Master M2 MVA 2015/2016 - Graphs in ML - TP 1

By: Lilian Besson (`lilian.besson at ens-cachan.fr`).

Attached programs

The programs for this TP are included in the `zip` archive I sent, and are regular **MATLAB/Octave** programs (tested with Octave only, v3.8 and v4, on both Linux and Windows).

Problem 1 : Graph Construction

Question 1.1

Remark: See the code `build_similarity_graph.m` for more details.

For k-nn graph, the value of σ^2 does not influence the links of the graphs, but it does influence the weights. Indeed, whatever σ^2 is, the exponential similarity function will give the same ordering of k closest neighbors of one node i, as showed in Figure 1.

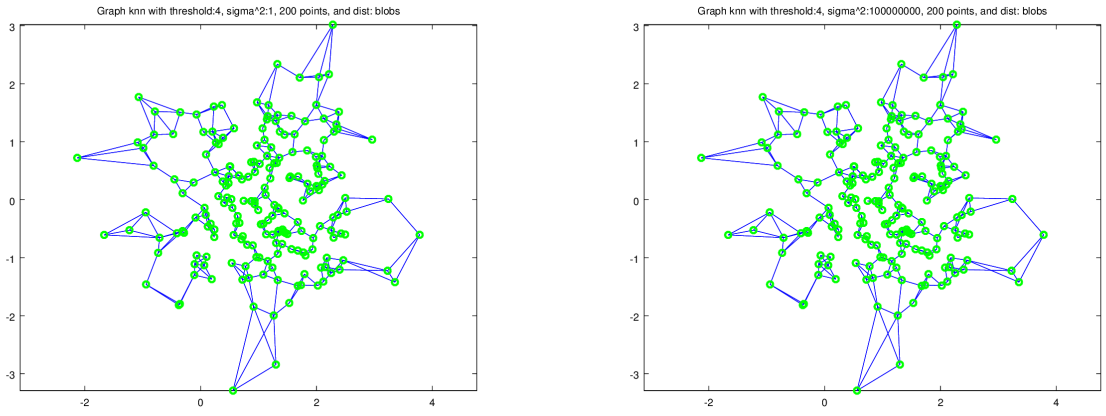

Figure 1: A 4-nn graph for 200 points, with $\sigma^2 = 1$ on the left but $\sigma^2 = 10^8$ on the right (same structure).

Therefore, I think *we cannot see a relationship* between the **structures** of these two graphs:

1. a k-nn graph with large k and small σ^2 (it will be very densely connected but with very small non-1 weights),
2. a k-nn graph with small k and large σ^2 (it will be very sparse but with high non-zero weights),

Question 1.2

Right now, the graph construction algorithms (the two variants, k-nn or ε) cannot scale to "big" graphs because of their *time complexity*:

- `exponential_euclidean` is always in $O(n^2 d)$ (whatever the σ^2) for n samples of dimension d,
- k-nn graph construction is in $O(n^2 \min(k, \log(n)))$ (we have to take the k smallest elements of a vector of size n, n times),
- and ε graph construction is also in $O(n^2)$.

Both the two variants face the same limitation of running in time $O(n^2)$ (or worse).

Another issue is the *memory complexity*, ie. the way we store the graph matrices `G` and `similarities`: if we use dense matrices, they will have occupy a memory size of $O(n^2)$, no matter the way we build the graph.

One possible improvement is to use *sparse* matrices for `G`, but I do not see a way to compute either the k-nn or the ε graph without *entirely* computing the similarities[1]. Ideas for a real improvement could be parallelization, or building only an approximation of the graph.

Question 1.3

For the `worst_case_blog` distribution, the parameter `dist_options` define the distance between the blob (the $n-1$ first points) and the last point which is located on the right of the blob. The higher `dist_options` will be, the further that last point will be.

That distribution is called "worst case" because it will be shaped exactly like an unique blob, except with one alien point really far away from the others.

Question 1.4

The generating parameter (`dist_options`) will move away the last generated point. Hence, in order to have a connected graph, we must choose a *very low* epsilon.

