Table of Contents

Numerical Methods - Assignment 2

Gaussian Features

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

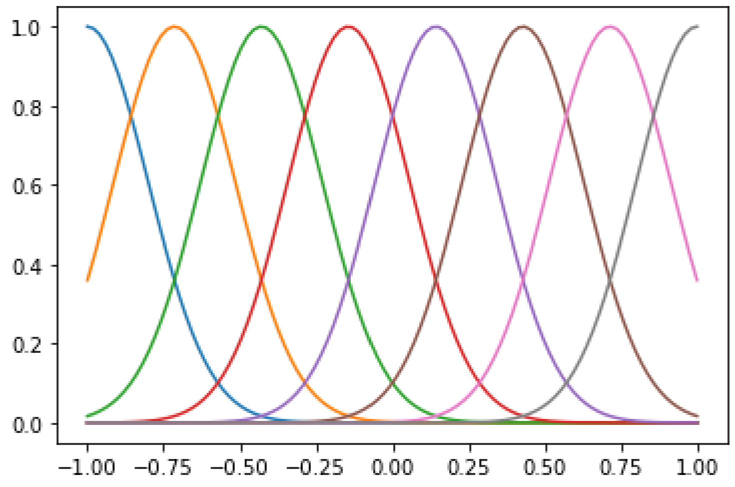
The input should be an vector x, a number of Gaussians N, and a fixed width σ.

In [6]:
```
import numpy as np

def gaussian_features(x, N, sigma):
    x = x.reshape(-1)
    xk_vec = np.linspace(min(x),max(x),N)
    features = []
    for xk in xk_vec:
        features.append(np.exp(-((x-xk)**2/(2*sigma**2))))
    return np.array(features).T
```

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

You can arbitrarily define the resolution of the range, but the resolution should be high enough that the plots look smooth.

In [71]:
```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1,1,100);

X_m = gaussian_features(x,8,0.2);

fig,ax = plt.subplots()

ax.plot(x,X_m);
#ax.plot(x,y)
```



General Linear Regression

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

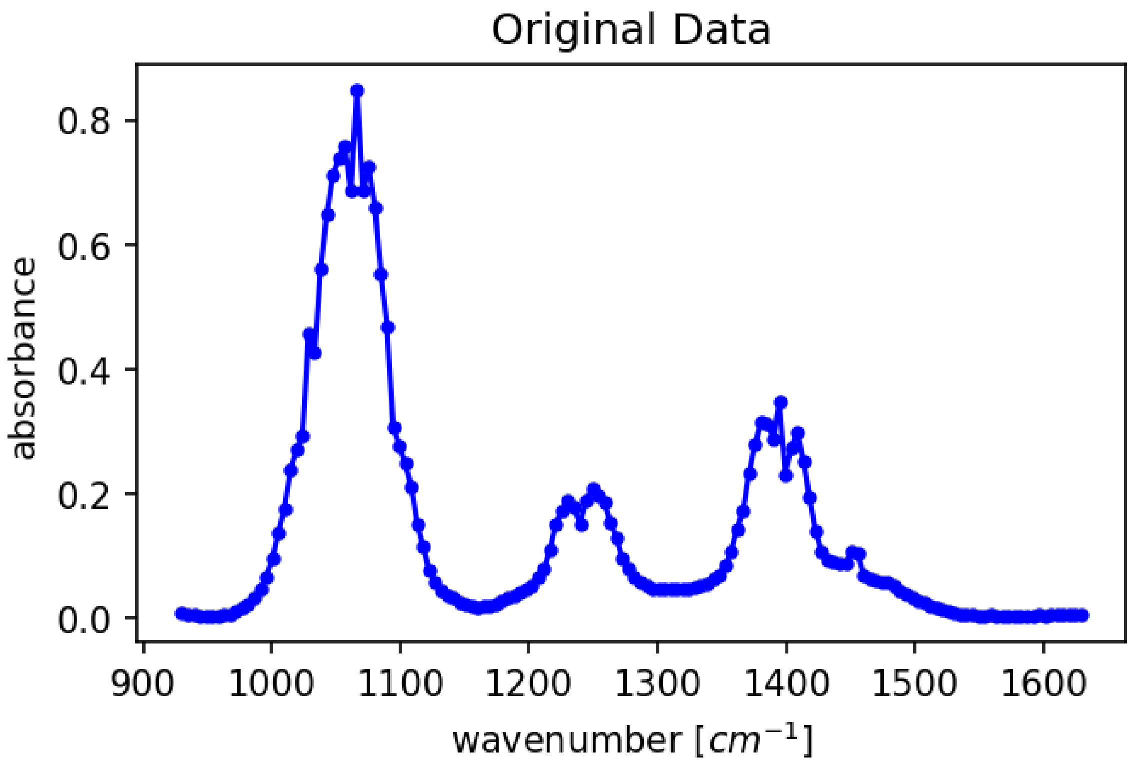
Plot the result of your regression model along with the original data. You can use visual inspection to determine the positions and widths of the peaks.

