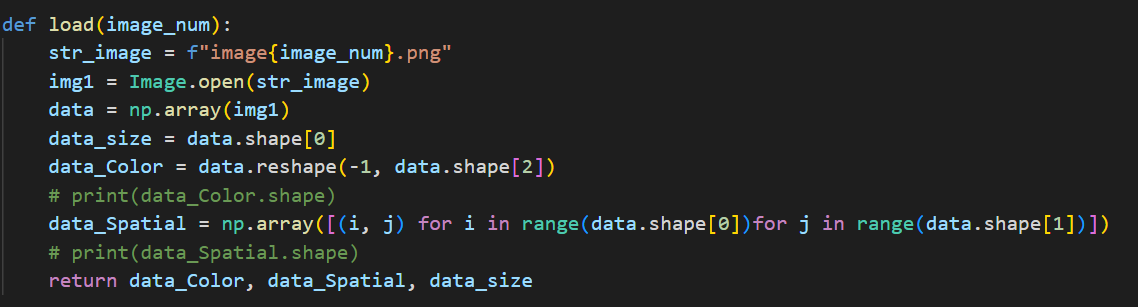
HW6 311552055 蕭育泓

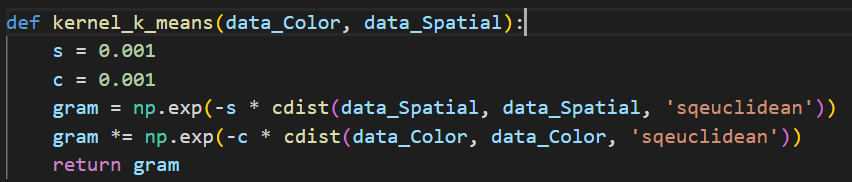
Part a. code explanations

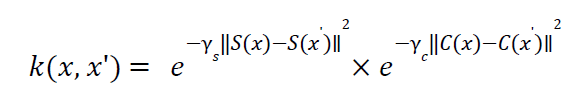
First, we need to load the data about the data\_color(10000\*3) ,the data\_spatial(10000\*2) and the data\_size = 100



**Part 1**

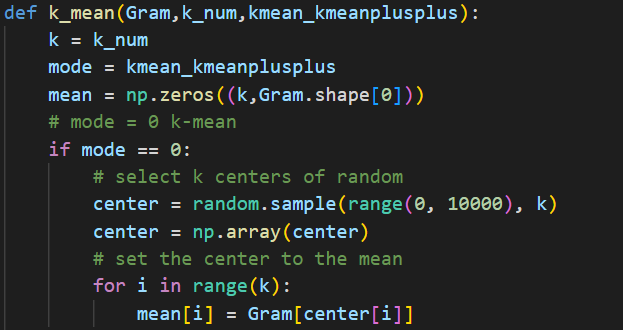
kernel k-means

****

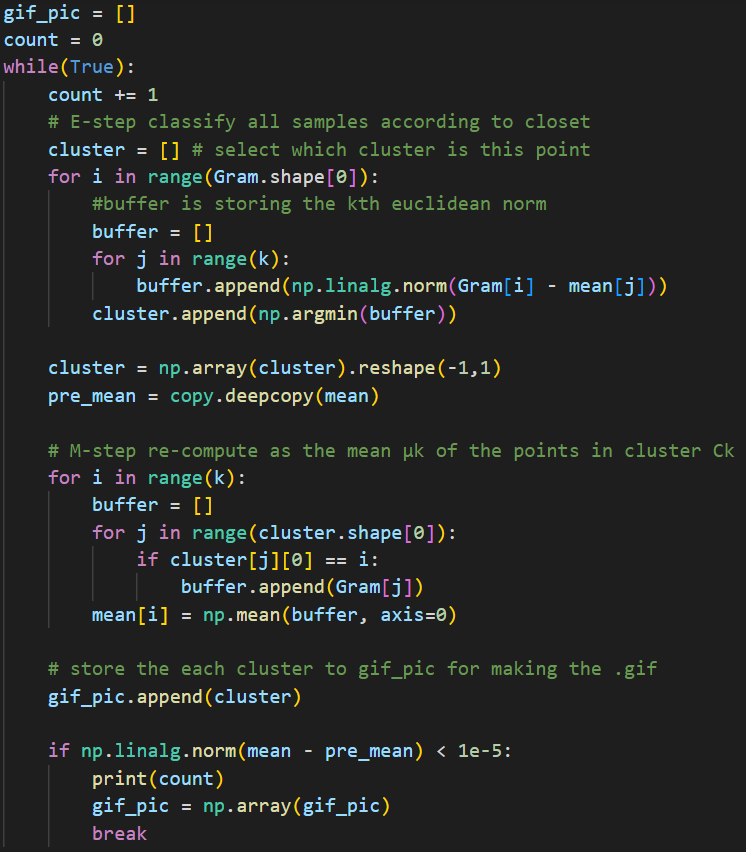
Using the data\_color and data\_spatial to find out the gram matrix using the formula 

