Numerical Methods - Assignment 2

1. Gaussian Features

Write a function that creates a set of evenly-spaced Gaussian functions.

The input should be an vector x, a number of Gaussians N, and a fixed width σ.

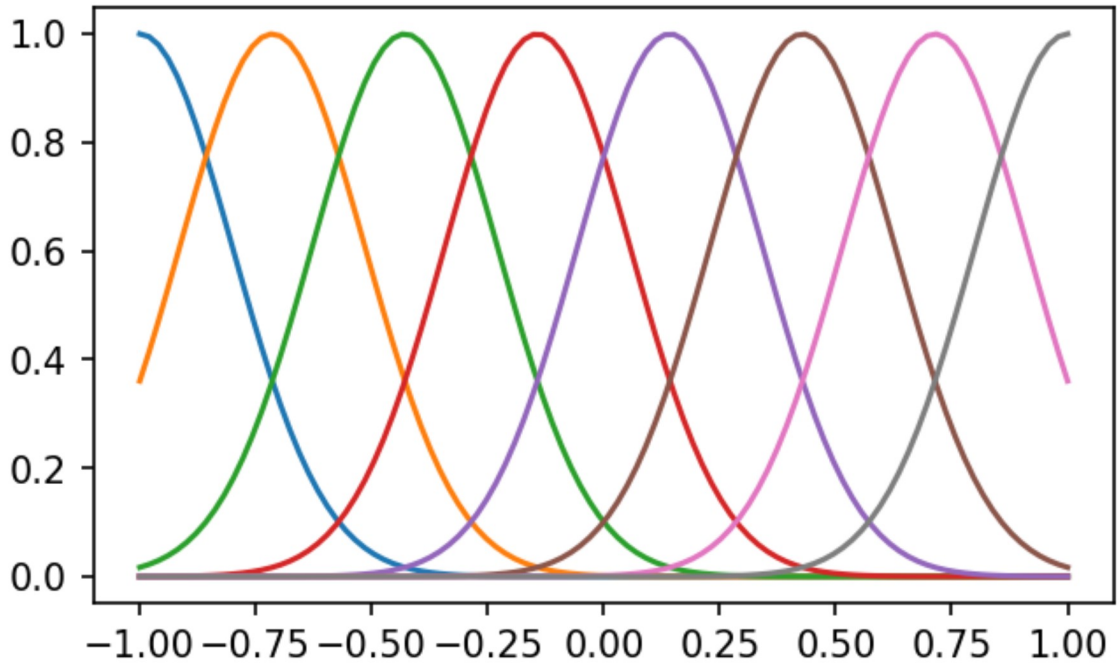
In [62]:
```
#making a function that will create as manuy gaussians as I need
#and will use this function later
import numpy as np
def gaussian_features(x, N, sigma):
    #this function assumes that the x input is a row vec
    #so we will reshape it into a col vec
    x = x.reshape(-1)
    #do not want to assign a mean for each peak, find evenly spaced gauss distrib
    #xvec is now the means of each of our gaussians
    xmeans = np.linspace(min(x),max(x),N) #do this for N number of gaussians
    features = []
    #do a for loop to do the gaussian eqn for each col of x and for N number gauss
    for xnow in xmeans:
        features.append(np.exp(-((x-xnow)**2)/(2*sigma**2)))
```

Use this function to plot 8 evenly-spaced Gaussians from -1 to 1 with a width of 0.2.

You can randomly define the resolution of the range.

In [63]:
```
import matplotlib.pyplot as plt
x = np.linspace(-1,1,100)#changing the number of the points to 100 made the curves
mygauss = gaussian_features(x,8,0.2)
fig, ax = plt.subplots(figsize = (5, 3), dpi = 150)
```

Out[63]:
```
[<matplotlib.lines.Line2D at 0x18d2fcb6640>,
 <matplotlib.lines.Line2D at 0x18d2f538fa0>,
 <matplotlib.lines.Line2D at 0x18d2f5383d0>,
 <matplotlib.lines.Line2D at 0x18d2f538400>,
 <matplotlib.lines.Line2D at 0x18d2f538d00>,
 <matplotlib.lines.Line2D at 0x18d2f538d90>,
 <matplotlib.lines.Line2D at 0x18d2f538610>,
 <matplotlib.lines.Line2D at 0x18d2f5380a0>]
```



2. General Linear Regression

Determine the best-fit of the peaks below using general linear regression.