You may assume that:

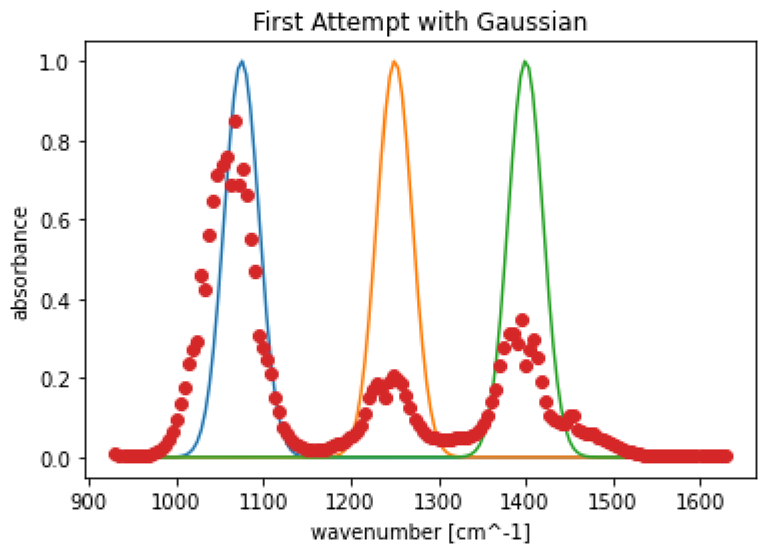
- The peaks follow a Gaussian distribution.
- There are 3 peaks of the **same width** in this region of the spectra below.

In [72]:

```
#PLOT ORIGINAL DATA BY EXTRACTING FROM ETHANOL IR DATASET

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]
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');

ax.set_title('Original Data');
```



In [73]:
```
#Try to fit data with Gaussian distributions
x_peak = x_peak.reshape(-1)
X_gauss = np.zeros((len(x_peak),3))
X_gauss[:,0] = np.exp(-(x_peak - 1075)**2/(2*(20**2)))
X_gauss[:,1] = np.exp(-(x_peak - 1250)**2/(2*(20**2)))
X_gauss[:,2] = np.exp(-(x_peak - 1400)**2/(2*(20**2)))

fig,ax = plt.subplots()
ax.plot(x_peak, X_gauss[:,0])
ax.plot(x_peak, X_gauss[:,1])
ax.plot(x_peak, X_gauss[:,2])
ax.plot(x_peak,y_peak, 'o')
ax.set_xlabel('wavenumber [cm^-1]')
ax.set_ylabel('absorbance');
ax.set_title('First Attempt with Gaussian');

#Optimize weights
A = X_gauss.T@X_gauss;
B = X_gauss.T@X_vdm;
w_lsr = np.linalg.solve(A,B);
yhat = X_gauss@w_lsr;

fig, ax = plt.subplots()
ax.plot(x_peak,y_peak, 'o')
ax.plot(x_peak, yhat_m, '--')
ax.set_xlabel('wavenumber [cm^-1]')
ax.set_ylabel('absorbance');
ax.set_title('Optimized Weights with Gaussian');
```



Briefly describe the result.

```
The initial approach was to create a matrix with three Gaussians that can take
the mean (mu = peak placement) and standard deviation (sigma = spacing of
peaks). The result was three bell curves with the generally correct
positioning along the x-axis. The issue with this result is that the heights
did not match up with those of the data. They were all at 100% absorbance when
it should fit the data better.

To improve this, the weights from the least-squares regression were optimized
using matrix multiplication, which was a better fit. It still had room for
improvement.
```

Continue improving the general linear regression model.

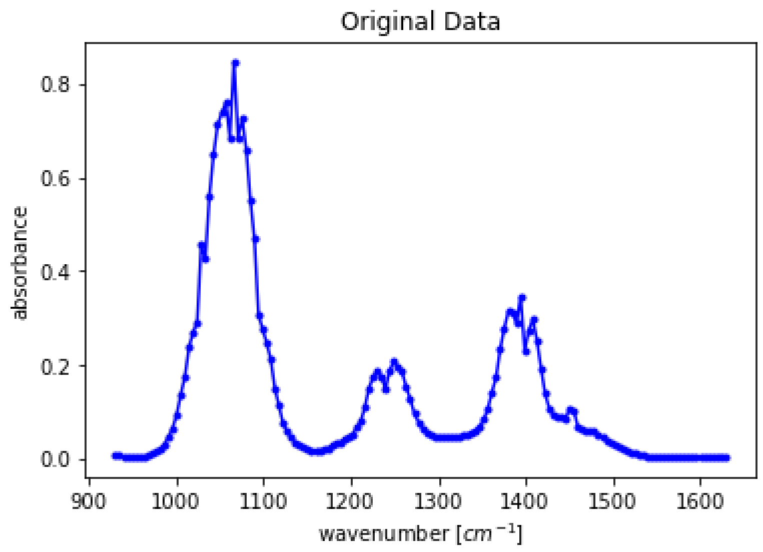
Now the second assumption is gone. You do not know how many peaks there are, or the widths of the peaks. However, you do know that they follow Gaussian distributions.

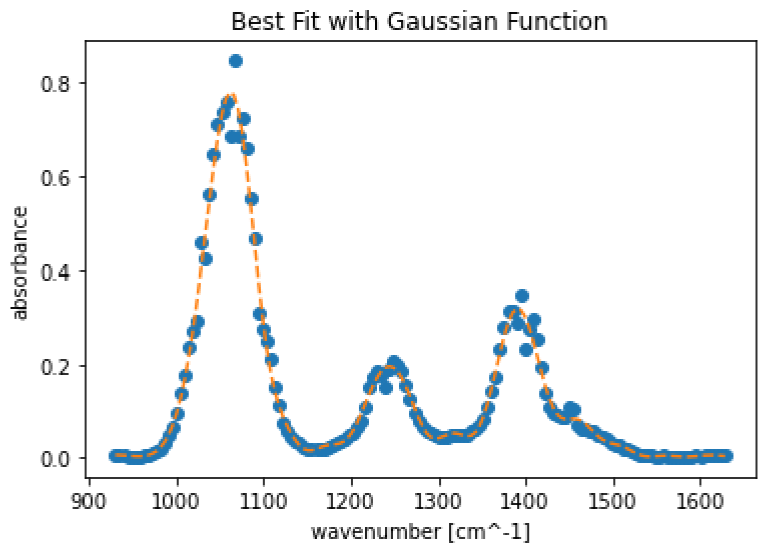
- Use your intuition and trial-and-error to find a model that describes the data.
- Plot the result along with the original data.
- This is not a spectroscopy class. There is no "right answer" to this question.

In [74]:

```
#PLOT ORIGINAL DATA BY EXTRACTING FROM ETHANOL IR DATASET

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]
y_peak = y_all[100:250]

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

ax.set_title('Original Data');

#Gaussian function
m = 30

X_gauss = gaussian_features(x_peak, m, 20)
b_m = np.dot(X_gauss.T, y_peak)
A_m = np.dot(X_gauss.T, X_gauss)
w_m = np.linalg.solve(A_m, b_m)

yhat_m = np.dot(X_gauss, w_m)
SSE_m = np.sum((y_peak - yhat_m)**2)

fig, ax = plt.subplots()
ax.plot(x_peak, y_peak, 'o')
ax.plot(x_peak, yhat_m, '--')
ax.set_xlabel('wavenumber [cm^-1]')
ax.set_ylabel('absorbance');

ax.set_title('Best Fit with Gaussian Function');

print('The final result was the best out of all the attempts. By using a func
```

```
The final result was the best out of all the attempts. By using a function
(gaussian_features) to add to the number of Gaussian peaks that are spaced
evenly, the best fit is achieved. I chose to use 30 Gaussians.
```





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 [75]:
```
import numpy as np

def gaussian_loss(lamda, x, y, n):
    yhat = np.zeros(len(y))
    for i in range(n):
        w_i = lamda[i]
        mu_i = lamda[n+i]
        sigma_i = lamda[2*n+i]
        yhat = yhat + w_i*np.exp(-(x-mu_i)**2/(2*sigma_i**2))
    squared_error = (yhat - y)**2 #for RMSE to forecast - observed values
    RMSE = np.sqrt(np.sum(squared_error)/len(y)) #divide by sample size
    return RMSE

#make Gaussian with IR spectra data from above to test out gaussian loss func

lamda = np.array([5., 5., 5., 1075., 1250., 1400., 30., 30., 30])

y = 5*np.exp(-(x - 1075)**2/(2*(30**2))) #Gaussian with w = 5, mu = 1075, sig
y += 5*np.exp(-(x - 1250)**2/(2*(30**2))) #Gaussian with w = 5, mu = 1250, si
y += 5*np.exp(-(x - 1400)**2/(2*(30**2))) #Gaussian with w = 5, mu = 1400, si

gLossFunc = gaussian_loss(lamda,x,y,3)
gLossFunc
```

Out[75]:
```
0.0
```

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

Find the derivative of the loss function when the parameter vector is [10., 10., 10., 1000., 1250., 1500., 30., 30., 30].

In [76]:
```
! pip install autograd

import autograd.numpy as np
from autograd import grad

def g(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])

diff_g = grad(g)
print(g(lamda))
print(diff_g(lamda))
diff_g
```

```
Requirement already satisfied: autograd in c:\users\margaret\anaconda3\lib
\site-packages (1.3)
Requirement already satisfied: future>=0.15.2 in c:\users\margaret\anaconda
3\lib\site-packages (from autograd) (0.18.2)
Requirement already satisfied: numpy>=1.12 in c:\users\margaret\anaconda3\l
ib\site-packages (from autograd) (1.18.5)
4.6880910408652365
[ 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[76]:
```
<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 [120]:
```
lamda1 = np.array([10., 10., 10., 1000., 1250., 1500., 30., 30., 30])
# g is the loss function
h = 0.2 #step size
tol = 0.01 #tolerance
N = 100 #iterations

def grad_descent(lamda, g, h, tol):
    #new_lamda = 0
    check = []
    #while sum(check) > tol:
        #new_lamda = lamda - h*np.array(diff_g(lamda)
    for i in range(N):
        new_lamda = lamda - h*np.array(diff_g(lamda))
        check = (new_lamda - lamda)**2
        if np.less(sum(check),tol):
            break
    return new_lamda

lamdaFinal = grad_descent(lamda1,g,h,tol)

print('Initial Loss: {:.4f}'.format(g(lamda1)))
print('Final Loss: {:.4f}'.format(g(lamdaFinal)))
print(lamdaFinal)
print('Final lamda vector is slightly different from original lamda1')
```

