**PROJECT #7 – Expectation Maximization**

**Divyanshu Sahay  
USC ID: 6364938199  
(dsahay@usc.edu)**

Important Note:

Tool Used: **Python 3.7.4**

1. **Random number generator for multivariate Gaussian distribution**

**Problem Statement:** A random number generator has to be created for random vector   
**X = [X1,X2,X3].** It’s given that,

,

**Understanding the Problem:** In simple terms this problem requires us to generate number, **Xi**, such that,

***X1 = a11 \* Z1 + a12 \* Z2 + a13\*Z3 + u1***

***X2 = a21 \* Z1 + a23 \* Z2 + a23\*Z3 + u2***

***X3 = a31 \* Z1 + a32 \* Z2 + a33\*Z3  + u3***

Here, ***Zi***isstandard normal and ***ui***  is the expectation of the random variable. In matrix notation we have:

***X = A \* Z* + *U***

Where, A\*A’ = Σ. We need to use a very important method which will help us solve this problem, namely Choleski Decompoaition. This method will help us find a lower triangular matrix such that this condition is satisfied and then we can use the above equation to find the random numbers. This problem also required us generate standard normal variable, this can be done by using Box – Muller method which we studied in the earlier projects.

**Program Written**:

import numpy as np

import math

Z\_list = []

Y = [[1],[2],[3]]

print(Y)

print('---CHOLESKY DECOMPOSITION---')

import numpy as np

a = [[3, -1, 1], [-1, 5, 3], [1, 3, 4]]

print("Original array:")

print(a)

A = np.linalg.cholesky(a)

print("Lower-trianglular A in the Cholesky decomposition of the said array:")

print(A)

for i in range(3):

    u1 = np.random.rand()

    u2 = np.random.rand()

    Z = (math.sqrt(-2\*math.log(u1)))\*math.cos(2\*math.pi\*u2)

    z = math.sqrt(1)\*Z + 0

    Z\_list.append([z])

X = np.dot(A,Z\_list)

result = [[X[i][j] + Y[i][j]  for j in range

(len(X[0]))] for i in range(len(X))]

c = 1

for i in result:

    print('x'+ str(c)+':' + str(i))

    c +=1

**Results and Observation:**

---CHOLESKY DECOMPOSITION---

Original array:

[[3, -1, 1], [-1, 5, 3], [1, 3, 4]]

Lower-trianglular A in the Cholesky decomposition of the said array:

[[ 1.73205081  0.          0.        ]

 [-0.57735027  2.1602469   0.        ]

 [ 0.57735027  1.5430335   1.13389342]]

x1:[-2.6418884241012255]

x2:[3.3105166718716763]

x3:[2.4180275291458786]

This was relatively easy method to generate random number from a multivariate Gaussian Distribution.

1. **Random number generator for Mixture distribution**

**Problem Statement:** This problem requires us to generate the random numbers from the following mixture of Gaussian distribution:

***f(X) = 0.4N(-1,1) + 0.6N(1,1)***

**Requirements:**

1. Generate Histogram
2. Overlay Theoretical PDF

**Understanding the Problem:** This equation reflects that the two normal distribution N (-1,1) and N (1,2) contributes to the mixture with weight 0.4 and 0.6 respectively. This problem was explained in detail on Piazza. The code that is written for this problem is a direct adaptation of the code made available by the Professor. Basically, it is just about choosing one of the normal distribution based on randomly (uniformly) generating probability to select one.

**Program Written**:

import numpy as np

import matplotlib.pyplot as plt

N = 1000

mu1, sigma1 = -1, 1 # mean and standard deviation of the two models

mu2, sigma2 = 1, 1

p = 0.4

s = np.zeros(N)

for i in range(N):

    if np.random.rand() < p:

        s[i] = np.random.normal(mu1, sigma1)

    else:

        s[i] = np.random.normal(mu2, sigma2)

count, bins, ignored = plt.hist(s, 20, density=True)

plt.plot(bins, p\*(1/(sigma1 \* np.sqrt(2 \* np.pi)) \*

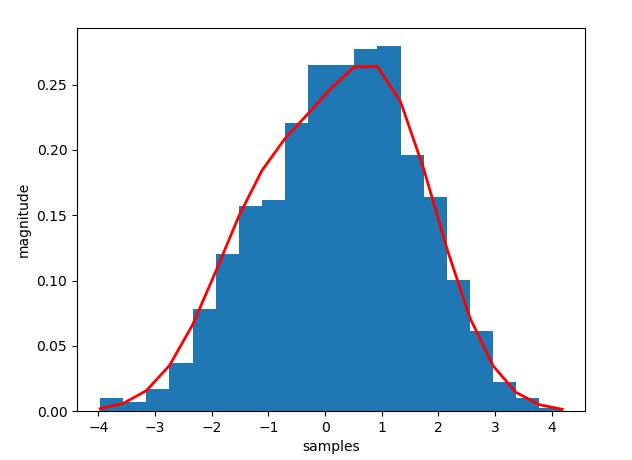
                np.exp( - (bins - mu1)\*\*2 / (2 \* sigma1\*\*2) )) + (1-p)\*(1/(sigma2 \* np.sqrt(2 \* np.pi)) \*

