AST325H1F Lab 2: Introduction to Spectroscopy

Parampreet Singh
parampreet.singh@mail.utoronto.ca
Lab Group G

November 6th, 2019

Abstract

In this lab report, a simple, visible light (350 - 700 nm) spectrometer was used to explore the spectra of laboratory and astrophysical sources. A wavelength calibration of the Ocean Optics spectrometer was conducted, which introduced the concept of spectroscopy and linear least square fitting. Also analyzed was observed data from an On-Campus Telescope spectrometer and observed spectra of astronomical sources. In the process, the basic steps of astronomical data reduction (e.g., dark subtraction, flat fielding, wavelength solution) were conduced to respective data sets. It will be shown that spectral data collected from the Ocean Optics spectrometer and the On-Campus Telescope will clearly exhibit a linear wavelength solution for the extent of this lab. This is done by investigating the relationship between Pixel Number and Wavelength through various Python programming analysis of the peaks, valleys, and centroids of the Neon spectra. Also explored are potential sources of error and possible reduction methods for these errors.

1 Introduction

Spectroscopy is a fundamental tool used in all physical sciences. Spectroscopy is the technique of splitting electromagnetic radiation into its constituent wavelengths (a spectrum). For astrophysicists, spectroscopy is essential for defining the physical nature of celestial objects and the universe. Astronomical spectroscopy has been and still is being used to measure the chemical composition and physical conditions (temperature, pressure, and magnetic field strength) in planets, stars, and galaxies.

Characterizing a spectrograph's instrumental parameters is a key for deriving the intrinsic spectra from a source. Spectrometers are instruments which employ the technique of spectroscopy. This allows an observer to be able to read spectral data given off by a given source. For this lab, a USB 2000 spectrometer manufactured by Ocean Optics was used. The guild for the USB 2000 spectrometer can be found on the AST325 course website [1]. The USB 2000 spectrometer is a type of Charge Couple Device (CCD) which is composed of a one-dimensional array of CCD pixels. When a photon (with a proper wavelength range) hits this array, it creates an electron (often called photo-electron) due to the photoelectric effect [2]. We measure the number of electrons for the signal intensity. For more information on the features of CCDs, refer to the AST325 course website [2]. It is also well noted that this process of spectral observation is not perfect. CCDs have flaws that introduce various systematic errors with the process of measurement which will be explored in Section 2.

In this experiment, we attempt to determine a wavelength solution to the Ocean Optics USB 2000 spectrometer and compare this to the wavelength solution of an On-Campus Telescope. This is done through applying the method of Linear Least Square fitting to Neon spectral data sets. It will be shown that a linear wavelength solution is the most appropriate fit for the given data (when modeled with Pixel Number vs. Wavelength). Various methods of data importation and manipulation through Python programming will also be explored.

The division of labour this lab was split evenly between the Author (Parampreet Singh) and Lab Group G members Nicholas Clark, Juan Pablo Alfonzo, and Lucas Louwerse. The methods and codes used to obtain and analyze data were determined as a partnership between all four group members. Individual members had their own approach to compile the spectral data and produced their own unique results, plots, and fit calculations. These were compared between group members to ensure the accuracy of the experiment.

2 Observations and Data

The first set of observations for this lab were collected from an Ocean Optics USB 2000 spectrometer based on a diffraction grating and a one-dimensional CCD detector array. This CCD has 1 x 2048 pixels so the spectrum reads out as a list of 2048 data numbers. Members of Lab Group G simultaneously conducted measurements using this instrument inside of the Undergraduate Laboratory located inside the Department of Astronomy and Astrophysics within the University of Toronto St. George campus. Greater details of the mechanisms for the USB 2000 device can be found on the AST325 course website (under "Spectrometer Guide") [1].

The second set of observations for this lab were collected from a spectrograph with a two-dimensional CCD array, which offers lower noise and is therefore better suited to conduct night time work. This spectrograph is located on the 16" telescope on the 16th floor of the McLennan Physical Laboratories Tower located within the University of Toronto St. George campus. Measurements were made under the discretion of one of the campus observers (a teaching assistant for the AST325 course). Members of Lab Group G simultaneously conducted measurements using this instrument. Greater details of the mechanisms for the 16" telescope and the two-dimensional CCD which was used can be found on the AST325 course website (under "Spectroscopic Measurements of Stars") [1]. Note, although a full set of two-dimensional data points were taken, only one flat line of data from the entire two-dimensional set of spectral data was used. This reduced the two-dimensional data to that of one-dimension, which can be compared, plotted, and manipulated more easily.

In total, 11 spectra measurements were conducted. 5 corresponding to measurements made with the USB 2000 spectrometer (Incandescent Bulb, Neon, Hydrogen, Fluorescent Bulb, and the Sun) and 6 corresponding to measurements made with the On-Campus Telescope spectrometer (Neon Calibration and Dark Calibration as well as stars Enif, Navi, Scheat, and Vega). To illustrate collected data, one data set from the USB 2000 spectrometer and one data set from the On-Campus Telescope spectrometer will be shown. This is because data sets corresponding the same CCD spectrometer greatly resemble each other. This is done to maintain directness.