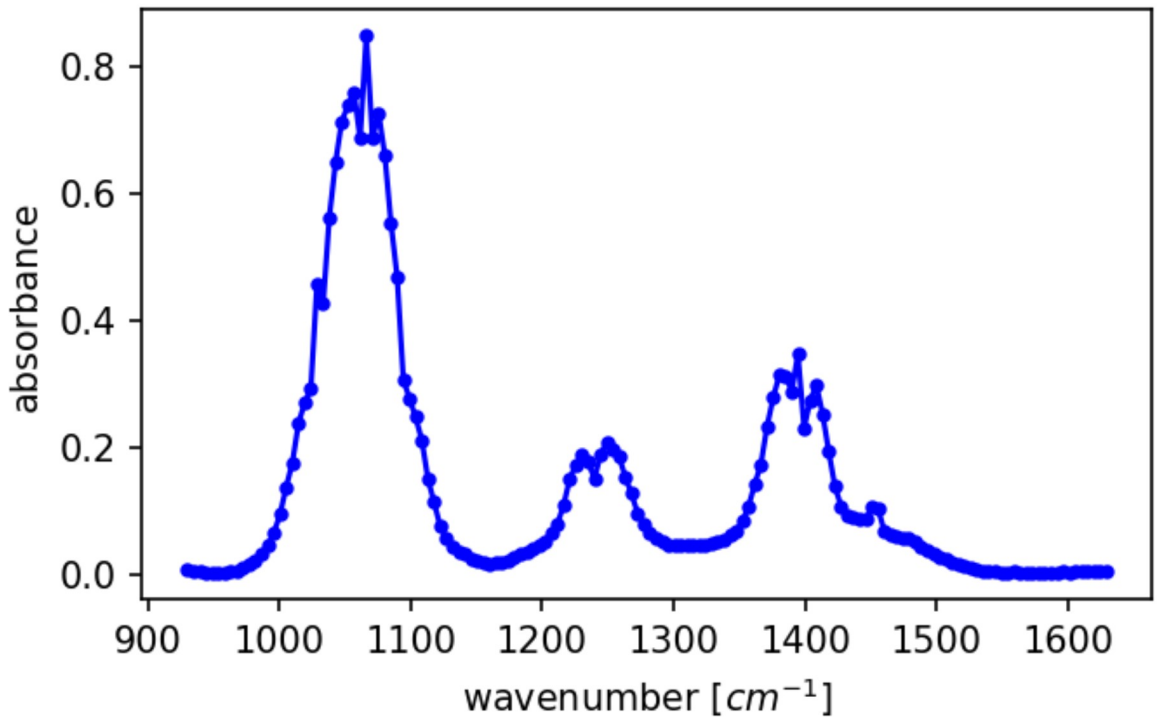
Plot the result of your regression model along with the original data.

You may assume that:

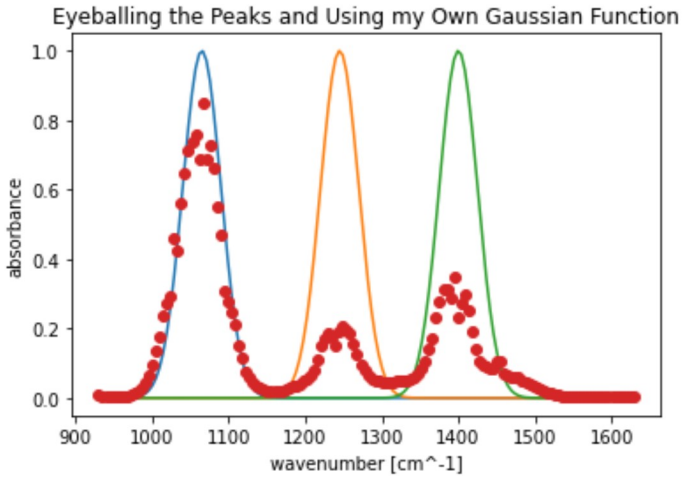
- The peaks follow a Gaussian distribution.
- There are 3 peaks of the **same width** in this region of the spectra below.

