```python
import astropy.units as u
import astropy.constants as c
from astropy.coordinates import SkyCoord
from astropy.time import Time
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import glob
from astropy.visualization import time_support
from astropy import constants as const
%matplotlib inline
```

```python
plt.rcParams["figure.figsize"] = (30, 10)
plt.rc('axes', labelsize=15)
plt.rc('axes', labelweight='bold')
plt.rc('axes', titlesize=15)
plt.rc('axes', titleweight='bold')
plt.rc('font', family='sans-serif')
```

Problem 1

Problem 1: Plotting experimental data

In the on-line resources you will find a file called sunspots.txt, which contains the observed number of sunspots on the Sun for each month since January 1749. The file contains two columns of numbers, the first being the month and the second being the sunspot number.

a) Write a program that reads in the data and makes a graph of sunspots as a function of time.

b) Modify your program to display only the first 1000 data points on the graph.

c) Modify your program further to calculate and plot the running average of the data, de- fined by

$Y_k = \frac{1}{2r}\sum_{m=-r}^{r} y_{k+m}$

where r = 5 in this case (and the yk are the sunspot numbers). Have the program plot both the original data and the running average on the same graph, again over the range covered by the first 1000 data points.

a) Write a program that reads in the data and makes a graph of sunspots as a function of time.

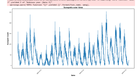
```python
def read_sunspots(path):
    """
    Reads in a file containing information about sunspot data and produces a plot of the sunspots over time

    Parameters
    ----------
    path: string, absolute path to the sunspot file

    Returns
    -------
    Produces a matplotlib plot of the sunspots over time. Sunspot count is the number of sunspots observed
    in a given month since 01-01-1749.
    """
    #Define an empty list to store the dates
    dates = []
    #Read in the file as a pandas DataFrame, indicate that the file contains no headers
    df = pd.read_table(path, header = None)
    #Months are indicated in the first column and spot counts are indicated in the second
    months = df[0].values
    spots = np.array(df[1].values)
    #Now I begin constructing a list of astropy Time objects to store the dates that these sunspots were
    #reported. I know that you probably just want me to plot the spots vs months as they are, but
    #this would technically not be correct since each month has a different length. To account for
    #month length variation and leap years, the easiest way to produce the correct plot is to use astropy
    #Time objects.
    #Initialize the first date at January 1st 1749
    string_time = "1749-01-01"
    #Add this to the list of date strings
    dates.append(string_time)
    #Loop through the a range the same length as the number of months minus the initialized month
    for month in range(len(months)-1):
        #Grab the month and the year from the string_time, assume data is always reported on the 1st of the month
        m = int(string_time.split("-")[1])
        yr = int(string_time.split("-")[0])
        #If the month is December, increase the year by one and reset the month to January
        if m == 12:
            yr += 1
            m = 1
            #If the month is of a single digit,add a 0 to the front of it in the string
            if len(str(m)) == 1:
                string_time = f"{yr}-0{m}-01"
                dates.append(string_time)
            #If the month is two digits, do not add an extra zero.
            else:
                string_time = f"{yr}-{m}-01"
                dates.append(string_time)
        #If the month is not december, only increase the month by one.
        else:
            m+=1
            if len(str(m)) == 1:
                string_time = f"{yr}-0{m}-01"
                dates.append(string_time)
            else:
                string_time = f"{yr}-{m}-01"
                dates.append(string_time)
    #Set the figure size to be 20x10
    plt.rcParams["figure.figsize"] = (30, 10)
    #Initialize matplotlib Time object plotting support
    time_support()
    #Plot the dates vs the sunspots
    plt.plot(Time(dates, format = 'iso'), spots)
    #Rotate the axis ticks so the dates are easier to read
    plt.xticks(rotation = 45)
    #Add minor ticks to the x axis
    plt.tick_params('x', length=10, width=1, which='minor')
    #Add a title and x and y labels
    plt.title("Sunspots over time")
    plt.xlabel("Date")
    plt.ylabel("Sunspot Count");
```

```python
read_sunspots(path = "/home/kelsey/computational_phys/Homework2/sunspots.txt")
```

```
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "dtf2d" yielded 2532 of "dubious year (Note 6)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "d2dtf" yielded 1 of "dubious year (Note 5)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "dtf2d" yielded 1 of "dubious year (Note 6)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "d2dtf" yielded 2 of "dubious year (Note 5)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
```



b) Modify your program to display only the first 1000 data points on the graph.