                np.exp( - (bins - mu2)\*\*2 / (2 \* sigma2\*\*2)) ),

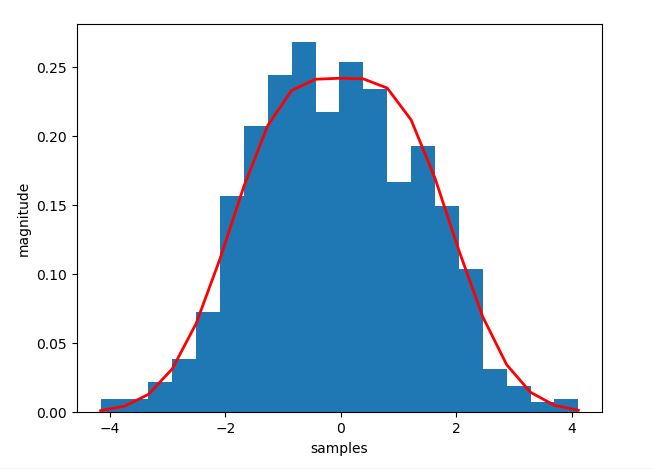
         linewidth=2, color='r')

plt.show()

**Results and Observations:**

**  
Fig.1: Histogram overlayed with theoretical pdf.**

As can be seen in the plot the bell is skewed towards a particular side and is not our generic well shaped bell curve. This is in accordance with our prediction. The shape is because of the uneven probability distribution of the normal distribution.

  
**Fig.2. The plot with equal probabilities.**

1. **Expectation Maximization**

**Problem Statement:** This problem requires to implement a 2-D random number generator for a Gaussian Mixture Model with 2 sub-populations. Furthermore, it is required to use Expectation Maximization Algorithm to estimate the pdf parameters. Finally, the cases of spherical Vs ellipsoidal covariance and close Vs well separated subpopulation has to be examined.

**Understanding the Problem:** This problem needs a detailed analysis of the Expectation-Maximization algorithm. The expectation maximization can be used for finding variables that are not directly observable but can be inferred from the values of the other observed variables, so called – **Latent Variable**. It is based on un-supervised clustering of algorithm. It has similarity to K-means (which will be used in our program to make initial guess)

It basically involves two steps (in very simple words):

**Expectation Step:** Using the available reading (initial reading in our case) we make possible expectation/prediction of the missing data.

**Maximization Step**: Update parameters from the data generated in the expectation step.

In our case we have to work with 2D Gaussian mixture model**.** Using K-means algorithm we can have the starting point (I have used the data provided by Professor that that I could cross verify). The program written has comments which explains the working clearly.

**Program Written:**

mport numpy as np

import matplotlib.pyplot as plt

import random

from sklearn.cluster import KMeans

from scipy.stats import multivariate\_normal

P\_expect\_list\_D1 =[]

P\_expect\_list\_D2 =[]

mu1 = np.array([0,4])

print('covariance matrix for distriution 1')

sigma1 = np.array([[3, 0],[0, 0.5]])

print(sigma1)

mu2 = np.array([-2,0])

print('covariance matrix for distriution 1')

sigma2 = np.array([[1, 0], [0, 2]])

print(sigma2)

np.random.seed(1) #For reproducibility

r1 = np.random.multivariate\_normal(mu1, sigma1,300)

r2 = np.random.multivariate\_normal(mu2, sigma2,300)

X = np.vstack((r1,r2)) #Cascade data points

# print("Shape of array:\n", np.shape(X))

print(len(X))

np.random.shuffle(X)

kmeans = KMeans(n\_clusters=2, random\_state=0).fit(X)

print('-- KMEANS ESTIMATE OF MEANS---')

mu\_init\_1 = kmeans.cluster\_centers\_[0,:]    #for first normal distribution

mu\_init\_2 = kmeans.cluster\_centers\_[1,:]    #for second normal distribution

print('INITIAL PARAMETERS--')

print(mu\_init\_1)

print(mu\_init\_2)

#intitializing the covariance matrix for 1st normal distributiuon by Identity

print('---INITIAL SIGMA---')

sigma\_init\_1 = np.array([[1,0],[0,1]])

sigma\_init\_2 = np.array([[1,0],[0,1]])

print(sigma\_init\_1)

print(sigma\_init\_2)

weight\_1 = 0.5

weight\_2 = 0.5

P\_expect\_D1\_list = []

P\_expect\_D2\_list = []

for j in range (5):

    P\_expect\_D1\_list = []

    P\_expect\_D2\_list = []

    #EXPECTATION STEP:

    for i in range(int(len(X))):

        elem1 = weight\_1\*multivariate\_normal.pdf(X[i,:], mu\_init\_1 ,sigma\_init\_1)

        elem2 = weight\_2\*multivariate\_normal.pdf(X[i,:], mu\_init\_2 ,sigma\_init\_2)

        P\_expect\_D1 = elem1/(elem1 + elem2)

        P\_expect\_D1\_list.append(P\_expect\_D1)

        P\_expect\_D2 = elem2/(elem1 + elem2)