In [65]:
```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('data/ethanol_IR.csv')
x_all = df['wavenumber [cm^-1]'].values
y_all = df['absorbance'].values

x_peak = x_all[100:250]#just so they can the peaks for that range of the ethanol s
y_peak = y_all[100:250]

fig, ax = plt.subplots(figsize = (5, 3), dpi = 150)
```

```
ax.plot(x_peak, y_peak, '-b', marker = '.')
ax.set_xlabel('wavenumber [$cm^{-1}$]')
ax.set_ylabel('absorbance');

#Follow Gaussian Distribution, so you have a gaussian basis matrix
#knowing they're the same width
```



In [66]:
```
import numpy as np
#1st thing I will do is eyeball the means and make my own gaussian
#using my own made gaussian functions
#mean guesses
mu1 = 1065
mu2 = 1245
mu3 = 1400
sig = 25
xgauss = np.zeros((len(x_peak),3)) #3 cols bc 3 peaks
#change the first col to be the gaussian eqn
xgauss[:,0] = np.exp(-(x_peak - mu1)**2/(2*(sig**2)))
#change the second col
xgauss[:,1] = np.exp(-(x_peak - mu2)**2/(2*(sig**2)))
#change the 3rd col
xgauss[:,2] = np.exp(-(x_peak - mu3)**2/(2*(sig**2)))

fig, ax = plt.subplots()
ax.plot(x_peak, xgauss[:,0])
ax.plot(x_peak, xgauss[:,1])
ax.plot(x_peak, xgauss[:,2])
ax.plot(x_peak, y_peak, 'o')
ax.set_title('Eyeballing the Peaks and Using my Own Gaussian Function')
ax.set_xlabel('wavenumber [cm^-1]')
ax.set_ylabel('absorbance');
```



Briefly describe the result.

The peaks shown here were made by eyeballing the means and the standard deviation, an then manually creating the gaussian vector myself and putting it together as one matrix, then I plotted each column together, where each column represents a Gaussian curve, and I made 3 of them as shown. These curves are at the same x values as the data and nearly the same width as I could get them, but I cannot get the heights of

Continue working on general linear regression.

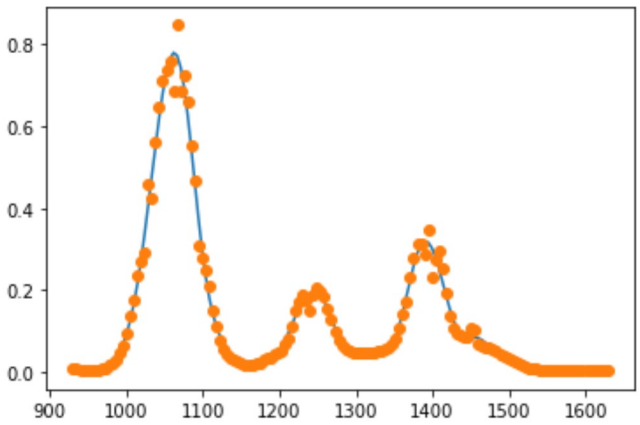
Now the second assumption is gone. You do not know how many peaks there are, or the widths of the peaks. However, you do know that they follow Gaussian distributions.

- Use your intuition and trial-and-error along to find a model that describes the data.
- Also plot the result along with the original data.

- This is not a spectroscopy class. There is no right answer to this question.

In [67]:
```
#since that first gaussian I made was really off from the data
#we will use the function I made for the gaussian function to make
#a better fit


m = 35 #this is the N number of Gaussians, arbitrarily guessed
#and checked to get tii this number 35
mygauss = gaussian_features(x_peak,m,25)
#need to get it again in Aw = b form, so create those components
A = np.dot(mygauss.T, mygauss)
b = np.dot(mygauss.T,y_peak)
w = np.linalg.solve(A,b)#solve the system
#now do the predictions
yhat = np.dot(mygauss,w)

#fig, ax = plt.subplots(figsize = (5, 3), dpi = 150)
fig, ax = plt.subplots()
ax.plot(x_peak, yhat)
ax.plot(x_peak, y_peak, 'o')
```

Out[67]:
```
[<matplotlib.lines.Line2D at 0x18d3313e160>]
```



3. Non-linear Regression

Write a loss function.