In our experiments, with 300 points and $\sigma^2 = 0.50$:

- For a distance of 0, for instance, ε is computed as 10^{-11} which is already small.
- For a distance of 2, for instance, ε is computed as 10^{-27}!
- For a distance of 10, for instance, ε is computed as 10^{-60} (which is extremely small!).
- In fact, after a certain value (20 in our experiments), the graph cannot be connected anymore as the last point has a similarity of 0 with all the other points.

Question 1.5

- It is easier to build a k-nn connected graph when the data is separable by affine plans (like for the `blobs` distribution),
- Oppositely, it is easier to build a ε connected graph when the data is compact class-wise (like `blobs` or `two_moons`).

Problem 2 : Spectral Clustering

Question 2.1

For this first example, `two_blobs_clustering.m` generates two compact blobs, so if k is too small, a k-nn graph will have *two* separated connected components. Hence, in order to keep the graph connected, we choose a k-nn graph with $k = 16$ (I tried different values, starting from 2 before having one unique click from $k \geqslant 16$), and we arbitrarily chose $\sigma^2 = 0.5$ (but it seems to not influence the result very much).

Therefore for spectral clustering we have taken the first *two* eigen-vectors corresponding to the unique zero eigen-value (u_0, $\lambda_0 = 0$) and the smallest non-zero eigen-value (u_1, with $\lambda_1 =$ `9.672e-03`). Then we could have done as seen in the class ("Relaxing Balanced Cuts"), by computing the sign of the spectral coordinate, but this might have been two simple (it works here, but not in general). So instead, we have run a k-means

[1] Indeed, we either have to sort them and pick the k smallest or to filter them – both operations require the entire array of similarities between a point `i` and all the others points...

algorithm[2] on the rows of the matrix U containing the vectors u_0 and u_1 as column, and assigned the resulting labels to our original samples (x_i).

The results are plotted bellow (Figure 2):

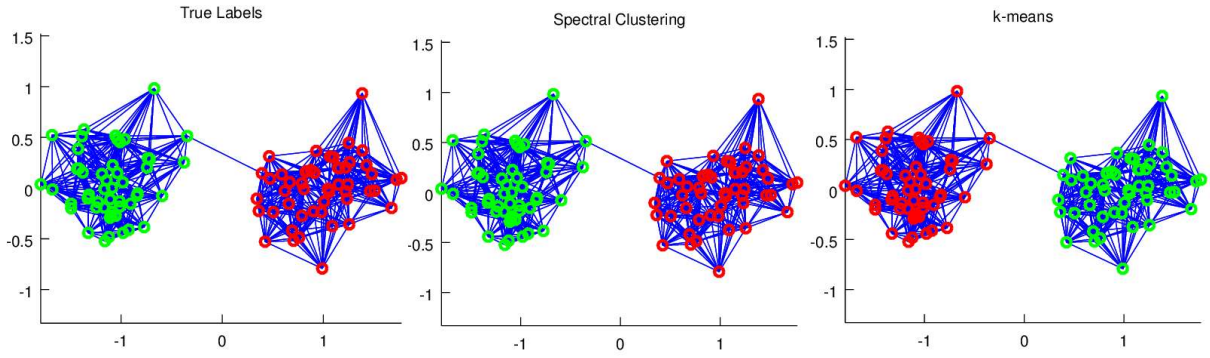

Figure 2: k-nn: True clusters *vs* Spectral clustering *vs* k-means (connected graph).

This leads to a *perfect separation* for spectral clustering thanks to the chosen values of u_0 and u_1. We get the same results with k-means, thanks to the compactness[3] of the data.

Question 2.2

This time, in order to build a graph with *two* separated connected components. We choose an ε-graph with $\varepsilon = 0.5$ to have exactly two separated blobs (I tried different values, from 0.1 to 1.0), and we arbitrarily chose $\sigma^2 = 0.5$ (but it does not influence the result very much).

Therefore for spectral clustering we have taken the first *two* eigen-vectors corresponding to the two zero eigen-values. That is the easier case: each eigen-vector leads to the indicator of its cluster. Their values are $u_1 = \mathbf{1}_{C_1}$ and $u_2 = \mathbf{1}_{C_2}$ where $(\mathbf{1}_{C_i})_j = \mathbf{1}_{x_j \in C_i}$ and C_i the i-th cluster. Then we have run a k-means algorithm on the rows of the matrix U containing the vectors u_1 and u_2 as column, and assigned the resulting labels to our original samples (x_i).