        P\_expect\_D2\_list.append(P\_expect\_D2)

    mu\_init\_1 = np.array([0,0])

    mu\_init\_2 = np.array([0,0])

    sigma\_init\_1 =  np.array([[0,0],[0,0]])

    sigma\_init\_2 =  np.array([[0,0],[0,0]])

    #MAXIMIZATION STEP:

    for k in range(int(len(X))):

        value1\_mu = (X[k,:]\*P\_expect\_D1\_list[k])/sum(P\_expect\_D1\_list)

        mu\_init\_1 = mu\_init\_1 + value1\_mu #UPDATING MEAN FOR 1ST DISTRIBUTION

        value2\_mu = (X[k,:]\*P\_expect\_D2\_list[k])/sum(P\_expect\_D2\_list)

        mu\_init\_2 = mu\_init\_2 + value2\_mu #UPDATING MEAN FOR 1ST DISTRIBUTION

    for k in range(len(X)):

        val\_1 = np.outer((X[k,:]-mu\_init\_1),(X[k,:]-mu\_init\_1))

        val\_2 = np.outer((X[k,:]-mu\_init\_2),(X[k,:]-mu\_init\_2))

        value1\_sigma = P\_expect\_D1\_list[k]\*val\_1/sum(P\_expect\_D1\_list)

        sigma\_init\_1 = sigma\_init\_1 + value1\_sigma #UPDATING SIGMA FOR 1ST DISTRIBUTION

        value2\_sigma = P\_expect\_D2\_list[k]\*val\_2/sum(P\_expect\_D2\_list)

        sigma\_init\_2 = sigma\_init\_2 + value2\_sigma #UPDATING SIGMA FOR 2ND DISTRIBUTION

    weight\_1 = sum(P\_expect\_D1\_list)/(len(X)) #UPDATING WEIGHTS

    weight\_2 = sum(P\_expect\_D2\_list)/(len(X))

print('mu1')

print(mu\_init\_1)

print('mu2')

print(mu\_init\_2)

print('sigma1')

print(sigma\_init\_1)

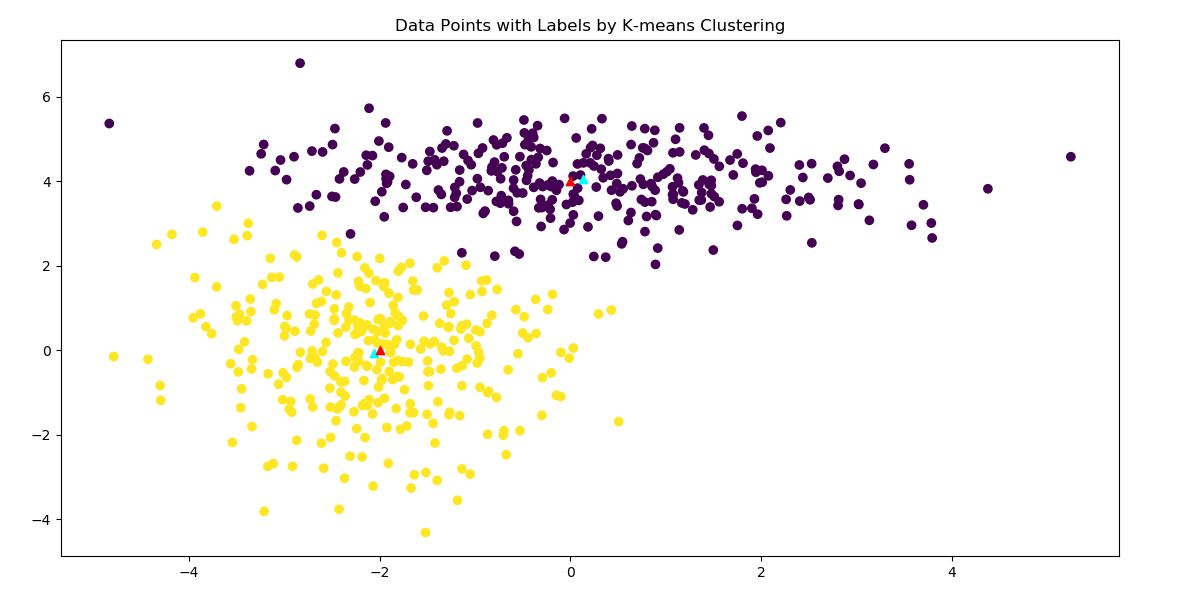
print('sigma2')

print(sigma\_init\_2)

**Results and Observations:**

For verification purposes I ran the code first for the data set provided by professor and obtained identical results:

The clusters looked like this for this data:

  
**Fig.3. Cluster for the data given in notes.**

**Output Screen:**

mean for 1st dist.

[0 4]

covariance matrix for distriution 1

[[3.  0. ]

 [0.  0.5]]

mean for 2nd dist.

