Projet M1 MFA

Classification non supervisée par la méthode de clustering spectral

---

Samy Braik et Ilias Badaoui
22321146 et 22312119

Mai 2024

1 Le partitionnement de données

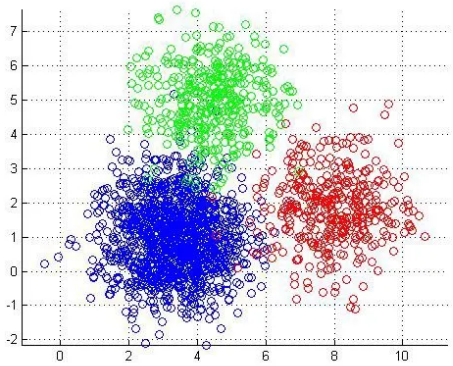
Possédant un jeu de données, une des choses naturelles que l'on voudrait faire est de le diviser par catégories "cohérentes".

C'est en cela que consiste le partitionnement de données. On cherche à classer des données dans des paquets en fonction de caractéristiques partagées, facilitant ainsi l'analyse ou la résolution de problèmes associés.

Le principal objectif des différentes techniques de partitionnement est de maximiser la cohérence et l'homogénéité entre les éléments d'une même classe tout en maximisant la disparité entre les différentes classes.

Plusieurs processus se proposent de répondre à cet exercice, parmi eux il y a l'incontournable algorithme des K-means, le partitionnement spectral, le DBSCAN, la méthode KPPV, et bien d'autres. Quant à nous, nous étudierons le cas du partitionnement spectral.

Le partitionnement est essentiel en analyse de données et en apprentissage automatique. De nombreuses applications existent, comme par exemple en biologie où il est crucial pour identifier des similitudes entre des groupes de patients affectés par une même maladie ou encore pour repérer des gènes associés à des caractéristiques spécifiques.



Il y a plusieurs manières de classifier ces données. Par exemple, soit on veut classifier des données en fonction de catégories déjà connues ; on parle alors de classification supervisée. Le KPPV est, par exemple, une méthode de classification supervisée. Ou alors, on possède simplement des données et on laisse notre méthode les classifier sans connaître les classes au préalable.

L'idée fondamentale derrière l'algorithme de classification spectrale consiste à représenter un jeu de données dans le domaine spectral afin de pouvoir utiliser des propriétés algébriques bien connues sur les matrices.

Nous allons introduire des concepts mathématiques fondamentaux afin de construire notre méthode, avant d'implémenter l'algorithme pour le comparer au fameux algorithme des K-means.

2 Définition des concepts de base

Nous allons capturer l'information contenue dans les données via deux matrices, la matrice de similarité et la matrice diagonale des degrés. Puis s'en servir pour créer une troisième matrice et jouer avec ses propriétés algébriques afin de mettre en évidence des propriétés sur les clusters.

Définition 2.1 Graphe pondéré
Un graphe pondéré est un triplet $G = (V, E, c)$ où V est l'ensemble des sommets, E est l'ensemble des arêtes, et $c : E \to \mathbb{R}_+$ tel que $\forall u, v \in E$, $c(u, v)$ est le poids de l'arête (u, v).

Définition 2.2 Matrice de similarité
La première matrice introduite est la matrice de similarité ; elle fait apparaître les similarités entre chaque paire d'éléments. Cette matrice permet de mettre en évidence les structures et les motifs sous-jacents dans les données, ce qui facilite la découverte de groupes ou de clusters naturels lors de l'analyse des données. Étant donné un ensemble de données $x_1, ..., x_n$, une matrice de similarité est une matrice symétrique $S = (s_{i,j})_{1 \leq i,j \leq n}$ où $s_{i,j}$ représente la "similarité" entre le point x_i et le point x_j.
A partir de cette matrice, il est possible de construire le graphe de similarité.

Remarque
Pour construire un graphe de similarité, il existe plusieurs possibilités. On peut considérer le ε-voisinage où on relie un sommet à tous les autres sommets qui sont à une distance inférieure à ε. Le graphe n'est alors pas pondéré.
On peut aussi utiliser une fonction de similarité comme la fonction de similarité gaussienne définie par $s(x_i, x_j) = \exp(-\|x_i - d_j\|^2/2\sigma^2)$. On obtient alors un graphe complet qui peut s'avérer moins pratique comme on le verra plus tard. Cette fonction est aussi assez sensible aux paramètres σ et les résultats peuvent changer du tout au tout.
Enfin, on peut considérer le graphe qui relie le sommet x_i à ses k-voisins les plus proches et pondérer chaque arête par la similarité entre les points qu'elle relie.

Définition 2.3 Matrice diagonale des degrés
La matrice des degrés est construite à partir de la matrice de similarité. C'est une matrice diagonale où chaque élément sur la diagonale représente le degré du nœud correspondant dans le graphe. Le degré d'un nœud est la somme des poids (similarités) de toutes les arêtes qui sont connectées à ce nœud. En considérant $G_s = (V, E, c)$ un graphe de similarité défini comme ci-dessus, on appelle matrice diagonale des degrés la matrice D telle que

$$d_{i,i} = \sum_j s_{i,j}.$$

Dans la suite, on notera d_i le coefficient $d_{i,i}$

3 Matrices laplaciennes et propriétés

Il existe plusieurs définitions de la matrice laplacienne, la plus simple étant la matrice laplacienne non normalisée.

Definition 3.1 Matrice laplacienne non normalisée
En considérant S une matrice de similarité et D la matrice diagonale des degrés, on appelle matrice laplacienne non normalisée la matrice L définie par $L = D - S$.
Cette matrice est utilisée car elle permet d'obtenir des propriétés intéressantes qui donnent des informations sur le graphe de similarité. Les valeurs propres et vecteurs propres de la matrice laplacienne permettent de déterminer des composantes fortement et faiblement connexes du graphe de similarité. En ayant détermines ces composantes, on pourra effectuer des coupes du graphes, c'est-à-dire une partition des sommets du graphe, et donc des clusters.

Proposition 3.2 Propriétés de la matrice laplacienne non normalisée

1. $\forall u \in \mathbb{R}^n,\ u^\intercal L u = \frac{1}{2} \sum_{i,j}^n s_{i,j}(u_i - u_j)^2$
2. L est symétrique et définie positive.
3. La plus petite valeur propre de L est 0 et le vecteur propre associé est le vecteur constant composé de 1.
4. L possède n valeurs propres réelles positives.

