**Machine Learning**

Homework 6

Report

**Name**: Sergiy Shelekh

**ID**: 0760822

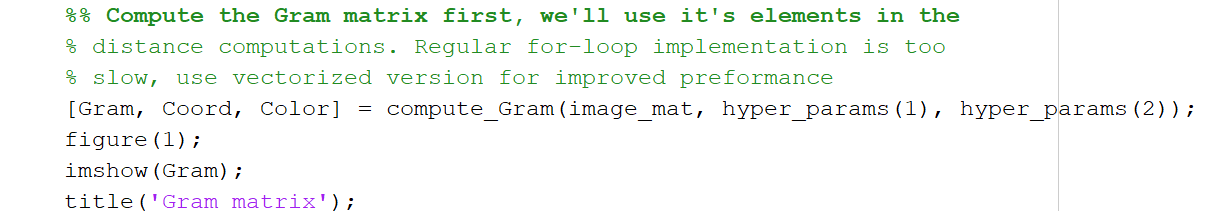
**Email**: [proxitrone@gmail.com](mailto:proxitrone@gmail.com)

**Introduction**

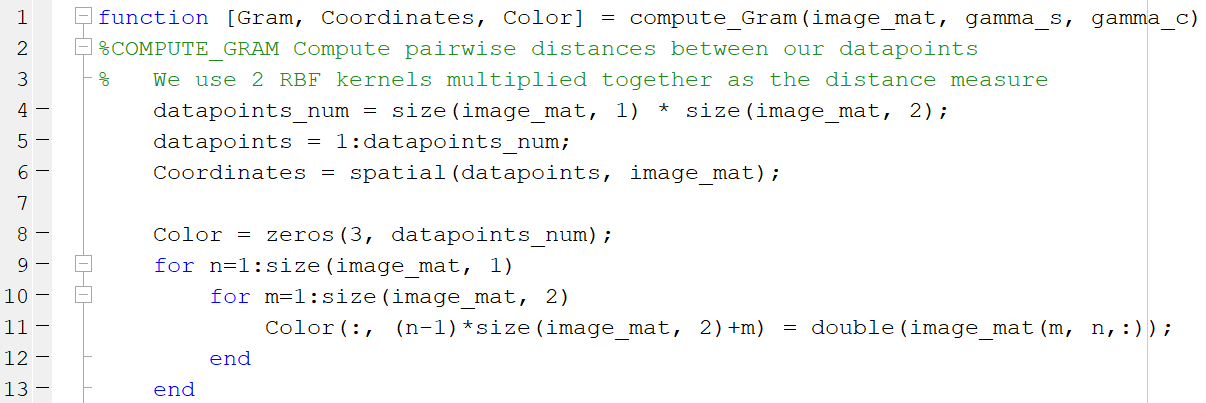
­The goal of this homework is to apply Kernel K-means and Spectral clustering techniques to perform image segmentation on given 2 images. As a similarity measure, we are given a kernel, which considers both spatial and color characteristics of our image data. Both Kernel K-means and Spatial clustering use K-means clustering, which is very sensitive to initialization, we need to show and compare different initialization strategies. For every clustering technique we use, it is required to show the clustering process in the form of GIF images, we should also see, that points within the same cluster have similar coordinates in the eigenspace of graph Laplacian, for spectral clustering. In this assignment I use MATLAB programming language, as it allows fast and efficient computations of things like Gram matrix and has good debugging capabilities.

**Kernel K-means: Implementaion**

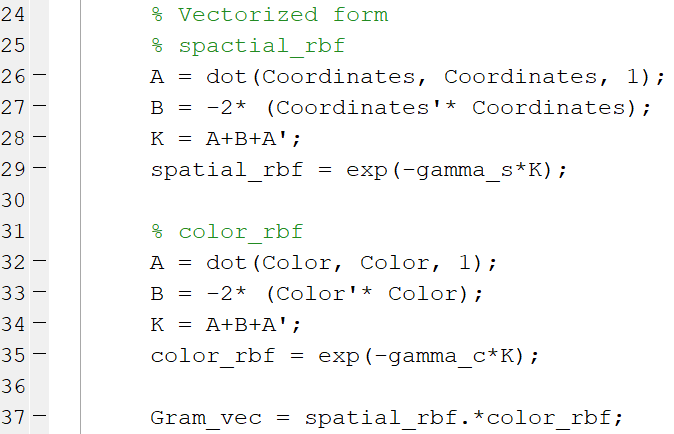
In kernel K-means, the most important thing is to compute the Gram matrix, a matrix of pairwise distances between our datapoints.



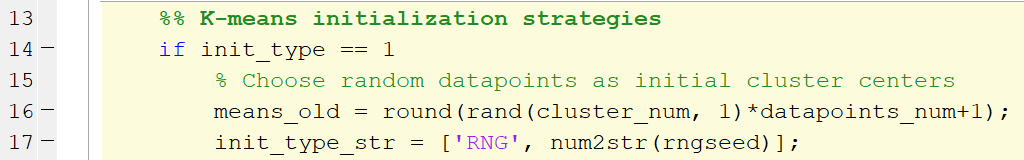
We start off with extracting spatial and color information from a given image



