Rapport final

Méthodes spectrales pour la détection de groupes d'affinités dans les grands réseaux

CentraleSupélec

Élèves du groupe 88
Thomas Andal
Mathieu Ha-Sum
Vincent Martin

Chargée de projet
Xiaoyi Mai

Sommaire

Introduction

1 Contexte

Internet et les réseaux sociaux sont devenus des piliers importants de la vie moderne et sont chaque jour un vecteur extrêmement intense de données brutes et de savoirs. De tels outils constituent les lieux idéaux pour recevoir, façonner et échanger les opinions et les sentiments des individus. Étudier et comprendre cette quantité de données brutes et non-structurées est un objectif primordial vers lequel la recherche de demain doit tendre.

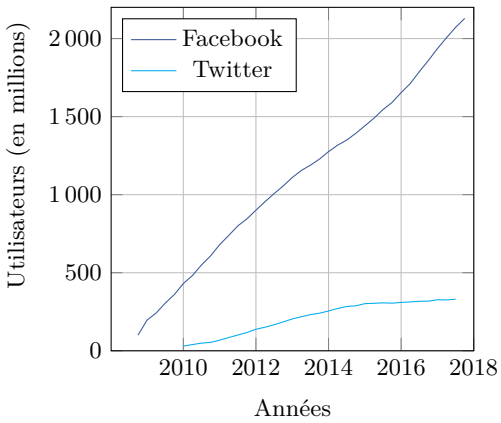


Figure 1 : Croissance des réseaux sociaux

Sur la base de statistiques récentes, le nombre d'internautes dans le monde est d'environ 2,4 milliards d'utilisateurs, soit 35% de la population totale de la planète. Ce nombre a bondi d'environ 361 millions en 2000, soit un taux de croissance d'environ 566%. Pour ce qui est des réseaux sociaux, Facebook compte environ 2,15 milliards d'utilisateurs, soit environ 30% de la population mondiale. Twitter quant à lui, possède environ 500 millions d'utilisateurs. L'ampleur du contenu généré et communiqué quotidiennement implique de devoir fournir des techniques efficaces et réalisables pour transformer ce contenu non structuré en données utilisables.

L'homme est une espèce sociale par nature, nous essayons toujours de nous grouper instinctivement au sein des communautés et des sociétés. Nous avons également tendance à nous regrouper avec des personnes ayant des idéologies et des antécédents légèrement similaires. Les schémas de communication tendent également à s'intensifier pour les membres du même groupe par comparaison avec les membres de l'extérieur. Tous ces principes forment l'essence des principes sociologiques de l'influence où les gens ont finalement tendance à développer des points de vue et des opinions similaires sur différentes questions.

Les réseaux sociaux, ou le monde social virtuel en général, sont un reflet du monde réel. Les gens du monde virtuel se groupent toujours instinctivement. La détection de ces groupements virtuels pourrait être d'une importance particulière pour l'analyse et le suivi des réseaux sociaux, car elle fournit la structure du réseau dans lequel les gens se connectent les uns aux autres.

La recherche dans le domaine de la détection de la communauté pour l'analyse des réseaux sociaux n'est pas nouvelle, cependant, l'application des procédures de détection de communautés pour les réseaux sociaux est relativement jeune.

2 Cahier des charges

Il s'agit dans ce projet de comparer les méthodes spectrales basées sur les matrices d'affinités classiques telles que la matrice d'adjacence, de modularité et la matrice Laplacienne, qui se révèlent peu efficaces lorsque le réseau est "sparse" c'est-à-dire lorsque le degré moyen des noeuds reste constant lorsque la taille du réseau grandit. Un nouvel algorithme basé sur la matrice appelée Bethe Hessian marche beaucoup mieux dans ce cas. L'étude des algorithmes de détection de communautés se fait sur le "Stochastic Block Model", le modèle le plus connu et utilisé dans la littérature.

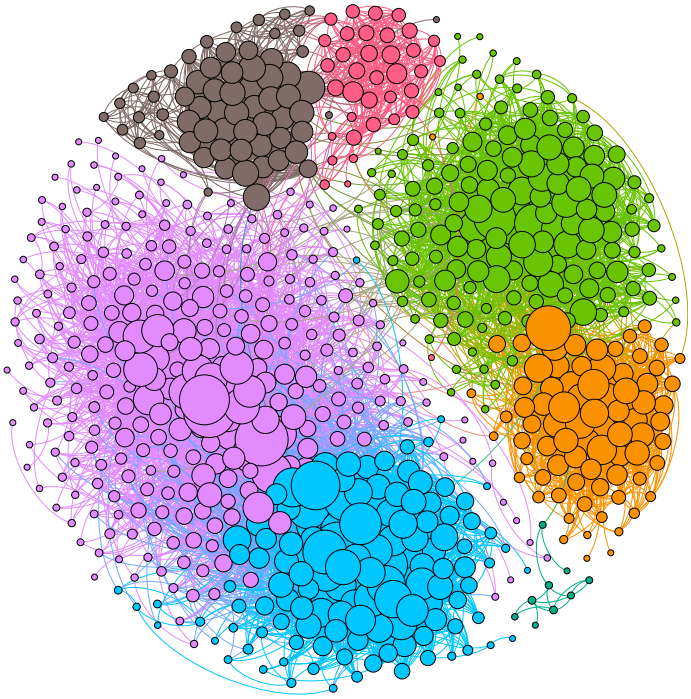


Figure 2 : Exemple de réseau d'amitiés sur Facebook et de détection de communautés

On trouvera ci-dessus un exemple de réseau social dans lequel chaque couleur représente une communauté détectée par la méthode de la modularité. Le diamètre d'un noeud est proportionnel à son degré, c'est-à-dire que plus un noeud est large et plus il est interconnecté avec les autres noeuds qui représentent ici des amitiés. C'est vers ce but final que doit tendre notre étude en permettant la comparaison, tant sur l'exactitude de le détection que de la complexité calculatoire, de nos différents algorithmes.

I Théorie de la détection de groupes d'affinités dans les grands réseaux

1 Représentation matricielle et propriétés intrinsèques d'un réseau

Dans cette sous-partie nous nous efforcerons de détailler les différents outils mathématiques permettant de représenter un réseau sous forme de graphe, avant de définir des indicateurs intrinsèques à celui-ci.

1.1 Graphes et matrice d'adjacence

Définition Un réseau peut être représenté [12] par un graphe G définit par l'ensemble de ses sommets V et l'ensemble de ses connexions E. On note $|V| = n$ le cardinal de V, *i.e.*. le nombre de sommets et on numérote ses éléments arbitrairement $V = \{v_1, ..., v_n\}$. On notera également $|E| = m$ le nombre de connexions observées. On le note alors $G = (V, E)$.

Matrice d'adjacence Un graphe $G = (V, E,)$ tel que $|V| = n$ peut-être décrit de manière univoque par sa matrice d'adjacence $A = (a_{ij}) \in [0, 1]^n \times [0, 1]^n$ définie comme suit :

$\forall (i, j) \in [\![1, n]\!]$, $a_{ij} = \mathbb{1}_E(v_i, v_j)$ où $\mathbb{1}_E$ est la fonction indicatrice de E.

On se limite dans notre étude à des graphes ne comportant pas de boucles, c'est à dire qu'aucune connexion ne lie un sommet à lui même ou de manière équivalente la diagonale de A est nulle.

Pondération On peut adjoindre à ce graphe une fonction $w : E \to \mathbb{R}^+$ qui donne à une connexion un certain poids. On le note alors $G = (V, E, w)$.

Matrice de poids On définit la matrice de poids comme étant la matrice $W = (w(v_i, v_j))_{1 \leq i,j \leq n}$.

Graphe non-orienté Dans le cas d'un graphe non-orienté comme ci-dessous E est composé de paires d'éléments de V que l'on nomme arrête. Par exemple ici $\{v_1, v_2\} \in E$.

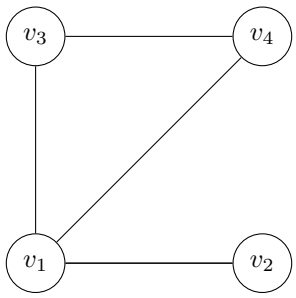


Figure 3 : Exemple de graphe non-orienté

Dans le cas d'un graphe non-orienté la matrice d'adjacence est symétrique et booléenne. Par exemple ci-dessus : $A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$.

Graphe orienté Dans le cas d'un graphe orienté comme ci-dessous E est composé de couples d'éléments de V tels que $(v_i, v_j) \in E$ que l'on nomme arcs. Par exemple ci-dessous $(v_1, v_4) \in E$ mais $(v_4, v_1) \notin E$.

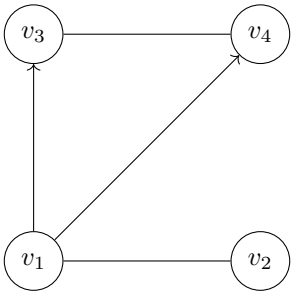


Figure 4 : Exemple de graphe orienté

Dans le cas d'un graphe non-orienté la matrice d'adjacence est non-symétrique et booléenne. Par exemple ci-dessus : $A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$.

Graphe pondéré La pondération nécessite de définir une matrice de poids W.

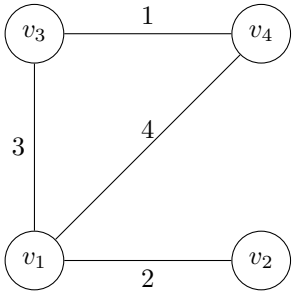


Figure 5 : Exemple de graphe pondéré et non-orienté

Par exemple ci-dessus : $W = \begin{pmatrix} 0 & 2 & 3 & 4 \\ 2 & 0 & 0 & 0 \\ 3 & 0 & 0 & 1 \\ 4 & 0 & 1 & 0 \end{pmatrix}$.

1.2 Chemin, connexité, distance géodésique

Chemin On définit [12] un chemin entre v_i et v_j comme étant une suite d'arêtes d'origine v_i et d'extrémité v_j.

Connexité Un graphe est dit connexe [12] s'il existe au moins un chemin entre chaque paire de noeuds, aucun sommet n'est isolé.

Distance géodésique Sous réserve de connexité, on définit [12] la distance géodésique comme le nombre d'arêtes du plus court chemin d'origine v_i et d'extrémité v_j, deux sommets distincts, elle est notée δ_{ij}. Dans le cas non-orienté $\delta_{ij} = \delta_{ji}$. Dans le cas pondéré on veillera à prendre en compte le poids de chaque arrête ou arc.

Diamètre Sous réserve de connexité, on définit [12] le diamètre d'un graphe comme étant la plus grande distance géodésique entre deux sommets, on le note $d = \max_{i \neq j} \delta_{ij}$.

Exemples

- Le graphe de la figure 3 est connexe, $\delta_{24} = \delta_{42} = 1$ et $d = 2$.

- Le graphe de la figure 4 n'est pas connexe car δ_{32} n'est pas défini.

- Le graphe de la figure 5 est connexe $\delta_{24} = \delta_{42} = 6$ et $d = 6$.

1.3 Densité, voisinage, degré de centralité

Densité La densité d'un graphe est définie [12] par le rapport entre le nombre d'arêtes potentielles sans boucles, elle est notée ρ.

- Dans le cas non-orienté : $\rho \triangleq \frac{|E|}{\frac{|V|(|V|-1)}{2}} = \frac{2|E|}{n(n-1)}$

- Dans le cas orienté : $\rho \triangleq \frac{|E|}{|V|(|V|-1)} = \frac{|E|}{n(n-1)}$

Un graphe sera dit creux ou sparse si ρ est relativement faible, si $|E| = \mathcal{O}(n)$ par exemple [13], dans le cas contraire on dira que le graphe est dense.

Voisinage On définit [12] le voisinage d'un sommet v_i comme l'ensemble des sommets qui lui sont adjacents, on le note N_i.

Degré d'un sommet Dans le cas non-orienté, on définit [12] le degré d'un sommet v_i par $d_i \triangleq \sum_j a_{ij}$ et on remarque que $d_i = |N_i|$. On définit la matrice des degrés comme étant la matrice diagonale de taille n composée des degrés de chaque sommet, on la note $D \triangleq \text{diag}(d_1, ..., d_n)$.

Degré moyen Dans le cas non-orienté, on définit [6] le degré moyen noté $\bar{d} \triangleq \frac{1}{|V|} \sum_i d_i = \frac{2|E|}{n}$.

Force d'un sommet Dans le cas orienté, on définit [1] la force d'un sommet par v_i par $s_i \triangleq \sum_j w_{ij}$. De manière analogue on définit la matrice des sommets comme étant la matrice diagonale de taille n composée des degrés de chaque sommet, on la note $S \triangleq \text{diag}(s_1, ..., s_n)$.

Force moyenne Dans le cas non-orienté, on définit [1] la force moyenne notée $\bar{s} \triangleq \frac{1}{|V|} \sum_i s_i$.

Exemples

- Le graphe de la figure 3 a pour densité $\rho = \frac{4}{6} = \frac{2}{3}$ et $D = \text{diag}(3, 1, 2, 2)$.

- Le graphe de la figure 4 a pour densité $\rho = \frac{6}{12} = \frac{1}{2}$ et $D = \text{diag}(3, 1, 2, 2)$.

- Le graphe de la figure 5 a pour densité $\rho = \frac{4}{6} = \frac{2}{3}$ et $S = \text{diag}(9, 2, 4, 5)$.

2 Génération de graphe aléatoires : Stochastic Block Model (SBM)

Cette partie est consacrée à la génération de graphes aléatoires selon un des modèles le plus répandu [13] Stochastic Block Model qui sous différentes variantes permet de tester les algorithmes de détections des communautés au sein d'un réseau.

2.1 Généralités

Les SBM définissent une distribution de probabilités sur des graphes [4], $\mathbb{P}[G|\Theta]$ où Θ est un ensemble de paramètres qui gouverne le modèle. Sachant Θ on peut alors générer un graphe G à partir de cette distribution. L'inférence est le procédé inverse qui consiste à estimer Θ à partir d'un graphe G donné.

Que ce soit dans le cas pondéré ou non, $\pi \in \Theta$ où $\pi = (\pi_1, ..., \pi_k) \in \mathbb{R}^k$ représente la proportion des sommets appartenant aux k blocs distincts que l'on nommera par la suite communautés [7]. Ainsi on associe aux n sommets une matrice d'appartenance à un bloc notée $B = (b_{ij}) \in \{0,1\}^n \times \{0,1\}^k$, aléatoire et stochastique, et dont chaques lignes sont indépendantes et identiquement distribuées selon une loi multinomiale de paramètre (1, π).

On définit [13] ainsi pour tout i in $[\![1, n]\!]$, g_i la variable aléatoire prenant ses valeurs dans $[\![1, k]\!]$ et représentant le bloc d'appartenance du sommet v_i. Ainsi, $B_{i,j} = 1$ si et seulement si $j = g_i$.

2.2 Cas non-orienté et non-pondéré

Dans le cas non orienté et non-pondéré [7], $\Theta = \{\pi, M\}$ où $M = (m_{ij}) \in [0,1]^k \times [0,1]^k$ est une matrice déterministe et symétrique telle que m_{ij} représente la probabilité qu'une arrête lie un sommet du bloc i à un sommet du bloc j. Le graphe est ainsi représenté par une matrice d'adjacence aléatoire $A = (a_{ij})$. Si $i < j$, sachant B, les a_{ij} sont indépendants, et sachant $g_i = 1$ et $g_j = l$ alors a_{ij} suit une loi binomiale de paramètre m_{kl}. Si $i = j$, alors a_{ij}=0 et si $i > j$ alors $a_{ij} = a_{ji}$.

Exemple pratique

Supposons que $\Theta = \{\pi, M\}$ avec $\pi = (0.75, 0.25)$ et $M = \begin{pmatrix} 0.5 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}$ et on pose $n = 8$ le nombre de sommets.

Dans un premier temps on tire aléatoirement chaque ligne de B selon une loi multinomiale de paramètre $(1, \pi)$. Un tirage nous donne $B = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}^T$. Ce qui correspond à la partition suivante:

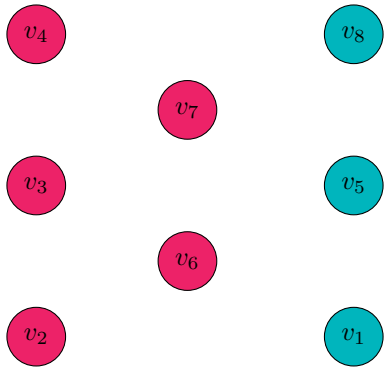


Figure 6 : Une partition des sommets au sein de deux communautés

Ensuite on procède à $\frac{8 \times 7}{2} = 28$ tirages d'une loi binomiale dont :

- 10 de paramètre 0.5 pour les arrêtes rouge/rouge
- 3 de paramètre 0.9 pour les arrêtes bleu/bleu
- 15 de paramètre 0.1 pour les arrêtes rouge/bleu ou bleu/rouge

On obtient ainsi une matrice d'adjacence représentable par son graphe :

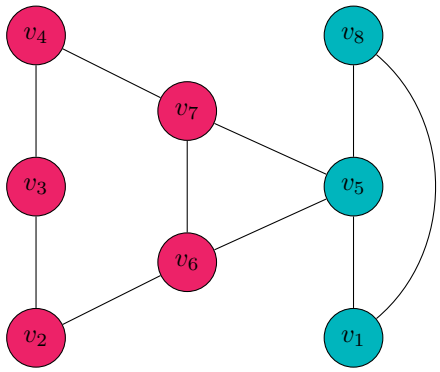


Figure 7 : Un graphe non-orienté possible généré par le SBM

2.3 Cas orienté et non-pondéré

Dans le cas orienté et non-pondéré [7], $\Theta = \{\pi, M\}$ toujours, mais M et A ne sont plus nécessairement symétriques.

Exemple pratique Reprenons l'exemple pratique précédent avec la partition de la figure 6 avec $M = \begin{pmatrix} 0.5 & 0.2 \\ 0.1 & 0.9 \end{pmatrix}$ cette fois-ci.

On procède à $8 \times 7 = 56$ tirages d'une loi binomiale dont :

- 20 de paramètre 0.5 pour les arcs rouge/rouge
- 6 de paramètre 0.9 pour les arcs bleu/bleu
- 15 de paramètre 0.2 pour les arcs rouge/bleu
- 15 de paramètre 0.1 pour les arcs bleu/rouge

On obtient ainsi une matrice d'adjacence représentable par son graphe :

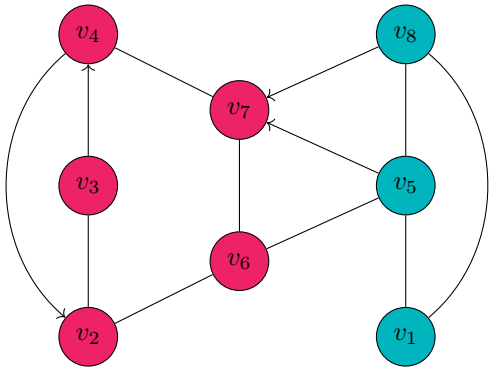


Figure 8 : Un graphe orienté possible généré par le SBM

2.4 Cas pondéré

Dans le cas pondéré [7], on conserve exactement le même modèle puis on rajoute un tirage d'une loi admettant une densité pour déterminer le poids de a_{ij} sachant les groupes g_i et g_j de manière indépendante :

$$\mu_{a_{ij}|(g_i,g_j)=(k,l)}(x) = m_{kl} f_{kl}(x) + (1 - m_{kl})\delta(x).$$

avec $f_{kl} : \mathbb{R}^+ \to \mathbb{R}^+$ qui représente la densité du poids d'un arc menant d'un sommet du bloc k vers un sommet du bloc l. Ainsi $\Theta = \{\pi, M, F\}$ avec $F = (f_{kl})$.

Exemple pratique Reprenons le graphe non-orienté généré à la figure 7, et supposons que pour tout $(k,l) \in [\![1,2]\!]^2$, $f_{kl}(x) = e^{-x}$, *i.e.*. sachant leur existences, les arêtes possèdent des poids qui suivent tous une loi exponentielle de paramètre 1. On tire ainsi indépendamment 9 réalisations d'une loi exponentielle de paramètre 1 et on représente ci-dessous, un graphe possible :

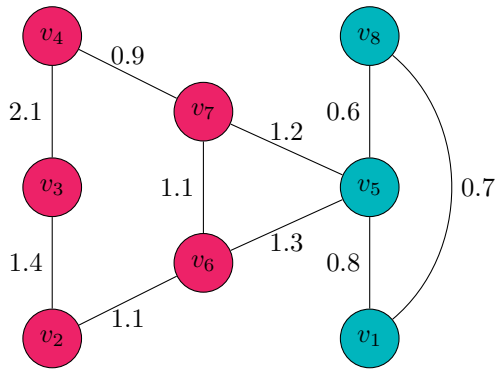


Figure 9 : Un graphe pondéré possible généré par le SBM

Remarque Si pour tout $(k,l) \in [\![1,k]\!]^2$, $f_{kl} = \mathbb{1}_{\mathbb{R}^*}$, alors on retrouve le cas non-pondéré.

