat all of this using a linear regression. Because there are many zero data points which we do not want, we will need to write a function that will truncate our data so that it does not include these.

1 def truncate\_data(dataframe,xcol,ycol,y\_threshold):

2 ’’’

3 returns a dataframe including only the first value

4 lower than or equal to y\_threshold

5 ’’’

6 for index, y\_value in enumerate(dataframe[ycol]):

7 if y\_value <= y\_threshold:

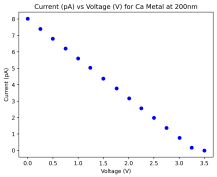
8 #return dataframe only up to and including this value

9 return dataframe.iloc[0:index + 1]

10 #nothing was below threshold

11 return dataframe

Now, plot your truncated data. It should look something like this; if it doesn’t, you will once more need to bother your TA.



Once we have prepared our truncated graph, repeating this procedure with a linear regression is as easy as re-using all our old functions in the same order, with the only changes being the filename and setting poly degree = 1.

1 #This is all that changes!

2 poly\_degree = 1

3 filename = ’Ca\_200nm\_trunc\_linear’

4 #all this is the same!

5 poly\_coeff = np.polyfit(data[xcol],data[ycol],poly\_degree)

6 plot\_data(data,xcol,ycol,title,poly\_coeff,output\_file=filename)

7 root\_yint = find\_root\_and\_yint(data,’Voltage (V)’, ’Current (pA)’, poly\_degree) 8 lin\_V0 = root\_yint[0] #I lied, this line is different

9 print(f"for degree {poly\_degree} fit : root: {root\_yint[0]:.3f} ; y-intercept: {root\_yint [1]:.3f}")

For your answer to part 3, include the code you used for this section, the graphical output, and the calculated root and y-intercept.

5

**Part 4**

Now, we are at the good stuff. We will calculate the work function of the metal, using the stopping potential calculated by finding the root of our regression. Write a function to calculate *ϕ* for your metal, using the formula *ϕ* = *hν − eV*0.

We will define the physical constants we will continually use:

1 c = 3e8

2 h = 6.626e-34

3 e = 1.602e-19

1 #this one’s on you... it’s just arithmetic!

2 def calculate\_phi(V\_0,wavelength\_nm):

3 #your function body goes here

4 return phi

Next, calculate phi for the polynomial fit, and for the linear fit. Are they significantly different?

1 phi\_poly = calculate\_phi(poly\_V0,200)

2 phi\_lin = calculate\_phi(lin\_V0,200)

3 print(f’Phi\_poly : {phi\_poly}’)

4 print(f’Phi\_linear : ’ + str(phi\_lin))

In your answer to Part 4, show your code and the output you obtained, as well as any commentary on the differences between these values.

**Part 5**

Here, we will calculate the cutoff frequency (and wavelength) for our metal using the work function obtained in Part 4. Remember, *V*0 = ( *heν*) *−* (*ϕe*) = 0.

1 def calculate\_cutoff\_nu\_lambda(phi):

2 #your logic goes here

3 return nu\_cutoff\_Hz,lambda\_cutoff\_nm

Now, use this to obtain values of cutoff frequency for each regression technique.

1 lin\_nu\_lambda = calculate\_cutoff\_nu\_lambda(phi\_lin)

2 nu\_cutoff\_lin = lin\_nu\_lambda[0]

3 print(f’Cutoff Frequency (Hz) (degree 1): {nu\_cutoff\_lin:.0f}’)

4 lambda\_cutoff\_lin = lin\_nu\_lambda[1]

5 print(f’Cutoff Wavelength (nm) (degree 1): {lambda\_cutoff\_lin:.0f}’) 6

7 poly\_nu\_lambda = calculate\_cutoff\_nu\_lambda(phi\_poly)

8 nu\_cutoff\_poly = poly\_nu\_lambda[0]

9 print(f’Cutoff Frequency (Hz) (degree 4): {nu\_cutoff\_poly:.0f}’)

10 lambda\_cutoff\_poly = poly\_nu\_lambda[1]

11 print(f’Cutoff Wavelength (nm) (degree 4): {lambda\_cutoff\_poly:.0f}’) For this section, show your calculate cutoff nu lambda function and the output you obtain.

6

**Part 6**

Now, we will bring it all together. We will create a function using every function we have created so far, which accepts a dataframe, an x column label and y column label, the wavelength in nm, and the polynomial degree, and returns all the derivable values in a Python dict object.

