COMP3670/6670 Programming Assignment 2 - Clustering and Vector Calculus

Enter Your Student ID: u7076437

Your Name: Zhichao Tu

Deadline:

Submit: Write your answers in this file, and submit a single Jupyter Notebook file (.ipynb) on Wattle. Rename this file with your student number as 'uXXXXXXX.ipynb'.

Enter Discussion Partner IDs Below: You could add more IDs with the same markdown format above.

Programming Section:

- 1.1: 15%
- 1.2: 20%
- 2.1: 10%
- 2.2: 15%
- 2.3: 15%
- 2.4: 10%
- 2.5: 10%
- 2.6: 5%

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
from IPython.core.display import HTML

np.random.seed(1)
```

Task1: Vector Calculus

This part is about vector calculus. In this section, we will use the rigorous definition of the derivative to calculus it.

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

Now, expand it to vectors.

Task 1.1: Calculate the gradient of $f(\mathbf{x}) = \mathbf{x}^T \mathbf{a}$ respect to $\mathbf{x}$. $\mathbf{x}, \mathbf{a} \in \mathbb{R}^N$

In [20]:
```
N = 10
x = np.random.rand(N)
a = np.random.rand(N)
```

In [21]:
```
def derive_function1(x, a, N):
    # The answer is a vector
    # Please follow the rigorous defination of derivative to answer this question
    # Directly return the conclusion from textbook will not receive any mark.

    h = 1e-4
    # Your Code Here
    d = np.zeros(N)
    for i in range(N):
        x_h = np.copy(x)
        x_h[i] += h
        di = ((x_h @ a) - (x @ a)) / h
        d[i] = di
    return d
```

Task 1.2: Calculate the gradient of $f(\mathbf{x}) = \mathbf{x}^T B\mathbf{x}$ respect to $\mathbf{x}$. $\mathbf{x} \in \mathbb{R}^N$, $B \in \mathbb{R}^{N \times N}$

In [22]:
```
x = np.random.rand(N)
B = np.random.rand(N, N)
```

In [23]:
```python
def derive_function2(x, B, N):
    # The answer is a vector
    # Please follow the rigorous defination of derivative to answer this question
    # Directly return the conclusion from textbook will not receive any mark.

    h = 1e-5
    # Your Code Here
    d = np.zeros(N)
    for i in range(N):
        x_h = np.copy(x)
        x_h[i] += h
        di = (((x_h @ B) @ x_h) - ((x @ B) @ x)) / h
        d[i] = di
    return d
```

Task2: Clustering

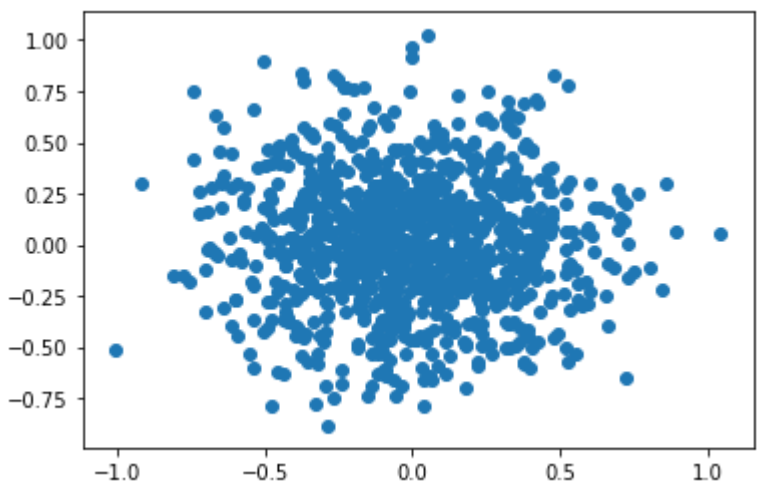
These programming exercises will focus on K-means clustering.

If you're unsure of how k-means works, read this very helpful and freely available online breakdown from Stanford's CS221 course; https://stanford.edu/~cpiech/cs221/handouts/kmeans.html (https://stanford.edu/~cpiech/cs221/handouts/kmeans.html)

This assignment requires you to loosely interpret how k-means is a specific case of a more general algorithm named Expectation Maximisation. This is explained toward the end of the above article.

First, lets loading the dataset.

In [24]:
```python
X = np.load("./data.npy")
plt.scatter(X[:,0], X[:,1])
plt.show()
```



The dataset contains 1000 4-dimensional samples. However, we don't know how many centroids it contains. The number of centroids is more than 5 but less than 10. We need to figure it out in the clustering procedure.

K-means is a special, simple case of the Expectation Maximisation (EM) algorithm.

This simplified EM (k-means), is divided into two steps.

The **E-Step**, where for every sample in your dataset you find which "centroid" that datapoint is closest to that sample, and record that information.

The **M-Step**, where you move each "centroid" to the center of the samples which were found to be closest to it in the **E-Step**.

Each *centroid* is simply an estimated mean of a cluster. If you have 1 centroid, then this centroid will become the mean of all your data.

If each of your samples, such as the 400 you generated in the previous question, are of dimension n, then each of your centroids will be of dimension n.

Centroids are initially random values, and the k-means algorithm attempts to modify them so that each one represents the center of a cluster.

TASK 2.1: Write a function $initialise\_parameters(m, n, X) = C$ which generates m centroids, each of dimension n, and stores them in a matrix $C \in \mathbb{R}^{m \times n}$.

No two centroids should be the same, and **must not** be hard coded. Generate these parameters using a sensible initialisation method such as those described in the first link below. You will be judged based on whether the method you choose is sensible and likely to result in kmeans converging to good result.

HINT:

- https://en.wikipedia.org/wiki/K-means_clustering#Initialization_methods (https://en.wikipedia.org/wiki/K-means_clustering#Initialization_methods)
- https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.random.randint.html (https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.random.randint.html)

In [25]:
```
def initialise_parameters(m, n, X):
    C = np.zeros((m,n))
    # Your Code Here
    # randomly shuffle the copy of X
    random_x = np.copy(X)
    np.random.shuffle(random_x)
    # pick the first m rows and n columns from the shuffled copy, thus avoid the duplicates
    C = random_x[0:m, 0:n]
    return C

C = initialise_parameters(2, 4, X)
print(C)
```

```
[[-0.08215101  0.11568247  1.0679666  -0.50048716]
 [-0.23389569 -0.09793914 -1.9299073  -1.8883966 ]]
```

Now we implement k-means.

TASK 2.2: Create a function $E\_step(C, X) = L$, where L is a matrix of the same dimension of the dataset X.

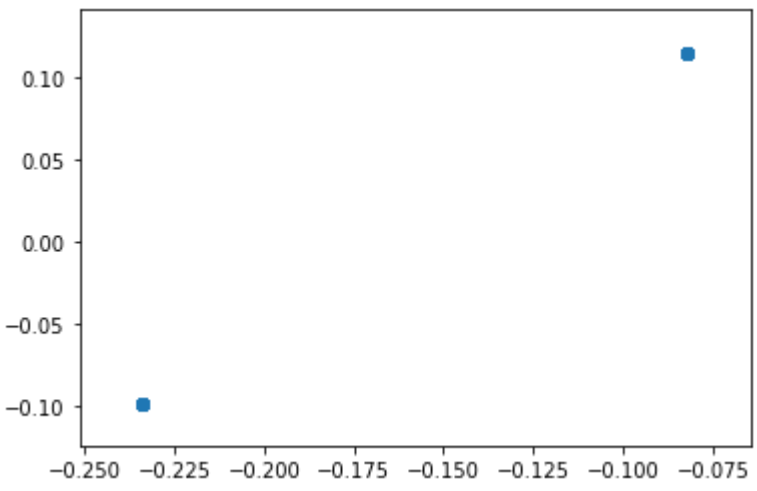
This function is is the **E-Step** (or "assignment step") mentioned earlier.

HINT:

- https://stanford.edu/~cpiech/cs221/handouts/kmeans.html (https://stanford.edu/~cpiech/cs221/handouts/kmeans.html)
- https://en.wikipedia.org/wiki/K-means_clustering#Standard_algorithm (https://en.wikipedia.org/wiki/K-means_clustering#Standard_algorithm)
- Each row of L is a centroid taken from C.

In [26]:
```
def E_step(C, X):
    L = np.zeros(X.shape)
    # Your Code Here
    # iterate through all datas in X
    for ind in range(X.shape[0]):
        data = X[ind]
        # array of distances between data and each centroid
        data_centroids_dists = np.sum((C - data) ** 2, axis = 1)
        # find the centroid which gives the minimum distance
        min_centroid = C[np.argmin(data_centroids_dists)]
        # assign the centroid to the corresponding data
        L[ind] = min_centroid
    return L

