Applications_of_Spectroscopy_2663452m

November 15, 2023

GUID: 2663452m

1 Introduction

Spectroscopy is the study of absorbtion and emission of electromagnetic radiation by matter. It allows for the determinition of the chemical composition of an object by observing the wavelengths of light that are absorbed or emitted by the object. This is beacuse the wavelength of light has a corresponding energy given by $E = \frac{hc}{\lambda}$ where h is Planck's constant and c is the speed of light. The energy levels of an atom are quantised and so only certain wavelengths of light can be absorbed or emitted by an atom. And thus by determining the energy of photons they can be matched to the difference in energy levels of an atom. The emssion of photons from an atom corresponds to the transition of an electron from a higher energy level to that of a lower energy, due to the conservation of energy, the difference between these levels is emitted as a photon, for the electron to be in the higher energy level (excited state) to begin with it must be supplied with some amount of energy this is done due to the absorbtion of a photon by the atom which has the same energy as the difference between the energy levels. In this lab the Balmer series of Hydrogen is being studied which corresponds to the visible wavelengths of a Hydrogen atoms emission spectrum and will be discussed in more detail later.

2 Aims

Spectroscopy is the study of absorbtion and emission of electromagnetic radiation by matter. It allows for the determinition of the chemical composition of a The aims of this experiment are to understand how a spectrometer works and understand how this can be used by obtaining results from a Hydrogen lamp, LED's and a Birefringent filter

3 Equipment

Experimental and analytical equipment used in this experiment include:

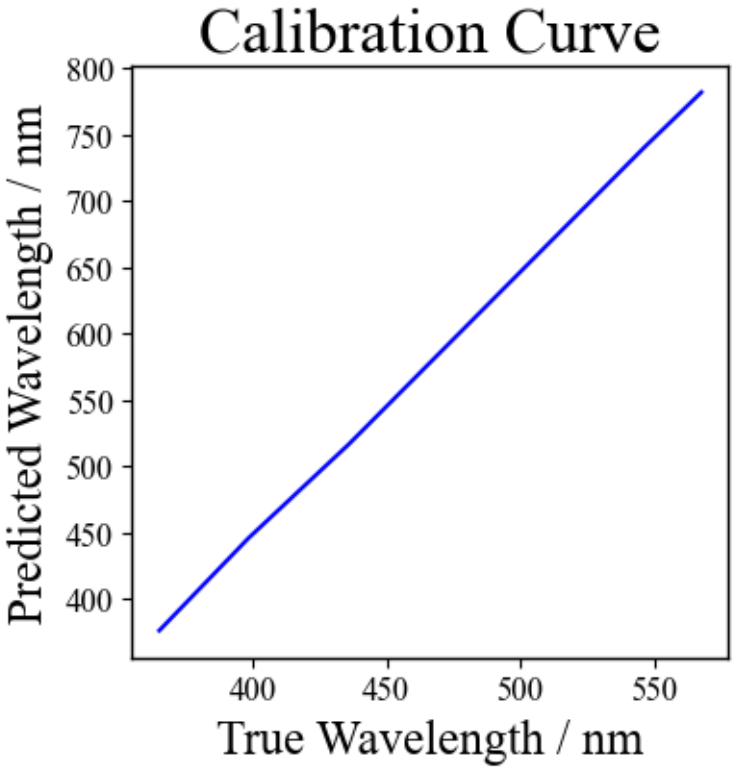
- Digital Spectrometer (Ocean Optics Red Tide USB650)
- Fiber Optic Cable
- Mercury Lamp
- Hydrogen Lamp
- LED's (Five colours)
- Birefringent Filter
- Tungsten Lamp
- Two Polarising Filters

- SpectraSuite Software
- Python 3.10.11

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.signal import find_peaks
import scipy.constants as const
import pandas as pd
import os
from scipy.optimize import curve_fit
plt.style.use('../report.mplstyle')
```

4 Calibration

[ ]:
```
data = np.loadtxt('data/cali.txt')