[-2  0]

covariance matrix for distriution 1

[[1 0]

 [0 2]]

-- KMEANS ESTIMATE OF MEANS---

INITIAL PARAMETERS--

[0.1362448  4.04888328]

[-2.05968925 -0.05726182]

---INITIAL SIGMA---

[[1 0]

 [0 1]]

[[1 0]

 [0 1]]

mu1

[0.1644242  4.07399681]

mu2

[-2.03278266  0.01963216]

sigma1

[[ 2.63591399 -0.20979163]

 [-0.20979163  0.50561292]]

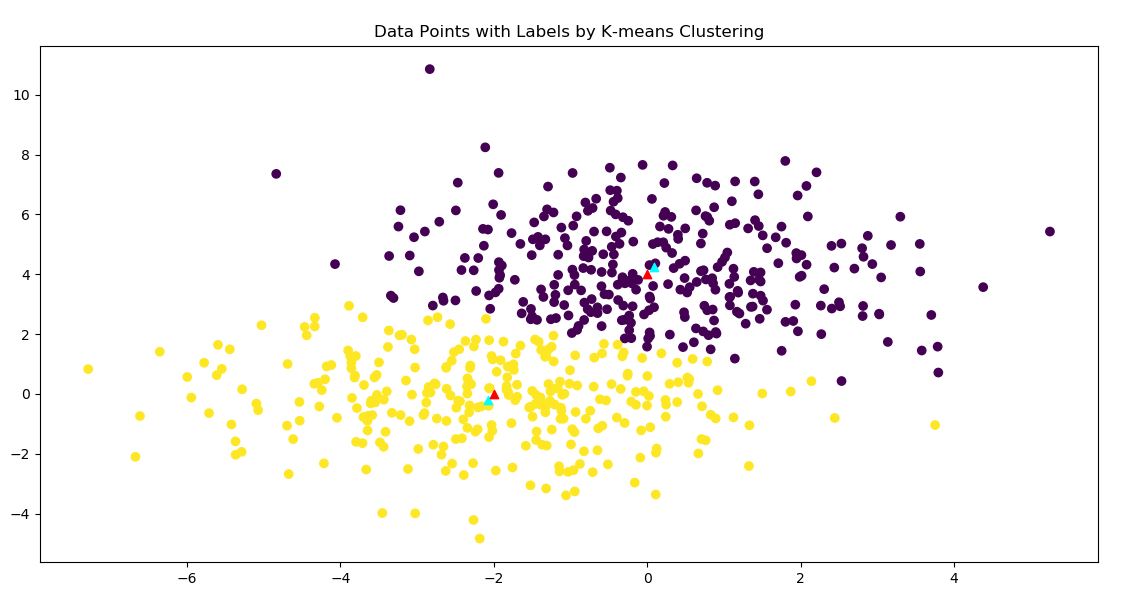
sigma2

[[ 0.91629201 -0.1003309 ]

 [-0.1003309   2.145652  ]]

The total number of Iteration Carried for this convergence were 3.

**Spherical Clustering:**

**  
Fig.4. Spherical Clustering**

mean for 1st dist.

[0 4]

covariance matrix for distriution 1

[[3 0]

 [0 3]]

mean for 2nd dist.

[-2  0]

covariance matrix for distriution 1

[[3 0]

 [0 3]]

-- KMEANS ESTIMATE OF MEANS---

[-2.07843073 -0.21393897]

INITIAL PARAMETERS--

[0.0831911  4.23028686]

[-2.07843073 -0.21393897]

---INITIAL SIGMA---

[[1 0]

 [0 1]]

[[1 0]

 [0 1]]

mu1

[0.0677939  4.12632036]

mu2

[-2.07325236 -0.12676545]

sigma1

[[ 2.71312008 -0.33745399]

 [-0.33745399  2.96137818]]

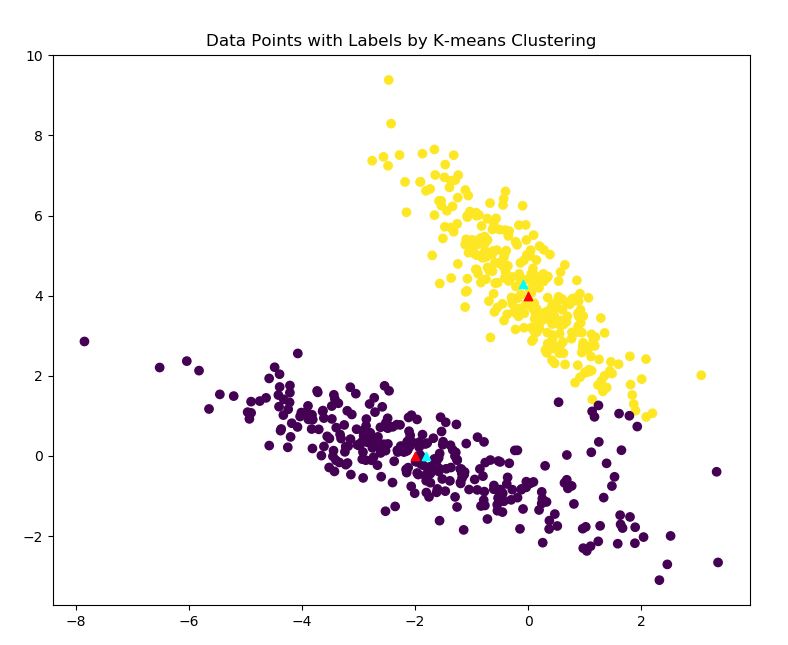
sigma2

[[ 3.18693099 -0.14196351]

 [-0.14196351  2.28626472]]

The number of iterations remained same.

