Lab 2: Introduction to Spectroscopy

Ritik Kothari
1006275843
ritik.kothari@mail.utoronto.ca
Professor Dae-Sik Moon
1 November 2021

Abstract

In this report, wavelength calibration was performed to determine various properties of a 1-dimensional Neon spectrum and a 2-dimensional dispersed image using OH telluric sky lines. Wavelengths and OH sky lines are identified from the provided datasets in order to determine their peaks. These peaks are then plotted against wavelengths and intensities to search for trends. Linear and nonlinear regression methods are also used to obtain wavelength solutions for each set of data. Several methods for error calculation are applied to measurements in an effort to determine the effect of error propagation on results. Applications of the wavelength solutions to their respective plots lead to the calculations of a blackbody's temperature and the velocity of a gas emitting an [Fe II] emission.

Table of Contents

1 Introduction

Spectroscopy is an important tool used by all physical sciences. Astrophysicists in particular use spectroscopy for characterizing the physical nature of celestial objects and the universe. Astronomical spectroscopy has been used to measure the chemical composition and physical conditions (temperature, pressure, etc.) in planets, stars, and galaxies, as well as their velocities. One of the key aspects of spectroscopy is wavelength calibration.

In this lab, we were introduced to spectroscopy and were taught to understand how it is done. We then conducted two types of wavelength calibrations, one with Neon lamp for 1-dimensional spectra and the other with OH telluric sky lines for 2-dimensional spectra. The solutions to these calibrations were then applied to determine the temperature of a blackbody spectrum and the velocity of ionized iron gas from a supernova explosion respectively.

Division of labour was handled evenly between the author (Ritik), along with lab partners Ian Niebres and Bo Han. All methods and code writing was done collaboratively, with special care being taken by each group member to ensure individuality in each person's submitted work.

2 Procedure

2.1 1-Dimensional Neon Spectrum and Blackbody Temperature

2.1.1 Observations and Data

Observations for the 1-Dimensional Neon Spectrum experiment were taken by detecting dispersed photons inside a spectrograph using a 1024 pixel 1-dimensional detector. Photons from the light source (e.g., Neon lamp, blackbody source) were transferred to the entrance of a USB4000 spectrograph by optical fiber. Inside the spectrograph, the entered photons are dispersed by grating before being recorded. The detector has 1024 pixels. The spectrograph was configured to be able to detect photons within the 500-800 nm range.

All data was provided in the "Lab 2 Handout" in the form of the "Group_I_BB.dat" and "Ne_calib.dat" files. The "Group_I_BB.dat" file contains the spectrum of a blackbody source, while the "Ne_calib.dat" file contains a spectrum of a Neon lamp obtained with the same spectrograph. Plotting, we have (see Appendix A for imports/read-in and Appendix B for code),

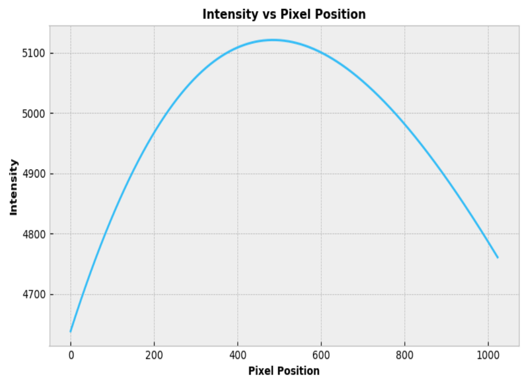

Figure 1: Spectrum of Blackbody Source.

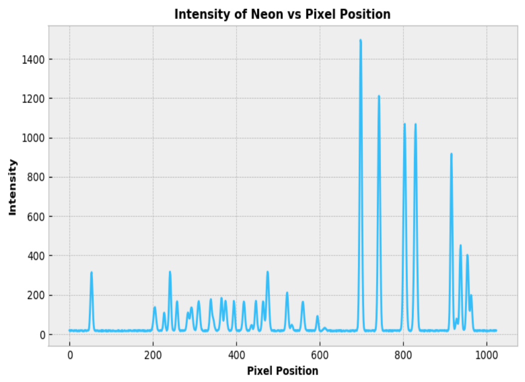

Figure 2: Spectrum of Neon Lamp.

We began by reading off the values on the x-axis of Figure 5 (while also referring to the tabular values in Figure 4) as provided in the Lab Handout to determine the wavelengths of the Neon lines. These values were then graphed against the centroids of the lines in Figure 3 of the Lab Handout (see 2.1.2).