Table 1: Hydrogen Spectra Measurements And Vega Spectra Measurements.

Beginning of Data			
Hydrogen Pixel Number	Hydrogen Measured Signal	Navi Pixel Number	Navi Measured Signal
1	531.3	1	1696
... 637	6820.32	... 134	5010
... 1691	53420.4	... 532	2252
... 2048	678.76	... 764	1288
End of Data			

Shown in Table 1 are samples of collected data from the USB 2000 (columns 1 and 2) and On-Campus Telescope (columns 3 and 4) spectrometers. Many trials were conducted for each experiment so the tables have been shortened in order to effectively display the data. Data sets for this lab were common between all group members, meaning each individual in the Author's group will have all four of the same data sets.

Like mentioned in Section 1, there are various flaws that are present with collecting spectral data from CCDs.

The USB 2000 data was collected by hand (by the use of a fiber optical tube) in a laboratory which is not concealed from outside detection of various electromagnetic sources. This means all data is somewhat tainted with spectral data from other sources than those directly measured. This factor hinders the accuracy of the data which was collected for each light source.

The On-Campus Telescope data was collected systematically at a location where outside sources of electromagnetic radiation were more negligible than those of the USB 2000 laboratory. However, this method of data collection introduced an external source (outside of those being measured) which was able to interfere with the required spectral data, called "Darks". The term "Darks" refers to the electromagnetic radiation given off by the CCD sensor itself (as forms of heat, etc.). Knowing this was prevalent, it was easy to isolate this source and determine exactly how this source was interacting with collected spectral data. The "Dark"

measurement is displayed in Figure 1. Besides this error factor (due to the equipment) the collected spectra was the most accurate that it could be for the given resources and conditions.

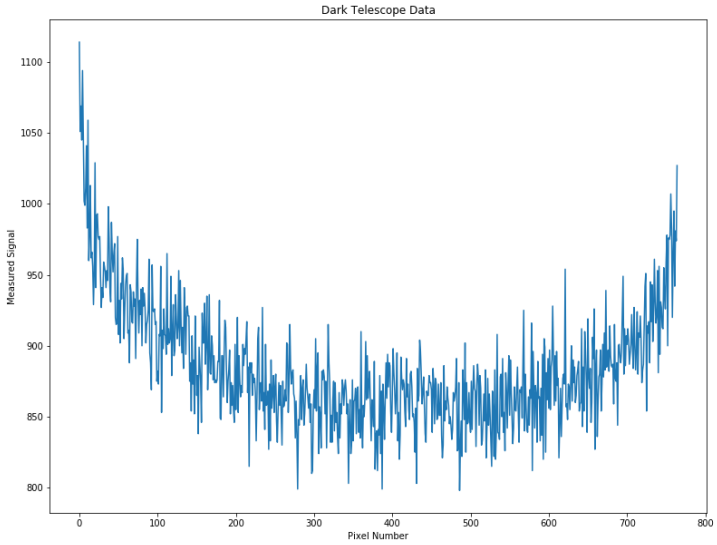

Figure 1: Telescope Spectrometer Dark Measurment

3 Data Reduction and Methods

For this lab the collected data was analyzed using numerous Python coding packages. These include `matplotlib`, `numpy`, `pandas`, and `astropy` , as well as many algorithms and functions, created by the Author, which heavily incorporate these packages for the purpose of processing the collected data and providing the required analysis. For this lab in particular, the required tasks were to perform a reduction of data following instructions which were given through a lab handout. What is unique is how each group member went about tackling the method for reducing and processing their data. The main goal for this lab was to determine the wavelength solutions of both the USB 2000 and the On-Campus Telescope spectrometers. This includes performing tasks such as generating plots to visualize spectral data and computing the peaks, valleys, and centroids, and to apply this with the method of linear least square fitting to determine a wavelength solution.

The reduction of the given data mainly focused on computing the peaks, valleys, and centroids. These quantities are best visualized through generating simple plots, which was the main spectral data visualization technique that was used for the purpose of this lab, combined with plots for linear least square fitting, which is discussed in Section 5.2. Throughout the lab, it was required that for a given spectral data set, the peaks, valleys, and centroids should be calculated. For all of these calculations, Python programming codes were implemented in order to manipulate the data in order calculate for a given quantity. The calculations were based on Python algorithms which can be found under Appendix B.

Displayed in Figure 2 is a plot of the Neon spectra from a USB 2000 measurement. It shows the relationship between Pixel Number and Measured Signal. Displayed in red are the peak values and peak locations, displayed in blue are the valley values and valley locations, finally, displayed as red dashed lines are the centroids and their locations. All of the previous quantities were found through algorithms created by the Author which can be found under Appendix B. These algorithms were also conducted for the Hydrogen spectra (USB 2000) and the Neon spectra (On-Campus Telescope) in a similar manner.