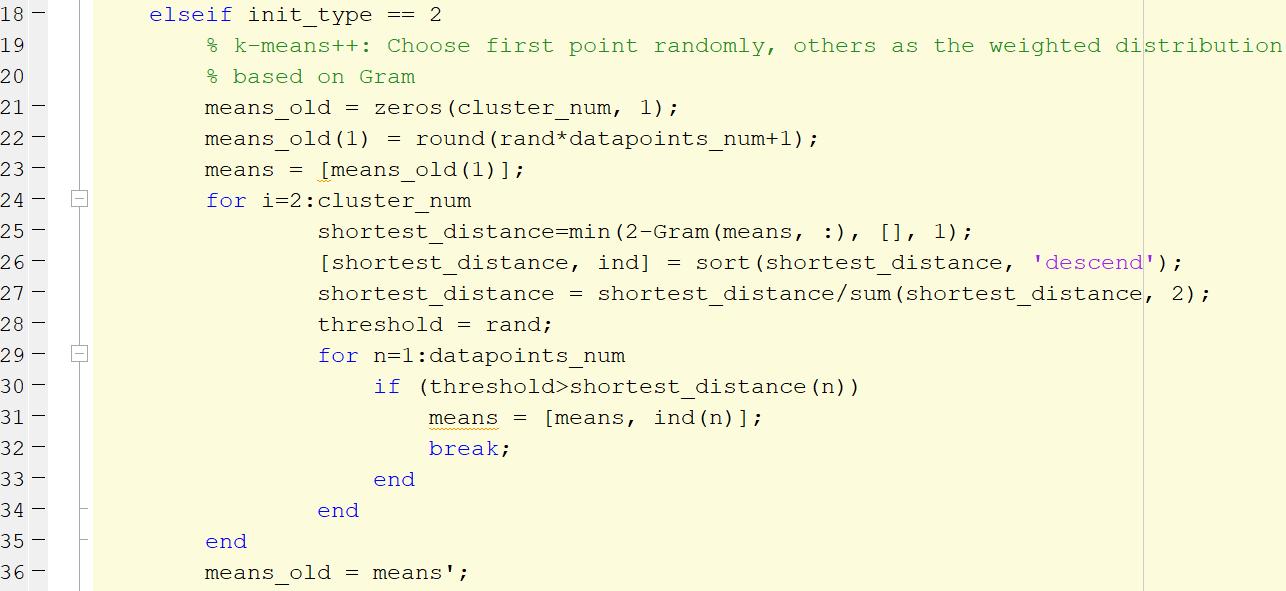
What we do here is creating a matrix, where rows contain spatial or color information (x, y coordinates, or R G B values) and columns represent each datapoints (10 000 in our case of 100x100 image). Next, for each RBF kernel, we take advantage of vectorization and expanding the 2-norm, in order to make computations much faster.



This computation still takes some time, but we only need to compute this matrix once for each image. Having Gram matrix computed, we proceed to initialization of K-means on this data. In this homework, I use two different initialization strategies:

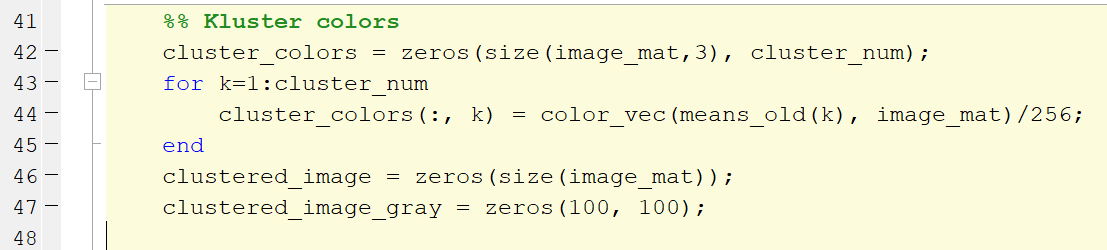


Random initialization, where we choose each cluster mean uniformly from all datapoints, and k-means++

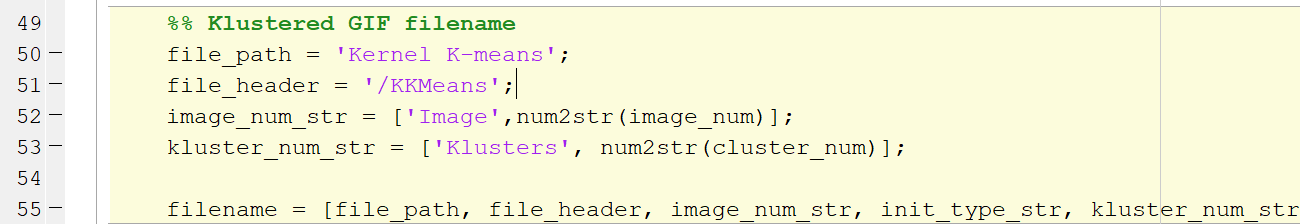


In k-means++ we only choose the first cluster mean randomly, and then draw other means from a weighted probability distribution, based on the square of the distances from datapoints to their closest mean.

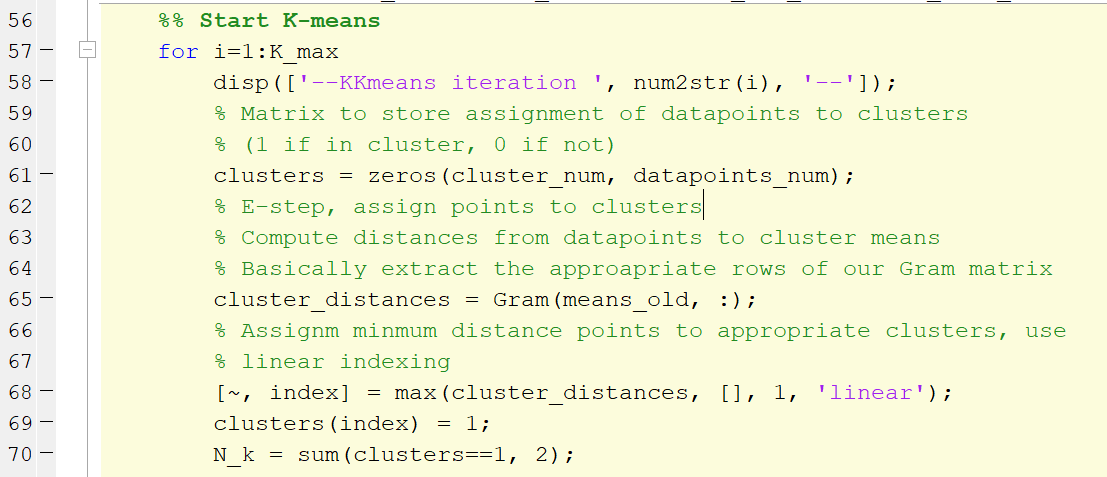
Before we start assigning datapoints to clusters, we still need to do some initialization. First, we record colors of our cluster means, for visualization.