Wavelengths of Neon Lines (nm)							
511.367	511.650	540.056	576.441	582.015	585.249	588.189	594.483
597.553	602.000	607.433	609.616	612.884	614.306	616.359	621.728
626.649	630.479	633.442	638.299	640.225	650.653	653.288	659.895
667.828	671.704	692.947	703.241	717.394	724.512	743.890	747.244
748.887	753.577	754.404	837.761	849.536	878.375	1117.752	1152.275

Table 1: Figure 4 from the Lab Handout containing wavelengths of Neon lines.

2.1.2 Data Reduction and Methods

Data collected for the 1-Dimensional Neon Spectrum experiment was analyzed using various Python packages, such as `numpy`, `scipy`, and `matplotlib`. Several additional functions were self-defined for the purposes of manipulating datasets in order to return values that better fit the tasks at hand. The methods outlined in the Lab Handout were followed closely during the data analysis process.

As the primary goal of this experiment was to determine the temperature of a blackbody source by analyzing its spectrum obtained with the spectrograph, the majority of the data reduction and analysis being done involved determining the positions of pixels and obtaining linear least square fittings. A particular example of determining the pixel positions involved the use of a self-defined function `find_peaks`, which allowed for the calculation of the peaks in Figure 2 and its subsequent re-plotting (see Appendix C for code and 2.1.4 for details).

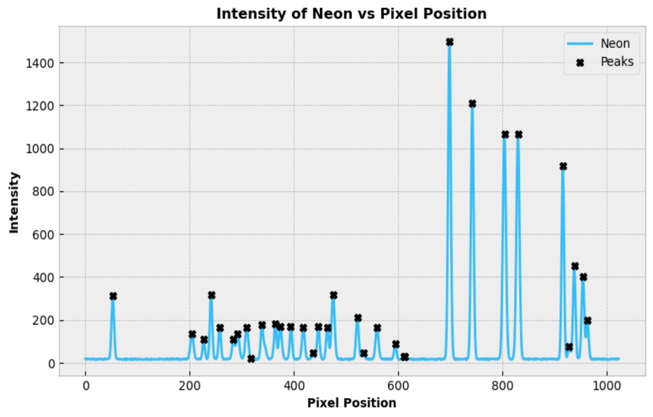

Figure 3: Re-plot of Figure 2 with Pixel Positions

The function `find_peaks` returned the values of the peaks and their respective sequential positions by first taking in `data` and `threshold` arguments. Values in the `data` array were compared against the `threshold` to determine whether or not it is a peak. This was done by comparing a datapoint to the point before and after it. If that point was larger than both points before and after it, then it was a peak. Values of the peak and its index were then stored in their respective return lists.

The uncertainties for this experiment stemmed from the calculation of the slope and intercept values for our least square fitting line. Using various Python functions, we determined the uncertainty in the slope to be about $\pm 4.567 \times 10^{-6}$, while the uncertainty in the intercept was about ± 1.508 (See Appendix E for code).

2.1.3 Data Analysis and Modelling

The final goal of this experiment was to determine the temperature of a blackbody source by analyzing its spectrum obtained with the spectrograph. In order to do this, we obtained a linear square fitting by following the procedure outlined in the Lab Handout's Appendix and manipulating it to fit our data. We first plotted the pixel positions of the peaks in Figure 3 against the wavelengths we determined from Table 1 and Figure 5 from the Lab Handout. Then, again referring to the Appendix from the Lab Handout, we constructed matrices using the provided formula,

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

where m, c, x_i, y_i, and N are the slope, intercept, independent variable, dependent variable, and the number of datapoints respectively. Using the results of Equation 1 allowed us to over-plot a least square fitting line onto our wavelengths data (see Appendix D for code).

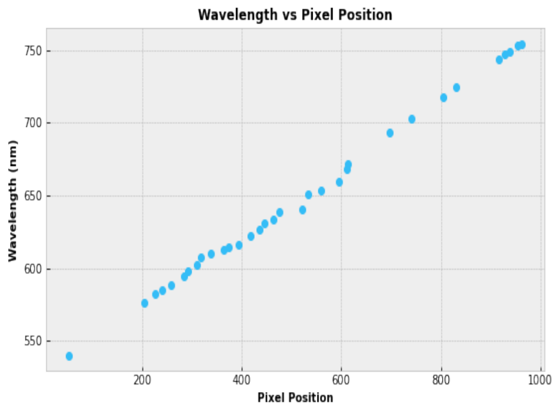

Figure 4: Pixel Positions vs. Wavelengths.