plt.figure(figsize=(4,4))
plt.plot(data[:, 0], data[:, 1], color ='blue')
plt.xlabel('True Wavelength / nm')
plt.ylabel('Predicted Wavelength / nm')
plt.title('Calibration Curve')
plt.show()
```



From this calibration curve we can see that our CCD has a linear systematic error

5 Investigating the Atomic Spectrum of Hydrogen

5.1 Objectives

- To measure the wavlengths of the Balmer series of Hydrogen from a discharge tube
- Determine a value for the Rydberg constant from the Balmer series
- Calculate the first ionisation energy of Hydrogen

5.2 Background

Hydrogen has a simple spectrum due to it only containing one electron. When an electron in the nth energy level of Hydrogen drops to a new lower energy level it emits a photon with energy equal to the difference in energy of the two levels. This energy is described by the Rydberg formula:

$$\frac{1}{\lambda} = R\left(\frac{1}{n_1^2} - \frac{1}{n_2^2}\right)$$

and reaaranging for ΔE gives:

$$\Delta E = \frac{hc}{\lambda} = hcR\left(\frac{1}{n_1^2} - \frac{1}{n_2^2}\right)$$

$$n_1 = 1, 2, 3, ...$$

$$n_2 = n_1 + 1, n_1 + 2, n_1 + 3, ...$$

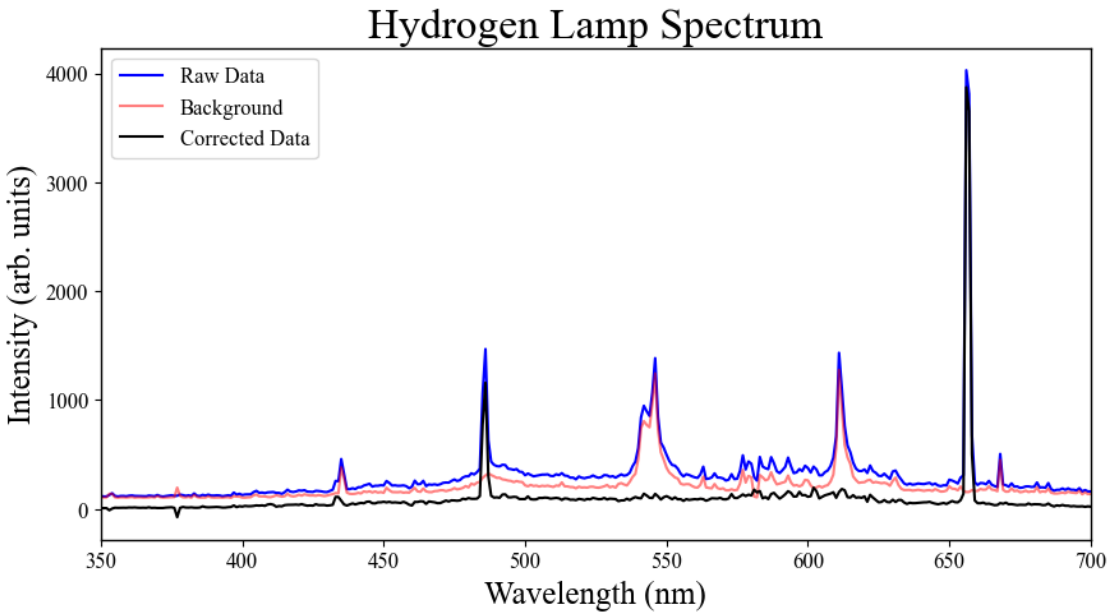
where R is the Rydberg constant and n_1 and n_2 are the energy levels of the electron. The Balmer series of Hydrogen corresponds to the visible wavelengths of light emitted by Hydrogen where $ n_1 = 2 $ and $ n_2 = 3,4,5,6 $ and emits photons in the region of 400-700nm. These wavelengths can be theorectically predicted by:

$$\lambda = \left(\frac{Bn_2^2}{n_2^2 - n_1^2}\right)$$

for the values of n_1 and n_2 given above and $B = 3654.6\times10^{-8}$ cm .

[ ]:
```
def loadtxt(filename):
    data = np.loadtxt(filename, delimiter='\t', skiprows = 17)
    return data

h_lamp = loadtxt('data/H_lamp/H_lamp0.txt')
background = loadtxt('data/H_lamp/H_lamp_background0.txt')
h_lamp_cor = h_lamp[:,1] - background[:,1]

plt.figure(figsize=(10,5))
```

```
plt.plot(h_lamp[:,0], h_lamp[:,1],color = 'blue', label = 'Raw Data')
plt.plot(background[:,0], background[:,1], color = 'red',alpha = 0.5, label =␣
↪'Background')
plt.plot(h_lamp[:,0], h_lamp_cor, color = 'black', label = 'Corrected Data')
plt.xlabel('Wavelength (nm)')
plt.xlim(350,700)
plt.ylabel('Intensity (arb. units)')
plt.title('Hydrogen Lamp Spectrum')
plt.legend()
plt.show()
```



[ ]:
```
peaks,peaks_height = find_peaks(h_lamp_cor, height = 45, distance = 22,␣
↪prominence = 0.1)
peaks = [peaks[0],peaks[1],peaks[3],peaks[9]]
peaks_height =␣
↪[peaks_height['peak_heights'][0],peaks_height['peak_heights'][1],peaks_height['peak_heights


balmer_series = (h_lamp[peaks,0], peaks_height)
wavenumber = (2*np.pi)/(balmer_series[0]*1e-9)
balm_freq = const.c/(balmer_series[0]*1e-9)
energy = const.h*balm_freq
balmer_num = np.array([6,5,4,3])

dic = {
```

```
r'Theorectical \n Wavelength (nm)':(410,434,486,656) ,
'Experimental \n Wavelength (nm)':balmer_series[0],
'Wavenumber \n (1/m) ': wavenumber,
'Frequency \n (Hz)': balm_freq,
'Energy (J)': energy,
'Balmer Number': balmer_num}

balmer = pd.DataFrame(dic)
N1 = 2
(balmer)
```

[ ]:

	Theorectical \n Wavelength (nm)	Experimental \n Wavelength (nm)	\
0	410	410.0	
1	434	433.0	
2	486	486.0	
3	656	656.0	

	Wavenumber \n (1/m)	Frequency \n (Hz)	Energy (J)	Balmer Number
0	1.532484e+07	7.312011e+14	4.844990e-19	6
1	1.451082e+07	6.923613e+14	4.587635e-19	5
2	1.292836e+07	6.168569e+14	4.087337e-19	4
3	9.578026e+06	4.570007e+14	3.028119e-19	3

[ ]:

[ ]:
```
def ryd_eq(del_e,n2,n1 = N1):
    x = 1/(n2**2) - 1/(n1**2)
    return del_e/(-const.h*const.c*x)

def ryd_eq3(wavelength,n2,n1 = N1):
    x = 1/(n2**2) - 1/(n1**2)
    return 1/((-wavelength*1e-9)*x)
for i in range(4):
    mean_ryd = (np.mean(ryd_eq(energy[i],balmer_num[i])))
    mean_ryd3 = (np.mean(ryd_eq3(balmer_series[0][i],balmer_num[i])))

def n7(n2,n1 = N1):
    x = 1/(n2**2) - 1/(n1**2)
    return 1/(-mean_ryd*x)