The results are plotted bellow (Figure 3):

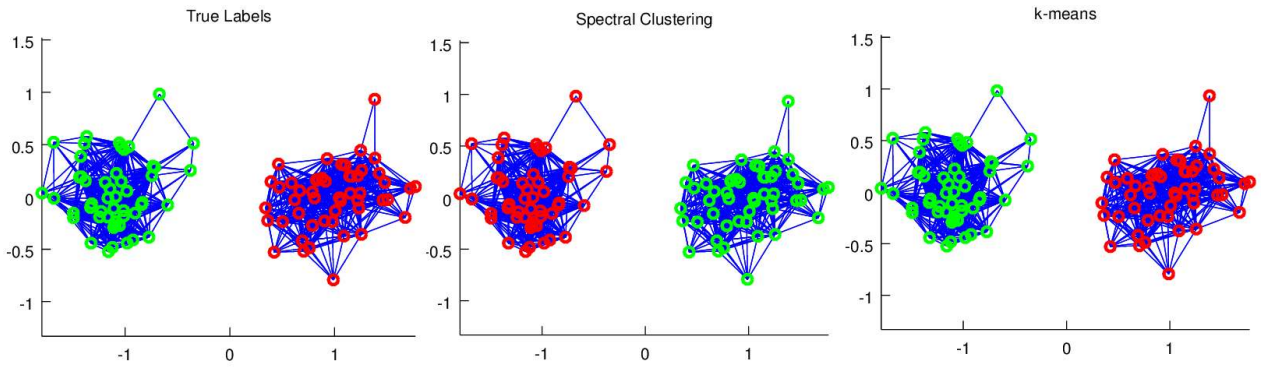

Figure 3: ε-graph: True clusters *vs* Spectral clustering *vs* k-means (2 connected components).

The second choice of type of similarity graph also leads to a *perfect separation* for spectral clustering. We get the same results with k-means, thanks to the compactness of the data.

[2] Thanks to the built-in `kmeans` function of Octave/Matlab, even if we could have rewritten ourself – it's not that complicated.
[3] This works because we worked with `two_blobs_clustering.m` that generates two compact blobs.

Question 2.3

For the `choose_eig_function`, we implement the elbow heuristic: the goal is to choose the number k such that all eigen-values $\lambda_1, \dots, \lambda_k$ are very small, but λ_{k+1} is relatively large in comparison (first big eigen gap).

In practice, for the first example with $\sigma^2 = 0.03$, I chose to build a 8-nn graph, and we found $k = 4$, as expected (there is 4 blobs). The `adaptative_spectral_clustering` function works quite well, giving a perfect separation once again. The results are plotted bellow (Figure 4), along with the first 15 eigen-values (we can clearly seen the "elbow" at $k = 4$):

