ENGR 421 Homework 6

Umur Demircioglu
64609

```python
import numpy as np
import matplotlib.pyplot as plt
import math
import pandas as pd
import scipy.spatial as spa
from scipy.stats import multivariate_normal

means = [ [2.5, 2.5], [-2.5, 2.5], [-2.5, -2.5], [2.5, -2.5], [0, 0] ]
sigmas = [ [[0.8, -0.6], [-0.6, 0.8 ]], [[0.8, 0.6], [0.6, 0.8 ]], [[0.8, -0.6], [-0.6, 0.8 ]], [[0.8, 0.6], [0.6, 0.8 ]], [[1.6, 0], [0, 1.6 ]] ]
n = [50, 50, 50, 50, 100]
colors = ['r.','g.','b.','c.','m.']

K = 5
N = 300


#np.random.seed(777)
X1 = np.random.multivariate_normal(means[0], sigmas[0], 50)
X2 = np.random.multivariate_normal(means[1], sigmas[1], 50)
X3 = np.random.multivariate_normal(means[2], sigmas[2], 50)
X4 = np.random.multivariate_normal(means[3], sigmas[3], 50)
X5 = np.random.multivariate_normal(means[4], sigmas[4], 100)
X = np.vstack((X1, X2, X3, X4, X5))

```

- I imported libraries and generated 300 data points using the parameters given in the pdf. Then combined every point into a single X numpy array.

```python
def update_centroids(memberships, X):
    if memberships is None:
        centroids = X[np.random.choice(range(N), K),:]
    else:
        centroids = np.vstack([np.mean(X[memberships == k,], axis = 0) for k in range(K)])
    return(centroids)

def update_memberships(centroids, X):
    D = spa.distance_matrix(centroids, X)
    memberships = np.argmin(D, axis = 0)
    return(memberships)


def clustering(centroids,memberships):
    iteration = 1
    while iteration in range(3):
        print("Iteration#{}:".format(iteration))

        old_centroids = centroids
        centroids = update_centroids(memberships, X)
        if np.alltrue(centroids == old_centroids):
            break


        old_memberships = memberships
        memberships = update_memberships(centroids, X)
        if np.alltrue(memberships == old_memberships):
            break


        iteration = iteration + 1
    return centroids, memberships

```

- I defined k-means clustering algorithm which iterates 2 times and returns centroid and memberships. I will use centroids as mean parameter later for initialization of the EM algorithm.

```
def getParameters(memberships):
    classSeperated = [None, None, None, None, None]
    classSeperated = np.array(classSeperated)
    for i in range(300):
        if(memberships[i] == 0):
            if type(classSeperated[0]) == type(None):
                classSeperated[0] = X[i]
            classSeperated[0] = np.vstack((classSeperated[0],X[i]))

        elif(memberships[i] == 1):
            if type(classSeperated[1]) == type(None):
                classSeperated[1] = X[i]
            classSeperated[1] = np.vstack((classSeperated[1],X[i]))

        elif(memberships[i] == 2):
            if type(classSeperated[2]) == type(None):
                classSeperated[2] = X[i]
            classSeperated[2] = np.vstack((classSeperated[2],X[i]))

        elif(memberships[i] == 3):
            if type(classSeperated[3]) == type(None):
                classSeperated[3] = X[i]
            classSeperated[3] = np.vstack((classSeperated[3],X[i]))

        else:
            if type(classSeperated[4]) == type(None):
                classSeperated[4] = X[i]
            classSeperated[4] = np.vstack((classSeperated[4],X[i]))

    cov = [None, None, None, None, None]
    priors = [None, None, None, None, None]
    for i in range(5):
        priors[i] = len(classSeperated[i])/N
        cov[i] = np.cov(np.transpose(classSeperated[i]))

    return classSeperated, priors, cov
```

- I created a getParameters algorithm which updates the covariances, priors and separates points according to the memberships array.

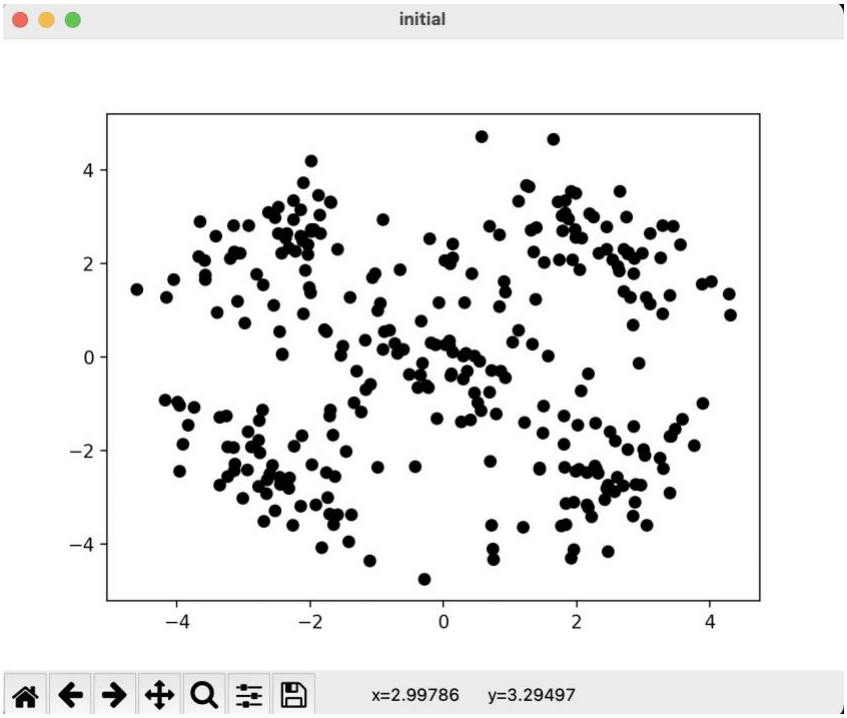
```
def score(x,v,m,p):
    return (-np.log(2*math.pi)) -1/2*(np.log(np.linalg.norm(v))) -1/2*np.matmul((x-m).T,np.matmul((np.linalg.inv(v)),(x-m)))+np.log(p)

def scoreIteration(cov, centroids, priors):
    ypred = []
    for j in range(N):
        idx = 0
        sc_max = -99999999999999999999999999999999999999999999999999999999999999999999999999999
        for i in range(5):
            sc = score(X[j, :], cov[i], centroids[i],priors[i])
            if sc > sc_max:
                idx = i
                sc_max = sc
        ypred.append(idx)

    return np.array(ypred)


def update_memberships_2(cov, centroids, priors):
    memberships = scoreIteration(cov, centroids, priors)
    return memberships




def EM(centroids,memberships):
    for i in range(100):
        old = centroids
        centroids = update_centroids(memberships, X)
        seperated, priors, cov = getParameters(memberships)
        memberships = update_memberships_2(cov, centroids, priors)
        if np.all(centroids == old):
            break
    return centroids, memberships, cov
```

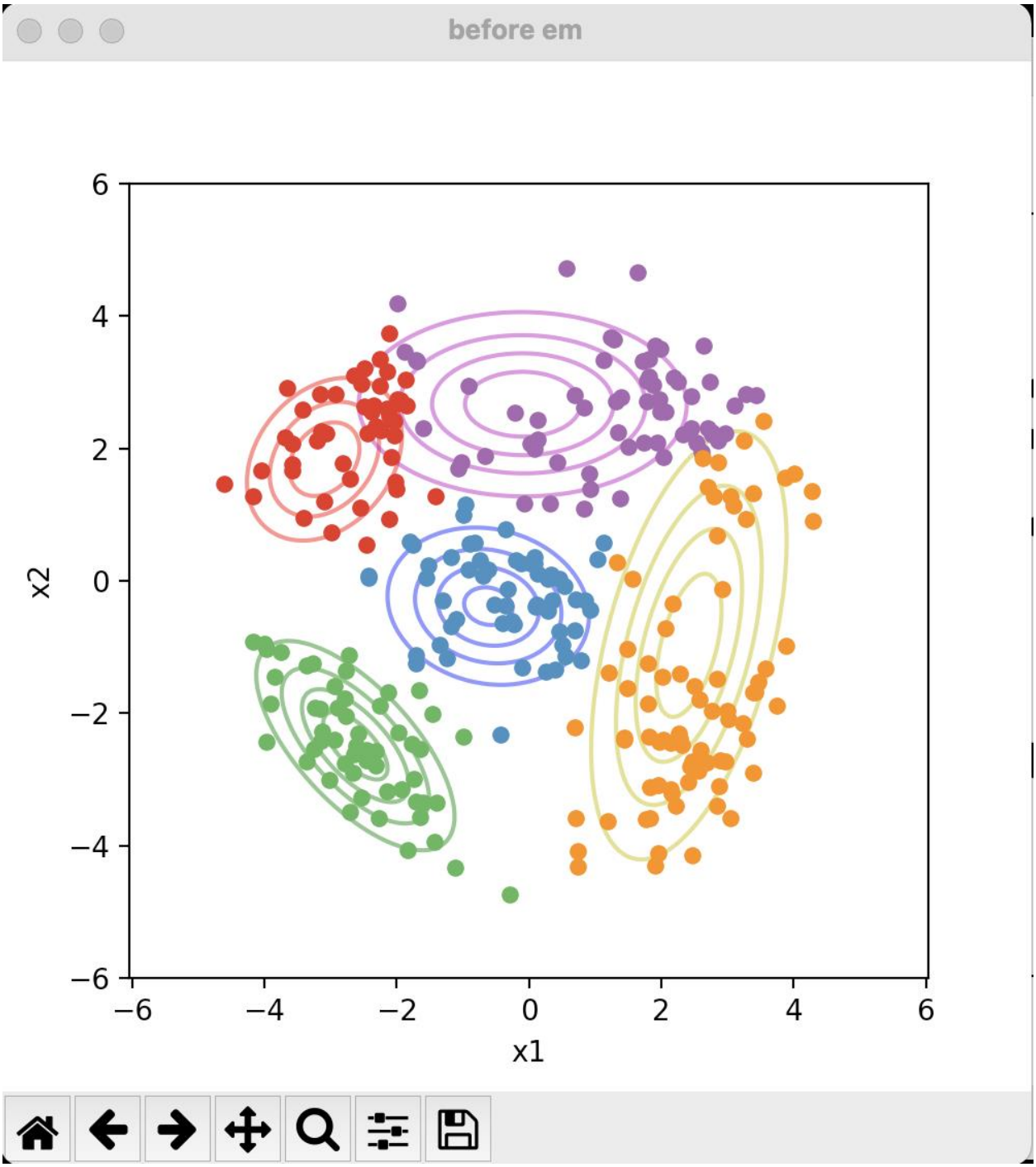
- Here I defined my score function which iterates through every point and gives a score using the mean and covariance and priors. Then I update my memberships according to results of scoreIteration function. EM function first starts with the parameters given by the k-means clustering algorithm iterates 100 times updating centroids(means), covariances, priors using the functions defined before.

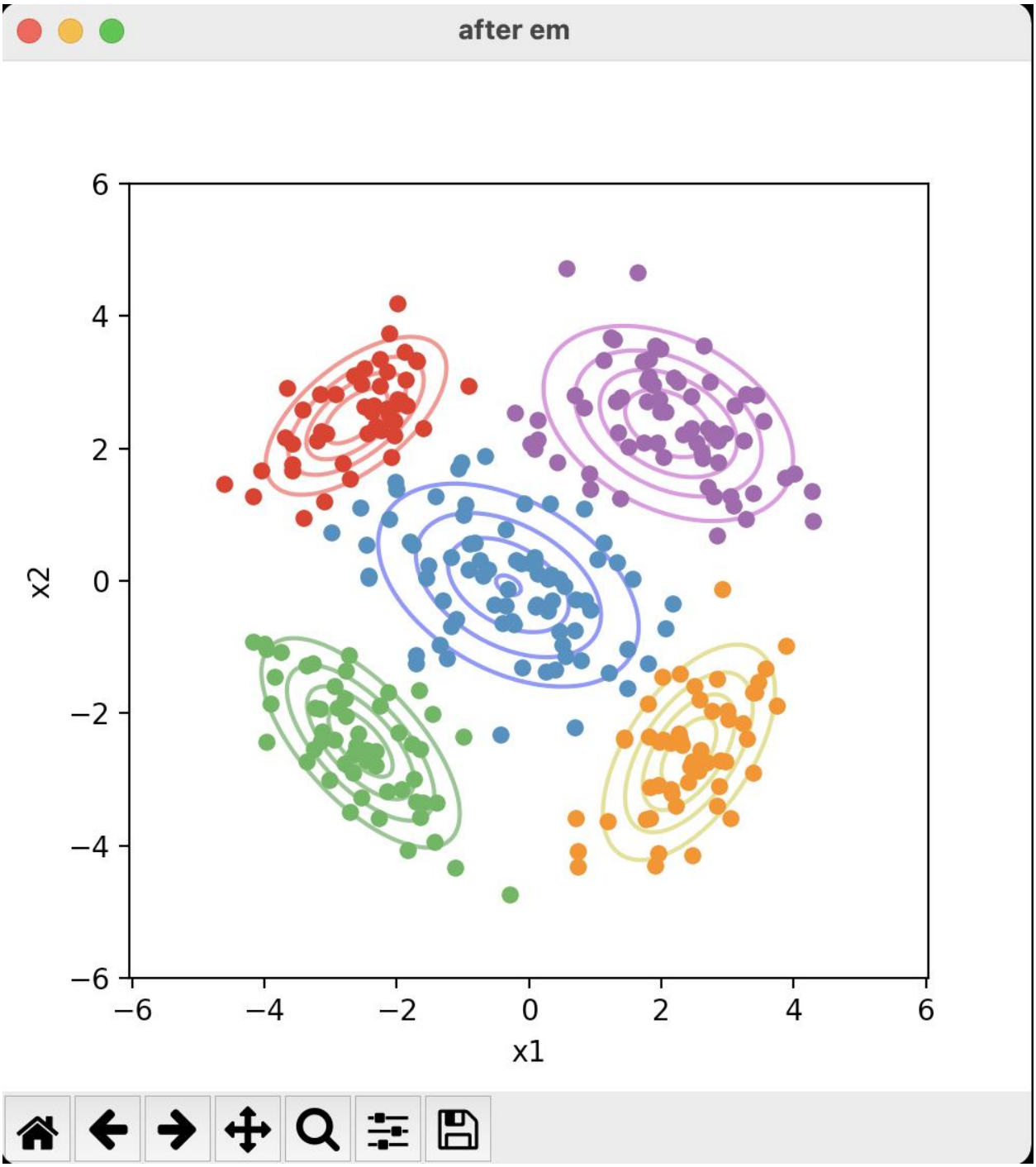
```python
def plot_current_state_2(centroids,cov,memberships, desc=1):
    x1 = np.linspace(-6,6,200)
    x2 = np.linspace(-6,6,200)
    XX, Y = np.meshgrid(x1,x2)

    Z1 = multivariate_normal(centroids[0], cov[0])
    Z2 = multivariate_normal(centroids[1], cov[1])
    Z3 = multivariate_normal(centroids[2], cov[2])
    Z4 = multivariate_normal(centroids[3], cov[3])
    Z5 = multivariate_normal(centroids[4], cov[4])

    pos = np.empty(XX.shape + (2,))
    pos[:, :, 0] = XX; pos[:, :, 1] = Y

    plt.figure(desc, figsize=(5,5))
    cluster_colors = np.array(["#e41a1c", "#377eb8", "#4daf4a", "#984ea3", "#ff7f00"])

    for c in range(K):
        plt.plot(X[memberships == c, 0], X[memberships == c, 1], ".", markersize = 10, color = cluster_colors[c])

    n = 4
    plt.contour(XX, Y, Z1.pdf(pos), n, colors="r" ,alpha = 0.5)
    plt.contour(XX, Y, Z2.pdf(pos), n, colors="b" ,alpha = 0.5)
    plt.contour(XX, Y, Z3.pdf(pos), n, colors="g" ,alpha = 0.5)
    plt.contour(XX, Y, Z4.pdf(pos), n, colors="m" ,alpha = 0.5)
    plt.contour(XX, Y, Z5.pdf(pos), n, colors="y" ,alpha = 0.5)
    plt.axis('equal')
    plt.xlabel('x1')
    plt.ylabel('x2')


plt.figure('initial')
plt.scatter(X[:, 0], X[:, 1], color="k")

centroids, memberships = None, None
centroids, memberships = clustering(centroids, memberships)
classSeperated, priors, cov = getParameters(memberships)
plot_current_state_2(centroids, cov, memberships, desc='before em')

#Use EM algorithm
centroids, memberships, cov = EM(centroids, memberships)
plot_current_state_2(centroids,cov, memberships, desc='after em')
print(centroids)
plt.show()
```

- I defined a plot_current_state_2 functions which plots the before em and after em figures. I called my k-means clustering algorithm updated parameters according to its result and used them to call em algorithm.

Outputs




before em
6
4
2
0
−2
−4
−6
x2
−6
−4
−2
0
2
4
6
x1




```
[[-2.61125539  2.49241743]
 [-0.30801355 -0.06507195]
 [-2.54257906 -2.44805021]
 [ 2.10844488  2.3767451 ]
 [ 2.42040429 -2.59047765]]
```