There are possible ways to shorten the amount of calculation steps that are required for these quantities

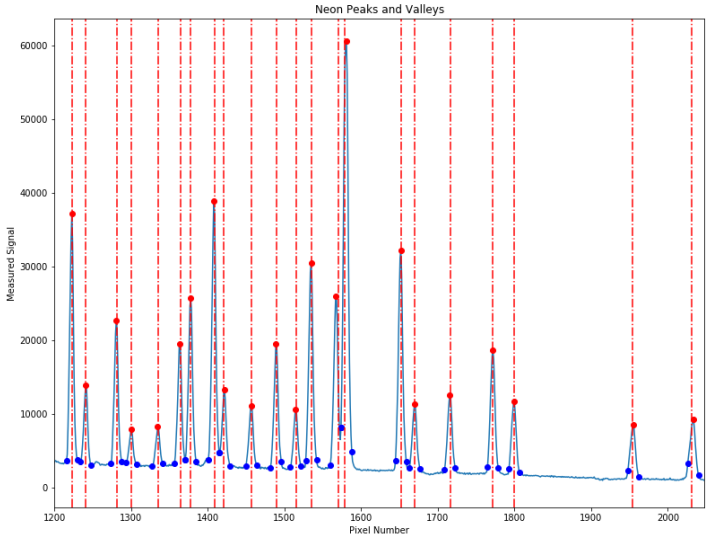

Figure 2: USB 2000 Neon Peaks, Valleys, and Centroids

through various shortcuts in Python programming, however, it was required that all algorithms be created by the Author to demonstrate understanding and to gain experience in Python programming. All calculation steps were shown thoroughly through code which is provided in Appendix B. The method of linear least square fitting was also calculated completely by algorithms created by the Author which uses the quantities determined by algorithms from Appendix B, this is discussed in Section 5.1 and Section 5.2.

4 Data Analysis and Modeling

4.1 Data Modeling

Recall that one of the main objectives for this lab was to perform the method of linear least square fitting to determine the wavelength solution for each spectrometer. To do this, it was crucial to determine the peaks, valleys, and centroids of at least one of the observed spectra for each spectrometer prior to investigating anything about the wavelength solution. These calculations are thoroughly described in Section 3 and Section 5.1.

Now that the peaks, valleys, and centroids are present for an observed spectral data set, it is feasible to conduct the respectable linear least square fit. The linear least square fit was determined for both Neon spectra measurements for the USB 2000 and the On-Campus Telescope through the following formula where m represents the slope of the linear fit, c represents the intercept of the linear fit, x_i represents the position of the i^{th} centroid, y_i represents the predicted wavelength which can be found on the AST325 course website (under "Ocean Optics Calibration Data") [1], and N represents the number of centroids:

$$\begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & N \end{bmatrix}^{-1} \begin{bmatrix} \sum x_i y_i \\ \sum y_i \end{bmatrix} \tag{1}$$

Using the calculated m and c values, it is possible to determine a line of best fit with the following formula where y represents the wavelength:

$$y = mx_i + c \tag{2}$$

The conclusions of this linear least square fitting will be more explored in Section 5.2.

4.2 Error Propagation

Note that the calculations for m and c produce uncertainties. These uncertainties are explored using the following method of error prorogation.

First, the uncertainty of m is given by the following formula where σ_m represents the uncertainty in m, σ represents the standard deviation, x_i represents the positions of the i^{th} centroid, and N represents the number of centroids:

$$\sigma_m^2 = \frac{N\sigma^2}{N\sum x_i^2 - (\sum x_i)^2} \tag{3}$$

Second, the uncertainty of c is given by the following formula where σ_c represents the uncertainty in c, σ represents the standard deviation, x_i represents the positions of the i^{th} centroid, and N represents the number of centroids:

$$\sigma_c^2 = \frac{\sigma^2 \sum x_i^2}{N\sum x_i^2 - (\sum x_i)^2} \tag{4}$$

Third, if the standard deviation is not known, a priori, the best estimate is derived from the deviations from the linear fit where σ represents the standard deviation and all other variables were previously defined:

$$\sigma^2 = \frac{1}{N-2}\sum [y_i - (mx_i + c)]^2 \tag{5}$$

Lastly, to determine the goodness of the linear model, the R^2 value was determined for each linear regression in Section 5.2. R^2 represents the goodness of the linear model, y represents the wavelength as calculated in Eq. 2, y_i represents the predicted wavelength, and $\bar{y}$ represents the average of all wavelengths produced by Eq. 2:

$$R^2 = 1 - \frac{\sum (y - y_i)^2}{\sum (y_i - \bar{y})^2} \tag{6}$$

The Python programming code for how the linear least square fitting, error propagation, and R^2 values were applied to a given set of spectral data set are given in Appendicies C and D.

5 Discussion

5.1 Peaks, Valleys, and Centroids

The first step in determining the wavelength solution for each spectrometer was to calculate three quantities known as peaks, valleys, and centroids. These quantities were determined for one set of spectral data corresponding to each spectrometer. Peaks represent areas of local maximums of the measured signal while valleys represent areas of local minimums of the measured signal around the peaks. Centroids can then be found as a simple weighted average point in between a peak and two valleys as displayed in Figures 2, 3, and 4. An example of the Python programming code for the calculation of these quantities can be found under Appendix B. The reason that these quantities are determined is because they are extremely important in any spectral data analysis. For the purpose of this lab, computing centroids is essential for determining a wavelength solution as the method of linear least square fitting requires centroids as an input parameter, as seen in Eq. 2. Centroids can only be determined by first determining peak and valley locations.