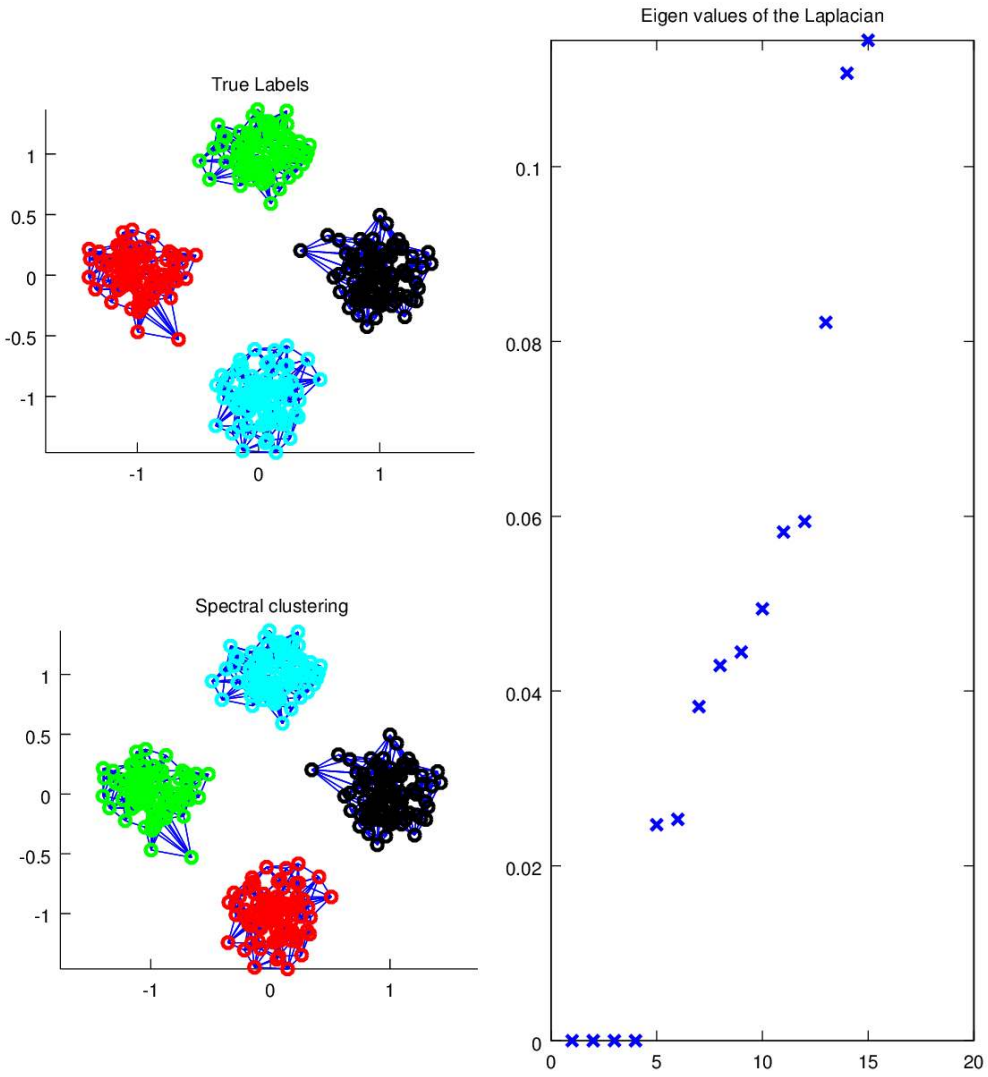

Figure 4: 8-nn graph: True clusters *vs* Adaptive Spectral clustering, and first 15 eigen-values (clear elbow).

Question 2.4

For the same example but with $\sigma^2 = 0.20$, we see that building either a k-nn or an ε graph will be harder, as the 4 blobs are mixed. The `adaptative_spectral` function worked very well, with the choice of 3 eigen values, but `adaptative_spectral_clustering` failed, because the elbow rule has difficulty to chose 3 eigen values. The results are plotted bellow (Figure 5), along with the first 15 eigen-values (the "elbow rule" is less obvious):

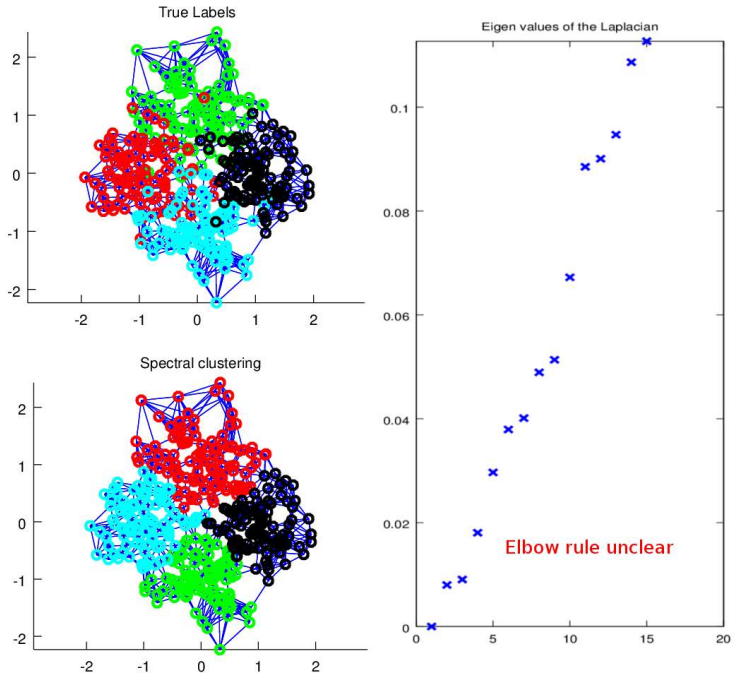

Figure 5: 8-nn graph: True clusters *vs* Spectral clustering, and first 15 eigen-values (unclear elbow).

The main difference we observe for this larger value of σ^2 is that the data are now more compact. But The k-means clustering works quite well too.

Question 2.5

The number of 0 in the spectrum gives the number of connected components. So if we expected the data to be really clustered in k groups but we find a different number of connected components, maybe the graph was wrongly built.

Looking at the distribution of the eigen-values can help us to determine if our cluster are well defined of not, the larger the gap of the bend will be, the better our clusters will be determined. I do not see other information that could be read on the spectrum.

Question 2.6

Here we compare spectral and k-means clustering on the "two moons" dataset, and we observe for the first time that k-means fails while spectral clustering works perfectly well (either with a 5-nn graph or with a ε-graph with $\varepsilon = 0.5$, and both had $\sigma^2 = 0.5$). The results with a 5-nn and a ε similarity graphs are plotted bellow (Figures 6 and 7):

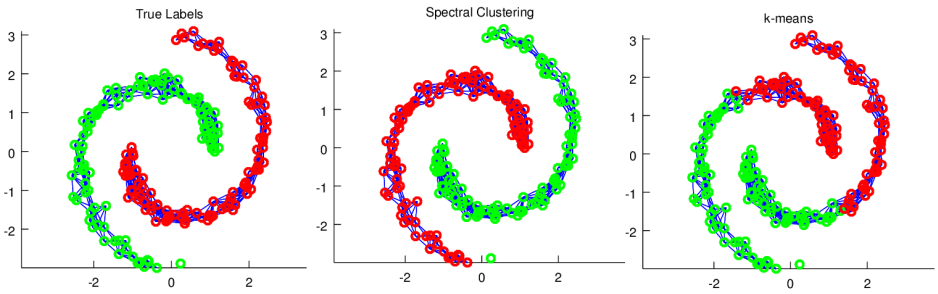

Figure 6: ε graph: True clusters *vs* Spectral clustering *vs* k-means (2 connected components).

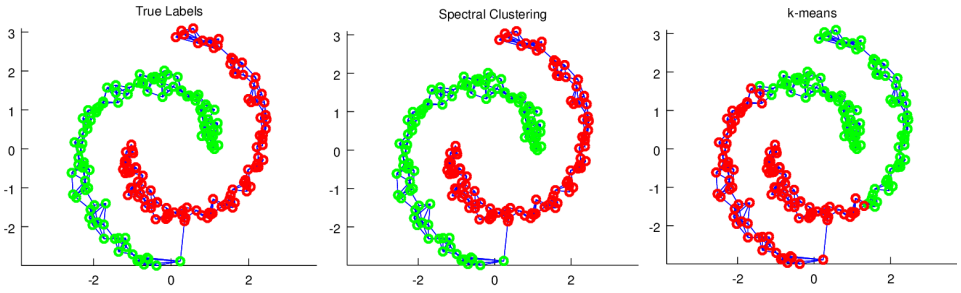

Figure 7: 5-nn graph: True clusters *vs* Adaptive Spectral clustering *vs* k-means (2 connected components).

Question 2.7

Here we compare spectral clusterings with L and L_{rw}, respectively the Laplacian and the "random walk" regularized Laplacian of the similarity graph, on the "point and circle" dataset. We observe for the first time that the first one fails to separate the point from the circle, while the random walk clustering works perfectly well, with a 20-nn graph (but not for a ε-graph), and $\sigma^2 = 0.5$ stayed constant).

