# TP1: Spectral Clustering

Bastien DECHAMPS

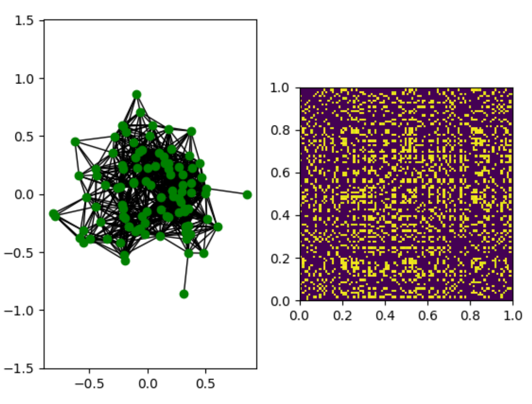
October 29, 2019

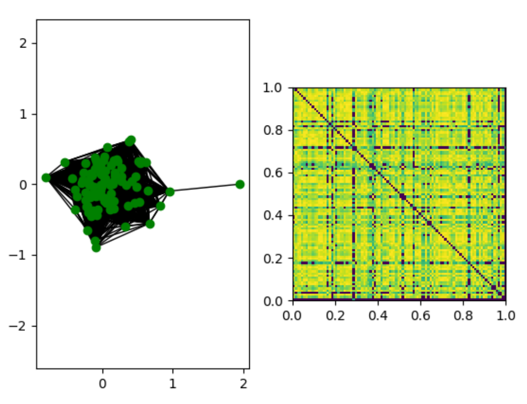
## 1 Graph construction

**1.1. What is the purpose of the option parameter in `worst_case_blob` (in the file `generate_data.py`)?**

The parameter `gen_pam` in `worst_case_blob` is added to the $x$-coordinate of the last point of the blob generated by `skd.make_blobs`. It will add an outlier to the blob (if its value is high enough) and thus it can be useful to check if the function `how_to_choose_epsilon` gives us a fully connected graph.

**1.2. What happens when you change the generating parameter of `worst_case_blob` in `how_to_choose_espilon` and run the function? What if the parameter is very large?**