print(f'Fifth Balmer line of hydrogen at {n7(7):.4e}m')
```

```
Fifth Balmer line of hydrogen at 3.9684e-07m
```

FIfth balmer line appears in the spectrum but as a very small peak that could be categorized as noise if it wasnt known to be there

[ ]:
```
def ion_energy(n):
    return mean_ryd*const.c*const.h/1/(n**2)*1e-3*const.N_A

print(f'First Ionisation energy of hydrogen is {ion_energy(1):.2f} kJ/mol')
```

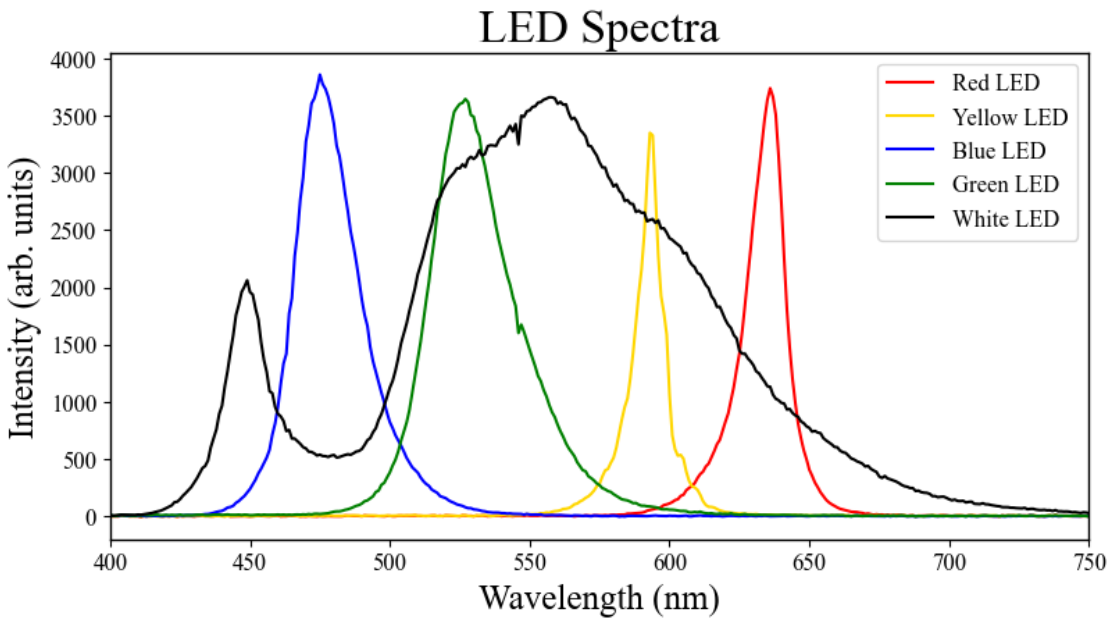
```
First Ionisation energy of hydrogen is 1312.97 kJ/mol
```

Ionisation energy in eV our value is 13.631 and accepted value in Kaye and Laby is 13.598

6 LED section

[ ]:
```
red = loadtxt('data/LED/data/red_led0.txt')
yellow = loadtxt('data/LED/data/yellow_led0.txt')
blue = loadtxt('data/LED/data/blue_led0.txt')
green = loadtxt('data/LED/data/green_led0.txt')
white = loadtxt('data/LED/data/white_led0.txt')
background = loadtxt('data/LED/background_intensity0.txt')

red[:,1] = red[:,1]- background[:,1]
yellow[:,1] = yellow[:,1] - background[:,1]
blue[:,1] = blue[:,1] - background[:,1]
green[:,1] = green[:,1] - background[:,1]
white[:,1] = white[:,1] - background[:,1]

plt.figure(figsize=(20,10))
plt.subplot(2,2,1)
plt.plot(red[:,0], red[:,1],color = 'red', label = 'Red LED')
plt.plot(yellow[:,0], yellow[:,1],color = 'gold', label = 'Yellow LED')
plt.plot(blue[:,0], blue[:,1],color = 'blue', label = 'Blue LED')
plt.plot(green[:,0], green[:,1],color = 'green', label = 'Green LED')
plt.plot(white[:,0], white[:,1],color = 'black', label = 'White LED')

plt.xlabel('Wavelength (nm)')
plt.xlim(400,750)
plt.ylabel('Intensity (arb. units)')
plt.title('LED Spectra')
plt.legend()
# plt.subplot(2,2,2)
# plt.plot(red[:,0], red[:,1],color = 'red', label = 'Red LED')
# plt.plot(yellow[:,0], yellow[:,1],color = 'gold', label = 'Yellow LED')
# plt.plot(blue[:,0], blue[:,1],color = 'blue', label = 'Blue LED')
# plt.plot(green[:,0], green[:,1],color = 'green', label = 'Green LED')
# plt.plot(white[:,0], white[:,1],color = 'black', label = 'White LED')
# plt.xlim(800,900)
# plt.show()
```

[ ]:
```
<matplotlib.legend.Legend at 0x1f581a72aa0>
```



[ ]:
```
path = 'data/LED/data/'
files = os.listdir(path)

peak = []
peaks_heights = []
for file in files:
    data = loadtxt(path+file)
    background = loadtxt('data/LED/background_intensity0.txt')
    data[:,1] = data[:,1]- background[:,1]
    peaks,peak_heights = find_peaks(data[:,1],height = 3000, distance = 50)
    peak.append(peaks[0])
    peaks_heights.append(peak_heights)
    print(file)
    print(peak_heights)

# for i in np.arange(len(peak)):
#     data = loadtxt(path+files[i])
#     intense = (data[peak[i],1])/2
#     edge = data[(data[:,1]- intense).argmin()]
#     print(edge)
```