**Ellipsoidal:**

  
**Fig.6. Ellipsoidal Clustering**

mean for 1st dist.

[0 4]

covariance matrix for distriution 1

[[ 1.  -1.5]

 [-1.5  3. ]]

mean for 2nd dist.

[-2  0]

covariance matrix for distriution 1

[[ 3.  -1.5]

 [-1.5  1.1]]

-- KMEANS ESTIMATE OF MEANS---

[-0.09701941  4.29146717]

INITIAL PARAMETERS--

[-1.80555372 -0.0044133 ]

[-0.09701941  4.29146717]

---INITIAL SIGMA---

[[1 0]

 [0 1]]

[[1 0]

 [0 1]]

mu1

[-1.92958228 -0.02896028]

mu2

[-0.03538585  4.15382258]

sigma1

[[ 3.12292072 -1.52134384]

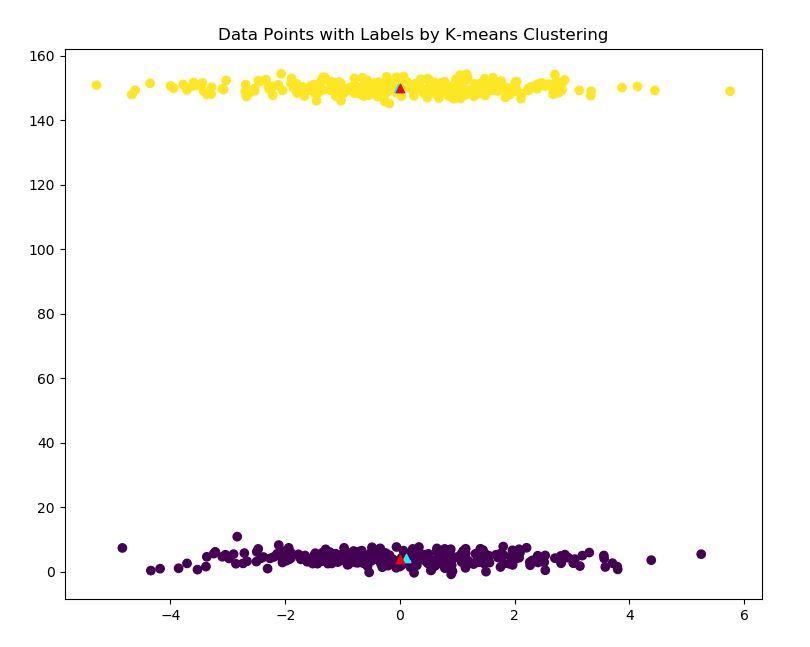
 [-1.52134384  1.05672055]]

sigma2

[[ 1.02563013 -1.41854091]

 [-1.41854091  2.61873371]]

Again the iterations remained same. Finally, I tried for well separated data. The convergence was effected. The number of iteration required to obtain very close resemblance in values increased.

  
**Fig. 7. Well Separated Data**

mean for 1st dist.

[0 4]

covariance matrix for distriution 1

[[3 0]

 [0 3]]

mean for 2nd dist.

[  0 150]

covariance matrix for distriution 1

[[3 0]

 [0 3]]

-- KMEANS ESTIMATE OF MEANS---

[-4.16800842e-02  1.49990156e+02]

INITIAL PARAMETERS--

[0.0968783  4.12989007]

[-4.16800842e-02  1.49990156e+02]

---INITIAL SIGMA---

[[1 0]

 [0 1]]

[[1 0]

 [0 1]]

mu1

[0.0968783  4.12989007]

mu2

[-4.16800842e-02  1.49990156e+02]

sigma1

[[ 2.85251777 -0.29415827]

 [-0.29415827  3.12310661]]

sigma2

[[ 3.03759399 -0.06795619]

 [-0.06795619  2.61207555]]

So, it took **100** iterations to reach this. Thus, it can be concluded that the type of covariance does not affect the convergence rate that effectively. However, if the data is separated enough, then the convergence might take some time.

1. **2D Scatter plot and Contour plot for GMM**

**Problem Statement:** In this problem a data file has been provided which contains the waiting times and the duration for eruption of a geyser. Following are the requirements:

1. 2-D scatter plot of the Data.
2. K-means clustering route has to be run for k=2.
3. GMM-EM algorithm has to be used to fit dataset to a GMM pdf.
4. A contour plot of the GMM pdf has to be drawn overlayed with scatterplot.

**Understanding the problem:** This problem is mostly an extension of the previous problem. However, the data set to be used has been provided in the data file. The GMM-EM algorithm helps to make a very good estimate of the missing data. The approach adopted can very well be understood in the comments that is provided.

