Homework #2
EE 541: Fall 2023

Name: Nissanth Neelakandan Abirami
USC ID: 2249203582
Instructor: Dr. Franzke

1) Raman spectroscopy is a technique that uses inelastic scattering of light to identify unknown chemical substances. Spectral “peaks” indicate vibrational and rotational modes and are of special importance because they act like a chemical fingerprint. Raman spectroscopy measures photon intensity vs. Raman shift. The Raman shift relates the frequencies of the exciting laser and the scattered photons and is often reported as a wavenumber — the frequency difference in wavelengths per cm (i.e., ,cm−1).

• Generate a molecular fingerprint using the spectroscopic data in raman.rod. The file contains intensity vs. wavenumber data for an unknown chemical sample. A Raman Open Database (ROD) file includes content in addition to the raw intensity data:

```
# content
more content
_raman_spectrum.intensity
wavenumber1 intensity1
wavenumber2 intensity2
...
Wavenumber n intensity n
```

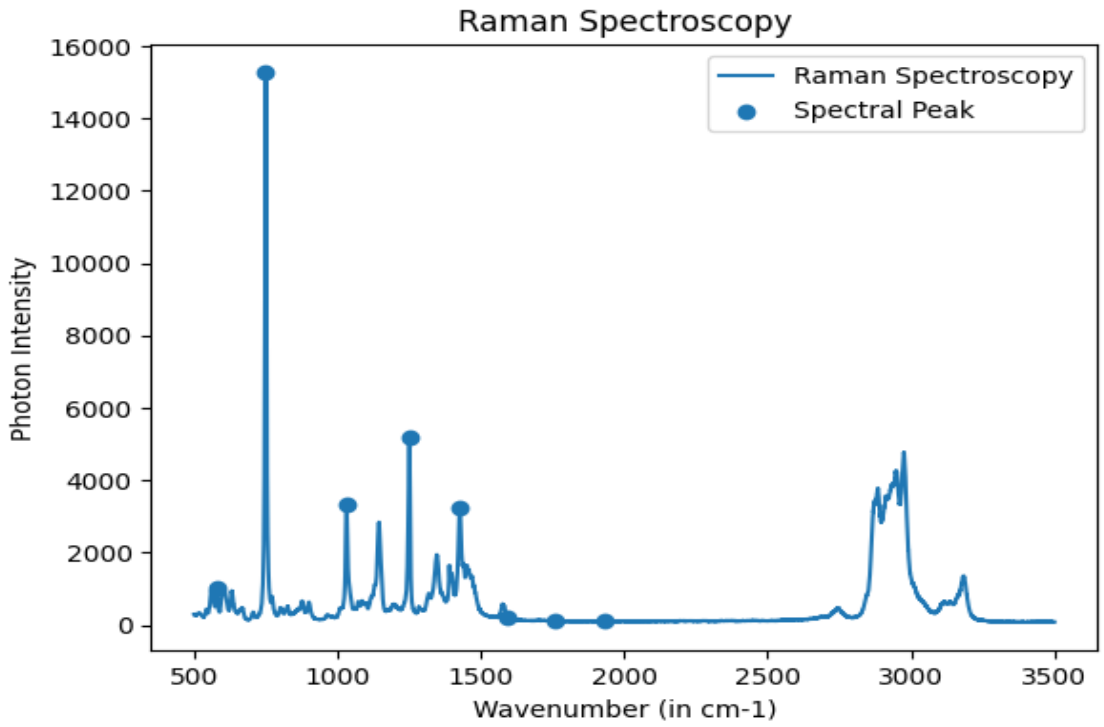
Use string matching to ignore all lines before raman spectrum.intensity. Load valid (wavenumber, intensity) pairs until the first invalid intensity line (or upon reaching the end of file). Use the method below to estimate the wavenumbers of all spectral peaks. You may use any standard NumPy or SciPy packages or experiment with your own algorithms.

• First detect peaks in the raw spectral data. Use the peak locations to focus on regions of interest within the spectrum. For instance: if you detect peaks at x1 cm−1 and x2 cm−1 use regions of interest: [x1 − n1, x1 + n1] and [x2 − n2, x2 + n2]. Experiment to find “good” widths n1, n2, etc. Then use a spline to interpolate intensity within each region of interest. Calculate zero-crossings of the derivative to estimate wavenumbers with maximum intensity.

