Université de Liege

FACULTÉ DES SCIENCES APPLIQUÉES
Ecole d'ingénieurs et d'informaticiens

LIÈGE université

Rapport de projet

MATH0488-1 : Eléments de processus stochastiques

Guillaume LERUTH (s191466)

Jérôme PIERRE (s190979)

Arnaud RENAUD (s193159)

3^e *bachelier ingénieur civil*

2021-2022

1 Première partie

1.1 Chaines de Markov

Soit la matrice de transition $\boldsymbol{Q}$:

$$\boldsymbol{Q} = \begin{pmatrix} 0 & 0.1 & 0.1 & 0.8 \\ 1 & 0 & 0 & 0 \\ 0.6 & 0 & 0.1 & 0.3 \\ 0.4 & 0.1 & 0.5 & 0 \end{pmatrix}$$

1. On veut calculer numériquement :
   — $\mathbb{P}(X_t = x)$ avec X_0 uniformément distribué parmi les états $\{1, 2, 3, 4\}$
   On obtient les $n = 30$ itérations suivantes :

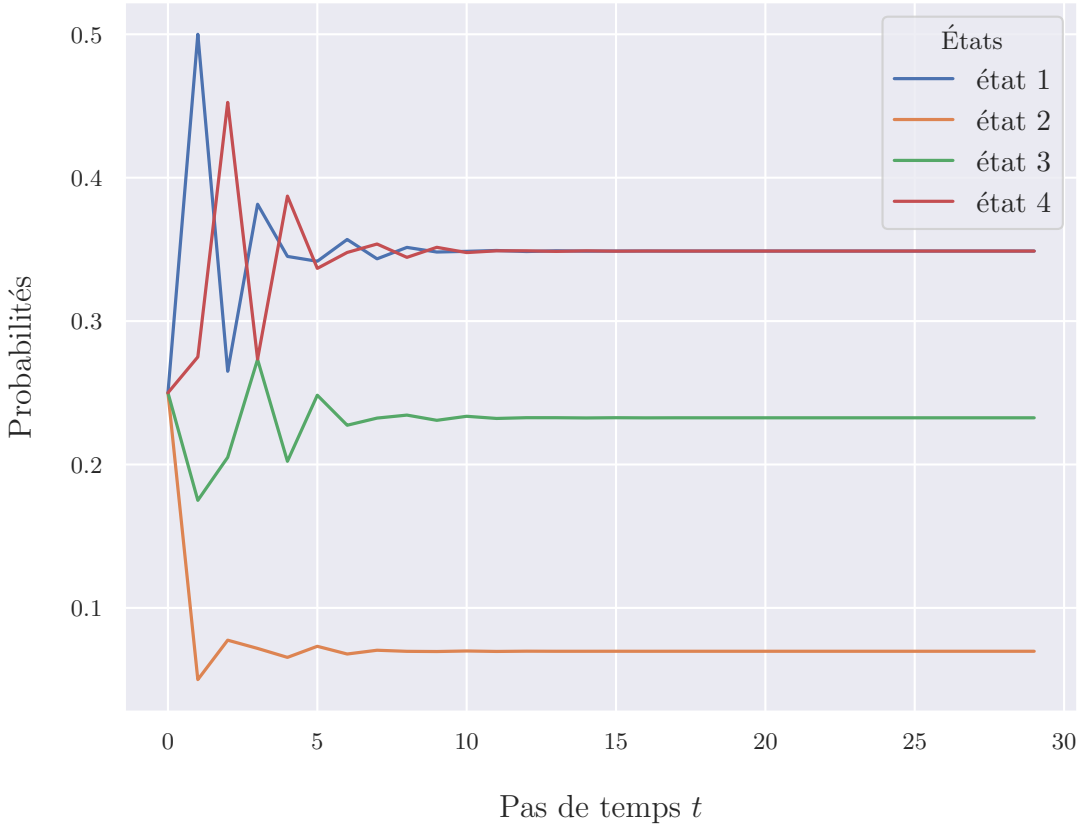


FIGURE 1 – Évolution de la distribution en partant d'une distribution initiale uniforme

A partir de la 10e itération, la distribution ne change sensiblement plus. On vient d'atteindre une distribution stationnaire. Ce résultat peut être expliqué par la théorie des chaînes de Markov. En effet, on ne fait que calculer la distribution $\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 \boldsymbol{Q}^t$ de la chaîne pour des valeurs croissantes de t. Or, on peut constater par inspection de la matrice de transition $\boldsymbol{Q}$ ou en représentant l'automate fini correspondant que la chaîne est irréductible. Dès lors, on sait que celle-ci possède une distribution stationnaire unique. Autrement dit, on a $\boldsymbol{\pi}_\infty = \lim_{t \to +\infty} \boldsymbol{\pi}_t$ avec la distribution $\boldsymbol{\pi}_\infty$ stationnaire, c'est-à-dire que $\boldsymbol{\pi}_\infty = \boldsymbol{\pi}_\infty \boldsymbol{Q}$. C'est donc pour ça que la distribution se stabilise autour d'une valeur fixe à mesure que t augmente.

— $\mathbb{P}(X_t = x)$ avec l'état initial $x_0 = 3$, autrement dit $\mathbb{P}(X_0 = 3) = 1$
On obtient numériquement :

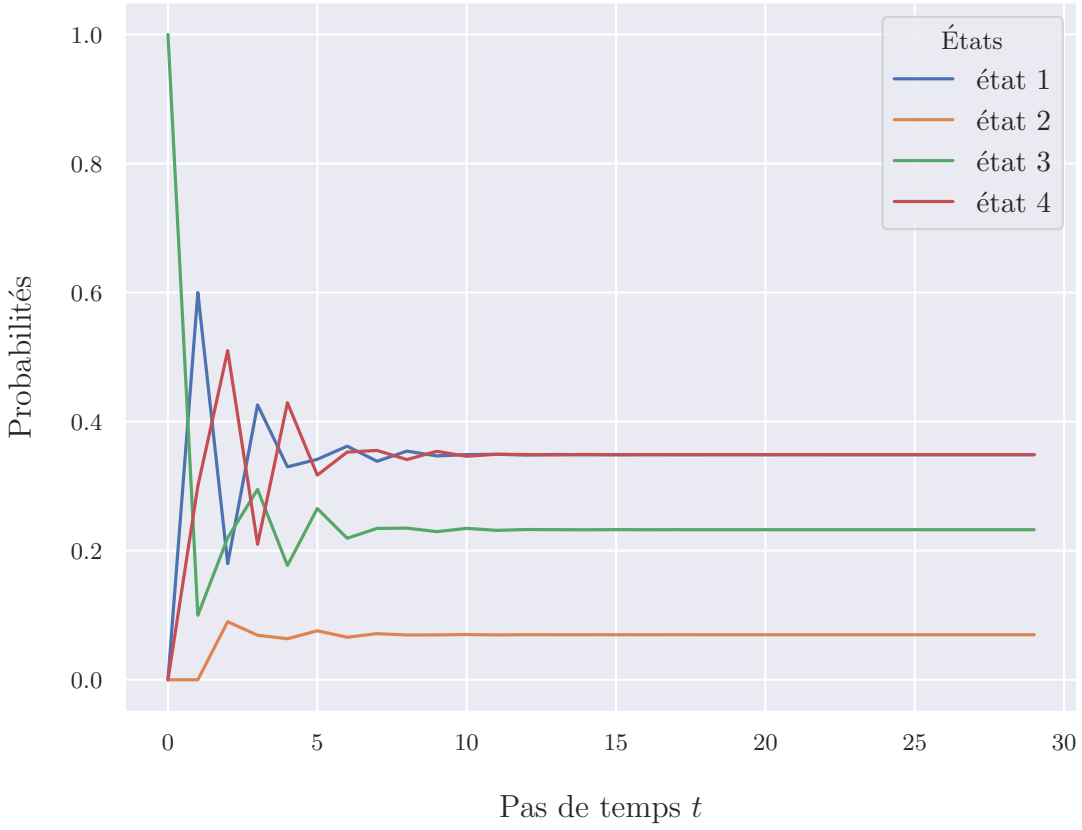


FIGURE 2 – Évolution de la distribution en partant de l'état initial 3

On remarque que l'on finit par atteindre la même distribution stationnaire que dans le cas précédent après une dizaine d'itérations. Ce résultat est également prévu par la théorie puisque, comme indiqué au point précédent, la distribution stationnaire $\boldsymbol{\pi}_\infty$ est unique et ne dépend par conséquent pas de la distribution initiale $\boldsymbol{\pi}_0$ choisie.