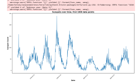
```python
def read_sunspots_first_1000(path):
    """
    Reads in a file containing information about sunspot data and produces a plot of the sunspots over time
    for the first 1000 data points.

    Parameters
    ----------
    path: string, absolute path to the sunspot file

    Returns
    -------
    Produces a matplotlib plot of the sunspots over time. Sunspot count is the number of sunspots observed
    in a given month since 01-01-1749.
    """
    #Define an empty list to store the dates
    dates = []
    #Read in the file as a pandas DataFrame, indicate that the file contains no headers
    df = pd.read_table(path, header = None)
    #Months are indicated in the first column and spot counts are indicated in the second
    months = df[0].values[0:1000]
    spots = np.array(df[1].values)[0:1000]
    #Initialize the first date at January 1st 1749
    string_time = "1749-01-01"
    #Add this to the list of date strings
    dates.append(string_time)
    #Loop through the a range the same length as the number of months minus the initialized month
    for month in range(len(months)-1):
        #Grab the month and the year from the string_time, assume data is always reported on the 1st of the month
        m = int(string_time.split("-")[1])
        yr = int(string_time.split("-")[0])
        #If the month is December, increase the year by one and reset the month to January
        if m == 12:
            yr += 1
            m = 1
            #If the month is of a single digit,add a 0 to the front of it in the string
            if len(str(m)) == 1:
                string_time = f"{yr}-0{m}-01"
                dates.append(string_time)
            #If the month is two digits, do not add an extra zero.
            else:
                string_time = f"{yr}-{m}-01"
                dates.append(string_time)
        #If the month is not december, only increase the month by one.
        else:
            m+=1
            if len(str(m)) == 1:
                string_time = f"{yr}-0{m}-01"
                dates.append(string_time)
            else:
                string_time = f"{yr}-{m}-01"
                dates.append(string_time)
    #Set the figure size to be 20x10
    plt.rcParams["figure.figsize"] = (30, 10)
    #Initialize matplotlib Time object plotting support
    time_support()
    #Plot the dates vs the sunspots
    plt.plot(Time(dates, format = 'iso'), spots)
    #Rotate the axis ticks so the dates are easier to read
    plt.xticks(rotation = 45)
    #Add minor ticks to the x axis
    plt.tick_params('x', length=10, width=1, which='minor')
    #Add a title and x and y labels
    plt.title("Sunspots over time, first 1000 data points")
    plt.xlabel("Date")
    plt.ylabel("Sunspot Count");
```

```python
read_sunspots_first_1000(path = "/home/kelsey/computational_phys/Homework2/sunspots.txt")
```

```
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "dtf2d" yielded 1000 of "dubious year (Note 6)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "d2dtf" yielded 1 of "dubious year (Note 5)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "dtf2d" yielded 5 of "dubious year (Note 6)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "d2dtf" yielded 4 of "dubious year (Note 5)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
```



c) Modify your program further to calculate and plot the running average of the data, de- fined by