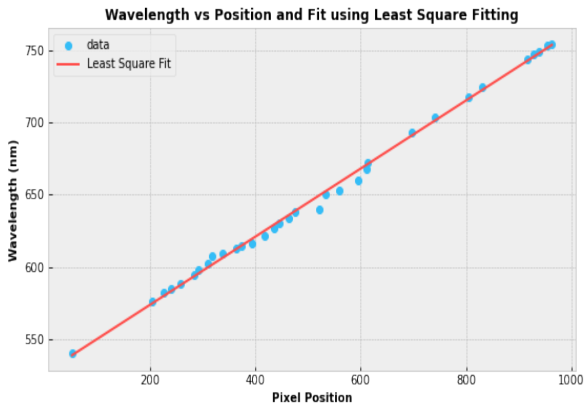

Figure 5: Least Square Fitting Over-plot.

It is clear from Figure 5 that the linear fitting is quite accurate, as it fits the data near perfectly. This is also evident from our calculation of the slope and intercept errors. The line was calculated through the use of a self-defined function `line`, which took in arguments of `pos`, `mfit`, and `cfit` as defined in the Lab Handout's Appendix. The parameter `pos` was calculated using `find_peaks` and so was equivalent to the returned array of the peaks' indices. Parameters `mfit` and `cfit` were subsequently calculated to be the slope and intercept of the least square fitting line, with the `line` function returning it in the form

$$y = mx + b = (\texttt{mfit})(\texttt{pos}) + (\texttt{cfit}) \tag{2}$$

with `mfit` ≈ 0.2357 and `cfit` ≈ 526.5057.

The final part of the experiment saw us apply the wavelength solution (which is the Least Square Fitting Line) to the Blackbody Spectrum we acquired from the "Group_I_BB.dat" file. By doing so, we obtained a plot which gives us the wavelengths of the Blackbody Spectrum (see Appendix E for code).

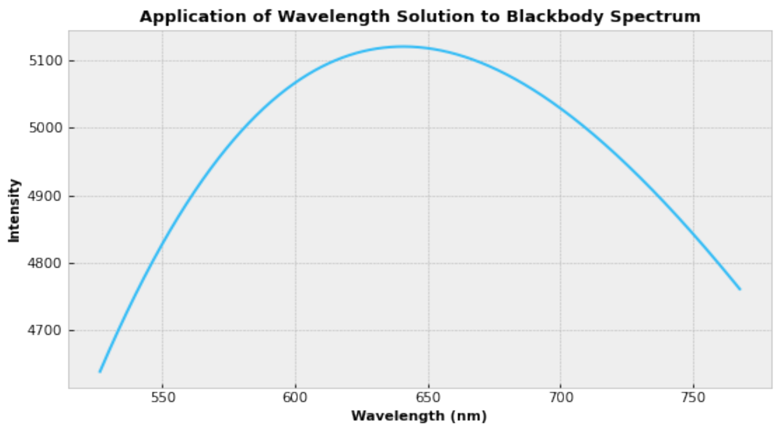

Figure 6: Applying the Wavelength Solution to the Blackbody Spectrum data.

Considering now Wien's Displacement Law [1], given by

$$\lambda_{peak} T = 2.898 \times 10^{-3}\,\mathrm{m} \cdot \mathrm{K} \tag{3}$$

where $\lambda_{peak} \approx 6.408 \times 10^{-7}\,\mathrm{m}$, we can determine the temperature of the blackbody to be

$$\begin{aligned} T &= \frac{2.898 \times 10^{-3}\,\mathrm{m} \cdot \mathrm{K}}{\lambda_{peak}} \\ &= \frac{2.898 \times 10^{-3}\,\mathrm{m} \cdot \mathrm{K}}{6.408 \times 10^{-7}\,\mathrm{m}} \\ &\approx 4521.961\,\mathrm{K} \end{aligned}$$

Thus, through the procedure of obtaining a least square fitting line using various matrices and then applying it to the blackbody spectrum, we determined the temperature of the blackbody to be $T \approx 4521.961\,\mathrm{K}$.

2.1.4 Discussion

Over the course of this experiment, we identified Neon wavelengths and used Python to determine the pixel positions of the various lines. While there are multiple ways to determine the pixel positions (centroids), our group decided on comparing values from the array of wavelengths to a `threshold` using a self-defined function `find_peaks`. The result of this application is seen in Figure 3, with which we were able to collect the peak positions and form an array for later use.

For the least square fitting line (Figure 5), we heavily referenced the Lab Handout's Appendix, with our calculations coming from the use of Equation 1 along with built in Python functions. The purpose of determining the least square fitting was to acquire a wavelength solution for the blackbody spectrum, which in turn led to the calculation of the blackbody's temperature through applying Wien's Displacement Law (Equation 3).

