EE 511 Project #3: Clusters and Mixtures
Name: Yang Liu
Student ID number:5847572002
Due: Oct 21

Discussion of the results

Problem #1

In problem #1, the 1st step is to extract data from the txt file. In this processing, I used the 'getfromtxt' Function. Because the txt file doesn't contain data only, I used the islice function to get particular lines which contain data. After extraction, I deleted the 1st column data, which is the order of the data set.

The most important part of this problem is k-means processing in the kmeans.py. There are four functions in this python file: euclDistance is responsible for calculating Euclidean distance, initcentroids function is to initiate centroids with random samples, kmeans function is to get k clusters and showcluster function to show clusters and samples, I also wrote some codes to get sets of samples which belongs to different clusters. In kmeans function, there are 4 main steps, 1:initial centroids; 2: find the centroid who is closest; 3: update its cluster and last step; 4: update centroids. Within the loop from step 2 to step 4, we could get the centroids and labels of different sample points. As is shown in the picture, the original sample points in the txt file has been classified into 2 set of clusters, one is in blue and another in red. The final centroids of different cluster have also been marked in diamond.

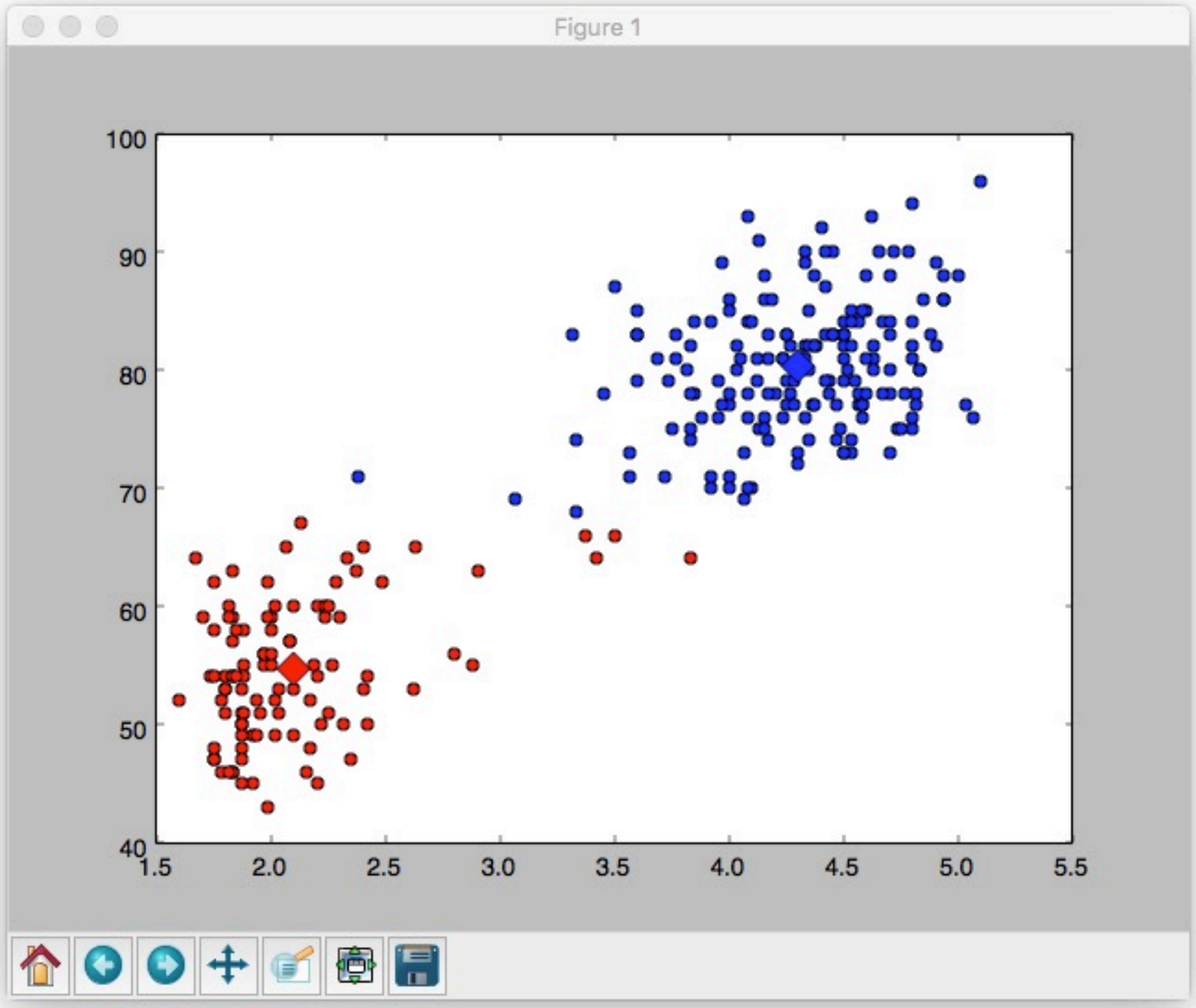
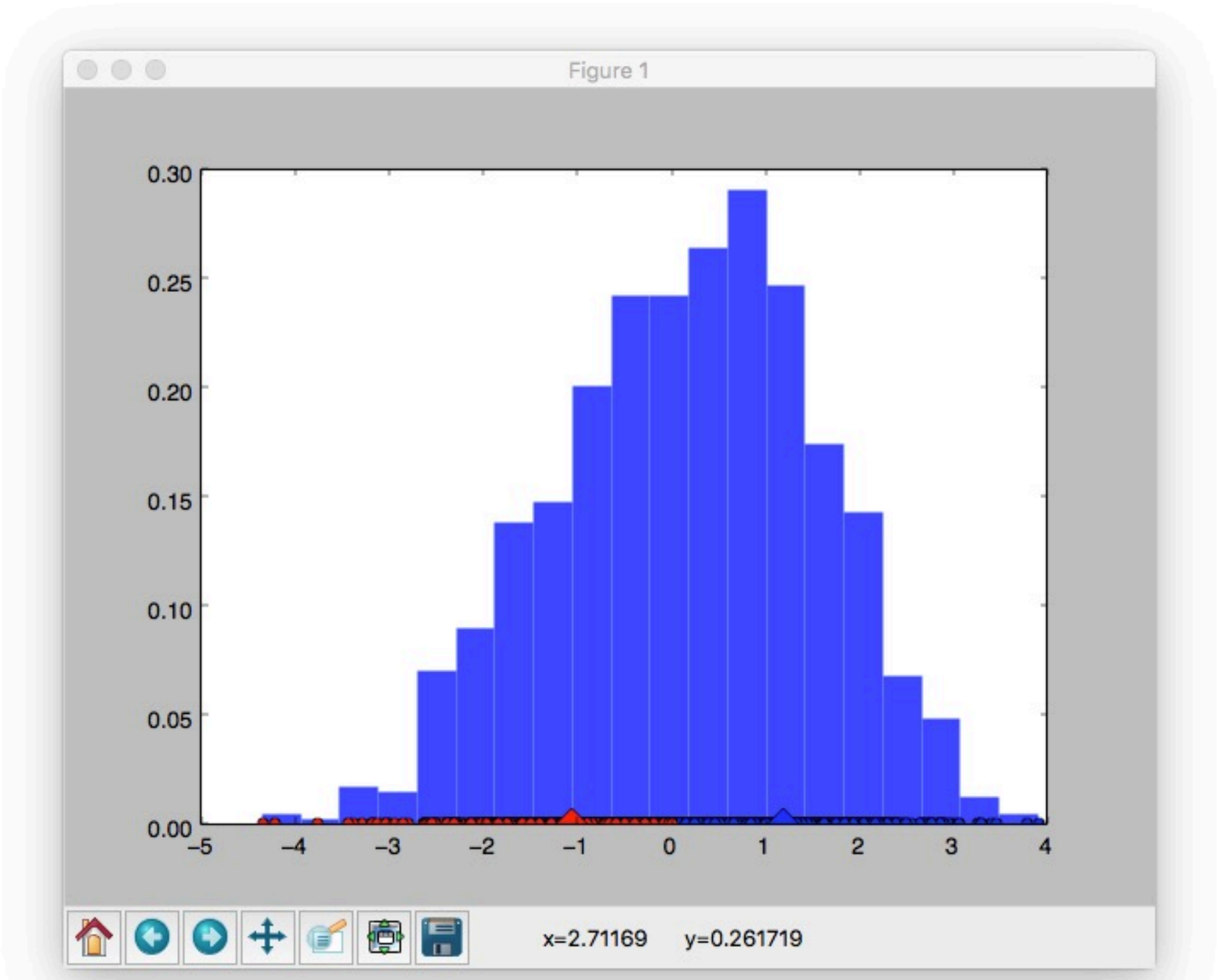


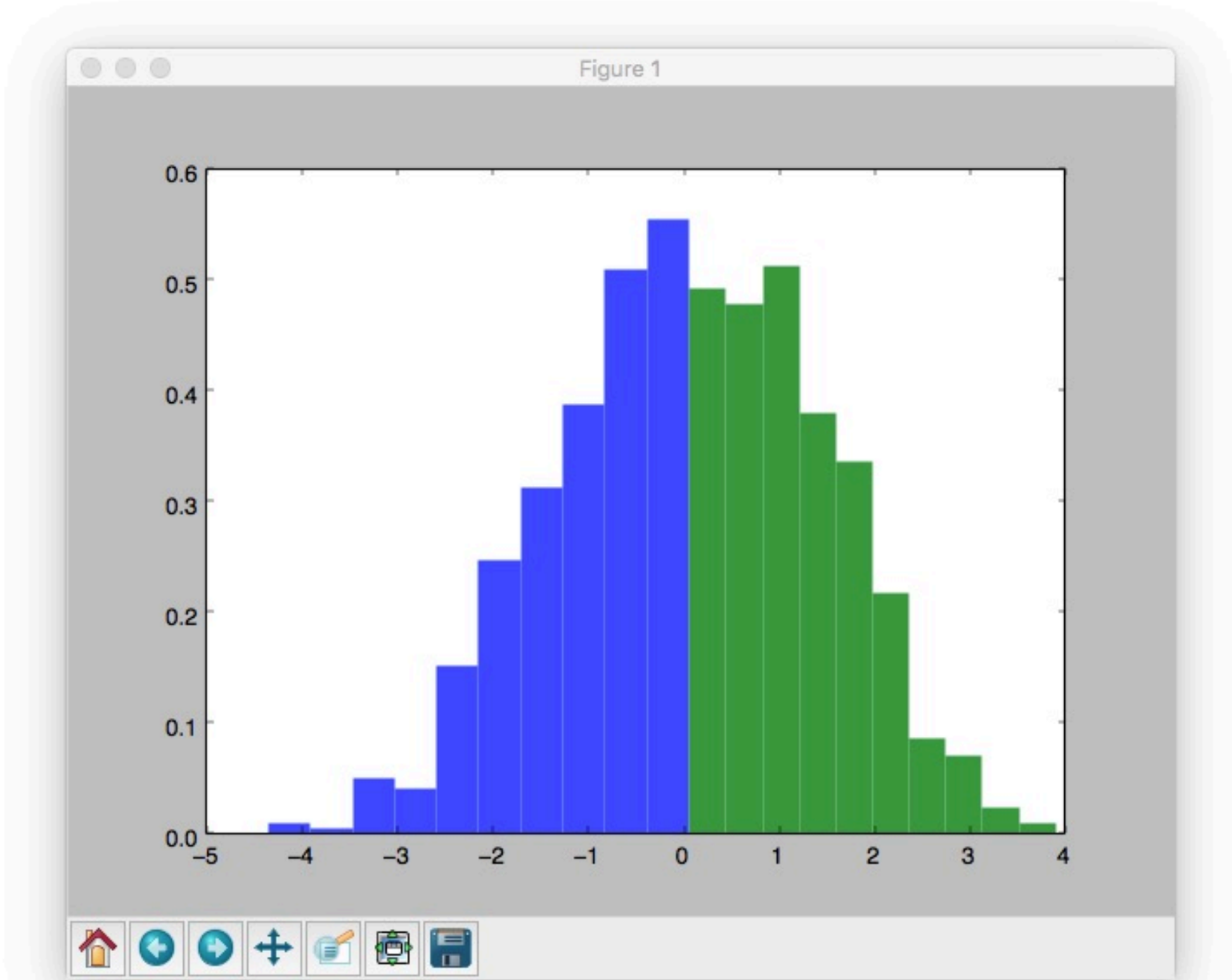
Figure 1
100
90
80
70
60
50
40
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5

Problem #2

Firstly, to generate a mixture random variable, I generate a uniform random variable p between 0 and 1, if p is less than 0.4 then random variable we want is equal to RV of Normal(-1,1), otherwise it's equal to RV of Normal(1,1). After sampling 1000 times, I got the RV and draw the histogram of it as follows.



For convenience, I make another 2d array according to this RV for k-means clustering. (2d array's kmeans is easier and clearer in python), and I used the similar method in problem 1 to do kmeans classification. we can see in the histogram that the sample points are also marked in the x axis in blue and red.
After classification, I also draw a new histogram as follows.



The green histogram is the cluster with bigger centroid and the blue one is the cluster with the smaller centroid. Because I used 2 separate cluster to calculate the histogram, the y axis is bigger(about twice) than the 1st histogram. As we can see, the two different clusters have been separated perfectly with no overlap.

Source Code

Problem 1

Kmeans.py

```
from numpy import *
import matplotlib.pyplot as plt


# calculate Euclidean distance
def euclDistance(vector1, vector2):
    return sqrt(sum(power(vector2 - vector1, 2)))


# init centroids with random samples
def initCentroids(dataSet, k):
    numSamples, dim = dataSet.shape
    centroids = zeros((k, dim))
    for i in range(k):
        index = int(random.uniform(0, numSamples))
        centroids[i, :] = dataSet[index, :]
    return centroids


# k-means cluster
def kmeans(dataSet, k):
    numSamples = dataSet.shape[0]
    # first column stores which cluster this sample belongs to,
    # second column stores the error between this sample and its centroid
    clusterAssment = mat(zeros((numSamples, 2)))
    clusterChanged = True

    ## step 1: init centroids
    centroids = initCentroids(dataSet, k)

    while clusterChanged:
        clusterChanged = False
        ## for each sample
        for i in xrange(numSamples):
            minDist = 100000.0
            minIndex = 0
```

```
            ## for each centroid
            ## step 2: find the centroid who is closest
            for j in range(k):
                distance = euclDistance(centroids[j, :], dataSet[i, :])
                if distance < minDist:
                    minDist = distance
                    minIndex = j

                    ## step 3: update its cluster
            if clusterAssment[i, 0] != minIndex:
                clusterChanged = True
                clusterAssment[i, :] = minIndex, minDist ** 2

                ## step 4: update centroids
        for j in range(k):
            pointsInCluster = dataSet[nonzero(clusterAssment[:, 0].A == j)[0]]
            centroids[j, :] = mean(pointsInCluster, axis=0)

    print 'Congratulations, cluster complete!'
    return centroids, clusterAssment


# show your cluster only available with 2-D data
def showCluster(dataSet, k, centroids, clusterAssment):

    numSamples, dim = dataSet.shape
    s1=[]
    s0=[]
    if dim != 2:
        print "Sorry! I can not draw because the dimension of your data is not 2!"
        return 1

    mark = ['or', 'ob', 'og', 'ok', '^r', '+r', 'sr', 'dr', '<r', 'pr']
    if k > len(mark):
        print "Sorry! Your k is too large!"
        return 1

        # draw all samples
    for i in xrange(numSamples):
        markIndex = int(clusterAssment[i, 0])
        plt.plot(dataSet[i, 0], dataSet[i, 1], mark[markIndex])
        if (markIndex==0):
```

```
            s0.append(dataSet[i, 0])
        if (markIndex == 1):
            s1.append(dataSet[i, 0])


    mark = ['Dr', 'Db', 'Dg', 'Dk', '^b', '+b', 'sb', 'db', '<b', 'pb']
    # draw the centroids
    for i in range(k):
        plt.plot(centroids[i, 0], centroids[i, 1], mark[i], markersize=12)

    plt.show()
    return s0,s1
```

Pronblem 1.py

```
from itertools import islice
import numpy as np
from kmeans import *

with open('old-faithful.txt') as lines:
    array = np.genfromtxt(islice(lines, 26, 298))

data = np.delete(array,0,1)
print data

data = mat(data)
k = 2
centroids, clusterAssment = kmeans(data, k)


plt.figure()
for i in range(0,272,1):
    plt.plot(data[i, 0], data[i, 1],'ro')

showCluster(data, k, centroids, clusterAssment)

plt.show()
```

Problem 2

```
import numpy as np
from kmeans import *

mu, sigma = -1, 1 # mean and standard deviation
mu2, sigma2 = 1, 1 # mean and standard deviation

data=[]
cc=np.zeros(2000).reshape(1000,2)
for i in range(0,1000,1):
    p=random.random()

    if p<0.4:
        temp=float(np.random.normal(mu, sigma, 1))
        data.insert(i,temp)
        cc[i,0]=temp
    else:
        temp=float(np.random.normal(mu2, sigma2, 1))
        data.insert(i, temp)
        cc[i, 0] = temp

l=len(data)

print data
plt.hist(data,20, normed=True,lw=0,alpha=.8)

print cc

cc = mat(cc)
k = 2
centroids, clusterAssment = kmeans(cc, k)
print clusterAssment

s0,s1=showCluster(cc, k, centroids, clusterAssment)
plt.hist(s0,10, normed=True,lw=0,alpha=.8)
plt.hist(s1,10, normed=True,lw=0,alpha=.8)
plt.show()
```