```
blue_led0.txt
{'peak_heights': array([3861.04])}
green_led0.txt
{'peak_heights': array([3647.84])}
red_led0.txt
{'peak_heights': array([3741.81])}
```

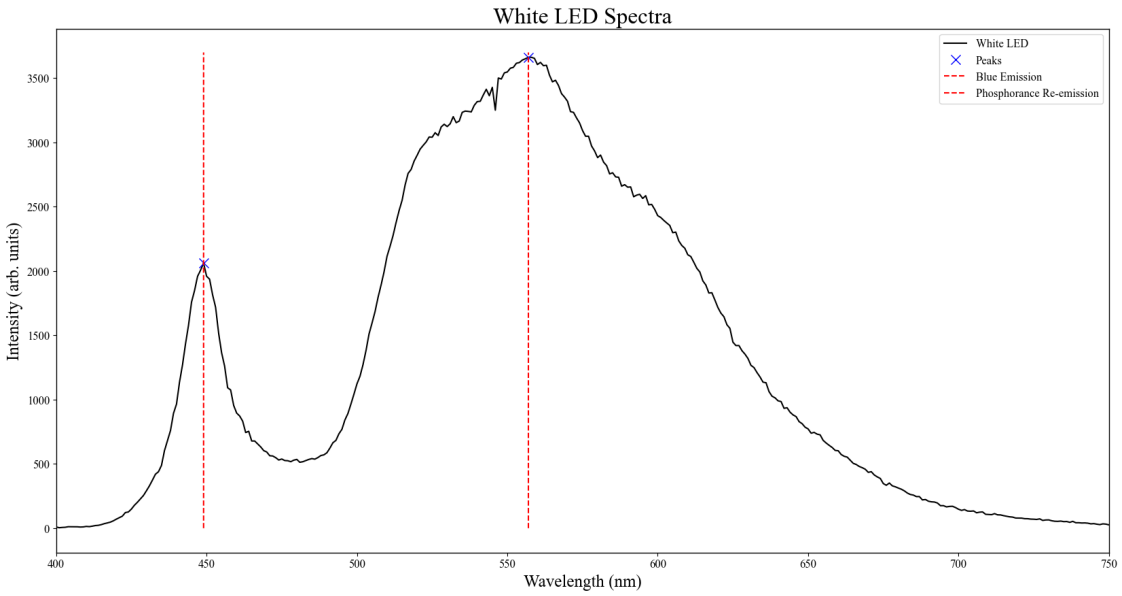
```
white_led0.txt
{'peak_heights': array([3662.79])}
yellow_led0.txt
{'peak_heights': array([3352.13])}
```

[ ]:
```python
from scipy.interpolate import splrep, sproot, splev

class MultiplePeaks(Exception): pass
class NoPeaksFound(Exception): pass

def fwhm(x, y, k=3):
    """
    Determine full-with-half-maximum of a peaked set of points, x and y.

    Assumes that there is only one peak present in the datasset.  The function
    uses a spline interpolation of order k.
    """

    half_max = max(y)/2.0
    s = splrep(x, y - half_max, k=k)
    roots = sproot(s)

    if len(roots) > 2:
        raise MultiplePeaks("The dataset appears to have multiple peaks, and "
                "thus the FWHM can't be determined.")
    elif len(roots) < 2:
        raise NoPeaksFound("No proper peaks were found in the data set; likely "
                "the dataset is flat (e.g. all zeros).")
    else:
        return abs(roots[1] - roots[0])


for file in files:
    if file == 'white_led0.txt':
        pass
    else:
        data = loadtxt(path+file)

        fwhm_val = fwhm(data[:,0],data[:,1])



def energy_peak(wavelength):
    return const.h*const.c/(wavelength*1e-9)
def drive_vol(wavelength):
    return energy(wavelength)/const.e
def energy(wavelength):
```

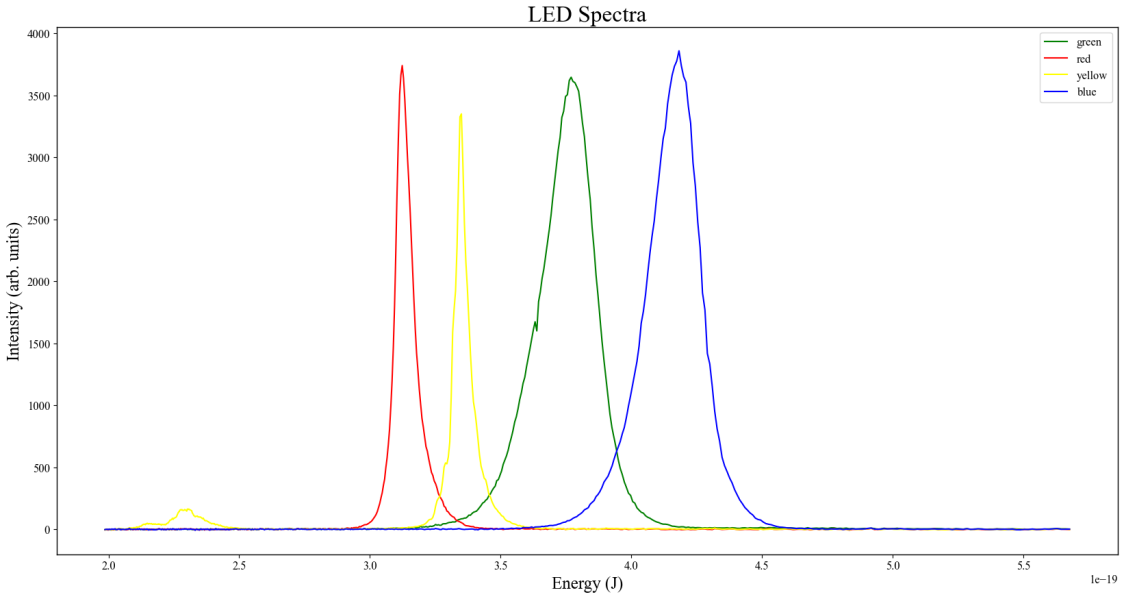
```
    return const.h*const.c/(wavelength*1e-9)