**In the k-mean.py**

Function - k\_mean



Picture1



Picture2

Doing the k mean by using the random select mean in the beginning(Picture 1). We will repeat the step E (classify the point in which group) and step M (compte the new mean point) until the mean and pre\_mean is converge(smaller than 1e-5). I also store the classify result for each loop.(Picture 2)

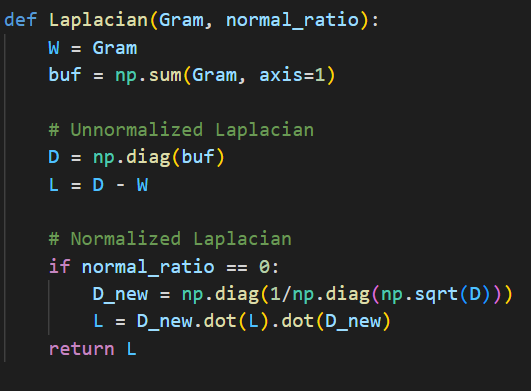
**In the spectral\_clustering.py**

Using some of the same functions in the k-mean.py like function -kernel\_k\_means, function – kmean to help me to generate the result.

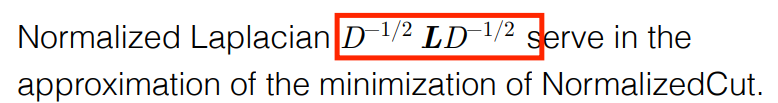
The most different things are we need to compute the Laplacian matrix

with normal cut or ratio cut and find out the eigenspace.

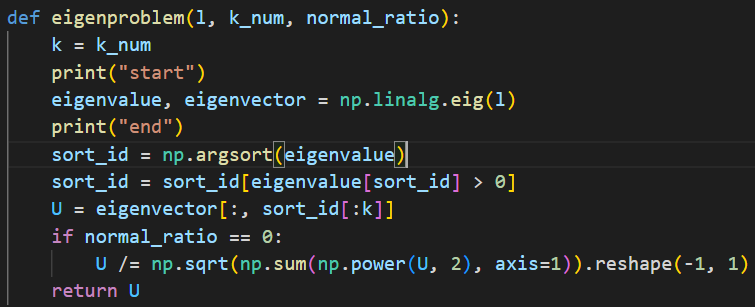
Function - Laplacian



The Function – Laplacian, the Laplacian matrix(L) is be find from the matrix W and matrix D, the matrix W is the weight of the each edge which can find from the function-kernel\_k\_means and the matrix D is the diagonal matrix which which can find from the matrix D. If we select the normal cut, the Laplacian matrix(L) is represent as D-W. If we select the ratio cut, we will use formula below to find out the Laplacian matrix.



Function - eigenproblem



The Function – eigenproblem is used to find out the eigenspace with non-zero eigenvalue. If we select the normal cut , we need to normalizing the rows to norm 1 by the formula below.