— $\boldsymbol{Q}^t$, la t^{eme} puissance de $\boldsymbol{Q}$
On obtient :

$$t = 2 \longrightarrow \boldsymbol{Q}^2 = \begin{pmatrix} 0.48 & 0.08 & 0.41 & 0.03 \\ 0 & 0.1 & 0.1 & 0.8 \\ 0.18 & 0.09 & 0.22 & 0.51 \\ 0.4 & 0.04 & 0.09 & 0.47 \end{pmatrix}$$

$$t = 3 \longrightarrow \boldsymbol{Q}^3 = \begin{pmatrix} 0.338 & 0.051 & 0.104 & 0.507 \\ 0.48 & 0.08 & 0.41 & 0.03 \\ 0.426 & 0.069 & 0.295 & 0.21 \\ 0.282 & 0.087 & 0.284 & 0.347 \end{pmatrix}$$

$$t = 4 \longrightarrow \boldsymbol{Q}^4 = \begin{pmatrix} 0.3162 & 0.0845 & 0.2977 & 0.3016 \\ 0.338 & 0.051 & 0.104 & 0.507 \\ 0.33 & 0.0636 & 0.1771 & 0.4293 \\ 0.3962 & 0.0629 & 0.2301 & 0.3108 \end{pmatrix}$$

$$\dots$$

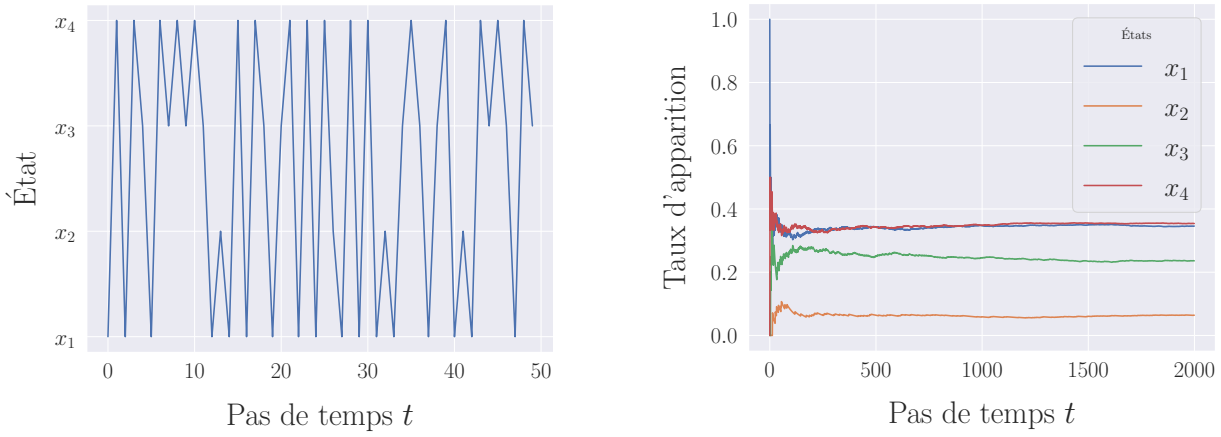
$$t = 15 \longrightarrow \boldsymbol{Q}^{15} = \begin{pmatrix} 0.349 & 0.07 & 0.232 & 0.349 \\ 0.349 & 0.07 & 0.232 & 0.349 \\ 0.349 & 0.07 & 0.232 & 0.349 \\ 0.349 & 0.07 & 0.232 & 0.349 \end{pmatrix}$$

Après un certain nombre d'itérations, on remarque que le fait de multiplier la matrice obtenue par $\boldsymbol{Q}$ donne la même matrice. De plus, les lignes de la matrice deviennent égales les unes aux autres. Cela est attendu puisqu'il faut que $\boldsymbol{Q}^t$ devienne stationnaire à mesure que t grandit pour que la distribution $\boldsymbol{\pi}_t = \boldsymbol{\pi}_0 \boldsymbol{Q}^t$ soit elle-même stationnaire. De plus, il faut bien que les lignes de $\lim_{t \to +\infty} \boldsymbol{Q}^t$ soient égales pour que $\boldsymbol{\pi}_\infty$ soit effectivement indépendante de la distribution de départ $\boldsymbol{\pi}_0$ choisie. On retrouve d'ailleurs bien la distribution $\boldsymbol{\pi}_\infty$ sur les lignes de $\boldsymbol{Q}^{15}$.

2. La distribution stationnaire obtenue numériquement au point précédent est

$$\boldsymbol{\pi}_\infty \approx \begin{pmatrix} 0.349 & 0.07 & 0.232 & 0.349 \end{pmatrix} \tag{1}$$

3. On initialise une réalisation de la chaîne en partant d'une distribution initiale arbitraire sur les états et en tirant un premier état au sort selon une loi multinomiale paramétrée par la distribution (on pourrait de manière équivalente choisir cet état initial arbitrairement). On doit ensuite prolonger cette réalisation en tenant compte de l'état dans lequel on se trouve à chaque itération. Autrement dit, on échantillonne selon la distribution conditionnelle $\mathbb{P}(X_t | x_{t-1})$ en utilisant la ligne correspondante de $\boldsymbol{Q}$. Une réalisation type de la chaîne est illustrée à la figure 3 ainsi que l'évolution du taux d'occupation de chaque état au cours du développement de la chaîne. On se rend ainsi compte que, passé quelques fluctuations initiales, les taux d'occupations convergent vers la distribution stationnaire $\boldsymbol{\pi}_\infty$ déterminée précédemment.



(a) Représentation du début de la réalisation

(b) Évolution des taux d'occupation

FIGURE 3 – Réalisation type de la chaîne de Markov

Les valeurs numériques des taux d'occupation obtenues pour des réalisations de longueur T différents sont rassemblées dans la table 1. On constate à nouveau que l'on se rapproche de $\boldsymbol{\pi}_\infty$ à mesure que T augmente.

T	x_1	x_2	x_3	x_4
5	0.4	0	0.4	0.2
20	0.4	0.15	0.1	0.35
50	0.38	0.06	0.16	0.4
100	0.36	0.06	0.23	0.35
500	0.346	0.06	0.248	0.346
1000	0.354	0.065	0.221	0.36
10000	0.349	0.071	0.231	0.349

TABLE 1 – Taux d'occupation des états pour différentes réalisations de chaîne

4. On peut ajouter à l'irréducibilité des points précédents que la chaîne est apériodique puisque l'on a toujours la possibilité de revenir à l'état actuel avec une probabilité non nulle en 2 ou 3 pas de temps. On vérifie que cela est vrai pour tous les états en examinant les diagonales des puissances $\boldsymbol{Q}^2$ et $\boldsymbol{Q}^3$ développées précédemment et qui ne comportent pas d'élément nul. Dès lors, on peut dire que le plus grand commun diviseur des nombres $n \in \mathbb{N}_0$ tels que $\boldsymbol{Q}^n(i,i) \neq 0$ pour tous les états i est 1, ce qui définit l'apériodicité. Puisque la chaîne est irréductible et apériodique, elle jouit des propriétés associées à l'ergodicité. Cela signifie que la distribution empirique sur les états échantillonnés à partir d'une seule réalisation de la chaîne tend vers la distribution stationnaire théorique de la chaîne d'autant plus que la longueur T de cette réalisation est importante. Une fois de plus, c'est bien ce que l'on observe en pratique comme illustré au point précédent.

   On peut également relier les résultats aux vecteurs propres et valeurs propres de la matrice. On se rend en fait compte que comme le prédit la théorie, la valeur propre maximale de la matrice est 1 et que le vecteur propre à gauche associé à cette dernière est en réalité la distribution stationnaire.

1.2 Méthode MCMC : analyse théorique dans le cas fini

1. On demande de démontrer que $\boldsymbol{\pi_0}$ est une distribution stationnaire de la chaîne de Markov dont $\boldsymbol{\pi_0}$ est la distribution initiale et $\boldsymbol{Q}$, la matrice de transition.
On sait également que l'on satisfait les équations de balance détaillée :

$$\forall i, j \in \{1, \dots, N\} \ : \ \pi_0(i)[Q]_{i,j} = \pi_0(j)[Q]_{j,i} \tag{2}$$

Puisque l'on veut démontrer que $\boldsymbol{\pi_0}$ est une distribution stationnaire de la chaîne de Markov, il faut vérifier l'égalité caractérisant une distribution stationnaire :

$$\pi_{\mathbf{0}} = \pi_{\mathbf{0}}\mathbf{Q} \iff \sum_i \pi_i [Q]_{i,j} = \pi_j \quad \forall i, j \in \{1, \dots, N\} \tag{3}$$

Or par les équations de balance détaillée, on obtient :

$$\sum_i \pi_i [Q]_{i,j} = \sum_i \pi_j [Q]_{j,i} = \pi_j \sum_i [Q]_{j,i} = \pi_j \tag{4}$$

car $\boldsymbol{Q}$ est une matrice stochastique [1] Ceci est bien la définition d'une distribution stationnaire. Si on veut assurer l'unicité, il faut que la chaîne soit irréductible.