2.5 Degree-Corrected Block Model (DCBM)

Les modèles cités précédemment impliquent une homogénéité des degrés au sein d'une même communauté ce qui dans le monde réel est rarement avérée. Ainsi on corrige cette lacune en définissant un nouveau jeu de variables $\theta = (\theta_1, ..., \theta_n)$ tel que $\tilde{M} = (\tilde{m}_{ij}) \in [0,1]^n \times [0,1]^n$ et pour tout $(i,j) \in [\![1,n]\!]$, $\tilde{m}_{ij} = \theta_i \theta_j m_{g_i g_j}$.

Ainsi comme précédemment, on a :

$$\mu_{a_{ij}|(g_i,g_j)=(k,l)}(x) = \tilde{m}_{ij} f_{kl}(x) + (1 - \tilde{m}_{ij})\delta(x).$$

3 Clustering

3.1 Principe général du clustering

Le terme clustering correspond au aux méthodes de partitionnement de données. Elle vise à diviser un ensemble de données en différents sous ensembles selon des critères de proximité. Pour obtenir un bon partitionnement, deux critères sont à prendre en comptes : il faut maximiser la proximité entre les éléments d'un même sous ensemble, et minimiser la proximité entre les différents sous ensembles.

$$\text{Soit } E = \left\{x_i \in \mathbb{R}^d | i \in [\![1, N]\!]\right\}, \text{ un ensemble de N points.}$$

L'objectif est donc de regrouper les points en un certain nombre K de clusters $C_1, C_2, ..., C_K$ tels que :

$$\text{Soit } E = \bigcup_{i=1}^{K} C_i.$$

Dissimilarité La dissimilarité est une fonction $D : E^2 \mapsto \mathbb{R}^+$ telle que pour tout $(x_1, x_2) \in E^2$, $D(x_1, x_2) = D(x_2, x_1)$ et $D(x_1, x_2) = 0$ implique $x_1 = x_2$.

On cite ci-dessous plusieurs fonctions de dissimilarités usuelles.

Distance de Manhattan La distance de Manhattan est définie comme suit :

$$D(x_1, x_2) = \|x_1 - x_2\|_1 = \sum_{i=1}^{d} |x_{1,i} - x_{2,i}|.$$

Distance euclidienne La distance euclidienne est définie comme suit :

$$D(x_1, x_2) = \|x_1 - x_2\|_2 = \sqrt{\sum_{i=1}^{d} (x_{1,i} - x_{2,i})^2}.$$

Métrique liée à une matrice définie positive Pour W une matrice définie positive :

$$D(x_1, x_2) = (x_1 - x_2)^T W (x_1 - x_2).$$

On peut ensuite définir des mesures de dissimilarités entre clusters :

Diamètre minimal On définit le diamètre minimale D_{min} entre deux clusters C_1 et C_2 comme suit :

$$D_{min}(C_1, C_2) = \min_{x_i \in C_1, x_j \in C_2} \{D(x_i, x_j)\}.$$

Diamètre maximal On définit le diamètre maximal D_{max} entre deux clusters C_1 et C_2 comme suit :

$$D_{max}(C_1, C_2) = \max_{x_i \in C_1, x_j \in C_2} \{D(x_i, x_j)\}.$$

Distance moyenne On définit la distance moyenne D_{moy} entre deux clusters C_1 et C_2 comme suit :

$$D_{moy}(C_1, C_2) = \frac{1}{|C_1||C_2|} \sum_{x_i \in C_1, x_j \in C_2} D(x_i, x_j).$$

Centre de gravité On définit le centre de gravité μ_k d'un cluster C_k :

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i.$$

Distance de Ward On définit la distance de Ward D_{Ward} entre deux clusters C_1 et C_2 comme suit :

$$D_{Ward}(C_1, C_2) = \sqrt{\frac{|C_1||C_2|}{|C_1| + |C_2|}} D(\mu_1, \mu_2).$$

3.2 Mesure de la qualité d'un clustering

Inertie intra-cluster L'inertie J_k d'un cluster C_k est définie comme :

$$J_k = \sum_{x_i \in C_k} D(x_i, \mu_k)^2.$$

Cette inertie permet de mesurer la dispersion des point dans le cluster : plus l'inertie est faible, plus la dispersion est faible.
On définit alors l'inertie intra-cluster J_{intra} comme étant la somme des inerties de chaque cluster :

$$J_{intra} = \sum_{k=1}^{K} J_k.$$

Pour obtenir une partition optimale, il faut minimiser l'inertie intra-cluster.

Inertie inter-cluster Les centres de gravité des clusters forment eux aussi un nuage de points, on peut alors mesurer la distance entre chaque clusters :

On nomme μ le centre de gravité du nuage de point initial :

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i.$$

L'inertie inter-cluster J_{inter} est définie par :

$$J_{inter} = \sum_{k=1}^{K} |C_k| D(\mu, \mu_k)^2.$$

Plus l'inertie inter-cluster est grande, plus les clusters sont bien séparés. Le meilleur clustering maximise donc l'inertie inter-cluster.

3.3 Difficulté du clustering

On cherche une partition de D en K sous ensembles $C_1, ..., C_K$ tels que :

$$E = \bigcup_{i=1}^{K} C_i \text{ minimisant l'inertie intra-cluster.}$$

Il s'agit d'un problème NP-difficile puisque le nombre de partition possible évolue de manière factorielle par rapport à N. Il n'est donc pas souhaitable de résoudre le problème de manière exhaustif, en testant chaque possibilité pour retenir la meilleur.

3.4 Algorithme K-moyennes

L'algorithme K-moyennes ou K-means est une approche heuristique du problème qui permet de donner une bonne partition, mais pas nécessairement la meilleure.

Principe

Entrées Ensemble de point E et le nombre de clusters souhaité K.

Sortie Un partitionnement de l'ensemble $E = \bigcup_{i=1}^{K} C_i$.

Initialisation On positionne aléatoirement les centres $\mu_1, ..., \mu_K$ des clusters $C_1, ..., C_K$.

Itération On itère jusqu'a convergence de l'algorithme vers une partition stable :

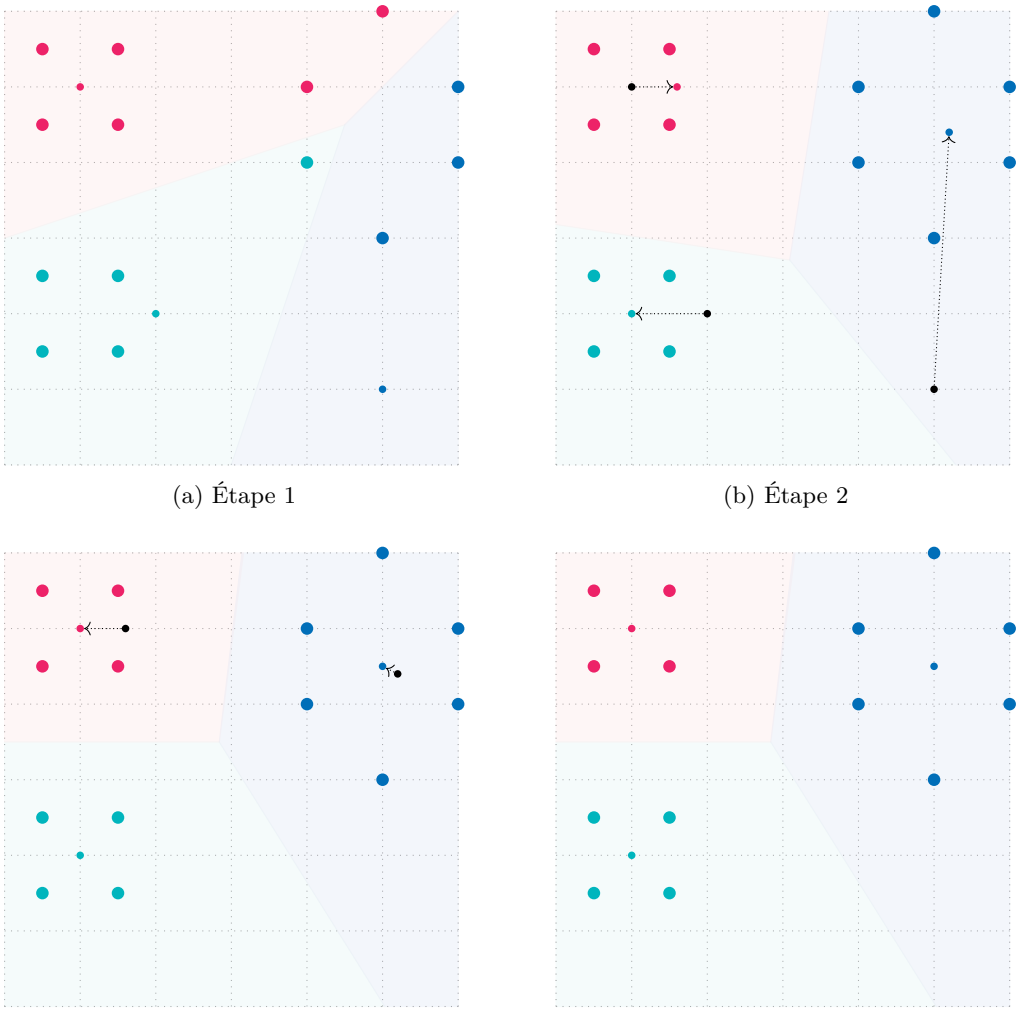
1. On génère une nouvelle partition en assignant chaque x_i au groupe dont il est le plus proche :

$$D(x_i, \mu_k) = \min_{j \in \{1,..K\}} D(x_i, \mu_j) \Rightarrow x_i \in C_k$$

2. On calcule les centres de gravité associés à la nouvelle partition :

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i.$$

Exemple Dans l'exemple ci-dessous les centres sont générés aléatoirement.



(a) Étape 1

(b) Étape 2

(c) Étape 3

(d) Étape 4

Figure 10 : Exemple d'application de l'algorithme K-moyennes

3.5 Algorithme espérance-maximisation

Principe L'algorithme EM repose sur le modèle de mélange fini de lois de probabilité. Ce modèle consiste à supposer que les donnés que l'on possède proviennent d'une source contenant plusieurs sous-populations. La population totale est alors un mélange de ces sous populations. L'objectif de l'algorithme est alors de déterminer les paramètres des différentes lois et à quelle loi appartiennent chacune des donnée en maximisant la vraisemblance, c'est à dire la probabilité que ces données suivent ces lois avec ces paramètres. On regroupe ensuite les points qui suivent la même loi pour former les différents clusters.

Approche probabiliste du clustering On a un ensemble de N points connus, dont on cherche une partition en K clusters.

$$X = (X_1, X_2, ..X_N)$$

En considérant ces N points comme un échantillons de N variables aléatoires indépendantes, issues d'un mélange de K lois différentes, la densité f_X associée à la variable peut s'écrire de manière générale sous la forme :

$$f_X(x) = \sum_{k=1}^{K} p_k f_k(x)$$

avec p_k la proportion du cluster C_k et f_k la densité de la loi des variables appartenants à C_k.

Ce mélange est un modèle à données manquantes puisqu'on ne connait pas le cluster auquel appartient chaque variable.

On définit alors la variable Z :

$$Z = (Z_1, Z_2, ..Z_N)$$

avec Z_i le cluster d'appartenance de X_i et $\mathbb{P}[Z_i = k] = p_k$. Z est inconnue.

On note aussi α_k le paramètre de la loi associée au cluster C_k. Enfin on définit $\Theta = (p_1, ..., p_K, \alpha_1, ..., \alpha_K)$ le paramètre inconnu du modèle de l'échantillon X.

Maximisation de la vraissemblance

L'algorithme EM consiste à maximiser la vraisemblance :

$$L(\Theta, X) = \prod_{i=1}^{N} \sum_{k=1}^{K} p_k f_k(X_i, \alpha_k)$$

Ou de manière équivalente, à maximiser la log-vraisemblance :

$$l(\Theta, X) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} p_k f_k(X_i, \alpha_k) \right)$$

Ce problème de maximisation ne peut être résolu puisqu'on ne connait pas Z, autrement dit, on ne sait pas quelle loi suit X_i. D'ou la nécessité d'utiliser un algorithme itératif.

Procédure On procède alors en deux étapes à chaque itération :

Une étape E « Expectation » qui consiste à estimer la vraisemblance des données complètes, dont la connaissance rendrait possible la maximisation de vraisemblance, en utilisant le paramètre Θ_m qui sera mis à jour à l'étape suivante.

Pour estimer la vraisemblance des données complètes, on utilise l'estimateur :

$$\mathbb{E}_{Z|\Theta_m}[\log(L(X, Z|\Theta))]$$

avec $\mathbb{E}_{Z|\Theta_m}$ l'espérance conditionnelle sur Z sachant X et Θ_m.

Une étape M « Maximisation » ou l'on procède à la maximisation de la vraisemblance que l'on a estimée. Le résultat de cette maximisation met à jour la variable Θ, on obtient donc Θ_{m+1} .

On maximise cette vraisemblance estimée :

$$\Theta_{m+1} = \arg\max_{\Theta} \mathbb{E}_{Z|\Theta_m}[\log(L(X, Z|\Theta))]$$

Approximation gaussienne En classification, on utilise souvent un modèle de mélange fini de loi gaussiennes pour utiliser l'algorithme EM. Autrement dit, pour un nuage de points, on va considérer que les points sont issues de K lois gaussiennes bidimensionnelles d'espérances et de matrices de covariance différentes. Le choix d'un modèle gaussien provient d'une part que les gaussiennes permettent d'approximer une grande variété de densités différentes. D'autre part, le théorème de centrale limite justifie l'utilisation de lois gaussiennes si le nombre de données est important.

Supposons que les K lois sont des lois gaussiennes bidimensionnelles :

$$\begin{cases} Z_i = 1 \Leftrightarrow X_i \sim \mathcal{N}_d(\mu_1, \Sigma_1) \\ Z_i = 2 \Leftrightarrow X_i \sim \mathcal{N}_d(\mu_2, \Sigma_2) \\ \vdots \\ Z_i = K \Leftrightarrow X_i \sim \mathcal{N}_d(\mu_K, \Sigma_K) \end{cases}$$

avec $\mathbb{P}[Z_i = 1] = p_1, \mathbb{P}[Z_i = 2] = p_2, ..., \mathbb{P}[Z_i = K-1] = p_{K-1}$ et $\mathbb{P}[Z_i = K] = 1 - \sum_{k=1}^{K-1} p_k$

On va donc chercher à déterminer $\Theta = (p_1, ..., p_{K-1}, \mu_1, ..., \mu_K, \Sigma_1, ..., \Sigma_k)$

La vraissemblance s'écrit alors :

$$L(X, Z|\Theta) = \prod_{i=1}^{N} \sum_{k=1}^{K} \mathbb{1}_{Z_i=k} p_k f_k(X_i)$$

avec $f_k(x) = \dfrac{1}{(2\pi)^{\frac{d}{2}} |\Sigma_k|^{\frac{1}{2}}} \exp\left\{-\dfrac{1}{2}(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)\right\}$

La log-vraissemblance s'écrit alors :

$$l(X, Z|\Theta) = \sum_{i=1}^{N} \left[\sum_{k=1}^{K} \mathbb{1}_{Z_i=k} (\log(p_k) - \frac{d}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma_k|)) - \frac{1}{2}(X_i-\mu_k)^T \Sigma_k^{-1}(X_i-\mu_k)) \right]$$

À l'itération m et à l'étape E on définit la probabilité que la variable X_i soit issue de la k-ème loi gaussienne sachant $X_i = x_i$ et Θ_m :

$$\tilde{p}_{i,k} = \mathbb{P}[Z_i = k | X = x_i, \Theta_m] = \frac{p_k f_k(x_i)}{\sum_{j=1}^{K} p_j f_j(x_i)}$$

L'estimation de la log-vraisemblance à maximiser s'écrit alors:

$$\mathbb{E}_{Z|\Theta_m}[l(X, Z|\Theta)] = \sum_{i=1}^{N} \sum_{k=1}^{K} \tilde{p}_{i,k} \left[(log(p_k) - \frac{d}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma_k|)) - \frac{1}{2}(X_i-\mu_k)^T \Sigma_k^{-1}(X_i-\mu_k) \right]$$

La maximisation de cette fonction Θ donne :

$$\begin{cases} p_k^{m+1} = \dfrac{1}{N} \sum_{i=1}^{N} \tilde{p}_{i,k} \\ \mu_k^{m+1} = \dfrac{\sum_{i=1}^{N} \tilde{p}_{i,k} x_i}{\sum_{i=1}^{N} \tilde{p}_{i,k}} \\ \Sigma_k^{m+1} = \dfrac{\sum_{i=1}^{N} \tilde{p}_{i,k} (x_i - \mu_k^{m+1})(x_i - \mu_k^{m+1})^T}{\sum_{i=1}^{N} \tilde{p}_{i,k}} \end{cases}$$

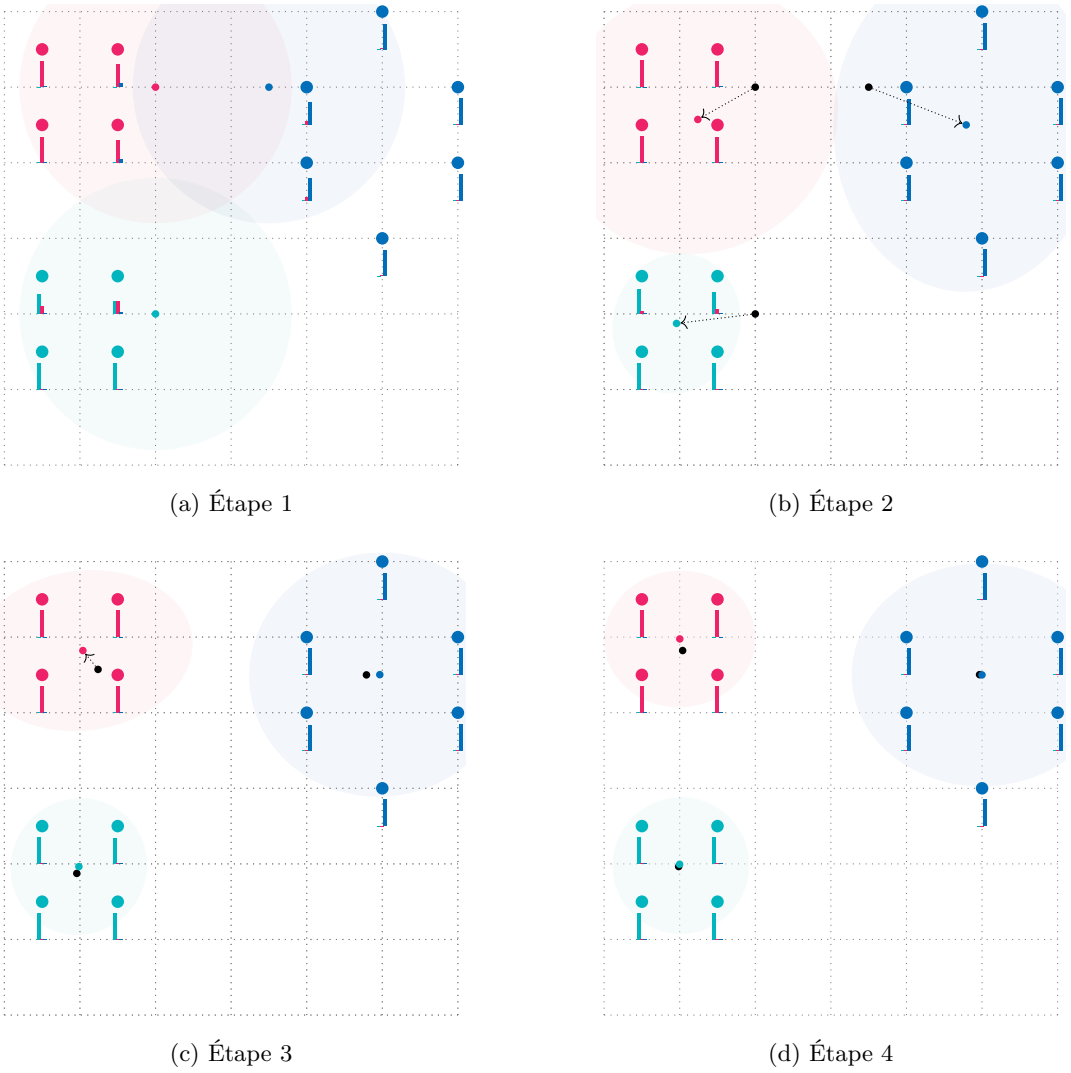
Une fois que l'on a convergence de l'algorithme, les valeurs des $\tilde{p}_{i,k}$ permettent de partitionner les données.

Le point X_i est associé à la population $j \in \{1, 2, ..., K\}$ où j vérifie $\tilde{p}_{i,j} = \max_{k \in \{1,...,K\}} \tilde{p}_{i,k}$.

Autrement dit, j est le numéro de la population deX_i tel que la probabilité que X_i appartienne effectivement à cette population soit maximum.

Exemple

Sur l'exemple ci-dessous, les ellipses représentent une zone de confiance à 80% de présence d'un point distribué selon la loi normale estimée à chaque étape et $\Sigma_0 = I_2$ Les batons situés dessous chaque point représentent les $\tilde{p}_{i,k}$.