$Y_k = \frac{1}{2r}\sum_{m=-r}^{r} y_{k+m}$

where r = 5 in this case (and the yk are the sunspot numbers). Have the program plot both the original data and the running average on the same graph, again over the range covered by the first 1000 data points.

```python
def read_sunspots_running_average_first_1000(path):
    """
    Reads in a file containing information about sunspot data and produces a plot of the sunspots over time

    Parameters
    ----------
    path: string, absolute path to the sunspot file

    Returns
    -------
    Produces a matplotlib plot of the sunspots over time. Sunspot count is the number of sunspots observed
    in a given month since 01-01-1749.
    """
    #Define an empty list to store the dates
    dates = []
    #Read in the file as a pandas DataFrame, indicate that the file contains no headers
    df = pd.read_table(path, header = None)
    #Months are indicated in the first column and spot counts are indicated in the second
    months = df[0].values[0:1000]
    spots = np.array(df[1].values)[0:1000]
    #Initialize the first date at January 1st 1749
    string_time = "1749-01-01"
    #Add this to the list of date strings
    dates.append(string_time)
    #Define an empty list to hold the running averages
    yks = []
    #Define the r variable
    r=5
    #Loop through each position in the sunspots data
    for pos in range(len(spots)):
        #Create an interval for the average of +/- 5 data points around the current position
        data_interval = spots[pos-r:pos+r]
        #Compute the yk value and append it to the running list of values for the plot
        yks.append(np.sum(data_interval)*(1/(2*r)))
    #Loop through the a range the same length as the number of months minus the initialized month
    for month in range(len(months)-1):
        #Grab the month and the year from the string_time, assume data is always reported on the 1st of the month
        m = int(string_time.split("-")[1])
        yr = int(string_time.split("-")[0])
        #If the month is December, increase the year by one and reset the month to January
        if m == 12:
            yr += 1
            m = 1
            #If the month is of a single digit,add a 0 to the front of it in the string
            if len(str(m)) == 1:
                string_time = f"{yr}-0{m}-01"
                dates.append(string_time)
            #If the month is two digits, do not add an extra zero.
            else:
                string_time = f"{yr}-{m}-01"
                dates.append(string_time)
        #If the month is not december, only increase the month by one.
        else:
            m+=1
            if len(str(m)) == 1:
                string_time = f"{yr}-0{m}-01"
                dates.append(string_time)
            else:
                string_time = f"{yr}-{m}-01"
                dates.append(string_time)
    #Set the figure size to be 20x10
    plt.rcParams["figure.figsize"] = (30, 10)
    #Initialize matplotlib Time object plotting support
    time_support()
    #Plot the dates vs the sunspots
    plt.plot(Time(dates, format = 'iso'), spots, label = "Raw Data")
    #Plot the running average of the sunspots
    plt.plot(Time(dates, format = 'iso'), yks, label = "Running Average", c = 'k', lw = 3)
    #Rotate the axis ticks so the dates are easier to read
    plt.xticks(rotation = 45)
    #Add minor ticks to the x axis
    plt.tick_params('x', length=10, width=1, which='minor')
    #Add a title and x and y labels
    plt.title("Sunspots over time, first 1000 points")
    plt.xlabel("Date")
    plt.ylabel("Sunspot Count")
    #Display the legend
    plt.legend();
```

```python
path = "/home/kelsey/computational_phys/Homework2/sunspots.txt"
```

```python
read_sunspots_running_average_first_1000(path)
```

```
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "dtf2d" yielded 1000 of "dubious year (Note 6)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "dtf2d" yielded 1000 of "dubious year (Note 6)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "d2dtf" yielded 1 of "dubious year (Note 5)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "dtf2d" yielded 5 of "dubious year (Note 6)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
/home/kelsey/anaconda3/envs/hera/lib/python3.8/site-packages/erfa/core.py:154: ErfaWarning: ERFA function "d2dtf" yielded 4 of "dubious year (Note 5)"
  warnings.warn('ERFA function "{}" yielded {}'.format(func_name, wmsg),
```

Problem 2

Problem 2: Scanning tunneling microscope There is a file in the on-line resources called stm.txt, which contains a grid of values from scanning tunneling microscope measurements of the (111) surface of silicon.

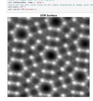
A scanning tun- neling microscope (STM) is a device that measures the shape of a surface at the atomic level by tracking a sharp tip over the surface and measuring quantum tunneling current as a function of position. The end result is a grid of values that represent the height of the surface and the file stm.txt contains just such a grid of values.

Write a program that reads the data contained in the file and makes a density plot of the values. Use the various options and variants you have learned about to make a picture that shows the structure of the silicon surface clearly.

```python
#Set the figure size to be 10x10
plt.rcParams["figure.figsize"] = (10, 10)
```

```python
#Load in the file with a pandas dataframe
stm = pd.read_table('/home/kelsey/computational_phys/Homework2/stm.txt', header = None)[0]
#Create a matrix of zeroes with dimensions of the number of x pixels and the number of y pixels
data = np.zeros([len(stm), len(stm[0].split(' '))])
#Loop through the enumerated dataframe
for xi, x in enumerate(stm):
    #Fill in each row in the data matrix with the corresponding row from the dataframe. This is necessary becau
    #the DataFrame reads in the text file as strings, and we need to indicate that the numbers in these strings
    #are separated by spaces and they should be interpreted as floats.
    ys = np.array(x.split(' '), dtype = float)
    for i in range(len(ys)):
        data[xi,i]=ys[i]
```

```python
#Show the image, make the color map grey
plt.imshow(data, cmap = "gray")
#Turn off the axis ticks since we are simply displaying an image, give the plot a title
plt.axis("off")
plt.title("STM Surface");
```



Porblem 3

Problem 3: The Mandelbrot set

The Mandelbrot set, named after its discoverer, the French mathematician Benoˆıt Mandelbrot, is a fractal, an infinitely ramified mathematical object that contains structure within structure within structure, as deep as we care to look. The definition of the Mandelbrot set is in terms of complex numbers as follows.

Consider the equation $z' = z^2 + c$

where z is a complex number and c is a complex constant. For any given value of c this equa- tion turns an input number z into an output number z'. The definition of the Mandelbrot set involves the repeated iteration of this equation: we take an initial starting value of z and feed it into the equation to get a new value z'. Then we take that value and feed it in again to get another value, and so forth.

The Mandelbrot set is the set of points in the complex plane that satisfies the following definition:

For a given complex value of c, start with z = 0 and iterate repeatedly. If the magnitude |z| of the resulting value is ever greater than 2, then the point in the complex plane at position c is not in the Mandelbrot set, otherwise it is in the set.

In order to use this definition one would, in principle, have to iterate infinitely many times to prove that a point is in the Mandelbrot set, since a point is in the set only if the iteration never passes |z| = 2 ever. In practice, however, one usually just performs some large number of iterations, say 100, and if |z| hasn't exceeded 2 by that point then we call that good enough.

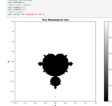
Write a program to make an image of the Mandelbrot set by performing the iteration for all values of c = x + iy on an N × N grid spanning the region where −2 ≤ x ≤ 2 and −2 ≤ y ≤2.

Make a density plot in which grid points inside the Mandelbrot set are colored black and those outside are colored white. The Mandelbrot set has a very distinctive shape that looks something like a beetle with a long snout—you'll know it when you see it.

Hint: You will probably find it useful to start off with quite a coarse grid, i.e., with a small value of N—perhaps N = 100—so that your program runs quickly while you are testing it. Once you are sure it is working correctly, increase the value of N to produce a final high-quality image of the shape of the set. If you are feeling enthusiastic, here is another variant of the same exercise that can produce amazing looking pictures. Instead of coloring points just black or white, color points according to the number of iterations of the equation before |z| becomes greater than 2 (or the maximum number of iterations if |z| never becomes greater than 2). If you use one of the more colorful color schemes Python provides for density plots, such as the "hot" or "jet" schemes, you can make some spectacular images this way. Another interesting variant is to color according to the logarithm of the number of iterations, which helps reveal some of the finer structure outside the set.