There were several limitations that we encountered throughout this experiment, with the most notable one being that the data was given to us. As we were unable to directly make observations and instead had to rely on the data being provided, we assumed the various findings with a high degree of certainty. A consequence of this would be uncertainties that carried through our experiment, thus reducing the final certainty of our results.

For values that were self-obtained, such as the slope and intercept of the least square fitting, we calculated uncertainties using the `numpy.sqrt` and `numpy.sum` functions. The calculated pixel position and wavelength arrays were used along with the determined values of slope and intercept to find the error propagation for these measurements. Small values for the uncertainties (see 2.1.2) imply that our calculations were close to the actual values of the slope and intercept.

Concerning future replications of this experiment, an improvement in the procedure would be for researchers to personally collect data, allowing them to be unequivocal about the uncertainties in their measurements.

2.2 2-Dimensional Dispersed Image and Velocity of Ionized Iron Gas

2.2.1 Observations and Data

Observations for the 2-Dimensional Dispersed Image experiment were taken by accessing the provided "Near-infrared.fits" file (see Appendix F for code). Using OH sky telluric lines, the file showed the dispersed image of ionized iron gas from a supernova explosion in the spatial and spectral directions. In particular, the file showed a clear [Fe II] emission measuring 1.644 μm almost overlapping with an OH line.

From the Lab Handout, Figure 7 illustrated a spectrum of the five most prominent OH lines from Figure 6, the wavelengths of which were identified using the relative intensities from Figure 8.

Relative Intensities of Known OH Telluric Sky Lines (1.610 μm - 1.690 μm)			
16128.608	16194.615	16235.376	16317.161
16530.630	16360.385	16388.492	16442.155
16475.648	16502.365	16692.380	

Table 2: Figure 8 from the Lab Handout containing wavelengths of Neon lines.

2.2.2 Data Reduction and Methods

Data collected for the 2-dimensional dispersed image experiment was again analyzed using various Python packages, such as `numpy`, `scipy`, and `matplotlib`. Several additional functions were self-defined for the purposes of manipulating datasets in order to return values that better fit the tasks at hand. The methods outlined in the Lab Handout were followed closely during the data analysis process.

As the primary goal of this experiment was to determine the velocity of the iron gas using the observed wavelength of the [Fe II] 1.644 μm line, the majority of the data reduction and analysis being done involved identifying OH lines, determining their central positions, and conducting polynomial fitting of the central positions to the wavelengths.

Similar to the methods in 2.1.2, we first read into the provided `fits` file and created a sample array of the initial data. Using the `numpy.transpose` function, we then plotted the transpose of the sample array to obtain the spectrum of the OH telluric sky lines (see Figure 7). This was then over-plotted using a self-defined function `find_more_peaks` which determined the peaks of the intensities (see Appendix G for code).

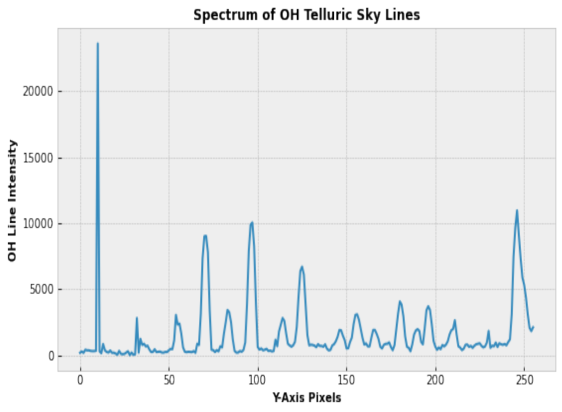

Figure 7: Spectrum of OH sky lines.

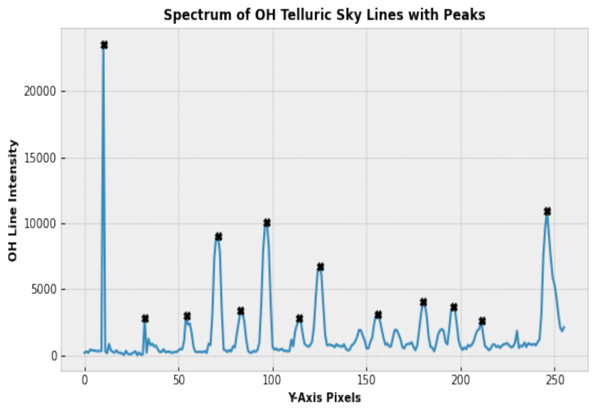

Figure 8: Over-plot of Figure 7 with Peaks.

The function `find_more_peaks` was defined very similarly to `find_peaks`. Following the same process as outlined by `find_peaks`, `find_more_peaks` stores values of the peak(s) and its index in their respective lists, which are then returned.