Shown in Figure 3 is a side by side comparison of the Neon spectra for the USB 2000 (left) and the On-Campus Telescope (right) spectrometers with calculated peaks (red dots), valleys (blue dots), and centroids (red dashed line). From Figure 3, it is visually clear how the centroid represents a weighted average of a peak and two surrounding valley points in comparison to the Python algorithm for this calculation (Appendix B).

Notice that only the x-value or Pixel Number of the centroid is considered. This is because we want a parameter with units of Pixel Number to later construct a linear best fit in accordance to Eq. 2. Another way to interpret a centroid is to simply consider it to be approximately the mid-point of two valleys. This can be done because only the Pixel Number of a centroid is considered. However, to elaborate a general

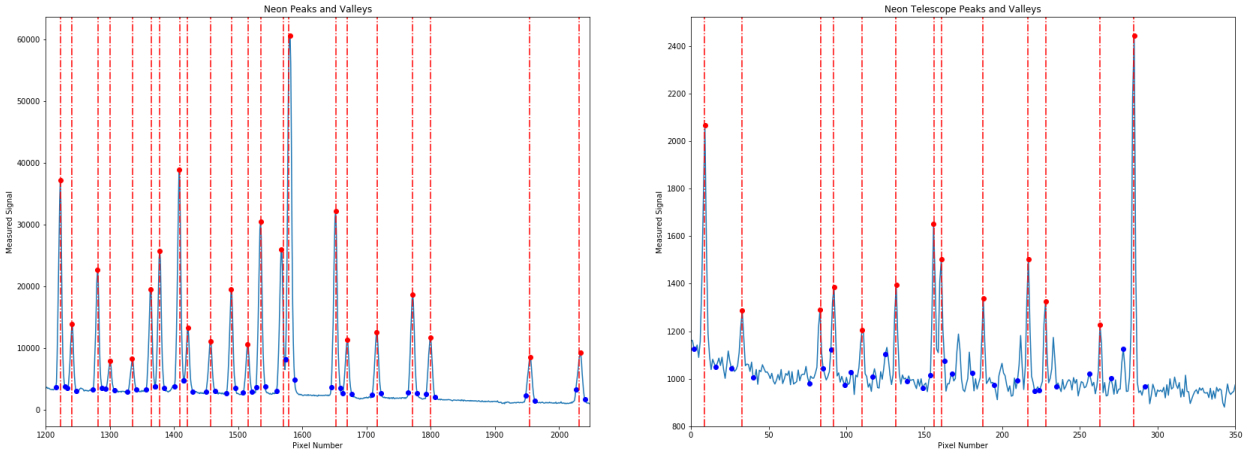

Figure 3: USB 2000 and On Campus Telescope Neon Peaks, Valleys, and Centroids

centroid calculation, the algorithm as illustrated in Appendix B is demonstrated. The significance of the centroid Pixel Number will further be illustrated in Section 5.3.

The reason that Neon is used to determine centroid positions is because the Neon spectra measurement was common for both the USB 2000 and the On-Campus Telescope spectrometers. This means the wavelength solutions are somewhat comparable. Neon also contains many peak positions (emission spectra) than elements such as Hydrogen, which could also have been used. Hydrogen was not used because this element contains less peak positions than Neon. This means there would not be as many centroid positions to study and consider into the line of best fit and ultimately the wavelength solution. This would lead to a less accurate line of best fit as there would be a decrease in the number of data points which are able to be considered.

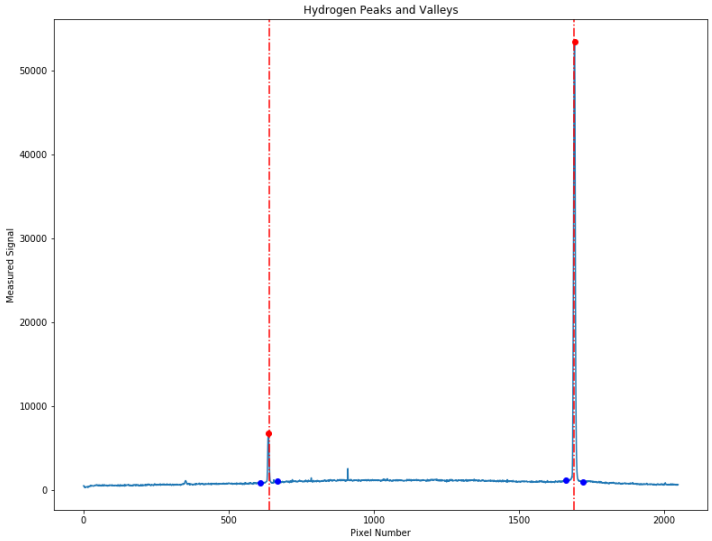

Figure 4: USB 2000 Hydrogen Peaks, Valleys, and Centroids

To illustrate this further, consider Figure 4 for the calculated Hydrogen peaks, valleys, and centroids. As you can see, there are only two centroid positions for the USB 2000 measurement for Hydrogen. If Eq. 2

were to be applied to this set of spectral data, it would result in a line constructed of two points. While for Neon, more data points are available which means the method of linear least square fitting will give us more information on the true nature of a linear best fit to determine a wavelength solution.

For further investigation, the method of determining peaks, valleys, and centroids could be applied to all measured spectra for both the USB 2000 and the On-Campus Telescope instead of just for Neon. This would help in approximating a better fit and wavelength solution for each spectrometer. For now and for the purpose of this lab, the Neon spectral data was common for both spectrometers and the choice of either using Neon or Hydrogen (or even both) was up to the Author. In the end, Neon is able to provide a sufficient linear best fit to determine the wavelength solution.

5.2 Linear Least Square Fitting

The ultimate goal of this lab was to determine the wavelength solution of two spectrometers. This is the reason why quantities such as peaks, valleys, and centroids were determined (aside from just being useful spectral analyzing tools). What will now be discussed is how quantities such as the peaks, valleys, and centroids of spectral data sets were used in order to determined the respectable wavelength solutions.

From Eq. 1, it is clear that the slope and the intercept of the linear fit can easily be found. The derivation for this equation was given in a lab handout and can be found on the AST325 course website. [1] Once the slope and the intercept are found, it is possible to construct an equation of a line using the centroids position as the input parameter which is summarized through Eq. 2. The centroid positions were determined, as shown in Figure 3, and a linear regression can be constructed for both the USB 2000 and the On-Campus Telescope spectrometers.

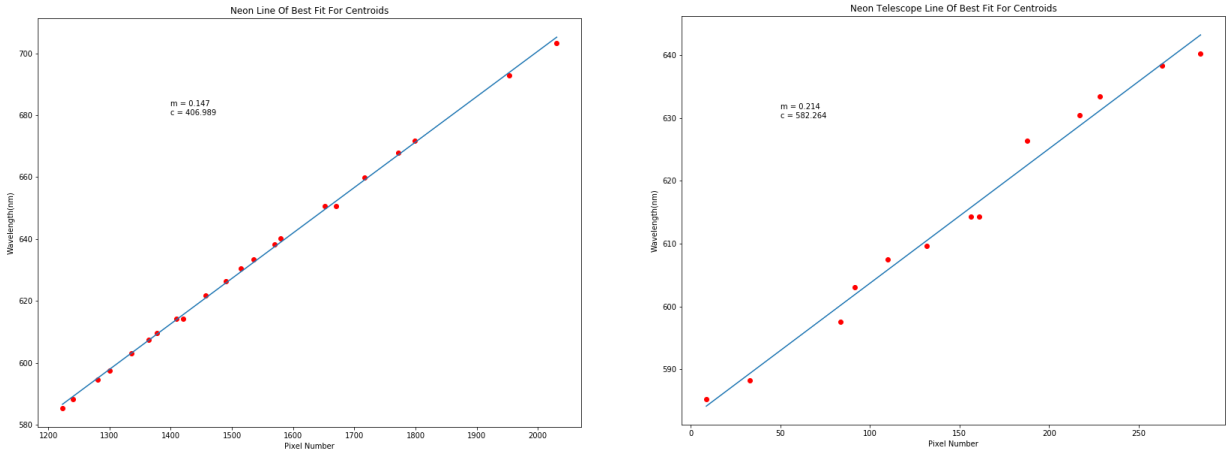

Figure 5: USB 2000 and On-Campus Telescope Linear Regression

Displayed in Figure 5 are the linear regression or linear line of best for for the USB 2000 (left) and the On-Campus Telescope (right) spectrometers. The blue line is the line which is determined from Eq. 2. The red points correspond to predicted wavelengths for a given Pixel Number which were obtained from the AST325 course website (under "Ocean Optics Calibration Data"). [1] The best fit is plotted with the relationship of Pixel Number and Wavelength. From Section 5.1, it was stated that the input parameter of the linear line of best fit must have units of Pixel Number. This is because to determine the wavelengths of other spectral sources, we would like to take relative Pixel Numbers and easily compute the respectable wavelength for that Pixel Number. The details of this method will be discussed in Section 5.3. The output parameter is of course wavelength. Here, both experimental (blue line) and predicted theoretical (red dots) have been overlaid in order to display the accuracy of the linear best fit line.

To determine the goodness of the line of best fit, an R^2 value was calculated for each linear regression. An R^2 value is a statistical tool for determining if an experimental regression matches well with some predicted result. When an R^2 value resides close to unity, it is said that the experimental data matches well with the predicted results. This value was calculated using Eq. 6. The Python programming algorithm for this

calculation is shown in Appendix D. The R^2 values are summarized in Table. 2. This result means the linear regression for both spectrometers agrees well with predicted results since both R^2 values attain to unity. It is also noted that the R^2 value for the USB 2000 spectrometer is closer to unity than the R^2 value for the On-Campus Telescope spectrometer. From Figure 5, we can see this by noticing that the linear regression for the USB 2000 encompasses more points then the linear regression for the On-Campus Telescope spectrometer, although both are still considerably accurate.