**Program-Written:**

import random

import matplotlib.pyplot as plt

import numpy as np

from scipy.stats import sem

from sklearn.cluster import KMeans

from scipy.stats import multivariate\_normal

wait = []

dura = []

with open('D:\\SHIVAM\\USC\_STUDY\\EE511\\project7\\code\\q4\\DATA.dat') as f:

    wait = [float(line.split()[1]) for line in f]

with open('D:\\SHIVAM\\USC\_STUDY\\EE511\\project7\\code\\q4\\DATA.dat') as f:

    dura = [float(line.split()[2]) for line in f]

print(wait)

print(dura)

mu\_wait = sum(wait)/len(wait)

mu\_dura = sum(dura)/len(dura)

print(mu\_dura)

print(mu\_wait)

kmeans\_plot = np.array(list(zip(wait, dura)))

plt.figure(figsize=(9, 9))

plt.xlabel('DURATION')

plt.ylabel('WAITING TIME')

plt.title('2D-SCATTER PLOT OF THE DATA')

plt.scatter(wait,dura,c='blue')

plt.show()

#K-MEANS CLUSTERING

kmeans = KMeans(n\_clusters=2, random\_state=0).fit(kmeans\_plot)

# plt.figure(figsize=(9, 9))

plt.scatter(kmeans\_plot[:,0], kmeans\_plot[:,1], c=kmeans.labels\_.astype(float))

plt.scatter(kmeans.cluster\_centers\_[:,0],kmeans.cluster\_centers\_[:,1], marker='^', c='red')

plt.title('Data Points with Labels by K-means Clustering')

plt.xlabel('DURATION')

plt.ylabel('WAITING TIME')

plt.show()

weight\_1 = 0.5

weight\_2 = 0.5

# initializing the mu by the one obtained in from Kmeans

mu\_init\_1 = kmeans.cluster\_centers\_[0,:]    #for first normal distribution

mu\_init\_2 = kmeans.cluster\_centers\_[1,:]    #for second normal distribution

print('INITIAL PARAMETERS--')

print(mu\_init\_1)

print(mu\_init\_2)

#intitializing the covariance matrix for 1st normal distributiuon by Identity

print('---INITIAL SIGMA---')

sigma\_init\_1 = np.array([[1,0],[0,1]])

sigma\_init\_2 = np.array([[1,0],[0,1]])

P\_expect\_D1\_list = []

P\_expect\_D2\_list = []

for j in range (5):

    P\_expect\_D1\_list = []

    P\_expect\_D2\_list = []

    #EXPECTATION STEP:

    for i in range(int(len(kmeans\_plot))):

        elem1 = weight\_1\*multivariate\_normal.pdf(kmeans\_plot[i,:], mu\_init\_1 ,sigma\_init\_1)

        elem2 = weight\_2\*multivariate\_normal.pdf(kmeans\_plot[i,:], mu\_init\_2 ,sigma\_init\_2)

        P\_expect\_D1 = elem1/(elem1 + elem2)

        P\_expect\_D1\_list.append(P\_expect\_D1)

        P\_expect\_D2 = elem2/(elem1 + elem2)

        P\_expect\_D2\_list.append(P\_expect\_D2)

    mu\_init\_1 = np.array([0,0])

    mu\_init\_2 = np.array([0,0])

    sigma\_init\_1 =  np.array([[0,0],[0,0]])

    sigma\_init\_2 =  np.array([[0,0],[0,0]])

    #MAXIMIZATION STEP:

    for k in range(int(len(kmeans\_plot))):

        value1\_mu = (kmeans\_plot[k,:]\*P\_expect\_D1\_list[k])/sum(P\_expect\_D1\_list)

        mu\_init\_1 = mu\_init\_1 + value1\_mu #UPDATING MEAN FOR 1ST DISTRIBUTION

        value2\_mu = (kmeans\_plot[k,:]\*P\_expect\_D2\_list[k])/sum(P\_expect\_D2\_list)

        mu\_init\_2 = mu\_init\_2 + value2\_mu #UPDATING MEAN FOR 1ST DISTRIBUTION

    for k in range(len(kmeans\_plot)):

        val\_1 = np.outer((kmeans\_plot[k,:]-mu\_init\_1),(kmeans\_plot[k,:]-mu\_init\_1))

        val\_2 = np.outer((kmeans\_plot[k,:]-mu\_init\_2),(kmeans\_plot[k,:]-mu\_init\_2))

        value1\_sigma = P\_expect\_D1\_list[k]\*val\_1/sum(P\_expect\_D1\_list)

        sigma\_init\_1 = sigma\_init\_1 + value1\_sigma #UPDATING SIGMA FOR 1ST DISTRIBUTION

        value2\_sigma = P\_expect\_D2\_list[k]\*val\_2/sum(P\_expect\_D2\_list)

        sigma\_init\_2 = sigma\_init\_2 + value2\_sigma #UPDATING SIGMA FOR 2ND DISTRIBUTION

    weight\_1 = sum(P\_expect\_D1\_list)/(len(kmeans\_plot)) #UPDATING WEIGHTS

    weight\_2 = sum(P\_expect\_D2\_list)/(len(kmeans\_plot))

print('mu1')

print(mu\_init\_1)

print('mu2')

print(mu\_init\_2)

print(sigma\_init\_1)

print(sigma\_init\_2)

r1 = np.random.multivariate\_normal(mu\_init\_1, sigma\_init\_1,300) #GENERATING GAUSSIAN MIXTURE FOR THE ESTIMATION

r2 = np.random.multivariate\_normal(mu\_init\_2, sigma\_init\_2,300)