2. Il nous faut désormais montrer que l'application de Metropolis-Hastings en remplaçant p_X par la fonction $f(x) = cp_X(x)$ où c est une constante génère une chaîne de Markov satisfaisant les équations de balance détaillée et ayant p_X comme distribution stationnaire.
Commençons par remarquer que α n'est rien d'autre qu'une probabilité. En effet, au vu de sa définition :

$$\alpha(x', x) = \min\left(1, \frac{f(x')}{f(x)} \frac{q(x|x')}{q(x'|x)}\right) \tag{5}$$

on voit que $\alpha \in [0, 1]$ et qu'il est utilisé comme une probabilité d'acceptation dans l'algorithme. En plus de cela, on remarque que l'on a un rapport de fonctions f, ce qui permet de simplifier les constantes. On obtient alors :

$$\alpha(x', x) = \min(1, c(x', x)) \text{ avec } c(x', x) = \frac{p_X(x')}{p_X(x)} \frac{q(x|x')}{q(x'|x)} \tag{6}$$

On se rend ici compte que les constantes n'ont aucune influence dans le développement.
De façon assez similaire, on peut interpréter $q(x'|x)$ comme étant la probabilité de proposition de x'. Autrement dit, il s'agit de la probabilité que x' soit proposé sachant que l'état précédent est x.
Ainsi, on peut écrire la probabilité de tomber sur l'état "a" à partir de l'état "b" de la manière suivante :

$$P(a|b) = q(a|b)\, \alpha(a, b)$$

A présent, nous devons vérifier que :

$$p_X(x)\, P(x'|x) = p_X(x')\, P(x|x') \tag{7}$$
$$\iff p_X(x)\, q(x'|x)\, \alpha(x', x) = p_X(x')\, q(x|x')\, \alpha(x, x') \tag{8}$$
$$\iff \frac{\alpha(x', x)}{\alpha(x, x')} = \frac{p_X(x')\, q(x|x')}{p_X(x)\, q(x'|x)} \tag{9}$$

1. La somme des éléments de chaque ligne vaut 1

Concentrons-nous davantage sur ce α. On remarque que l'on peut affirmer que soit $\alpha(x', x)$ soit $\alpha(x, x')$ est égal à 1. En effet, cela vient du fait que l'on ait :

$$c(x', x) = \frac{1}{c(x, x')}$$

Ainsi, si $c(x', x) < 1$, $c(x, x') > 1 \longrightarrow \alpha(x', x) = c(x', x), \alpha(x, x') = 1$.
De même, si $c(x, x') < 1$, $c(x', x) > 1 \longrightarrow \alpha(x, x') = c(x, x'), \alpha(x', x) = 1$.
Enfin, pour n'importe lequel de ces deux cas possibles, on arrive à l'égalité des deux membres. Ce qui assure donc ce que nous cherchions.
Des démonstrations alternatives existent évidemment. En guise d'exemple, en voici une autre, plus directe :

$$\begin{aligned}
p_X(x)\, P(x'|x) &= p_X(x)\, q(x'|x)\, \alpha(x', x) && (10)\\
&= p_X(x)\, q(x'|x)\, \min\left(1, \frac{p_X(x')}{p_X(x)}\, \frac{q(x|x')}{q(x'|x)}\right) && (11)\\
&= p_X(x)\, q(x'|x)\, \min\left(\frac{p_X(x)}{p_X(x)}\, \frac{q(x'|x)}{q(x'|x)}, \frac{p_X(x')}{p_X(x)}\, \frac{q(x|x')}{q(x'|x)}\right) && (12)\\
&= \min\left(p_X(x)\, q(x'|x), p_X(x')\, q(x|x')\right) && (13)\\
&= p_X(x')\, q(x|x')\, \min\left(\frac{p_X(x)}{p_X(x')}\, \frac{q(x'|x)}{q(x|x')}, \frac{p_X(x')}{p_X(x')}\, \frac{q(x|x')}{q(x|x')}\right) && (14)\\
&= p_X(x')\, q(x|x')\, \min\left(\frac{p_X(x)}{p_X(x')}\, \frac{q(x'|x)}{q(x|x')}, 1\right) && (15)\\
&= p_X(x')\, q(x|x')\, \alpha(x, x') && (16)\\
&= p_X(x')\, P(x|x') && (17)
\end{aligned}$$

On a prouvé que la distribution $p_X(x)$ respecte l'équation de balance détaillée vis-à-vis de la probabilité de transition $P(x'|x)$ correspondant à l'algorithme de Métropolis-Hastings. Au vu des résultats du point précédent, on a donc la garantie que la chaîne de Markov développée aura $p_X(x)$ pour distribution stationnaire.
Cependant, si on veut que l'algorithme de Métropolis-Hastings retourne la solution correcte, il faut que la distribution stationnaire soit unique, ce qui est garantit si on forme une chaine de Markov irréductible. (Comme nous sommes dans un cas réel, nous avons un nombre fini d'états. Cela force l'existence d'une distribution de probabilité stationnaire). Pour être encore plus précis, il nous faut avoir l'ergodicité de la chaîne de Markov si on veut que la fréquence d'apparition de chaque état au cours d'une seule réalisation de la chaîne converge vers la distribution $p_X(x)$. Dans ce cas, il faut encore que la chaine de Markov soit apériodique.

1.3 Méthode MCMC : illustration sur un exemple simple

Nous allons désormais appliquer l'algorithme de Metropolis-Hastings sur un exemple simple : une distribution binomiale $X \sim \text{Bin}(n = K = 10, p = 0.3)$. On a directement :

$$p_X(x = k) = C_K^k \, p^k \, (1 - p)^{K-k}$$

De plus, il est demandé d'utiliser la distribution de proposition qui suit :

$$q(y|x) = \begin{cases} r & \text{si } x = 0 \text{ et } y = 0 \\ 1 - r & \text{si } x = K \text{ et } y = K \\ r & \text{si } 0 < x \leq K \text{ et } y = x - 1 \\ 1 - r & \text{si } 0 \leq x < K \text{ et } y = x - 1 \\ 0 & \text{sinon} \end{cases} \qquad \text{avec } r \in]0; 1[\tag{18}$$

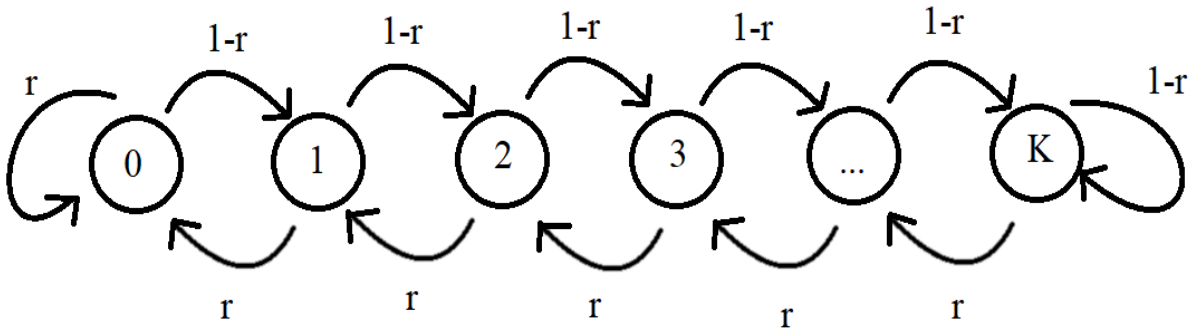
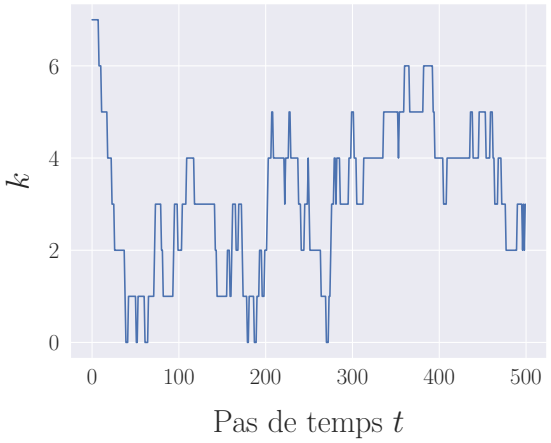


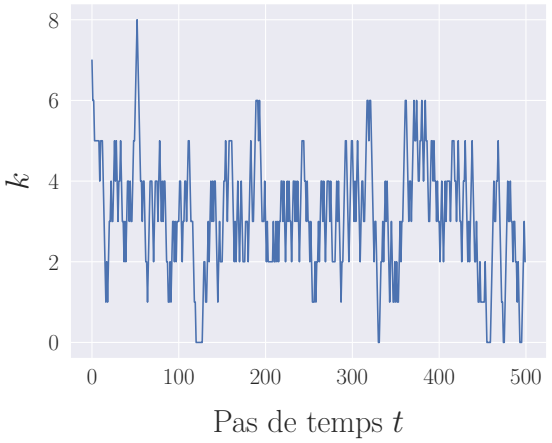
FIGURE 4 – Représentation de q(y|x)

1. On peut montrer assez facilement que dans ce cas, l'algorithme de Métropolis-Hastings permettra bien d'échantillonner la distribution binomiale. En effet, nous avons démontré plus haut que dès que $f(x) = cp_X(x)$, on prend bien des échantillons de cette distribution. Or, dans notre cas, $f(x) = p_X(x)$. Ainsi, il nous faut juste démontrer que $q(y|x)$ permet bien d'atteindre tous les états possibles. En regardant la Figure 4, il est évident que l'on peut atteindre tous les états [2] à partir de n'importe lequel. Nous avons une chaine irréductible. Il convient néanmoins de remarquer que plus r est grand, plus on aura des états proches de 0 proposés au fil du temps. De même, si r est petit, on aura plus souvent des états proches de K qui seront proposés.
2. Intéressons-nous désormais à l'évolution de la moyenne et de la variance des valeurs de la chaine de Markov ainsi formée. Comme nous venons de le souligner les valeurs de r vont biaiser l'apparition des prochains états. C'est pour cela qu'une situation symétrique (avec r=0.5 et donc avec aucun biais provenant de r) donne de biens meilleurs résultats.

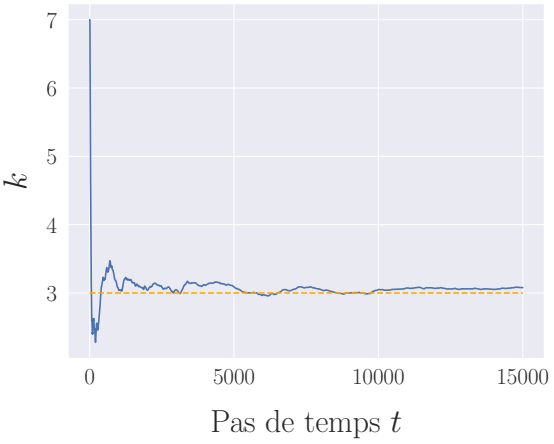
2. r est différent de 0 et 1 !



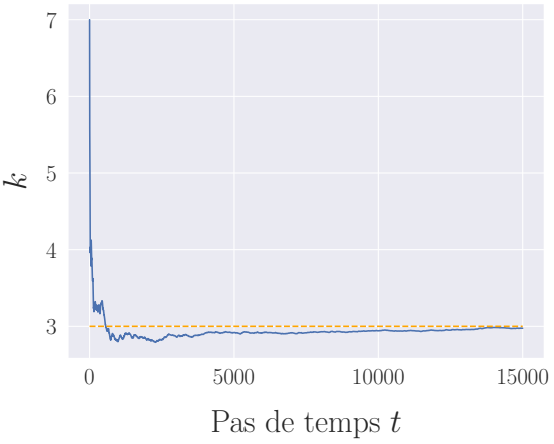
(a) Début de réalisation de la chaîne pour $r = 0.1$



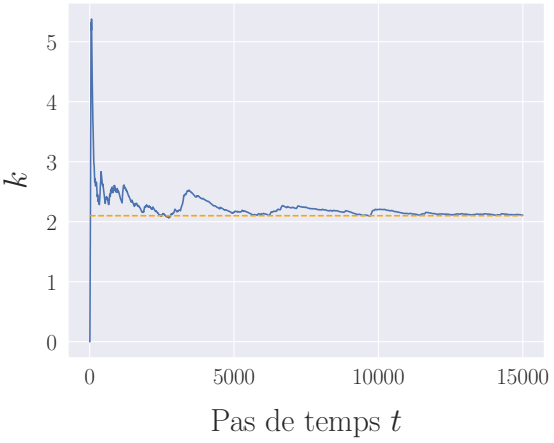
(b) Début de réalisation de la chaîne pour $r = 0.5$



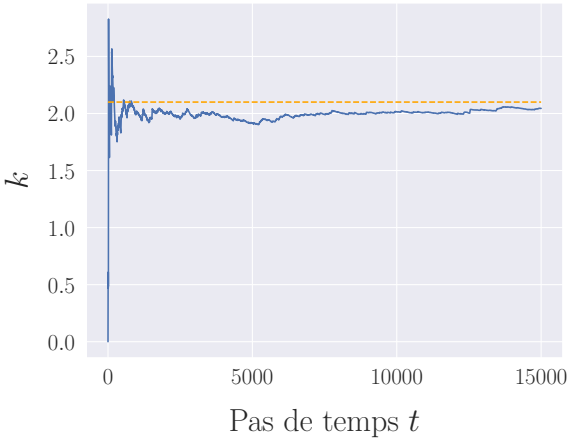
(c) Évolution de la moyenne pour $r = 0.1$



(d) Évolution de la moyenne pour $r = 0.5$



(e) Évolution de la variance pour $r = 0.1$



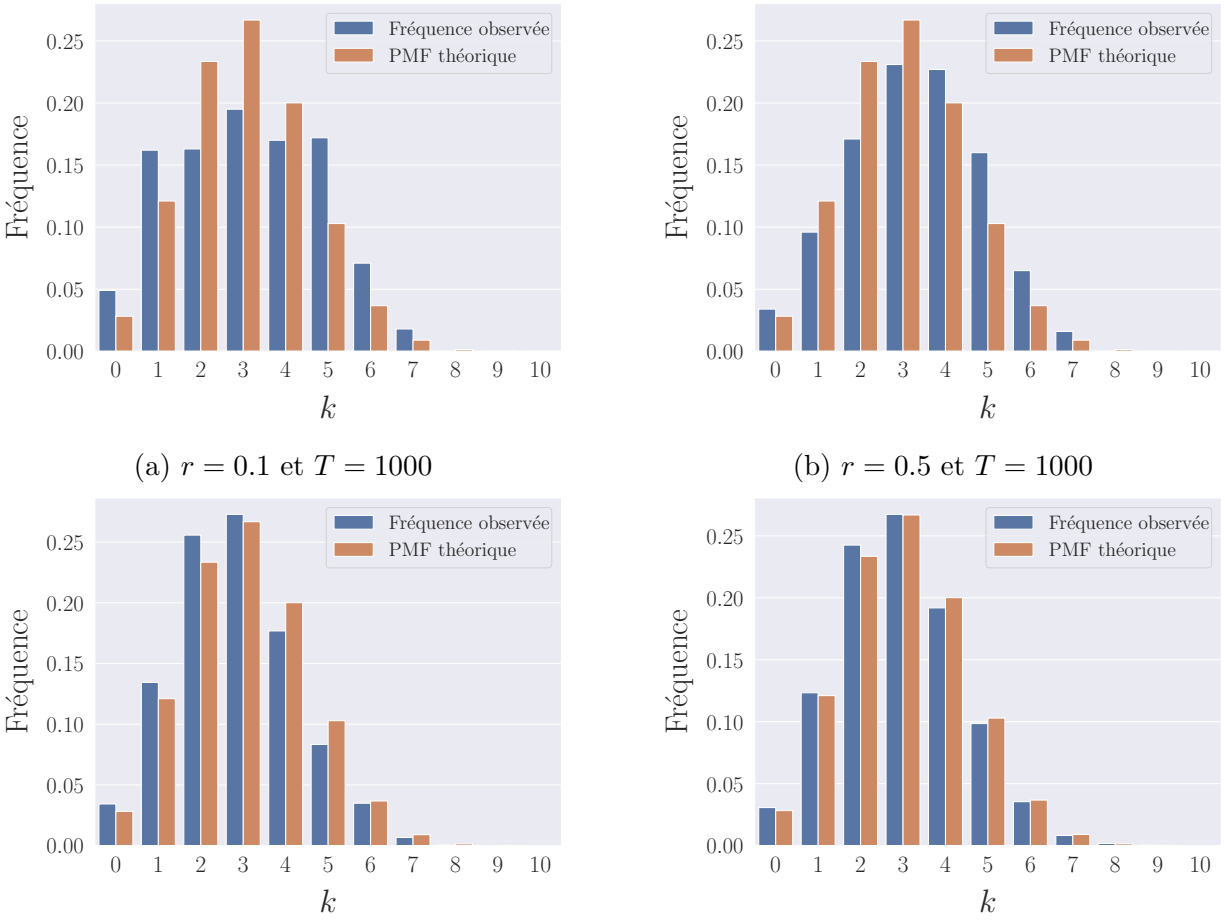
(f) Évolution de la variance pour $r = 0.5$

FIGURE 5 – Réalisations de la chaîne générées par l'algorithme de Métropolis-Hastings

Les valeurs théoriquement attendues pour la distribution binomiale sont $\mathrm{E}(X) = K \cdot p = 3$ pour la moyenne et $\mathrm{Var}(X) = K \cdot p \cdot (1 - p) = 2.1$ pour la variance. On observe bien que les valeurs de la moyenne et de la variance échantillonnée à partir de la chaîne de Markov générée par l'algorithme de Métropolis-Hastings tendent bien vers ces valeurs théoriques à mesure que la longueur de la chaîne augmente. On note également, comme attendu, que la convergence est plus efficace avec une valeur de r "neutre" comme $r = 0.5$.
Qui plus est, on observe dans les réalisations que pour $r = 0.1$, on reste un certain temps dans les maxima. Ceci est simplement dû au fait que notre distribution de proposition propose

bien plus de valeurs plus grandes dans ce cas. Contrairement à cela, dans le cas $r = 0.5$, on observe des maxima assez brefs et une situation beaucoup plus symétrique.

3. Nous allons désormais tracer les histogrammes des fréquences d'apparitions des différentes valeurs de k correspondants. En parallèle, on affiche les valeurs attendues de la PMF théorique de la distribution binomiale.



(a) $r = 0.1$ et $T = 1000$

(b) $r = 0.5$ et $T = 1000$

(c) $r = 0.1$ et $T = 20000$

(d) $r = 0.5$ et $T = 20000$

FIGURE 6 – Histogrammes des fréquences d'apparition des valeurs k dans la chaîne

On constate qu'avec une chaîne de longueur $T = 1000$, l'histogramme des fréquences observées pour $r = 0.1$ ne ressemble que très vaguement à la PMF d'une loi binomiale. L'histogramme pour $r = 0.5$, quant à lui, se rapproche de l'allure attendue même s'il est encore loin des valeurs théoriques. En augmentant la longueur des chaînes jusqu'à $T = 20000$, les deux histogrammes se rapprochent sensiblement des valeurs théoriques attendues, $r = 0.5$ restant plus proche.

Tout comme avancé précédemment, on remarque que fixer $r = 0.5$ permet d'être symétrique par rapport à la moyenne alors que ce n'est pas le cas pour $r = 0.1$. On observe en effet dans notre histogramme pour $r = 0.1$, une légère surestimation des valeurs inférieures à la moyenne (pour $T = 20000$).

2 Deuxième partie

2.1 Étude théorique

1. Notons $\mathbf{x}$, le vecteur de taille N associant à chaque noeud sa communauté et $\mathbf{G}$, la matrice d'adjacence du graphe. Dans la suite, on note aussi N, le nombre de noeuds ; K, le nombre de communautés ; p, le vecteur de dimension K associant à chaque communauté sa probabilité et W, un matrice de dimension $K \times K$ dont les éléments $W_{i,j}$ définissent la probabilité de l'existence d'une arrête entre deux noeuds de la communauté i et j.
   En utilisant la loi de Bayes, on a :

$$P(\mathbf{x}|\mathbf{G}) = \frac{P(\mathbf{G}|\mathbf{x}) \cdot P(\mathbf{x})}{P(\mathbf{G})} \tag{19}$$

   - Dans le modèle SBM(N, K, p, W), on a que :

$$P(\mathbf{G}|\mathbf{x}) = \sqrt{\prod_{i}^{N}\prod_{j}^{N} M(i,j)^{1-\delta_{i,j}}} \text{ avec } M(i,j) = \begin{cases} M(i,j) = W_{x(i),x(j)} \text{ si } G_{i,j} = 1 \\ M(i,j) = 1 - W_{x(i),x(j)} \text{ si } G_{i,j} = 0 \end{cases} \tag{20}$$

     Ceci peut encore se réécrire :

$$P(\mathbf{G}|\mathbf{x}) = \prod_{1 \leq i < j \leq N} W_{x(i),x(j)}^{G_{i,j}} \cdot (1 - W_{x(i),x(j)})^{1-G_{i,j}} \tag{21}$$

     Si on se trouve dans le cas du modèle simplifié SBM(N, K, p, A, B), il suffit de remplacer $W_{x(i),x(j)}$ par A si $x(i) = x(j)$ et par B sinon, ce qui donne :

$$P(\mathbf{G}|\mathbf{x}) = \prod_{1 \leq i < j \leq N} \left(A^{\delta_{x(i),x(j)}} B^{1-\delta_{x(i),x(j)}}\right)^{G_{i,j}} \cdot \left((1-A)^{\delta_{x(i),x(j)}} (1-B)^{1-\delta_{x(i),x(j)}}\right)^{1-G_{i,j}} \tag{22}$$

   - Comme on suppose que l'on connaît p, on trouve directement que $P(\mathbf{x})$ peut s'écrire comme :

$$P(\mathbf{x}) = \prod_{i=1}^{N} p(x(i)) \tag{23}$$

   - Enfin, $P(\mathbf{G})$ peut s'écrire grâce à la loi des probabilités totales. Si on note $\mathbf{x}_i$ une possibilité des vecteurs $\mathbf{x}$ possibles, on a :

$$P(\mathbf{G}) = \sum_{\mathbf{x_i}} P(\mathbf{G}|\mathbf{x_i}) \cdot P(\mathbf{x_i}) \tag{24}$$

     La manière de calculer $P(\mathbf{G}|\mathbf{x_i})$ et $P(\mathbf{x_i})$ viennent d'être détaillées ci-dessus.

2. Afin de comprendre pourquoi le calcul de P($\mathbf{x}|\mathbf{G}$) pose problème, faisons une analyse asymptotique de la complexité des trois probabilités à calculer.

   - Complexité de $P(\mathbf{G}|\mathbf{x})$
     On remarque de suite qu'il y a un double produit (sur N) dans notre calcul. On a donc que la complexité pour trouver $P(\mathbf{G}|\mathbf{x})$: $A \in O(N^2)$.
   - Complexité de $P(\mathbf{x})$
     On a un simple produit sur N et ainsi, la complexité pour calculer $P(\mathbf{x})$: $B \in O(N)$

- Complexité de $P(\mathbf{G})$

  Il faut déjà calculer une somme sur tous les vecteurs $\mathbf{x}$ possibles. Il y en a K^N. On comprend que cela n'est pas possible. Ce terme qui a une complexité de $C \in O(K^N \cdot A \cdot B) = O(N^3 K^N)$ prend beaucoup trop de temps en pratique.
  Cela n'est cependant pas un problème pour la méthode de Métropolis-Hasting. En effet, la probabilité $P(\mathbf{G})$ n'est rien d'autre qu'une constante dans notre calcul de $p_X = P(\mathbf{x}|\mathbf{G})$ et on a démontré dans la section 1.2 que si on multiplie p_X par une constante quelconque, l'algorithme marche encore. Ainsi, on peut juste laisser tomber le calculer de $P(\mathbf{G})$.

3. On propose d'utiliser dans un premier temps q_s comme distribution de proposition dans l'algorithme. Cette distribution de proposition consiste à choisir aléatoirement un noeud du graphe et à changer la communauté aléatoirement. On a donc

$$q_s(x|y) = \frac{1}{N}\frac{1}{K} \, \forall x, y \tag{25}$$

   Nous savons que l'algorithme de Métropolis-Hasting échantillonnera la distribution recherchée tant que p_X est la distribution recherchée à une constante près et que q_s permet d'accéder à tous les états. Ici, c'est trivialement le cas ! En effet, la distribution de proposition donne la possibilité de modifier tous les noeuds et d'attribuer toutes les communautés (de façon aléatoire).

4. On a directement que :

$$\alpha = \min\left\{1, \frac{p_X(y^{(t)})}{p_X(x^{(t-1)})} \frac{q_s(x^{(t-1)}|y^{(t)})}{q_s(y^{(t)}|x^{(t-1)})}\right\} \tag{26}$$

$$= \min\left\{1, \frac{P(\mathbf{y}^{(t)}|\mathbf{G})}{P(\mathbf{x}^{(t-1)}|\mathbf{G})} \frac{N\,K}{N\,K}\right\} \tag{27}$$

$$= \min\left\{1, \frac{P(\mathbf{G}|\mathbf{y}^{(t)})P(\mathbf{y}^{(t)})}{P(\mathbf{G}|\mathbf{x}^{(t-1)})P(\mathbf{x}^{(t-1)})}\right\} \tag{28}$$

   Or, on sait qu'il n'y a que le noeud k qui change entre $y^{(t)}$ et $x^{(t-1)}$, c'est à dire que la seule différence est $x^{(t-1)}(k) \Longrightarrow y^{(t)}(k)$. Ainsi, sur base de l'équation (21), on a directement que :

$$\frac{P(\mathbf{G}|\mathbf{y}^{(t)})}{P(\mathbf{G}|\mathbf{x}^{(t-1)})} = \prod_{i=1}^{N} \frac{W_{y^{(t)}(i),y^{(t)}(k)}^{G_{i,k}} \cdot (1 - W_{y^{(t)}(i),y^{(t)}(k)})^{1-G_{i,k}}}{W_{x^{(t-1)}(i),x^{(t-1)}(k)}^{G_{i,k}} \cdot (1 - W_{x^{(t-1)}(i),x^{(t-1)}(k)})^{1-G_{i,k}}} \text{ avec } i \neq k \tag{29}$$

   Or, puisque nous nous trouvons dans le modèle simplifié, il n'y aura que les facteurs correspondant à l'ancienne communauté ($A \longrightarrow B$) et à la nouvelle ($B \longrightarrow A$) qui changent. Utilisons la notation suivante :

$$\begin{cases} X_1 : \text{le nombre de connections entre le noeud k et les noeuds de la communaute } \mathrm{x^{(t-1)}(k)} \\ X_2 : \text{le nombre de connections entre le noeud k et les noeuds de la communaute } \mathrm{y^{(t)}(k)} \\ n_1 : \text{ le nombre de noeuds de la communaute } \mathrm{x^{(t-1)}(k)} \text{ au temps t} - 1 \\ n_2 : \text{ le nombre de noeuds de la communaute } \mathrm{y^{(t)}(k)} \text{ au temps t} \end{cases}$$

   on peut écrire que :

$$\frac{P(\mathbf{G}|\mathbf{y}^{(t)})}{P(\mathbf{G}|\mathbf{x}^{(t-1)})} = \frac{A^{X_2} \cdot (1-A)^{n_2-X_2} \cdot B^{X_1} \cdot (1-B)^{n_1-X_1}}{A^{X_1} \cdot (1-A)^{n_1-X_1} \cdot B^{X_2} \cdot (1-B)^{n_2-X_2}} \tag{30}$$

   Soit, si on note $\Delta = X_2 - X_1$ et $\Delta n = n_2 - n_1$:

$$\frac{P(\mathbf{G}|\mathbf{y}^{(t)})}{P(\mathbf{G}|\mathbf{x}^{(t-1)})} = \left(\frac{A}{B}\right)^{\Delta} \cdot \left(\frac{1-A}{1-B}\right)^{\Delta n - \Delta} \tag{31}$$

En ce qui concerne l'autre partie de l'expression, les choses sont plus simples :

$$\frac{P(\mathbf{y}^{(t)})}{P(\mathbf{x}^{(t-1)})} = \frac{\prod_{i=1}^{N} p(y^{(t)}(i))}{\prod_{i=1}^{N} p(x^{(t-1)}(i))} = \frac{p(y^{(t)}(k))}{p(x^{(t-1)}(k))} \tag{32}$$

On vient de montrer que α pouvait être écrit de la façon suivante :

$$\alpha = \min\left\{1, \left(\frac{A}{B}\right)^{\Delta} \cdot \left(\frac{1-A}{1-B}\right)^{\Delta n - \Delta} \cdot \frac{p(y^{(t)}(k))}{p(x^{(t-1)}(k))}\right\} \tag{33}$$

2.2 Analyse expérimentale

Il nous faut désormais mettre en évidence la transition de phase à partir de laquelle il est possible d'identifier correctement les différentes communautés (dans le cas K=2). Pour ce faire, on calcule la valeur suivante en fonction du rapport b/a :

$$E_{(x^*,G)}\{E_{P(x|G)}\{A(x^*,x)\}\} \tag{34}$$

Pour calculer cette valeur, on génère plusieurs graphes sur base des mêmes paramètres. On exécute alors l'algorithme de Metropolis-Hastings sur chacun de ces graphes afin d'obtenir des réalisations de chaînes. On ne garde alors que la queue de chaque réalisation pour être sûr d'avoir convergé. On peut alors calculer la moyenne de la concordance des vecteurs au sein de chaque réalisation vis-à-vis du vrai vecteur de communauté. On peut enfin calculer la moyenne de ces moyennes sur les différents graphes.

1. Commençons par définir $A = \frac{a}{N}$ et $B = \frac{b}{N}$. Si on représente la valeur définie ci-dessus en fonction du rapport $\frac{b}{a}$, on obtient la figure 7

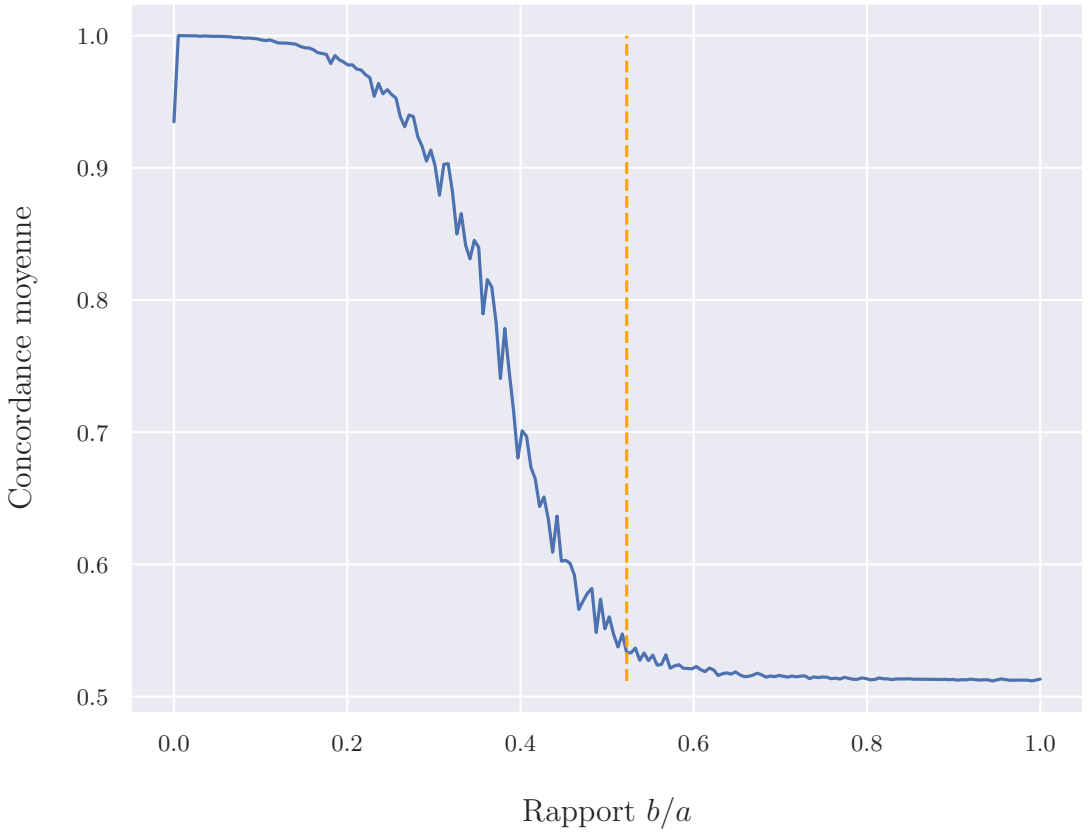


FIGURE 7 – Évolution de la concordance moyenne en fonction de $\frac{b}{a}$, pour $N = 5000$, $\frac{a+b}{2} = 10$

On remarque de nombreuses fluctuations dans la courbe. Celles-ci sont, pour la plupart, dues au fait que l'on réalise des moyennes sur seulement une trentaine de graphes. Néanmoins, la grande diminution pour des valeurs au voisinage de 0 semblent avoir une origine différente. Nous avons en effet répété plusieurs fois l'algorithme pour des graphes différents mais toujours avec un rapport de b sur a de 0.00001. On assiste assez souvent à une convergence vers 1 de la concordance. Il arrive néanmoins que la concordance finisse par tendre vers 0.5, parfois après avoir légèrement augmenter.

Nous interprétons cela comme une conséquence du faible degré moyen des différents noeuds du graphe. Voici ce qu'il se passe selon nous. A un moment donné l'algorithme se trompe et met un noeud dans la mauvaise communauté. Néanmoins le faible nombre de lien entre les deux communautés l'empêche de se rendre compte de son erreur et il peut alors mettre de plus en plus d'éléments dans la mauvaise communauté jusqu'à ne plus s'y retrouver.

2. La figure 7 vérifie bel et bien que l'on peut détecter les communautés de manière plus efficace qu'une méthode purement aléatoire (concordance supérieure à 50%) si :

$$(a-b)^2 > 2 \cdot (a+b) \tag{35}$$

En effet si on fixe le degré moyen à une valeur l fixée, on peut trouver la limite que l'on obtient expérimentalement :

$$\begin{cases} (a-b)^2 > 2 \cdot (a+b) \\ a+b = 2l \end{cases} \tag{36}$$

$$\iff \begin{cases} 4\,a^2 - 8\,a\,l + 4\,l\,(l-1) > 0 \\ b = 2l - a \end{cases} \tag{37}$$

Pour vérifier la première inégalité, il faut que $a \in R^+ \setminus \left[l - \sqrt{l}; l + \sqrt{l}\right]$.
On a dès lors que :

(a) Si $a \in \left]0; l - \sqrt{l}\right[\implies b \in \left]l + \sqrt{l}; 2l\right[$

(b) Si $a \in \left]l + \sqrt{l}; +\infty\right[\implies b \in \left]0; l - \sqrt{l}\right[$

Remarquons que dans notre modèle, on suppose que $A > B \implies a > b$, il faut donc rejeter la première possibilité.
Ainsi, la théorie prédit que nous aurons des résultats supérieurs à une attribution aléatoire de communautés si :

$$\frac{b}{a} \in \left]0, \frac{l - \sqrt{l}}{l + \sqrt{l}}\right[\tag{38}$$

Pour vérifier cela empiriquement dans le cas où K=2, il faut regarder l'intervalle dans lequel nous sommes significativement supérieurs à 50% [3]. Le graphe de la Figure 7 montre que l'expérience a un comportement vérifiant ce que la théorie prédit. Nous sommes en effet largement au-dessus de 50% dans l'intervalle théorique $(]0; 0.5195[)$.

3. Car le graphe est réalisé pour $\mathbf{p} = [0.5, 0.5]$

3 Notes sur le code

3.1 Première optimisation

Il existe évidemment plusieurs manières de coder l'algorithme de Métropolis-Hasting pour le problème de détection de communautés. La principale source de temps de calcul se trouve dans le calcul de α. La méthode la plus simple serait de directement implémenter la formule 21 ou 29 pour le premier facteur. Nous avons toutefois montré qu'il était possible de tirer avantage du fait qu'un seul noeud changeait de communauté par itération pour obtenir l'équation 31.

Celle-ci semble plus efficace mais demande de connaître le nombre de liens entre le noeud qui va possiblement changer de communauté et les différents noeuds composant son ancienne et sa nouvelle communauté ainsi que le nombre de noeuds présents dans celles-ci. Une lecture de la matrice G permettrait de réaliser cela facilement mais nous restons alors avec $O(N^2)$ opérations par itération de Métropolis-Hasting, ce qui n'est pas très intéressant. Voyant cela, nous avons cherché une manière efficace d'obtenir ces informations à chaque itération tout en restant assez général pour que l'algorithme puisse être utilisé dans des cas hypothétiques où nous aurions plus de 2 communautés [4]. Nous avons alors fait le pari qu'avoir une phase d'initialisation plus importante était payant tant que nous avions des itérations très rapides par la suite. Il s'agit de la philosophie principale du code.

Dans cette optique, nous avons décidé de créer une table de taille $N \cdot K$ dont l'élément (i, j) correspond au nombre de liaisons entre le noeud i et la communauté x(j).

Néanmoins, cette table ne donne en O(1) qu'une partie des informations que l'on souhaite. Il nous faut encore trouver une manière efficace d'accéder au nombre de noeuds de chaque communauté. Pour cela aussi, nous allons créer un vecteur de longueur K au début de notre initialisation. Cela permettra d'y accéder en O(1) par la suite. On obtient alors que le calcul de α se fait en O(1).

Maintenant que l'idée principale est posée, il faut souligner quelques subtilités. À chaque fois qu'un noeud change de communauté, il faut mettre à jour le vecteur de taille K. Pour ce faire, dès que le noeud a changé de communauté, il faut ajouter 1 à la nouvelle communauté et retirer 1 à l'ancienne. Au passage, cela signifie également que le calcul de α se fait à l'aide du vecteur correspondant à la fin de l'itération précédente (temps t-1).

Si le noeud est accepté dans sa nouvelle communauté, tous les noeuds auxquels il est lié verront le nombre de liens vers chaque communauté changer. En effet, le nombre de lien vers l'ancienne communauté du noeud ayant changé diminuera d'une unité alors que le nombre correspondant à la nouvelle communauté augmentera d'une unité. Le moyen le plus simple serait de parcourir tous les noeuds pour voir si il faut les modifier mais cela serait sous-optimal. A la place, nous formons lors de l'initialisation du code une paire de vecteurs qui permettent, pour chaque noeud, d'accéder directement à la liste des noeuds auquel il est lié. En effet, on remarque que les seuls noeuds concernés par une mise-à-jour du nombre de liaison envers les différentes communautés sont ceux liés à celui dont la communauté a changé.

Le lecteur attentif fera, à juste titre, la remarque que cela nous laisse un nombre d'opérations en O(N) pour chaque itération. Néanmoins, N constitue une borne supérieure rarement approchée puisqu'on ne réalise plus qu'un nombre d'opérations égal au degré du noeud changé par itération du Métropolis-Hasting. Qui plus est, dans le cadre de ce projet (ainsi que dans l'extrême majorité des cas réels), on se trouve dans le cas où le degré des sommets est largement inférieur au nombre de noeuds. Il s'agit en fait d'une grande amélioration !

4. Bien que nous ne rendions que le code optimisé pour le cas K=2, l'algorithme est facilement généralisable.

Intuitivement, cette méthode va faire varier notre temps de calcul de façon assez particulière. Évidemment, plus le nombre de noeuds augmente, plus il y aura d'arêtes entre les noeuds et plus le temps de calcul augmentera. Le nombre de noeuds ne semble cependant ne plus être la seule variable du temps de calcul. Par exemple, si ce dernier est fixé mais que le nombre de communautés augmente (imaginons qu'elles soient équiprobables), il y aura de moins en moins d'arêtes car a>b et ainsi le temps de calcul par itération diminuera. De même, la probabilité de chaque communauté influencera le temps de calcul. Enfin, on peut également noter que les valeurs de a et b font varier le temps de calcul. De fait, plus ces valeurs sont petites, plus le degré des noeuds sera petit en moyenne et ainsi plus, l'algorithme sera rapide.

3.2 Autres optimisations

La deuxième partie du calcul de α demande d'obtenir le quotient $\frac{P(\mathbf{y}^{(t)})}{P(\mathbf{x}^{t-1})}$. Nous avons déjà obtenu l'équation 32. Celle-ci montre trivialement que l'on peut calculer ce quotient en O(1) à l'aide du vecteur **p**.

Enfin, la dernière partie du alpha correspond au quotient $\frac{q_s(x^{(t-1)}|y^{(t)})}{q_s(y^{(t)}|x^{(t-1)})}$. Comme expliqué précédemment, le q_s choisi de base est un choix totalement aléatoire. Par symétrie du problème, on remarque donc que le quotient se simplifie totalement. Nous parlerons davantage des q_s envisagés dans la section suivante.

En ce qui concerne l'implémentation en Python, nous avons décidé de calculer autant de choses que possible dès l'initialisation. En effet, nous avons par exemple constaté qu'il est plus efficace avec Numpy d'initialiser un vecteur de valeurs aléatoires et d'utiliser celles-ci au compte goutte que de générer une seule valeur aléatoire à chaque itération de l'algorithme.

Pour terminer, notre algorithme renvoie simplement le dernier élément vers lequel il a convergé. En effet, on remarque dans la Figure 8 que lorsqu'on laisse l'algorithme tourner assez longtemps, ce dernier converge vers une valeur approximant de façon satisfaisante [5] le vecteur maximisant la probabilité $P(x^{(t)}|G)$.

5. Pour cela, il est préférable que a et b ne soient pas trop petits, sinon les erreurs peuvent coûter cher comme expliqué dans la section 2.2

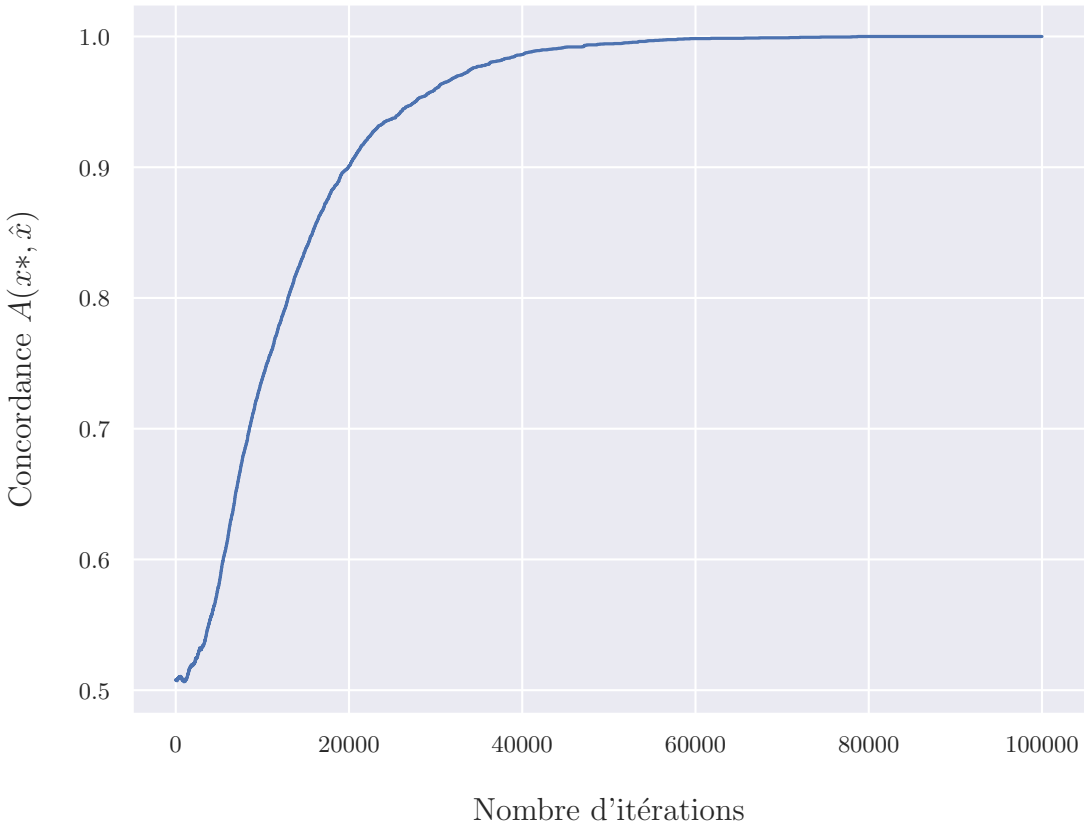


FIGURE 8 – Évolution de la concordance en fonction des itérations pour un graphe de $N = 5000$ noeuds, $A = 0.04$ et $B = 0.005$

Néanmoins, on aurait pu également essayer de détecter le vecteur $x^{(t)}$ maximisant la probabilité $P(x^{(t)}|G)$. Ceci permet notamment de s'émanciper des effets néfastes de maxima locaux. En effet, en gardant en mémoire ces vecteurs et en relançant l'algorithme plusieurs fois, cette méthode permet de déterminer un maximum plus proche du maximum global (si ce n'est ce dernier exactement !).

Voilà, tout est dit pour la base de l'algorithme que nous avons utilisé lors de la compétition liée à ce projet. Son avantage est sa grande rapidité. En effet, pour le graphe de la compétition, notre algorithme était capable de réaliser 1 million d'itérations toutes les 5 ou 6 secondes (l'initialisation ne prenant aussi que quelques secondes). En moins de 2 minutes, il avait alors déjà largement convergé et le résultat avait une concordance d'environ 93.55 %.

3.3 Distribution de proposition

Nous allons désormais discuter des différentes probabilités de distribution q_s que l'on aurait pu mettre au point dans notre code. Nous en avions trouvé plusieurs.

La plus simple que nous avions trouvée consistait à modifier le noeud qui risquait de se faire modifier aléatoirement selon les probabilités liées à chaque communauté. Cela permet de simplifier le rapport de l'équation 32. Néanmoins, cela ne change pas grand chose en pratique.

Nous avons également considéré une troisième q_s possible. Il s'agit d'une piste que nous conseillerions de suivre dans l'éventualité d'une recherche d'un algorithme encore plus optimisé.

Pour l'expliquer, recentrons-nous sur ce qui se passe lorsqu'on propose un nouveau vecteur $y^{(t)}$:

1. D'abord, on propose un noeud à possiblement modifier de façon aléatoire.
2. Ensuite, on choisit de façon aléatoire la nouvelle communauté possible du noeud en question (ou on lui attribue cette dernière en fonction des probabilités des communautés).

Nous proposerions d'utiliser une donnée qui n'a pas été utilisée jusqu'alors : le degré théorique de chaque noeud. En effet, dans un modèle SBM, on sait que le degré théorique d d'un noeud de la communauté z est :

$$d_z = N \cdot p_z \cdot A + \sum_{i \neq z} N \cdot p_i \cdot B = N \cdot (A \cdot p_z + B \cdot (1 - p_z)) \tag{39}$$

Cette formule dit simplement que le degré d'un noeud appartenant à la véritable communauté z peut être calculé en additionnant les arrêtes partant de ce noeud vers les autres noeuds de cette même communauté et ceux des autres communautés. Cette formule montre aussi que dans le cas K=2 et p=[0.5 ; 0.5] (ou plus généralement lorsque les communautés sont équiprobables), le degré n'est plus une caractéristique des communautés. Il s'agit d'une autre raison pour laquelle nous n'avons pas implémenté cette technique (voir plus loin)...

On pourrait alors garder en mémoire ces degrés théoriques dans un vecteur. Il suffirait alors de créer une fonction de coût W_1 qui associerait à chaque noeud, dans une certaine communauté, l'écart en valeur absolue entre le degré réel du noeud et le degré théorique. Il nous faudrait ainsi plus que sélectionner le noeud ayant la pire fonction de coût en priorité. En plus de cela, cette méthode permettrait de décider dans quelle communauté doit aller ce noeud en essayant de minimiser cette fonction de coût W_1.

Le principal problème avec cette technique est qu'elle est basée sur le modèle de façon exagérée. Dans la vie réelle, nous avons souvent des différences par rapport au modèle étudié. Afin de prendre en compte des cas réels, il faut alors faire un compromis entre cette méthode "déterministe" et l'exploration aléatoire. Une possibilité consiste à tirer une probabilité uniformément dans $[0; 1]$. Si celle-ci est supérieure à une valeur β (correctement choisie au départ [6]), on fait comme auparavant avec des choix aléatoires. Une autre technique serait de passer à de l'aléatoire dès qu'on considère qu'on a réalisé assez d'itérations, par exemple dès qu'on remarque qu'on a "convergé" (cela peut se faire en calculant la concordance entre des vecteurs à différents pas de temps).

Pour terminer ces propositions pour des améliorations futurs, on remarque qu'ajouter une fonction de coût W_2 donnant un score à chaque communauté serait aussi intéressant. Cela nous permettrait d'essayer d'améliorer la communauté la moins proche de celle théorique avant les autres. Une telle fonction de coût devrait d'une part, prendre en compte la différence du nombre de noeuds dans la communauté i par rapport au nombre théorique $N \cdot p_i$ et d'autre part, évaluer à quel point les noeuds lui étant attribués au moment donné lui correspondent véritablement (en faisant la somme des fonctions de coût W_1 des noeuds la composant par exemple).

Les fonctions de coût W_1 et W_2 proposées ne sont évidemment que des exemples. L'essentiel est qu'elles suivent la même idée tout en étant aussi rapides que possible pour l'algorithme.

6. On peut également imaginer une valeur β variant dans le temps, en diminuant par exemple.

En bref, la partie correspondant à la distribution de proposition (avant qu'on passe possiblement en exploration aléatoire uniquement) serait donc dans les grandes lignes comme ceci :

Si la probabilité tirée est inférieure à β :

1. Déterminer la "pire" communauté avec W_2
2. Déterminer le "pire" noeud de la "pire" communauté à l'aide de W_1
3. Mettre le noeud en question dans la communauté lui correspondant le mieux

Sinon : Tirage du noeud aléatoire et nouvelle communauté aléatoire.

A titre d'information, remarquons que l'on pourrait utiliser des raisonnements similaires pour fixer l'état initial $x^{(0)}$ de Métropolis-Hasting.

Néanmoins, au vu de la rapidité de notre code et de la vitesse de la convergence (jugée satisfaisante), nous avons considéré que de telles améliorations n'étaient pas nécessaires, qu'elles allaient complexifier le code pour de possibles améliorations superflues dans le cadre de ce travail.