led_data = {
'colour': ['green','red','yellow','blue'],
'peak_wavelength':␣
 ↪[green[peak[0],0],red[peak[1],0],yellow[peak[3],0],blue[peak[4],0]],
'peak_intensity':␣
 ↪[peaks_heights[1]['peak_heights'][0],peaks_heights[3]['peak_heights'][0],peaks_heights[4]['
'FWHM': [fwhm(green[:,0],green[:,1]),fwhm(red[:,0],red[:,1]),fwhm(yellow[:
 ↪,0],yellow[:,1]),fwhm(blue[:,0],blue[:,1])],
'peak_energy':␣
 ↪[energy_peak(green[peak[0],0]),energy_peak(red[peak[1],0]),energy_peak(yellow[peak[3],0]),e
'drive_vol':␣
 ↪[drive_vol(green[peak[0],0]),drive_vol(red[peak[1],0]),drive_vol(yellow[peak[3],0]),drive_v
'energy': [energy(green[:,0]),energy(red[:,0]),energy(yellow[:,0]),energy(blue[:
 ↪,0])],
'intensity': [green[:,1],red[:,1],yellow[:,1],blue[:,1]],
'energy_ev': [energy(green[:,0])/const.e,energy(red[:,0])/const.
 ↪e,energy(yellow[:,0])/const.e,energy(blue[:,0])/const.e],
}
```

[ ]:

[ ]:
```
peaks,peak_heights = find_peaks(white[:,1],height = 2000, distance = 100)

intense = np.linspace(0,3700,10000)
peak_1 = np.full_like(intense,white[peaks[0],0])
peak_2 = np.full_like(intense,white[peaks[1],0])
plt.figure(figsize=(20,10))
plt.plot(white[:,0], white[:,1],color = 'black', label = 'White LED')
plt.plot(white[peaks,0], white[peaks,1], 'x',color = 'blue', label = 'Peaks',␣
 ↪markersize = 10)
plt.plot(peak_1, intense, '--',color = 'red', label = 'Blue Emission')
plt.plot(peak_2, intense, '--',color = 'red', label = 'Phosphorance␣
 ↪Re-emission')
plt.xlabel('Wavelength (nm)')
plt.xlim(400,750)
plt.ylabel('Intensity (arb. units)')
plt.title('White LED Spectra')
plt.legend()
plt.show()
```



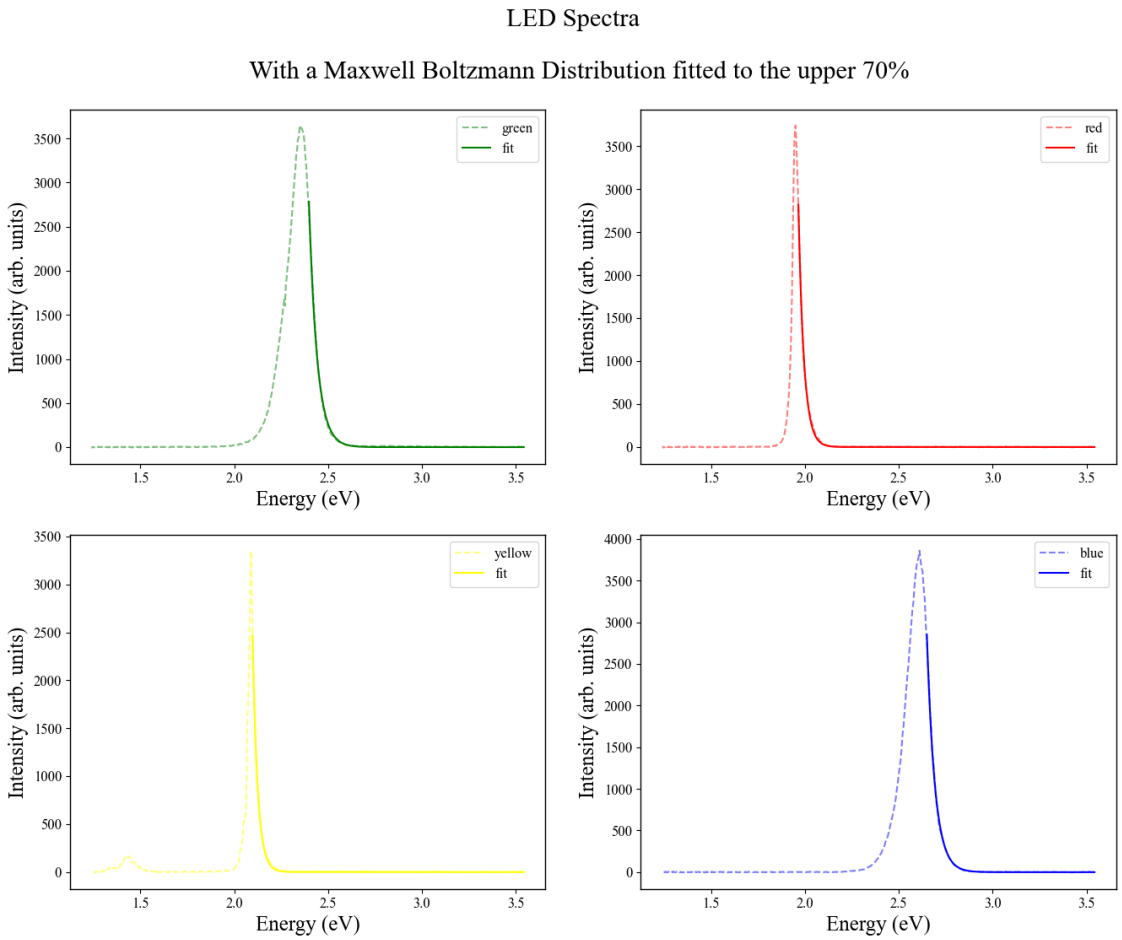
[ ]:
```
plt.figure(figsize=(20,10))

for i in np.arange(len(led_data['energy'])):
    plt.plot(led_data['energy'][i],led_data['intensity'][i], label =␣
 ↪led_data['colour'][i],color = led_data['colour'][i])
    plt.xlabel('Energy (J)')
    plt.ylabel('Intensity (arb. units)')
    plt.title('LED Spectra')
    plt.legend()
plt.show()
```



[ ]:
```
def maxwell_boltzmann(x, a, b):
    return a*np.exp(-x/(kB*b))
guess = [[1.9*10**30,350],[1.9*10**30,390],[1.9*10**30,350],[1.9*10**30,450]]
kB = 8.617e-5  # Boltzmann constant in eV/K
plt.figure(figsize=(16,12))

index_70_percentx = []

for i in np.arange(len(led_data['energy'])):
    data_array = led_data['intensity'][i]

    # Step 1: Identify the peak value
    peak_value = np.max(data_array)

    # Step 2: Calculate 70% of the peak value
    value_70_percent = peak_value * 0.7

    # Step 3: Find the index of the first occurrence of the 70% value
    index_70_percent = np.where(data_array >= value_70_percent)[0][1]
    index_70_percentx.append(index_70_percent)
    #fit a maxwell boltzmann distribution to the data




    popt, pcov = curve_fit(maxwell_boltzmann, led_data['energy_ev'][i][:
 ↪index_70_percentx[i]], led_data['intensity'][i][:index_70_percentx[i]], p0 =␣
 ↪guess[i],maxfev = 1000000)
    err = np.sqrt(np.diag(pcov))
# print(popt, maxwell_boltzmann(led_data['energy_ev'][0][index_70_percent:],␣
 ↪*popt))
    print(f'The colour temperature of the {led_data["colour"][i]} LED is␣
 ↪{popt[1]:.2f} +/- {err[1]:.2f} K')
    plt.suptitle('LED Spectra \n \n With a Maxwell Boltzmann Distribution␣
 ↪fitted to the upper 70%', fontsize=22)
    plt.subplot(2,2,i+1)
    plt.plot(led_data['energy_ev'][i],led_data['intensity'][i], label =␣
 ↪led_data['colour'][i],color = led_data['colour'][i], linestyle = '--',alpha␣
 ↪= 0.5)
    plt.plot(led_data['energy_ev'][i][:index_70_percentx[i]],␣
 ↪maxwell_boltzmann(led_data['energy_ev'][i][:index_70_percentx[i]],␣
 ↪*popt),color = led_data['colour'][i], label='fit')
    plt.xlabel('Energy (eV)')
```

```
    plt.ylabel('Intensity (arb. units)')

    plt.legend()
plt.show()
```

```
The colour temperature of the green LED is 502.10 +/- 3.34 K
The colour temperature of the red LED is 341.45 +/- 1.51 K
The colour temperature of the yellow LED is 309.15 +/- 1.35 K
The colour temperature of the blue LED is 495.61 +/- 3.23 K
```



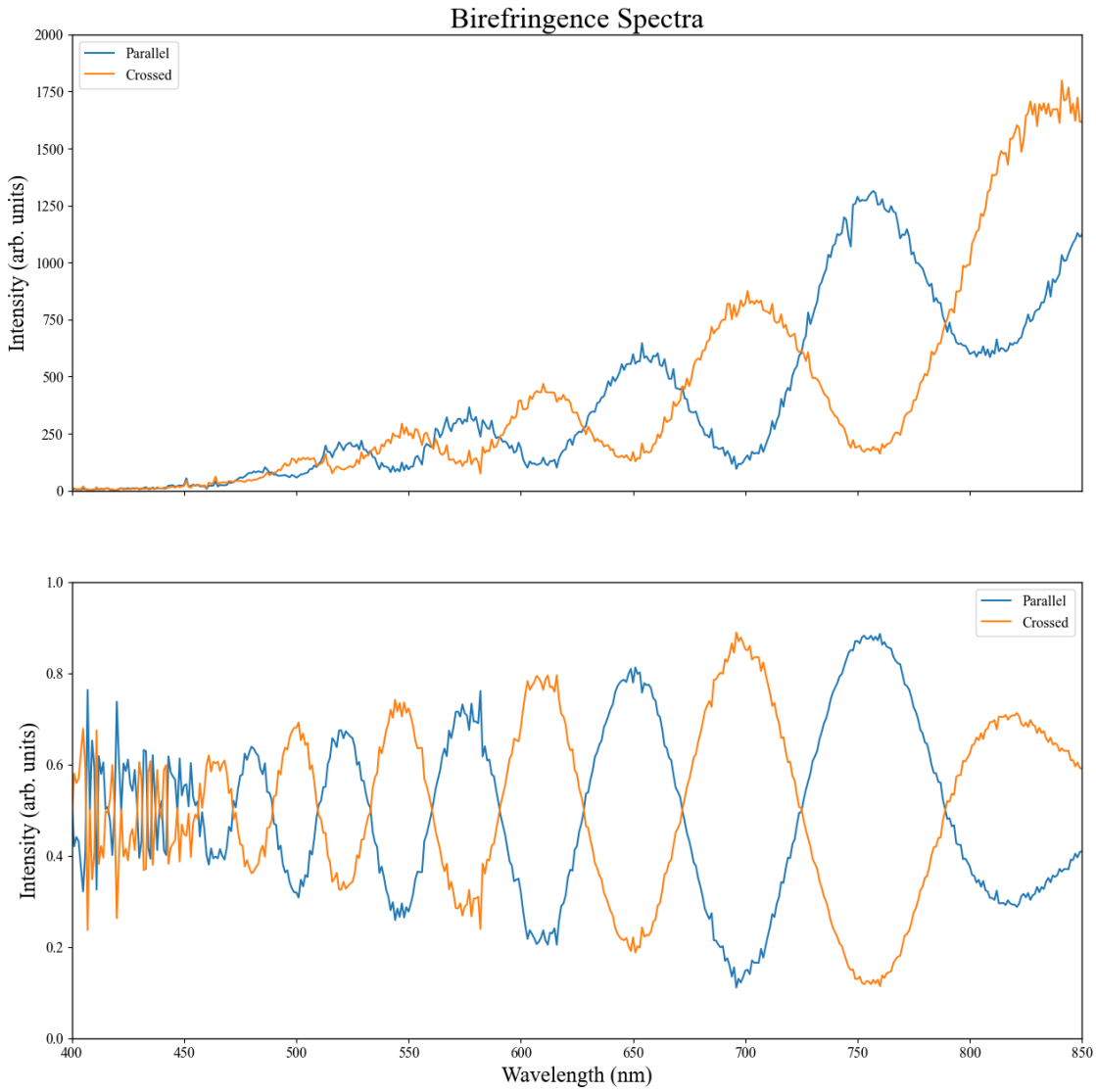
[ ]:

[ ]:

7 Birefringment

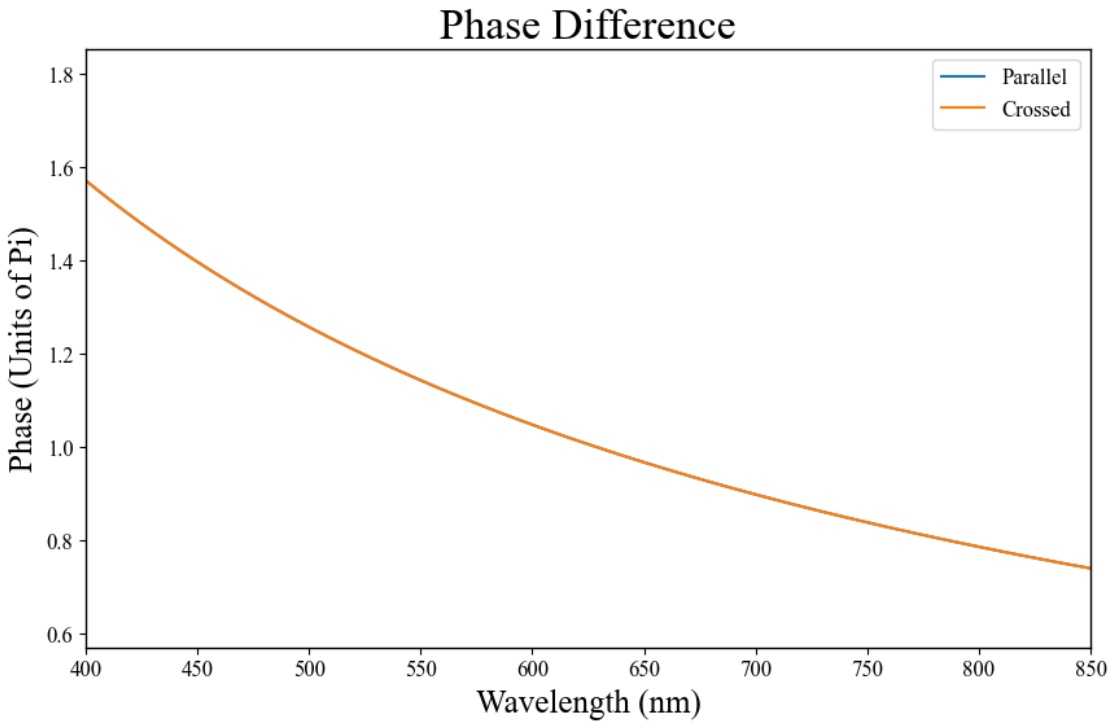
[ ]:
```
para = loadtxt('data/birefringement/bi_para0.txt')
cross = loadtxt('data/birefringement/bi_cross0.txt')
back = loadtxt('data/birefringement/bi_back0.txt')

para[:,1] = para[:,1] - back[:,1]
cross[:,1] = cross[:,1] - back[:,1]
theo_i_para = para[:,1]/(para[:,1]+cross[:,1])
theo_i_cross = cross[:,1]/(para[:,1]+cross[:,1])

del_phi = np.arccos((2*theo_i_para)-1)



plt.figure(figsize=(15,15))
plt.subplot(2,1,1)
plt.plot(para[:,0], para[:,1], label = 'Parallel')
plt.plot(cross[:,0], cross[:,1], label = 'Crossed')
plt.xlim(400,850)
plt.xticks(ticks = np.arange(400,850,50),labels = [])
plt.ylabel('Intensity (arb. units)')
plt.ylim(0,2000)
plt.title('Birefringence Spectra')
plt.legend()
plt.subplot(2,1,2)
plt.plot(para[:,0], theo_i_para, label = 'Parallel')
plt.plot(cross[:,0], theo_i_cross, label = 'Crossed')
plt.xlabel('Wavelength (nm)')
plt.xlim(400,850)
plt.ylabel('Intensity (arb. units)')
plt.ylim(0,1)
plt.legend()
plt.show()
```

```
C:\Users\lewis\AppData\Local\Temp\ipykernel_31196\3565421494.py:10:
RuntimeWarning: invalid value encountered in arccos
  del_phi = np.arccos((2*theo_i_para)-1)
```



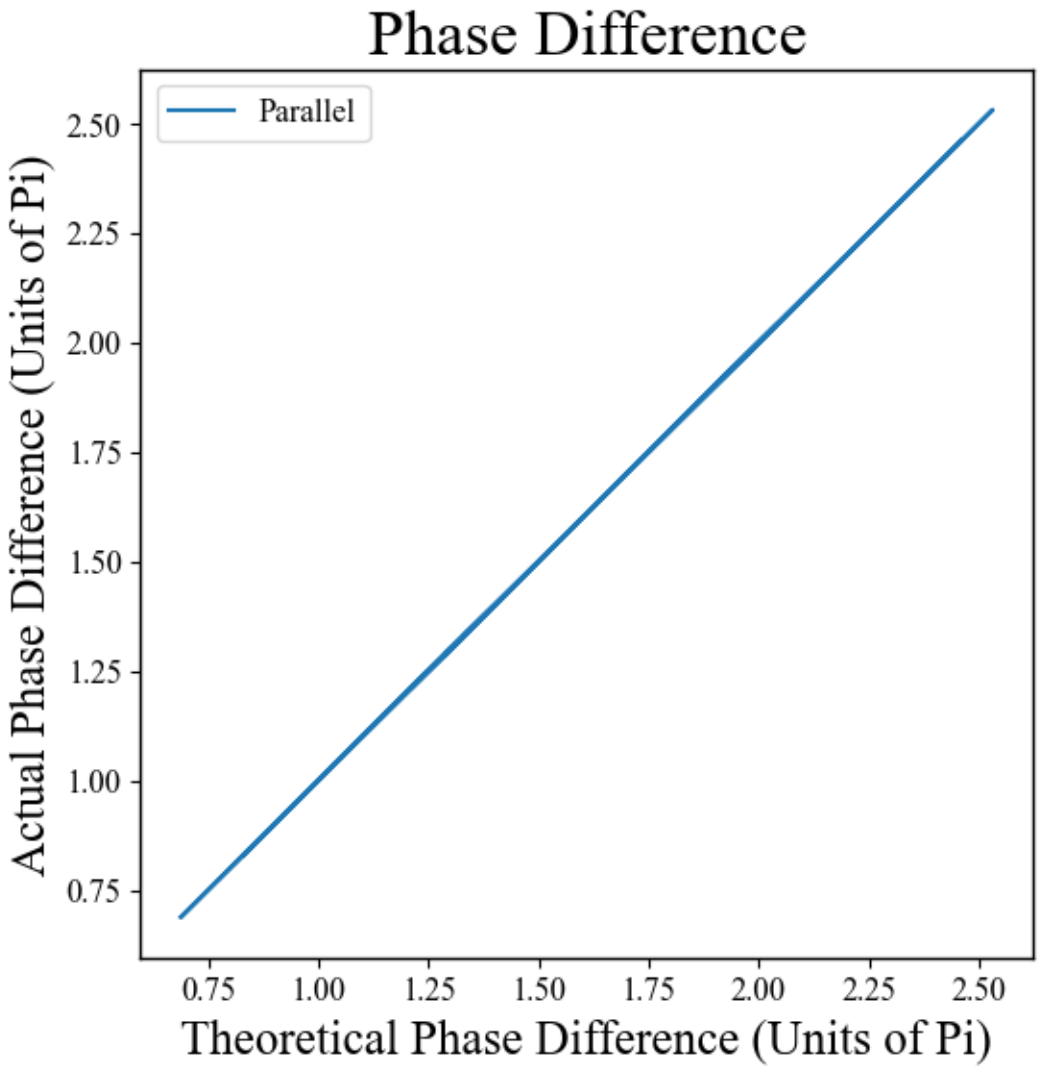
[ ]:
```
phase = (2*np.pi/para[:,0])*100
phase_cross = (2*np.pi/cross[:,0])*100
plt.figure(figsize=(10,6))
plt.plot(para[:,0], phase, label = 'Parallel')
plt.plot(cross[:,0], phase_cross, label = 'Crossed')
plt.xlim(400,850)
plt.xlabel('Wavelength (nm)')
plt.ylabel('Phase (Units of Pi)')
plt.title('Phase Difference')
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x1f583fffe50>
```



[ ]:
```
phase_act = 2*np.arctan(np.sqrt((cross[:,1]/para[:,1])))

plt.figure(figsize=(6,6))
plt.plot(del_phi,phase_act, label = 'Parallel')
plt.xlabel('Theoretical Phase Difference (Units of Pi)')
plt.ylabel('Actual Phase Difference (Units of Pi)')
plt.title('Phase Difference')
plt.legend()
plt.show()
```

```
C:\Users\lewis\AppData\Local\Temp\ipykernel_31196\1084237382.py:1:
RuntimeWarning: invalid value encountered in sqrt
  phase_act = 2*np.arctan(np.sqrt((cross[:,1]/para[:,1])))
```


Phase Difference
Parallel
Actual Phase Difference (Units of Pi)
2.50
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
Theoretical Phase Difference (Units of Pi)

[ ]: