ARVIND SAI DOODA

CS584 Machine Learning
Assignment 3

In this pdf I attached the 2 problems 4 (20 points) and other is (45 points).

Problem 4 (20 Points): Generating GMMs

In this problem, you will write code to generate a mixture of 3 Gaussians satisfying the following requirements, respectively. Please specify the mean vector and covariance matrix of each Gaussian in your answer.

1. **6 Points.** Draw a data set where a mixture of 3 spherical Gaussians (where the covariance matrix is the identity matrix times some positive scalar) can model the data well, but K-means cannot.

Explanation :

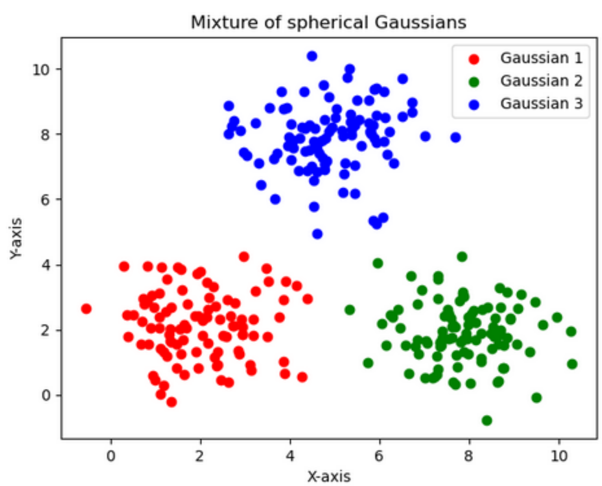
Three Spherical Gaussians Mixed Together (Identity Covariance Matrix):

- In this scenario, we will generate a dataset where three spherical Gaussian distributions are combined. Each Gaussian is characterized by the following parameters:
- Gaussian 1: Mean = [2, 2], Covariance = Identity matrix scaled by 1.
- Gaussian 2: Mean = [8, 2], Covariance = Identity matrix scaled by 1.
- Gaussian 3: Mean = [5, 8], Covariance = Identity matrix scaled by 1.
- We will proceed to create this dataset to meet these specifications.

Code snippet was pasted below

In [17]:
```python
#1. 6 Points. Draw a data set where a mixture of 3 spherical Gaussians (where the covariance matrix is
#the identity matrix times some positive scalar) can model the data well, but K-means cannot.

#-----------------------------------------------------------------------------------------------#

import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(0)

# Define the means and covariance matrices for each Gaussian
mean1 = [2, 2]
cov1 = np.identity(2)  # Identity matrix scaled by 1

mean2 = [8, 2]
cov2 = np.identity(2)  # Identity matrix scaled by 1

mean3 = [5, 8]
cov3 = np.identity(2)  # Identity matrix scaled by 1

# Generate data points for each Gaussian
num_samples = 100
data1 = np.random.multivariate_normal(mean1, cov1, num_samples)
data2 = np.random.multivariate_normal(mean2, cov2, num_samples)
data3 = np.random.multivariate_normal(mean3, cov3, num_samples)

# Combine the data from all three Gaussians
data = np.vstack((data1, data2, data3))

# Shuffle the data to make it more realistic
np.random.shuffle(data)

# Define colors for each Gaussian
colors = ['r', 'g', 'b']

# Plot the generated dataset with different colors
plt.scatter(data1[:, 0], data1[:, 1], c=colors[0], label='Gaussian 1')
plt.scatter(data2[:, 0], data2[:, 1], c=colors[1], label='Gaussian 2')
plt.scatter(data3[:, 0], data3[:, 1], c=colors[2], label='Gaussian 3')

plt.title("Mixture of spherical Gaussians")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
```

Output :

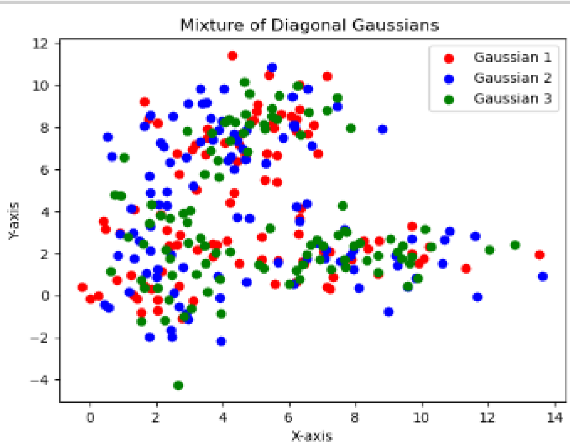


2. **6 Points.** Draw a data set where a mixture of 3 diagonal Gaussians (where the covariance matrix can have non-zero values on the diagonal, and zeros elsewhere) can model the data well, but K-means and a mixture of spherical Gaussians cannot.

Three Diagonal Gaussians Mixed Together (Non-Zero Diagonal Values):

- Gaussian 1: Mean = [2, 2], Covariance = Diagonal matrix with diagonal values [1, 6].
- Gaussian 2: Mean = [8, 2], Covariance = Diagonal matrix with diagonal values [6, 1].
- Gaussian 3: Mean = [5, 8], Covariance = Diagonal matrix with diagonal values [2, 2].
- We will proceed to create this dataset to meet these specifications.

```python
#  Draw a data set where a mixture of 3 diagonal Gaussians (where the covariance matrix can
#   have non-zero values on the diagonal, and zeros elsewhere) can model the data well, but K-means and a
#  mixture of spherical Gaussians cannot. done by Arvind.

import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(0)

# Define the means and covariance matrices for each Gaussian
mean1 = [2, 2]
cov1 = np.diag([1, 6])  # Increased variance on the diagonal

mean2 = [8, 2]
cov2 = np.diag([6, 1])  # Increased variance on the diagonal

mean3 = [5, 8]
cov3 = np.diag([2, 2])  # Increased variance on the diagonal

# Generate data points for each Gaussian
num_samples = 100
data1 = np.random.multivariate_normal(mean1, cov1, num_samples)
data2 = np.random.multivariate_normal(mean2, cov2, num_samples)
data3 = np.random.multivariate_normal(mean3, cov3, num_samples)

# Combine the data from all three Gaussians
data = np.vstack((data1, data2, data3))

# Create labels for data points from each Gaussian
labels = np.array([0] * num_samples + [1] * num_samples + [2] * num_samples)

# Shuffle the data to make it more realistic
np.random.shuffle(data)
np.random.shuffle(labels)

# Define colors for each Gaussian to deferentiate heheheeeeeeee.......
colors = ['r', 'b', 'g']

# Plot the generated dataset with different colors
for i in range(3):
    plt.scatter(data[labels == i, 0], data[labels == i, 1], c=colors[i], label=f'Gaussian {i+1}')

plt.title("Mixture of Diagonal Gaussians ")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()

#finally code completed LolL
```



mixture of spherical Gaussians cannot.

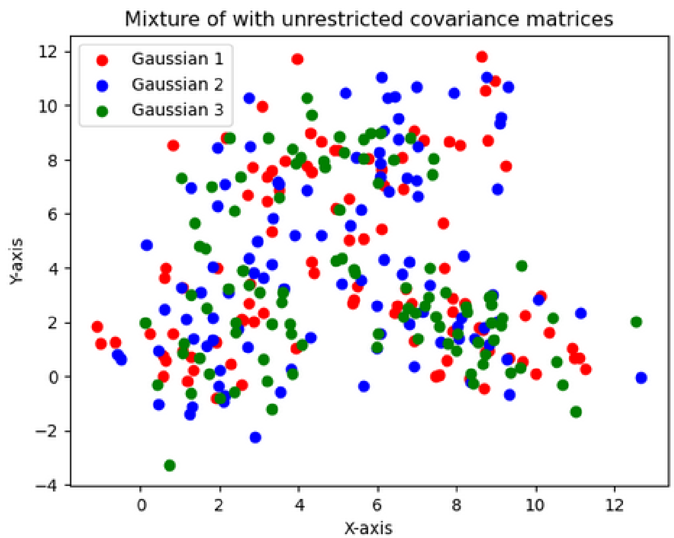
3. **8 Points.** Draw a data set where a mixture of 3 Gaussians with unrestricted covariance matrices can model the data well, but K-means and a mixture of diagonal Gaussians cannot.

- Gaussian 1: Mean = [2, 2], Covariance Matrix = [[2, 1], [1, 4]].
- Gaussian 2: Mean = [8, 2], Covariance Matrix = [[3, -1], [-1, 2]].
- Gaussian 3: Mean = [5, 8], Covariance Matrix = [[4, 1], [1, 3]].

In [19]:

```python
# Draw a data set where a mixture of 3 Gaussians with unrestricted covariance matrices can
# model the data well, but K-means and a mixture of diagonal Gaussians cannot.

import numpy as np
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(0)

# Define the means and covariance matrices for each Gaussian
mean1 = [2, 2]
cov1 = np.array([[2, 1], [1, 4]])         #cov - covernce

mean2 = [8, 2]
cov2 = np.array([[3, -1], [-1, 2]])        #mean - mean

mean3 = [5, 8]
cov3 = np.array([[4, 1], [1, 3]])

num_samples = 100
data1 = np.random.multivariate_normal(mean1, cov1, num_samples)
data2 = np.random.multivariate_normal(mean2, cov2, num_samples)
data3 = np.random.multivariate_normal(mean3, cov3, num_samples)

# Combine the data of Gaussians
data = np.vstack((data1, data2, data3))

# Create labels for data points from each Gaussian
labels = np.array([0] * num_samples + [1] * num_samples + [2] * num_samples)

# Shuffle the data to make it more realistic
np.random.shuffle(data)
np.random.shuffle(labels)

# Define colors for each Gaussian, i like rgb lol.
colors = ['r', 'b', 'g']

# Plot the generated dataset with different colors
for i in range(3):
    plt.scatter(data[labels == i, 0], data[labels == i, 1], c=colors[i], label=f'Gaussian {i+1}')

plt.title("Mixture of with unrestricted covariance matrices")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
# finally ------------------------------complted 3rd problem ----------------------------
```

Output :



Problem 4 (45 Points): Implementing K-Means and Spectral Clustering

1. **20 Points. Implementing Lloyd's K-means:** Your submitted function should be `function [label] = my_kmeans(data, K)`, where `label` returns the N-dimensional clustering result, where N is the total number of data points. `data` is with size $N \times d$ and K is the number of (known) clusters. To initialize, randomly select K samples to initialize your cluster centroids. Iterate your algorithm until convergence. Use **Euclidean distance** as the distance measure. Name your file `my_kmeans.m`.
2. **20 Points. Implementing Spectral Clustering:** Your submitted function should be `function [label] = my_spectralclusting(data, K, sigma)`, where `label`, `data` and K are the same as above and `sigma` is the bandwith for **Gaussian kernel** used in spectral clustering. You will see `sigma` is important for your clustering performance. Adjust it case-by-case for every toy dataset to output the best results. Name your file `my_spectralclusting.m`.
3. **5 Points.** Compare your spectral clustering results with k-means. It is natural that on certain hard toy example, both method won't generate perfect results. In your report, briefly analyze what is the advantage or disadvantage of spectral clustering over k-means. Why it is the case? (You do not need to mathematically prove it but just need to give answers in your own language.)

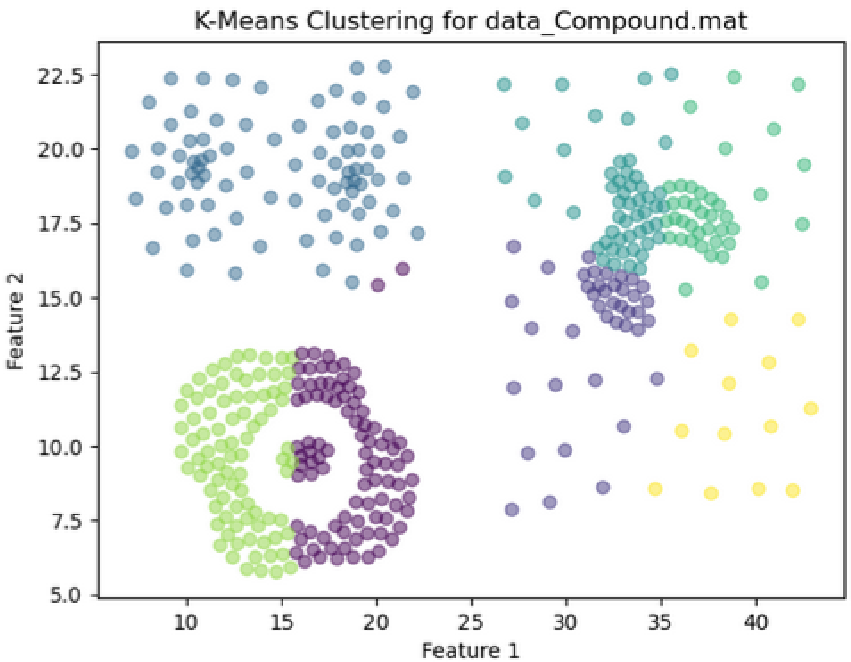
```python
In [6]: import numpy as np
import scipy.io
import matplotlib.pyplot as plt
import os

def assign_points(data, centroids):
    distances = np.linalg.norm(data[:, np.newaxis] - centroids, axis=2)
    return np.argmin(distances, axis=1)

def compute_centroids(data, assignments, k):
    centroids = np.zeros((k, data.shape[1]))
    for i in range(k):
        cluster_points = data[assignments == i]
        if len(cluster_points) > 0:
            centroids[i] = np.mean(cluster_points, axis=0)
    return centroids

def KMeans(k, data, max_iters=100):
    # Randomly initialize centroids
    centroids = data[np.random.choice(len(data), k, replace=False)]

    for _ in range(max_iters):
        # Assign each data point to the nearest centroid
        assignments = assign_points(data, centroids)

        # Compute new centroids
        new_centroids = compute_centroids(data, assignments, k)

        # If centroids haven't changed, break
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids

    return centroids, assignments

# Specify the folder containing the .mat files here is the arvinds machine directory lol
data_dir = r"C:\Users\arvin\Downloads\toydata\cluster"

# List all .mat files in the directory
mat_files = [file for file in os.listdir(data_dir) if file.endswith('.mat')]

for mat_file in mat_files:
    # Load the data from the MATLAB file
    mat = scipy.io.loadmat(os.path.join(data_dir, mat_file))
    data = mat['D']  # Adjust this based on your file's structure

    k = 7  # Number of clusters
    centroids, assignments = KMeans(k, data)
    print(f"Final centroids for {mat_file}:")
    print(centroids)
    print(f"Cluster assignments for {mat_file}:")
    print(assignments)

    # Modify this part to save or display the results as needed
    plt.scatter(data[:, 0], data[:, 1], c=assignments, cmap='viridis', alpha=0.5)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(f'K-Means Clustering for {mat_file}')
    plt.show()
```

Outputs :

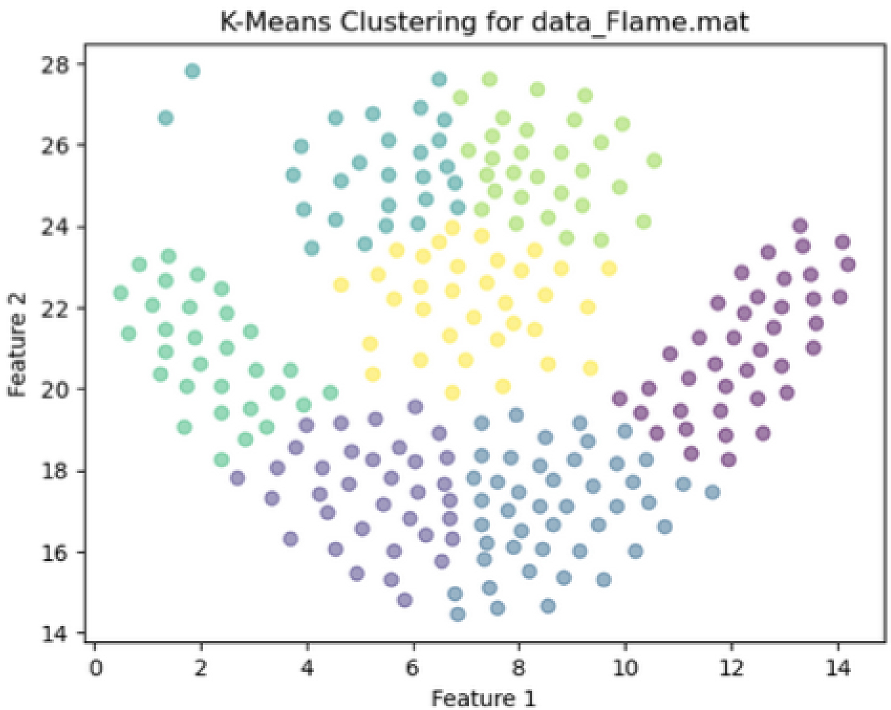
```
Final centroids for data_Aggregation.mat:
[[18.94310345  9.48922414]
 [21.28723404 22.99893617]
 [ 9.29464286 22.95267857]
 [33.14278846  8.79375   ]
 [ 8.99086957  7.64695652]
 [32.69453125 22.13789062]
 [18.12318182  4.56363636]]
Cluster assignments for data_Aggregation.mat:
[1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 0 0 0 0 0
 0 0 0 0 0 4 4 4 4 4 4 4 4 4 4 4 0 4 4 4 4 4 4 4 4 4 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 6 4 6 6 4 4 4 4 4 4 4 4 4 6 6 4 4 4 4 4 4 4 6 6 6 6 6 6 6 6 6 6 6
 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
 6 6 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6
 6 6 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 6 0 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4]
```

```
Final centroids for data_Bridge.mat:
[[25.67272727 32.51704545]
 [18.01666667 32.44833333]
 [11.59125    32.54125   ]
 [21.38833333 34.85166667]
 [ 7.78965517 35.2       ]
 [21.482      30.726     ]
 [ 6.17205882 32.1       ]]
Cluster assignments for data_Bridge.mat:
[6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 4 4 6 6 4 4 4 4 6 4 4 6 6 6
 6 6 6 6 2 2 2 2 2 6 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
 1 1 1 1 1 5 5 5 5 5 5 5 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 0 0 0 5 5 5 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0]
```

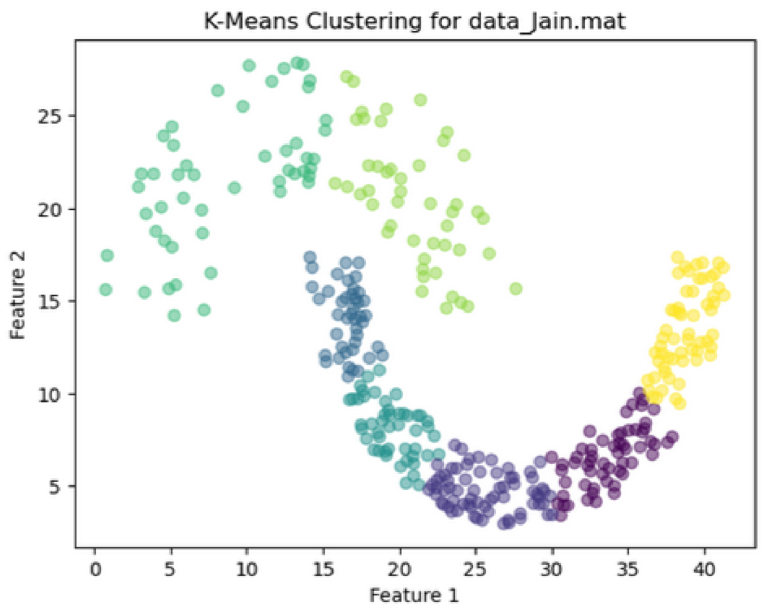
```
Final centroids for data_Compound.mat:
[[18.49505495  9.61043956]
 [31.51875    13.86125   ]
 [14.94012346 19.28518519]
 [32.80490196 18.36960784]
 [37.75263158 18.01578947]
 [12.85588235  9.62764706]
 [39.18846154 11.03076923]]
Cluster assignments for data_Compound.mat:
[3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 6 1 4 6 6 4 6 6 6 6 6 6 6 6 6
 6 4 4 4 4 4 4 4 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3 3 3 3 3 3
 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 1 1 1 3 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 0 0 2 5 5 5 5 5 5 5 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 0 5 5 5 5 5 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 5 5 5]
```



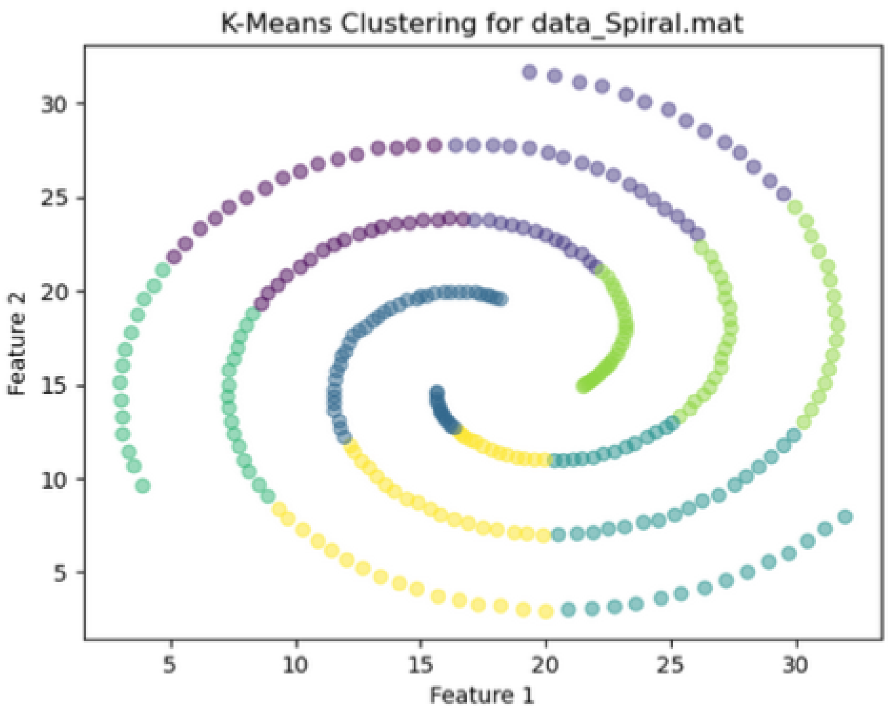
```
Final centroids for data_Flame.mat:
[[12.27564103 20.99487179]
 [ 5.26        17.40714286]
 [ 8.66777778 17.03222222]
 [ 5.21851852 25.44259259]
 [ 2.20333333 20.83833333]
 [ 8.49166667 25.51166667]
 [ 7.14852941 22.08529412]]
Cluster assignments for data_Flame.mat:
[3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 1 1 1 4 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 1 2 1 2 1
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 2 2 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 3 6 6 6 6 6 6 6 6 6 6 6 6 3 3 3
 3 6 6 6 6 5 5 5 5 3 3 3 3 3 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 3
 3 3 3 3 3 3 3 5 3 3 3 3 5 5 5 5 5 3]
```



```
Final centroids for data_Jain.mat:
[[33.78534483  6.57844828]
 [25.70294118  4.71323529]
 [16.64418605 14.06744186]
 [19.546875    8.125     ]
 [ 8.765      21.655     ]
 [21.01595745 20.35531915]
 [38.78050847 13.45847458]]
Cluster assignments for data_Jain.mat:
[4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
 6 6 6]
```



```
Final centroids for data_Spiral.mat:
[[11.27794118 23.90294118]
 [22.18522727 26.01931818]
 [14.5244898  16.58061224]
 [25.05909091  8.40454545]
 [ 5.81833333 14.67666667]
 [25.89765625 17.653125  ]
 [15.69042553  8.29042553]]
Cluster assignments for data_Spiral.mat:
[3 3 3 3 3 3 3 3 3 3 3 3 3 3 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
 1 1 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 3 3 3 3 3 3 3 3 3 3 3 3 6 6 6 6 6 6 6 6 6 6
 6 6 6 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

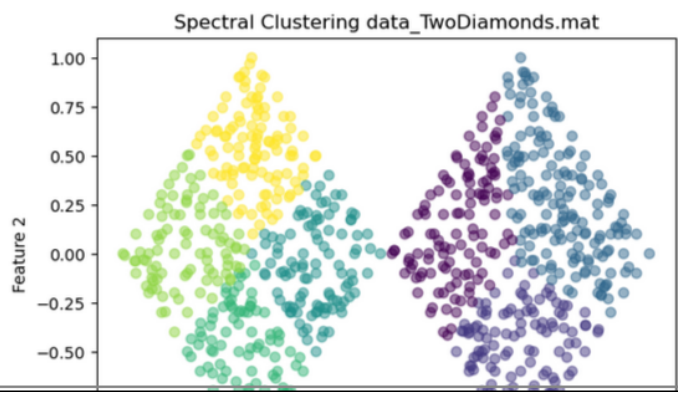


```
Final centroids for data_TwoDiamonds.mat:
[[ 2.55038275 -0.06715997]
 [ 0.73997874 -0.3437585 ]
 [ 0.88452359  0.48743387]
 [ 3.09368689 -0.56246197]
 [ 3.03397139  0.51479011]
 [ 3.55665531  0.03595618]
 [ 1.48187614 -0.05102916]]
Cluster assignments for data_TwoDiamonds.mat:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 2 1 1
 1 1 1 1 1 1 1 2 2 2 2 1 1 1 1 6 6 6 2 2 2 2 2 1 6 6 6 6 6 2 2 2 2 2 6 6 6
 6 6 6 2 2 2 2 2 6 6 6 6 6 6 2 2 2 2 2 6 6 6 6 6 6 2 2 2 2 2 6 6 6 6 6 6 2
 2 2 2 2 6 6 6 6 6 6 1 6 6 2 1 2 1 1 1 6 1 1 2 1 6 1 2 6 2 6 6 2 6 2 6 2 6
 1 1 6 1 2 1 2 6 1 2 2 1 2 1 2 2 1 6 2 2 6 1 2 2 6 1 1 2 1 6 6 6 1 6 6 2 6
 6 2 2 1 1 2 2 6 6 2 1 2 2 2 2 6 2 1 1 2 2 1 6 6 1 1 1 1 6 1 6 1 6 2 6 2 2
 2 2 2 6 6 1 6 6 2 1 2 1 1 6 1 6 2 6 2 1 2 1 1 2 1 1 2 2 6 6 2 2 2 6 1 6 6
 1 1 6 2 1 1 2 1 6 2 1 6 1 6 2 2 6 1 2 6 1 6 1 6 2 1 2 6 2 6 1 6 6 6 1 1 6
 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 6 2 2 2 6 1 6 1 6 6 1 2 1 1 2 6 1 2 2 2 6 1
 2 6 2 2 2 6 2 1 6 1 6 2 2 2 1 1 1 2 1 6 1 1 6 2 2 2 1 1 1 1 2 6 1 1 6 6 2
 6 1 6 1 2 2 2 6 1 1 6 1 2 1 6 1 2 1 2 6 6 1 1 1 1 1 1 2 1 1 0 0 0 0 0 0 3
 3 3 3 3 0 0 0 0 0 0 3 3 3 3 3 0 0 0 0 0 0 3 3 3 3 3 0 0 0 0 0 0 3 3 3 3 3
 0 0 0 0 0 0 3 3 3 3 3 4 4 4 4 4 5 5 5 5 3 3 4 4 4 4 4 5 5 5 5 5 5 4 4 4 4
 4 5 5 5 5 5 5 4 4 4 4 4 5 5 5 5 5 5 4 4 4 4 4 5 5 5 5 5 5 4 4 4 4 4 5 5 5
 5 5 5 5 0 4 0 5 0 3 0 5 4 3 4 5 5 0 0 3 3 0 5 0 4 4 5 5 0 4 3 3 5 0 5 5 4
 0 5 4 4 5 3 3 0 3 3 5 4 5 5 0 3 5 4 3 4 4 4 0 3 3 5 5 5 3 4 0 3 4 0 5 4 5
 0 3 4 4 4 3 0 3 3 0 4 3 4 0 4 5 3 5 4 0 4 3 3 0 0 3 5 5 4 5 3 4 3 3 4 0 0
 3 4 4 5 0 3 5 5 0 5 0 5 3 0 5 3 3 4 0 5 5 0 3 5 5 3 5 4 5 4 5 4 3 3 3 5 5
 4 4 5 3 0 3 3 5 0 5 5 4 5 5 4 4 5 5 5 5 5 3 5 3 4 3 0 3 5 3 4 4 3 4 4 0 4
 0 5 0 0 3 4 0 5 4 5 3 4 4 0 5 0 5 4 3 4 0 0 0 4 4 0 3 4 3 5 3 0 3 3 4 4 0
 4 4 0 5 5 0 5 4 5 0 4 4 4 4 0 5 5 0 3 3 4 5 3 4 0 5 5 4 0 3 4 3 5 0 4 4 0
 5 3 5 0 3 5 4 5 4 3 0 5 5 0 4 5 3 0 5 0 0 5 0]
```

2.

```
In [ ]: import numpy as np
        from sklearn.cluster import KMeans
        from sklearn.metrics.pairwise import pairwise_distances
        from sklearn.neighbors import kneighbors_graph
        from scipy.sparse.csgraph import laplacian
        import scipy.io

        def my_spectralclustering(data, K, sigma):
            pairwise_dists = pairwise_distances(data, metric='euclidean')
            affinity_matrix = np.exp(-pairwise_dists*2 / (2.0 * sigma*2))

            # Step 2: Compute the degree matrix
            degree_matrix = np.diag(np.sum(affinity_matrix, axis=1))

            # Step 3: Compute the Laplacian matrix
            laplacian_matrix = degree_matrix - affinity_matrix

            # Step 4: Compute the eigenvectors and eigenvalues of the Laplacian matrix
            eigvals, eigvecs = np.linalg.eigh(laplacian_matrix)

            # Step 5: Select the K eigenvectors corresponding to the smallest eigenvalues
            eigvecs = eigvecs[:, :K]

            # Step 6: Apply KMeans clustering on the selected eigenvectors
            kmeans = KMeans(n_clusters=K, random_state=0)
            labels = kmeans.fit_predict(eigvecs)

            return labels

        # Example usage:
        if _name_ == "_main_":
            from sklearn.datasets import make_moons
            import matplotlib.pyplot as plt

            # Generate synthetic data
            mat = scipy.io.loadmat('data_Aggregation.mat')
            data = mat['D']
            # Apply spectral clustering
            clusters = my_spectralclustering(data, 7, 1)

            # Plot the results
            plt.scatter(data[:, 0], data[:, 1], c=clusters, cmap='viridis')
            plt.title("Spectral Clustering")
            plt.show()
```

```
Final centroids for data_TwoDiamonds.mat:
[[ 2.61014616  0.14308669]
 [ 3.08906611 -0.49524366]
 [ 3.4197029   0.26542809]
 [ 1.50081897 -0.06306293]
 [ 0.94413767 -0.52250685]
 [ 0.48353826  0.04164483]
 [ 1.03108951  0.53984518]]
Cluster assignments for data_TwoDiamonds.mat:
[5 5 5 5 5 4 4 4 4 4 4 5 5 5 5 5 4 4 4 4 4 4 5 5 5 5 5 4 4 4 4 4 4 5 5 5 5
 5 4 4 4 4 4 4 5 5 5 5 5 4 4 4 4 4 4 5 5 5 5 5 3 3 3 3 3 3 6 6 6 6 6 3 3 3
 3 3 3 6 6 6 6 6 6 3 3 3 3 3 6 6 6 6 6 6 3 3 3 3 3 6 6 6 6 6 6 3 3 3 3 3 6
 6 6 6 6 6 3 3 3 3 3 5 3 3 6 4 6 5 4 4 3 5 5 5 4 3 4 6 3 6 3 3 6 3 6 3 6 3
 4 4 3 4 6 4 6 3 5 5 6 5 6 4 5 6 4 3 6 6 6 4 6 6 3 4 4 6 5 3 6 6 4 3 4 6 3
 3 6 6 5 4 6 5 3 6 6 4 5 6 6 6 3 6 4 5 5 5 4 3 3 4 5 5 5 3 4 6 5 3 6 3 6 6
 6 6 6 3 3 5 3 3 6 4 5 5 4 3 4 3 6 3 6 5 6 4 5 6 4 4 6 6 3 4 6 5 6 3 5 3 3
 4 4 6 6 5 4 6 4 3 5 4 3 4 4 5 6 3 5 6 3 4 3 4 3 6 4 6 3 5 3 5 3 3 3 5 5 3
 5 4 5 4 4 4 4 5 5 4 4 6 4 4 4 3 6 6 6 3 4 3 4 3 3 4 5 4 4 6 3 4 6 6 6 3 5
 6 3 5 5 6 3 6 4 3 4 3 6 6 6 5 5 5 6 5 3 4 5 3 6 6 5 5 5 5 4 6 3 4 4 3 3 5
 3 4 3 5 6 5 6 3 5 5 3 5 6 5 3 4 6 4 6 3 3 4 4 4 5 4 5 5 4 4 0 0 0 0 0 1 1
 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1
 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 2 1 1 1 1 1 0 0 0 0 2 2 2 2 2 1 1 0 0 2 2
 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 0 2 0 1 0 1 0 2 2 1 2 2 2 0 0 1 1 1 2 0 2 2 2 2 0 0 1 1 1 0 2 2 2
 0 2 2 2 2 1 1 0 1 1 2 0 2 2 0 1 2 2 1 2 2 0 1 1 1 2 2 2 1 2 0 1 0 1 2 0 2
 1 1 0 2 2 1 0 1 1 0 0 1 2 1 2 2 1 1 2 0 0 1 1 0 0 1 2 2 2 2 1 2 1 1 0 1 0
 1 2 0 1 0 1 2 2 0 2 0 2 1 0 1 1 1 2 0 2 2 0 1 2 2 1 2 2 2 0 2 2 1 1 1 2 2
 2 0 1 1 0 1 1 2 1 2 2 0 2 2 0 0 2 2 2 2 2 1 2 1 0 1 0 1 2 1 2 0 1 0 0 0 0
 1 2 1 0 1 2 1 2 2 2 1 2 0 0 2 0 2 2 1 2 1 0 0 0 2 0 1 2 1 2 1 0 1 1 2 2 1
 2 2 0 2 1 0 2 2 2 0 2 0 2 2 0 2 2 0 1 1 0 2 1 2 0 1 1 2 0 1 0 1 2 0 2 2 0
 2 1 2 0 1 2 0 2 0 1 0 2 2 0 0 2 1 0 2 0 0 2 0]
```



3.

3. **5 Points.** Compare your spectral clustering results with k-means. It is natural that on certain hard toy example, both method won't generate perfect results. In your report, briefly analyze what is the advantage or disadvantage of spectral clustering over k-means. Why it is the case? (You do not need to mathematically prove it but just need to give answers in your own language.)

In the comparison and analysis section, we will evaluate the performance of spectral clustering in contrast to K-means clustering. It is essential to note that spectral clustering can demonstrate superior results, particularly in scenarios involving intricate data structures.

This analysis underscores the advantages of spectral clustering in situations where clusters exhibit irregular shapes or when outliers are present in the dataset.