1 def process\_data(dataframe,xcol,ycol,lambda\_nm,poly\_degree):

2 data = truncate\_data(dataframe,xcol,ycol,0.01)

3 data\_dict = {}

4

5 poly\_coeff = np.polyfit(data[xcol],data[ycol],poly\_degree)

6 root\_yint = find\_root\_and\_yint(data,xcol,ycol, poly\_degree)

7 V0 = root\_yint[0]

8 phi = calculate\_phi(V0,200)

9 nu\_lambda = calculate\_cutoff\_nu\_lambda(phi)

10 cutoff\_nu\_Hz = nu\_lambda[0]

11 cutoff\_lambda\_nm = nu\_lambda[1]

12

13 data\_dict[’Wavelength(nm)’] = lambda\_nm

14 data\_dict[’Frequency(Hz)’] = c / (lambda\_nm \* 1e-9)

15 data\_dict[’Stopping Potential(V)’] = V0

16 data\_dict[’Work function(J)’] = phi

17 data\_dict[’Cut-off Frequency(Hz)’] = cutoff\_nu\_Hz

18 data\_dict[’Cut-off Wavelength(nm)’] = cutoff\_lambda\_nm

19

20 return data\_dict

Run this function on your data. Show the output you obtain.

1 data\_dict= process\_data(data,xcol,ycol,200,1)

2 print(data\_dict)

**Part 7**

In this section, we will use this function to read all of the data we collected into a table. You may see a pattern here- once you define a procedure in python, you can scale up by doing it many, many times. Compare this to your experience with excel, which involves all too much manual labor!

First, define a method to read all of the files in the directory, and return a dict mapping wavelengths to dataframes.

1 def read\_files():

2 filenames = os.listdir(’./’) #’./’ is the current directory

3 csv\_files = [file for file in filenames if file.endswith(’.csv’)] 4 data = {}

5 for filename in csv\_files:

6 wavelength\_nm = re.search(’\d+’,filename).group(0)

7 data[wavelength\_nm] = pd.read\_csv(filename)

8 return data

Plot all of these data sets to ensure that everything works:

1 file\_data = read\_files()

2 for \_data in file\_data:

3 trunc\_data = truncate\_data(file\_data[\_data],xcol,ycol,0.0)

4 plot\_data(trunc\_data,xcol,ycol,\_data)

7

Now, we will define a function that allows us to process all of our data at once- we bring it all together here.

1 def create\_table(table\_data,poly\_degree):

2 table = pd.DataFrame({’Wavelength(nm)’:[],

3 ’Frequency(Hz)’:[],

4 ’Work Function(J)’:[],

5 ’Stopping Potential(V)’:[],

6 ’Cut-off Frequency(Hz)’:[],

7 ’Cut-off Wavelength(nm)’:[]})

8 for wavelength in table\_data:

9 derived\_values = process\_data(table\_data[wavelength],xcol,ycol,wavelength, poly\_degree)

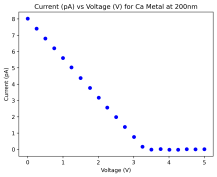
10 derived\_values = {key : [value] for key,value in derived\_values.items()} 11 row = pd.DataFrame(derived\_values)

12 numeric\_values = pd.to\_numeric(row.values.flatten(), errors=’coerce’) 13 if not np.isnan(numeric\_values).any():

14 table = pd.concat([table,row])

15 return table

For your answer to this section, display the table generated by the last function by simply executing a line with only the variable containing it. The output should look something like this:



Include screenshots of your linear and polynomial fit DataFrames in this manner.

**Part 8**

In this section, we will plot this tabulated data to obtain Planck’s constant. This will be rather simple; we need only re-use the functions we already have for obtaining coefficients and plotting.

1 new\_xcol = ’Frequency(Hz)’

2 new\_ycol = ’Stopping Potential(V)’

3 title = f’{new\_ycol} vs {new\_xcol}’

4 lin\_coeff = np.polyfit(table\_lin[new\_xcol],table\_lin[new\_ycol],1)

5 lin\_coeff

6 plot\_data(table\_lin,new\_xcol,new\_ycol,’Frequency vs. Stopping Potential’,lin\_coeff, output\_file=’PlanckRegression.png’)

8

7

8 lin\_coeff\_poly = np.polyfit(table\_poly[new\_xcol],table\_poly[new\_ycol],1) 9 plot\_data(table\_poly,new\_xcol,new\_ycol,’Frequency vs. Stopping Potential’,lin\_coeff\_poly, output\_file=’PlanckRegression.png’)

Find the calculated Planck’s constant from the linear coefficients obtained by regression. Compare this to literature values.

For your answer to this part, include plots of frequency vs. stopping potential for the tables obtained by both linear and polynomial fits. Print the value of Planck’s constant obtained for each, and calculate percent error for each.

**Concluding Questions**

Which method of fitting was better? Why do you think this is?

How does using Python to process data compare to using tools like Excel?

How did we transform our raw data into the data we needed? How could we articulate what we did as a generally applicable problem solving process?