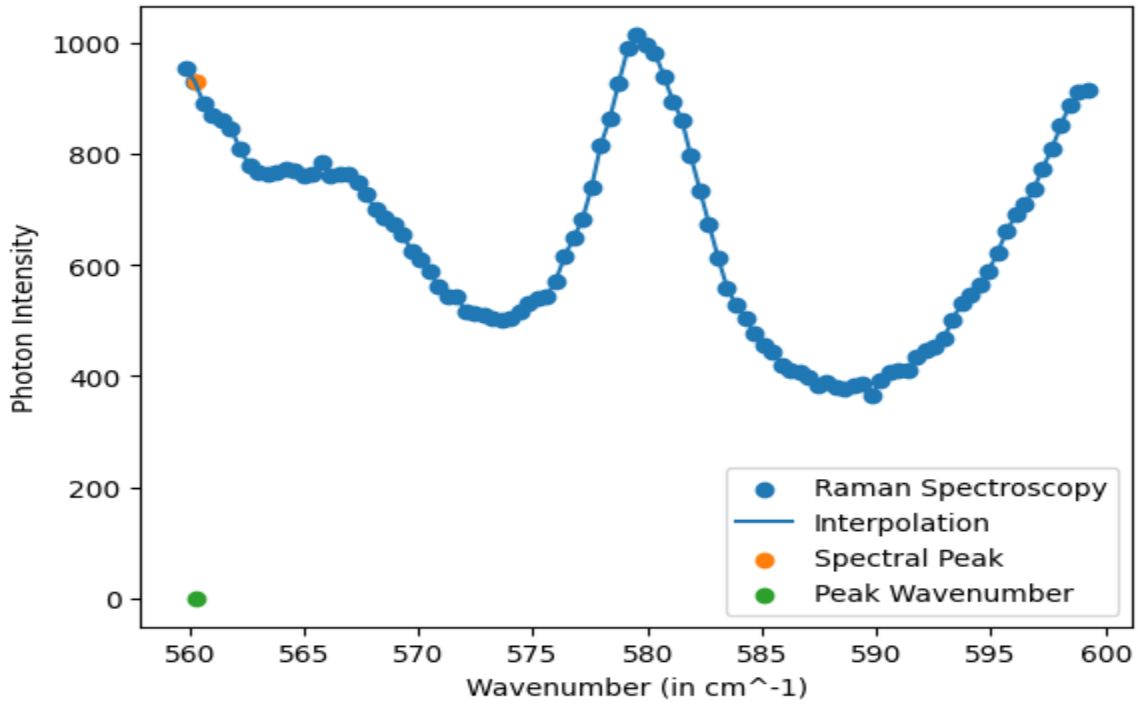
(a) Print the wavenumber estimates for the eight largest spectral peak to STDOUT sorted by magnitude (largest first).

(b) Create a figure that shows the Raman data (intensity vs. wavenumber) and mark each of the maximum intensity values.

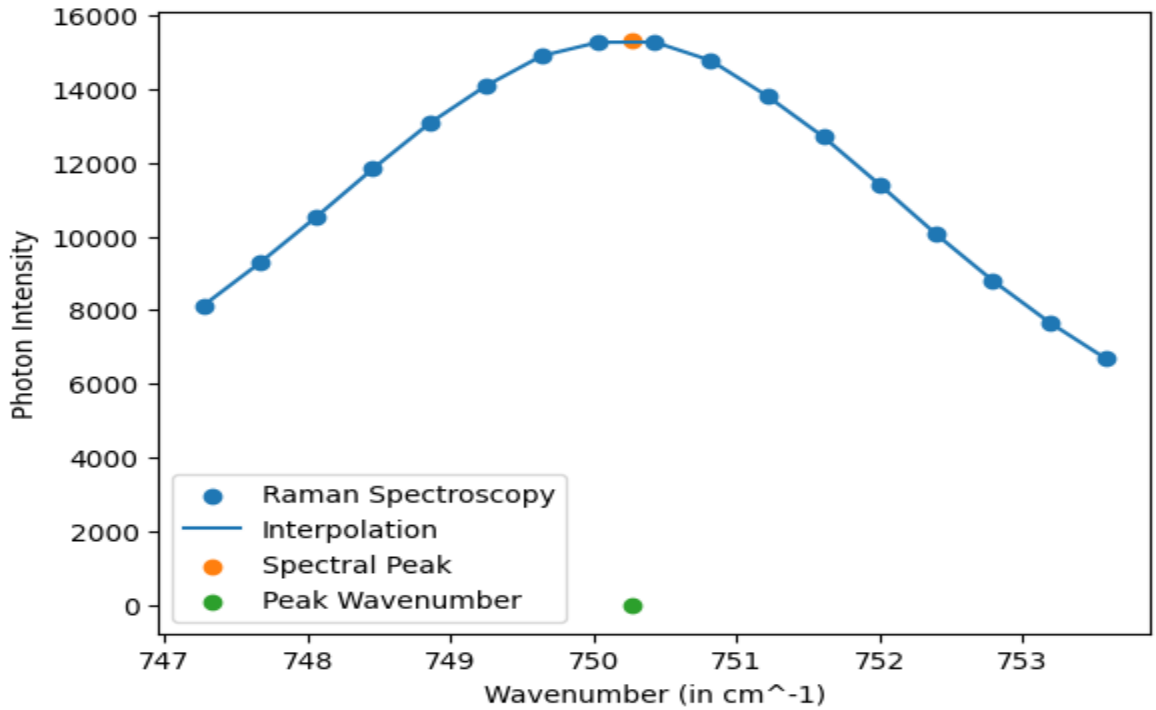
(c) Produce a “zoomed-in” figure for the “regions of interest” corresponding to the four largest peaks. Plot the raw spectral data and overlay your interpolating function. Use a marker to show the wavenumber with maximal intensity.