****

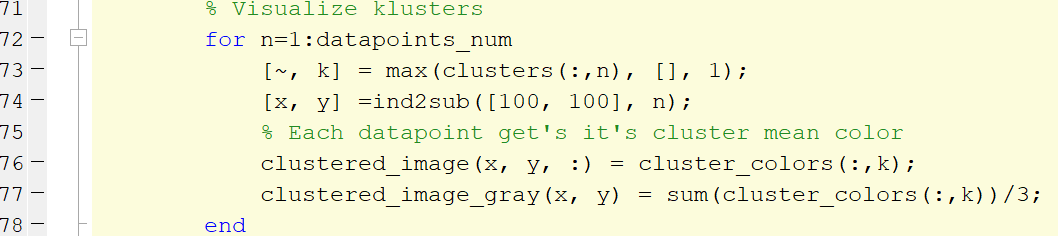
Assign a name to the GIF file we are going to store as our clustering process visualization.



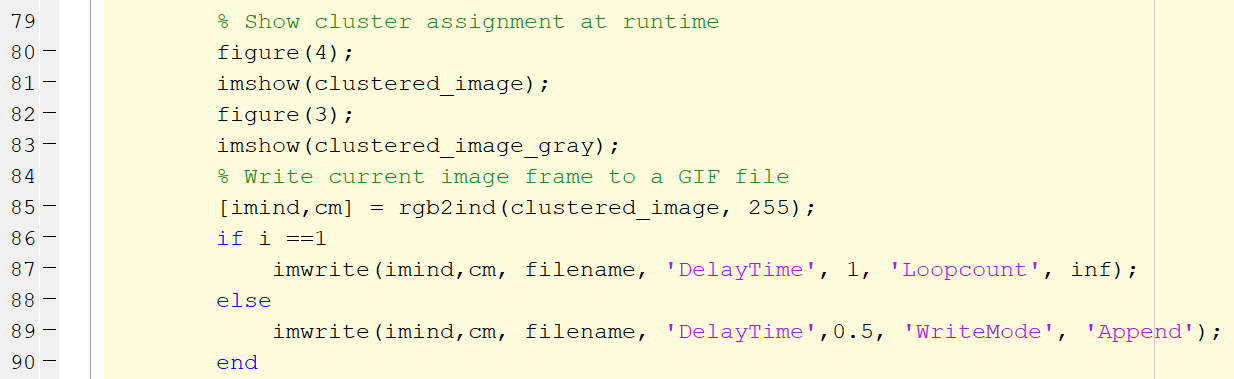
Now we have everything we need to start K-means clustering alrorithm



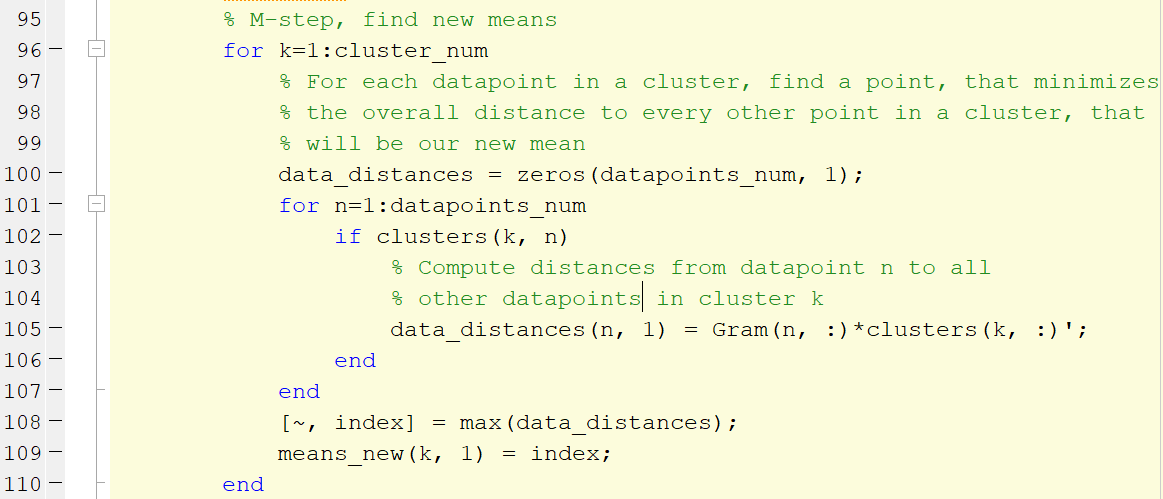
We start with E-step, where we assign datapoints to closest clusters. This is very easy, since we already have our Gram matrix precomputed and it has exactly what we need, those distances. So we, basically, extract those distances to clusters from our Gram matrix and choose the smallest distance among them, for each datapoint (We take the maximum of Gram matrix entries, since our RBF kernels give us larger values, the closer points are to each other). The largest kernel distance between a datapoint and cluster mean gets that datapoint assign to the appropriate cluster. E-step is done, all datapoints are assigned to clusters, we now do the visualization part.



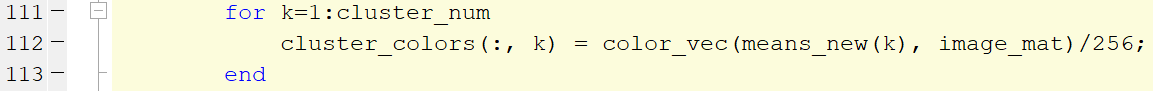
We first assign an appropriate color to each datapoint in a 100x100 image, then show the RGB and grayscale images during the process, and finally write GIF images for visualization.



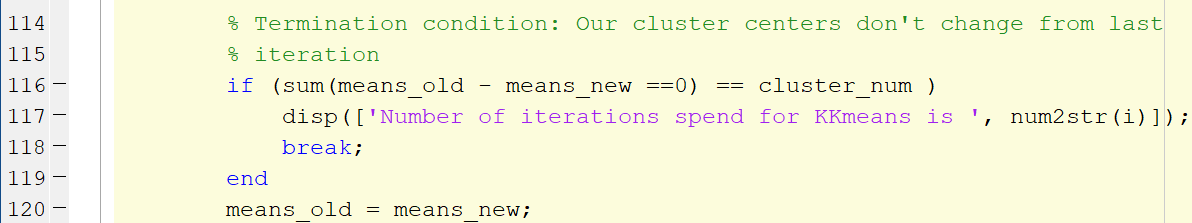
Now we need to proceed to the next part of K-means, which is the M-step, where we reassign cluster means.