Preuve

1.

$$\begin{aligned} u^\intercal Lu &= u^\intercal Du - u^\intercal Su \\ &= \sum_{i=1}^{n} u_i^2 d_i - \sum_{i,j} u_i s_{ij} u_j \\ &= \frac{1}{2}\Big(\sum_{i=1}^{n} d_i u_i^2 - 2\sum_{i,j} u_i u_j s_{ij} + \sum_{j=1}^{n} d_j u_j^2 \Big) \\ &= \frac{1}{2} \sum_{i,j} s_{ij}(u_i - u_j)^2, \text{ car } d_{i,i} = \sum_j s_{i,j}. \end{aligned}$$

2. L est symétrique comme différence d'une matrice diagonale et d'une matrice symétrique et définie positive par la première propriété.
3. On considère $u = (1, ..., 1) \in \mathbb{R}^n$.
   On a que $Lu = Du - Su = 0$ car $Du = Su$.
   Donc 0 est une valeur propre associée au vecteur propre constant composé de 1.
   C'est la plus petite valeur propre car L est définie positive.
4. L admet n valeurs propres réelles positives car elle est symétrique réelle, donc le théorème spectral s'applique, et définie positive.

4 Matrices laplaciennes normalisées

La matrice laplacienne est essentielle pour le clustering spectral, et peut être normalisée de différentes manières pour améliorer l'analyse des données. On considère L une matrice laplacienne non normalisée, S et D les matrices de similarité et diagonale des degrés associées.
Voici deux méthodes courantes de normalisation :

Normalisation par Division 4.1

La matrice laplacienne normalisée par division est définie comme suit :

$$L_d := D^{-1}L = I - D^{-1}S$$

Normalisation par Division Symétrique 4.2

La matrice laplacienne normalisée par division symétrique est définie par :

$$L_{ds} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} S D^{-\frac{1}{2}}$$

Ces normalisations permettent de prendre en compte les différentes échelles des degrés des nœuds et d'améliorer la performance et la précision du clustering spectral.

Tout comme pour le cas de la matrice laplacienne non normalisée, on a quatre propriétés.

Propriétés des matrices laplaciennes normalisées 4.3

1. $\forall x \in \mathbb{R}^n,\ x^\intercal L_{ds} x = \frac{1}{2} \sum_{i,j}^{n} s_{ij} \left(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}} \right)^2$
2. λ est valeur propre de L_d de vecteur propre u si et seulement si λ est valeur propre de L_{ds} de vecteur propre $D^{\frac{1}{2}} u$
3. λ est valeur propre de L_d de vecteur propre u si et seulement si λ est solution du problème aux valeurs propres généralisé $Lu = \lambda Du$
4. L_d et L_{ds} sont semi-définies positives et admettent n valeurs propres réelles positives

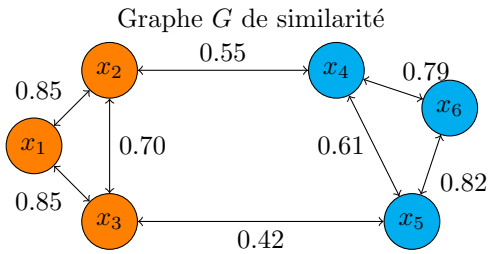
Preuve

1. Soit $x \in \mathbb{R}^n$.
   En remarquant que $x^\intercal D^{-\frac{1}{2}} = \left(\frac{x_1}{\sqrt{d_1}}, ..., \frac{x_n}{\sqrt{d_n}} \right)^\intercal$ et que $D^{-\frac{1}{2}} x = \left(\frac{x_1}{\sqrt{d_1}}, ..., \frac{x_n}{\sqrt{x_n}} \right)$
   On a le résultat en appliquant directement la première propriétés énoncée dans le paragraphe 3.2.
2. Soit λ une valeur propre de L_d de vecteur propre u. On a donc $L_d u = \lambda u \iff D^{-1} L u = \lambda u \iff D^{-\frac{1}{2}} L u = \lambda D^{\frac{1}{2}} u$. On substitue alors u par $D^{-\frac{1}{2}} w$ et on a $D^{-\frac{1}{2}} L D^{-\frac{1}{2}} w = \lambda w$. Or, $w = D^{\frac{1}{2}} u$ d'où l'équivalence.
3. Soit λ valeur propre de L_d de vecteur propre u, on a alors $L_d u = \lambda u \iff D^{-1} L u = \lambda u \iff L u = \lambda D u$.
4. Tout comme la matrice laplacienne non normalisée, les matrices laplaciennes normalisées sont symétriques réelles par construction. L_{ds} est définie positive par la première propriété et on peut déduire par la seconde propriété, en effectuant la bonne substitution, que L_d est également définie positive. D'où le fait qu'elles admettent toutes deux n valeurs propres réelles positives.

5 Coupe de graphe

Comme discuté précédemment, on cherche à partitionner notre ensemble de données de sorte à ce que chaque partie contiennent les données les plus semblables possibles entre elles. Cela revient donc à chercher à identifier des ensembles de sommets tels que la similarité entre les sommets d'un même ensemble soit forte mais que la similarité entre les sommets en dehors de l'ensemble et ceux en son sein soit faible.

Exemple



Plusieurs méthodes sont définies pour pouvoir partitionner le graphe de manière pertinente. Avant de les introduire on définit quelques notations. On considérera pendant le reste du chapitre $G = (V, E, c)$ un graphe de similarité.

Notations 5.1
Soit $k \in \mathbb{N}$ et $(A_i)_{i=1,...,k}$ une partition de V. On définit

$$cut(A_1, ..., A_k) := \frac{1}{2} \sum_{j=1}^{k} W(A_j, A_j^{\complement})$$

où $W(A, B) := \sum_{i \in A, j \in B} c(x_i, x_j) = \sum_{i \in A, j \in B} s_{i,j}$

L'objectif serait alors de minimiser cette fonction afin d'avoir une similarité entre deux points issus d'ensembles distincts la plus faible possible, or procéder ainsi pourrait créer des partitionnements absurdes. On pourrait, par exemple, obtenir un ensemble de sommets réduit à un singleton.
Pour s'épargner de tels cas, d'autres fonctions objectifs ont été introduites.

Coupe normalisée et ratio de coupe 5.2
Soit $k \in \mathbb{N}$ et $(A_i)_{i=1,...,k}$ une partition de V. On définit la coupe normalisée $Ncut$ et le ratio de coupe $Rcut$ par :

$$Ncut(A_1, ..., A_k) = \frac{1}{2} \sum_{j=1}^{k} \frac{W(A_j, A_j^{\complement})}{Vol(A_j)} = \sum_{j=1}^{k} \frac{cut(A_j, A_j^{\complement})}{Vol(A_j)}$$

$$Rcut(A_1, ..., A_k) = \frac{1}{2} \sum_{j=1}^{k} \frac{W(A_j, A_j^{\complement})}{\#A_j} = \sum_{j=1}^{k} \frac{cut(A_j, A_j^{\complement})}{\#A_j}$$

où $Vol(A_j) = \sum_{i \in A_j} d_i$
Définir de telles fonctions objectifs permet d'éviter de se retrouver avec des clusters qui contiennent trop peu d'éléments ou qui n'ont pas de sens d'un point de vue similarité.
Dans le cas de la coupe normalisée, la valeur $Vol(A_j)$ diminue si l'ensemble A_j est composé de trop peu de points, faisant augmenter la valeur de la fonction objectif.
Même chose pour le ratio de coupe, où le terme $\#A_j$ augmente la valeur de la fonction objectif si le cluster est trop petit.