X = np.vstack((r1,r2))

np.random.shuffle(X)

plt.figure(figsize=(8,8))

x, y = np.mgrid[0:5:.01, 30:100:.01]

pos = np.empty(x.shape + (2,))

pos[:, :, 0] = x; pos[:, :, 1] = y

rv1 = multivariate\_normal(mu\_init\_1,sigma\_init\_1)

plt.contour(x, y, rv1.pdf(pos)) #CONTOUR PLOT

rv2 = multivariate\_normal(mu\_init\_2,sigma\_init\_2)

plt.contour(x, y, rv2.pdf(pos))

plt.scatter(X[:,0], X[:,1])

plt.scatter(mu\_init\_1[0],mu\_init\_1[1], marker='^', c='red')

plt.scatter(mu\_init\_2[0],mu\_init\_2[1], marker='^', c='red')

plt.xlabel('DURATION')

plt.ylabel('WAITING TIME')

plt.title('CONTOUR OVERLAYED WITH SCATTER')

plt.show()

#cluster prediction by GMM

plt.figure(figsize=(8,8))

plt.scatter(X[:,0], X[:,1])

plt.scatter(mu\_init\_1[0],mu\_init\_1[1], marker='^', c='red')

plt.scatter(mu\_init\_2[0],mu\_init\_2[1], marker='^', c='red')

plt.xlabel('DURATION')

plt.ylabel('WAITING TIME')

plt.title('CLUSTER PREDICTION BY GMM')

plt.show()

**Observation and Results:**

**Output Screen:**

INITIAL PARAMETERS USED FROM KMEANS—

[ 4.29793023 80.28488372]

[ 2.09433 54.75   ]

---INITIAL SIGMA ESTIMATED BY EM ALGO---

mu1

[ 4.28977936 79.96953203]

mu2

[ 2.0365214 54.4798593]

sigma1

[[ 0.16981955  0.93871918]

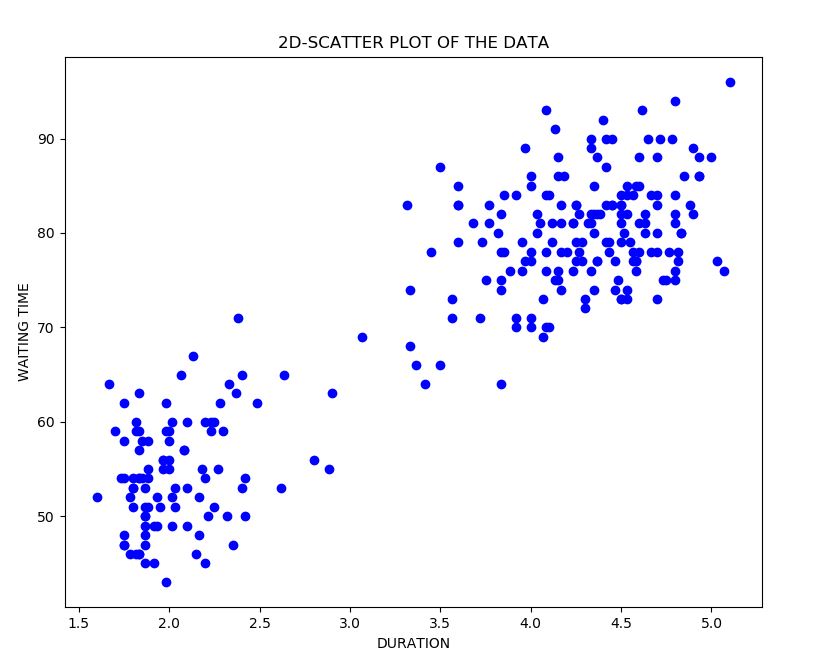
 [ 0.93871918 36.02498353]]