(a) `gen_pam` $= 0$


(b) `gen_pam` $= 1$

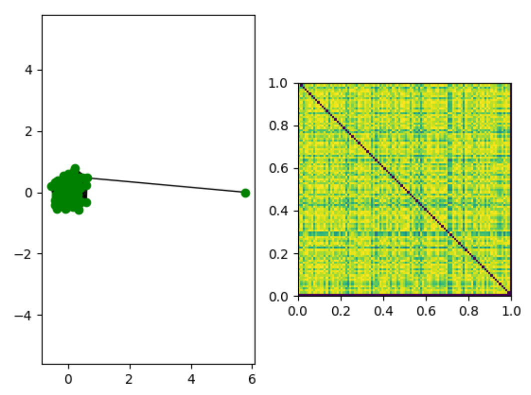
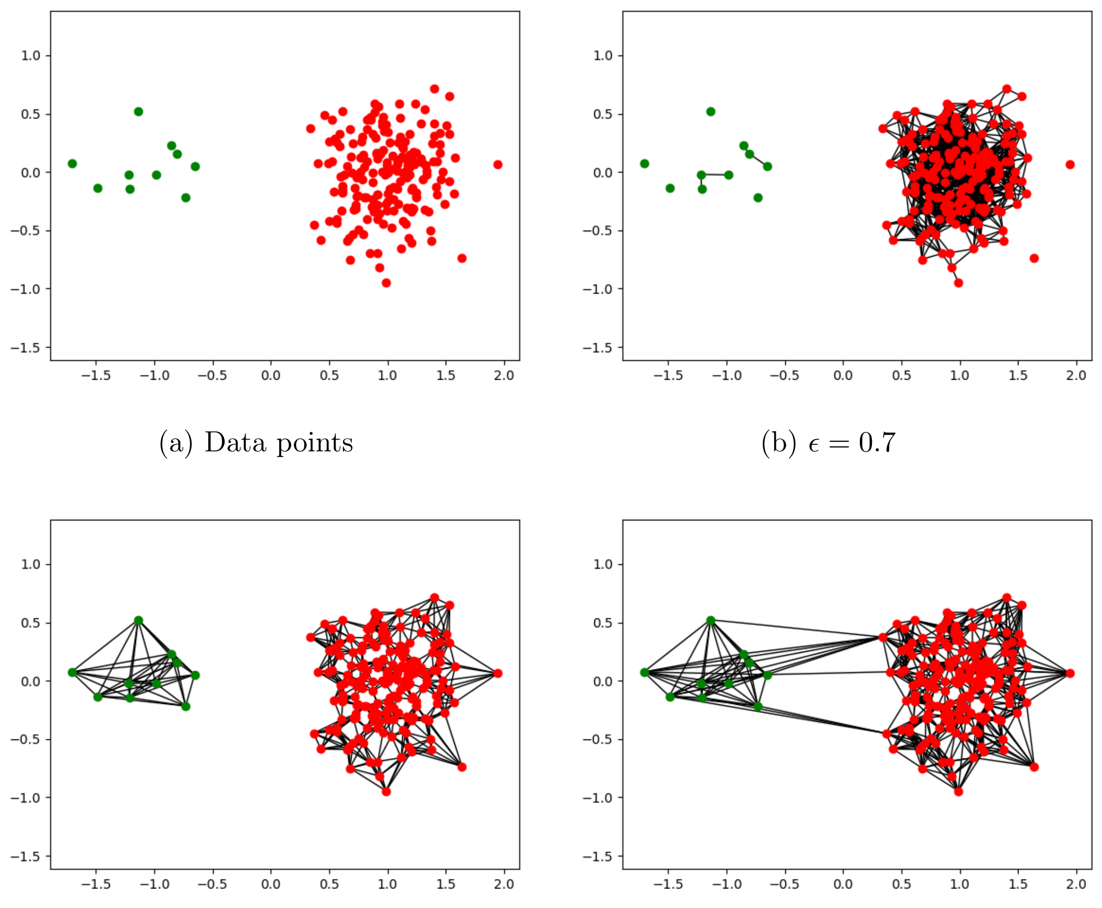

(c) `gen_pam` $= 5$

Figure 1: `how_to_choose_espilon` result given the parameter `gen_pam`

The larger `gen_pam` is, the furthest the outlier will be from the initial blob. Then, as this outlier has to be part of the minimum spanning tree, the maximum distance between two edges in it will be the value of `gen_pam` when it is large enough. Thus, `eps` will decrease exponentially to zero when it increases, and all the similarities between the data points will be larger than `eps`. Therefore, when `gen_pam` is very large, the graph is **fully connected**.

**1.3. Using `plot_similarity_graph` and one of the datasets, compare $k$-nn to $\varepsilon$ graphs. When is it easier to build a connected graph using $k$-nn? When using $\varepsilon$ graphs?**

The main difficulty for $\epsilon$-graphs is to choose correctly the parameter $\epsilon$. This problem occurs especially if we have data on different scales, meaning that some clusters are less dense than others. On figure 2, we choose two blobs with different densities, and we can see that for $= 0.7$, the points in the left blob are barely connected whereas the right one are well connected.



(a) Data points

(b) $\epsilon = 0.7$

(c) $k = 7$

(d) $k = 10$

Figure 2: Example of $\epsilon$ and $k$-nn graphs built on two blobs of different densities

The $k$-nearest neighbor graph does not care about the scales between the points and connect them anyway, as shown in the figure 2. We can even connect the two components to make the graph connex (on figure 2 for $k = 10$).

## 2 Spectral Clustering

**2.1. Build a graph starting from the data generated in `two_blobs_clustering`, and remember to keep the graph connected. Motivate your choice on which eigenvectors to use and how you computed the clustering assignments from the eigenvectors. Now compute a similar clustering using the built-in $k$-means and compare the results.**

According to the Rayleigh-Ritz theorem, the second eigenvector $u_2$ has to be taken to separate the graph in two clusters. The first eigenvector is $\mathbb{1}_n$ and corresponds to the smallest eigenvalue 0, which have a multiplicity of 1 as the graph is connex. Then we apply a $k$-means algorithm to cluster $u_2$, which gives us an assignment of the graph nodes. An example of spectral clustering on two connected blobs is represented on figure 3.

