EE511 Project7

Name: Yijing Jiang

Email: yijingji@usc.edu

USC ID: 5761477714

Question 1

Explane:

- Generate random vector X with known means and covariance:

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \qquad \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \qquad \varepsilon = \begin{bmatrix} cov_{11} & cov_{12} & cov_{13} \\ cov_{21} & cov_{22} & cov_{23} \\ cov_{31} & cov_{32} & cov_{33} \end{bmatrix} = \begin{bmatrix} 3 & -1 & 1 \\ -1 & 5 & 3 \\ 1 & 3 & 4 \end{bmatrix}$$

  - First, we generate random x1,x2 and x3 from three standard normal distribution, then the relationship between these independent x1,x2, x3 and X is:

$$\begin{cases} X_1 &= a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \mu_1 \\ X_2 &= a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \mu_2 \\ X_3 &= a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \mu_3 \end{cases}$$

    that is,

$$X = A^T x + \mu$$

  - here A is a decomposition of covariance matrix:

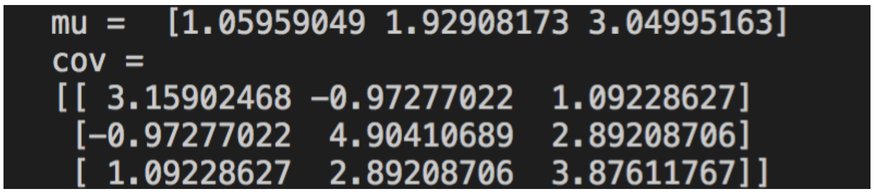
$$\varepsilon = A^T A$$

    we can get this A from covairance matrix by using *np.linalg.cholesky()*.

- To check if we create a correct generator, we use it to generate T samples. Then compute and compare the means and covariance matrix.

Result:

- Compute means and covariance matrix from T samples:



```
mu =  [1.05959049 1.92908173 3.04995163]
cov =
[[ 3.15902468 -0.97277022  1.09228627]
 [-0.97277022  4.90410689  2.89208706]
 [ 1.09228627  2.89208706  3.87611767]]
```

  - Our result is similar to the original means and covariance matrix.
  - Thus, we create a correct random vector generator.

Code:

```python
import numpy as np

def generate_X(mu, cov, T):
    A = np.linalg.cholesky(cov) #return L (lower triangle)
    z1 = np.random.normal(0,1,(T,))
    z2 = np.random.normal(0,1,(T,))
    z3 = np.random.normal(0,1,(T,))
    z = [z1,z2,z3]
    X = np.dot(A,z)+np.reshape(mu,(3,1))
    return X


T = 1000
mu = [1,2,3]
cov = [[3,-1,1],[-1,5,3],[1,3,4]]
print(np.linalg.cholesky(cov))
X = generate_X(mu,cov,T)
mu = np.mean(X,axis=-1)
cov = np.cov(X)
print("mu = ",mu)
print("cov = ")
print(cov)
```

Question 2

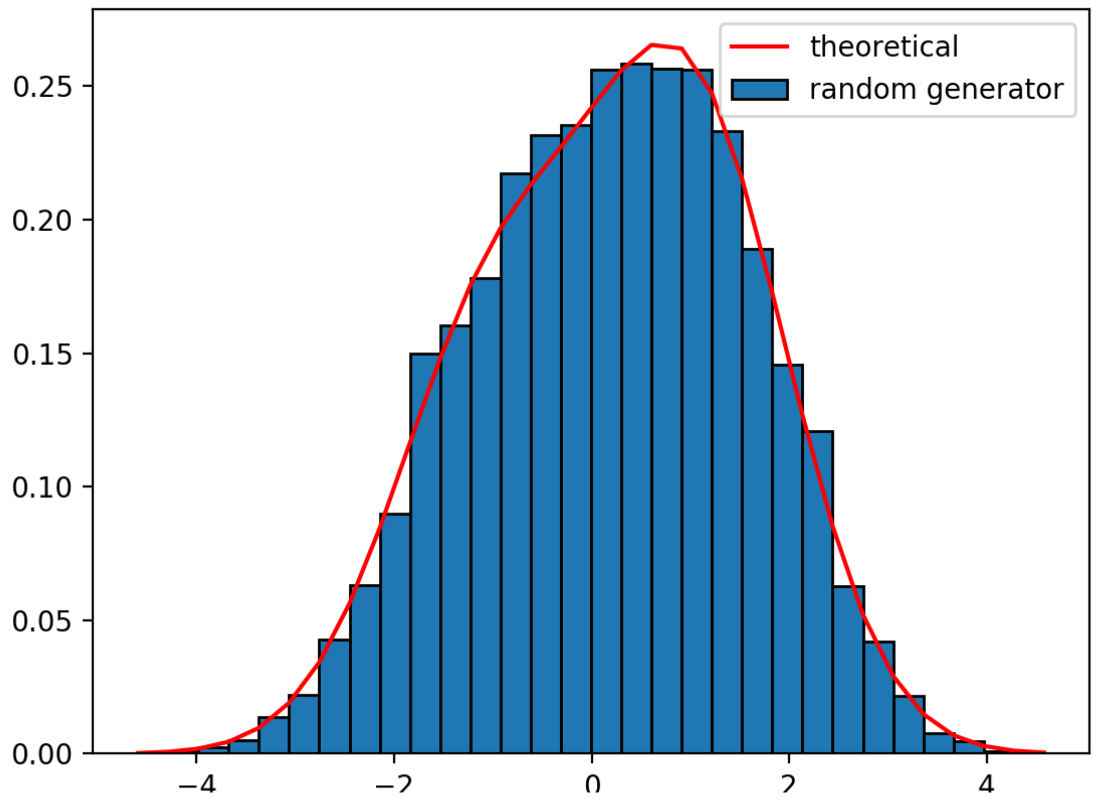
Explane:

- Generate random number with mixture distribution f(x) = 0.4N(-1,1)+0.6N(1,1):
  - generate random u from uniform distribuion U[0,1]
  - if u < 0.4, return a random number from N(-1,1)
  - else, return a random number from N(1,1)
- Generate a histogram:
  - repeat to generate T samples of random variable
  - plot the histogram of T samples with density
- Overlay the theoretical pdf:

$$f(x) = 0.4f_1(x) + 0.6f_2(x)$$

Results:

- **Plot of samples histogram and theoretical pdf:**
  - The histogram is similar to the theretical pdf plot.
  - We can see that one pick is at around x = 1. And at around x = -1, there is a slightly convex curve. This is corresponding to the two means of N(-1,1) and N(1,1).
  - Also the value of x = 1 is greater than x = -1 at around 3:2, that is corresponding to the coefficient of the combination of pdf.



Code:

```python
import numpy as np
import matplotlib.pyplot as plt

def generate_X(p,m1,s1,m2,s2,T):
    X = []
    for i in range (T):
        if np.random.rand()<p:
            X.append(np.random.normal(m1, s1))
        else:
            X.append(np.random.normal(m2, s2))
    return X


T = 10000
#random generator
p = 0.4
m1,s1 = -1,1
m2,s2 = 1,1
X = generate_X(p,m1,s1,m2,s2,T)

count, bins, ignored = plt.hist(X, 30, edgecolor='black',density=True, label='random generator')
#theoretical
X_th1 = p*(1/(s1*np.sqrt(2*np.pi))*np.exp(-(bins-m1)**2/(2*s1**2)))
X_th2 = (1-p)*(1/(s2*np.sqrt(2*np.pi))*np.exp(-(bins-m2)**2/(2*s2**2)))
plt.plot(bins,X_th1+X_th2, color='r', label='theoretical')
plt.legend()
plt.show()
```

Question 3

Explain:

- **Generate a GMM (Gaussian mixture model) with 2 subpopulations, and each subpopulation is a 2 dimensional random vector:**
  - Suppose 2 subpopulations have following 2 dimensional random vector. Generators of these two random vectors have the same idea asquestion 1.

$$\begin{cases} X_1 : (N(\mu_{11}, \sigma_{11}^2), N(\mu_{12}, \sigma_{12}^2)), \mu_1 = [\mu_{11}, \mu_{12}], \varepsilon_1 = \begin{bmatrix} \sigma_{11}^2 & 0 \\ 0 & \sigma_{12}^2 \end{bmatrix} \\ X_2 : (N(\mu_{21}, \sigma_{21}^2), N(\mu_{22}, \sigma_{22}^2)), \mu_2 = [\mu_{21}, \mu_{22}], \varepsilon_2 = \begin{bmatrix} \sigma_{21}^2 & 0 \\ 0 & \sigma_{22}^2 \end{bmatrix} \end{cases}$$

  - Combine two random vectors together with some coefficient, get the GMM. Generator of this has the same idea as question 2.

$$X = w_1 X_1 + w_2 X_2, w_1 + w_2 = 1$$

- **Generate 300 samples by using GMM.**
- **Estimate the distribution parameters from these 300 samples by using EM (expectation maximization) algorithem**:
  - estimate $w_1, \mu_1, \varepsilon_1, w_2, \mu_2, \varepsilon_2$ to maximize likelihood function:

$$L^{(t)} = \frac{1}{n} \sum_{i=1}^{n} log(\sum_{j=1}^{2} w_j^{(t)} f(x_i; \mu_j^{(t)}, \varepsilon_j^{(t)}))$$

where, f is the pdf of x:

$$f(x_i; \mu_j^{(t)}, \varepsilon_j^{(t)}) = \frac{1}{2\pi(\left|\varepsilon_j^{(t)}\right|)^{\frac{1}{2}}} e^{-\frac{1}{2}(x_i - \mu_j^{(t)})^T \varepsilon^{-1} (x_i - \mu_j^{(t)})}$$

  - initialize the parameters and compute the likelihood funcion
  - E-step: compute

$$r_{i,j}^{(t)} = \frac{w_j^{(t)} f(x_i; \mu_j^{(t)}, \varepsilon_j^{(t)})}{\sum_{l=1}^{2} w_l^{(t)} f(x_i; \mu_l^{(t)}, \varepsilon_l^{(t)})} \qquad n_j^{(t)} = \sum_{i=1}^{n} r_{i,j}^{(t)}$$

  - M-step: update parameters:

$$\begin{cases} w_j^{(t+1)} = \frac{n_j^{(t)}}{n} \\ \mu_j^{(t+1)} = \frac{1}{n_j^{(t)}} \sum_{i=1}^{n} r_{i,j}^{(t)} x_i \\ \varepsilon_j^{(t+1)} = \frac{1}{n_j^{(t)}} \sum_{i=1}^{n} r_{i,j}^{(t)} (x_i - \mu_j^{(t+1)})^T (x_i - \mu_j^{(t+1)}) \end{cases}$$

- Check if it is already converged, if not:

$$\left|L^{(t+1)} - L^{(t)}\right| > \delta, (\delta = 0.00001)$$

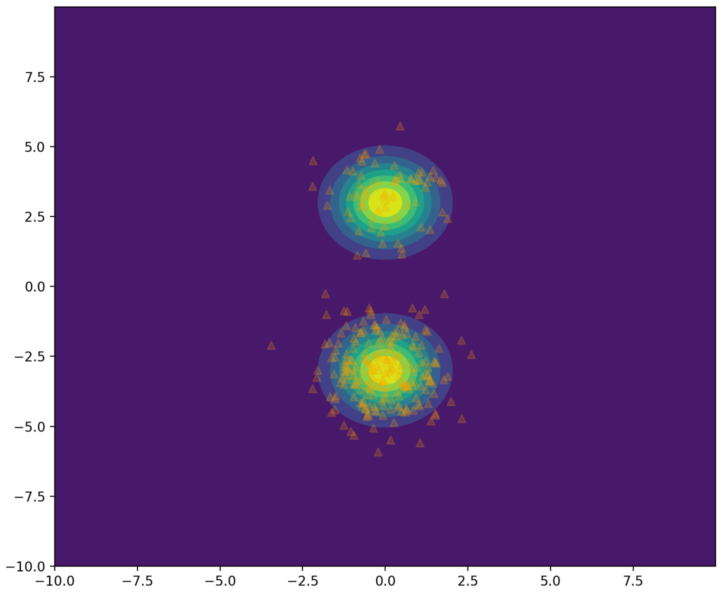
  record the iteration times and then go back to E-step.

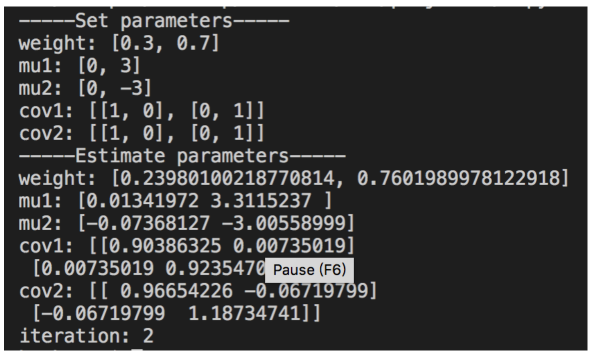
- For different GMM distributions, I choose following:

	mean(closed vs well-spread)	**cov(spherical vs ellipsoidal)**
spherical, well-spread	[0,3] [0,-3]	[[1,0],[0,1]] [[1,0],[0,1]]
spherical, closed	[0,2] [0,0]	[[1,0],[0,1]] [[1,0],[0,1]]
ellipsoidal, well-spread	[0,3] [0,-3]	[[4,0],[0,1]] [[1,0],[0,2]]
ellipsoidal, closed	[0,2] [0,0]	[[4,0],[0,1]] [[1,0],[0,2]]

Results:

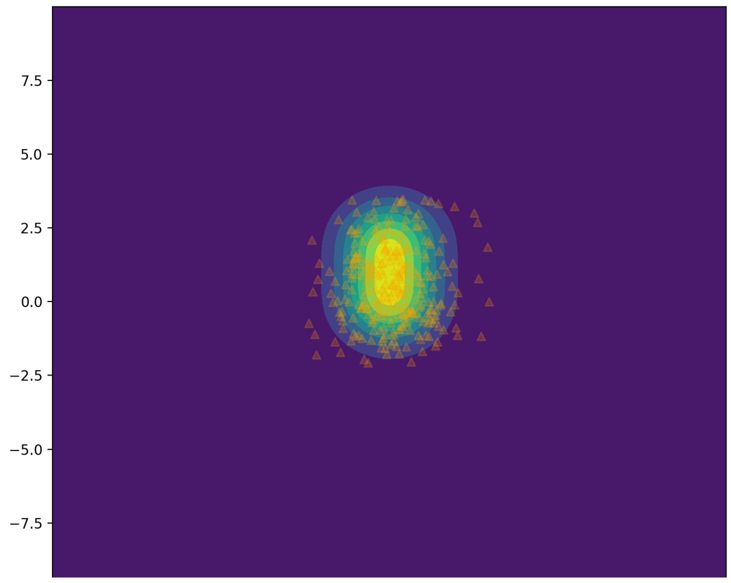
- **For spherical, well-spread GMM distribution:**
  - The image is a two separate circle.
  - iteration time is 2
  - estimation results are close to the origin parameters

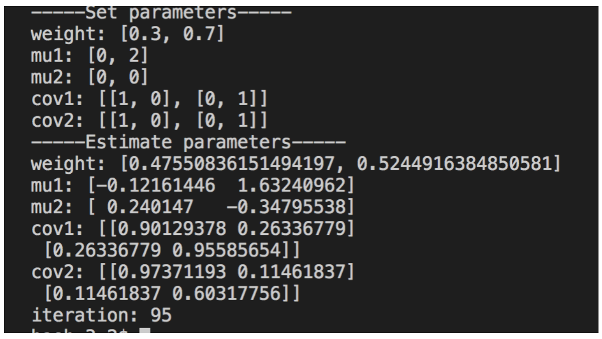




```
------Set parameters------
weight: [0.3, 0.7]
mu1: [0, 3]
mu2: [0, -3]
cov1: [[1, 0], [0, 1]]
cov2: [[1, 0], [0, 1]]
------Estimate parameters------
weight: [0.23980100218770814, 0.7601989978122918]
mu1: [0.01341972 3.3115237 ]
mu2: [-0.07368127 -3.00558999]
cov1: [[0.90386325 0.00735019]
 [0.00735019 0.9235470Pause (F6)
cov2: [[ 0.96654226 -0.06719799]
 [-0.06719799  1.18734741]]
iteration: 2
```

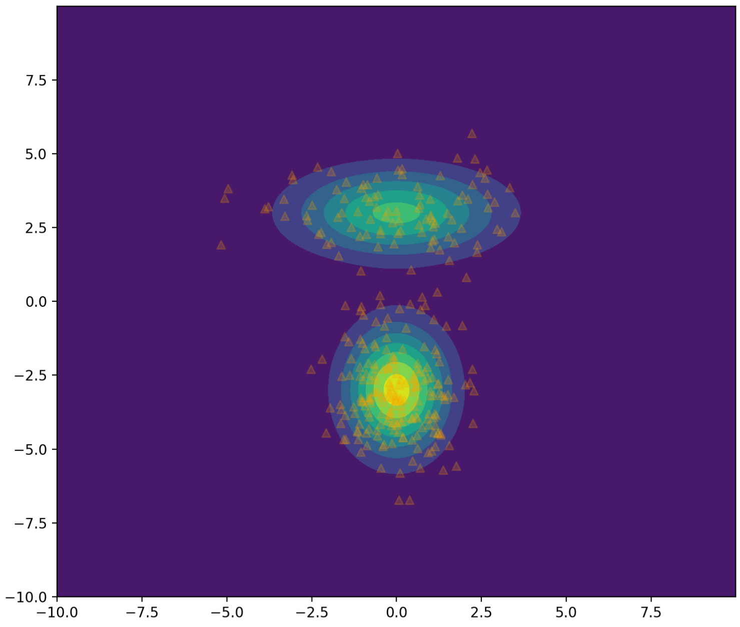
- **For spherical, closed distribution:**
  - The image is a combination of two circles.
  - iteration time is 95
  - estimation results are still not very similar to the origin parameters

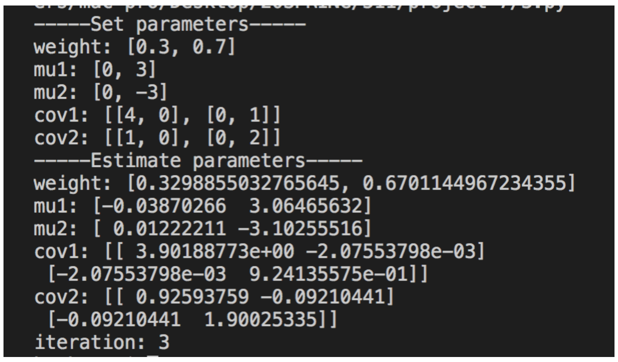




```
------Set parameters------
weight: [0.3, 0.7]
mu1: [0, 2]
mu2: [0, 0]
cov1: [[1, 0], [0, 1]]
cov2: [[1, 0], [0, 1]]
------Estimate parameters------
weight: [0.47550836151494197, 0.5244916384850581]
mu1: [-0.12161446  1.63240962]
mu2: [ 0.240147   -0.34795538]
cov1: [[0.90129378 0.26336779]
 [0.26336779 0.95585654]]
cov2: [[0.97371193 0.11461837]
 [0.11461837 0.60317756]]
iteration: 95
```

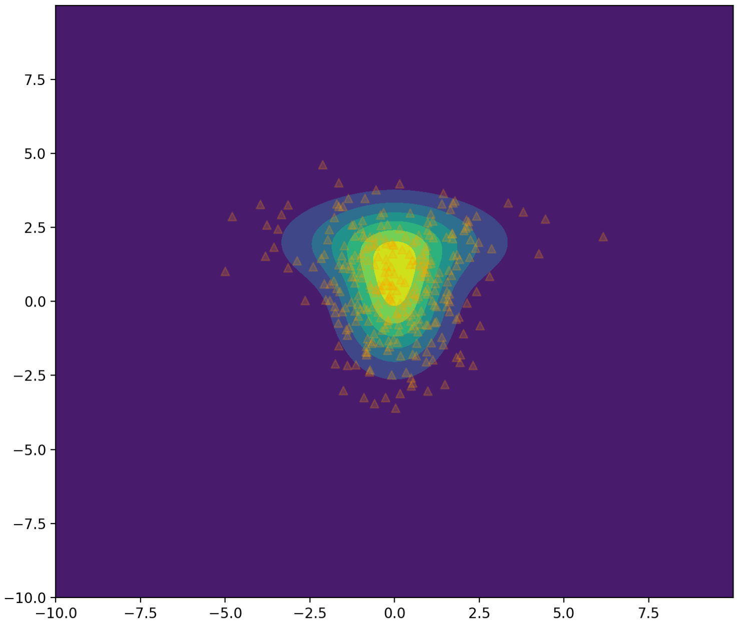
- **For ellipsoidal, well-spread GMM distribution:**
  - The image is a two separate ellipsoid.
  - iteration time is 3
  - estimation results are close to the origin parameters

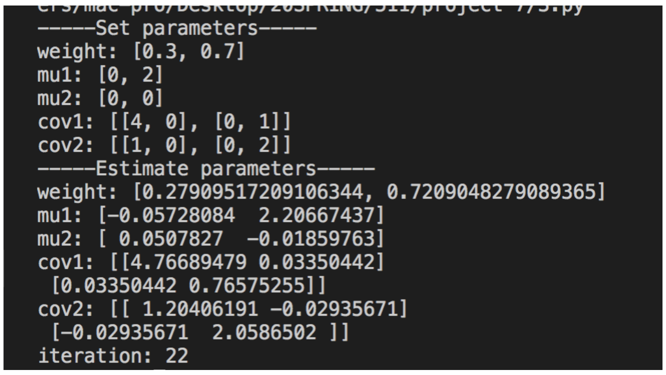




```
-----Set parameters-----
weight: [0.3, 0.7]
mu1: [0, 3]
mu2: [0, -3]
cov1: [[4, 0], [0, 1]]
cov2: [[1, 0], [0, 2]]
-----Estimate parameters-----
weight: [0.3298855032765645, 0.6701144967234355]
mu1: [-0.03870266  3.06465632]
mu2: [ 0.01222211 -3.10255516]
cov1: [[ 3.90188773e+00 -2.07553798e-03]
 [-2.07553798e-03  9.24135575e-01]]
cov2: [[ 0.92593759 -0.09210441]
 [-0.09210441  1.90025335]]
iteration: 3
```

- **For ellipsoidal, closed distribution:**
  - The image is a combination of two ellipsoids.
  - iteration time is 22
  - estimation results are still not so similar to the origin parameters





```
-----Set parameters-----
weight: [0.3, 0.7]
mu1: [0, 2]
mu2: [0, 0]
cov1: [[4, 0], [0, 1]]
cov2: [[1, 0], [0, 2]]
-----Estimate parameters-----
weight: [0.27909517209106344, 0.7209048279089365]
mu1: [-0.05728084  2.20667437]
mu2: [ 0.0507827  -0.01859763]
cov1: [[4.76689479 0.03350442]
 [0.03350442 0.76575255]]
cov2: [[ 1.20406191 -0.02935671]
 [-0.02935671  2.0586502 ]]
iteration: 22
```

- **Conclusion:**
  - **If two subpopulations are well-spread, then the spead of EM algorithm is quick, that is, we can get a more similar estimation of parameters within less iterations in EM algorithm.**
  - **But if two subpopulations are close, then it will be more difficult to estimate the parameters by using EM algorithm, that is, we will need more iterations to get the similar estimation. EM algorithem needs more calculations to seperate and estimate two close subpopulations.**

<u>Code:</u>

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal

def generate_X(p,mu1,cov1,mu2,cov2,T):
    X = np.zeros((T,2)) # points
    label = [] # class
    for i in range (T):
        if np.random.rand()<p:
            X[i][0] = np.random.normal(mu1[0],np.sqrt(cov1[0][0]))
            X[i][1] = np.random.normal(mu1[1],np.sqrt(cov1[1][1]))
            label.append(1)
        else:
            X[i][0] = np.random.normal(mu2[0],np.sqrt(cov2[0][0]))
            X[i][1] = np.random.normal(mu2[1],np.sqrt(cov2[1][1]))
            label.append(2)
    return X, np.array(label)

def f(X,mu,cov):
    exp = -np.inner(np.dot(X-mu,np.linalg.inv(cov)),X-mu)/2
    f = (1/(2*np.pi*np.sqrt(np.linalg.det(cov))))*np.exp(exp)
    return f

def cov(X,n,r,mu,T):
    cov = np.zeros((2,2))
    for i in range(T):
        cov = cov + r[i]*np.dot(np.reshape(X[i]-mu,(2,1)),np.reshape(X[i]-mu,(1,2)))
    cov = cov/n #(2,2)
    return cov

```

```python
def L(X,w,mu_1,cov_1,mu_2,cov_2,T):
    L = 0
    for i in range(T):
        fi = [f(X[i],mu_1,cov_1),f(X[i],mu_2,cov_2)]
        Li = np.log(np.inner(w,fi))
        L = L + Li
    L = L/T
    return L

def EM(X,label,T,d):
    #EM estimation with T samples
    r1 = -(label-2) #(300,) with class1 = 1
    r2 = label-1 #(300,) with class2 = 1
    n0_1 = np.sum(r1)
    n0_2 = np.sum(r2)
    #initial w, mu, cov
    w = [n0_1/T,n0_2/T] #(2,)
    mu_1 = np.dot(r1,X)/n0_1 #(2,)
    cov_1 = cov(X,n0_1,r1,mu_1,T) #(2,2)
    mu_2 = np.dot(r2,X)/n0_2 #(2,)
    cov_2 = cov(X,n0_2,r2,mu_2,T) #(2,2)

    L0 = L(X,w,mu_1,cov_1,mu_2,cov_2,T)

    L1 = L0 + 1
    iter = 0
    while np.abs(L1-L0)>d:
        L0 = L1
        #E-step
        r1 = np.zeros((T,))
        r2 = np.zeros((T,))
        for i in range (T):
            fi = [f(X[i],mu_1,cov_1),f(X[i],mu_2,cov_2)]
            r1[i] = w[0]*fi[0]/np.inner(w,fi)
            r2[i] = w[1]*fi[1]/np.inner(w,fi)
            n1 = np.sum(r1)
            n2 = np.sum(r2)
        #M-step
        w = [n1/T,n2/T]
        mu_1 = np.dot(r1,X)/n1
        cov_1 = cov(X,n1,r1,mu_1,T)
        mu_2 = np.dot(r2,X)/n2
        cov_2 = cov(X,n2,r2,mu_2,T)
        L1 = L(X,w,mu_1,cov_1,mu_2,cov_2,T)
        iter = iter + 1
    print("-----Set parameters-----")
    print("weight:",wt)
    print("mu1:",mu1)
    print("mu2:",mu2)
    print("cov1:",cov1)
    print("cov2:",cov2)
    print("-----Estimate parameters-----")
    print("weight:",w)
    print("mu1:",mu_1)
    print("mu2:",mu_2)
    print("cov1:",cov_1)
    print("cov2:",cov_2)
    print("iteration:",iter)
    return w,mu_1,mu_2,cov_1,cov_2

#parameters of: X = X1 + X2, X1 and X2 are 2-D r.v.
mu1 = [0,2] # X1
cov1 = [[4, 0], [0, 1]]
mu2 = [0,0] # X2
cov2 = [[1, 0], [0, 2]]
wt = [0.3,0.7]

#GMM generator
T = 300
X, label = generate_X(wt[0],mu1,cov1,mu2,cov2,T)

#EM estimation
EM(X,label,T,10**(-5))

#theoretical distribution
x, y = np.mgrid[-10:10:.01, -10:10:.01]
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x; pos[:, :, 1] = y
rv1 = multivariate_normal(mu1, cov1)
rv2 = multivariate_normal(mu2, cov2)
plt.contourf(x, y, rv2.pdf(pos)+rv1.pdf(pos))

#GMM generator
plt.scatter(X[:,0],X[:,
1],marker='^',c='orange',alpha=0.2,linewidths=None) #show sam
ples

plt.show()
```

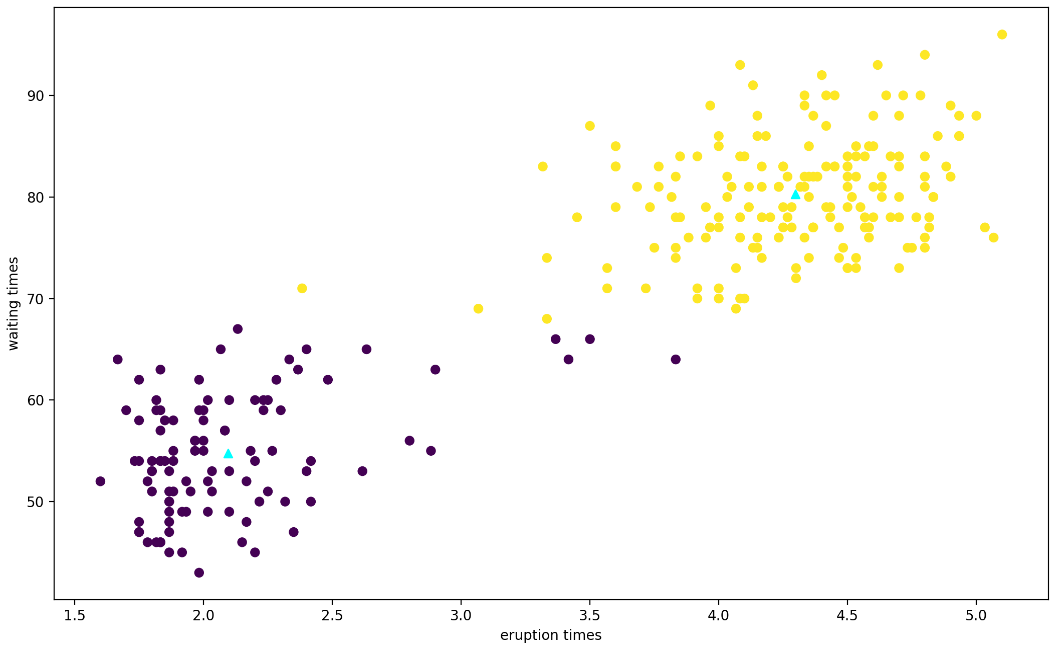
Question 4

Explain:

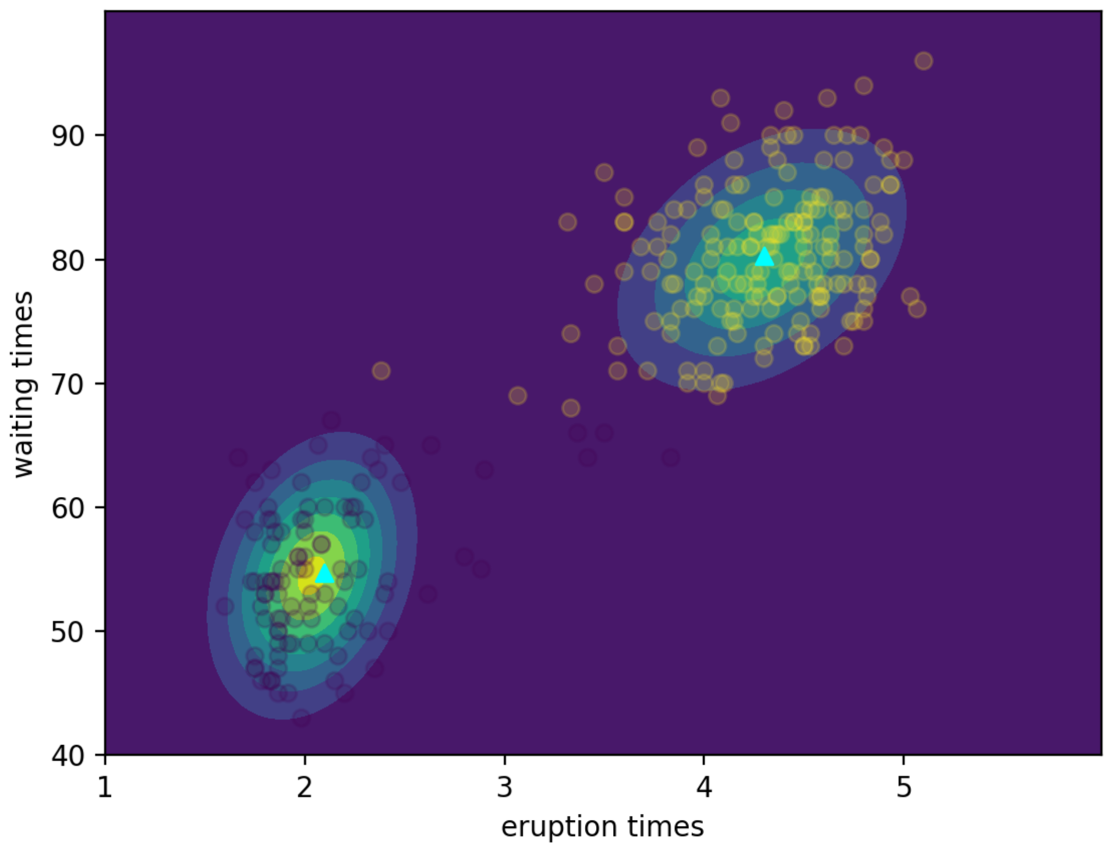
- **Generate 2D scatter of the 272 samples in pairs (eruptions, waiting times).**
- **Find the two means by running the k-means clustering ruting (set k=2).**
- **Use the GMM-EM algorithm in question 3 to estimate the GMM distribution and plot the GMM in a contour plot:**
    - run EM algorithm by using 272 samples here and get the estimated parameters of GMM
    - plot GMM in a contour plot
    - plot the 272 samples and two means

Results:

- **Plot the samples and show the means:**
    - **there are two classes in the 272 samples**



- **Plot the estimated GMM distribution:**
    - The reason that we can use GMM-EM algorithm is:
        - samples are 2 dimensional: eruption times and waiting timesand
        - there is 2 means of the samples, that is, samples can be seen as 2 separate subpopulations.
        - Thus these samples are the same kind of GMM as in question 3.
    - Our final means estimated by GMM-EM algorithm is close to the theretical means computed by k-means method.



Code:

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from GMMEM import EM
from scipy.stats import multivariate_normal

data = np.loadtxt('/Users/mac-pro/Desktop/20SPRING/511/
project 7/data.txt')

kmeans = KMeans(n_clusters=2, random_state=0).fit(data[:,
[1,2]])
label = kmeans.predict(data[:,[1,2]])+1

w,mu1,mu2,cov1,cov2 = EM(data[:,[1,2]],label,272,0.00001)
x, y = np.mgrid[1:6:.01, 40:100:.01]
pos = np.empty(x.shape + (2,))
pos[:, :, 0] = x; pos[:, :, 1] = y
rv1 = multivariate_normal(mu1, cov1)
rv2 = multivariate_normal(mu2, cov2)
plt.contourf(x, y, rv2.pdf(pos)+rv1.pdf(pos))

plt.scatter(data[:,1],data[:,
2], c=kmeans.labels_.astype(float),alpha=0.2)
plt.scatter(kmeans.cluster_centers_[:,
0],kmeans.cluster_centers_[:,1], marker='^', c='cyan')

plt.xlabel("eruption times")
plt.ylabel("waiting times")

plt.show()
```

Reference:

① https://en.wikipedia.org/wiki/Expectation%E2%80%93maximization_algorithm

② https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.cholesky.html

③ https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/
scipy.stats.multivariate_normal.html

④ https://matplotlib.org/3.2.1/api/_as_gen/matplotlib.pyplot.scatter.html