Lien entre matrice laplacienne et ratio de coupe 5.3
Dans le cas où on partitionne en deux ensembles, on cherche à résoudre le problème d'optimisation suivant :

$$\min_{A \subset V} Rcut(A, A^{\complement})$$

Pour ce faire, on pose $x = (x_1, ..., x_n) \in \mathbb{R}^n$ avec

$$\forall\, i \in \{1, ..., n\},\ \ x_i = \begin{cases} \sqrt{\frac{\#A^{\complement}}{\#A}} & \text{si } i \in A \\ -\sqrt{\frac{\#A}{\#A^{\complement}}} & \text{si } i \in A^{\complement} \end{cases}$$

On alors les deux propriétés suivantes sur x :

$$\begin{aligned} \sum_{i=1}^{n} x_i &= \sum_{i \in A} \sqrt{\frac{\#A^{\complement}}{\#A}} - \sum_{i \in A^{\complement}} \sqrt{\frac{\#A}{\#A^{\complement}}} = \#A\sqrt{\frac{\#A^{\complement}}{\#A}} - \#A^{\complement}\sqrt{\frac{\#A}{\#A^{\complement}}} \\ &= \sqrt{\#A}\sqrt{\#A^{\complement}} - \sqrt{\#A^{\complement}}\sqrt{\#A} = 0 \end{aligned}$$

Ce qui est équivalent à dire que $< x, \mathbf{1} >= 0$ et donc que x est orthogonal au vecteur $\mathbf{1} = (1, ..., 1) \in \mathbb{R}^n$.

$$\|x\|^2 = \sum_{i=1}^{n} x_i^2 = \#A\frac{\#A^{\complement}}{\#A} + \#A^{\complement}\frac{\#A}{\#A^{\complement}} = \#A^{\complement} + \#A = \#V = n$$

Autrement dit $x^{\intercal}x = n$.

Or par une propriété de la matrice laplacienne non normalisée on a que :

$$
\begin{aligned}
x^\intercal Lx &= \frac{1}{2}\sum_{i,j}^{n} s_{i,j}(x_i - x_j)^2 \\
&= \frac{1}{2}\sum_{i\in A, j\in A^\complement} s_{i,j}\left(\sqrt{\frac{\#A^\complement}{\#A}} + \sqrt{\frac{\#A}{\#A^\complement}}\right)^2 + \frac{1}{2}\sum_{i\in A^\complement, j\in A} s_{i,j}\left(-\sqrt{\frac{\#A^\complement}{\#A}} - \sqrt{\frac{\#A}{\#A^\complement}}\right)^2 \\
&= \frac{1}{2}W(A, A^\complement)\left(\frac{\#A^\complement}{\#A} + \frac{\#A}{\#A^\complement} + 2\right) + \frac{1}{2}W(A^\complement, A)\left(\frac{\#A^\complement}{\#A} + \frac{\#A}{\#A^\complement} + 2\right) \\
&= cut(A, A^\complement)\left(\frac{\#A^\complement}{\#A} + \frac{\#A}{\#A^\complement} + 2\right) \\
&= cut(A, A^\complement)\left(\frac{\#A^\complement}{\#A} + \frac{\#A}{\#A^\complement} + \frac{\#A}{\#A} + \frac{\#A^\complement}{\#A^\complement}\right) \\
&= cut(A, A^\complement)\left(\frac{\#V}{\#A} + \frac{\#V}{\#A^\complement}\right) \\
&= n \cdot Rcut(A, A^\complement)
\end{aligned}
$$

En considérant tout cela, on peut réécrire le problème de minimisation du ratio de coupe en considérant le même vecteur x défini précédemment : $\begin{cases} \min_{A\subset V} x^\intercal Lx \\ < x, \mathbf{1} >= 0 \\ \|x\| = \sqrt{n} \end{cases}$

En généralisant au cas où le vecteur x est un vecteur quelconque de $\mathbb{R}^n$, on a le problème relaxé : $\begin{cases} \min_{x\in\mathbb{R}^n} x^\intercal Lx \\ < x, \mathbf{1} >= 0 \\ \|x\| = \sqrt{n} \end{cases}$.

Sachant que $\sqrt{n} > 0$, on peut normaliser et faire en sorte que la dernière condition soit $\|x\| = 1$, on aurait alors le problème de minimisation $\begin{cases} \min_{x\in\mathbb{R}^n} \frac{x^\intercal Lx}{x^\intercal x} \\ < x, \mathbf{1} >= 0 \\ \|x\| = 1 \end{cases}$.

Or par définition d'une valeur propre et sachant que 0 a pour vecteur propre $\mathbf{1}$ et donc ne vérifie pas la seconde condition, le minimum est la plus petite valeur propre non nulle de la matrice laplacienne non normalisée que l'on notera γ.
On peut alors se ramener à une solution du problème initial en assignant les sommets aux clusters en fonction du signe des composantes de γ.
Ainsi, pour $i \in V$ un sommet, on a que $i \in A$ si $\gamma_i \geq 0$ et $i \in A^\complement$ si $\gamma_i < 0$.
Ou alors, comme dans les algorithmes qui seront introduits, utiliser l'algorithme des K-means pour assigner chaque composantes de x dans un cluster C_j où $j \in \{1, ..., k\}$ puis poser $A_i = \{a \mid x_a \in C_i\}, \forall i \in \{1, ..., k\}$.