The results are plotted bellow (Figures 8 and 9):

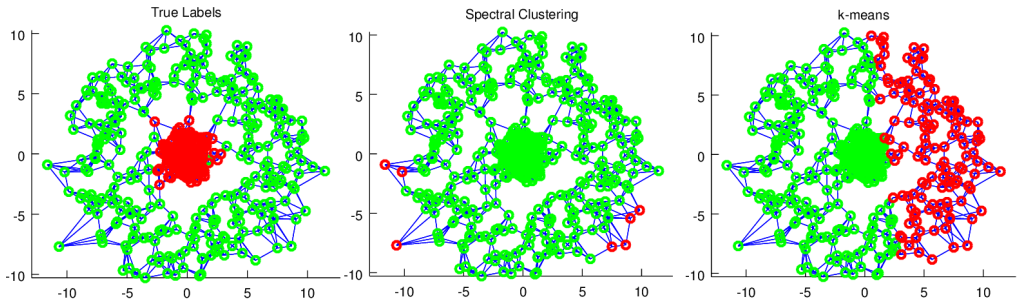

Figure 8: 20-nn graph: True clusters *vs* Spectral clustering with L *vs* k-means (*failing*).

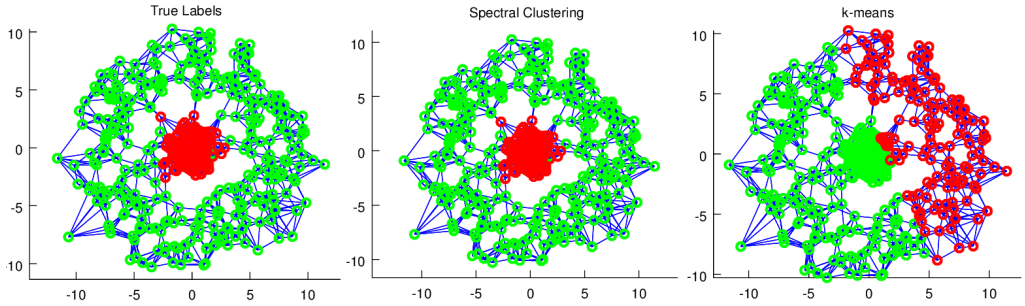

Figure 9: 20-nn graph: True clusters *vs* Spectral clustering with L_{rw} *vs* k-means (works well).

We can interpret this difference of performance by the fact that the random-walk clustering takes even more into account the connectivity of the graph. If we normalize, we do observe a much better clustering, and this is because here we are approximating the *Normalized Cut* problem, and this leads to have more "homogeneous" clusters. We can see on the figure above, that even if we do not retrieve the original segmentation we retrieve something very consistent. As in the previous example, k-means is not adapted to solve this problem.

Question 2.8

For visualizing the ARI "performance score" as a function of the similarity graph parameter (k or ε), we work here with the "two moons" dataset (with $N = 500$ points). The results are plotted bellow (Figures 10 and 11).

- For the ε graph, we chose to make the parameter vary from 0.05 to 0.95 with a step of 0.05, and as expected the ARI score has a maximum value of 1 for some values of ε (as seen in Question 2.6). (Figure 10)

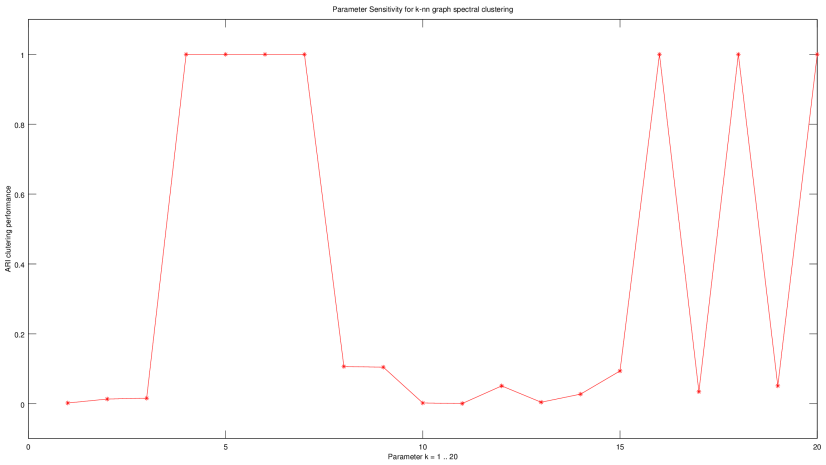

Figure 10: k-nn graph: ARI performance as a function of $k = 1 \dots 20$.

- For the ε graph, we chose to make the parameter vary from 0 to 1 with a step of 0.1, and as expected the ARI score has a maximum value of 1 for some values of ε (as seen in Question 2.6). (Figure 11)

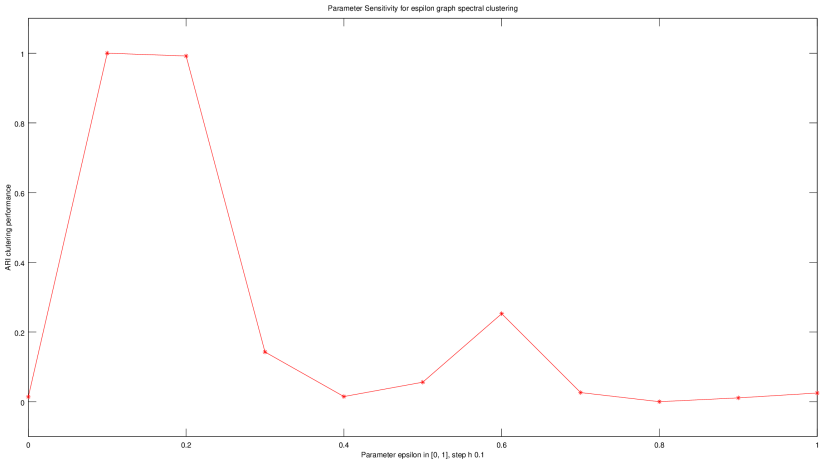

Figure 11: ε graph: ARI performance as a function of $\varepsilon = \texttt{0 : 0.1 : 1}$.

As we can observe there exists a "stability area" for both type of graph. But we also can get very poor results, even for values close from the good ones: hence they cannot be qualified as stable.

Question 2.9

In ARI we use the label inherent to the creation of the distribution. They are "true" in the sense that they are the original ones, but they not necessarily are the labels of the best (ie. most useful) clustering. If we did not

have the "true" labels, we could *try to* evaluate the quality of our clustering[4] by looking at the stability of our clustering over experiments or looking if our clusters are homogeneous (but we must define a measure for homogeneity before, not an easy task).

Question 2.10

We used `eig` and not `eigs` in the `spectral_clustering` and `spectral_clustering_adaptative` functions.

- The first one, `eig(A)` computes *all* the eigen-values (and eigen-vectors) of the matrix (solves the eigen problem $Ax = \lambda x$), and `eig(A, D)` computes *all* the eigen-values (and eigen-vectors) of the matrix (solves the *generalized* eigen problem $Ax = \lambda Dx$).
- The second one `eigs` offers many options, but simply said: `eigs(A, K)` computes *only* the first K eigen-values (and eigen-vectors), and `eigs(A, B, K)` gives the first K solution to the generalized problem ($Ax = \lambda Bx$).

Therefore, using `eigs` rather than `eig` is useful when we need to compute only few eigen-values. For spectral clustering, we usually only need the smallest ones, and computing for example only the 20 smallest ones should be faster than computing all of them (and it is enough for our examples here, as we saw where is the bend each time). In practice, I did not observed any speedup on this aspect...

Question 2.11

`eigs` scale to large graphs better that `eig` but both will be limited for very large graphs. We could use parallelization for speeding up the construction process, but I do not know exactly how (how to split up the data and so the graph in sub problems that can be run in parallel?).

Question 2.12

We used k-means, but I don't know what *thresholding* refers to for this situation.

Problem 3 : Image Segmentation

Question 3.1

Cf. `image_segmentation.m`. We have not done anything too complicated or fancy here.

We chose the ε-graph variant, with a small $\varepsilon = 0.005$ and a high value for $\sigma^2 = 90$ (in order to have a densely connected graph). We tried to cluster the image into 6 clusters (the white background, the gray shadows and 4 balls – I also tried to assimilate white and gray, to just have 5 clusters).

As suggested, we improved the function `exponential_euclidean` to use `pdist` to be more efficient, and `build_similarity_graph` for ε graph with a one-liner `W = similarities .* (similarities >= eps)` which is way quicker than two `for` loops (Matlab/Octave have this weakness). Still, our code was quite long to run (22 seconds for building the 5000×3 similarity graph, 29 seconds to build to Laplacian, 75 seconds for non-adaptive clustering, 82 for adaptive clustering...).

First, we tried with non-adaptive spectral clustering, by choosing manually 5 eigen values (I also tried other values), see Figure 12:

[4] The page https://en.wikipedia.org/wiki/Cluster_analysis#Evaluation_and_assessment explains other possibilities.

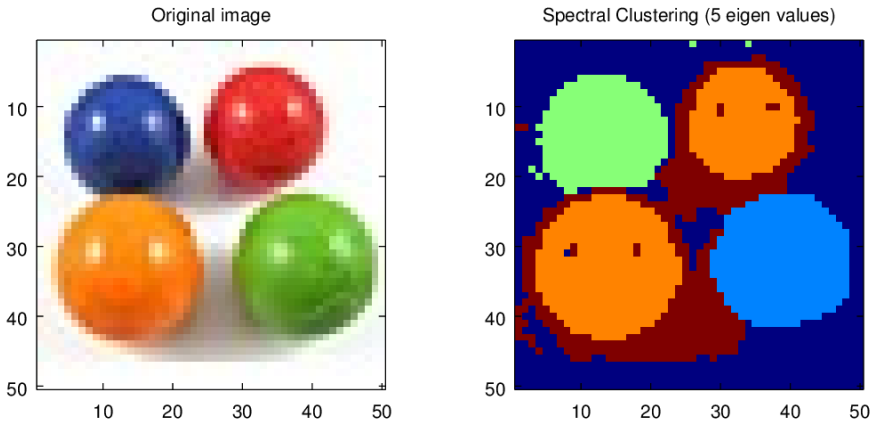

Figure 12: Example of non-adaptive image segmentation (5 eigen values).

Then, the adaptive version gives similar results, and we clearly have been able to obtain better results with the adaptive one, see Figure 13:

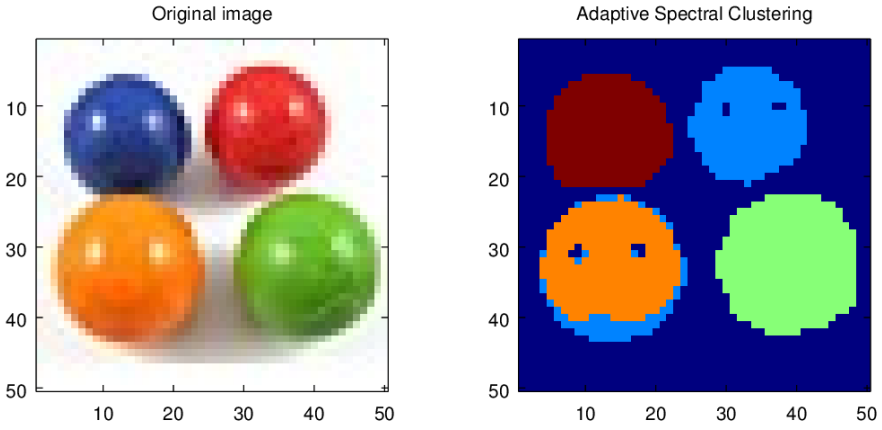

Figure 13: Example of a "good" image segmentation (with "elbow" rule).

We also tried to implement two tricks found on-line: one is to add two more features, x and y coordinates (`1:50` here), but it failed greatly; and the other one is a trick to assimilate white and grays by removing to each RGB triplet (r, g, b) its mean (so white and any grays (g, g, g) will be projected to the same color $(0, 0, 0)$), and it worked quite well (we used it for the two figures 12 and 13 above).

Question 3.2

We could observe the image at different scales, for example we could reduce its size to something we can cluster more easily, by averaging pixels (it would be like blurring the image). Then if we want to use all the pixels we have, we can cluster on the clusters previously obtained in a recursive manner. Also we could parallelize the computations, as said in 2.10 (even if I don't know how to do it).

Conclusion

Remark: I found the TP to be really too long, and both hard and tedious, so I really did not enjoyed it, sorry.
Note: I edited the images with Gimp in order to remove useless white parts, but nothing was modified.