(a) Étape 1

(b) Étape 2

(c) Étape 3

(d) Étape 4

Figure 11 : Exemple d'application de l'algorithme EM

4 Méthodes spectrales pour la détection de communautés

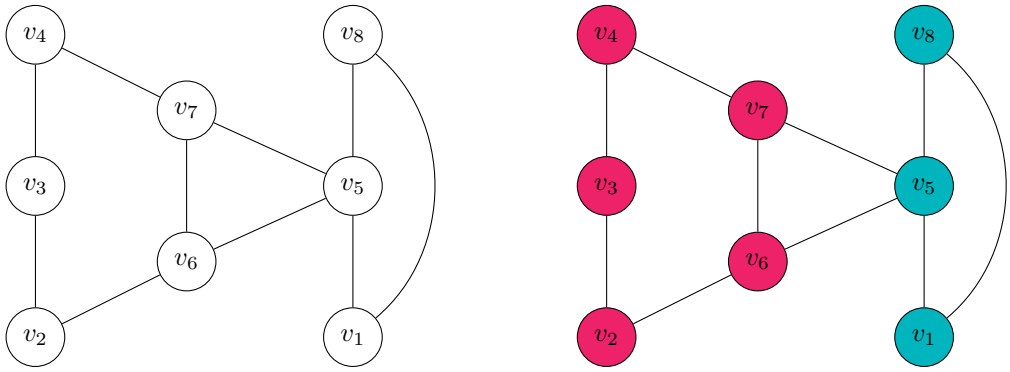
De nombreuses techniques de détection de communautés existent. Dans le cadre de notre étude, nos recherches se portent exclusivement sur une série de méthodes qui ont déjà fait preuve de leur efficacité [3] : les méthodes spectrales. Il s'agit de ramener le problème de détection de communautés à un problème d'optimisation basé sur les valeurs et vecteurs propres de matrices. Traditionnellement, la matrice Laplacienne joue un rôle majeur dans la partition de graphes mais d'autres méthodes telles que celle de la modularité et la Hessienne de Bethe peuvent se montrer plus adaptées à notre problème.

Par souci de simplicité, nombreux des exemples présentés ci-dessous se limiteront au cas des graphes non-orientés qui possèdent une matrice d'adjacence A symétrique, mais les méthodes et principes restent applicables au cas général.

4.1 Notion de communauté

Approche empirique Les graphes sont des approches mathématiques capables de représenter efficacement des systèmes complexes, qu'ils soient physiques, biologiques, sociaux, etc. On parle de communautés au sein de ces systèmes lorsque les éléments qui les forment entretiennent des liens privilégiés, parce qu'ils ont des affinités particulières, ou présentent des caractéristiques similaires. Ce qui conduit à la formations de groupes, qui entretiennent de fortes interactions en leur sein mais peu avec l'extérieur. Les réseaux sociaux, les structures chimiques, les systèmes d'information et plus généralement l'internet présentent tous une très forte tendance à la formation de structures communautaires.

Au sens du graphe, une communauté est constituée par un ensemble de noeuds qui sont fortement liés entre eux, et faiblement liés avec les noeuds situés en dehors de la communauté.



(a) Graphe initial (b) Identification des communautés

Figure 12 : La détection de communauté à partir d'un graphe : inférence du SBM

Formalisme en théorie des graphes Il n'existe pas de définition universelle et précise d'une communauté [4]. Néanmoins, l'approche empirique nous incite à définir le *degré* des relations qui

existent au sein d'un graphe.

Soit C un sous-graphe de G, avec $|C| = n_c$ et $|G| = n$. On définit le degré externe d_v^{ext} de chaque sommet v comme le degré de v au sein du graphe $G \setminus C$. Le degré interne d_v^{int} est le degré de v au sein du graphe C.

- d_v^{ext} est le nombre d'arêtes reliant v aux autres sommets du sous-graphe C.
- d_v^{int} est le nombre d'arêtes reliant v aux sommets du restant du graphe C.

On a donc $d_v = d_v^{ext} + d_v^{int}$.

De même, on peut définir la densité *intra-communautaire* $\rho_{int}(C)$ comme le rapport entre le nombre d'arêtes de C et le nombre d'arêtes potentielles.

$$\rho_{int}(C) = \frac{|\text{arêtes internes de C}|}{n_c(n_c - 1)/2}$$

La densité *inter-communautaire* sera alors le rapport entre le nombre d'arêtes reliant C et le reste du graphe sur le nombre total des arêtes inter-communautaires potentielles.

$$\rho_{ext}(C) = \frac{|\text{arêtes inter-communautaires de C}|}{n_c(n - n_c)}$$

Ainsi, pour que C soit une communauté, on s'attend à ce que $\rho_{int}(C)$ soit très supérieur à ρ, la densité du graphe. À l'inverse, $\rho_{ext}(C)$ doit être très inférieur à ρ. La recherche d'un compromis entre un $\rho_{int}(C)$ élevé et un $\rho_{ext}(C)$ est à la base, implicitement ou explicitement, de la plupart des méthodes et algorithmes de détection de communautés.

Partitions disjointes ou chevauchantes Tout comme dans une personne peut faire partie de plusieurs communautés au sein d'un même réseau, un noeud d'un graphe peut appartenir à plusieurs communautés. Dans l'exemple de la figure 12 , le graphe possède deux communautés et v_5 est l'interface entre les deux communautés. Il peut faire partie des deux communautés [5].

- On appelle cover une division en communautés chevauchantes
- On parle de hard clustering lorsque le graphe est partagé en sous-groupes disjoints

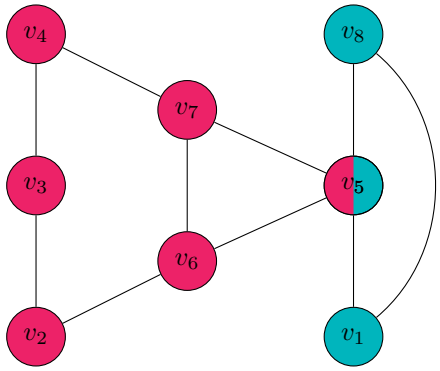


Figure 13 : v_5 fait partie des deux communautés

4.2 Matrice de Modularité

Dans cette partie, nous nous intéressons à une méthode très répandue dans la littérature, introduite par Newman et Girvan [11] et largement revisitée : l'optimisation de la modularité. La méthode proposée pour la détection de communautés se base sur la mesure de la modularité Q qui s'avère être une *bonne* mesure de la qualité d'une partition en communautés.

Modularité d'un cluster On va chercher le découpage pour lequel les liens intra-communautaires sont plus fortes que ce à quoi on pourrait s'attendre. Autrement dit, une communauté est un ensemble de noeuds dont les arêtes vers les autres noeuds sont bien plus rares que la moyenne.

De façon équivalente, on peut aussi décider de chercher les groupes pour lesquels les arêtes internes sont plus nombreuses que la moyenne (à définir) pour un tel groupe. On choisit cette définition.

$$Q_k = |\text{ arêtes internes à } C_k| - |\text{ attendu d'arêtes internes à } C_k|$$

Si le nombre d'arêtes internes est un calcul direct, il reste à définir le nombre d'arêtes attendu. En considérant des distributions aléatoires de réseaux, on note P_{ij} la probabilité pour qu'une arête relie les noeuds i et j telle que :

$$Q_k = \sum_{\substack{i \in C_k \\ j \in C_k}} [A_{ij} - P_{ij}]$$

Modularité d'un réseau On définit la modularité d'un réseau comme la somme des modularités de chaque cluster :

$$\begin{aligned} Q &= \sum_{k=1}^{K} Q_k \\ &= \sum_{k=1}^{K} \sum_{\substack{i \in C_k \\ j \in C_k}} [A_{ij} - P_{ij}] \end{aligned}$$

Choix de P_{ij} Le choix de P_{ij} revient à faire le choix de la distribution de référence pour mesurer la qualité de nos partitions. On peut considérer une distribution de Bernoulli pour la génération de graphes comme vu en première partie. Néanmoins, la recherche actuelle semble s'accorder sur le fait que celle-ci n'est pas la meilleure représentation pour les réseaux réels [10]. Une solution préférable est une distribution aléatoire où la probabilité que l'extrémité d'une arête s'attache au noeud i ne dépende que du degré d_i. Ainsi dans notre modèle, la probabilité P_{ij} *i.e.* le nombre d'arêtes moyen entre le noeud i et le noeud j est un produit de fonctions dépendant des degrés respectifs des noeuds :

$$P_{ij} \propto f(d_i) f(d_j)$$

On prend en compte les contraintes suivantes :

1. $Q = 0$ dans le cas d'une partition unique, donc $\sum_{i,j} P_{ij} = \sum_{i,j} A_{ij} = 2m$
2. Le degré de chaque noeud s'exprime en fonction de P_{ij} ainsi $\sum_{j=1}^{n} P_{ij} = d_i$

On obtient :

$$\sum_{j=1}^{n} P_{ij} = f(d_i) \sum_{j=1}^{n} f(d_j) = d_i$$

Donc $f(d_i) \propto d_i$ et on note $f(d_i) = C d_i$. On obtient d'après la condition 1 :

$$2m = \sum_{ij} P_{ij} = C^2 \sum ij d_i d_j = (2mC)^2$$

D'où $C = 1/\sqrt{2m}$ et

$$P_{ij} = \frac{d_i d_j}{2m}$$

Matrice de modularité Notons ainsi $B = (B_{ij})$ avec $B_{ij} = A_{ij} - P_{ij} = A_{ij} - \frac{d_i d_j}{2m}$ telle que :

$$\begin{aligned} Q &= \sum_{k=1}^{K} \sum_{\substack{i \in C_k \\ j \in C_k}} B_{ij} \\ &= \sum_{k=1}^{K} e_k^T B e_k \\ &= \mathrm{Tr}[E^T B E] \end{aligned}$$

Enfin notons d le vecteur colonne composé des degrés : $d = (d_1 d_2 \dots d_n)^T$ alors :

$$B = A - \frac{1}{2m} d d^T$$

Notons que B est une matrice symétrique.

Relaxation spectrale On choisit de lever la contrainte sur E supposée booléenne tout en conservant l'hypothèse d'orthonormalité $E^T E = I$. Le problème d'optimisation s'écrit alors :

$$\max_{\substack{Y \in \mathbb{R}^{n \times K} \\ Y^T Y = I}} \mathrm{Tr}\left[Y^T B Y\right]$$

Théorème de Ky-Fan et clusterisation Le théorème de Ky-Fan assure que la solution de ce problème de maximisation est la matrice U qui a pour colonnes les k vecteurs propres normaux associés aux plus grandes valeurs propres de B et en notant $\lambda_1 \geq \lambda_2, ..., \geq \lambda_n$ les valeurs propres de B :

$$\max_{\substack{Y \in \mathbb{R}^{n \times K} \\ Y^T Y = I}} \mathrm{Tr}\left[Y^T B Y\right] = \sum_{i=1}^{K} \lambda_i$$

On obtient ainsi un estimateur de E sur lequel on peut effectuer un clustering de type K-means ou EM sur ses lignes afin de détecter correctement les différentes communautés.

Choix du nombre de communautés K Notons p le nombre de valeurs propres positives de B alors $\lambda_{p+1} = 0$ car :

$$\begin{aligned} B \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} &= A \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} - \frac{1}{2m} d d^T \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\ &= d - \frac{1}{2m} d \times 2m \\ &= 0 \end{aligned}$$

Ainsi :

$$\max_{\substack{Y \in \mathbb{R}^{n \times K} \\ Y^T Y = I}} \mathrm{Tr}\left[Y^T B Y\right] = \sum_{i=1}^{k} \lambda_i \leq \sum_{i=1}^{p+1} \lambda_i$$

Pour maximiser la modularité il semblerait judicieux de sélectionner pour K les p vecteurs propres associés aux valeurs propres positives de B. Néanmoins en pratique, les restrictions sur la forme de la matrice booléenne Y nous empêche de conclure que le maximum est atteint pour exactement cette valeur de K.

Le nombre optimal de communautés est $K \leq p$.

En pratique, cette méthode nous invite à soit itérer sur les valeurs de K possibles de 1 à p, soit considérer, comme nous le verrons pour la matrice Laplacienne, l'écart entre les valeurs propres.

En effet pour un graphe d-régulier on a $B = A - \dfrac{d^2}{2m} \begin{pmatrix} 1 & \dots & 1 \\ \vdots & & \vdots \\ 1 & \dots & 1 \end{pmatrix}$

Exemple Supposons que l'on dispose d'un graphe $G = (V, E)$ avec $|V| = 10$ on calcule la matrice de modularité de ce graphe et ses valeurs propres notées $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_{10}$.

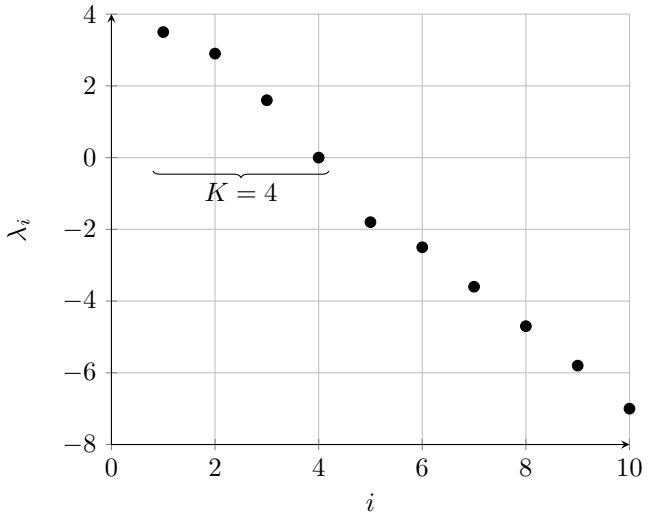


Figure 14 : Exemple de détermination du nombre de communautés K

Ainsi dans l'exemple de la figure on réalisera le clustering spectral sur $K \leq 4$ communautés.

Vers l'implémentation Nous avons ainsi obtenu un premier algorithme permettant la détection de communautés.

Données : Matrice d'adjacence A
Résultat : Matrice de partitionnement E
Calculer le vecteur des degrés d
Calculer la matrice de modularité $B = A - \frac{1}{2m} dd^t$
Calculer et ordonner les valeurs propres de B : $\lambda_1 \geq ... \geq \lambda_n$
Trouver p le nombre de valeurs propres positives ou nulles de B
Former la matrice U à partir des p premiers vecteurs propres de B
pour K allant de 2 à p **faire**
    Extraire $\tilde{U}$ en prenant les $K - 1$ premières colonnes de U
    Réaliser un clustering de type K-means ou EM sur les lignes de U
fin
Choisir le clustering qui maximise Q
Affecter arbitrairement le résultat de ce clustering à E

Algorithme 1 : Détection de communautés par la matrice de modularité B

4.3 Matrice Laplacienne

On dispose d'un graphe pondéré $G = (V, E)$ défini par sa matrice d'adjacence A et tel que $|V| = n$.

Le cut size On cherche K clusters disjoints $C = (C_1, ..., C_K)$ tels que $\bigcup_{k=1}^{K} C_k = V$ qui optimise une certaine fonction de coût. Une telle fonction est le cut-size :

$$R(C) = \sum_{k=1}^{K} \sum_{\substack{i \in C_r \\ j \in \bar{C}_r}} A_{ij}$$

On définit également le cut-size normalisé R_n :

$$R_n(C) = \sum_{k=1}^{K} \left(\sum_{\substack{i \in C_r \\ j \in \bar{C}_r}} A_{ij} / \sum_{\substack{i \in C_r \\ j \in V}} A_{ij} \right)$$

Le vecteur d'indexe On définit le vecteur d'indexe $e_k \in \{0, 1\}^n$ du k-ième cluster tel que e_k a des termes non-nuls exactement au points appartenants au k-ième cluster. On pose $E = (e_{ik})$ la matrice de taille $n \times K$ qui possède pour colonnes les vecteurs $e_1^T, ..., e_K^T$.

On note que $\sum_{\substack{i \in C_k \\ j \in V}} A_{ij} = \sum_{i \in C_k} \sum_{j=1}^{n} A_{ij} = \sum_{i=1}^{n} e_{ik}^2 d_i = e_k^T D e_k$.

De plus $\sum_{\substack{i \in C_k \\ j \in \bar{C}_k}} A_{ij} = \sum_{\substack{i \in C_k \\ j \in V}} A_{ij} - \sum_{\substack{i \in C_k \\ j \in C_k}} A_{ij} = e_k^T D e_k - e_k^T A e_k = e_k^T (D - A) e_k$

Et donc on obtient une nouvelle expression pour le cut-size :

$$\begin{aligned} R(E) &= \sum_{k=1}^{K} e_k^T (D - A) e_k \\ &= \mathrm{Tr}\left[E^T (D - A) E\right] \end{aligned}$$

Et celle du cut-size normalisé :

$$\begin{aligned} R_n(E) &= \sum_{k=1}^{K} \frac{e_k^T (D-A) e_k}{e_k^T D e_k} \\ &= \mathrm{Tr}\left[E^T (D-A) E (E^T D E)^{-1}\right] \\ &= \mathrm{Tr}\left[E^T (D-A) E E^T D^{-1} E\right] \\ &= \mathrm{Tr}\left[E^T (D-A) D^{-1} E\right] \end{aligned}$$

Ces deux fonctions de coût doivent être minimisées pour obtenir le partitionnement optimal.

Relaxation spectrale On choisit de lever la contrainte sur E supposée booléenne tout en conservant l'hypothèse d'orthonormalité $E^T E = I$. On définit la matrice de Laplace $L = D - A$ telle que le problème d'optimisation s'écrive :

$$\min_{\substack{Y \in \mathbb{R}^{n \times K} \\ Y^T Y = I}} \mathrm{Tr}\left[Y^T L Y\right]$$

Dans le cas normalisé on pose $Y = D^{1/2} E$ tel que :

$$\begin{aligned} R_n &= \mathrm{Tr}\left[E^T (D-A) D^{-1} E\right] \\ &= \mathrm{Tr}\left[Y^T D^{-1/2} (D-A) D^{-1} D^{1/2} Y\right] \\ &= \mathrm{Tr}\left[Y^T D^{-1/2} (D-A) D^{-1/2} Y\right] \end{aligned}$$

Enfin on définit la matrice de Laplace normalisée $L_n = D^{-1/2}(D-A)D^{-1/2} = D^{-1/2} L D^{-1/2}$ telle que la minimisation de R_n soit équivalente à :

$$\min_{\substack{Y \in \mathbb{R}^{n \times K} \\ Y^T Y = I}} \mathrm{Tr}\left[Y^T L_n D Y\right]$$

Théorème de Ky-Fan et clusterisation Le théorème de Ky-Fan assure que la solution de ce problème de minimisation est la matrice U qui a pour colonnes les K premiers vecteurs propres normaux de L ou de L_n.

On obtient ainsi un estimateur de E sur lequel on peut effectuer un clustering de type K-means ou EM sur ses lignes afin de détecter correctement les différentes communautés.

Choix du nombre de communautés K Supposons l'existence de K communautés au sein d'un graphe non-pondéré muni de sa matrice d'adjacence L. Asymptotiquement, après ordonnancement des noeuds, L peut se mettre sous la forme d'une matrice diagonale par blocs :

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_K \end{pmatrix}.$$

Or des matrices par blocs possèdent des vecteurs propres en blocs également. Ainsi les vecteurs de type : $\begin{pmatrix} v_1 \\ \\ \end{pmatrix}, \begin{pmatrix} v_2 \\ \\ \end{pmatrix}, ..., \begin{pmatrix} \\ \\ v_K \end{pmatrix}$ avec v_i un vecteur propre de L_i sont des vecteurs propres de L.

Or remarquons que les vecteurs $v_i = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ sont des vecteurs propres de L_i associés à la valeur propre 0. En effet :

$$\begin{aligned} L_i \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} &= (D_i - A_i) \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\ &= 0 \end{aligned}$$

Dans la pratique L n'est pas diagonale par blocs mais s'approche de cette forme et les vecteurs propres de L conservent partiellement leurs propriétés également.

Enfin définissons une nouvelle matrice dite matrice d'incidence $M \in \mathbb{R}^n \times \mathbb{R}^m$ telle que :

$$m_{ij} = \begin{cases} 1 & \text{si le sommet } v_i \text{ est une extrémité de l'arrête } x_j \\ 0 & \text{sinon} \end{cases}$$

On remarque que $L = MM^T$ et ainsi pour tout $x \in \mathbb{R}^n$:

$$\begin{aligned} x^T L x &= x^T M M^T x \\ &= (M^T x)^T (M^T x) \\ &= ||M^T x||_2^2 \\ &\geq 0 \end{aligned}$$

D'où L est semi-définie positive et symétrique, le théorème spectral assure alors que :

$$\mathrm{Sp}(L) \subset \mathbb{R}^+$$

Ainsi nous savons qu'asymptotiquement L possède la valeur propre 0 avec une multiplicité K et que 0 est la valeur propre de plus faible magnitude de L.