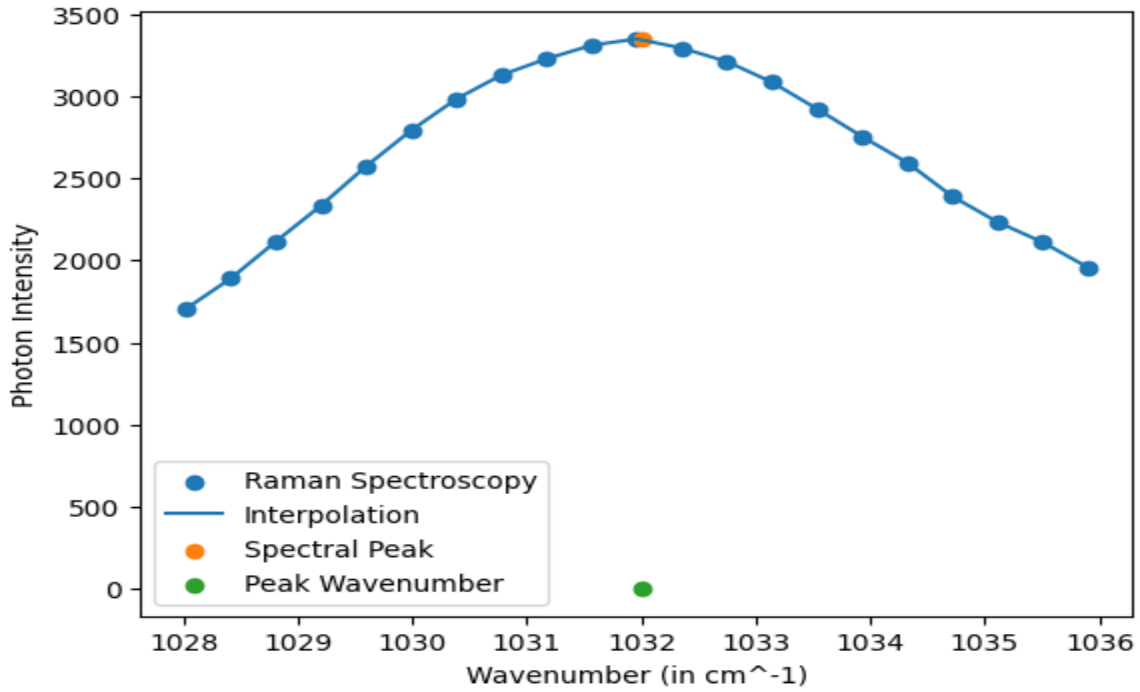
Peak Wavenumber (in cm^-1): 560.2481297979798



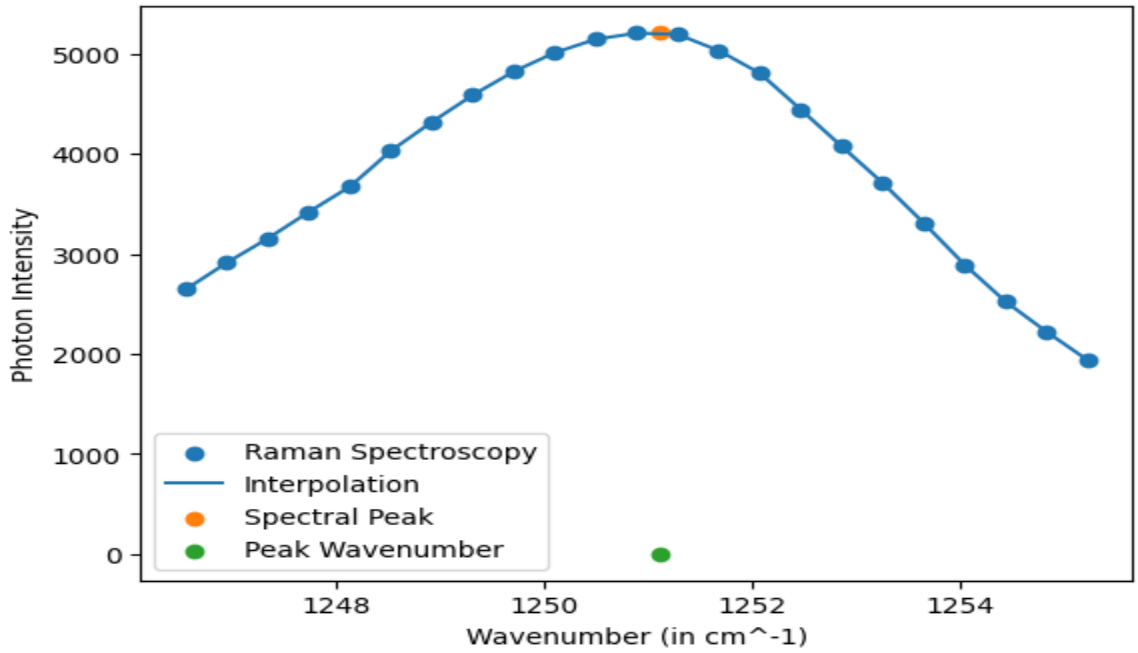
Peak Wavenumber (in cm^-1): 750.2674828282828