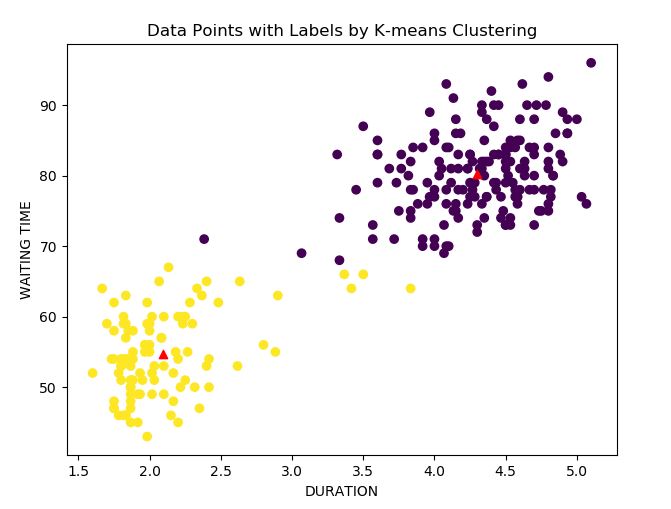
sigma2

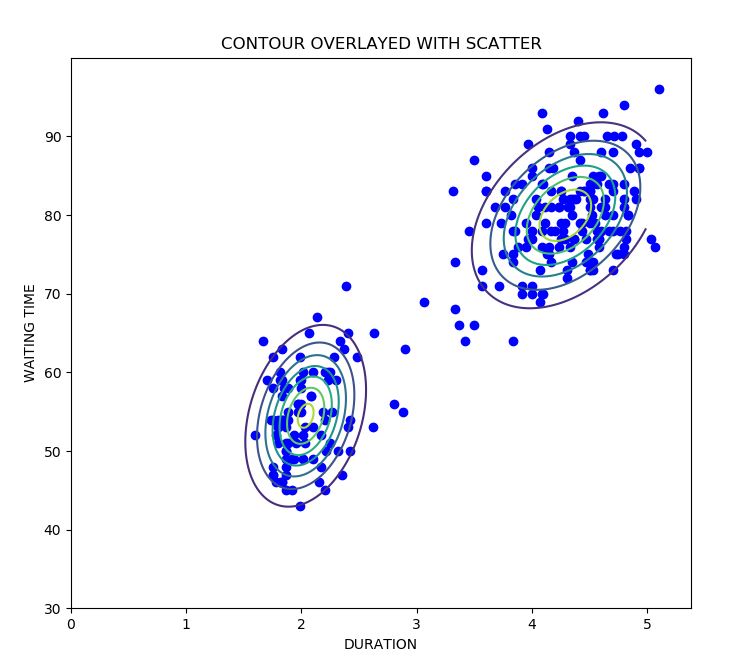
[[ 0.06927341  0.43627648]

 [ 0.43627648 33.70492759]]

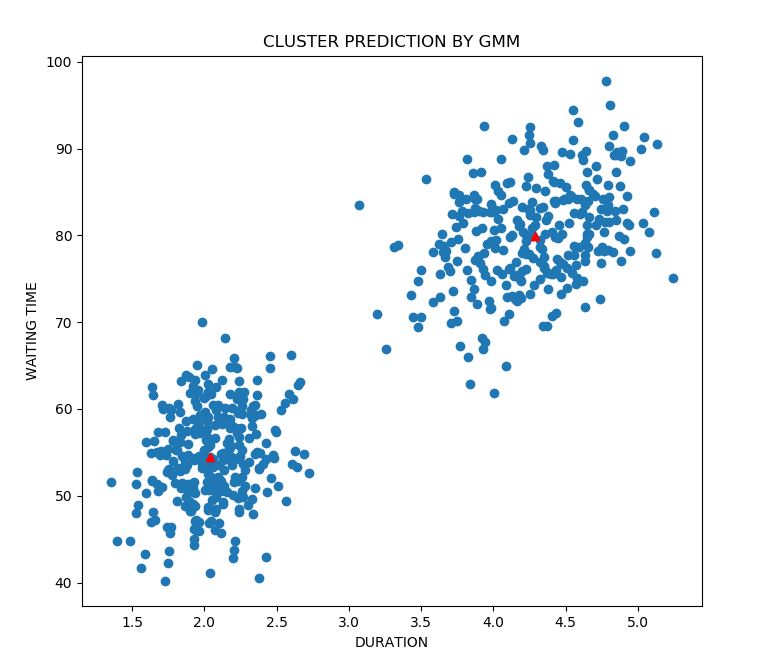
It can be seen in the output screen that EM algorithm does an excellent job of estimating the means.

  
**Fig.1. 2D-scatter plot of data**

  
**Fig 2: Clusters after running K-means clustering (k=2)**

**  
Fig 3. Contour Plots overlayed with data set.**

Lastly, we can certainly conclude that the GMM pdf helps to estimate the clusters of the data in an extremely efficient way. It can be seen in the Fig.4, the clusters predicted by the GMM pdf is an accurate estimate of the data.

  
**Fig.4. Clusters predicted by GMM algorithm**

Additionally, the 3D plot for the Gaussian distribution was also plotted using MATLAB 2019.Using following code (the mean and covariance were calculated from Python):

% mean

mu = [4.29175822 79.99160736;2.03902353 54.50872341];

% covaraiance

sigma = cat(3,[0.16742984 0.91046897;0.91046897 35.73559318],[0.07144018 0.46253011;0.46253011 33.95271124]);

% weight

p = [0.5,0.5];

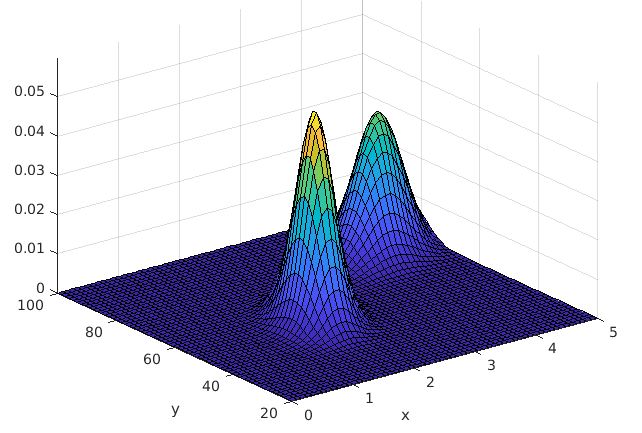
% build GMM

obj = gmdistribution(mu,sigma,p);

% view PDF surface

figure;

ezsurf(@(x,y)pdf(obj,[x y]),[0 5],[20 100])

  
**Fig. 5. 3D Plot of Gaussian Mixture. (x: duration, y: waiting time)**