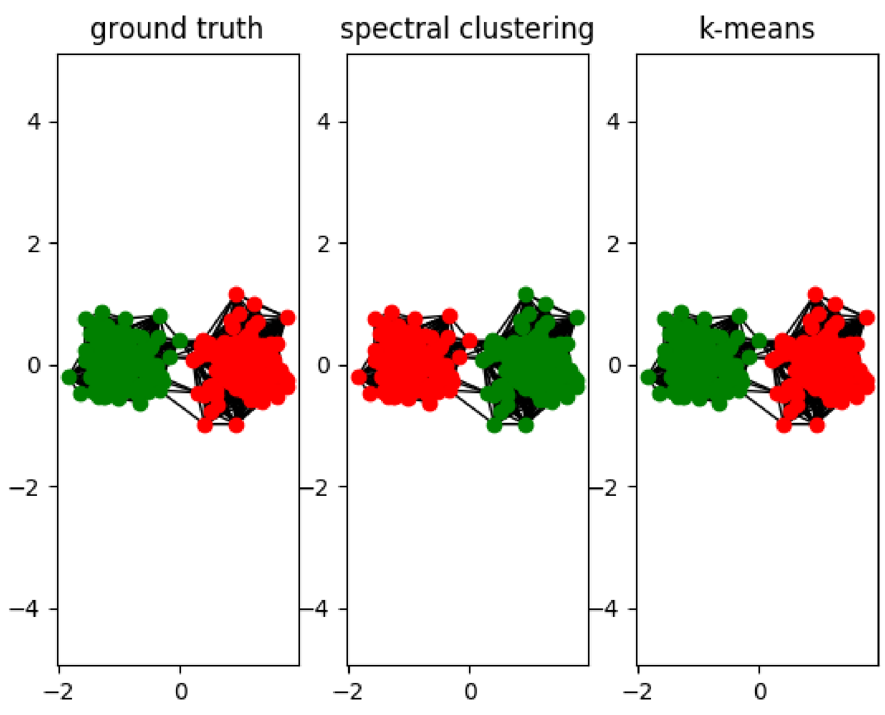


Figure 3: Spectral clustering vs. $k$-means for $k = 2$

**2.2. Build a graph starting from the data generated in `two_blobs_clustering`, but this time make it so that the two components are separate. How do you choose which eigenvectors to use in this case? Motivate your answer.**

The solution is the same as for the connected two blobs. Now we have two separate components, we should take the second vector $u_2$ which is this time associated with the eigenvalue 0. Then we apply a $k$-means algorithm with $k = 2$ to $u_2$. An example is represented on figure 4