Following the plotting of Figure 8, we defined a function `cubic` which took in 5 parameters and returned a third-degree polynomial based on them. This was done to conduct polynomial fitting of the central positions (peaks) to the wavelengths we identified in Table 2 (see Appendix G for code).

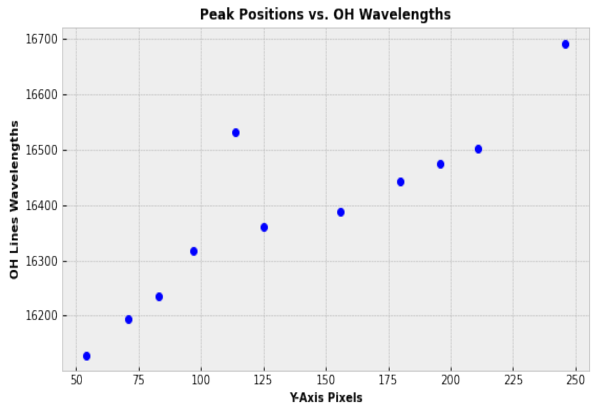

Figure 9: Peaks vs. Wavelengths, Table 2.

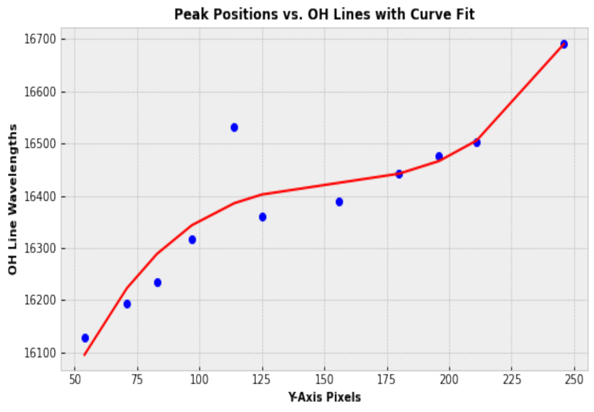

Figure 10: Over-plot of Figure 9 with Curve Fit.

The uncertainties for this experiment stemmed from the calculation of the `c2` parameter used to determine the coefficients of the polynomial returned by `cubic`. Using the `numpy.sqrt` and `numpy.diag` functions, we found an array of uncertainties in each of the four coefficient parameters taken by `cubic`. These uncertainties were then added up to get a cumulative uncertainty of the wavelength solution (curve fit line), which was approximately ± 0.302 km/s.

2.2.3 Data Analysis and Modelling

The goal of this experiment was to find the velocity of the gas emitting the [Fe II] emission from Figure 6 of the Lab Handout. To do so, we first determined the central position of the [Fe II] emission in the y-axis, which was about 181. Applying then the wavelength solution from Figure 10, we can estimate the wavelength by using the `cubic` function with the central position being a parameter (see Appendix H for code).

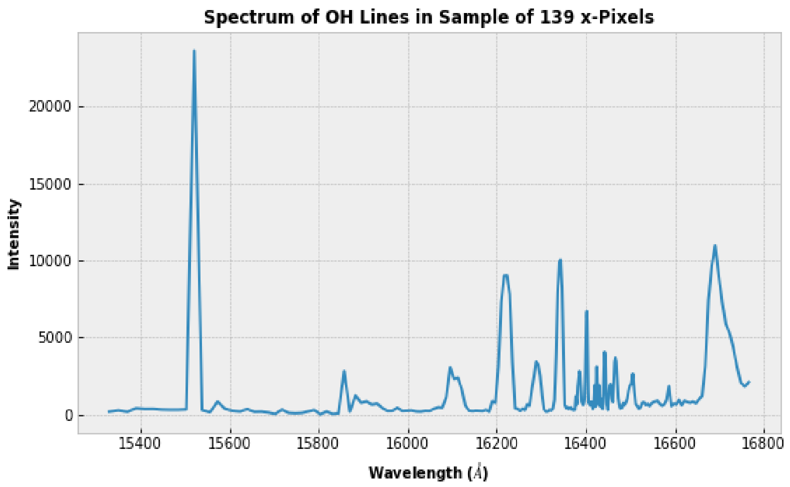

Figure 11: Applied Wavelength Solution to Find Wavelength Estimate.

The returned value from `cubic` gives an estimated wavelength for the [Fe II] emission of about 16443.269 Å, or 1.644 μm. Comparing with our intrinsic wavelength of 1.6439981 μm, our estimate is within 0.002 μm.