Since we work in a high dimensional feature space indirectly via our kernel function, we can’t simply average cluster points, so we compute the overall distance for each point in a cluster to all other points in that cluster, and choose the new mean to be a point with the minimum distance (maximum in our case, since RBF kernel is used). This process gives us new cluster means, so we also need to reassign our cluster colors for visualization purposes.

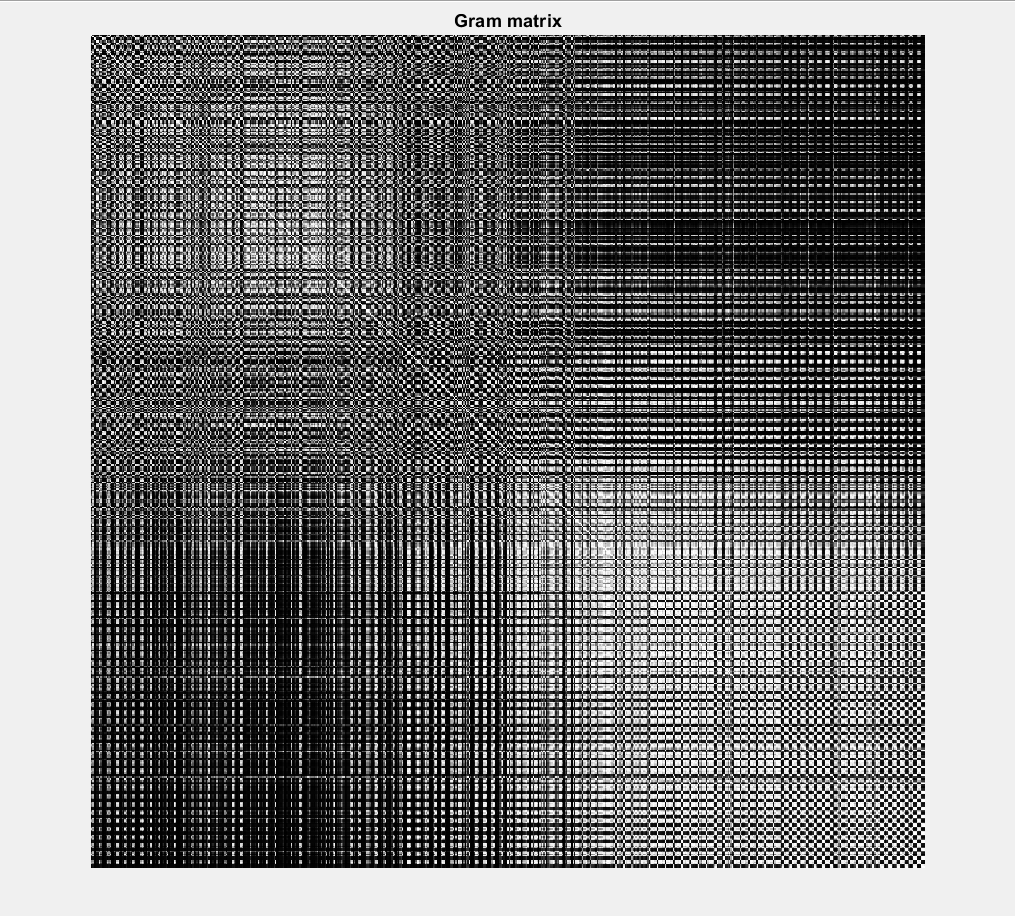


Finally, we check whether our termination condition has been reached, which is that the cluster means remain the same from previous iteration of K-means.



**Kernel K-means: Results**

Experiments are run for different images, numbers of clusters, initialization strategies. Hyperparameters were pre-tuned in a heuristic manner, so that our Gram matrix has a more-or-less block-diagonal structure. Hyperparameters we use are [] , since we want the color component of our image influence the kernel distance more than spatial component. First, look at gram matrix for image 1:



We can argue, that 3 clusters can be seen in this Gram matrix, it depends on the Kernel we use and it’s hyperparameters

Final clustering images are shown below, the GIF versions can be found in the folder Kernel K-means

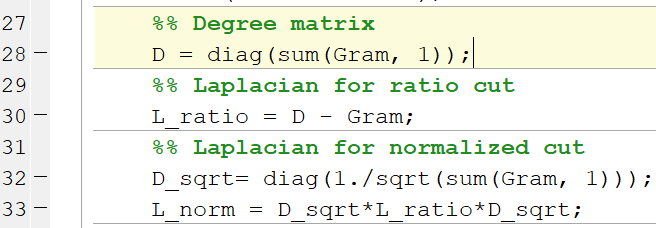
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Image 1** | 2 Cluster | 3 Clusters | 5 Clusters | 10 Clusters | Original |
| Random init | https://im6.ezgif.com/tmp/ezgif-6-f581736d7b55-gif-jpg/frame_4_delay-0.5s.jpg | https://im6.ezgif.com/tmp/ezgif-6-bddbda4eb141-gif-jpg/frame_10_delay-0.5s.jpg | https://im6.ezgif.com/tmp/ezgif-6-cc217198547c-gif-jpg/frame_8_delay-0.5s.jpg | https://im6.ezgif.com/tmp/ezgif-6-34534017567c-gif-jpg/frame_8_delay-0.5s.jpg | D:\Programming\MATLAB\ML\ml-hw6\image1.png |
| k-means++ | https://im6.ezgif.com/tmp/ezgif-6-92bd34052129-gif-im/frame_4_delay-0.5s.gif | https://im6.ezgif.com/tmp/ezgif-6-2f7713c17743-gif-jpg/frame_4_delay-0.5s.jpg | https://im6.ezgif.com/tmp/ezgif-6-3af55ccf7094-gif-jpg/frame_7_delay-0.5s.jpg | https://im6.ezgif.com/tmp/ezgif-6-1e148079d611-gif-jpg/frame_10_delay-0.5s.jpg | D:\Programming\MATLAB\ML\ml-hw6\image1.png |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Image 2** | 2 Cluster | 3 Clusters | 5 Clusters | 10 Clusters | Original |
| Random init | https://im6.ezgif.com/tmp/ezgif-6-6358832e294f-gif-im/frame_5_delay-0.5s.gif | https://im6.ezgif.com/tmp/ezgif-6-a5082b6d21a8-gif-im/frame_8_delay-0.5s.gif | https://im6.ezgif.com/tmp/ezgif-6-07f3c67613ec-gif-im/frame_8_delay-0.5s.gif | https://im6.ezgif.com/tmp/ezgif-6-36663543baf8-gif-im/frame_5_delay-0.5s.gif | D:\Programming\MATLAB\ML\ml-hw6\image2.png |
| k-means++ | https://im6.ezgif.com/tmp/ezgif-6-b42248013756-gif-im/frame_5_delay-0.5s.gif | https://im6.ezgif.com/tmp/ezgif-6-279f7de66c26-gif-im/frame_5_delay-0.5s.gif | https://im6.ezgif.com/tmp/ezgif-6-b762263aa87f-gif-im/frame_6_delay-0.5s.gif | https://im6.ezgif.com/tmp/ezgif-6-6c603baae134-gif-im/frame_7_delay-0.5s.gif | D:\Programming\MATLAB\ML\ml-hw6\image2.png |