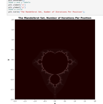
```python
#Pixels
N = 1000
#Allowed Iterations
n = 1000
#Define xspan and yspan to cover axes from -2 to 2 and contain N pixels between these values
xspan,yspan == np.linspace(-2,2,N)
#Define an empty matrix to store the data. It should have dimensions of the number of pixels by the number
#of pixels
matrix = np.zeros([N,N])
#enumerate the xspan and yspan variables so we can loop through each position in the matrix. xi and yi
#are the indices and x and y are the respective x and y values in the matrix
for xi, x in enumerate(xspan):
    for yi, y in enumerate(yspan):
        #Initialize z to be 0
        z = complex(0,0)
        #Loop from 0 to the maximum allowed iterations
        for i in range(0,n):
            #Define c to be a complex number indicated by the position on the x and y axes
            c = complex(x,y)
            #Increase z by z squared and c
            z = (z**2) + c
            #If at any point the absolute value of z is greater than 2, break the loop and add 1 to the
            #x and y position in the matrix
            if abs(z)<2:
                continue
            else:
                matrix[xi,yi] +=1
                break
```

```python
#Use imshow to display the matrix. Set the origin to be the lower right corner and define the extent to
#indicate the range over which x and y were defined
plt.imshow(matrix, origin = 'lower', cmap = 'gray', extent=[-2,2,-2,2])
#Display the colorbar
plt.colorbar()
#Add x and y labels
plt.xlabel('x')
plt.ylabel('y')
#Add a title
plt.title('The Mandelbrot Set');
```




```python
#Allowed iterations
n = 1000
#Pixels
N = 1000
#Define xspan and yspan to cover axes from -2 to 2 and contain N pixels between these values
xspan, yspan == np.linspace(-2,2,N)
#Define an empty matrix to store the data. It should have dimensions of the number of pixels by the number
#of pixels
matrix2 = np.zeros([N,N])
#enumerate the xspan and yspan variables so we can loop through each position in the matrix. xi and yi
#are the indices and x and y are the respective x and y values in the matrix
for xi, x in enumerate(xspan):
    for yi, y in enumerate(yspan):
        #Initialize z to be 0
        z = complex(0,0)
        #Loop from 0 to the maximum allowed iterations
        for i in range(0,n):
            #Define c to be a complex number indicated by the position on the x and y axes
            c = complex(x,y)
            #Increase z by z squared and c
            z = (z**2) + c
            #If at any point the absolute value of z is greater than 2, break the loop and add the number
            #of iterations to the x and y position in the matrix
            if abs(z) < 2:
                continue
            else:
                matrix2[xi,yi] +=i
                break
```

```python
#Use imshow to display the matrix. Set the origin to be the lower right corner and define the extent to
#indicate the range over which x and y were defined
plt.imshow(matrix2, origin = 'lower', cmap = 'pink', extent=[-2,2,-2,2])
#Display the colorbar
plt.colorbar()
#Add x and y labels
plt.xlabel('x')
plt.ylabel('y')
#Add a title
plt.title('The Mandelbrot Set, Number of Iterations Per Position');
```



Problem 4

Problem 4: Least-squares fitting and the photoelectric effect

It's a common situation in physics that an experiment produces data that lies roughly on a straight line, like the dots in this figure:

The solid line here represents the underlying straight-line form, which we usually don't know, and the points representing the measured data lie roughly along the line but don't fall exactly on it, typically because of measurement error.

The straight line can be represented in the familiar form y = mx + c and a frequent ques- tion is what the appropriate values of the slope m and intercept c are that correspond to the measured data.

Since the data don't fall perfectly on a straight line, there is no perfect answer to such a question, but we can find the straight line that gives the best compromise fit to the data. The standard technique for doing this is the method of least squares. Suppose we make some guess about the parameters m and c for the straight line. We then calculate the vertical distances between the data points and that line, as represented by the short vertical lines in the figure, then we calculate the sum of the squares of those distances, which we denote χ2.

If we have N data points with coordinates (xi, yi), then χ2 is given by

$\Chi^2 = \sum_{i=1}^{N}(mx_i + c - y_i)^2$

The least-squares fit of the straight line to the data is the straight line that minimizes this total squared distance from data to line. We find the minimum by differentiating with respect to both m and c and setting the derivatives to zero, which gives

see homework

For convenience, let us define the following quantities:

see hw

in terms of which our equations can be written

...