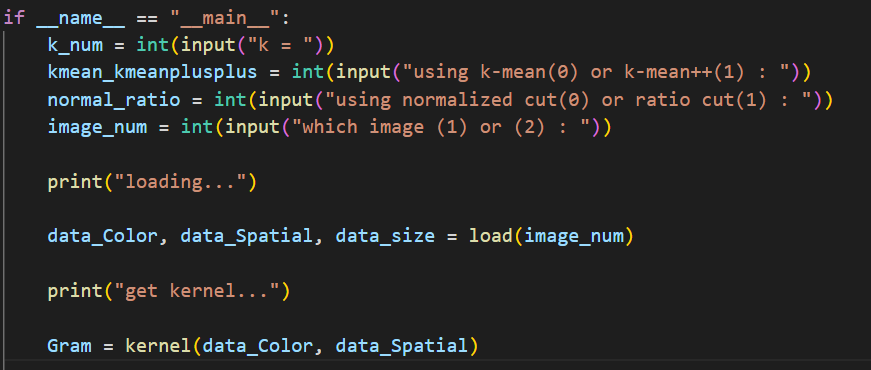
We will find the clusters like kmean.py in spectral\_clustering.py with the reuse functions. Both of the .py will make the .gif to show the result.

The function – make\_gif 

**Part 2**

The different between part1 and part2 is the number of clusters(k), so in the main function we can select the value k what we want to be.

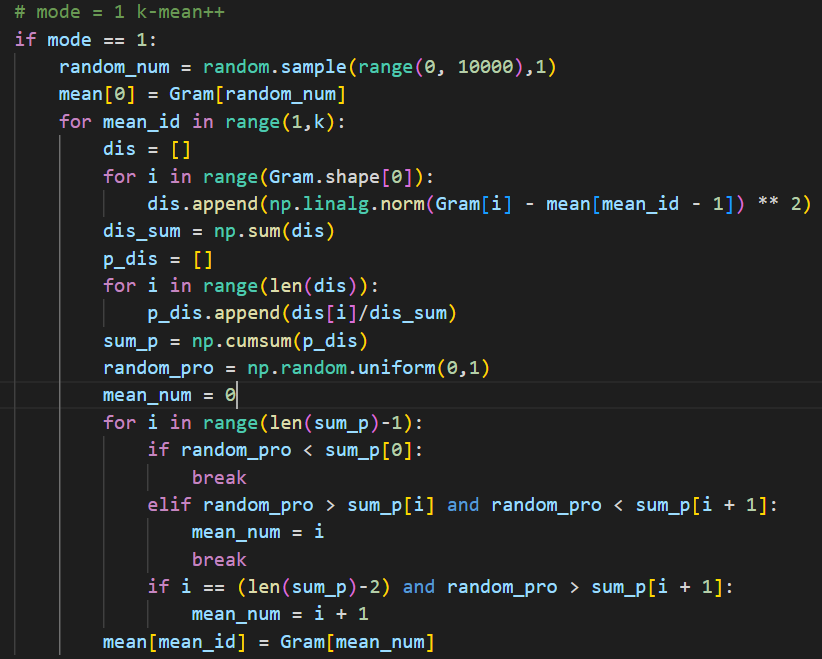
In addition, we are same as part1.



**Part 3**

We try the different way k-means++ to show corresponding results for initialization of k-means clustering it can be select from the main function (the main function is in the part2).