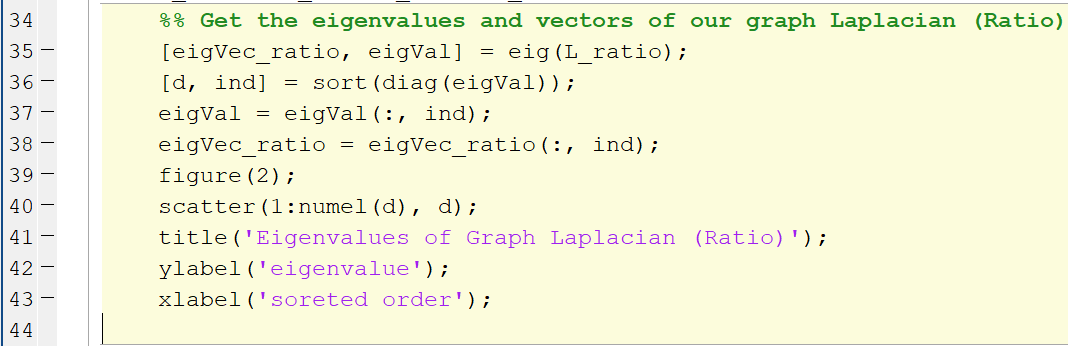
We see, that random initialization still gives bearable results, main difference can be seen in 3 clusters for image 1. Subjectively, k-means++ version looks better, but random initialization gives more information, since it also captures the blue part in the lower right corner of our image. However, the main advantage of k-means++ is that we need less iterations (on average) to converge.

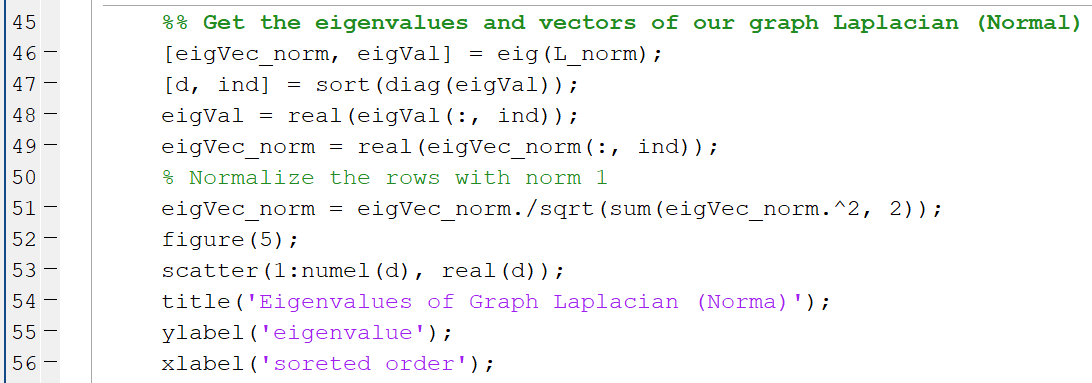
**Spectral Clustering: Implementation**

For spectral clustering we are, once again, using the Gram matrix computed before. However, now it acts as a graph representation of our data, we can call it the weighted adjacency matrix. Terminology aside, we use this matrix to first compute the degree of each node (datapoint) and graph Laplacian.



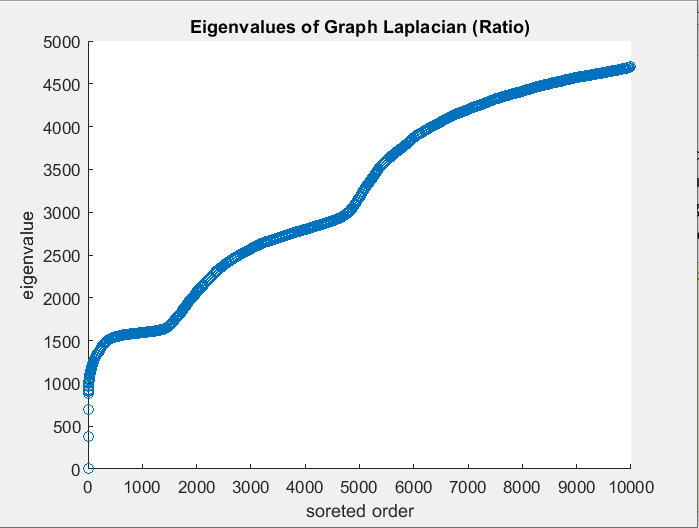
Here comes the first difference between the RatioCut and NormalizaedCut, in the first case Laplacian is simply given as the difference between the degree matrix and weight matrix, but in the second case we also normalize the previous Laplacian with the squared inverse of the degree matrix. Next, we use Laplacian matrices to compute corresponding eigenvalues and eigenvectors.