Notons $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$ les n valeurs propres de L. On choisira K de façon à maximiser le saut entre deux valeurs propres consécutives : $\Delta_k = \lambda_{k+1} - \lambda_k$, *i.e.* :

$$K = \arg \max_{k \in [\![0, n-1]\!]} \Delta_k$$

On peut appliquer le même raisonnement à $L_n = D^{-1/2} L D^{-1/2}$ qui est elle aussi semi-définie positive car pour tout $x \in \mathbb{R}^n$:

$$\begin{aligned} x^T L_n x &= x^T D^{-1/2} L D^{-1/2} x \\ &= ((D^{-1/2})^T x)^T L ((D^{-1/2})^T x) \\ &\geq 0 \end{aligned}$$

De plus les vecteurs propres $D_i^{1/2} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$ sont des vecteurs propres de L_{n_i} associés à la valeur propre 0 car :

$$\begin{aligned} L_{n_i} D_i^{1/2} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} &= D_i^{-1/2} L_i D_i - 1/2 D_i^{1/2} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\ &= D_i^{-1/2} L_i \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \\ &= 0 \end{aligned}$$

On peut ainsi détecter le nombre de communauté de la même manière que pour la matrice Laplacienne non-normalisée.

Exemple Supposons que l'on dispose d'un graphe $G = (V, E)$ avec $|V| = 10$ on calcule le Laplacien de ce graphe et ses valeurs propres notées $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_{10}$. On trace à la figure suivante la magnitude des valeurs propres de L

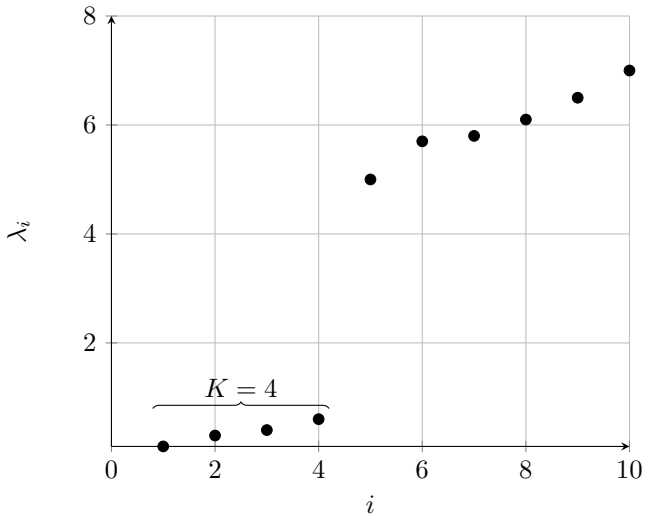


Figure 15 : Exemple de détermination du nombre de communautés K

Ainsi dans l'exemple de la figure on réalisera le clustering spectral sur $K = 4$ communautés.

Vers l'implémentation Nous avons ainsi obtenu deux algorithmes permettant la détection de communautés.

Le premier concerne la matrice Laplacienne L :

Données : Matrice d'adjacence A
Résultat : Matrice de partitionnement E
Calculer la matrice des degrés D
Calculer la matrice Laplacienne $L = D - A$
Calculer et ordonner les valeurs propres de L : $\lambda_1 \leq ... \leq \lambda_n$
Trouver $K = \arg\max(\lambda_{k+1} - \lambda_k)$
Former la matrice U à partir des K premiers vecteurs propres de L
Réaliser un clustering de type K-means ou EM sur les lignes de U
Affecter arbitrairement le résultat de ce clustering à E

Algorithme 2 : Détection de communauté par la matrice de Laplacienne L

Le deuxième concerne la matrice Laplacienne normalisée L_n :

Données : Matrice d'adjacence A
Résultat : Matrice de partitionnement E
Calculer la matrice des degrés D
Calculer la matrice Laplacienne normalisée $L_n = D^{-1/2} L D^{-1/2}$
Calculer et ordonner les valeurs propres de L_n : $\lambda_1 \leq ... \leq \lambda_n$
Trouver $K = \arg\max(\lambda_{k+1} - \lambda_k)$
Former la matrice U à partir des K premiers vecteurs propres de L
Réaliser un clustering de type K-means ou EM sur les lignes de U
Affecter arbitrairement le résultat de ce clustering à E

Algorithme 3 : Détection de communauté par la matrice de Laplacienne normalisée L_n

4.4 Matrice d'Adjacence

Hypothèse de régularité Supposons davantage que le graphe est d-régulier c'est à dire que chacun de ses sommets possède d voisins ou $D = dI$. Cette hypothèse est particulièrement vérifiée dans le cas de graphe dense.

Supposons que l'on dispose d'un graphe G d-régulier et notons alors $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$ les valeurs propres de sa matrice Laplacienne $L = D - A = dI - A$. On note $\mu_1 \geq \mu_2 \geq ... \geq \mu_n$ les valeurs propres de A telles que pour tout i, $\mu_i = d - \lambda_i$. On peut ainsi appliquer le même raisonnement que précédemment en calculant directement les valeurs propres et vecteurs propres de A.

Exemple Supposons que l'on dispose d'un graphe $G = (V, E)$ avec $|V| = 10$ on calcule la matrice d'adjacence de ce graphe et ses valeurs propres notées $\mu_1 \geq \mu_2 \geq ... \geq \mu_{10}$. On trace à la figure suivante la magnitude des valeurs propres de L

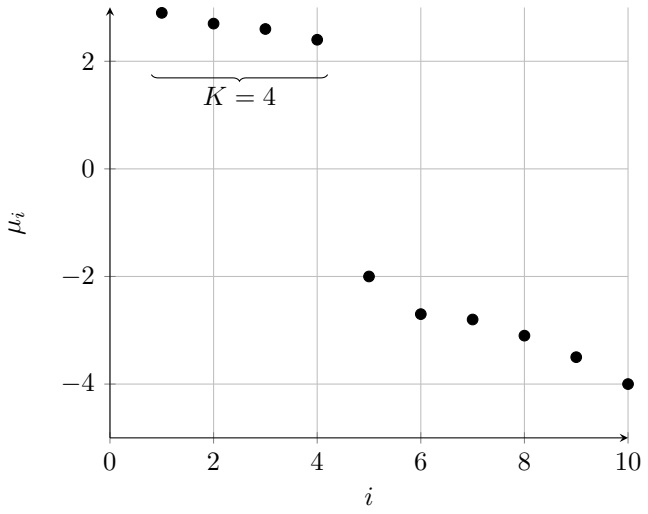


Figure 16 : Exemple de détermination du nombre de communautés K

Ainsi dans l'exemple de la figure on réalisera le clustering spectral sur $K = 4$ communautés.

Vers l'implémentation Nous avons ainsi un nouvel algorithme permettant la détection de communautés.

Données : Matrice d'adjacence A
Résultat : Matrice de partitionnement E
Calculer et ordonner les valeurs propres de A : $\mu_1 \geq \mu_2 \geq ... \geq \mu_n$
Trouver $K = \arg\max |\mu_{k+1} - \mu_k|$
Former la matrice U à partir des K premiers vecteurs propres de A
Réaliser un clustering de type K-means ou EM sur les lignes de U
Affecter arbitrairement le résultat de ce clustering à E

Algorithme 4 : Détection de communauté par la matrice d'adjacence A

4.5 Matrice Hessienne de Bethe

Nous présentons maintenant une autre méthode spectrale pour la détection de communautés, basée sur la matrice dite Hessienne de Bethe. Cette méthode se révèle particulièrement efficace dans le cas de graphes sparses, et dans le cas de graphes générés par le SBM [14].

Expression de la matrice On définit la matrice Hessienne de Bethe de la manière suivante :

$$H(r) = (r^2 - 1)I - rA + D$$

avec A la matrice d'adjacence, D la matrice diagonale des degrés, et $|r|>1$ un terme de régularisation qui doit être optimisé.

Des travaux récents [] suggèrent de choisir $r_c = \frac{< d^2 >}{< d >} - 1$ pour un graphe réel et dans le cas particulier du SBM $r_c =< d >$ où $< \dot{>}$ représente l'opérateur moyenne

Choix du nombre de communauté Le critère de choix du nombre de communauté est alors simplifié : on choisit les K valeurs propres négatives de $H(r_c)$ et on réalise ensuite un clustering à partir des K vecteurs propres associés.

Exemple Supposons que l'on dispose d'un graphe $G = (V, E)$ avec $|V| = 10$. On trace à la figure suivante la magnitude des valeurs propres de $H(r_c)$, $\lambda_1 \leq \lambda_2 \leq ...\lambda_{10}$.

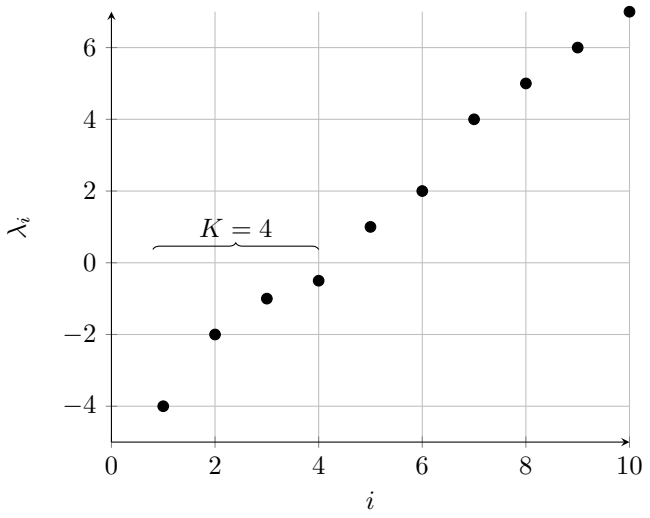


Figure 17 : Exemple de détermination du nombre de communautés K

Ainsi dans l'exemple de la figure on réalisera le clustering spectral sur $K = 4$ communautés.

Vers l'implémentation Nous avons ainsi obtenu un dernier algorithme permettant la détection de communautés.

Données : Matrice d'adjacence A
Résultat : Matrice de partitionnement E
Calculer la matrice des degrés D
Calculer le degré moyen d_m
Calculer le critère régularisateur optimal $r_c = \sqrt{d_m}$
Calculer la matrice Hessienne de Bethe $H(r_c) = (r_c^2 - 1)\mathbb{1} - r_c A + D$
Calculer et ordonner les valeurs propres de $H(r_c)$: $\lambda_1 \leq ... \leq \lambda_n$
Trouver K le nombre de valeurs propres négatives de $H(r_c)$
Former la matrice U à partir des K premiers vecteurs propres de $H(r_c)$
Réaliser un clustering de type K-means ou EM sur les lignes de U
Affecter arbitrairement le résultat de ce clustering à E

Algorithme 5 : Détection de communauté par la matrice Hessienne de Bethe $H(r_c)$

II Implémentation des méthodes de détection de communautés

Notations

N : nombre de noeuds du réseau

K : nombre de communautés

p_{in} : probabilité qu'un noeud soit liée à un des membres de sa communauté

p_{out} : probabilité qu'un noeud soit liée à un des membres extérieur à sa communauté

$c_{in} = N \times p_{in}$: nombre moyen de liaisons d'un noeud effectués au sein d'une communauté

$c_{out} = N \times p_{out}$: nombre moyen de liaisons d'un noeud effectués à l'extérieur d'une communauté

c : nombre moyen total de liaisons d'un noeud

On a la relation :

$$\begin{aligned} c &= \left(\frac{N}{K} - 1\right) p_{in} + \left(N - \frac{N}{K}\right) p_{out} \\ &= \left(\frac{N}{K} - 1\right) \frac{c_{in}}{N} + \left(N - \frac{N}{K}\right) \frac{c_{out}}{N} \end{aligned}$$

D'où à c et c_{out} fixés on fixe aussi c_{in} :

$$c_{in} = \frac{NKc - N(K-1)c_{out}}{N - K}$$

Condition théorique de détection : $c_{in} - c_{out} \geq K\sqrt{c}$

N_v : nombre de vecteur choisis pour le clustering

g_i : groupe d'appartenance du noeud v_i

$\hat{g}_i$: groupe d'appartenance du noeud v_i estimé

$\eta = \left(\frac{1}{N}\sum_i \mathbb{1}_{g_i = \hat{g}_i} - 1/K\right) \Big/ (1 - 1/K)$

$\rho = \frac{1}{N_s}\sum_s \mathbb{1}_{K_s = \hat{K}_s}$ avec N_s le nombre de simulation $\hat{K}$ le nombre de communauté estimé.

1 Complexité calculatoire

1.1 Algorithmes de clustering

On compare tout d'abord l'efficacité des deux algorithmes de clustering que nous avons présentés précédemment : EM et K-moyennes.
η correspond au taux d'estimation correcte de l'appartenance d'un noeud à sa communauté.
On compare également l'efficacité temporelle des deux solutions.

Matrice d'Adjacence

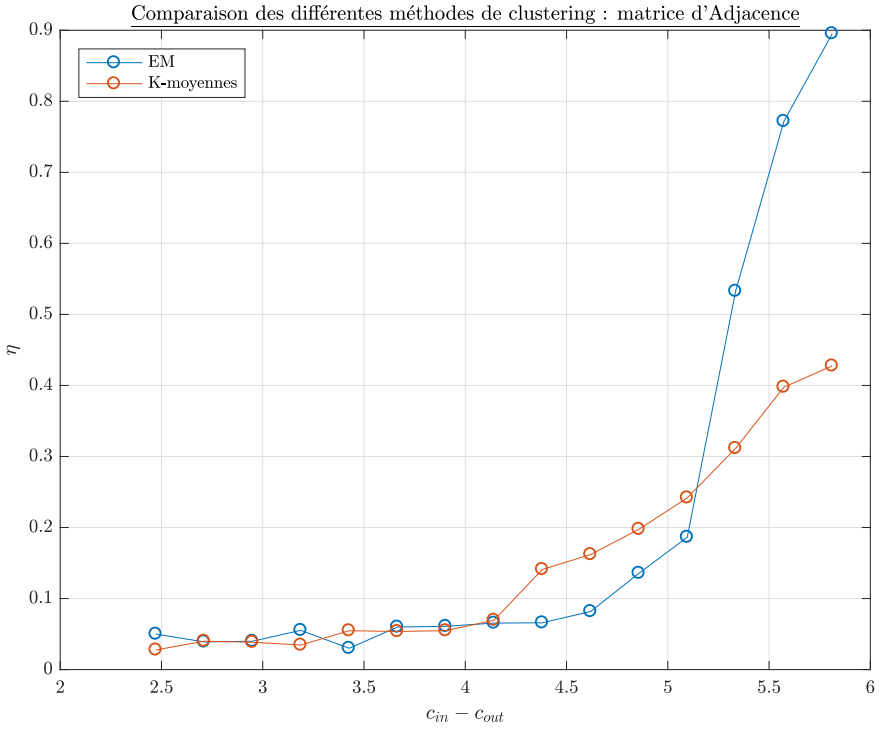


Figure 18 : $c = 3$ $K = 2$ $N = 1000$

Dans le cadre de la matrice d'adjacence, ces résultats montrent que les algorithmes EM et K-moyennes ne sont pas efficaces pour estimer l'appartenance d'un noeud à une communauté lorsque le graphe vérifie $c_{in} - c_{out}$ inférieur à 5. L'algorithme EM devient efficace lorsque $c_{in} - c_{out}$ est supérieur à 5.5, son taux d'estimation correcte devient alors supérieur à 80%. Cela correspond à la condition théorique de détection $c_{in} - c_{out} \geq K\sqrt{c}$.

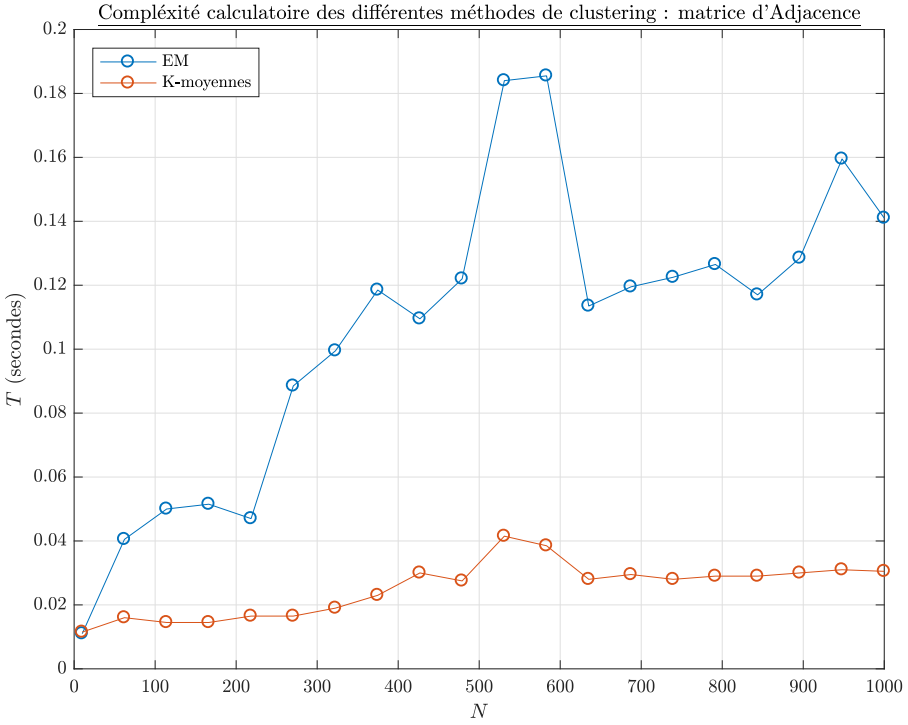


Figure 19 : $c = 10$ $c_{out} = 1$ $K = 2$

L'algorithme EM, plus performant que l'algorithme K-moyennes, nécéssite néanmoins beaucup plus de temps de calcul, en particulier lorsque le nombre de noeuds du graphe dépasse 200, il est environ 4 fois plus couteux en terme de calculs nécéssaires.

Matrice Laplacienne

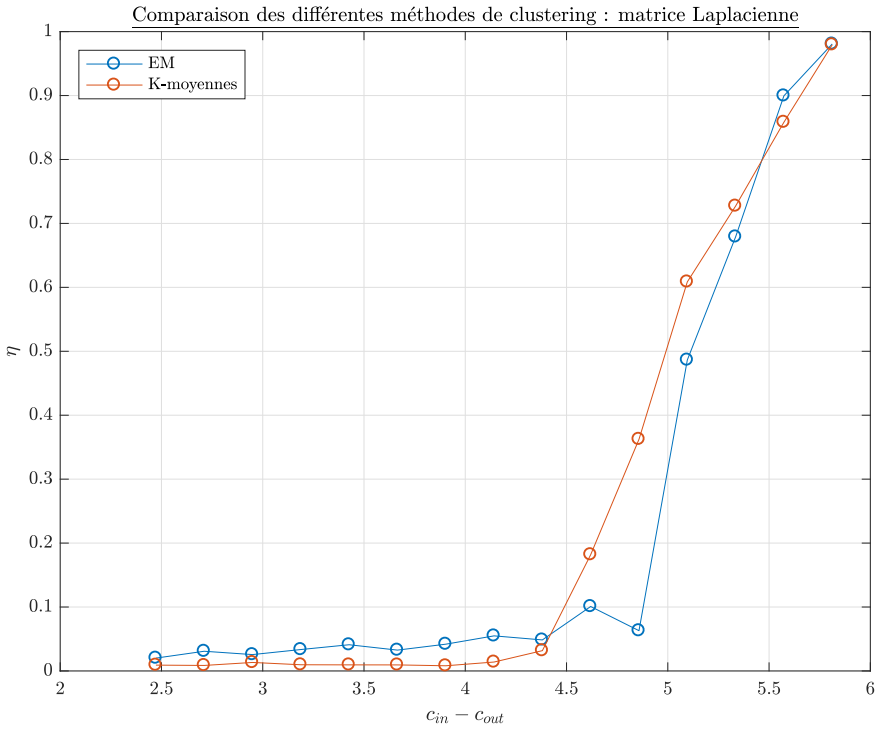


Figure 20 : $c = 3$ $K = 2$ $N = 1000$

Les résultats sur l'efficacité des algorithmes EM et K-moyennes en utilisant la matrice Laplacienne indique que les deux algorithmes ont des performances similaires. On remarque à nouveau qu'ils sont efficaces à partir de $c_{in} - c_{out} \geq 5.5$ et que la condition théorique de détection est vérifiée. De plus l'algorithme K-moyennes a un bien meilleur taux d'estimation correcte avec la matrice Laplacienne qu'avec la matrice d'adjacence.

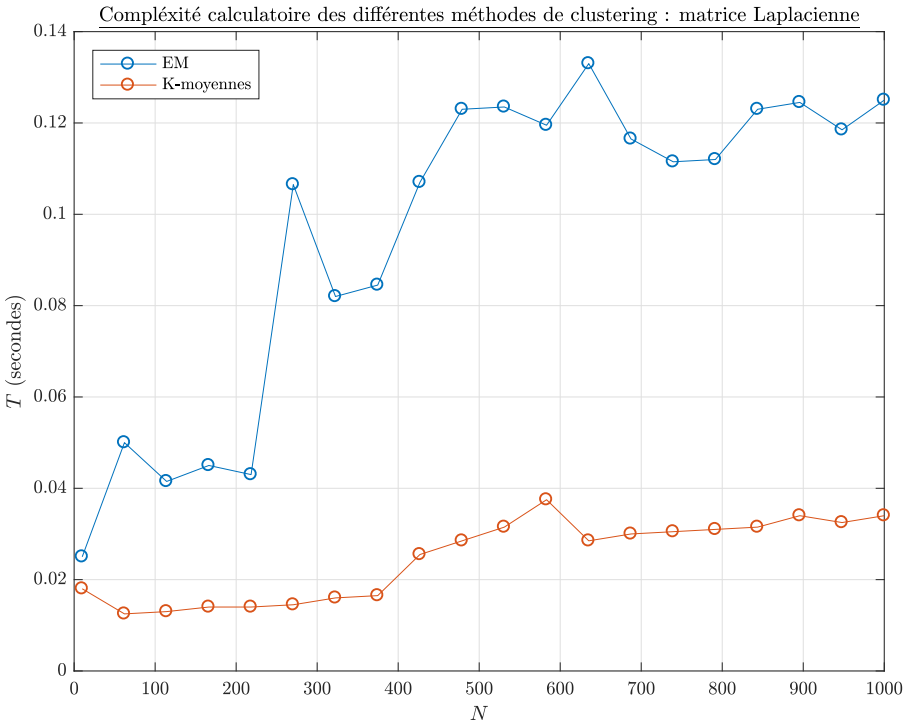


Figure 21 : $c = 10$ $c_{out} = 1$ $K = 2$

Les résultats sur la complexité calculatoire sont identiques à ceux obtenus sur la matrice d'adjacence.

Matrice Laplacienne normalisée

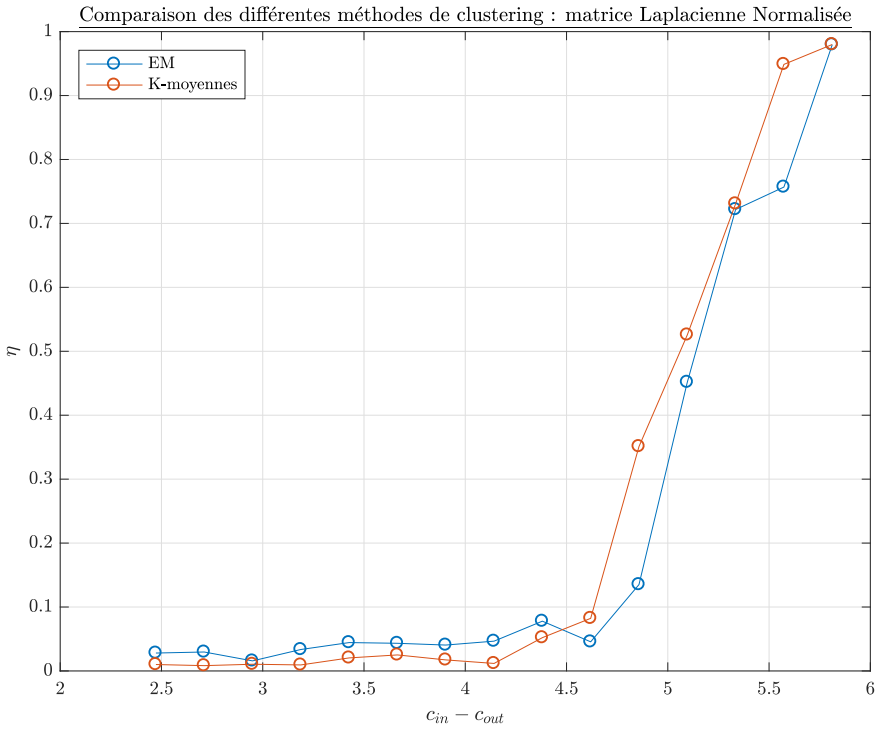


Figure 22 : $c = 3$ $K = 2$ $N = 1000$

La normalisation de la matrice Laplacienne ne modifie pas les performances des algorithmes K-moyennes et EM par rapport à la matrice Laplacienne non normalisée.

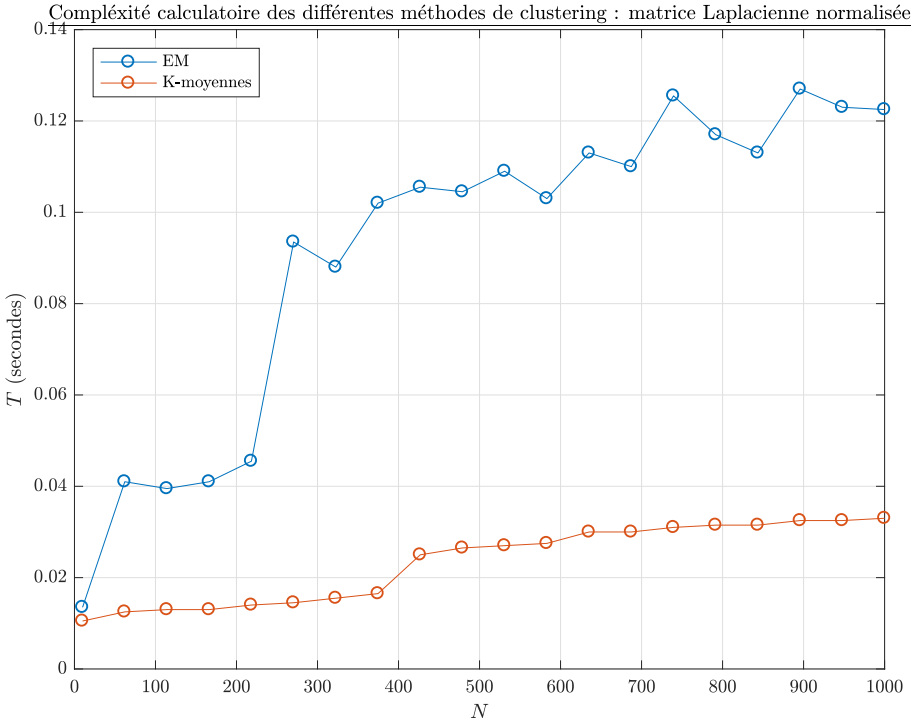


Figure 23 : $c = 10$ $c_{out} = 1$ $K = 2$

La normalisation de la matrice Laplacienne ne modifie pas non plus les coûts calculatoires des algorithmes EM et K-moyennes.

Matrice de Modularité

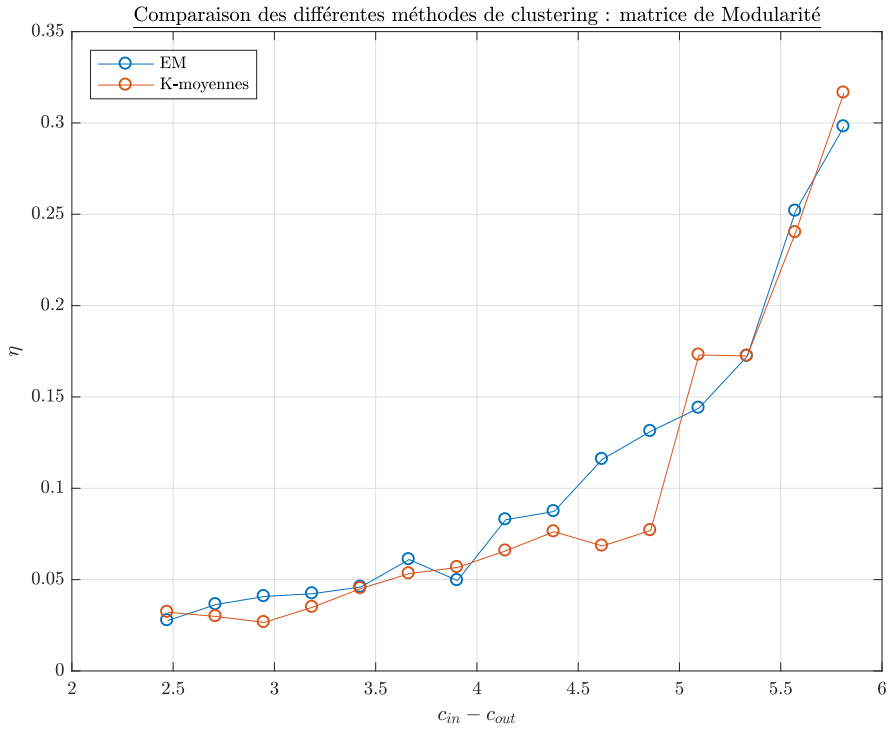


Figure 24 : $c = 3$ $K = 2$ $N = 1000$

On remarque que les deux algorithmes de clustering appliqués aux matrices de modularités ont des taux d'estimation correcte bien inférieurs à ceux obtenus avec les matrices d'adjacence, Laplacienne non normalisée et Laplacienne normalisée. La condition théorique de détection est toujours vérifiée : en dessous de $c_{in} - c_{out} = 3.46$ le taux d'estimation correcte est inférieur à 6%.

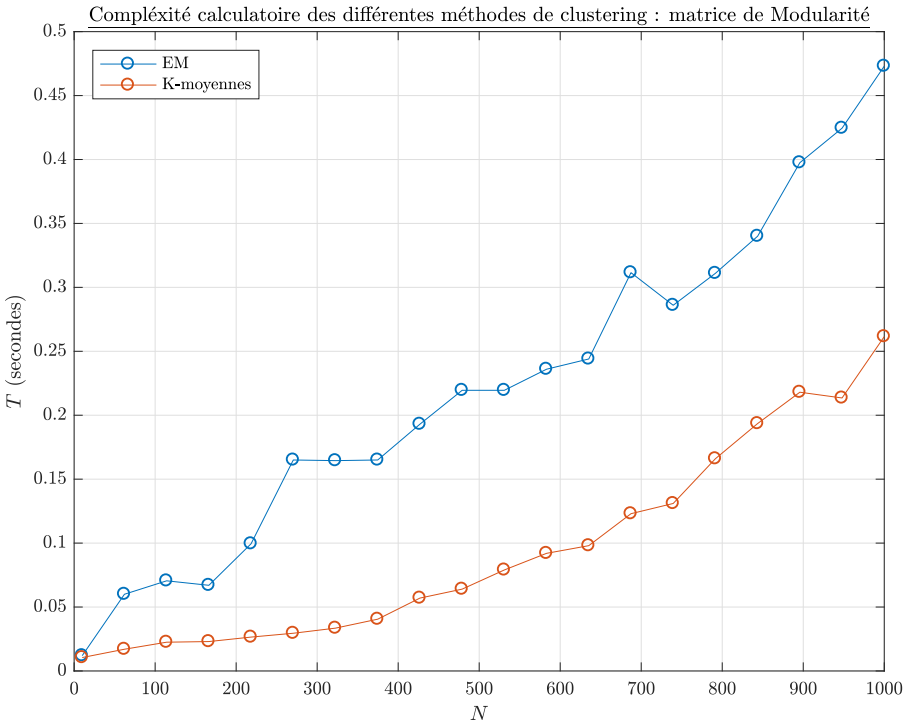


Figure 25 : $c = 10$ $c_{out} = 1$ $K = 2$

Les temps de calcul avec la matrice de modularité sont beaucoup plus élevés qu'avec les matrices d'adjacence et Laplacienne (normalisée et non normalisée). Avec ces dernières, le temps de calcul était de 120 et 30ms respectivement pour EM et K-moyennes avec N=1000. Pour la matrice de modularité, les temps de calcul sont de 470 et 250ms.

Matrice Hessienne de Bethe

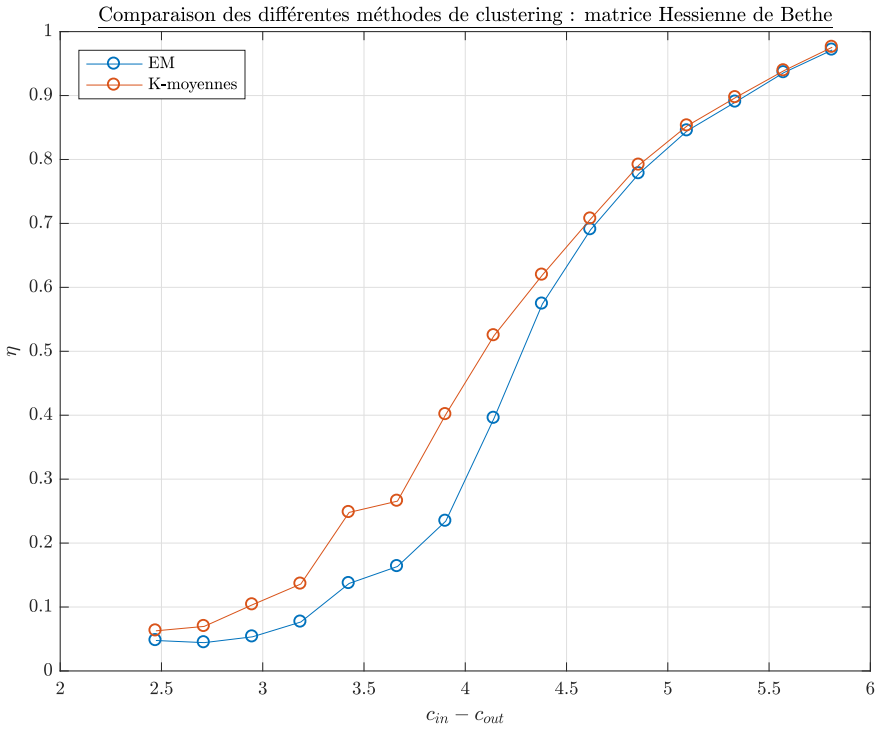


Figure 26 : $c = 3$ $K = 2$ $N = 1000$

Les algorithmes de clustering appliqués à la matrice Hessienne de Bethe ont de bien meilleurs résultats pour la détection de l'appartenance d'un noeud à sa communauté qu'avec les matrices précédentes. En particulier pour $c_{in} - c_{out}$ compris entre 3.46 (limite théorique de détection) et 5, le taux de détection correcte passe d'environ 20% à 80%. En comparaison, les autres matrices donnaient des taux de détection correcte entre 5% et 40% sur le même intervalle. La matrice Hessienne de Bethe est donc bien plus efficace que les autres matrices sur les graphes pour lesquels la différence entre le nombre de liaisons d'un noeud avec des noeuds appartenant à sa communauté et le nombre de liaisons du noeud avec des noeuds n'appartenant pas à sa communauté est faible.

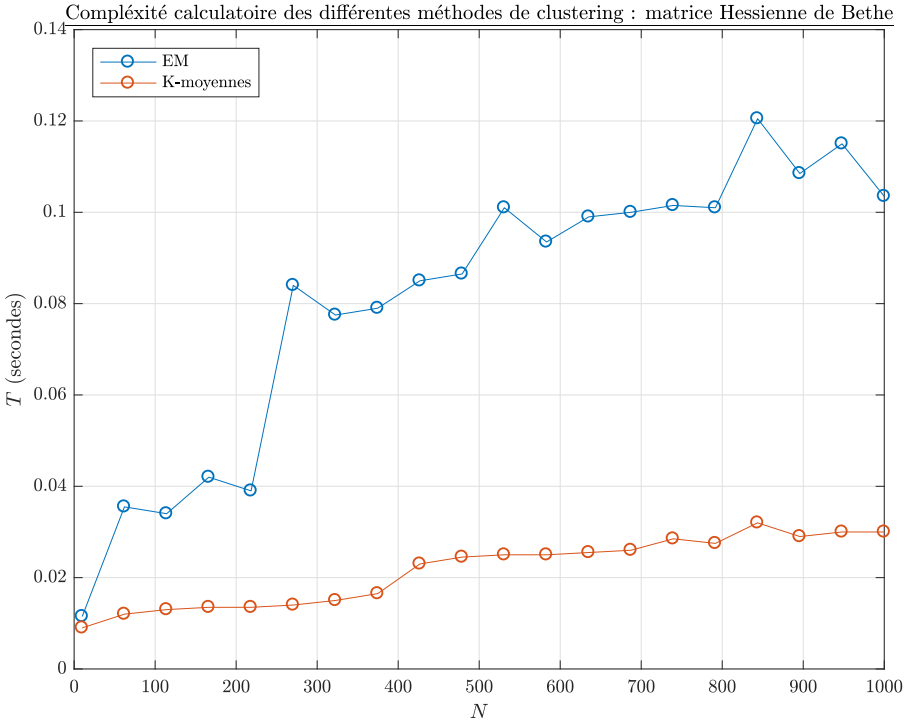


Figure 27 : $c = 10$ $c_{out} = 1$ $K = 2$

Les coûts de calcul sont similaires à ceux obtenus pour les matrices d'adjacence et Laplacienne.

1.2 Algorithmes spectraux de détection de communautés

Par la suite, on utilisera généralement la méthode de clustering K-moyennes qui donne de meilleurs résultats. On compare maintenant les temps de calcul associés à l'utilisation des différentes matrices.

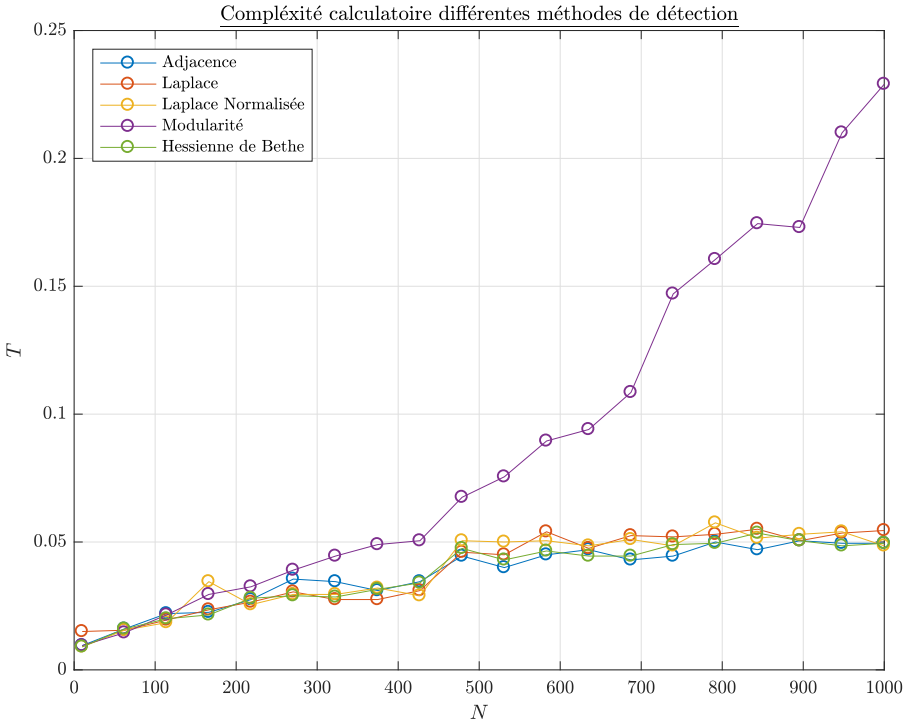


Figure 28 : $c = 3$ $K = 2$ $N = 1000$

On remarque que la matrice de modularité nécessite un temps de calcul beaucoup plus important que les autres matrices, qui ont des temps de calcule similaires.

En prenant des graphes contenant plus de noeuds (100 000 noeuds au maximum), on remarque que les temps de calcul nécessaires pour les différentes matrices évoluent linéairement par rapport à la taille du graphe. La matrice de modularité n'est plus représentée car son temps de calcul est trop élevé.

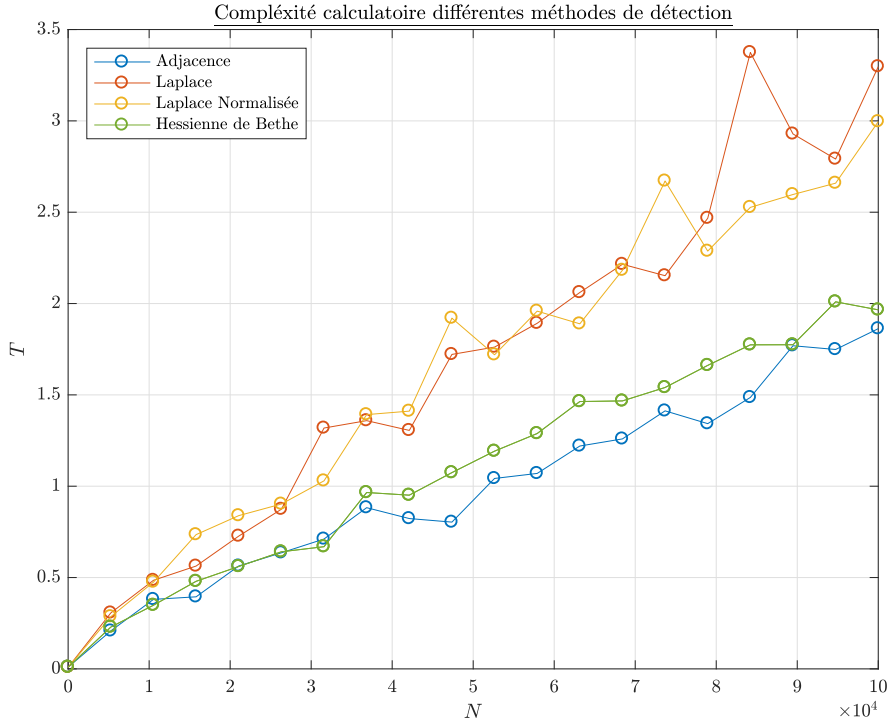


Figure 29 : $c = 3$ $K = 2$ $N = 100000$

2 Détection du nombre de communautés

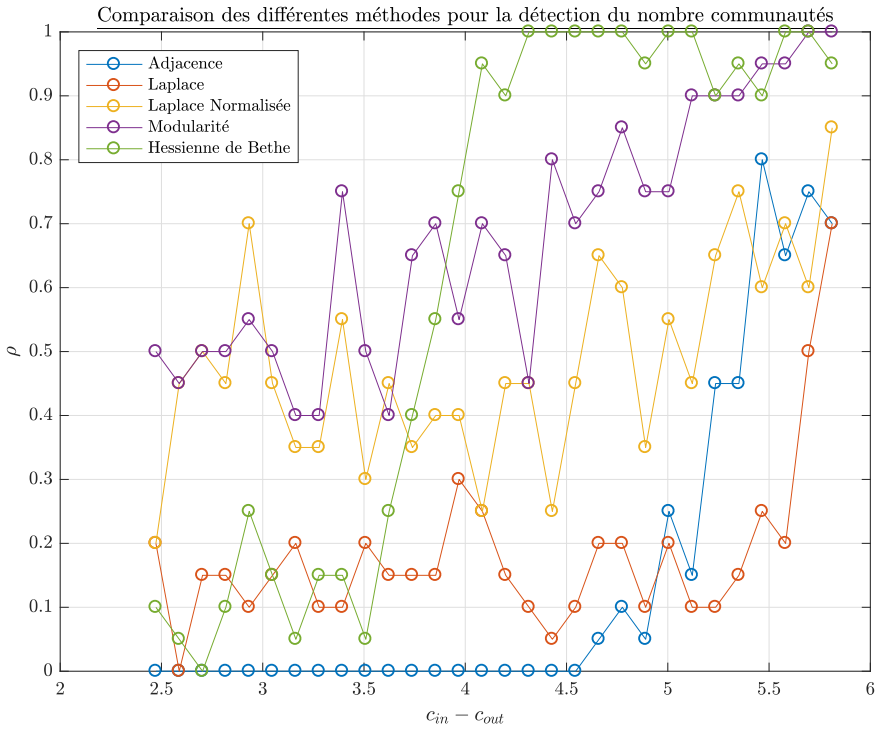


Figure 30 : $c = 3$ $K = 2$ $N = 1000$

Le graphique représente le taux de réussite de la détermination du nombre de communautés en fonction de $c_{in} - c_{out}$. L'utilisation des matrices d'adjacences et Laplacienne donnent une assez mauvaise détermination du nombre de communauté qui composent le graphe lorsque $c_{in} - c_{out}$ est inférieur à 5. La matrice qui permet de déterminer le mieux le nombre de communautés est la matrice Hessienne de Bethe : après la limite théorique de détection (de 3.46), le taux de détermination du nombre de communautés augmente fortement et devient proche de 1 (ce qui signifie que l'on obtient le bon nombre de communautés) pour $c_{in} - c_{out} \geq 4$.

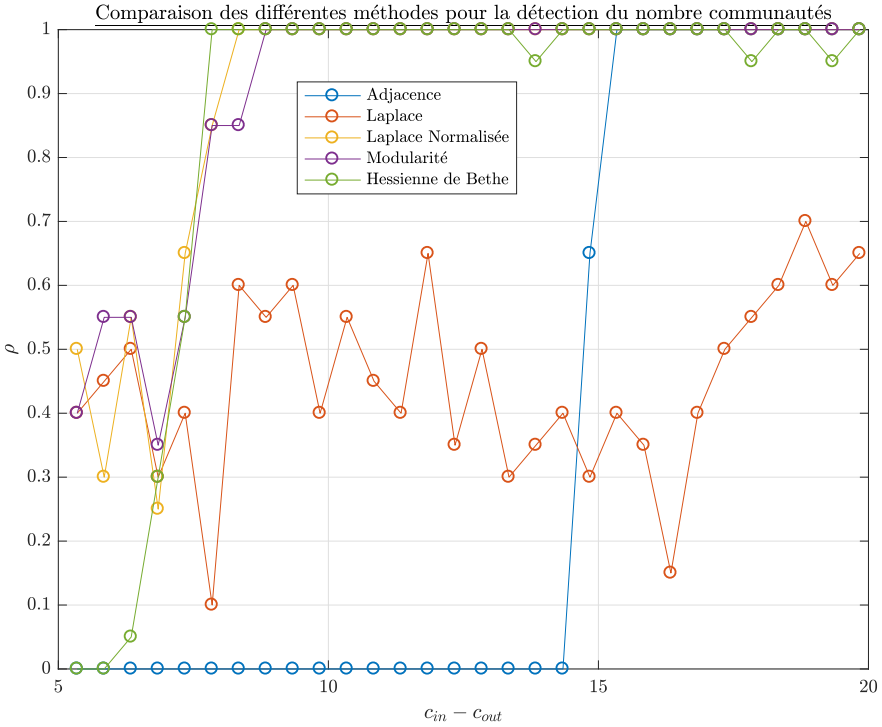


Figure 31 : $c = 10$ $K = 2$ $N = 1000$

Lorsque le nombre moyen de liaisons par noeud augmenté (il était à 3 sur les graphes précédents, il passe à 10). La limite théorique de détection devient égale à 6. On remarque qu'après la valeur 6 pour $c_{in} - c_{out}$, les matrices Hessienne de Bethe, d'adjacence et Laplacienne normalisée permettent de détecter efficacement le nombre de communautés qui composent le graphe.

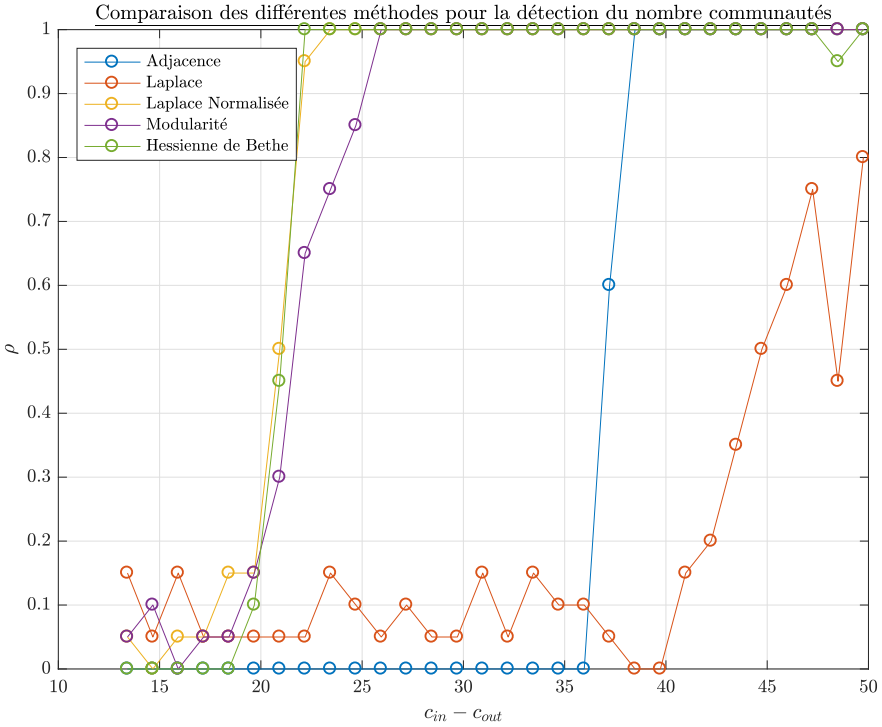


Figure 32 : $c = 10$ $K = 5$ $N = 1000$

On a augmenté K, le nombre de communautés était de 2 sur les graphiques précédents et vaut maintenant 5. La limite théorique de détection vaut maintenant environ 16. On retrouve des résultats similaires au graphique précédent : au delà de limite théorique de détection, les matrices d'adjacence, de modularité et Hessienne de Bethe permettent de déterminer efficacement le nombre de communauté du graphe.

3 Séparation des communautés

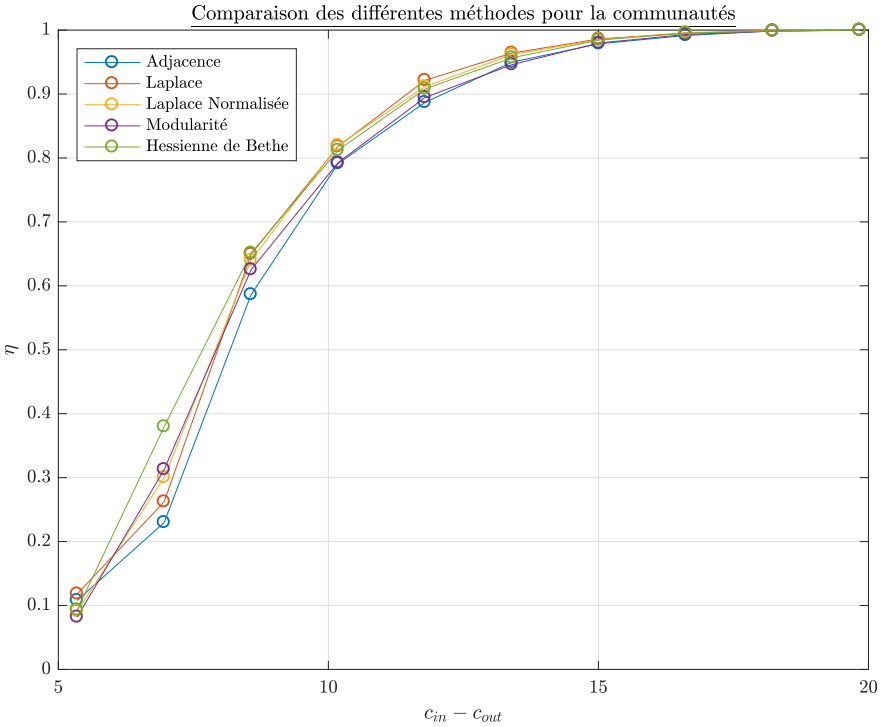


Figure 33 : $c = 10$ $K = 2$ $N = 1000$

Les résultats obtenus montrent que les efficacité des différents algorithmes pour la détection de la communauté à laquelle appartient un noeud sont similaires.

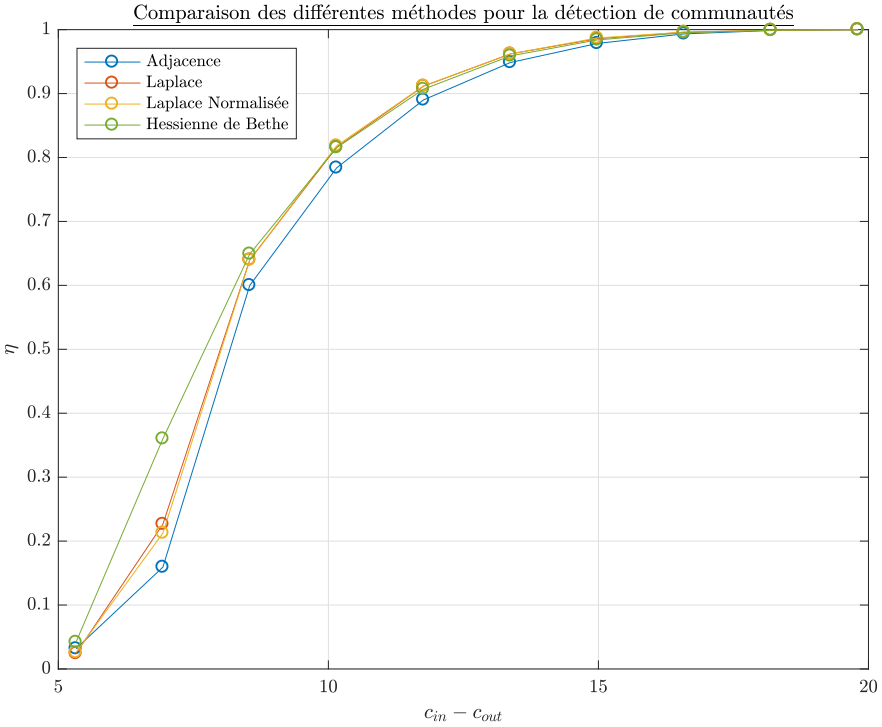


Figure 34 : $c = 10$ $K = 2$ $N = 10000$

Pour des graphes contenant plus de noeuds (ici 10 000), l'efficacité des algorithmes reste similaire. Ici, la matrice de modularité n'est pas testée car sont coût calculatoire est trop important.

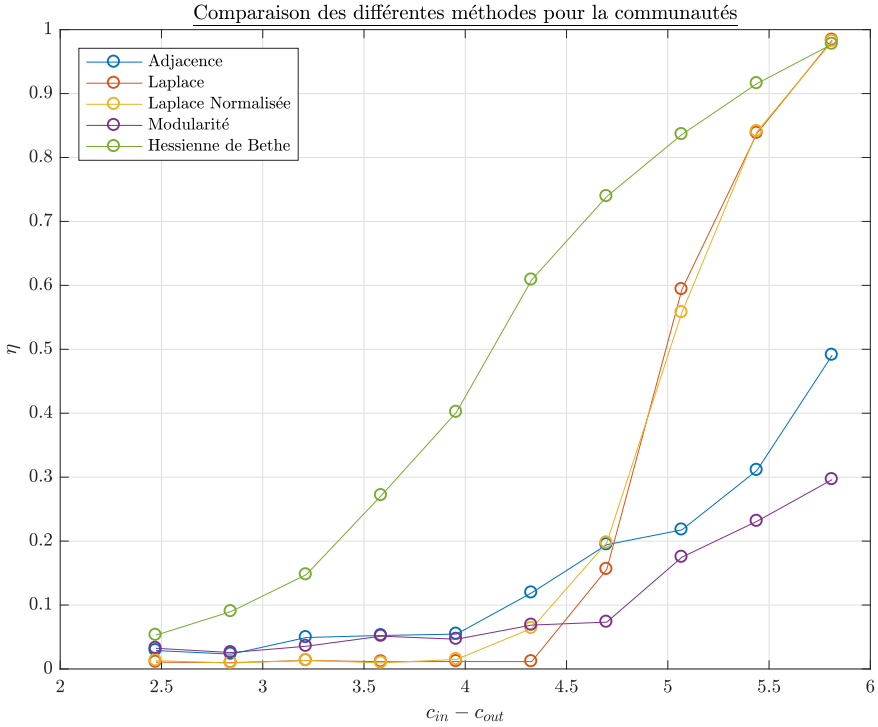


Figure 35 : $c = 3$ $K = 2$ $N = 1000$

La limite théorique de détection est ici de 3.46. On a effectivement pour les valeurs de $c_{in} - c_{out}$ des efficacités faibles pour la détection de la communauté des noeuds. L'utilisation de la matrice Hessienne de Bethe pour ce jeu de données est ici la plus efficace.

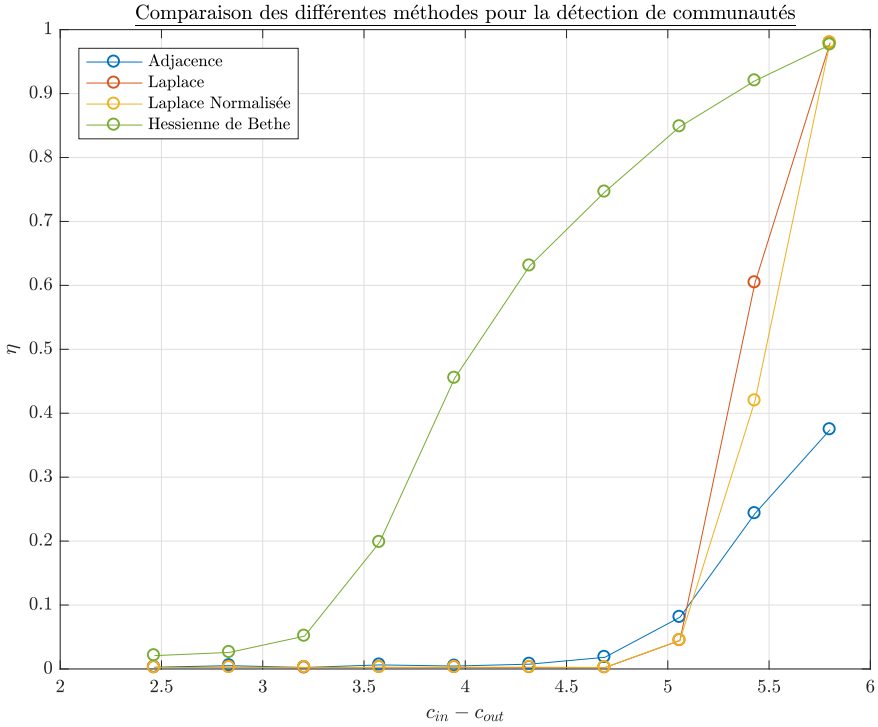


Figure 36 : $c = 3$ $K = 2$ $N = 10000$

On augmente le nombre de noeuds du graphe, (on ne teste pas la matrice de modularité). Par rapport au graphique précédent, la taille des graphes sur lesquelles on a testé les différents algorithmes a augmenté, mais le nombre moyen de liaison par noeud n'a pas changé. On remarque que la matrice Hessienne de Bethe est beaucoup plus efficace que les autres matrices dans ce cas. Par exemple pour $c_{in} - c_{out} = 7$, on détecte la bonne communauté d'appartenance d'un noeud pour 80% des noeuds du graphe, contre moins de 10% pour les autres méthodes.

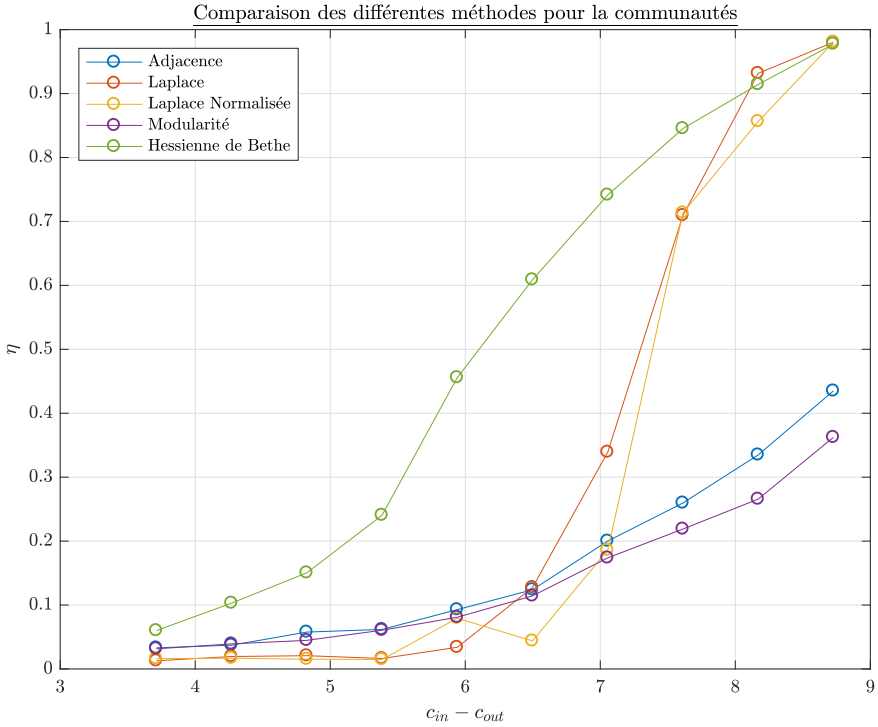


Figure 37 : $c = 3$ $K = 3$ $N = 1000$

On augmente le nombre de communautés (il valait 2, il est maintenant à 3). La limite théorique de détection vaut maintenant 5.2. Les résultats sont similaires à ceux obtenus sur la figure 35.

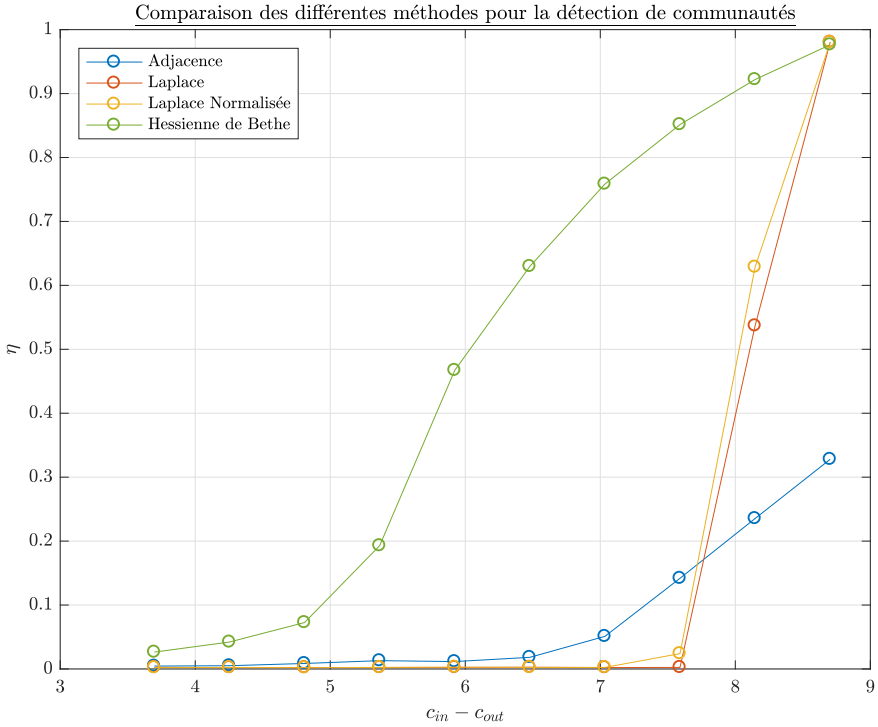


Figure 38 : $c = 3$ $K = 3$ $N = 10000$

La taille du graphe a été multipliée par 10 sans changer le nombre moyen de liaison par noeud. Les conclusions sont identiques.

4 Choix du nombre de vecteurs informatifs

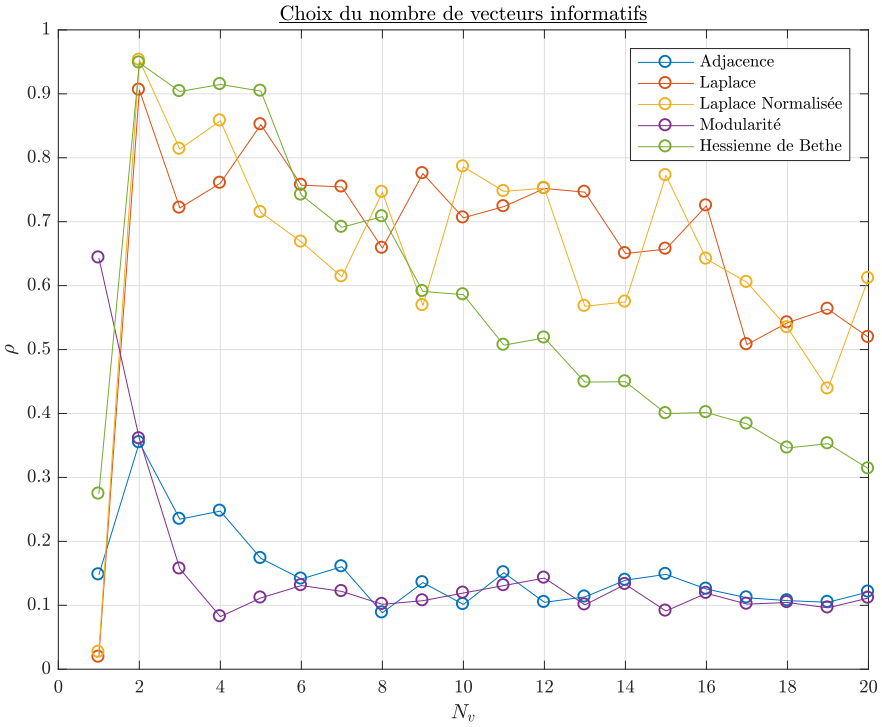


Figure 39 : $c = 3$ $K = 2$ $N = 1000$

Sur ce graphique est représenté la probabilité pour que les algorithmes détectent le bon nombre de communautés en fonction du nombre de vecteurs informatifs utilisés (N_v). On remarque que pour $K = 2$ la meilleure valeur du nombre de vecteurs informatifs est 2, ce qui est cohérent avec nos hypothèses. En particulier pour cette valeur, l'utilisation des matrices Laplacienne, Laplacienne normalisée et Hessienne de Bethe détectent le bon nombre de communautés dans plus de 90% des cas.

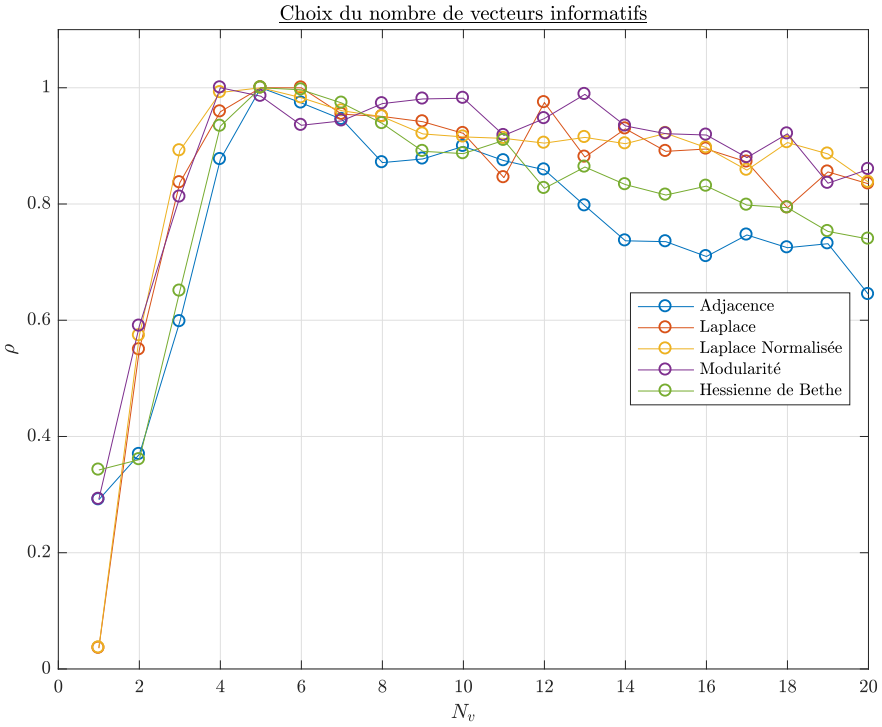


Figure 40 : $c = 10$ $K = 5$ $N = 1000$

On augmente le nombre de communautés ($K = 5$) ainsi que le nombre moyen total de liaisons par noeud. La valeur optimale du nombre de vecteurs informatif est alors de 5. Pour cette valeur, Les algorithmes détectent le bon nombre de communauté dans 90% des cas, y compris avec les matrices de modularité et d'adjacence, ce qui n'étais pas le cas précédemment.

5 Application aux réseaux réels

5.1 Dauphins

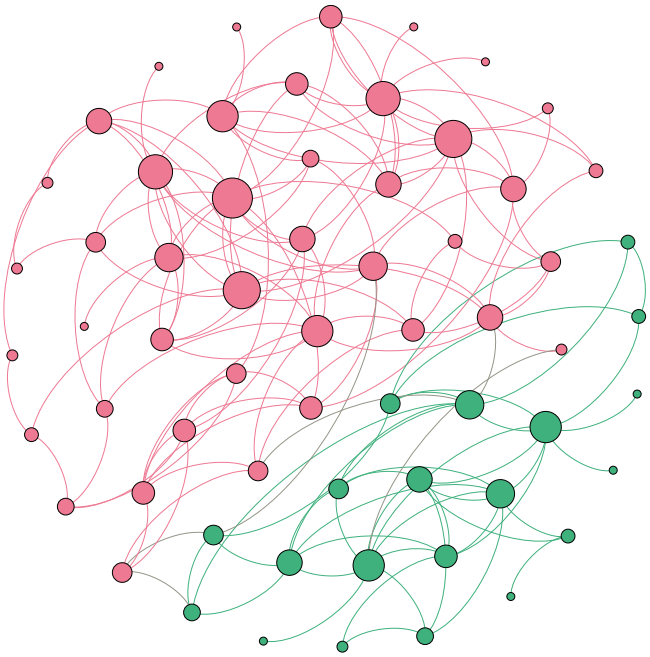


Figure 41 : Application sur un réseau réel : Dauphins ($K = 2$)

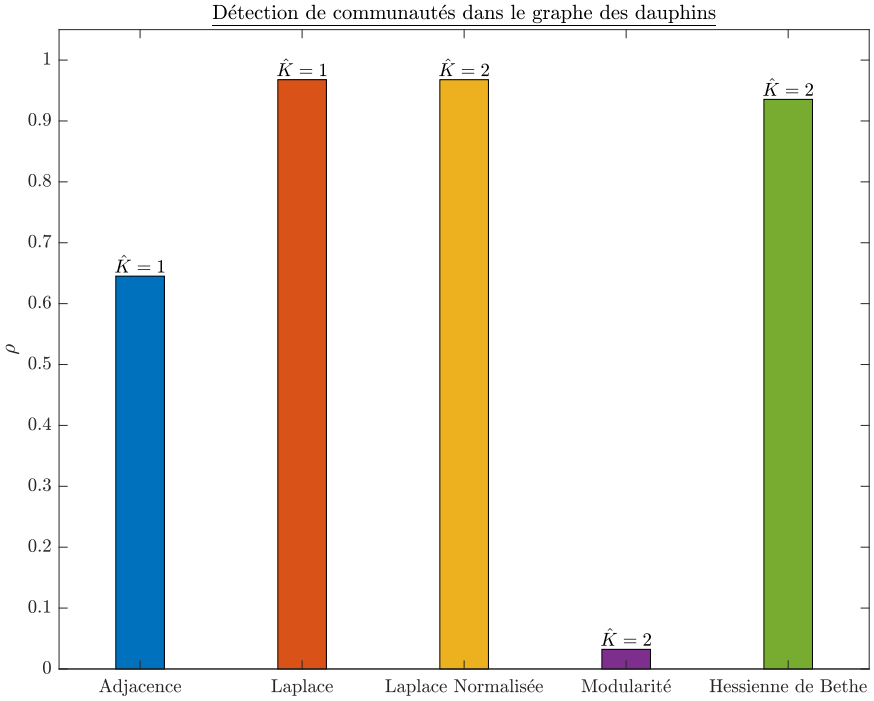


Figure 42 : Détection de $\hat{\mathrm{K}}$ communautés dans le graphe des dauphins ($K = 2$)

5.2 Livres politiques

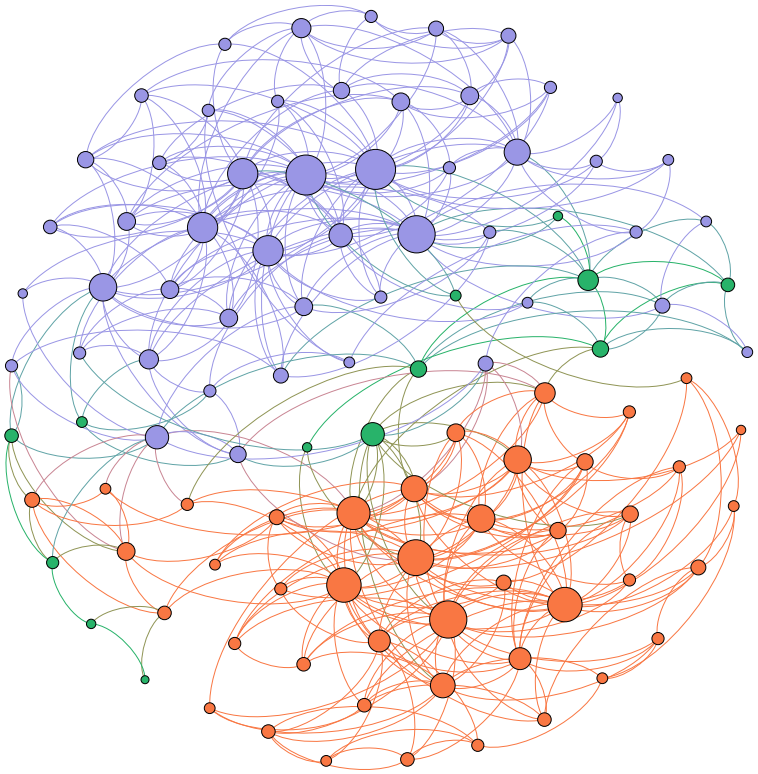


Figure 43 : Application sur un réseau réel : Livres politiques ($K = 3$)

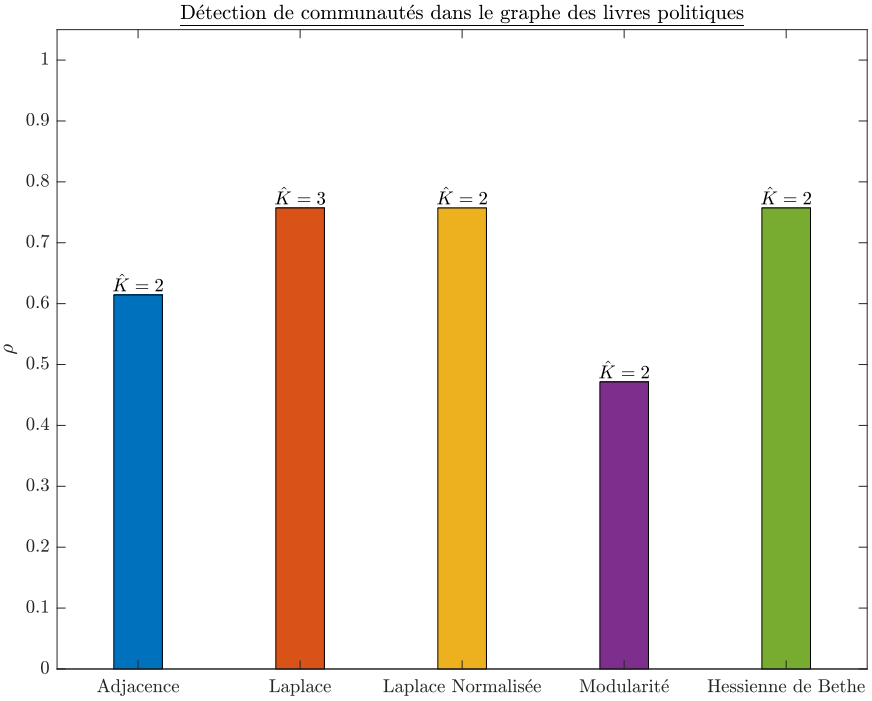


Figure 44 : Détection de $\hat{\mathrm{K}}$ communautés dans le graphe des livres politiques ($K = 3$)

5.3 Blogs politiques

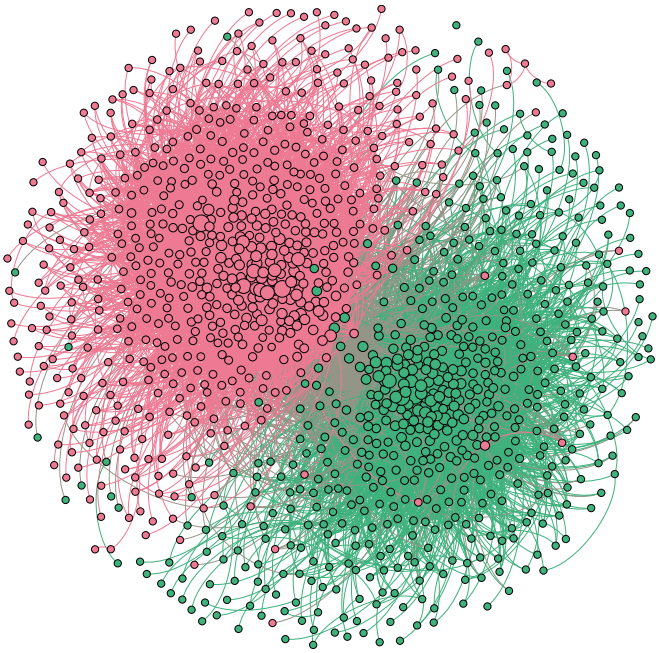


Figure 45 : Application sur un réseau réel : Blogs politiques ($K = 2$)

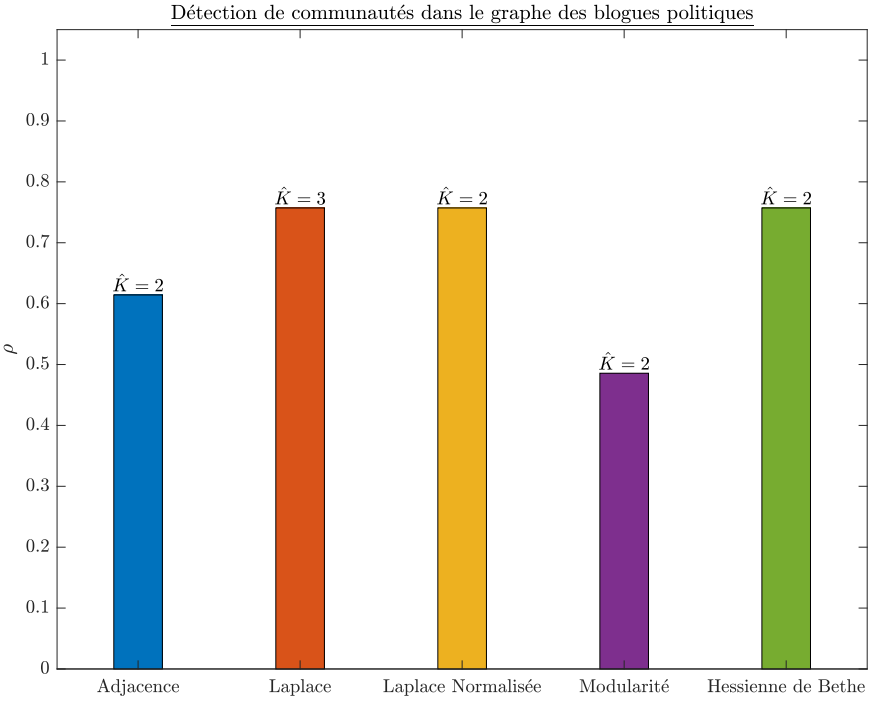


Figure 46 : Détection de $\hat{\mathrm{K}}$ communautés dans le graphe des blogs politiques ($K = 2$)

5.4 Karaté

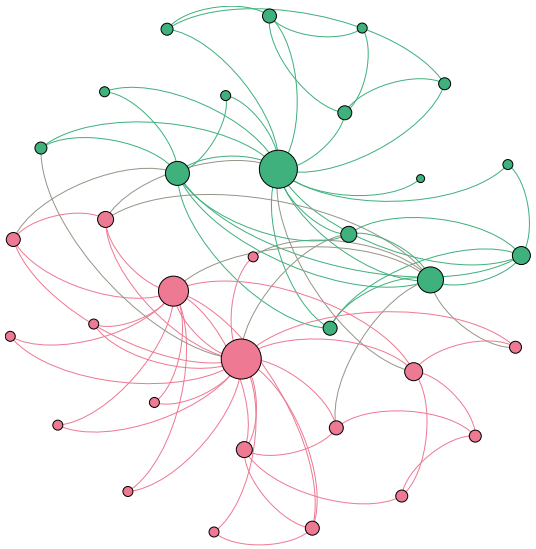


Figure 47 : Application sur un réseau réel : Karaté ($K = 2$)

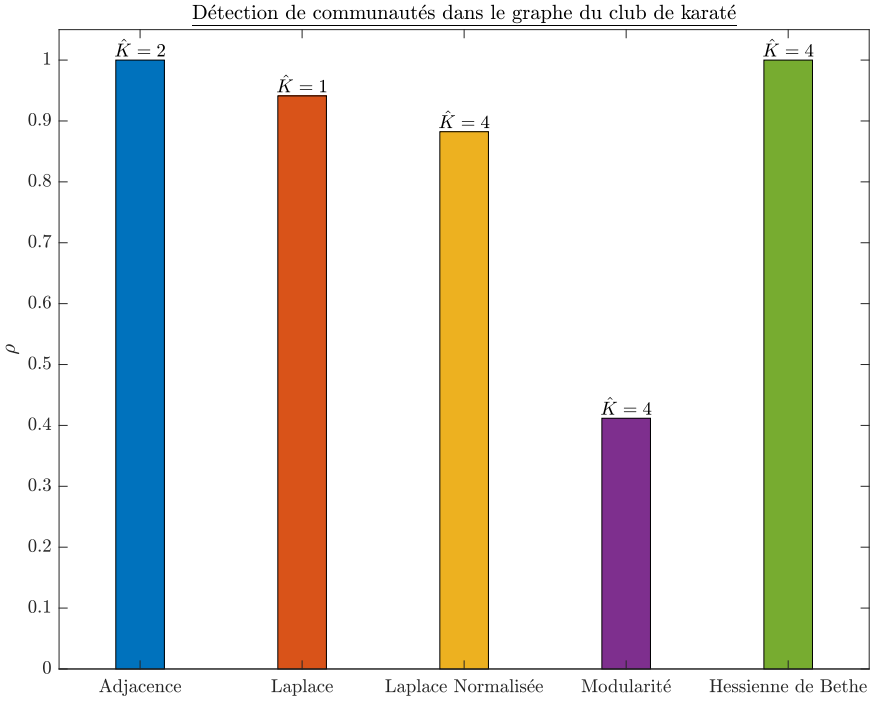


Figure 48 : Détection de $\hat{\mathrm{K}}$ communautés dans le graphe des karaté ($K = 2$)

Conclusion

Bilan

Notre étude théorique sur les méthodes spectrales pour la détection de groupes d'affinités nous a d'abord amené à comprendre les différents outils mathématiques permettant de représenter les réseaux.

La matrice d'adjacence permet de représenter la structure des graphes. Les notions de voisinages, de distances entre les noeuds ou encore de degrés permettent de décrire certaines propriétés des réseaux. La connaissance de ces propriétés nous a permis de générer des graphes à l'aide du Stochastic Block Model.

Nous avons ensuite étudié deux algorithmes itératifs de clustering : l'algorithme EM et l'algorithme K-moyennes, permettant de partitionner des données.

Enfin nous nous sommes intéressés aux méthodes spectrales de partitionnement, permettant de déterminer les clusters du graphes à partir des valeur propres de certaines matrices. On fait appel aux algorithmes de clustering cités précédemment dans le nouvel espace obtenu afin de détecter les communautés.

La partie expérimentale nous a permis d'implémenter et de tester les performances des différentes méthodes de détection de groupes de communautés.

En particulier, nous avons pu comparer la complexité des algorithmes de clustering, puis pour les méthodes utilisant les matrices d'adjacence, Laplacienne (non normalisée et normalisée), de modularité et Hessienne de Bethe, nous avons testé les performances de détection du nombre de communautés, de séparation, ainsi que les temps de calcul nécessaires aux différentes méthodes. L'étude de ces caractéristiques à été réalisé en variant les paramètres des graphes, notamment leurs tailles et le nombre moyen de liaisons des noeuds au sein et en dehors de leurs communautés.

Les résultats que nous avons obtenus nous ont également permis de vérifier expérimentalement la limite théorique de détection et de comparer les différentes méthodes et algorithmes.

On pourra retenir de notre étude que l'utilisation de la matrice de modularité ne semble pas adaptée au grands graphes car son temps de calcul est trop élevé. D'autre part, la matrice Hessienne de Bethe s'est prouvée relativement efficace par rapport aux autres méthodes spectrales, surtout lorsque le nombre moyen de liaisons des noeuds est faible par rapport à la taille du graphe.

Enfin, nous avons pu appliquer nos implémentations à des exemples réels, grâce à des bases de données référencées dans la littérature.

A Bibliographie

[1] A. Barrat, M. Barthélemy, R. Pastor-Satorras, and A. Vespignani. The architecture of complex weighted networks. PNAS, 2004.

[2] Pin-Yu Chen and Lingfei Wu. Revisiting spectral graph clustering with generative community models. Data Mining (ICDM), 2017 IEEE International Conference, 2017.

[3] Leon Danon, Albert Díaz-Guilera, Jordi Duch, and Alex Arenas. Comparing community structure identification. Journal of Statistical Mechanics: Theory and Experiment, 2005(09):P09008–P09008, sep 2005.

[4] Santo Fortunado. Community detection in graphs. Physics Reports, 2010.

[5] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. Physics Reports, 659:1–44, nov 2016.

[6] Stefanie Kosuch. Théorie des Graphes. Université Paris-Sud, 2009.

[7] Catherine Matias. Stochastic block models for random graphs. CNRS.

[8] Mohammad Sal Moslehian. Ky fan inequalities. Linear and Multilinear Algebra, 2012.

[9] Raj Rao Nadakuditi and Mark EJ Newman. Graph spectra and the detectability of community structure in networks. Physical review letters, 2012.

[10] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. Physical Review E, 74(3), sep 2006.

[11] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. Physical Review E, 69(2), feb 2004.

[12] Ricco Rakotomalala. Détection de communautés dans les réseaux sociaux. Université Lyon 2.

[13] Alaa Saade, Florent Krzakala, and Lenka Zdeborova. Spectral clustering of graphs with the bethe hessian. Advances in Neural Information Processing Systems, 2014.

[14] Alaa Saade, Florent Krzakala, and Lenka Zdeborová. Spectral clustering of graphs with the bethe hessian.

[15] Daniel Spielman. Spectral Graph Theory. Yale University.

[16] Scott White and Padhraic Smyth. A spectral clustering approach to finding communities in graphs. In Proceedings of the 2005 SIAM International Conference on Data Mining, pages 274–285. Society for Industrial and Applied Mathematics, apr 2005.

B Codes Matlab

1 Stochastic Block Model

Générateur de graphes

```
function [A E] = generateSBM(N,K,cin,cout)
% generateSBM — Génère une matrice d'adjacence A et une matrice d'appartenance
% E d'un graphe possèdant N noeuds et K communautés de taille identique. N
% doit être un multiple de K sans quoi l'entier le multiple le plus proche
% sera choisi.
%
% Entrées:
%    N — Nombre de noeud
%    K — Nombre de communautés
%    cin — Degré intracommunautaire
%    cout — Degré extracommunautaire
%
% Sorties:
%    A — Matrice d'adjacence du graphe généré de taille NxN
%    E — Matrice d'appartenance des noeuds de taille NxK

taille_com=floor(N/K);
N=K*taille_com;
E=zeros(N,K);

for k=1:K
    E(((k-1)*taille_com+1):(k*taille_com),k)=1;
end

A=sparse(N,N);
for i=1:K
    com_courante=sprand(taille_com,taille_com,cin/N);
    com_courante=(com_courante~=0);
    com_courante=triu(com_courante,1);
    com_courante=com_courante+com_courante';
    A((1+(i-1)*taille_com):((i)*taille_com),(1+(i-1)*taille_com):((i)*taille_com))=
        com_courante;
    for j=i+1:K
        com_courante=sprand(taille_com,taille_com,cout/N);
        com_courante=(com_courante~=0);
        A((1+(i-1)*taille_com):((i)*taille_com),1+(j-1)*(taille_com):(j)*taille_com)
            =com_courante;
        A(1+(j-1)*(taille_com):(j)*taille_com,(1+(i-1)*taille_com):((i)*taille_com))
            =com_courante';
    end
```

```
38 end
39
40 [A p]=getGrapheConnexe(A);
41 E=E(p,:);
42
43 end
```

2 Algorithme de clustering

Algorithme K-means

```
function [Y] = Kmeans(U,K)
% EM – Réalise un clustering selon la méthode K–means sur les lignes de U
% selon K communautés
%
% Entrées:
%    U – Matrice sur laquelle réaliser le clustering
%    K – Nombre de communautés
%
% Sorties:
%    Y – Matrice de partitionnement de taille NxK

options = statset('MaxIter',300,'TolFun',1e–6);

N=size(U,1);

IDY = kmeans(U, K,'Replicates',5,'Options',options);

Y=zeros(N,K);

for i=1:N
    Y(i,IDY(i))=1;
end

end
```

Algorithme Espérance-Maximisation

```
function [Y] = EM(U,K)
% EM — Réalise un clustering selon la méthode Espérance—Maximisation sur
% les lignes de U selon K communautés
%
% Entrées:
%    U — Matrice sur laquelle réaliser le clustering
%    K — Nombre de communautés
%
% Sorties:
%    Y — Matrice de partitionnement de taille NxK

options = statset('MaxIter',300,'TolFun',1e—6);

N=size(U,1);

try
    obj = fitgmdist(U,K,'Options',options,'Replicates',5);
    IDY = cluster(obj,U);
    Y=zeros(N,K);
    for i=1:N
        Y(i,IDY(i))=1;
    end

catch exception
    disp('There was an error fitting the Gaussian mixture model')
    Y=zeros(N,K);
end

end
```

3 Algorithme de détection du nombre de communautés

Matrice d'Adjacence

```
function [K] = NCAdjacence(A)
% NCAdjacence — Estime le nombre de communautés que contient le graphe
% via le calcul de sa matrice d'adjacence
%
% Entrées:
%    A — Matrice d'adjacence
%
% Sorties:
%    K — Nombre de communautés

opts.maxit=50;
opts.isreal=1;
opts.issym=1;
opts.tol=1e-3;

N=size(A,1);
[U , eigenvalues]=eigs(A,N);

eigenvalues=sort(diag(eigenvalues),'descend');

deltaeig=zeros(N-1,1);

for i=1:(N-1)
    deltaeig(i)=abs(eigenvalues(i+1)-eigenvalues(i));
end

[maxi K]=max(deltaeig);

end
```

Matrice Laplcienne

```
function [K] = NCLaplacien(A)
% NCLaplacien — Estime le nombre de communautés que contient le graphe
% via le calcul de sa matrice lapalcienne
%
% Entrées:
%    A — Matrice d'adjacence
%
% Sorties:
%    K — Nombre de communautés

opts.maxit=50;
opts.isreal=1;
opts.issym=1;
opts.tol=1e-3;

D=diag(sum(A));

L=D-A;

N=size(A,1);
[U , eigenvalues]=eigs(L,N);

eigenvalues=sort(diag(eigenvalues),'ascend');

deltaeig=zeros(N-1,1);

for i=2:(N-100)
    deltaeig(i)=abs(eigenvalues(i+1)-eigenvalues(i));
end

[maxi K]=max(deltaeig);

end
```

Matrice Laplacienne normalisée

```
function [K] = NCLaplacienNorm(A)
% NCLaplacienNorm — Estime le nombre de communautés que contient le graphe
% via le calcul de sa matrice laplacienne normalisée
%
% Entrées:
%    A — Matrice d'adjacence
%
% Sorties:
%    K — Nombre de communautés

opts.maxit=50;
opts.isreal=1;
opts.issym=1;
opts.tol=1e-3;

D=diag(sum(A));

Dmoinsundemi=spfun(@(x)1./sqrt(x),D);

LN=Dmoinsundemi*(D-A)*Dmoinsundemi;


N=size(A,1);
[U , eigenvalues]=eigs(LN,N);

eigenvalues=sort(diag(eigenvalues),'ascend');


deltaeig=zeros(N-1,1);

for i=2:(N-10)
    deltaeig(i)=abs(eigenvalues(i+1)-eigenvalues(i));
end

[maxi K]=max(deltaeig);

end
```

Matrice de Modularité

```
function [K] = NCModularite(A)
% NCModularite — Estime le nombre de communautés que contient le graphe
% via le calcul de sa matrice de modularite
%
% Entrées:
%    A — Matrice d'adjacence
%
% Sorties:
%    K — Nombre de communautés

opts.isreal=1;
opts.issym=1;
opts.tol=1e—3;

N=size(A,1);

d=full(sum(A));
A=full(A);
m=full(sum(A(:)))/2;
temp=triu((1/(2*m))*d'*d,1);
B=A—(temp+diag(diag((1/(2*m))*d'*d))+temp');

[eigenvectors , eigenvalues]=eigs(B,N);

eigenvalues=sort(diag(eigenvalues),'descend');

for i=1:(N—100)
    deltaeig(i)=abs(eigenvalues(i+1)—eigenvalues(i));
end

[maxi K]=max(deltaeig);

K=K+1;

end
```

Hessienne de Bethe

```
function [K] = NCBetheHessian(A)
% NCBetheHessian — Estime le nombre de communautés que contient le graphe
% via le calcul de sa hessienne de Bethe
%
% Entrées:
%    A — Matrice d'adjacence
%
% Sorties:
%    K — Nombre de communautés

opts.maxit=50;
opts.isreal=1;
opts.issym=1;
opts.tol=1e-3;

N=size(A,1);
D=diag(sum(A));
r=sum(A(:))/N;
r=sqrt(mean(diag(D).^2)/mean(diag(D))-1);
degrees=sum(A);

BH=(r^2-1)*eye(N,N)-r*A+D;

[eigenvectors , eigenvalues]=eigs(BH,N);

eigenvalues=sort(diag(eigenvalues));

K=sum(eigenvalues<0);

end
```

4 Algorithme de détection des communautés

Matrice d'Adjacence

```
function [Y] = Adjacence(A,K,Nv,ClusterMethod)
% Adjacence — Réalise une séparation du graphe en K communautés à l'aide
% de la matrice d'adjacence
%
% Entrées:
%    A — Matrice d'adjacence
%    K — Nombre de communautés
%    Nv — Nombre de vecteurs informatifs
%    ClusterMethod — Méthode de clustering : EM ou KM
%
% Sorties:
%    Y — Matrice de partitionnement de taille NxK

opts.maxit=50;
opts.isreal=1;
opts.issym=1;
opts.tol=1e-3;

N=size(A,1);

[U , eigenvalues]=eigs(A,Nv,'la',opts);

if ClusterMethod=='EM'
    Y=EM(U,K);
else
    Y=Kmeans(U,K);
end

end
```

Matrice Laplcienne

```
function [Y] = Laplacien(A,K,Nv,ClusterMethod)
% Laplacien - Réalise une séparation du graphe en K communautés à l'aide
% de la matrice d'adjacence
%
% Entrées:
%    A - Matrice d'adjacence
%    K - Nombre de communautés
%    Nv - Nombre de vecteurs informatifs
%    ClusterMethod - Méthode de clustering : EM ou KM
%
% Sorties:
%    Y - Matrice de partitionnement de taille NxK

opts.maxit=50;
opts.isreal=1;
opts.issym=1;
opts.tol=1e-3;

N=size(A,1);

D=diag(sum(A));

Dmoinsundemi=spfun(@(x)1./sqrt(x),D);

LN=Dmoinsundemi*(D-A)*Dmoinsundemi;

[U , eigenvalues]=eigs(LN,Nv,'sa',opts);

if ClusterMethod=='EM'
    Y=EM(U,K);
else
    Y=Kmeans(U,K);
end

end
```

Matrice Laplacienne normalisée

```
function [Y] = LaplacienNorm(A,K,Nv,ClusterMethode)
% LaplacienNorm — Réalise une séparation du graphe en K communautés à l'aide
% de la matrice d'adjacence
%
% Entrées:
%    A — Matrice d'adjacence
%    K — Nombre de communautés
%    Nv — Nombre de vecteurs informatifs
%    ClusterMethod — Méthode de clustering : EM ou KM
%
% Sorties:
%    Y — Matrice de partitionnement de taille NxK

opts.maxit=50;
opts.isreal=1;
opts.issym=1;
opts.tol=1e—3;

N=size(A,1);

D=diag(sum(A));

Dmoinsundemi=spfun(@(x)1./sqrt(x),D);

LN=Dmoinsundemi*(D—A)*Dmoinsundemi;

[U , eigenvalues]=eigs(LN,Nv,'sa',opts);


if ClusterMethode=='EM'
    Y=EM(U,K);
else
    Y=Kmeans(U,K);
end

end
```

Matrice de Modularité

```
function [Y] = Modularite(A,K,Nv,ClusterMethod)
% Modularite — Réalise une séparation du graphe en K communautés à l'aide
% de la matrice d'adjacence
%
% Entrées:
%    A — Matrice d'adjacence
%    K — Nombre de communautés
%    Nv — Nombre de vecteurs informatifs
%    ClusterMethod — Méthode de clustering : EM ou KM
%
% Sorties:
%    Y — Matrice de partitionnement de taille NxK

opts.maxit=50;
opts.isreal=1;
opts.issym=1;
opts.tol=1e-3;

d=sum(A);
m=sum(A(:))/2;
temp=triu((1/(2*m))*d'*d,1);
B=A-(temp+diag(diag((1/(2*m))*d'*d))+temp');

t=cputime;
[eigenvectors , eigenvalues]=eigs(B,Nv,'la',opts);
U=eigenvectors;

if ClusterMethod=='EM'
    Y=EM(U);
else
    Y=Kmeans(U,K);
end

end
```

Hessienne de Bethe

```
function [Y] = BetheHessian(A,K,Nv,ClusterMethod)
% BetheHessian — Réalise une séparation du graphe en K communautés à l'aide
% de la matrice d'adjacence
%
% Entrées:
%    A — Matrice d'adjacence
%    K — Nombre de communautés
%    Nv — Nombre de vecteurs informatifs
%    ClusterMethod — Méthode de clustering : EM ou KM
%
% Sorties:
%    Y — Matrice de partitionnement de taille NxK

opts.maxit=50;
opts.isreal=1;
opts.issym=1;
opts.tol=1e-3;

N=size(A,1);
D=diag(sum(A));
r=sum(A(:))/N;
r=sqrt(mean(diag(D).^2)/mean(diag(D))-1);
degrees=sum(A);

BH=(r^2-1)*speye(N,N)-r*A+D;

[U , eigenvalues]=eigs(BH,Nv,'sa',opts);

if ClusterMethod=='EM'
    Y=EM(U,K);
else
    Y=Kmeans(U,K);
end

end
```

5 Simulations

Complexité calculatoire

```
function [time] = getComplexite(N,K,cin,cout,Methode,Nv,ClusterMethode,Ns);
% getComplexite - Calcul de la compléxité des méthodes matricielles de
% détections de communautés
%
% Entrées:
%    N - Nombre de noeud
%    K - Nombre de communautés
%    cin - Degré intracommunautaire
%    cout - Degré extracommunautaire
%    Methode - Méthode matricielle : 'A','L','LN','M','BH'
%    Nv - Nombre de vecteurs informatifs choisis
%    ClusterMethode - Méthode de clustering : 'EM' ou 'KM'
%    Ns - Nombre d'échantillons pour Monte-Carlo
%
% Sorties:
%    time - Temps CPU consommé


t=cputime;
MonteCarlo(N,K,cin,cout,Methode,Nv,ClusterMethode,Ns);
time=(cputime-t)/Ns;

end
```

Recouvrement

```
function [ eta ] = getRecouvrement(E,Y,K)
% getRecouvrement — Calcul de le recouvrement d'une matrice Y sur une
% matrice E d'appartenance de communautés
%
% Entrées:
%    E — Matrice d'appartenance
%    Y — Estimée de la matrice d'appartenance
%    K — Nombre de communautés

% Sorties:
%    time — Temps CPU consommé

N=size(E,1);
P=perms(1:K);
recouvs=zeros(factorial(K),1);

for i=1:factorial(K)
    p=P(i,:);
    Yp=Y(:,p);
    M=Yp+E;
    BC=sum(M(:) == 2);
    recouvs(i)=(BC/N-1/K)/(1-1/K);
end

eta=max(recouvs);

end
```

Détection du nombre de communautés

```
function [Ke] = NCsimul(N,K,cin,cout,Methode)
% NCsimul — Réalise une simulation de détéction du nomnre de commuanutés
% selon la méthode matricielle choisie
%
% Entrées:
%    N — Nombre de noeud
%    K — Nombre de communautés
%    cin — Degré intracommunautaire
%    cout — Degré extracommunautaire
%    Methode — Méthode matricielle : 'A','L','LN','M','BH'
%
% Sorties:
%    Ke — Estimée du nombre de communautés

[A E]=generateSBM(N,K,cin,cout);

if Methode=='A'
   Ke=NCAdjacence(A);
end

if Methode=='L'
   Ke=NCLaplacien(A);
end

if Methode=='LN'
   Ke=NCLaplacienNorm(A);
end

if Methode=='M'
   Ke=NCModularite(A);
end

if Methode=='BH'
   Ke=NCBetheHessian(A);
end

end
```

Détection des communautés

```
function [eta] = simul(N,K,cin,cout,Methode,Nv,ClusterMethode)
% simul — Réalise une simulation de détéction des commuanutés selon
% la méthode de matricielle choisie
%
% Entrées:
%    N — Nombre de noeud
%    K — Nombre de communautés
%    cin — Degré intracommunautaire
%    cout — Degré extracommunautaire
%    Methode — Méthode matricielle : 'A','L','LN','M','BH'
%    Nv — Nombre de vecteurs informatifs choisis
%    ClusterMethode — Méthode de clustering : 'EM' ou 'KM'
%
% Sorties:
%    eta — Recouvrement

[A E]=generateSBM(N,K,cin,cout);

if Methode=='A'
   Y=Adjacence(A,K,Nv,ClusterMethode);
   eta=getRecouvrement(Y,E,K) ;
end

if Methode=='L'
   Y=Laplacien(A,K,Nv,ClusterMethode);
   eta=getRecouvrement(Y,E,K) ;
end

if Methode=='LN'
   Y=LaplacienNorm(A,K,Nv,ClusterMethode);
   eta=getRecouvrement(Y,E,K) ;
end

if Methode=='M'
   Y=Modularite(A,K,Nv,ClusterMethode);
   eta=getRecouvrement(Y,E,K) ;
end

if Methode=='BH'
   Y=BetheHessian(A,K,Nv,ClusterMethode);
    eta=getRecouvrement(Y,E,K) ;
end

end
```

Monte-Carlo

```
function [rho] = NCMonteCarlo(N,K,cin,cout,Methode,Ns)
% MonteCarlo — Réalise une simulation de détéction du nomnre de commuanutés
% selon la méthode de Monte—Carlo de Ns échantillons
%
% Entrées:
%    N — Nombre de noeud
%    K — Nombre de communautés
%    cin — Degré intracommunautaire
%    cout — Degré extracommunautaire
%    Methode — Méthode matricielle : 'A','L','LN','M','BH'
%    Ns — Nombre d'échantillons pour Monte—Carlo
%
% Sorties:
%   rho — Pouvoir de détection du nombre de communautés

Kes=zeros(Ns,1);

for i=1:Ns
    Kes(i)=NCsimul(N,K,cin,cout,Methode);
end

rho=sum(Kes==K)/Ns;

end
```

```
function [ eta stdrecouvs ] = MonteCarlo(N,K,cin,cout,Methode,Nv,ClusterMethode,Ns);
% MonteCarlo — Réalise une simulation de détéction des commuanutés selon
% la méthode de Monte—Carlo de Ns échantillons
%
% Entrées:
%    N — Nombre de noeud
%    K — Nombre de communautés
%    cin — Degré intracommunautaire
%    cout — Degré extracommunautaire
%    Methode — Méthode matricielle : 'A','L','LN','M','BH'
%    Nv — Nombre de vecteurs informatifs choisis
%    ClusterMethode — Méthode de clustering : 'EM' ou 'KM'
%    Ns — Nombre d'échantillons pour Monte—Carlo
%
% Sorties:
%    moyrecouvs — Moyenne des recouvrements
%    stdrecouvs — Écart type des recouvrements

for i=1:Ns
    recouvs(i)=simul(N,K,cin,cout,Methode,Nv,ClusterMethode);
end

eta=mean(recouvs);
stdrecouvs=std(recouvs);

end
```