Now, we can calculate the velocity of the gas by considering the following formula (from Lecture 3 Slides):

$$v = \frac{\Delta\lambda}{\lambda_{intrin}} c \tag{4}$$

where $\Delta\lambda = \lambda_{est} - \lambda_{intrin}$, $\lambda_{intrin} = 1.6439981\,\mu\text{m}$, and c is the speed of light.

Calculating, we have

$$\begin{aligned} v &= \frac{\Delta\lambda}{\lambda_{intrin}} c \\ &= \frac{\lambda_{est} - \lambda_{intrin}}{\lambda_{intrin}} c \\ &= \frac{1.644\,\mu\text{m} - 1.6439981\,\mu\text{m}}{1.6439981\,\mu\text{m}} \left(3.0 \times 10^5\,\text{km/s}\right) \\ &\approx 55.091\,\text{km/s} \end{aligned}$$

Thus, through the procedure of conducting a polynomial fitting of the central positions to the wavelengths and applying that solution to estimate the wavelength of the [Fe II] emission, we determine the velocity of the emitting gas to be $v \approx 55.091\,\mathrm{km/s}$.

2.2.4 Discussion

Over the course of this experiment, we identified OH lines and used Python to determine their central positions. This was done in a similar fashion to the method outlined in 2.1.4, which involved comparing values from the array of identified OH lines to a `cutoff` using a self-defined function `find_more_peaks`. The result of this application is seen in Figure 8, with which we were able to collect the peak positions and form an array for later use.

For the polynomial fitting (Figure 10), we used the `curve_fit` function from the Python library `scipy.optimize`. The purpose of determining the polynomial fitting was to again acquire a wavelength solution for the wavelength of the [Fe II] emission, which in turn led to the calculation of the velocity of the emitting gas using Equation 4.

Limitations encountered in this experiment were very similar to those mentioned in 2.1.4. Being provided with data instead of being able to personally collect it meant having to assume observations with near absolute certainty, which could cause uncertainties to carry through.

For values that were self-obtained, such as the coefficients of the cubic polynomial fitting, we calculated uncertainties using the `numpy.sqrt` and `numpy.diag` functions. The `numpy.diag` function converted the covariance matrix `c2` into a diagonal array, of which the square root was taken to determine the errors in each of the four coefficient variables. Adding these values gave us a determining uncertainty that related to the wavelength solution.

Future replications of this experiment would benefit from the improvements mentioned in 2.1.4, such as collecting data first-hand so that the observations are made in confidence.

3 Conclusion

In this lab, we were introduced to spectroscopy through two experiments involving the wavelengths of a Neon spectrum and OH telluric sky lines. Through provided data, we determined wavelength solutions for each experiment and applied it to peaks in their respective spectra (Figure 3 and Figure 8). Uncertainties were calculated concerning the various parameters used in each solution, such as the slope/intercept for the Neon spectrum (2.1.2) and the polynomial coefficients for the OH sky lines (2.2.2). Reasonable uncertainty calculations further implied that our solutions were accurate. We concluded by calculating the temperature of a blackbody (2.1.3) and the velocity of an emitting gas (2.2.3).

4 Appendix

4.1 Appendix A

The following contains code for the imports and plot aesthetics used throughout.

```python
#imports
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import scipy.constants as sc
from scipy.optimize import curve_fit as cf
from astropy.io import fits

#Changing the look of plots
#Changing plotting aesthetics
plt.style.use('bmh')
mpl.rcParams['figure.figsize'] = (9, 5)
mpl.rcParams['axes.titleweight'] = 'bold'
mpl.rcParams['axes.titlesize'] = 12
mpl.rcParams['axes.labelweight'] = 'bold'
mpl.rcParams['axes.labelsize'] = 10

red = '#ff4d4a'
green = '#2ffa7d'
blue = '#34bdf7'
```

Below is also the code for the first part of the 1D Neon spectrum experiment, which consisted of reading in the "Group_I_BB.dat" and "Ne_calib.dat" files and identifying the wavelengths of Neon lines.

```python
#Importing data
Bb = np.loadtxt('Group_I_BB.dat') #Blackbody data
neon = np.loadtxt('Ne_calib.dat') #Neon calibration data

#Wavelengths of bright neon lines (from Lab Handout)
wavelengths = np.array([540.056, 576.441, 582.015, 585.249, 588.189,
                         594.483, 597.553, 602.000, 607.433, 609.616,
                         612.884, 614.306, 616.359, 621.728, 626.649,
                         630.479, 633.442, 638.299, 640.225, 650.653,
                         653.288, 659.895, 667.828, 671.704, 692.947,
                         703.241, 717.394, 724.512, 743.890, 747.244,
                         748.887, 753.577, 754.404])
```