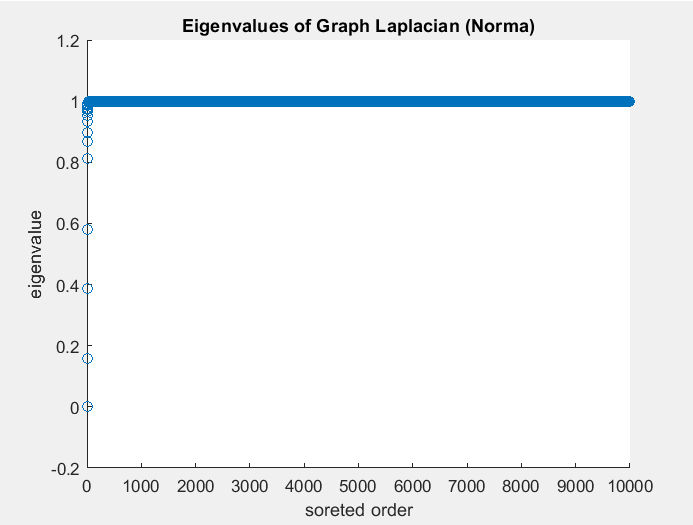


For the normalized version, we also normalize eigenvectors of Normalized Laplacian

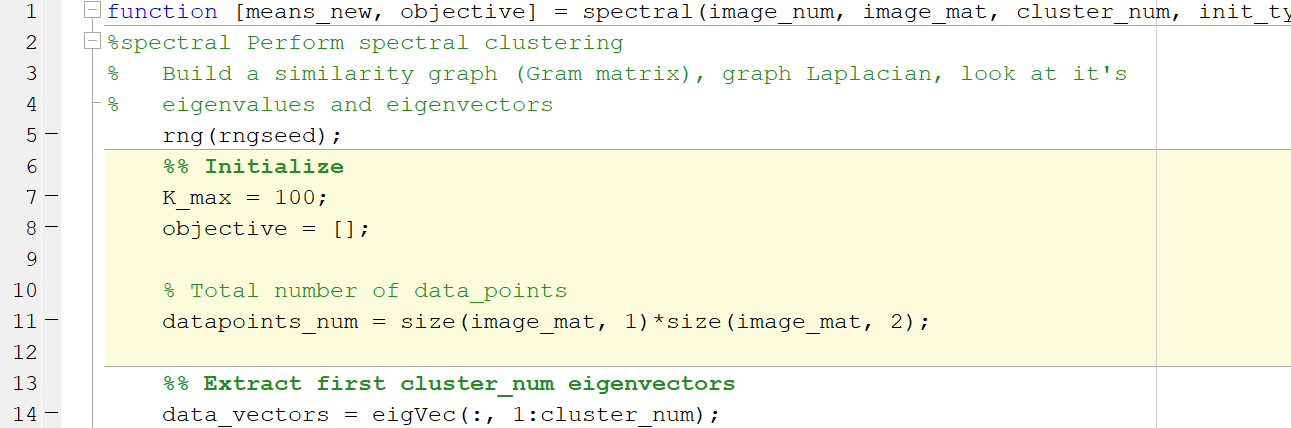
Example of sorted eigenvalues of graph laplacian for image 1:

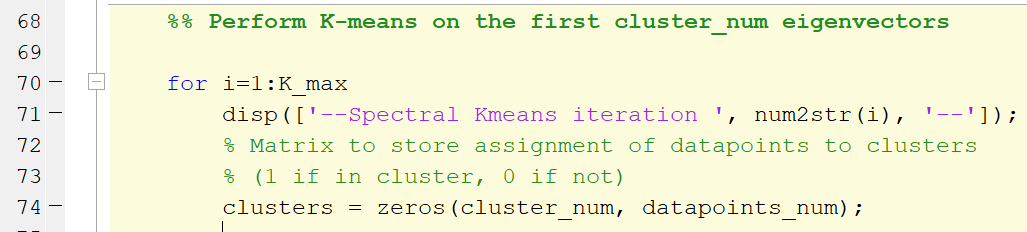


Notice the gap between the first three eigenvalues

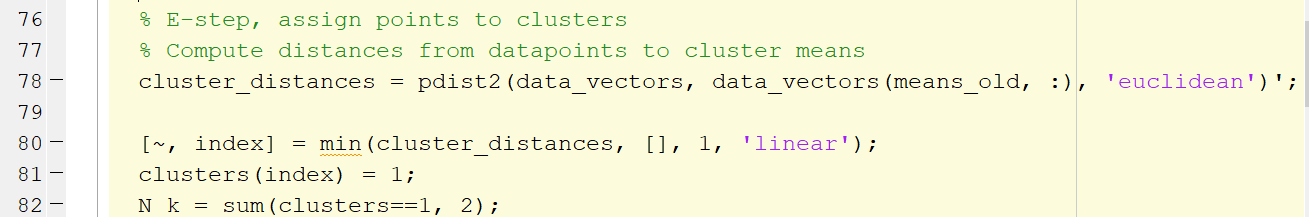


After having these eigenvectors, we have everything we need to start clustering our datapoints

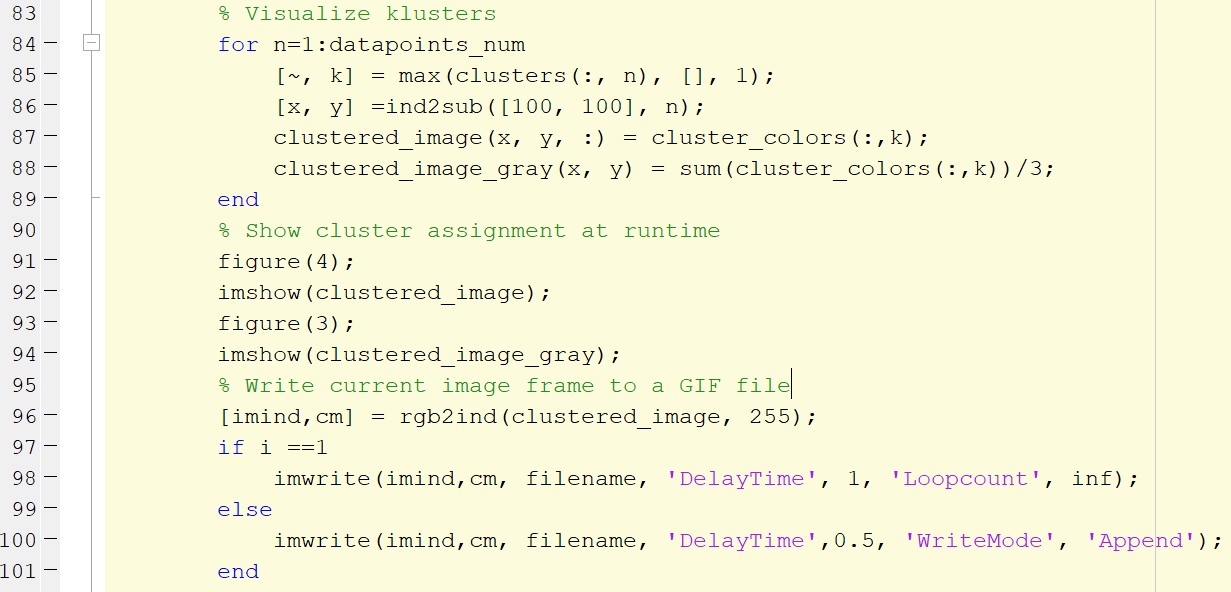


Depending on the number of clusters we want to partition our graph into, we work only with the first number of eigenvectors. Then we do similar initialization, assigning cluster colors, building GIF image name as in the Kernel K-means part, and can start the clustering procedure. Start as before: 

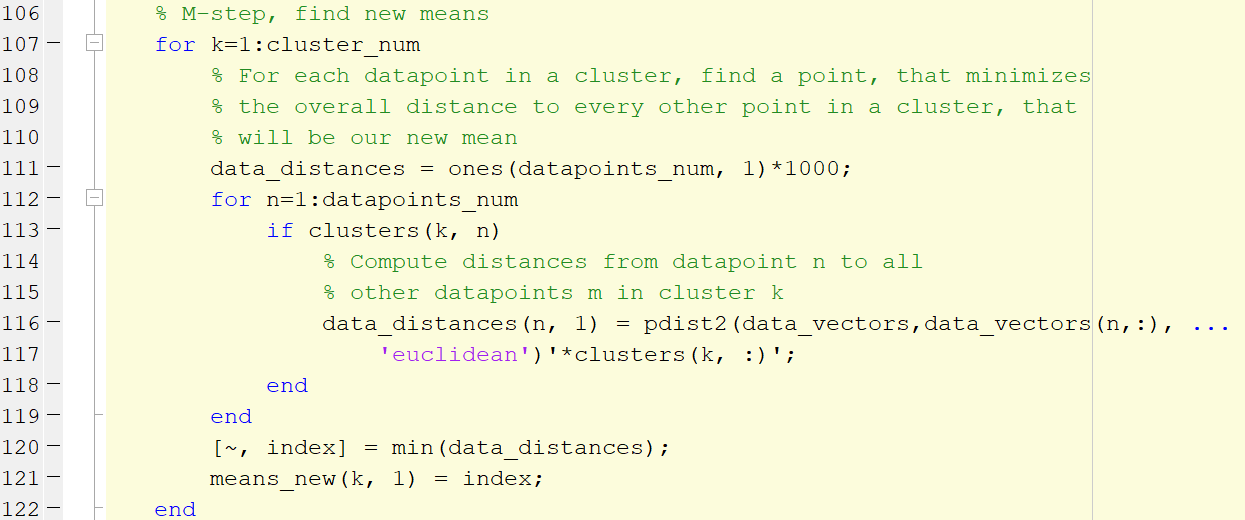
Next, compute distances from each eigenvector datapoint to cluster means. We don’t use Gram matrix, but compute direct Euclidean distances.



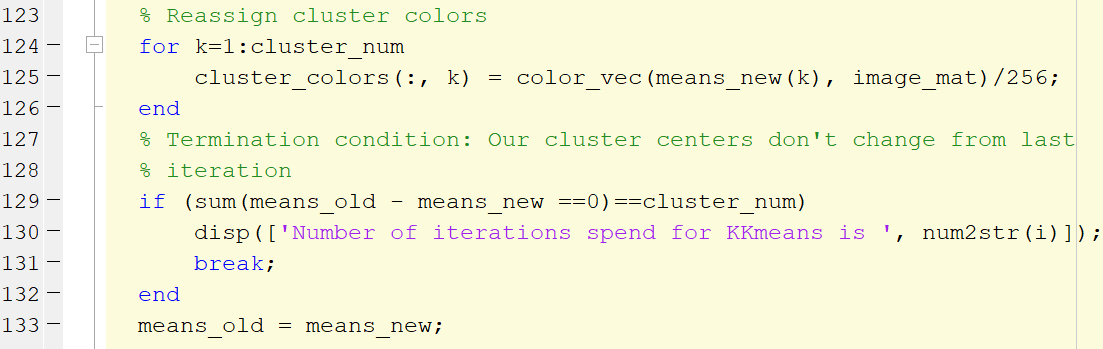
Again, we choose the minimum distance points to be assigned to appropriate clusters and proceed to visualization part. Visualization is the same as in Kernel K-means,



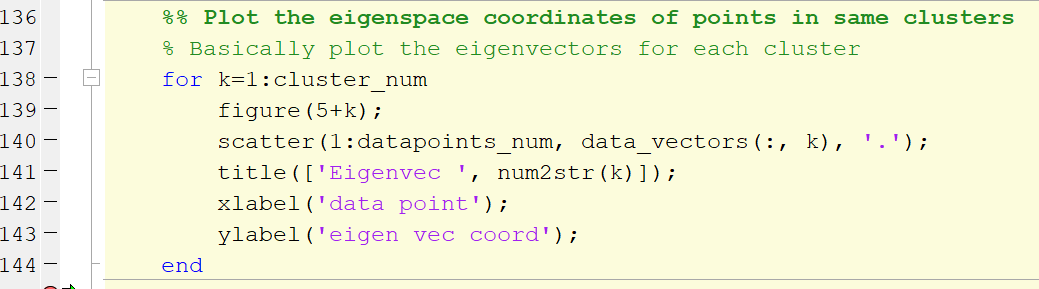
Next comes the M-step part. Idea is still the same, we compute distances from each point in a cluster to all the other points in a cluster, but now, instead of using Gram matrix as the distance measure, we use distances between eigenvector values.