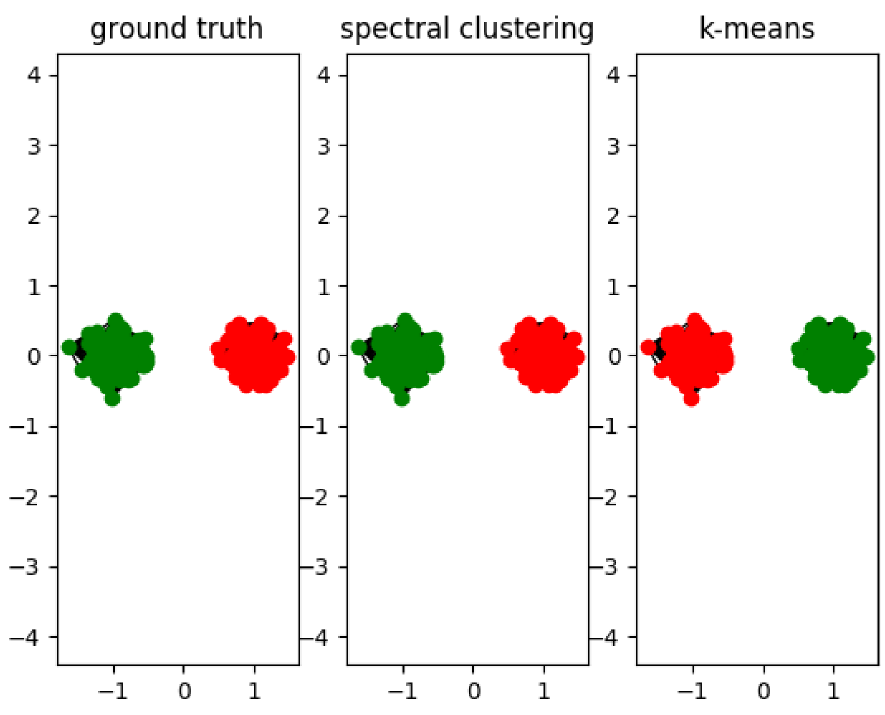


Figure 4: Spectral clustering vs. $k$-means for $k = 2$

However, in practice, we take the two first eigenvectors for both connected or non-connected blobs, and more generally the $k$ first eigenvectors to form $k$ clusters. Indeed, when the multiplicity of 0 is 2, the corresponding eigenspace is spanned by the two indicators $\mathbb{1}_{A_1}$ and $\mathbb{1}_{A_2}$, but the algorithm that finds the eigenvectors does not necessarily converge to those particular vectors, but rather some combination $\alpha_1 \mathbb{1}_{A_1} + \alpha_2 \mathbb{1}_{A_2}$.

**2.3. Look at the function `find_the_bend`. Generate a dataset with 4 blobs and $\sigma^2 = 0.03$. Build a graph out of it and plot the first 15 eigenvalues of the Laplacian. Complete the function `choose_eig_function` to automatically choose the number of eigenvectors to include. The decision rule must adapt to the actual eigenvalues of the problem.**

We generated 4 blobs with variance $\sigma^2 = 0.03$ and plotted the results in figure 5.

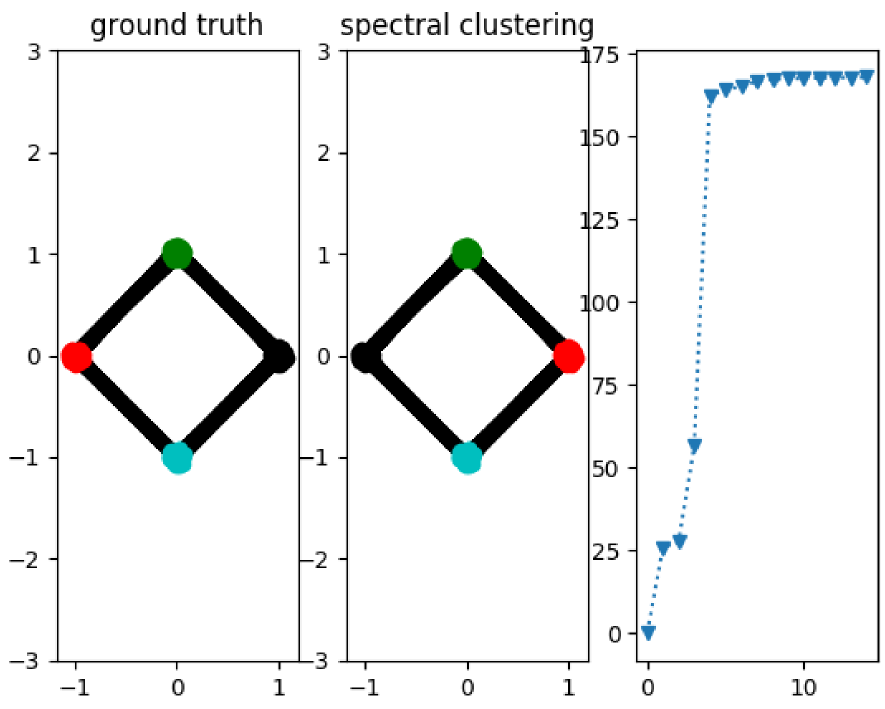


Figure 5: $k = 200, \sigma^2 = 0.03$

For the function `choose_eig_function`, we define a threshold on the second derivative od the sorted eigenvalues (the differences between two consecutive gaps between eigenvalues). As for the elbow method, we look for the first index $m$ of this second derivative to become significantly negative, and we choose the eigenvectors $u_2, ..., u_m$ to perform our clustering.

**2.4. Now increase the variance of the Blobs to $\sigma^2 = 0.20$ as you keep plotting the eigenvalues. Use the function `choose_eig_function`. Do you see any difference?**

We increase the variance to $\sigma^2 = 0.2$ without changing the other parameters. The results are represented on the figure 6. We see that by increasing the variance, the clusters become closer to each other, and it is more difficult to choose the gap in the eigenvalues plot which will define the number of classes. With the heuristic we defined, we managed to find the right number anyway.

**2.5. When you built the cluster assignment, did you use thresholding, $k$-means or both? Do you have any opinion on when to use each?**

To bluid the cluster assignments, I used $k$-means rather than directly thresholding the values of the eigenvectors. If the clusters are well separated, we know that the eigenvectors of $L$ and $L_{\text{rw}}$ are piecewise constant, and in this case thresholding or even any clustering algorithm sould be a good final step for the spectral clustering algorithm. However, when the data is noisier and the clusters become less distinct, the eigenvectors are not piecewise constant and clustering algorithms more complex than thresholding should be used.

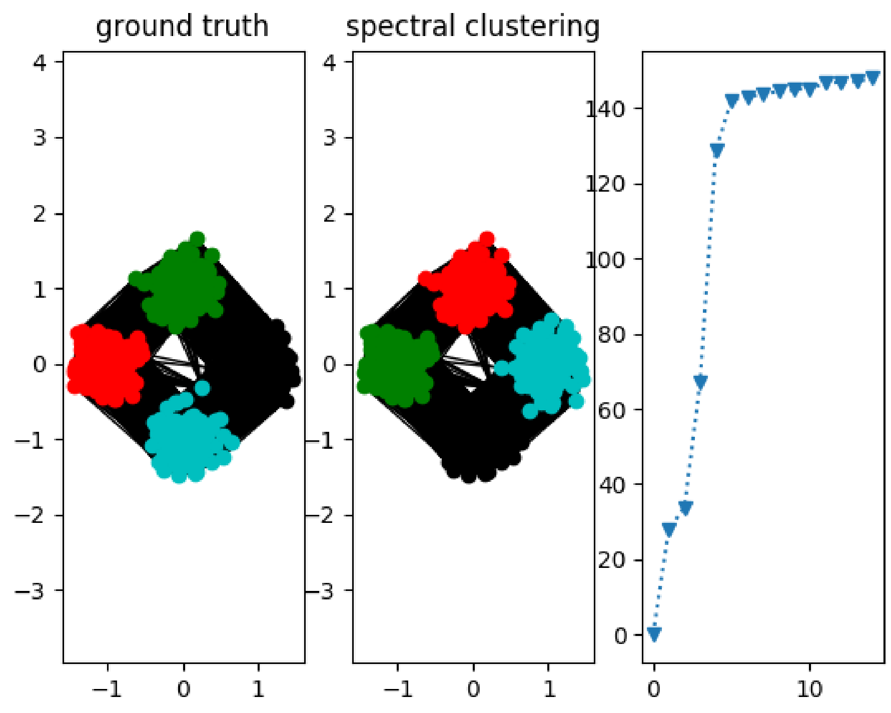


Figure 6: $k = 200, \sigma^2 = 0.2$

**2.6. What is another use that looking at the distribution of the eigenvalues can have during clustering, beside choosing which eigenvectors to include?**

If the number of classes is not given, it can be approximated by looking at the eigenvalues of the Laplacian with appropriate techniques such as the eigengap heuristic. Futhermore, some geometric invariants of the graph can be expressed with the eigenvalues of the Laplcaian. For example, the size of the cuts are related with the first eigenvalues.

**2.7. Plot your results using spectral clustering and $k$-means in `two_moons_clustering` and compare the results. Do you notice any difference? Taking into consideration the graph structure, can you explain them?**

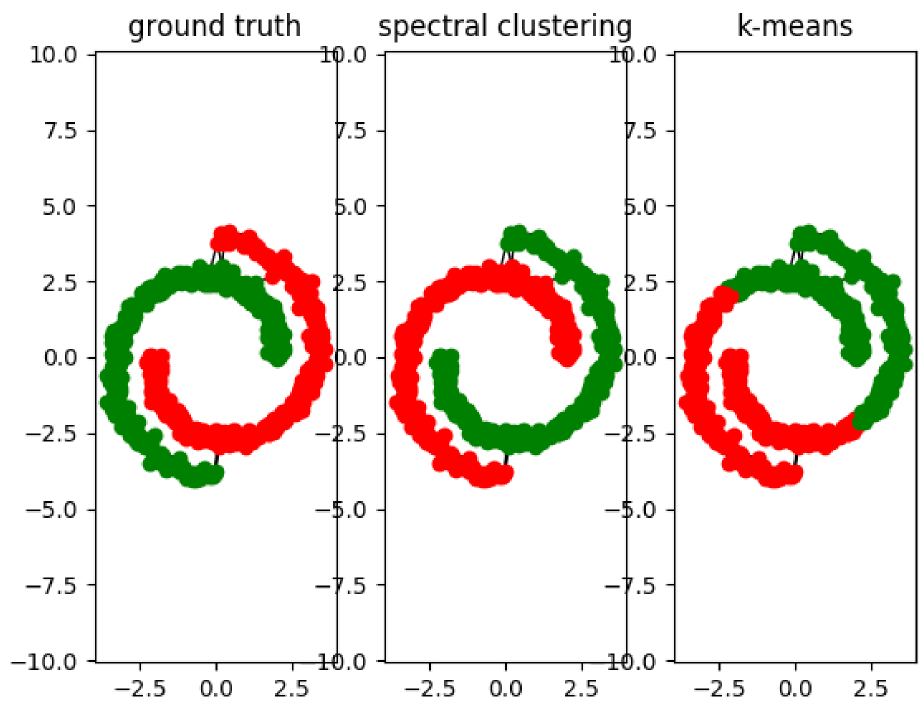


Figure 7: `two_moons_clustering` with $k = 15$

For moon-shaped data, wee can see on figure 7 that $k$-means is inneffective as the data is not

Gaussian and more complicated than simple blobs. On the other hand, spectral clustering performs well by choosing correct parameters. On figure 7, I chose to build a $k$-nn graph with $k = 15$ so that the two moons are barely connected to each other, making the cut easier.

**2.8. In the function `point_and_circle_clustering`, compare spectral clustering using the normal laplacian $L$ and the random-walk regularized Laplacian $L_{rw}$. Do you notice any difference? Taking into consideration the graph structure, can you explain them?**

We've seen that the normalized Laplacian solves the N-cut problem, whereas the unnormalized pectral clustering solves the Ratio-cut problem. The difference between those two problems is that Ratio-cut maximizes the cardinal of each cluster $|C_i|$ while N-cut tries to maximize the volume of each cluster $\text{vol}(C_i)$. For the `point_and_circle` dataset, represented on figure 8, we have a dense blob in the middle surrounded by a dense ring of points. As there are a lot more points in the ring than in the blob, separate the blob from the ring will penalize more the Ratio-cut than the N-cut, which is more adapted inthis case (as shown in figure 8).

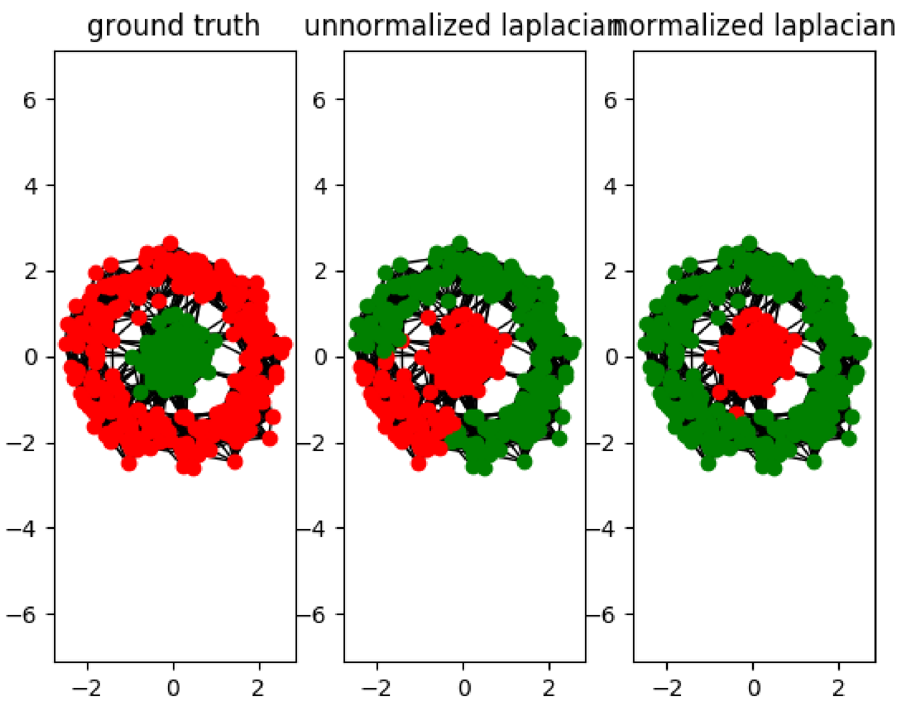


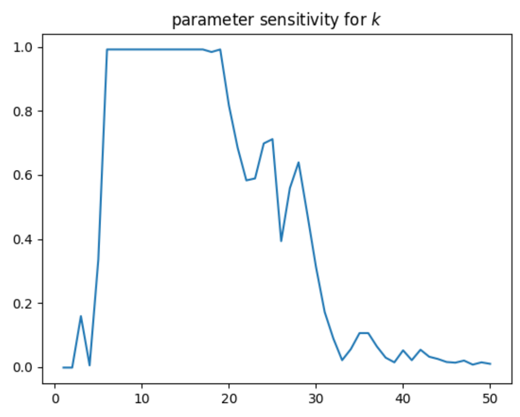
Figure 8: `point_and_circle` with $\varepsilon = 0.7$ and $\sigma^2 = 0.3$

**2.9. Complete the function `parameter_sensitivity`, and generate a plot of the ARI index while varying one of the parameters in the graph construction $\varepsilon$ or $k$. Comment on the stability of spectral clustering.**

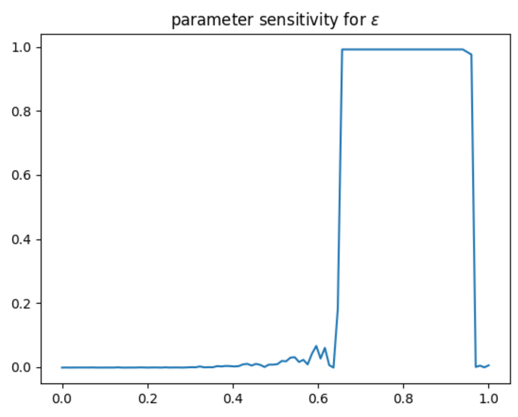
The ARI for the spectral clustering of each type of graph ($\varepsilon$ and $k$-nn) built from two-moon-shaped data is plotted on figure 9. For each parameter, we see that there is an interval in which the clustering is good (ARI close to 1). For $k$, it's between $[5, 20]$, and for $\varepsilon$, it is $[0.6, 0.95]$. Of course, this interval may vary given the shape of the data and its variance.

**2.10. If we did not have access to *true* labels how could we evaluate the clustering result (or what should we not use as evaluation)?**

If we don't have the true labels, the performance of the clustering algorithm cannot be measured with *external measures* as the ARI (or the Adjusted Mutual Information for example). We have to use *internal measures* instead, i.e. that only depends on the clustering result. A



(a) varying $k$



(b) varying $\varepsilon$

Figure 9: Parameter sensitivity for two moon spectral clustering

well-known internal measure is the **silhouette score**, defined as :

$$s(i) = \begin{cases} \dfrac{b(i) - a(i)}{\max\{a(i), b(i)\}} & \text{if } |C_i| > 1 \\ 0 & \text{else} \end{cases}, \tag{1}$$

$$\text{where } a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_j \neq i} d(i,j) \text{ and } b(i) = \max_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$$

The distance $d$ here have to be defined between the spectral embeddings rather than the points of the original data to be meaningful.

Another internal measure used in graph theory is the **modularity** and many algorithms use it to find communities into graphs (eg in Louvain algorithm). It is defined as follow :

$$Q = \frac{1}{2m} \sum_{i,j} \left[ w_{ij} - \frac{d_i d_j}{2m} \right] \delta(c_i, c_j) \tag{2}$$

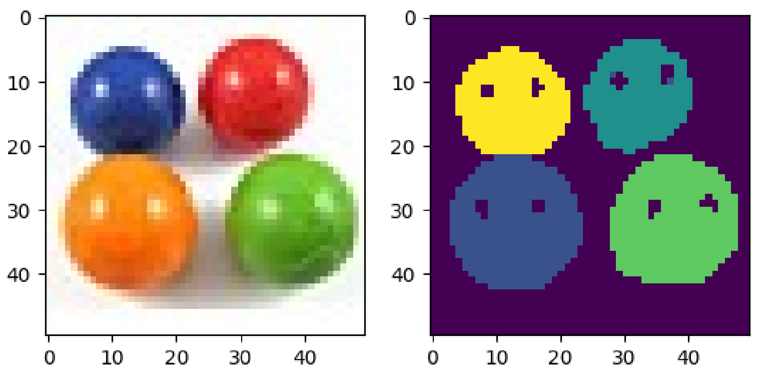
where $m$ is the sum of all weights in the graph, $d_i = \sum_j w_{ij}$ the degree of the node $i$ and $\delta(c_i, c_j) = 1$ if $i$ and $j$ are in the same cluster and 0 else.

## 3 Image Segmentation

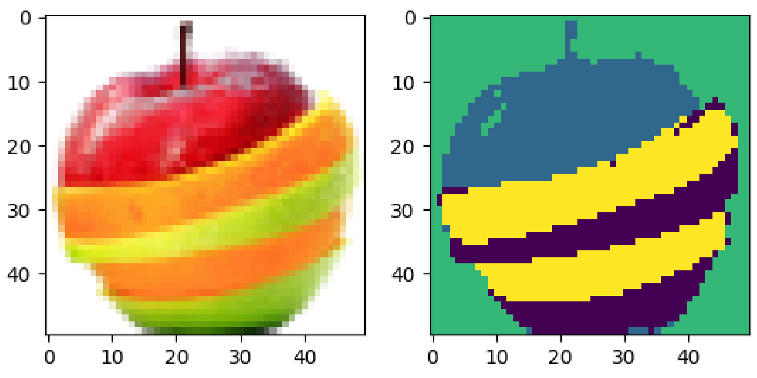
**3.1. The first documentation is the code. If your code is well written and well commented, that is enough. If you think you need to better express some of the choices you made in the implementation, you can use this question in the report. Remember to include all the related code in the submission.**

I ran `image_segmentation` with the two provided images, the results are represented on figure 10.

**3.2. A full graph built between the pixels of a $50 \times 50$ image corresponds to $50^2$ nodes. Solving the full eigenvalue problem in this case would scale in the order of $2^{34}$. Even on weak hardware (i.e. iPhone) this takes only seconds to minutes. Segmenting a Full HD picture of $1920 \times 1080$ would scale in the order of $2^{64}$ (about a month on a decent machine i.e. not an iPhone). Beyond that, the large picture would require to store in memory a graph over millions of nodes. A full graph on that scale requires about 1TB of memory. Can you think two simple techniques to reduce the computational and occupational cost of Spectral Clustering?**



(a) $k = 100$



(b) $k = 150$

Figure 10: Image segmentation using $k$-nn graph

As well as using sparse representation, some large-scale graph clustering methods can be used such as the Nyström Method that can find eigenvectors without storing the Laplacian. One can also downsample the image to be clustered using for example maxpooling or average pooling, then apply spectral clustering, and finally upsample the cluster assignment on the original image.

**3.3. Did you use `eig` or `eigs` to extract the final eigenvectors? Shortly, what is the difference between the two? How do they scale to large graphs (order of complexity)?**

I used `eigs` and more precisely its adapted version for symetric matrices `eigsh`. Indeed, with correctly chosen paramaters $k$ or $\varepsilon$, the laplacian of the graph is very sparse, and the computation of the first eigenvectors is way faster. Inverting a dense matrix have generally a time complexity of $\mathcal{O}(n^3)$, but finding the $m$ first eigenvectors of a sparse matrix can be done in $\mathcal{O}(nm\,\langle d\rangle)$, where $\langle d\rangle$ is the average degree of the associated graph.