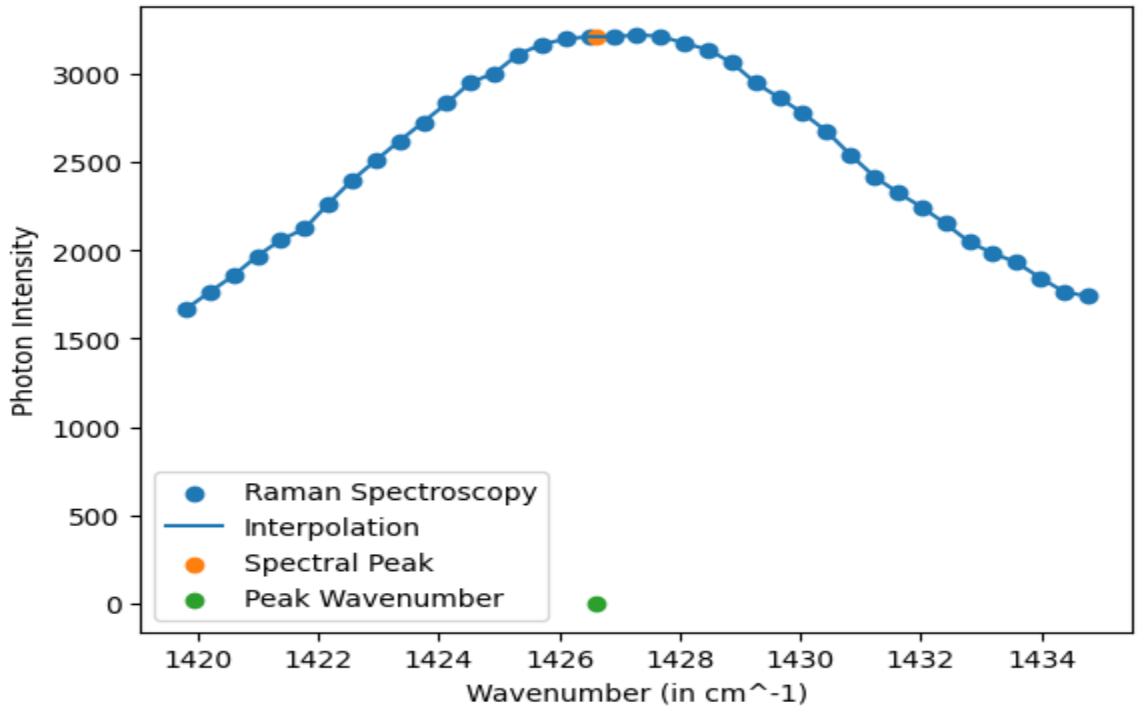
Peak Wavenumber (in cm^-1): 1031.9993727272727