4.2 Appendix B

The code in this section was written to illustrate the models in each of the "Group_I_BB.dat" and "Ne_calib.dat" files.

```
#Plotting the blackbody spectrum
plt.figure(1)
plt.plot(BB, color = blue)

plt.title('Intensity vs Pixel Position')
plt.xlabel('Pixel Position')
plt.ylabel('Intensity')

#Plotting the Neon spectrum
plt.figure(2)
plt.plot(neon, color = blue)

plt.title('Intensity of Neon vs Pixel Position')
plt.xlabel('Pixel Position')
plt.ylabel('Intensity')
```

4.3 Appendix C

In this section, we identified the centroids (pixel positions) of the lines and then re-plotted.

```
#Identifying centroids

def find_peaks(data, threshold):
    """
    Arguments are an array and a threshold, with which values of the array
    are compared against the threshold to determine whether or not it is a
    peak.
    This is done by comparing a datapoint to the point before and after it
    .
    If that point is larger than both points before and after it,
    then it is a peak. Values of the peak and its index are then stored
    in their respective lists.
    """
    peaks = []
    indices = []
    for i in range(1, len(data)-1): #Start at 2nd data point, end at 2nd
                                    #last.
        if data[i] >= threshold:
            if data[i] > data[i-1] and data[i] > data[i+1]:
                #proceeds iff the current point is greater than the points
                #before and after
                peaks.append(data[i])
                indices.append(i)
    return peaks, indices
peaks, pos = find_peaks(neon, 21) #Peaks and pixel positions
pos = np.array(pos) #Converting to an array

#Plotting the neon calibration and peaks
plt.figure(2)
plt.plot(neon, color = blue, label = 'Neon')
plt.plot(pos, peaks, 'X', color = "black", label = 'Peaks') Plotting
    position of peaks

plt.title('Intensity of Neon vs Pixel Position')
plt.xlabel('Pixel Position')
plt.ylabel('Intensity')
plt.legend()
```

4.4 Appendix D

Here we obtained a least square fitting for the pixel positions following the procedure from the Lab Handout.

```python
#Obtaining least square fitting

#Function to fit
def line(x, m, b):
    return m * x + b

#Optimal parameters and covariance matrices
popt, pcov = cf(line, pos, wavelengths)

#Best fit using least squares (from Lab Handout)

data_points = 33 #Number of data points (len(wavelengths))
plt.figure(5)
plt.plot(pos, wavelengths, 'o', label = 'data', color = blue) #Data
plt.xlabel('Pixel')
plt.ylabel('Wavelength (nm)')

#Constructing matrices
ma = np.array([[np.sum(pos**2), np.sum(pos)], [np.sum(pos), data_points]])
mc = np.array([[np.sum(pos * wavelengths)], [np.sum(wavelengths)]])

#Computing gradient and intercept
mai = np.linalg.inv(ma)
md = np.dot(mai, mc)

#Overplotting best fit
mfit = md[0, 0]
cfit = md[1, 0]
plt.plot(pos, line(pos, mfit, cfit), label = 'Least Square Fit', color =
    red) #Fit

plt.title('Wavelength vs Pixel Position')
plt.xlabel('Pixel Position')
plt.ylabel('Wavelength (nm)')
plt.legend()
```

4.5 Appendix E

Using the code from the previous appendices, we now applied our wavelength solution to the blackbody spectrum. We also calculated errors for the slope and intercept.

```
#Applying the wavelength solution

xs = range(len(Bb)) #Range of pixel positions (0 - 1024)

plt.figure(7)
plt.plot(line(xs, mfit, cfit), Bb, color = blue) #Applying the wavelength
   solution
plt.title('Application of Wavelength Solution to Blackbody Spectrum')
plt.xlabel('Wavelength (nm)')
plt.ylabel('Intensity')

peak_index = np.where(Bb == max(Bb)) #Index of peak
peak_wavelength = line(peak_index[0], mfit, cfit)[0] * 1.e-9 # nanometers
print("Peak Wavelength:", peak_wavelength)

b = sc.Wien #Wien's displacement constant in meters * Kelvin
print("Displacement constant:", b)

temp = b/peak_wavelength #Blackbody temperature in Kelvin
print("Blackbody temperature:", temp)

#Error calculation for slope and intercept

uncertain = (1/len(wavelengths))* \
            np.sum((wavelengths - (mfit*pos + cfit))**2)

slope_error = (len(wavelengths)*uncertain)/(len(wavelengths)* \
               np.sum(pos**2) - np.sum(pos)**2) #error in slope (mfit)
print("Slope error:", slope_error)

intercept_error = (uncertain*np.sum(pos**2))/(len(wavelengths)* \
                   np.sum(pos**2) - np.sum(pos)**2) #error in intercept
                                                    #(cfit)
print("Intercept error:", intercept_error)
```

This appendix marks the end of the code for the 1D Neon spectrum experiment.

4.6 Appendix F

In this section, we read in the provided FITS files and create variables with values from the file.

```
#Part 2
#Loading in the data

near_inf = fits.open("Near-Infrared.fits")
new_data = near_inf[0].data
r_data = new_data[0]
```

We also created an array of additional identified OH lines from a region near the [Fe II] emission as see in Figure 6 of the Lab Handout.

```
oh_lines = np.array([16128.608, 16194.615, 16235.376, 16317.161,
                     16530.650, 16360.385, 16388.492, 16442.155,
                     16475.648, 16502.365, 16692.380])

print(len(oh_lines))
```

4.7 Appendix G

Here, we plotted the spectrum of the OH telluric sky lines using a sample of data from the FITS file. The original plot was then recreated with identified peaks using a self-defined function `find_more_peaks`.

```python
sample = np.transpose(np.array(r_data))[139]
plt.plot(sample)
plt.xlabel("Y-Axis Pixels")
plt.ylabel("OH Line Intensity")
plt.title("Spectrum of OH Telluric Sky Lines")

def find_more_peaks(data, cutoff): #Similar documentation to find_peaks
    peaks = []
    indices = []
    for num in range(1, len(data) - 1):
        if data[num] >= cutoff:
            if data[num] > data[num-1] and data[num] > data[num+1]:
                peaks.append(data[num])
                indices.append(num)
    return peaks, indices

total_peaks, pixelnum = find_more_peaks(sample, 2500)

#Convert pixelnum into an array
pixelnum = np.array(pixelnum)

#Replot the given graph
plt.figure(3)
plt.plot(sample)

#Plot the position of the peaks
plt.plot(pixelnum, total_peaks, 'X', color='black', label='Peaks')
plt.xlabel("Y-Axis Pixels")
plt.ylabel("OH Line Intensity")
plt.title("Spectrum of OH Telluric Sky Lines with Peaks")
```

We then plotted the various peaks against their wavelengths, with which we conducted a polynomial fitting using a self-defined function `cubic` and `scipy.optimize.curve_fit`. We also calculated the errors in our measurements.

```python
def cubic(x, a, b, c, d):
    return a*(x**3) + b*(x**2) + c*x + d

plt.figure(4)
plt.plot(pixelnum[2:], oh_lines, 'o', color='blue') #Plot unfit data
plt.xlabel("Y-Axis Pixels")
plt.ylabel("OH Lines Wavelengths")
plt.title("Peak Positions vs. OH Wavelengths")

#Two-Dimensional Curve Fit

c1, c2 = cf(cubic, pixelnum[2:], oh_lines) #Applying curve_fit function
error = np.sqrt(np.diag(c2))
print(error)
print("The uncertainty in the wavelength solution is:", \
      (error[0]) + (error[1]) + (error[2]) + (error[3]), "m.")

plt.figure(5)
plt.plot(pixelnum[2:], oh_lines, 'o', color='blue')
plt.title("Peak Positions vs. OH Lines with Curve Fit")
plt.xlabel("Y-Axis Pixels")
plt.ylabel("OH Line Wavelengths")
plt.plot(pixelnum[2:], cubic(pixelnum[2:], c1[0], c1[1], c1[2], c1[3]), \
         '-', color='red', label='Curve Fitting') #Plot curve_fit data
```

4.8 Appendix H

In this section, we determined the central position of the [Fe II] emission and estimated its wavelength by applying the wavelength solution. Using the given intrinsic wavelength value, we calculated the velocity of the emitting gas using a formula from Lecture 3.

```
y_s = np.array(range(len(sample))) #Range of pixel positions

plt.figure(6)
plt.title('Spectrum of OH Lines in Sample of 139 x-Pixels')
plt.xlabel('Wavelength ($\AA$)')
plt.ylabel('Intensity')

#Applying wavelength solution
plt.plot(cubic(y_s, c1[0], c1[1], c1[2], c1[3]), sample)

estimate = cubic(181, c1[0], c1[1], c1[2], c1[3]) #Estimating wavelength

intrin_wave = 16439e-10

Delta = estimate*1.e-10 - intrin_wave

print("The estimated wavelength of the [Fe II] emission is:", \
      round(estimate, 3))

print("The velocity of the gas is:", \
      abs(((Delta / intrinsic) * (3.e8)) / 1000), "km/s")
```

This appendix marks the end of the code of the 2D OH telluric sky lines experiment.

References

[1] R. Nave, “Wien’s displacement law.”