L = E_step(C, X)
plt.scatter(L[:, 0], L[:, 1])
plt.show()
```



TASK 2.3: Create a function $M\_step(C, X, L) = C$ which returns C modified so that each centroid in C is placed in the middle of the samples assigned to it. This is the **M-Step**.

In other words, make each centroid in C the average of all the samples which were found to be closest to it during the **E-step**. This is also called the "update step" for K-means.

HINT: https://docs.scipy.org/doc/numpy/reference/generated/numpy.array_equal.html (https://docs.scipy.org/doc/numpy/reference/generated/numpy.array_equal.html)

In [27]:
```
def M_step(C, X, L):
    # Your Code Here
    # iterate through all centroids in C
    for ind in range(C.shape[0]):
        centroid = C[ind]
        sum_r = np.zeros(centroid.shape)
        count_r = 0
        for r_ind in range(L.shape[0]):
            # find all the datas that belong to the centroid's cluster
            if np.array_equal(L[r_ind], centroid):
                sum_r += X[r_ind]
                count_r += 1
        # if the centroid's cluster is not empty, update the centroid
        if count_r > 0:
            C[ind] = sum_r / count_r
    return C
```

TASK 2.4: Implement $kmeans(X, m, i) = C, L$ which takes a dataset X (of any dimension) and a scalar value m, and uses the previous 3 functions you wrote to:

- generate m centroids.
- iterate between the E and M steps i times (ie, it iterates i times) to classify the m clusters.

...and then returns:

- C, the centers of the m clusters after i iterations.
- L, the labels (centroid values) assigned to each sample in the dataset after i iterations.

In [28]:
```
def kmeans(X, m, i):
    L = np.zeros(X.shape)
    C = np.zeros((m, X.shape[1]))
    # Your Code Here
    n = X.shape[1]
    C = initialise_parameters(m, n, X)
    for iteration in range(i):
        L = E_step(C, X)
        C = M_step(C, X, L)
    return C, L
```

Task 2.5: The following code is to display the result. However, due to the limitation of our visualization tools, it can only presents the data in the two dimensional space. While the dimension of the dataset is 4, we really want to visualize the data in the two dimensional figure. Besides, the number of centroids is not determined yet.

This task is ask you to modify the following code so as to give the best visualization effect.

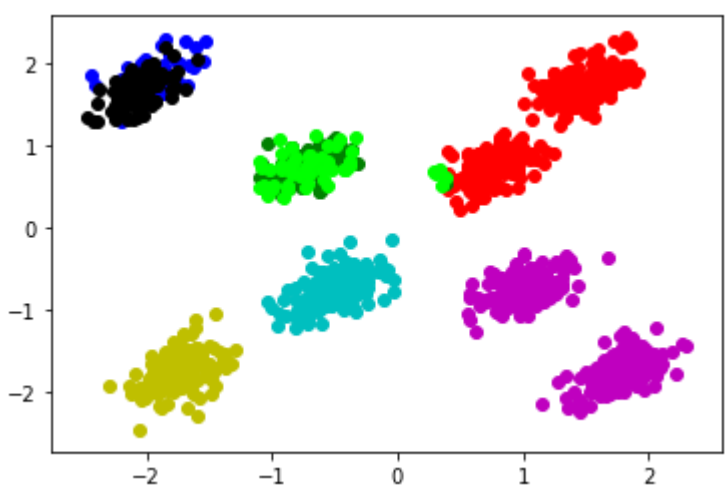
HINT: You only need to change "number of centroid", "dimension1", "dimension2" to a number, which are quoted by "#" in the following code.

In [29]:
```
m = 8
i = 100
#CODE TO DISPLAY YOUR RESULTS.
C_final, L_final = kmeans(X, m, i)
print('Initial Parameters:')
print(C)
print('\nFinal Parameters:')
print(C_final)