The function – kmean is also express the way k-means++

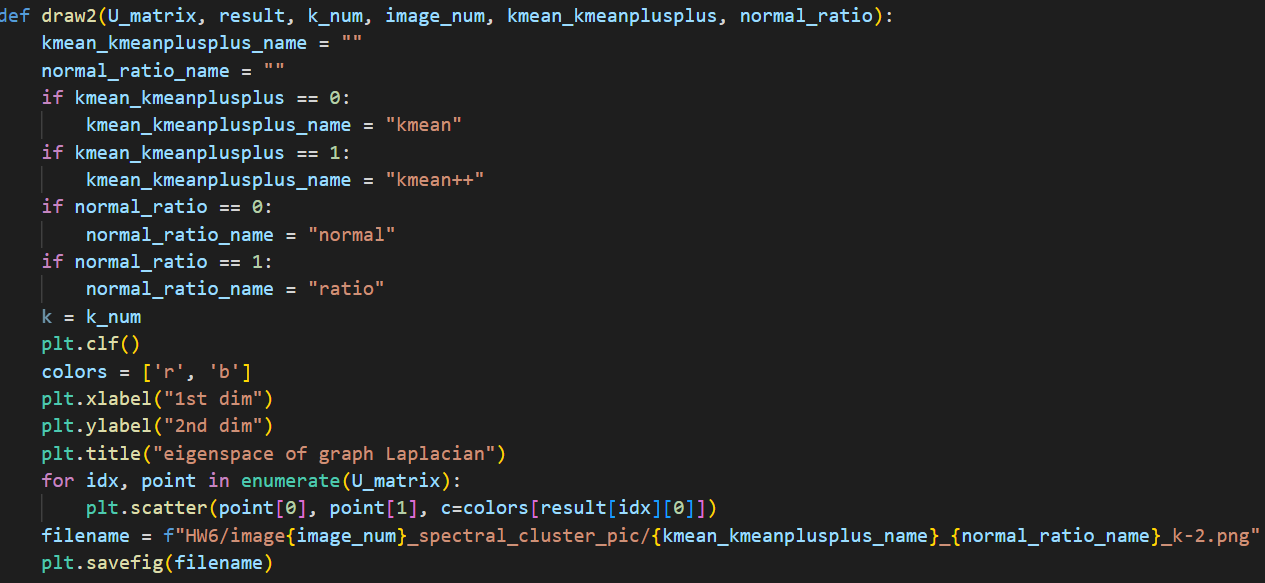


First, we select a random mean in the beginning and then compute the distance between each point to the mean, accumulated value to find out the interval. We will random choose the number between 0 – 1, the more far from the mean has the more probability to be the next mean cluster.

**Part 4**

I will show the picture to example the same coordinates in the eigenspace of graph Laplacian for spectral clustering when k is 2 and 3.

The function-draw2



The function-draw3



I will set the colors to show the different points and save the picture in the file.

Part b. results & discussion

**In the k-mean.py results:**

**Image1**

****

|  |  |  |  |
| --- | --- | --- | --- |
|  | **K = 2** | **K = 3** | **K = 4** |
| **random** |  |  |  |
| **Kmean++** |  |  |  |

**Image2**

****

|  |  |  |  |
| --- | --- | --- | --- |
|  | **K = 2** | **K = 3** | **K = 4** |
| **random** |  |  |  |
| **Kmean++** |  |  |  |

**In the k-mean.py discussion:**

1. There were get the same result in random and kmean++.
2. In the picture2, we can observe when k = 3, we get the more clear output which mean kmean++ has more stability. It is because the random case choose the cluster from the random.
3. In the picture1, we can discover that the land and the sea is far different color in human eyes, but the computer think the cluster is between the two different color of the ocean. It is because of the kernel function and the color weight.

**In the spectral\_clustering.py results:**

**Image1**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Random | K = 2 | Examine point | K = 3 | Examine point |
| Normalized cut |  |  |  |  |
| Ratio cut |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kmean++ | K = 2 | Examine point | K = 3 | Examine point |
| Normalized cut |  |  |  |  |
| Ratio cut |  |  |  |  |

**Image2**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Random | K = 2 | Examine point | K = 3 | Examine point |
| Normalized cut |  |  |  |  |
| Ratio cut |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kmean++ | K = 2 | Examine point | K = 3 | Examine point |
| Normalized cut |  |  |  |  |
| Ratio cut |  |  |  |  |

**In the spectral\_clustering.py discussion:**

1. We can observe the worst result is the set of ratio cut and k=2 in image2.
2. There are no different between Normalized cut and ratio cut most of the time.
3. The k is higher and there is more different between Normalized cut and ratio cut.
4. There is same as k-mean.py, kmean++ has a little bester output than kmean.
5. In the examine point, there are useless in image2 ratio cut either random or kmean++.
6. The normalized cut is always differ to ratio cut.
7. In image1, the normalized cut in kmean++ and ratio cut in random output image are same in k=2.

Part c. observations

1. The execution time in spectral\_clustering.py is always about 10 more minutes that is far longer than k-mean.py.
2. When using the kernel kmean++, the execution time is a liitle more than using the random kernel.
3. When execute the spectral\_clustering.py, I discover the converge time is less than k-mean.py.
4. The execution time in spectral\_clustering.py takes the most of its time to compute the eigenvector and eigenvalue.