One final thing to note about the linear regression for both spectrometers are the uncertainties in both m and c as mentioned briefly in Section 4.2. Upon utilizing Eq. 3 and Eq. 4, the uncertainties are found to be appropriate and in range of the experiment performed, as mentioned in Section 2. All results for R^2 values and uncertainties σ_m and σ_c are summarized in Table 2.

Table 2: R^2, σ_m, and σ_c values for the USB 2000 and On-Campus Telescope spectrometers.

Beginning of Data		
Quantity	USB 2000 Spectrometer	On-Campus Telescope Spectrometer
R^2	0.9999999948210662	0.9999999238110674
σ_m	0.0009779383861198733	0.007526789903642793
σ_c	1.512846333364572	1.2880612582110627
End of Data		

5.3 Wavelength Solution

The wavelength solution for both spectrometers had already been determined from Eq. 2, however, it was necessary to compare the experimental and predicted results from Section 5.2. This ensures that the wavelength solution (Eq. 2) is accurate and precise. The reason that a linear model was chosen was due to the fact that higher order models appeared to have vanishing coefficients of second order or higher (coefficients approach 0). This was determined by Lab Group G Member Lucas Louwerse. Therefore, any models with polynomial fits of degree 2 or higher have been considered negligible and a polynomial of degree 1 was taken as the best fit. From Eq. 2, the wavelength of any spectra for the corresponding CCD will pursue the following algorithm:

1. Take any Pixel Number for any spectral data corresponding to the considered spectrometer.
2. Take the Pixel Number and perform the following algorithm where x_i is a Pixel Number: $mx_i + c$.

This algorithm was performed for all other remaining spectra which had not been considered in the linear least square fitting. For directness, one table will be shown, this is because it is fairly intuitive on how to perform the wavelength solution for any general spectral data corresponding to the considered spectrometers.

Table 3: Wavelength Solution Applied to Star Navi

Beginning of Data	
Pixel Number	Wavelength ($mx_i + c$) (nm)
134	610.9698234258619
532	696.2318545550573
End of Data	

6 Conclusion

In this lab, the wavelength solution for the USB 2000 and the On-Campus Telescope were determined. From collected spectral data, computations to determine quantities known as the peaks, valleys, and centroids were approached through the writing of manual Python programming codes. Determining these quantities was required in order to perform linear least square fitting for determining the wavelength solution. Overall, it was determined that the best fit for the collected spectral data was a linear line of best fit. When the best fit was determined and confirmed with predicted results, a wavelength solution could be defined for the respectable spectrometer in question. The errors associated with any measurement and/or computation were calculated, thoroughly discussed, and accounted for in the context of this experiment.

7 Appendix

7.1 Appendix A Data Importation Code

Below is a sample of code which is used to import spectral data from a CCD into Python

```
#Data Impotation
import pandas as pd #Import Pandas as a Tool to Read an Excel File and
                    #Convert Data into a List
import astropy as ast #Import Astropy as a Tool to Read a Fits File
from astropy.io import fits #Import Astropy.io to Extract Data from a Fits
                           #File


#USB 2000 Data Importing
Data=pd.read_excel
    (r'C:\Users\param\Desktop\Astro Labs\Lab 2\Excel CCD Data.xlsx')
                        #Read the Excel File which Contains All Data

IncandescentBulbList=Data['IncandescentBulb']
                        #Import Incandescent Bulb Data into a List

NeonList=Data['Neon'] #Import Neon Data into a List

HydrogenList=Data['Hydrogen'] #Import Hydrogen Data into a List

FluorescentBulbList=Data['FluorescentBulb'] #Import Fluorescent Bulb
                                            #Data into a List

SunList=Data['Sun'] #Import Sun Data into a List



#Telescope Data Importing
TelescopeNeonWhole=ast.io.fits.getdata
(r'C:\Users\param\Desktop\Astro Labs\Lab 2\Telescope Data\NeonCalibration.fit')
                        #Read Fits File Which Contains Neon Data
TelescopeNeon=TelescopeNeonWhole[250,:] #Take One Row Of Data into a List

TelescopeDarkWhole=ast.io.fits.getdata
(r'C:\Users\param\Desktop\Astro Labs\Lab 2\Telescope Data\Dark1.fit')
                        #Read Fits File Which Contains Dark Data
TelescopeDark=TelescopeDarkWhole[250,:] #Take One Row Of Data into a List

TelescopeEnifWhole=ast.io.fits.getdata
(r'C:\Users\param\Desktop\Astro Labs\Lab 2\Telescope Data\EnifSpec.fit')
                        #Read Fit File Which Contains Enif Star Data
TelescopeEnif=TelescopeEnifWhole[400,:] #Take One Row Of Data into a List

TelescopeNaviWhole=ast.io.fits.getdata
(r'C:\Users\param\Desktop\Astro Labs\Lab 2\Telescope Data\NaviSpec.fit')
                        #Read Fits File Which Contains Navi Star Data
TelescopeNavi=TelescopeNaviWhole[250,:] #Take One Row Of Data into a List

TelescopeScheatWhole=ast.io.fits.getdata
```

```
(r'C:\Users\param\Desktop\Astro Labs\Lab 2\Telescope Data\ScheatSpec.fit')
                        #Read Fits File Which Contains Scheat Star Data
TelescopeScheat=TelescopeScheatWhole[370,:] #Take One Row Of Data into a List

TelescopeVegaWhole=ast.io.fits.getdata
(r'C:\Users\param\Desktop\Astro Labs\Lab 2\Telescope Data\Vega.fit')
                        #Read Fits File Which Contains Scheat Star Data
TelescopeVega=TelescopeVegaWhole[220,:] #Take One Row Of Data into a List
```

7.2 Appendix B Peaks, Valleys, and Centroids Code

Below is a sample of code which is used to determine the peaks, valleys, and centroids for a given spectral data set. One example is shown for the USB 2000 Neon data set. This code was applied in the same way to any other calculations of this kind.

```
#Peaks, Valleys, and Centroids
import matplotlib.pyplot as plt #Import MatplotLib for Generating Plots


def LeftShift2(x):
    LShift2=x-7
    return(LShift2) #Define a function to shift a point 7 units to the left

def RightShift2(y):
    RShift2=y+7
    return(RShift2) #Define a function to shoft a point 7 units to the right

def plotLine(x):
    plt.axvline(x, color='red', ls='-.') #Define a function to create a
                                         #Verticle red line at a point


#USB 2000 Neon Peaks, Valleys, and Centroids
#Peaks
peaksNeon=[] #Create an Empty List to Store Peaks
wherepeaksNeon=[] #Create an Empty List to Store Where the Peaks of the Neon
                  #Data Set Are (Pixel Number)
thresh2=7900 #Define a Threshold to Get Rid of Noise
k=1 #Create an Index
while k<len(NeonList)-1: #Create a While Loop to Determine all of
                         #the Peaks of the Neon Data Set
    if NeonList[k]>NeonList[k+1] and NeonList[k]>NeonList[k-1]
        and NeonList[k]>thresh2: #Checks If Point is Larger than Surrounding
                                 #Points and Threshold
        peaksNeon.append(NeonList[k]) #If it is then add it to list of peaks
        wherepeaksNeon.append(k) #Store the peaks location
    k=k+1 #Add One to the Index

#Valleys
valleysNeon=[] #Create an empty list to store valleys
wherevalleysNeon=[] #Create an empty list to store Pixel Number of valley
t=0 #Create an index
while t<len(peaksNeon): #Create a while loop to determine all of the
                        #valleys of the Neon data set
    valleysNeon.append(NeonList[LeftShift2(wherepeaksNeon[t])])
```

```
    valleysNeon.append(NeonList[RightShift2(wherepeaksNeon[t])])
                        #Find minimum points around the peaks and store them in
                        #the list of valleys
    wherevalleysNeon.append(LeftShift2(wherepeaksNeon[t]))
    wherevalleysNeon.append(RightShift2(wherepeaksNeon[t]))
                        #Store the valleys location
    t=t+1 #Add one to the index

#Centroids
d=0 #Create an index
CentroidsNeon=[] #Create an empty list to store Centroids
while d<len(valleysNeon): #Create a while loop to determine Centroids for the
                          #Neon data set
    CentroidsNeon.append(((valleysNeon[d]*
                           wherevalleysNeon[d])+(valleysNeon[d+1]*
                           wherevalleysNeon[d+1]))
                           /(valleysNeon[d]+valleysNeon[d+1]))
                #Apply a simple weighted average to determine a Centroid and
                #store its location
    d=d+2 #Increase index by two

#Generate a Plot
plt.subplot()
plt.title("Neon Peaks and Valleys") #Plot title
plt.xlabel("Pixel Number") #Label x-axis
plt.ylabel("Measured Signal") #Label y-axis
plt.xlim(xmin=1200,xmax=len(NeonList)) #Restrict x axis
plt.plot(NeonList) #Plot raw data
plt.plot(wherepeaksNeon,peaksNeon,'ro') #Overplot peaks
plt.plot(wherevalleysNeon,valleysNeon,'bo') #Overplot valleys
b=0
while b<len(CentroidsNeon):
    plotLine(CentroidsNeon[b]) #Overplot centroids
    b=b+1
plt.show()
```

7.3 Appendix C Method Of Linear Least Squares Code

Below is a sample of code which is used to calculate the linear least fit for a given spectral data set. One example is shown for the USB 2000 Neon data set. This code was applied in the same way to any other calculations of this kind.

```
#Linear Least Square Fitting
import matplotlib.pyplot as plt #Import MatplotLib for generating plots
import numpy as np #Import Numpy for accessing mathematical tools


NeonWave=[585.294, 588.189, 594.483, 597.553, 602.999, 607.434, 609.616,
          614.306, 614.306, 621.728, 626.310, 630.479, 633.433, 638.299,
          640.225, 650.653, 650.653, 659.895, 667.828, 671.704, 692.947,
          703.241] #Predicted wavelength values

CentroidsSquaredNeon=[] #Create a list that stores all of the centroid values
                        #squared
i=0 #Create an index
```

```
while i<len(CentroidsNeon): #Create a while loop that takes a centroid
                            #and squares it and stores this into a list
    CentroidsSquaredNeon.append(CentroidsNeon[i]**2)
    i=i+1

CentroidsxyNeon=[] #Create an empty list that stores all of the centroid
                   #values multipied by the relative predicted wavelength
i=0 #Create an index
while i<len(CentroidsNeon): #Create a while loop that takes the centroids
                            #and multiplies it with its relative predicted
                            #wavelength and store this in a list
    CentroidsxyNeon.append(CentroidsNeon[i]*NeonWave[i])
    i=i+1


ma=np.array([[sum(CentroidsSquaredNeon),sum(CentroidsNeon)],
             [sum(CentroidsNeon),len(CentroidsNeon)]])
mc=np.array([[sum(CentroidsxyNeon)],[sum(NeonWave)]])
maI=np.linalg.inv(ma)
md=np.dot(maI,mc) #Create two arrays with required elecments and
                  #perform matrix manipulations

slope=md[0,0] #Take a specific entry for the slope
intercept=md[1,0] #Take a specific entry for the intercept


#Plot Linear Regression
peaksNeon.sort() #Sort peaks from least to greatest

plt.subplot()
plt.title("Neon Line Of Best Fit For Centroids") #Plot title
plt.xlabel("Pixel Number") #Label x-axis
plt.ylabel("Wavelength(nm)") #Label y-axis
plt.plot(CentroidsNeon,NeonWave,'ro') #Plot predicted relation as red dots
yvaluesLine=[] #Create an empty list to store outputs of linear best fit
i=0
while i<len(CentroidsNeon):
    yvaluesLine.append((slope*CentroidsNeon[i])+intercept) #Perform linear
                                                           #best fit
    i=i+1

plt.plot(CentroidsNeon,yvaluesLine) #Overlay linear best fit plot
plt.text(1400,680,'m = {:.3f}\nc = {:.3f}'.format(slope,intercept)) #Label
                                                                   #slope
                                                                   #and
                                                                   #intercept
plt.show() #Show plot
```

7.4 Appendix D Error Propagation and R^2 Value Code

Below is a sample of code which is used to calculate the error propagation of m (slope) and c (intercept) and also to calculate the R^2 value. One example is shown for the USB 2000 Neon data set. This code was applied in the same way to any other calculations of this kind.

```
#R Squared Value Calculation
averageY=sum(yvaluesLine) #Take the average of experimental wavelengths

w=0 #Create an index
ExpPred=[] #Create an empty list to store exp. wavelength multiplied by
           #Predicted wacelength
while w<len(yvaluesLine): #Create a while loop to compute exp. wavelength
                          #subtracted by Predicted wavelength and store it
    ExpPred.append((yvaluesLine[w]-NeonWave[w])**2)
    w=w+1

w=0 #Set index to 0
PredAv=[] #Create an empty list to store predicted wavelength subtracted by
          #average experimental wavelength
while w<len(yvaluesLine): #Create a while loop to compute predicted wavelength
                          #subtracted by average experimental wavelength and
                          #store it
    PredAv.append((NeonWave[w]-averageY)**2)
    w=w+1

rSquared=1-(sum(ExpPred)/sum(PredAv)) #Calculate R-Squared value


#Error Propogation
ErrorList=[] #Create an empty list to store standard deviations
g=0
while g<len(NeonWave): #Create a while loop to compute and store standard
                       #deviations
    ErrorList.append((NeonWave[g]-yvaluesLine[g])**2)
    g=g+1

sigmaSquared=(1/(len(peaksNeon)-2))*sum(ErrorList) #Compute the standard
                                                   #deviation to use in the
                                                   #other error propogations

#Standard Deviation For Slope
sigmaMSquared=((len(CentroidsNeon))*sigmaSquared)
              /((len(CentroidsNeon))*
                sum(CentroidsSquaredNeon)-(sum(CentroidsNeon))**2)
sigmaM=(sigmaMSquared)**(1/2) #Compute the error for m using the given formula

#Standard Deviation For Intercept
sigmaCSquared=(sigmaSquared*(sum(CentroidsSquaredNeon)))
              /((len(CentroidsNeon))
              *sum(CentroidsSquaredNeon)-(sum(CentroidsNeon))**2)
sigmaC=(sigmaCSquared)**(1/2) #Compute the error for c using the given formula
```

7.5 Appendix E Predicted Wavelength Calculation Code

Below is a sample of code which is used to predict wavelengths from the linear wavelength solution. One example is shown for the On-Campus Telescope observed star Navi. This code was applied in the same way to any other calculations of this kind.

```
#Predicted Wavelength
PredictionN=[] #Create a list to store experimentally predicted wavelengths
w=0 #Create an index
while w<len(wherepeaksN): #Create a while loop which uses the wavelength
                          #to determine wavelengths from centroids
    PredictionN.append((slopeT*Centroids[w]+interceptT))
    w=w+1
```

References

[1] AST325/326 Course Website. http://www.astro.utoronto.ca/ astrolab/.

[2] Dae-Sik Moon. Introduction to Charge Coupled Devices (CCDs). http://www.astro.utoronto.ca/ astrolab.