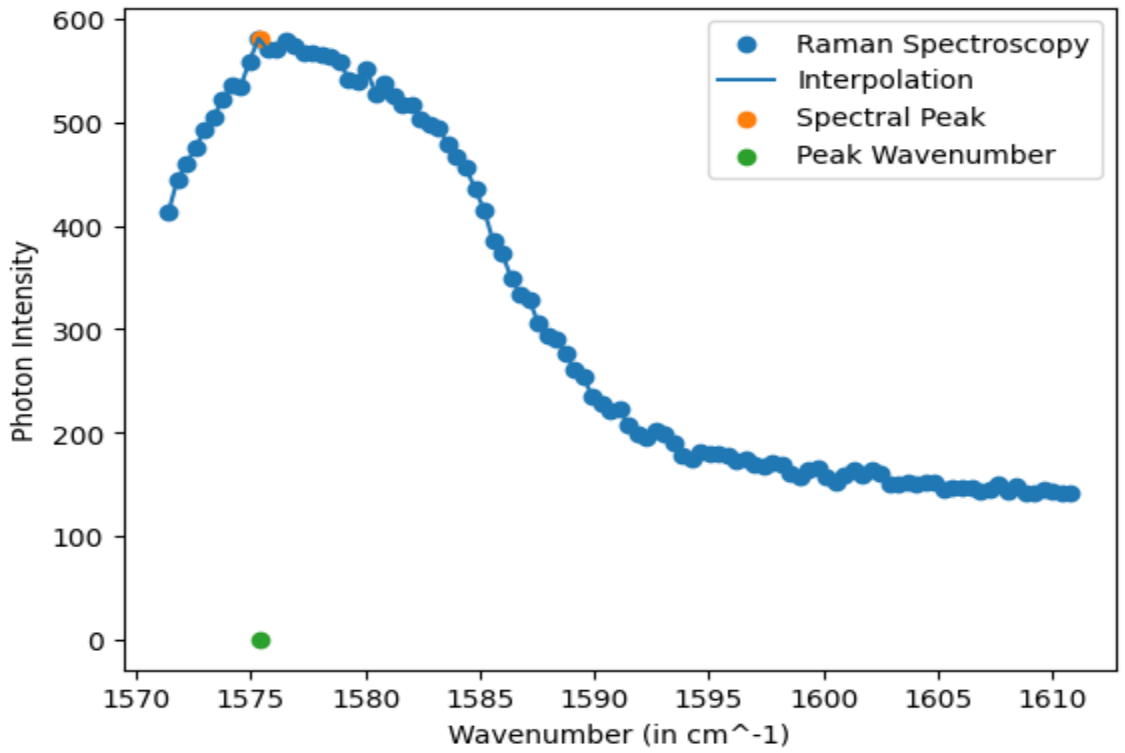
Peak Wavenumber (in cm^-1): 1251.1047



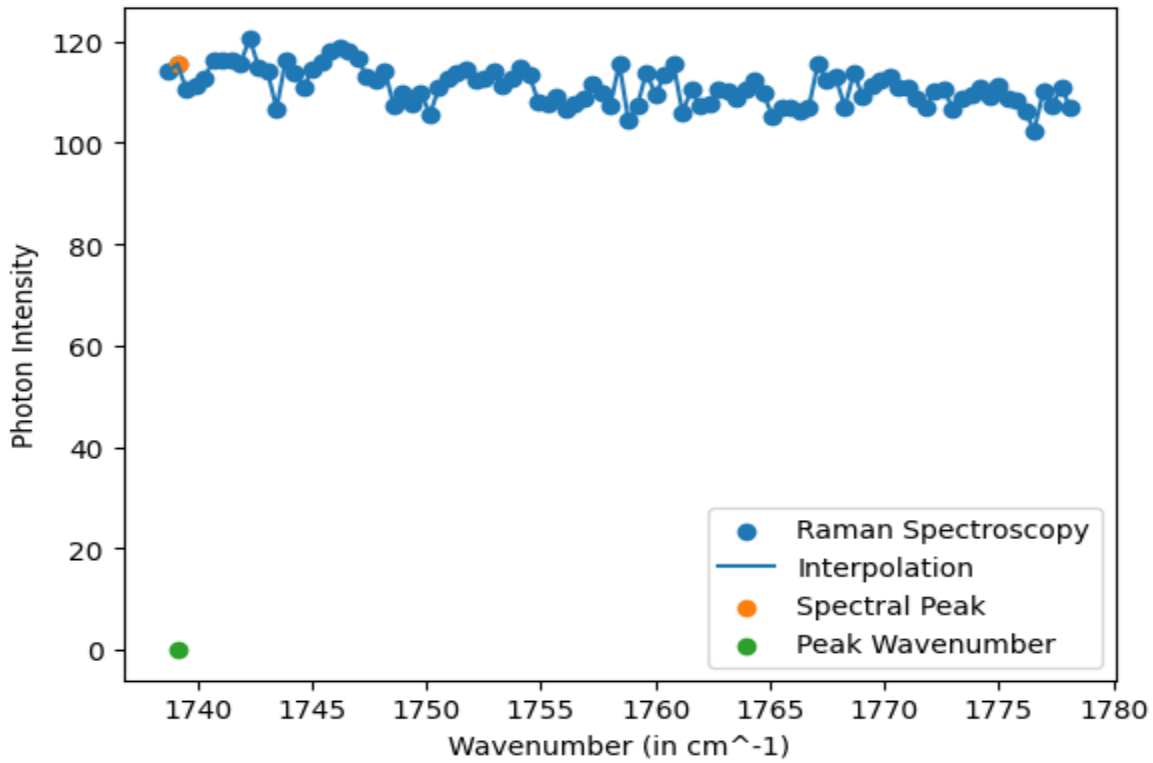
Peak Wavenumber (in cm^-1): 1426.6069363636364



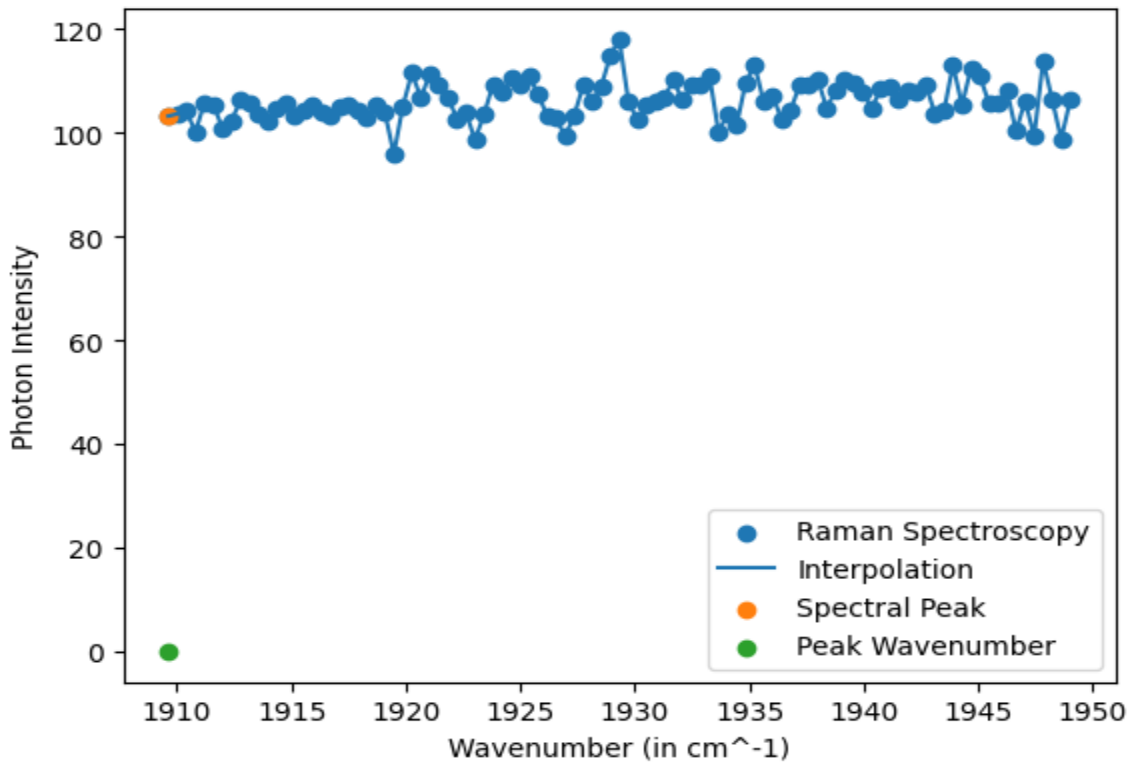
Peak Wavenumber (in cm^-1): 1575.3776929292928