Solving these equations simultaneously for m and c now gives

...

These are the equations for the least-squares fit of a straight line to N data points. They tell you the values of m and c for the line that best fits the given data.

a) In the on-line resources you will find a file called millikan.txt. The file contains two columns of numbers, giving the x and y coordinates of a set of data points. Write a program to read these data points and make a graph with one dot or circle for each point.

b) Add code to your program, before the part that makes the graph, to calculate the quanti- ties Ex, Ey, Exx, and Exy defined above, and from them calculate and print out the slope m and intercept c of the best-fit line.

c) Now write code that goes through each of the data points in turn and evaluates the quan- tity mxi + c using the values of m and c that you calculated. Store these values in a new array or list, and then graph this new array, as a solid line, on the same plot as the orig- inal data. You should end up with a plot of the data points plus a straight line that runs through them.

d) The data in the file millikan.txt are taken from a historic experiment by Robert Millikan that measured the photoelectric effect. When light of an appropriate wavelength is shone on the surface of a metal, the photons in the light can strike conduction electrons in the metal and, sometimes, eject them from the surface into the free space above. The energy of an ejected electron is equal to the energy of the photon that struck it minus a small amount φ called the work function of the surface, which represents the energy needed to remove an electron from the surface. The energy of a photon is hν, where h is Planck's constant and ν is the frequency of the light, and we can measure the energy of an ejected electron by measuring the voltage V that is just sufficient to stop the electron moving. Then the voltage, frequency, and work function are related by the equation V = h e ν − φ, where e is the charge on the electron. This equation was first given by Albert Einstein in 1905. The data in the file millikan.txt represent frequencies ν in hertz (first column) and volt- ages V in volts (second column) from photoelectric measurements of this kind. Using the equation above and the program you wrote, and given that the charge on the electron is 1.602 × 10−19 C, calculate from Millikan's experimental data a value for Planck's con- stant. Compare your value with the accepted value of the constant, which you can find in books or on-line. You should get a result within a couple of percent of the accepted value. 4 This calculation is essentially the same as the one that Millikan himself used to determine of the value of Planck's constant, although, lacking a computer, he fitted his straight line to the data by eye. In part for this work, Millikan was awarded the Nobel prize in physics in 1923

a) In the on-line resources you will find a file called millikan.txt. The file contains two columns of numbers, giving the x and y coordinates of a set of data points. Write a program to read these data points and make a graph with one dot or circle for each point.

```python
#Read in the data from the text file as a pandas dataframe
mil = pd.read_table('/home/kelsey/computational_phys/Homework2/millikan.txt', header = None, delimiter = ' ')
#Let x be the first comulmn of the data and y be the second
x = mil[0]
y = mil[1]
```

```python
#Define the figure to be 10x10
plt.rcParams["figure.figsize"] = (10, 10)
#Create a scatterplot of the data
plt.scatter(x,y)
#Add axes labels and a title
plt.title('Y(X)')
plt.xlabel('X')
plt.ylabel('Y');
```

b) Add code to your program, before the part that makes the graph, to calculate the quanti- ties Ex, Ey, Exx, and Exy defined above, and from them calculate and print out the slope m and intercept c of the best-fit line.

```python
#Let N be the number of data points
N = len(x)
#Calculate the Ex quantities
Ex = (1/N)*sum(x)
Ey = (1/N)*sum(y)
Exx = (1/N)*sum(x**2)
Exy = (1/N)*sum(x*y)
#Print the quantities
print(f'Ex = {Ex}, Ey = {Ey}, Exx = {Exx}, Exy = {Exy}')
#Compute the slope and y intercept from the Ex values
m = ((Exy - Ex*Ey)/(Exx- Ex*Ex))
c = ((Exx*Ey - Ex*Exy)/(Exx - Ex*Ex))
#Print the slope and y intercept
print(f'm = {m}, c = {c}')
#Print out the graph again because the question seemed to ask for it
plt.scatter(x,y)
plt.title('Y(X)')
plt.xlabel('X')
plt.ylabel('Y');
```