Lien entre matrice laplacienne et coupe normalisée 5.4
On procède de la même manière que précédemment en commençant par étudier le cas où on partitionne en deux ensembles et où le problème d'optimisation à résoudre est :

$$\min_{A \subset V} Ncut(A, A^{\complement}).$$

On pose $x_i = \begin{cases} \sqrt{\frac{Vol(A^{\complement})}{Vol(A)}} & \text{si } i \in A \\ -\sqrt{\frac{Vol(A)}{Vol(A^{\complement})}} & \text{si } i \in A^{\complement} \end{cases}$

On a deux propriétés similaires au cas précédent :

$$\begin{aligned} < Dx, \mathbf{1} > = \sum_{i=1}^{n} x_i d_i &= \sum_{i \in A} d_i \frac{\sqrt{Vol(A^{\complement})}}{\sqrt{Vol(A)}} - \sum_{i \in A^{\complement}} d_i \frac{\sqrt{Vol(A)}}{\sqrt{Vol(A^{\complement})}} \\ &= \sum_{i \in A} d_i \frac{\sqrt{\sum_{j \in A^{\complement}} d_j}}{\sqrt{\sum_{j \in A} d_j}} - \sum_{i \in A^{\complement}} d_i \frac{\sqrt{\sum_{j \in A} d_j}}{\sqrt{\sum_{j \in A^{\complement}} d_j}} \\ &= \sqrt{\sum_{j \in A^{\complement}} d_j} \sqrt{\sum_{i \in A} d_i} - \sqrt{\sum_{j \in A} d_j} \sqrt{\sum_{i \in A^{\complement}} d_i} = 0 \end{aligned}$$

$$x^{\intercal} D x = \sum_{i=1}^{n} d_i x_i^2 = \sum_{i \in A} \frac{Vol(A^{\complement})}{Vol(A)} d_i + \sum_{i \in A^{\complement}} d_i \frac{Vol(A)}{Vol(A^{\complement})} = Vol(A^{\complement}) + Vol(A) = Vol(V)$$

Par la même propriété de la matrice laplacienne que dans le cas de la coupe de ratio :

$$\begin{aligned} x^{\intercal} L x &= \frac{1}{2} s_{i,j} (x_i - x_j)^2 \\ &= \frac{1}{2} \sum_{i \in A, j \in A^{\complement}} s_{i,j} \left(\frac{\sqrt{Vol(A^{\complement})}}{\sqrt{Vol(A)}} + \frac{\sqrt{Vol(A)}}{\sqrt{Vol(A^{\complement})}} \right)^2 + \frac{1}{2} \sum_{i \in A^{\complement}, j \in A} s_{i,j} \left(\frac{\sqrt{Vol(A^{\complement})}}{\sqrt{Vol(A)}} + \frac{\sqrt{Vol(A)}}{\sqrt{Vol(A^{\complement})}} \right)^2 \\ &= \frac{1}{2} \sum_{i \in A, j \in A^{\complement}} s_{i,j} \left(\frac{Vol(A^{\complement})}{Vol(A)} + \frac{Vol(A)}{Vol(A^{\complement})} + 2 \right) + \frac{1}{2} \sum_{i \in A^{\complement}, j \in A} s_{i,j} \left(\frac{Vol(A^{\complement})}{Vol(A)} + \frac{Vol(A)}{Vol(A^{\complement})} + 2 \right) \\ &= cut(A, A^{\complement}) \left(\frac{Vol(A^{\complement})}{Vol(A)} + \frac{Vol(A)}{Vol(A^{\complement})} + \frac{Vol(A^{\complement})}{Vol(A^{\complement})} + \frac{Vol(A)}{Vol(A)} \right) \\ &= cut(A, A^{\complement}) \left(\frac{Vol(V)}{Vol(A)} + \frac{Vol(V)}{Vol(A^{\complement})} \right) \\ &= Vol(V) \cdot Ncut(A, A^{\complement}) \end{aligned}$$

On a alors le problème de minimisation de la coupe normalisée suivant $\begin{cases} \min_{x\in\mathbb{R}^n} x^\intercal Lx \\ < Dx, \mathbf{1} >= 0 \\ x^\intercal Dx = Vol(V) \end{cases}$

En posant $y = D^{\frac{1}{2}}x$, on se ramène à $\begin{cases} \min_{y\in\mathbb{R}^n} y^\intercal L_{ds} y \\ < y, D^{\frac{1}{2}}\mathbf{1} >= 0 \\ y^\intercal y = Vol(V) \end{cases}$

puis en normalisant on obtient $\begin{cases} \min_{y\in\mathbb{R}^n} \frac{y^\intercal L_{ds} y}{y^\intercal y} \\ < y, D^{\frac{1}{2}}\mathbf{1} >= 0 \\ y^\intercal y = 1 \end{cases}$.

Comme dans le cas précédent, c'est la seconde plus petite valeur propre δ de L_{ds} qui est solution du problème.
On revient à la solution du problème initiale : pour $i \in V$, $i \in A$ si $\delta_i \geq 0$ et $i \in A^{\complement}$ si $\delta_i < 0$.

Généralisation au cas à k clusters avec le ratio de coupe 5.5
On fixe $k \in \mathbb{N}$ et $A_1, ..., A_k$ une partition de l'ensemble des sommets V.
On veut alors résoudre le problème de minimisation suivant : $\min_{A_1,...,A_k \subset V} Rcut(A_1, ..., A_k)$
Un peu comme dans le cas à deux clusters, on va définir k vecteurs u_j où $j \in \{1, ..., k\}$ et se ramener à une expression de la coupe de ratio sur les ensembles $(A_1, .., A_k)$ à l'aide des vecteurs définis et de la matrice laplacienne.
On définit la famille de vecteurs $(u_j)_{j=1,...,k} \in \mathbb{R}^n$ par :

$$u_{i,j} = \begin{cases} \frac{1}{\sqrt{\#A_j}} \text{ si } i \in A_j \\ 0 \text{ sinon} \end{cases} \qquad \forall i \in \{1, ..., n\},\ \forall j \in \{1, ..., k\}$$

On pose alors $U \in \mathcal{M}_{n,k}(\mathbb{R})$ avec comme colonnes de la matrice les vecteurs u précédemment définis.
On remarque que $\forall m \in \{0, ..., k\}$

$$(U^\intercal LU)_{m,m} = \sum_{i=1}^{n}\sum_{j=1}^{n} u_{i,m} \times u_{j,m} \times l_{i,j} = u_m^\intercal L u_m$$

Or par une propriété de la matrice laplacienne, on a

$$\begin{aligned} u_m^\intercal L u_m &= \frac{1}{2}\sum_{i,j}^{n} s_{i,j}(u_{i,m} - u_{j,m})^2 \\ &= \frac{1}{2}\sum_{i\in A_m, j\in A_m^{\complement}} s_{i,j}\left(\frac{1}{\sqrt{\#A_m}}\right)^2 + \frac{1}{2}\sum_{i\in A_m^{\complement}, j\in A_m} s_{i,j}\left(-\frac{1}{\sqrt{\#A_m}}\right)^2 \\ &= \frac{1}{2}\frac{W(A_m, A_m^{\complement})}{\#A_m} = \frac{cut(A_m, A_m^{\complement})}{\#A_m} \end{aligned}$$

Par définition, $Rcut(A_1, ..., A_k) = \sum_{i=1}^{k} \frac{cut(A_i, A_i^{\complement})}{\#A_i} = \sum_{i=1}^{k} u_i^{\intercal} L u_i = \sum_{i=1}^{k} (U^{\intercal} L U)_{i,i} = Tr(U^{\intercal} L U)$. De plus, par définition les vecteurs $(u_j)_{j=1,...,k}$ sont orthogonaux entre eux et $\forall m = 0, ..., k$, $\sum_{i=1}^{n} u_{i,m}^2 = \sum_{i \in A_m} \left(\frac{1}{\sqrt{\#A_m}}\right)^2 = \frac{\#A_m}{\#A_m} = 1$
Donc $U^{\intercal} U = I$
Le problème de minimisation du ratio de coupe peut alors se réécrire comme $\begin{cases} \min_{A_1,...,A_m \subset V} Tr(U^{\intercal} L U) \\ U^{\intercal} U = I \end{cases}$ qui peut être relaxer et devenir $\begin{cases} \min_{U \in \mathcal{M}_{n,k}(\mathbb{R})} Tr(U^{\intercal} L U) \\ U^{\intercal} U = I \end{cases}$
La solution est alors est donnée par la matrice U qui est composée des vecteurs propres associés aux k plus petites valeurs propres de la matrice laplacienne non normalisée.
On se ramène alors à une solution du problème initial en utilisant l'algorithme des K-means sur les lignes de la matrice U et en partitionnant les données en fonctions du cluster de chacune des lignes de la matrice U.

Généralisation au cas à k clusters avec la coupe normalisée 5.6
On fixe $k \in \mathbb{N}$ et $A_1, ..., A_k$ une partition de l'ensemble des sommets V.
On cherche à résoudre le problème de minimisation $\min_{A_1,...,A_k \subset V} Ncut(A_1, ..., A_k)$
On définit la famille de vecteurs $(v_j)_{j=1,...,k} \in \mathbb{R}^{n \times k}$ par

$$v_{i,j} = \begin{cases} \frac{1}{\sqrt{Vol(A_j)}} \text{ si } i \in A_j \\ 0 \text{ sinon} \end{cases} \qquad \forall i \in \{1, ..., n\},\ \forall j \in \{1, ..., k\}$$

et on pose $V \in \mathcal{M}_{n,k}(\mathbb{R})$ avec comme colonnes de la matrice les vecteurs u précédemment définis.
Même chose qu'auparavant, $(V^{\intercal} L V)_{m,m} = v_m^{\intercal} L v_m$ et

$$\begin{aligned} v_m^{\intercal} L v_m &= \frac{1}{2} \sum_{i,j}^{n} s_{i,j} (v_{i,m} - v_{j,m})^2 \\ &= \frac{1}{2} \sum_{i \in A_m, j \in A_m^{\complement}} s_{i,j} \left(\frac{1}{\sqrt{Vol(A_m)}}\right)^2 + \frac{1}{2} \sum_{i \in A_m^{\complement}, j \in A_m} s_{i,j} \left(-\frac{1}{\sqrt{Vol(A_m)}}\right)^2 \\ &= \frac{1}{2} \frac{W(A_m, A_m^{\complement})}{Vol(A_m)} = \frac{cut(A_m, A_m^{\complement})}{Vol(A_m)} \end{aligned}$$

Par définition,

$$Ncut(A_1, ..., A_k) = \sum_{i=1}^{k} \frac{1}{2} \frac{W(A_i, A_i^{\complement})}{Vol(A_i)} = \sum_{i=1}^{k} (V^{\intercal} L V)_{i,i} = Tr(V^{\intercal} L V)$$

En remarquant que les vecteurs $(v_j)_{j=1,...,k}$ sont orthogonaux et que

$$\forall m \in \{1, ..., k\},\ (V^{\intercal} D V)_{m,m} = \sum_{i=1}^{n} d_i v_{i,m}^2 = \frac{Vol(A_m)}{Vol(A_m)} = 1 \text{ on a } V^{\intercal} D V = I$$

Le problème de minimisation de la coupe normalisée peut s'écrire $\begin{cases} \min_{A_1,...,A_m \subset V} Tr(V^\intercal L V) \\ V^\intercal D V = I \end{cases}$

On peut alors faire apparaitre la matrice laplacienne normalisée par division symétrique en substituant V par $W = D^{\frac{1}{2}} V$.

Le problème devient donc $\begin{cases} \min_{A_1,...,A_m \subset V} Tr(W^\intercal D^{-\frac{1}{2}} L D^{-\frac{1}{2}} W) \\ W^\intercal W = I \end{cases}$

qui après relaxation est $\begin{cases} \min_{W \in \mathcal{M}_{n,k}(\mathbb{R})} Tr(W^\intercal L_{ds} W) \\ W^\intercal W = I \end{cases}$

La solution est alors est donnée par la matrice W qui est composée des vecteurs propres associés aux k plus petites valeurs propres de la matrice laplacienne normalisée par division symétrique. On se ramène alors à une solution du problème initial en utilisant l'algorithme des K-means sur les lignes de la matrice W et en partitionnant les données en fonction du cluster de chacune des lignes de la matrice W.

Marche aléatoire et coupe normalisée 5.7

Une autre manière d'exprimer la coupe normalisée serait d'utiliser un argument probabiliste.

En remarquant que la probabilité de passer du sommet i au sommet j est donnée par $p_{i,j} = \frac{s_{i,j}}{d_i}$, on peut construire la matrice de transition $P = D^{-1} S$.

En posant, $\pi = (\pi_i)_{i=1,...,n}$ définie par $\pi_i = \frac{d_i}{Vol(V)}$.

On a alors que $(P^\intercal \pi)_i = \sum_j \frac{s_{i,j}}{d_i} \frac{d_i}{Vol(V)} = \frac{1}{Vol(V)} \sum_j s_{i,j} = \pi_i$ donc $P^\intercal \pi = \pi$.

Ainsi π est la probabilité stationnaire de la chaîne de Markov définie par P.

On peut alors faire un lien entre P et la coupe normalisée.

On considère $A, B \subset V$ on pose $P_{A,B}$ la probabilité de passer d'un état dans A à un état dans B.

On a alors

$$P_{A,B} = \frac{\sum_{\in A, j \in B} \pi_i p_{i,j}}{\pi(A)} = \frac{\sum_{i \in A, j \in B} s_{i,j}}{Vol(V)} \frac{Vol(V)}{Vol(A)} = \frac{\sum_{i \in A, j \in B} s_{i,j}}{Vol(A)}$$

D'où le fait que $Ncut(A, A^\complement) = P_{A,A^\complement} + P_{A^\complement,A}$

Remarque : Différence avec l'algorithme des K-means 5.8

L'algorithme des K-means fonctionne comme suit :

En supposant que l'on cherche $k \in \mathbb{N}$, on initialise k points $m_1^{(1)}, ..., m_k^{(1)}$ pris au hasard dans le jeu de données qui seront le centre des premiers clusters.

On répète alors les deux opérations suivantes jusqu'à ce que l'on observe une convergence :

On définit k clusters par $C_i^{(t)} = \left\{ x_j : \|x_j - m_i^{(t)}\| \leq \|x_j - m_n^{(t)}\| \forall n = 1, ..., k \right\}$, $i \in \{1, ..., k\}$

Puis on définit de nouveaux centres : $m_i^{(t+1)} = \frac{1}{\#C_i^{(t)}} \sum_{x_j \in C_i^{(t)}} x_j$

Contrairement à l'algorithme de clustering spectral, l'algorithme des K-means ne fonctionne pas bien si les clusters sont non convexes puisque par définition des clusters produits par l'algorithme des K-means ces derniers sont convexes.
De plus, l'algorithme des K-means peut se voir converger vers un minimum local. En effet si un point est très détaché du reste des données, il peut être choisi comme centre d'un cluster qui deviendrait alors un singleton rendant le clustering absurde. Pour empêcher ce genre de résultat, l'algorithme doit être exécuté plusieurs fois, en initialisant les centres des premiers clusters différemment.
A l'inverse, la méthode de clustering spectral s'affranchit de ces problèmes.
On peut même l'appliquer sans disposer des données, à condition d'avoir la matrice de similarité.

6 Description des algorithmes

Algorithme clustering spectral non normalisé 6.1

Pseudo-code :
Entrée : Données $(x_i)_{1\leq i\leq n} \in \mathbb{R}^d$ et k le nombre de clusters.
· Construire la matrice de similarité en utilisant la fonction de similarité $s(x_i, x_j) = \exp(-\|x_i - x_j\|^2/2\sigma^2)$ et la matrice diagonale des degrés D.
· Construire la matrice laplacienne non normalisée L.
· Calculer les k premiers vecteurs propres de L.
· Construire U la matrice où les colonnes sont les k vecteurs propres précédemment calculés.
· Utiliser l'algorithme des K-means pour partitionner les vecteurs obtenus à partir des lignes de la matrice U dans les clusters $C_1, ..., C_k$.
· Partitionner les données dans les clusters $A_1, ..., A_k$ tels que $A_i = \{j | y_j \in C_i\}$.

Implémentation en Python

```
1
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from sklearn.metrics.pairwise import euclidean_distances
5  from sklearn.cluster import KMeans
6
7  n=600 #Nombre de points
8  k = 3 #Nombre de clusters
9  delta = 5
10 S = np.zeros((n,n)) #Matrice de similarité
11
12 #Extraction des données
13 data = list()
14 X = list()
15 Y = list()
16 g = open("data.txt","r")
17 for l1 in g:
18     x = (float(l1.rsplit(",")[0]),float(l1.rsplit(",")[1]))
19     data.append(x)
20     X.append(float(l1.rsplit(",")[0]))
21     Y.append(float(l1.rsplit(",")[1]))
22 g.close()
23
```

```
#Calcul de la matrice de similarité
def GSF(d):
    return np.exp(-delta * d **2)

GSF_v = np.vectorize(GSF)
S = euclidean_distances(data,data)
S = GSF(S)

#Calcul de la matrice laplacienne non normalisée
D = np.diag(np.sum(S, axis=1))
L = np.subtract(D,S)

#Calcul des k premiers vecteurs propres de la matrice laplacienne
valeurP, vecteurP = np.linalg.eig(L)
indice = np.argsort(valeurP)[:k]
vecteurP = vecteurP[:,indice]

#Application de l'algorithme des K-means
label = KMeans(n_clusters=k).fit(vecteurP).labels_

plt.scatter(X,Y,c=label)
plt.show()


```

Algorithme de clustering spectral normalisé de Shi et Malik 6.2

Pseudo-code

Entrée : Données $(x_i)_{1\leq i\leq n} \in \mathbb{R}^d$ et k le nombre de clusters.
· Construire la matrice de similarité en utilisant la fonction de similarité $s(x_i, x_j) = \exp(-\|x_i - x_j\|^2/2\sigma^2)$ et la matrice diagonale des degrés D.
· Construire la matrice laplacienne normalisée $L_d = D^{-1}L$.
· Calculer les k premiers vecteurs propres de L_d
· Construire U la matrice où les colonnes sont les k vecteurs propres précédemment calculés.
· Utiliser l'algorithme des K-means pour partitionner les vecteurs obtenus à partir des lignes de la matrice U dans les clusters $C_1, ..., C_k$.
· Partitionner les données dans les clusters $A_1, ..., A_k$ tels que $A_i = \{j | y_j \in C_i\}$.

Implémentation en Python

```
1
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from sklearn.metrics.pairwise import euclidean_distances
5  from sklearn.cluster import KMeans
6
7  n=600 #Nombre de points
8  k = 3 #Nombre de clusters
9  delta = 5
10 S = np.zeros((n,n)) #Matrice de similarité
11
12 #Extraction des données
13 data = list()
14 X = list()
15 Y = list()
16 g = open("data.txt","r")
17 for l1 in g:
18     x = (float(l1.rsplit(",")[0]),float(l1.rsplit(",")[1]))
19     data.append(x)
20     X.append(float(l1.rsplit(",")[0]))
21     Y.append(float(l1.rsplit(",")[1]))
22 g.close()
23
24 #Calcul de la matrice de similarité
```

```
25  def GSF(d):
26      return np.exp(-delta * d **2)
27
28  GSF_v = np.vectorize(GSF)
29  S = euclidean_distances(data,data)
30  S = GSF(S)
31
32  #Calcul de la matrice laplacienne normalisée de S&M
33  D = np.diag(np.sum(S, axis=1))
34  L = np.subtract(D,S)
35  Lds = np.matmul(np.linalg.inv(D),L)
36
37  #Calcul des k premiers vecteurs propres de la matrice laplacienne
38  valeurP, vecteurP = np.linalg.eig(L)
39  indice = np.argsort(valeurP)[:k]
40  vecteurP = vecteurP[:,indice]
41
42  #Application de l'algorithme des K-means
43  label = KMeans(n_clusters=k).fit(vecteurP).labels_
44
45  plt.scatter(X,Y,c=label)
46  plt.show()
47
48  ____________________________________________________________________
```

Algorithme de clustering spectral normalisé de Ng, Jordan et Weiss 6.3

Pseudo-code

Entrée : Données $(x_i)_{1\leq i\leq n} \in \mathbb{R}^d$ et k le nombre de clusters.
· Construire la matrice de similarité en utilisant la fonction de similarité $s(x_i, x_j) = \exp(-\|x_i - x_j\|^2/2\sigma^2)$ et la matrice diagonale des degrés D.
· Construire la matrice laplacienne normalisée L_{ds}.
· Calculer les k premiers vecteurs propres de L_{ds}
· Construire U la matrice où les colonnes sont les k vecteurs propres précédemment calculés.
· Construire T la matrice où $t_{i,j} = \frac{u_{i,j}}{(\sum_k u_{i,k}^2)^{1/2}}$
· Utiliser l'algorithme des K-means pour partitionner les vecteurs obtenus à partir des lignes de la matrice T dans les clusters $C_1, ..., C_k$.
· Partitionner les données dans les clusters $A_1, ..., A_k$ tels que $A_i = \{j | y_j \in C_i\}$.

Implémentation en Python

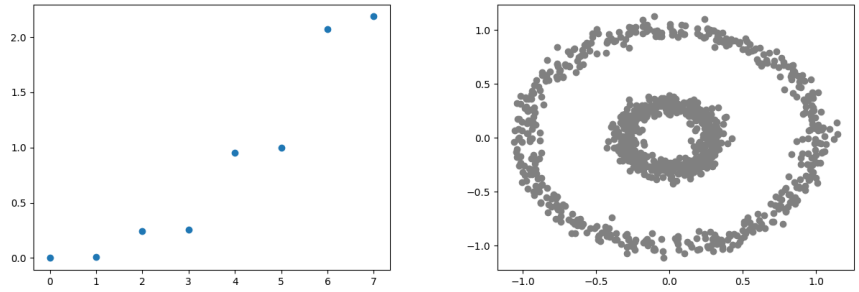
```
1
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from sklearn.metrics.pairwise import euclidean_distances
5  from sklearn.cluster import KMeans
6  from scipy.linalg import fractional_matrix_power
7
8  n=600 #Nombre de points
9  k = 3 #Nombre de clusters
10 delta = 5.5
11 S = np.zeros((n,n)) #Matrice de similarité
12
13 #Extraction des données
14 data = list()
15 X = list()
16 Y = list()
17 g = open("data.txt","r")
18 for l1 in g:
19     x = (float(l1.rsplit(",")[0]),float(l1.rsplit(",")[1]))
20     data.append(x)
21     X.append(float(l1.rsplit(",")[0]))
22     Y.append(float(l1.rsplit(",")[1]))
```

```
g.close()

#Calcul de la matrice de similarité
def GSF(d):
    return np.exp(-delta * d **2)

GSF_v = np.vectorize(GSF)
S = euclidean_distances(data,data)
S = GSF(S)

#Calcul de la matrice laplacienne normalisée de J&W
D = np.diag(np.sum(S, axis=1))
L = np.subtract(D,S)
Dm = fractional_matrix_power(D,-0.5)
Ls = np.matmul(np.matmul(Dm,L),Dm)

#Calcul des k premiers vecteurs propres de la matrice laplacienne
valeurP, vecteurP = np.linalg.eig(L)
vecteurP = vecteurP - np.linalg.norm(vecteurP, axis = 1)[:,None]
indice = np.argsort(valeurP)[:k]
vecteurP = vecteurP[:,indice]

#Application de l'algorithme des K-means
label = KMeans(n_clusters=k).fit(vecteurP).labels_

plt.scatter(X,Y,c=label)
plt.show()
```

Choix du nombre de clusters 6.4

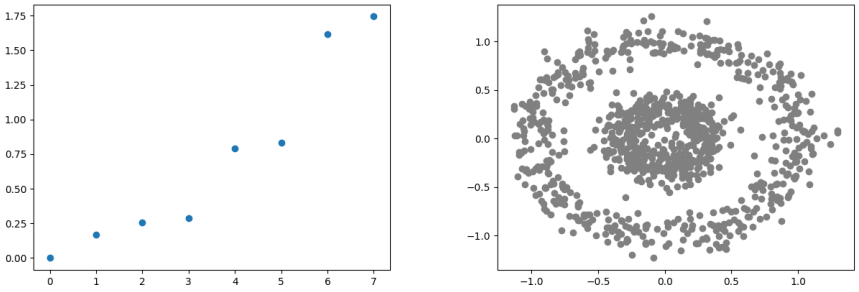
Afin de choisir un nombre adéquat de clusters, on peut utiliser l'heuristique de l'eigengap. On cherche un entier $k \in \{1, ..., n\}$ de sorte à ce que les valeurs propres $\lambda_1, ..., \lambda_k$ soient relativement proches entre elles et proches de 0, mais plus petites que λ_{k+1}.

En revanche, l'efficacité de cette méthode dépend de la présence claire de clusters. Plus il existe des groupes de points denses et disjoints, plus la méthode est efficace.
Ci-dessous une illustration en utilisant l'algorithme de clustering spectral non normalisé.



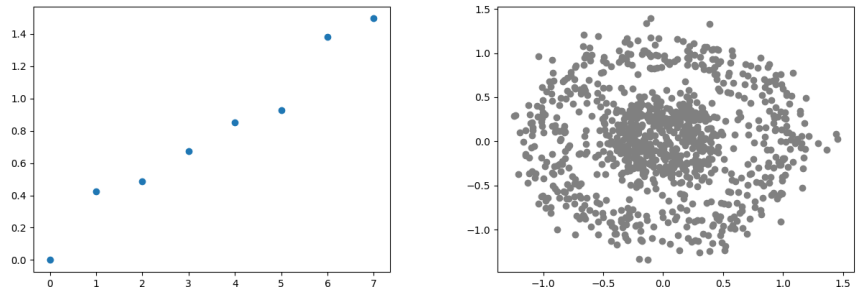
(a) Sept premières valeurs propres (b) Représentation des données utilisées

FIGURE 1 – Lorsque les clusters sont bien apparents



(a) Sept premières valeurs propres (b) Représentation des données utilisées

FIGURE 2 – Lorsque les clusters sont un peu plus difficiles à déterminer



(a) Sept premières valeurs propres (b) Représentation des données utilisées

FIGURE 3 – Lorsqu'aucun cluster n'est visible

En réalité cette approche n'est plus une heuristique dès lors que le graphe de similarité construit n'est plus un graphe connexe. Dans l'exemple précédent, on a construit un graphe de similarité complet grâce à la fonction de similarité gaussienne. En revanche, si on choisit de construire le graphe de similarité en utilisant la méthode des k-voisins les plus proches, on peut obtenir un graphe qui est non connexe mais composé de composantes connexes. Le nombre de ces composantes nous donne directement le nombre de clusters à choisir.

Nombre de composantes connexes et matrice laplacienne 6.5
La multiplicité k du vecteur propre 0 de la matrice laplacienne non normalisée est égale au nombre de composantes connexes du graphe.
Le résultat est vrai si on considère une matrice laplacienne normalisée.

7 Comparaison numérique entre K-means et clustering spectral

Comme on l'a dit précédemment, la méthode de clustering spectral fonctionne beaucoup mieux lorsque les clusters sont des ensembles non convexes. Pour illustrer, on a choisi un exemple classique, à savoir simuler des données de sorte à avoir des nuages de points représentant trois cercles concentriques.

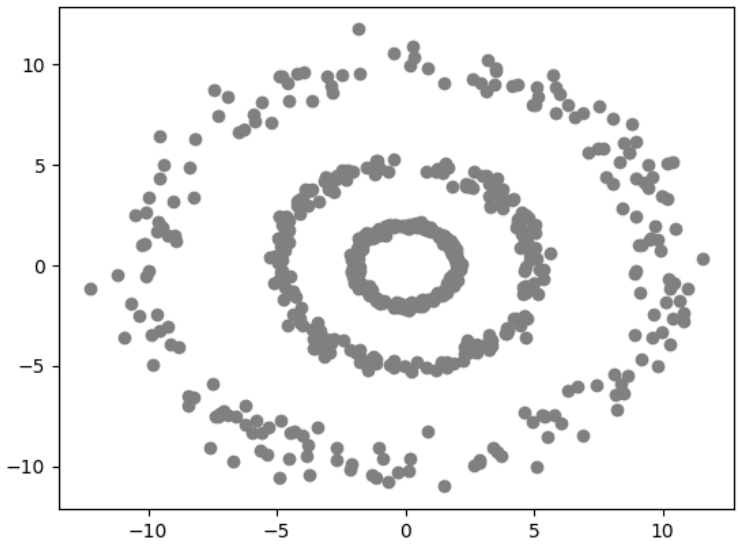


FIGURE 4 – Représentation des données utilisées

En appliquant n'importe lequel des trois algorithmes décrits dans la section 6, le résultat attendu est obtenu.

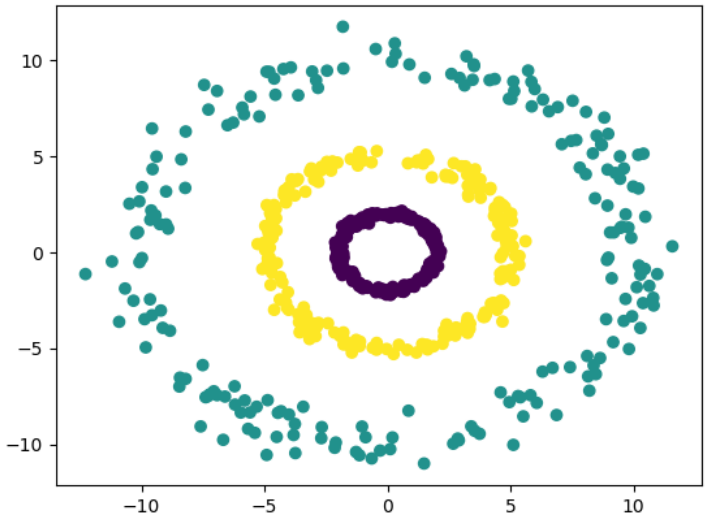


FIGURE 5 – Représentation des données clustérisées à l'aide de l'algorithme de clustering spectral non normalisé

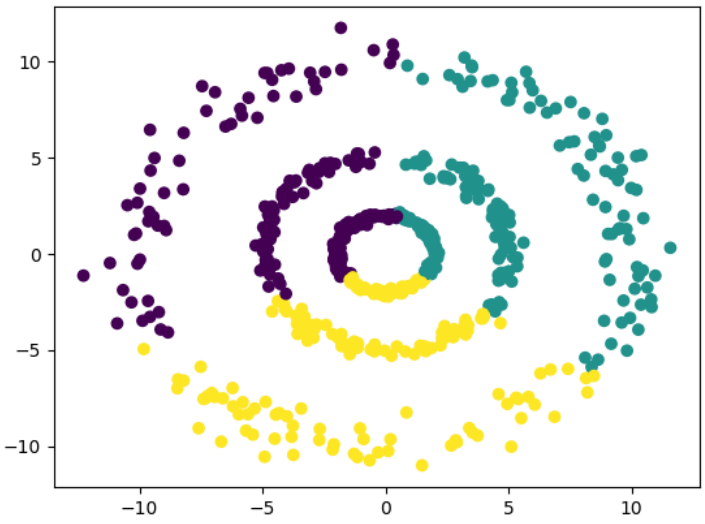


FIGURE 6 – Représentation des données partitionnées à l'aide de l'algorithme K-means

On voit clairement que les clusters résultants de l'algorithme des K-means ne sont pas satisfaisant.
En revanche, sur des données où les clusters forment des ensembles convexes, l'algorithme des K-means présente un avantage. Essentiellement, le résultat obtenu avec un algorithme de clustering spectral et l'algorithme des K-means sera le même, néanmoins la vitesse d'exécution peut être beaucoup plus rapide dans le cas de l'algorithme des K-means.
La complexité en temps d'un algorithme de clustering spectral est $O(n^3)$ tandis que celle de l'algorithme des K-means est $O(ntkd)$ où n est le nombre de points, k le nombre de clusters, t le nombre d'itérations et d la dimension des points.

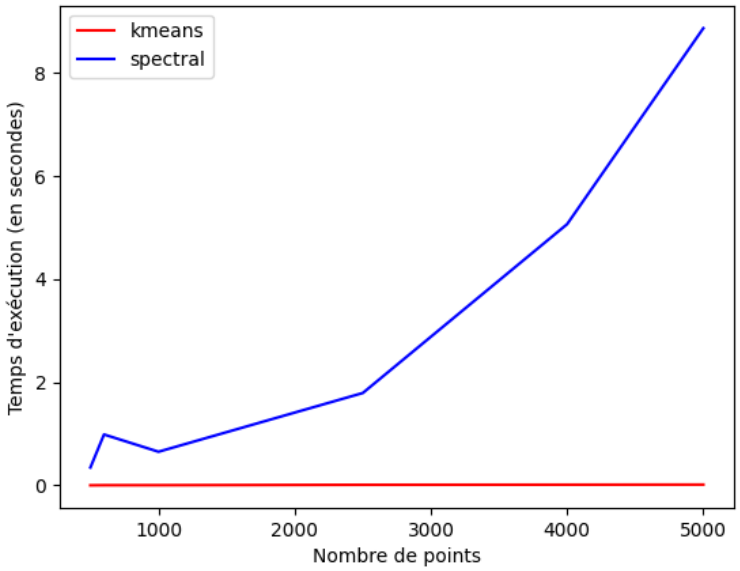


FIGURE 7 – Représentation du temps d'exécution des deux algorithmes en fonction du nombre de points

Références

[1] U von Luxburg, *A Tutorial on spectral Clustering* , Springer, 2007.

[2] L.Zelnik-Manor et P.Perona, *Self-Tuning spectral Clustering*, MIT Press, 2005.

[3] T.Rebafka, *Analyse statistiques de graphes*, LPSM, 2021

[4] A.Marsden, *Eigenvalues of the Laplacian and their relationship to the connectedness of a graph*, University of Chicago REU, 2013

[5] J.H.Gallier, *Fundamentals of Linear Algebra and Optimization*, University of Pennsylvania, 2024