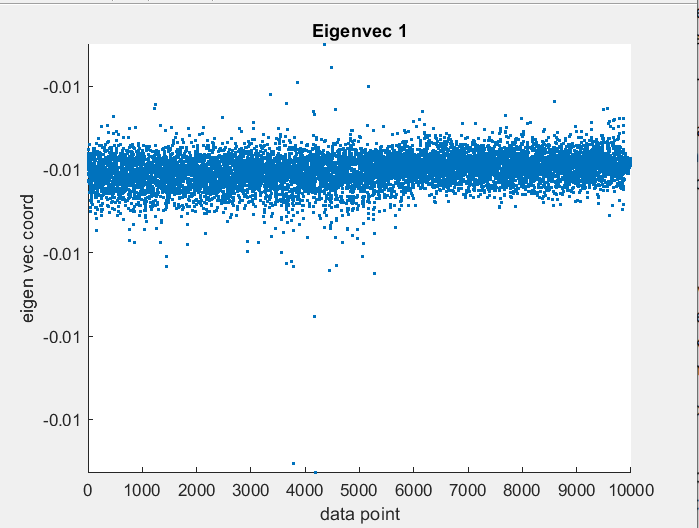
Last part in this K-means, as before, is clusters colors and termination criteria



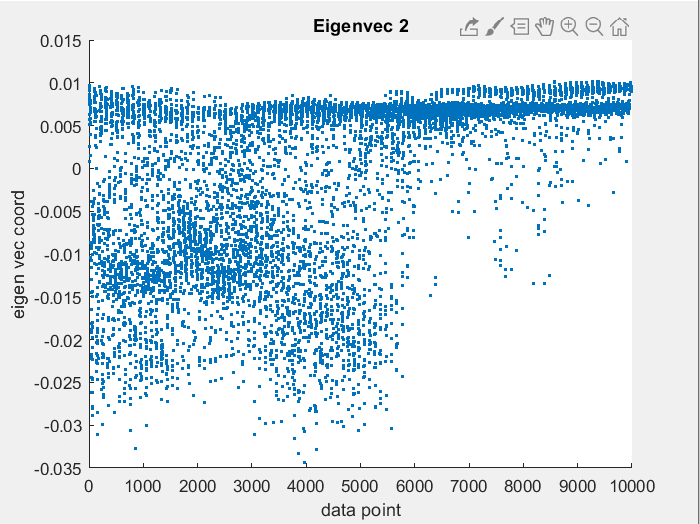
Finally, we plot the eigenvector values for each coordinate (same as number of clusters we choose) and look at whether points in the same cluster are have similar coordinates in this eigenspace.



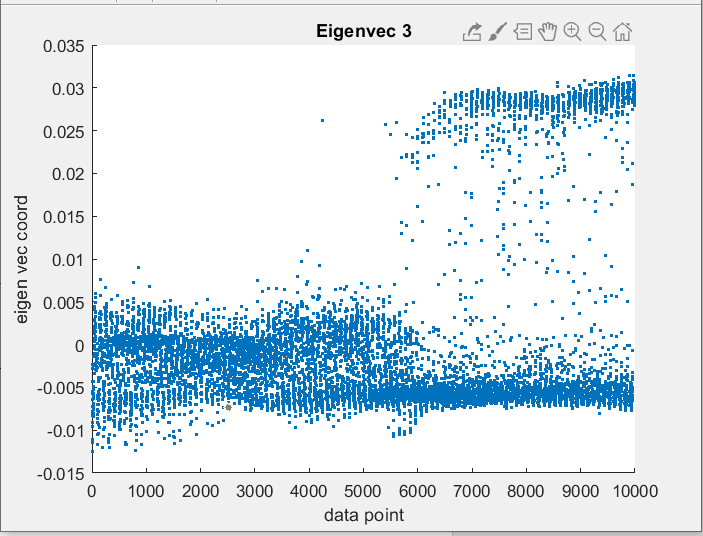
Eigenvectors for RatioCut graph Lapalcian are quite messy:



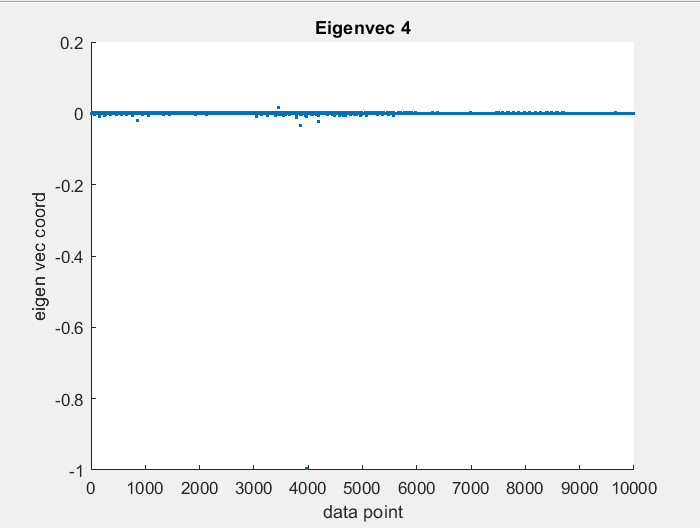
Every datapoint is the same, notice how the y-axis values don’t change



We see different values laying below and above zero, those are our different clusters



Same thing here, we decide which cluster our points belong to using these eigenvectors, difference in 0 level.



Next vectors essentially don’t influence our clustering procedure

Now look at what the normalized cut Laplacian eigenvectors