Peak Wavenumber (in cm^-1): 1739.1429303030304



Peak Wavenumber (in cm^-1): 1909.6338



APPENDIX:

```
#Import necessary libraries and modules
import sys
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
from scipy.interpolate import CubicSpline

#Set the number of spectral peaks to analyze (N)
N = 8
raman_data = np.loadtxt('./raman (1).txt')
#Load Raman spectroscopy data into arrays.
wavenumber = raman_data[:, 0]
intensity = raman_data[:, 1]

#Detect the N highest spectral peaks
peak=find_peaks(intensity, height=0,distance = 400)[0][:N]
#Create a new figure for plotting
plt.figure()
```

```
#Plot the raw Raman spectroscopy data.
plt.plot(wavenumber, intensity)
#Scatter plot the detected spectral peaks
plt.scatter(wavenumber[peak], intensity[peak])
#Label the x-axis and y-axis.
plt.xlabel('Wavenumber (in cm-1)')
plt.ylabel('Photon Intensity')
plt.title('Raman Spectroscopy')
plt.legend(['Raman Spectroscopy', 'Spectral Peak'])
#Display the Raman spectroscopy plot
plt.show()

#Initialize an empty list for "good" peak widths
good_widths = []
#Define a function to calculate "good" peak width
def get_good_width(peak_index, intensity):
#Loop through the detected spectral peaks
    for width in range(50, 0, -1):
        threshold = intensity[peak_index] / 2
#Calculate and append "good" peak widths to the list.
        if intensity[peak_index] - intensity[peak_index - width] <
threshold:
            return width
#Iterate through the N spectral peaks.
for i in range(len(peaks)):
    good_width = get_good_width(peaks[i], intensity)
    good_widths.append(good_width)

# Iterate through the regions of interest corresponding to the spectral
peaks
for j in range(N):
#Define a neighborhood around each spectral peak
    neighborhood = np.arange(peaks[j] - good_widths[j], peaks[j] +
good_widths[j] + 1, dtype=np.int32)
#Create a cubic spline interpolation for the intensity
    intensity_interpolate = CubicSpline(wavenumber[neighborhood],
intensity[neighborhood])
#Generate a list of evenly spaced wavenumbers.
    wavenumber_list = np.linspace(wavenumber[neighborhood[0]],
wavenumber[neighborhood[-1]], num=100)
```

```
#Calculate the derivative of intensity.
    derivative_list = np.array([((intensity_interpolate(i + 0.01) -
intensity_interpolate(i - 0.01)) / 0.02) for i in wavenumber_list])
#Find zero-crossings in the derivative.
    zero_crossings = wavenumber_list[derivative_list < 0]
#If zero-crossings are found, select the first one.
    if len(zero_crossings) > 0:
        peak_wavenumber = zero_crossings[0]


    # Create a figure for the spectral peak
    plt.figure()
    #Scatter plot the data points in the neighborhood
    plt.scatter(wavenumber[neighborhood], intensity[neighborhood])
    #Plot the intensity interpolation curve
    plt.plot(wavenumber[neighborhood],
intensity_interpolate(wavenumber[neighborhood]))
    #Scatter plot the peak wavenumber
    plt.scatter(peak_wavenumber, intensity_interpolate(peak_wavenumber))
    #Scatter plot a reference point at y=0
    plt.scatter(peak_wavenumber,0)
    #Print the peak wavenumber to the console
    print(f'Peak Wavenumber (in cm^-1): {peak_wavenumber}')
    #Label the x-axis and y-axis
    plt.xlabel('Wavenumber (in cm^-1)')
    plt.ylabel('Photon Intensity')
    #Add a legend to the plot
    plt.legend(['Raman Spectroscopy', 'Interpolation', 'Spectral Peak',
'Peak Wavenumber'])
    #Display the spectral peak plot
    plt.show()
```

2. Unsupervised clustering algorithms are an efficient means to identify groups of related objects within large populations. Implement the following two clustering algorithms and apply them to the data in cluster.txt. The file contains data as: x, y, class. Use a regular expression to remove lines that are empty or that are invalid data. You may safely ignore any line that fails the regular expression.

(a) Use K-Means clustering with 3-clusters to label each (x, y) pair as Head, Ear Right, or Ear - Left. You may use any standard NumPy or SciPy packages or experiment with your own Implementation.

Produce a scatter plot marking each (x, y) pair as either BLUE (class = Head), RED (class = Ear Left) or GREEN (class = Ear Right). Compare the K-means predicted labels to the true label and generate a confusion matrix showing the respective accuracies.

(b) Gaussian Mixture Models (GMM) are a common method to cluster data from multi-modal probability densities. Expectation maximization (EM) is an iterative procedure to compute (locally) optimal GMM parameters – GMM cluster means μk, covariances Σk, and mixing weights wk. EM consists of two-steps. The E[xpectation]-step uses the mixture parameters to update estimates of hidden variables. The true but unknown class is an example of a hidden variable. The M[aximization]-step then uses the new hidden variable estimates to update the mixture parameter estimates. This back-and forth update provably increases the likelihood function and the estimate eventually converges to a local likelihood maximum. The GMM update equations follow.

E-Step: use current mixture parameters estimates to calculate membership probabilities (a.k.a., the hidden variables) for each sample, γk (xn),

$$\gamma_k(x_n) = \frac{w_k f(x_n; \mu_k, \Sigma_k)}{\sum_{j=1}^{K} w_j f(x_n; \mu_j, \Sigma_j)}$$

M-step: use new γk (xn) to update mixture parameter estimates,

$$\mu_k = \frac{\sum_{n=1}^{N} \gamma_k(x_n)\, x_n}{\sum_{n=1}^{N} \gamma_k(x_n)}$$

$$\Sigma_k = \frac{\sum_{n=1}^{N} \gamma_k(x_n)\, (x_n - \mu_k)(x_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma_k(x_n)}$$

$$w_k = \frac{1}{N} \sum_{n=1}^{N} \gamma_k(x_n)$$

for $k \in \{1, \ldots, K\}$ where $K \in Z+$ is the (predefined) number of mixture components, x_n for $n \in \{1, \ldots, N\}$ are the data samples, and $f(x; \mu_k, \Sigma_k)$ is the d-dimensional jointly Gaussian pdf:

$$f(x; \mu_k, \Sigma_k) = \frac{\exp -1/2\, (x - \mu_k)^T \Sigma^{-1}_k (x - \mu_k)}{\sqrt{(2\pi)^d |\Sigma_k|}}$$

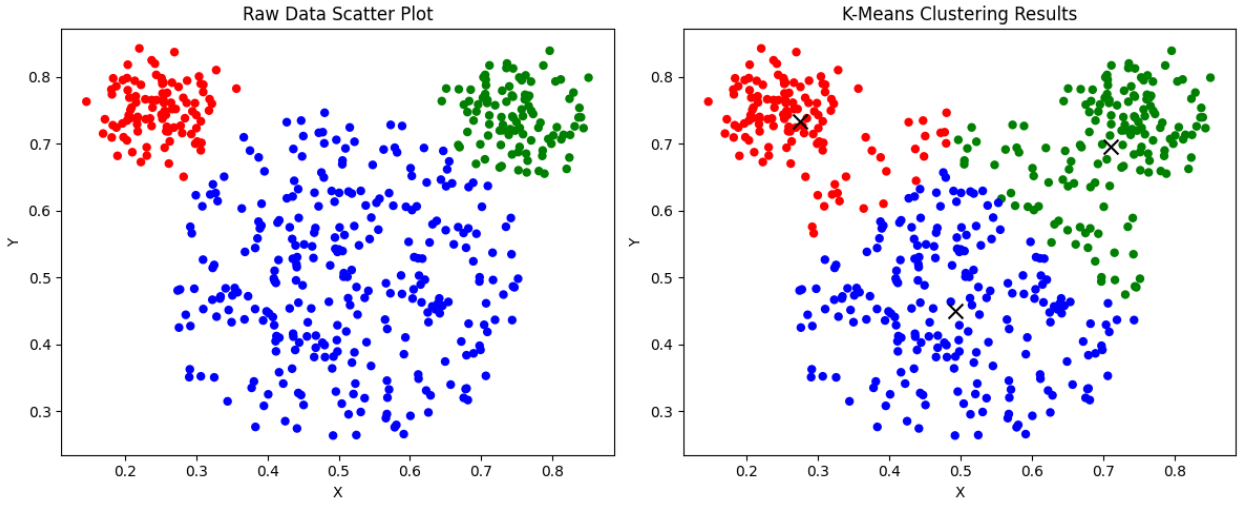
• Implement Expectation Maximization and use it to estimate mixture parameters for a 3-component GMM for the cluster.csv dataset. DO NOT use an EM implementation from NumPy, SciPy, or any other package. You must implement and use the above EM Equations.

• Initialize γk (xn) for each sample using the K-means labels from part (a) as “one-hot” membership probabilities (i.e., initialize one of the probabilities as “1” and all others are “0”). Then compute initial μk and Σk for each mixture component.
• Run EM until it has sufficiently converged. Use either the negative log-likelihood

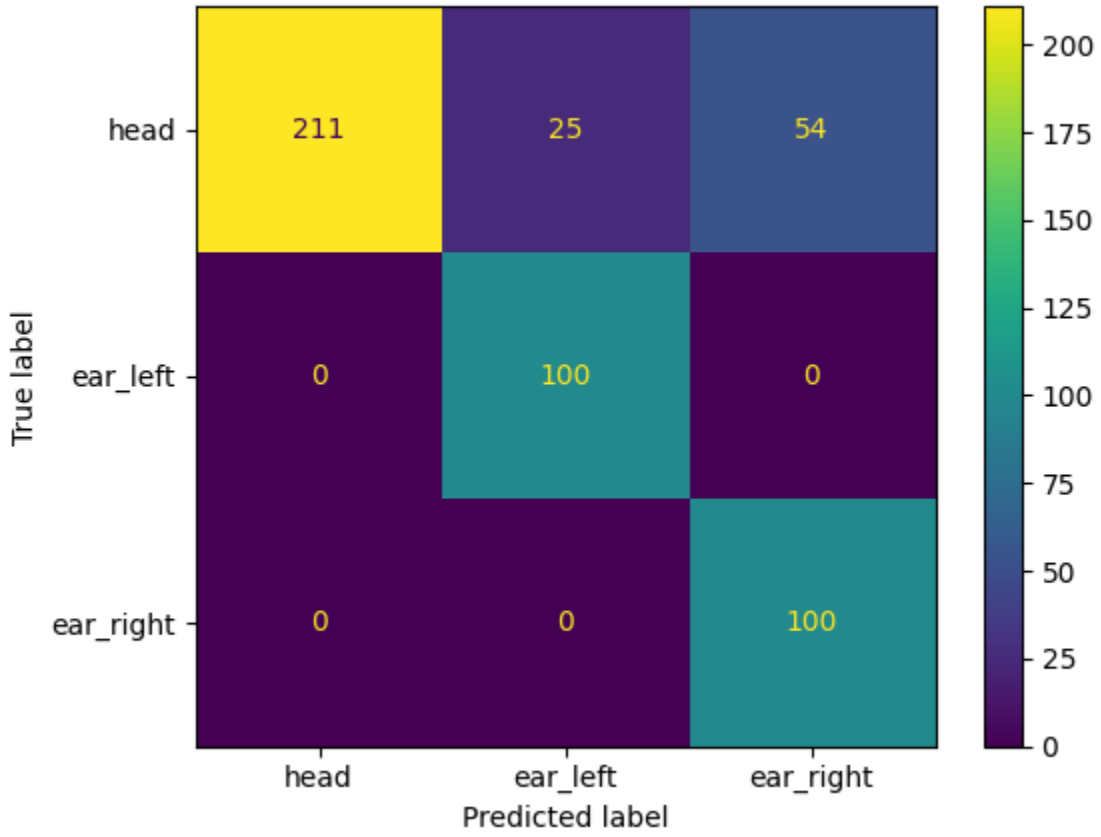
l =XNn=1log XKk=1wkf(xn; μk, Σk)!

as a convergence metric or monitor the class assignments until there are only small changes. Be aware that some points may “flip-flop” even when fully converged. Assign each datapoint to the mixture component with the largest membership probability. Produce a Blue-Red-Green scatter plot as in part (a) and generate a confusion matrix showing the respective classification accuracies.
• Generate figures showing the class assignments during the first four iterations.
• Comment on the difference between the clustering result in (a) and (b). Describe any obvious difference between the plots and indicate which performs better.

a)
Confusion Matrix:
[[211 25 54]
 [0 100   0]
 [0   0 100]]



<Figure size 640x480 with 0 Axes>



b)

APPENDIX:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.vq import kmeans, vq
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Load data from 'cluster.txt'
data = np.loadtxt('cluster.txt', dtype={'names': ('x', 'y',
'label'), 'formats': (float, float, 'U10')})
X, Y = data['x'], data['y']
XY = np.column_stack((X, Y))  # Combine X and Y into a 2D array

# Perform K-Means clustering with 3 clusters
```

```
centroids, _ = kmeans(XY, 3)
cluster_labels, _ = vq(XY, centroids)

# Normalize and map class labels
true_labels = data['label']
true_labels = [label.replace(' ', '_').lower() for label in
true_labels]
class_mapping = {0: 'head', 1: 'ear_left', 2: 'ear_right'}
predicted_labels = [class_mapping[label] for label in
cluster_labels]

# Create a confusion matrix
confusion = confusion_matrix(true_labels, predicted_labels,
labels=['head', 'ear_left', 'ear_right'])

# Define colors for class labels
colors = {'head': 'blue', 'ear_left': 'red', 'ear_right': 'green'}

# Plot raw data
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
label_colors = [colors[label] for label in true_labels]
plt.scatter(X, Y, c=label_colors, marker='o', s=25)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Raw Data Scatter Plot')

# Plot K-Means results
plt.subplot(1, 2, 2)
cluster_colors = [colors[label] for label in predicted_labels]
plt.scatter(X, Y, c=cluster_colors, marker='o', s=25)
plt.scatter(centroids[:, 0], centroids[:, 1], c='black', marker='x',
s=100)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('K-Means Clustering Results')
plt.tight_layout()

# Print the confusion matrix
print("Confusion Matrix:")
```

```
print(confusion)

# Plot the confusion matrix
plt.figure()
cm_display = ConfusionMatrixDisplay(confusion_matrix=confusion,
display_labels=['head', 'ear_left', 'ear_right'])
cm_display.plot()
plt.show()

# Show the plots
plt.show()
```