def allocator(X, L, c):
    cluster = []
    for i in range(L.shape[0]):
        if np.array_equal(L[i, :], c):
            cluster.append(X[i, :])
    return np.asarray(cluster)

colours = ['r', 'g', 'b', 'y', 'c', 'm', 'k', 'lime', 'wheat', 'fuchsia', 'pink']
for i in range(m):
    cluster = allocator(X, L_final, C_final[i, :])
    plt.scatter(cluster[:,2],
                cluster[:,3],
                c=colours[i])
plt.show()
```

```
Initial Parameters:
[[-0.08215101  0.11568247  1.0679666  -0.50048716]
 [-0.23389569 -0.09793914 -1.9299073  -1.8883966 ]]
Final Parameters:
[[-1.29773432e-02  6.76570468e-03  1.13299061e+00  1.25319890e+00]
 [ 1.08990353e-01 -2.05048054e-01 -7.09552498e-01  7.59647554e-01]
 [-2.63741195e-01 -1.01372712e-01 -1.95439086e+00  1.85097060e+00]
 [ 1.80182894e-02 -2.84548382e-02 -1.73314207e+00 -1.74422749e+00]
 [-1.15362482e-03  3.56500354e-02 -5.02332315e-01 -7.41382419e-01]
 [-1.96292884e-02  2.13094467e-02  1.38462173e+00 -1.23177286e+00]
 [ 1.52120058e-01  8.61174692e-02 -2.05431167e+00  1.68016846e+00]
 [-2.05454759e-01  2.85470637e-01 -6.53870070e-01  7.30059592e-01]]
```



TASK 2.6: Use your own words to explain how you found the number of centroids in Task 2.5 and how you might do this in the real world.

Answer:

According to the information given, the number of centroids is more than 5 but less than 10. Based on the mechanism of the K-means, more centroids will provide a lower cost function. So I started trying with 9 centroids. Then with the 4-dimensional dataset, I tried different combination of the two chosen dimensions that might provided a clearer visualization of clustering which can more likely representing the real 4-dimensional situation. The third and forth dimension of the dataset seemed to provided a clearer clustering. Even though sometimes this k-means method may provide a bad local optima, but still it shows a clear 8 clusters given the chosen two dimensions. Hence I chose the number of centroids as 8 in Task2.5.

Normally with K no more than 10, after initialize K-means multiple times it can provide a good local or global optima. Hence we can choose the clustering that gives lowest cost function. While with K much larger than 10, the first random initialization will more possible to provide a relatively good solution. Sometimes using a Elbow Method by visualizing the relation between the number of centroids and its corresponding cost function can help identify the elbow and a resonable number of centroids. But still, the number of centroids might need to be chosen manually by looking at the visualizations or the output of the algorithm. Also, it is helpful to consider the real-world purpose of the K-means algorithm and then choose the reasonable number of centroids that serve the purpose well.

In [ ]:

In [ ]: