```
# k-Means demo
import numpy as np
import matplotlib.pyplot as plt
```

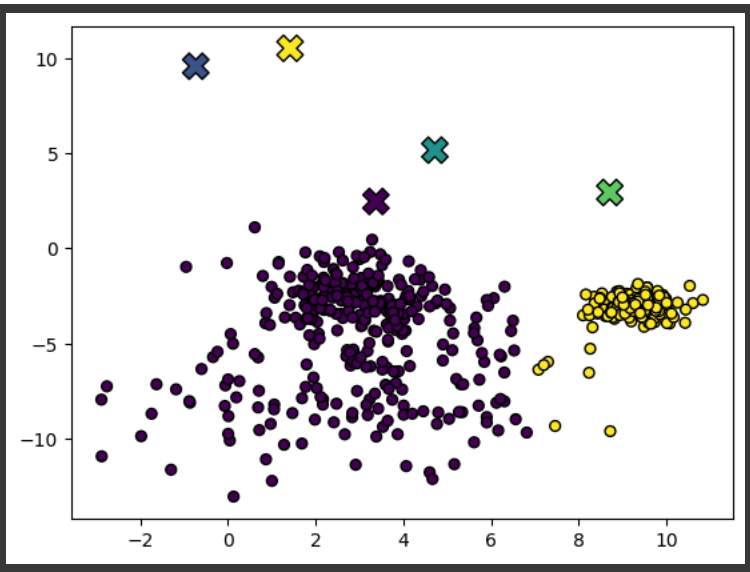
```
# Set the number of data vectors (n) and the dimension of the data space (m)
n = 12  # Example 1
# n = 1000 # random data
m = 2

# Set the number of clusters (k)
k = 5 # changed from 4 to 5 for q1 part b

# Initialize the data - either as in Example 1 or using random data
XData = np.array([[1/np.sqrt(2), 1/np.sqrt(2)],
                  [-1/np.sqrt(2), 1/np.sqrt(2)],
                  [1/np.sqrt(2), -1/np.sqrt(2)],
                  [-1/np.sqrt(2), -1/np.sqrt(2)],
                  [1/2, np.sqrt(3)/2],
                  [-1/2, np.sqrt(3)/2],
                  [1/2, -np.sqrt(3)/2],
                  [-1/2, -np.sqrt(3)/2],
                  [np.sqrt(3)/2, 1/2],
                  [-np.sqrt(3)/2, 1/2],
                  [np.sqrt(3)/2, -1/2],
                  [-np.sqrt(3)/2, -1/2]])
# XData = np.load("blobs.npy")
# n = len(XData) # Note: Added for Q1 part b
```

Question 1a part i:

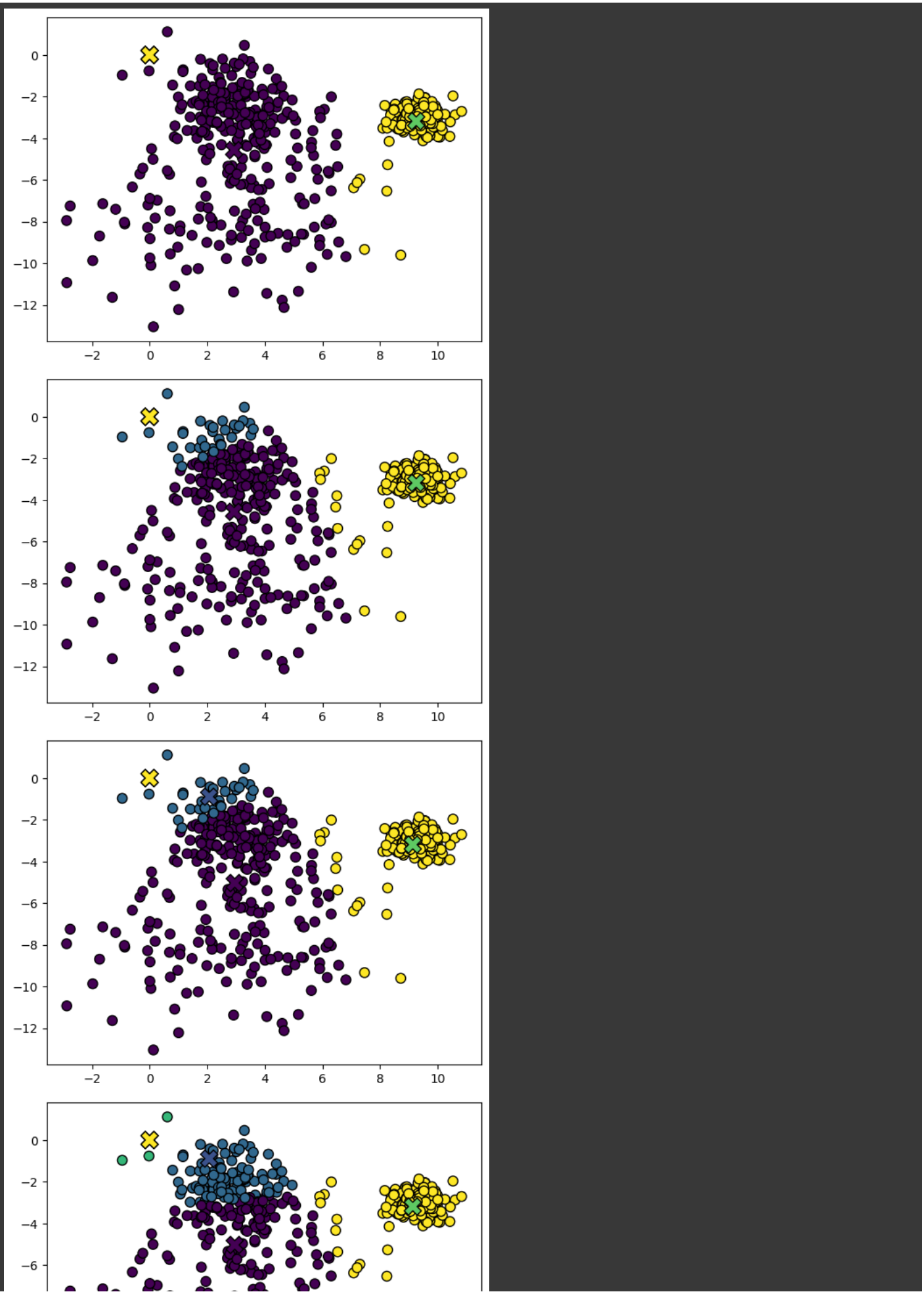
```
# XData = -1 * np.ones((n, m)) + 2 * np.random.rand(n, m)

# Create data structures to store the (randomly selected) representative vectors for cluster (c)

# Question 1 part a, i: Random Initialization
c = np.vstack([np.random.uniform(-2,12,k), np.random.uniform(-2,12,k)]).T
```

```
# Create a data structure to store closest representative vector for each data point
closestCluster = np.zeros(n)
# Assign each data vector to the new, closest cluster
for d in range(n):
    # Store the coordinates of the current data vector
    xD = XData[d, :]

    # Set the minimum distance tracker to be a very large number
    sqDistMin = 1e16

    # Find the closest representative vector (cluster) to the current data vector
    for i in range(k):
        sqDist = np.linalg.norm(c[i, :] - xD, ord=2)

        # If the distance is less than the current min, assign the
        # current data vector to this cluster
        if sqDist < sqDistMin:
            closestCluster[d] = i
            sqDistMin = sqDist
```

```
# Update the assignments of the data vectors to their new clusters
IndexSet = closestCluster.astype(int)

# Plot the data
plt.scatter(XData[:, 0], XData[:, 1], s=32, c=IndexSet, cmap='viridis', edgecolors='k', marker='o')
plt.scatter(c[:, 0], c[:, 1], s=200, c=np.linspace(1, k, k), cmap='viridis', edgecolors='k', marker='X')
plt.show()
```


10
5
0
−5
−10
−2
0
2
4
6
8
10

```
# Create data structures to store the representative vectors from the previous iteration (cPrev)
cPrev = np.copy(c)

# The Alternating Minimization Scheme
doneFlag = False

# Keep alternating updates to representative vectors and cluster assignments until representative vectors no longer change their location
while not doneFlag:
    # Update the representative vectors in each cluster via the centroid formula
    for i in range(k):
        # Find the indices for all data vectors currently in cluster i
        ClusterIndices = np.where(IndexSet == i)[0]

        # Find the number of data vectors currently in cluster i
        NumVecsInCluster = len(ClusterIndices)

        # Create a data structure to store representative vector for the current cluster
        c[i, :] = np.zeros(m)

        # Update cluster vector using the centroid formula
        for j in range(NumVecsInCluster):
            c[i, :] += XData[ClusterIndices[j], :] / NumVecsInCluster

    # Plot the updated representative vectors for each cluster
    plt.scatter(XData[:, 0], XData[:, 1], s=64, c=IndexSet, cmap='viridis', edgecolors='k', marker='o')
    plt.scatter(c[:, 0], c[:, 1], s=200, c=np.linspace(1, k, k), cmap='viridis', edgecolors='k', marker='X')
    plt.show()

    # Now reassign all data vectors to the closest representative vector (cluster)
    # Create a data structure to store closest representative vector for each data point
    closestCluster = np.zeros(n)

    # Reassign each data vector to the new, closest cluster
    for d in range(n):
        # Store the coordinates of the current data vector
        xD = XData[d, :]

        # Set the minimum distance tracker to be a very large number
        sqDistMin = 1e16

        # Find the closest representative vector (cluster) to the current data vector
        for i in range(k):
            sqDist = np.linalg.norm(c[i, :] - xD, ord=2)

            # If the distance is less than the current min, assign the
            # current data vector to this cluster
            if sqDist < sqDistMin:
                closestCluster[d] = i
                sqDistMin = sqDist

    # Update the assignments of the data vectors to their new clusters
    IndexSet = closestCluster.astype(int)

    # Plot the data and the updated representative vectors
    plt.scatter(XData[:, 0], XData[:, 1], s=64, c=IndexSet, cmap='viridis', edgecolors='k', marker='o')
    plt.scatter(c[:, 0], c[:, 1], s=200, c=np.linspace(1, k, k), cmap='viridis', edgecolors='k', marker='X')
    plt.show()

    # Terminate the alternating scheme if the representative vectors are unaltered
    # relative to the previous iteration
    if np.array_equal(c, cPrev):
        doneFlag = True
    else:
        cPrev = np.copy(c)
```


0
−2
−4
−6
−8
−10
−12
−2
0
2
4
6
8
10