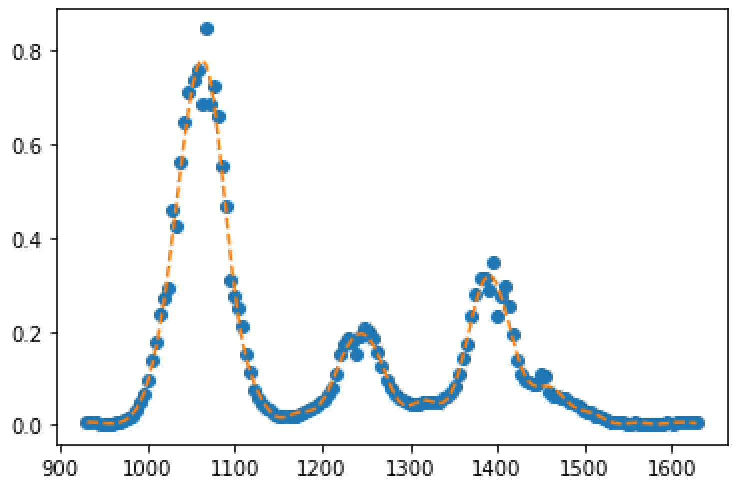
```
Initial Loss: 4.6881
Final Loss: 4.6726
[   9.96859885    9.96833834    9.9679071  1000.00028129 1249.99998319
 1499.99995881   29.99509187   29.99473112   29.99469858]
Final lamda vector is slightly different from original lamda1
```

Find the optimal parameters.

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

In [121]:
```
n = 5
lamda2 = np.array([10., 10., 10., 10., 10., 1000., 1200., 1250., 1350., 1500.

lamFin = grad_descent(lamda2,g,h,tol)

diff_g = grad(g)

fig, ax = plt.subplots()
ax.plot(x_peak, y_peak, 'o')
ax.plot(x_peak, yhat_m, '--')
```

Out[121]:
```
[<matplotlib.lines.Line2D at 0x19bc1a72f10>]
```



Print the weights $\vec{w}$.

In [122]:
```
print('Weights: {}'.format(w_lsr))
```

```
Weights: [[1.41421356e+00 1.52027957e+03 1.63486622e+06 1.75869741e+09
  1.89256155e+12 2.03731758e+15 2.19390152e+18 2.36333369e+21
  2.54672664e+24 2.74529380e+27 2.96035904e+30 3.19336713e+33
  3.44589523e+36 3.71966552e+39 4.01655907e+42 4.33863109e+45
  4.68812774e+48 5.06750452e+51 5.47944657e+54 5.92689103e+57
  6.41305156e+60 6.94144545e+63 7.51592335e+66 8.14070215e+69
  8.82040109e+72 9.56008164e+75 1.03652914e+79 1.12421127e+82
  1.21972158e+85 1.32379179e+88]
 [1.41421245e+00 1.76776540e+03 2.21027220e+06 2.76425414e+09
  3.45796955e+12 4.32688411e+15 5.41552019e+18 6.77978350e+21
  8.48989108e+24 1.06340567e+28 1.33231310e+31 1.66964452e+34
  2.09291715e+37 2.62415978e+40 3.29108156e+43 4.12854536e+46
  5.18042556e+49 6.50195094e+52 8.16266038e+55 1.02501322e+59
  1.28746912e+62 1.61753507e+65 2.03273155e+68 2.55514590e+71
  3.21262950e+74 4.04031065e+77 5.08250668e+80 6.39514143e+83
  8.04880216e+86 1.01326062e+90]
 [1.41421246e+00 1.97989761e+03 2.77242254e+06 3.88297573e+09
  5.43949326e+12 7.62150373e+15 1.06809847e+19 1.49716708e+22
  2.09902467e+25 2.94342558e+28 4.12835242e+31 5.79146725e+34
  8.12621909e+37 1.14045060e+41 1.60085651e+44 2.24758567e+47
  3.15622513e+50 4.43309979e+53 6.22780211e+56 8.75084137e+59
  1.22985093e+63 1.72879200e+66 2.43063956e+69 3.41810878e+72
  4.80771420e+75 6.76361377e+78 9.51713646e+81 1.33943327e+85
  1.88548509e+88 2.65468079e+91]]
```

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 [ ]:
```
def g_simwidth(lamda, x = x_peak, y=y_peak, N=2):
    return gaussian_loss(lamda,x,y,N) + (lamda[-2] - lamda[-1])**2

result = minimize(g_simwidth, guess, method = 'BFGS')
fitted = result.x
```