You want to solve the same problem above using non-linear regression to find the optimal positions and widths of the peaks.

The inputs of the loss function should be:

- a parameter vector $\vec{\lambda} = [\vec{w}, \vec{\mu}, \vec{\sigma}]$
- an input vector x
- an output vector y
- a number of Gaussians n

The function should return a root-mean-squared error of the estimation.

In [68]:
```
import numpy as np
def gaussian_loss(lamda, x, y, m):
    yhat = np.zeros(len(y))
    for i in range(m):
        w = lamda[i] #making a lambda that contains all the parameters w,mu,sigma
```

```
        mu = lamda[m+i]
        sigma = lamda[2*m+i]
        yhat = yhat + w*np.exp(-(x-mu)**2/(2*(sigma**2)))
    RMSE = np.sqrt(np.sum(((y-yhat)**2)/len(y)))
    return RMSE
lamda = np.array([10., 10., 10., 1080., 1300., 1500., 40., 40., 40.])
y = 10*np.exp(-(x-1080)**2/(2*(40**2)))
y += 10*np.exp(-(x-1300)**2/(2*(40**2)))
y += 10*np.exp(-(x-1500)**2/(2*(40**2)))
gloss = gaussian_loss(lamda,x,y,3)
```

Use `autograd` to compute the derivative of the loss function.

Find the derivative of the loss function when all of the parameters are 1.

In [71]:
```
! pip install autograd
import autograd.numpy as np
from autograd import grad
#need to make a lamda later so this function can be used


def lossfunc(lamda, x = x_peak, y = y_peak, m = 3):
    return gaussian_loss(lamda,x,y,m)
lamda = np.array([10., 10., 10., 1000., 1250., 1500., 30., 30., 30.])

diffg = grad(lossfunc)
print(lossfunc(lamda))
print(diffg(lamda))
```

```
Requirement already satisfied: autograd in c:\users\inara\anaconda3\lib\site-pac
kages (1.3)4.688091040865237

Requirement already satisfied: future>=0.15.2 in c:\users\inara\anaconda3\lib\si
te-packages (from autograd) (0.18.2)
Requirement already satisfied: numpy>=1.12 in c:\users\inara\anaconda3\lib\site-
packages (from autograd) (1.18.5)
[ 1.57005748e-01  1.58308313e-01  1.60464500e-01 -1.40645451e-03
  8.40456069e-05  2.05938443e-04  2.45406270e-02  2.63444151e-02
  2.65071027e-02]
```

Out[71]:
```
<function autograd.wrap_util.unary_to_nary.<locals>.nary_operator.<locals>.nary_
f(*args, **kwargs)>
```

Implement gradient descent method.

Write a function for an iteration of gradient descent that returns the optimal parameters.

The inputs are:

- a parameter vector $\vec{\lambda}$
- a function g
- a step size
- a tolerance

In [ ]:
```
lamda0 = np.array([10., 10., 10., 1000., 1250., 1500., 30., 30., 30.])
h = 0.2 #step size
tol = 0.01 #tolerance
N = 500 #iterations
def grad_descent(lamda, lossfunc, h, tol):
    for i in range(N):
        new_lamda = lamda - h*np.array(diffg(lamda))
```

```
    return new_lamda
lamdaFinal = grad_descent(lamda0,lossfunc,h,tol)
print('Initial Loss: {:4f}'.format(lossfunc(lamda0)))
print('Final Loss: {:4f}'.format(loss(lamdaFinal)))
print(lamdaFinal)
```

Find the optimal parameters.

Plot the result of non-linear regression along with the original data. Set the number of Gaussians as 5.

In [ ]:

```
n = 5
lamda2 = np.array([10., 10., 10., 1000., 1250., 1500., 30., 30., 30.])
lamFin = grad_descent(lamda2,lossfunc,h,tol)
diffg = grad(g)
```

Print the weights $\vec{w}$.

In [ ]:

Constrain the weights.

Modify the loss function to constrain the weights to be positive. You can write this in code, or you can write an analytical version of the loss function.

In [ ]: