

# Processus stochastiques : cryptanalyse

Stegen Thomas s154315  
Adrien Minne s154340  
Delaunoy Arnaud s153059

# 1 Première partie : chaines de Markov pour la modélisation du langage et MCMC

## 1.1 Chaîne de Markov pour la modélisation du langage

### Question 1

L'élément  $(i,j)$  de la matrice de transition correspond à la probabilité de passer de l'état  $i$  à l'état  $j$ . Il correspond donc à la probabilité que la lettre  $i$  soit suivie de la lettre  $j$  dans la séquence. Dès lors, soit  $\theta$  l'élément  $(i,j)$  de la matrice de transition,  $\theta$  est le paramètre d'une loi de Bernouilli avec comme possibilités :

- l'élément  $i$  est suivi de  $j$  (avec une probabilité  $\theta$ )
- l'élément  $i$  n'est pas suivi de  $j$  (avec une probabilité  $1 - \theta$ )

La méthode du maximum de vraisemblance consiste à maximiser  $P(\mathbf{D}_n|\theta)$  avec  $\mathbf{D}_n$  l'échantillon de donnée, ici seq1 et  $n$  le nombre de données. Pour une variable de Bernouilli, on a : Soit,

$$x_i = \begin{cases} 1 & \text{si } i \text{ est suivi de } j \\ 0 & \text{si } i \text{ n'est pas suivi de } j \end{cases}$$

et  $m$  le nombre d'occurrences de la lettre  $i$ .

$$\begin{aligned} P(\mathbf{D}_n|\theta) &= \prod_{i=1}^m (x_i\theta + (1 - x_i)(1 - \theta)) \\ &= \theta^{n_1}(1 - \theta)^{n_0} \end{aligned}$$

Avec  $n_0$  le nombre de fois où  $x_i = 0$  et  $n_1$  le nombre de fois où  $x_i = 1$ . Déterminons maintenant le  $\theta$  maximisant cette fonction :

$$\begin{aligned} \frac{\partial P(\mathbf{D}_n|\theta)}{\partial \theta} &= n_1\theta^{n_1-1}(1 - \theta)^{n_0} - n_1\theta^{n_1}(1 - \theta)^{n_0-1} \\ &= \theta^{n_1-1}(1 - \theta)^{n_0-1}(n_1(1 - \theta) - n_0\theta) \\ &= \theta^{n_1-1}(1 - \theta)^{n_0-1}(n_1 - \theta(n_1 + n_0)) \end{aligned}$$

La valeur de  $\theta$  maximisant la fonction  $P(\mathbf{D}_n|\theta)$  est donc :

$$\theta_{i,j} = \frac{n_1}{n_0 + n_1} = \frac{\text{nombre d'occurrences de } i \text{ suivies de } j}{\text{nombre d'occurrences de } i}$$

Ceci est implémenté par la fonction `transition_matrix`. Dans le cas de la séquence qui a été fournie,

$$Q = \begin{bmatrix} 0 & 0.0857 & 0.1000 & 0.8143 \\ 1.0000 & 0 & 0 & 0 \\ 0.6744 & 0 & 0 & 0.3256 \\ 0.3662 & 0.1268 & 0.5070 & 0 \end{bmatrix}$$

Ce qui donne le diagramme d'état repris à la figure 1.

De la même manière, la distribution de probabilité initiale sera calculée de la façon suivante.

La probabilité que la chaîne commence par  $x$  vaut  $\frac{\text{nombre de caractères débutant une chaîne valant } x}{\text{nombre de caractères débutant une chaîne}}$ . Dans ce cas ci, vu qu'il n'y a que une seule chaîne et quelle débute par  $a$ ,

$$\pi(0) = (1 \ 0 \ 0 \ 0)$$

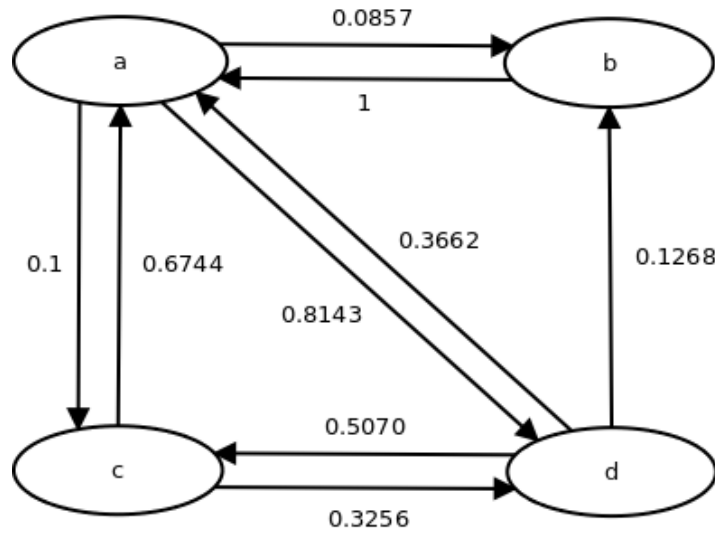


FIGURE 1

### Question 2

Cette question est résolue par la fonction `estimate_prob`. La  $t$ -ième puissance de  $Q$  est simplement calculée avec l'opérateur exposant de matlab.

Pour le calcul de  $P(X_t = x)$ , on calcule  $\pi(t) = \pi(0) * Q^t$  avec  $\pi(0) = (0.25 \ 0.25 \ 0.25 \ 0.25)$  dans le cas où la première lettre est choisie au hasard et  $\pi(0) = (0 \ 0 \ 1 \ 0)$  quand la première lettre est c.  $P(X_t = x)$  est l'élément de  $\pi(t)$  correspondant à  $x$  (le premier pour a, le deuxième pour b, ...).

L'évolution de la probabilité est reprise sur les graphiques 2, 3, 4 et 5. On constate qu'en  $t = 0$ , elle est uniquement dépendante de  $\pi(0)$ , ce qui est normal au vu de la définition de  $\pi(0)$ . Quand  $t$  augmente, la probabilité est de moins en moins dépendante de  $\pi(0)$  et dépend donc de plus en plus de la matrice de transition. C'est en accord avec la théorie car quand le temps augmente, la distribution tend vers la distribution stationnaire si  $Q^t$  converge vers une valeur limite pour  $t \rightarrow +\infty$ . C'est le cas ici car

$$Q^{1000} = Q^{1001} = \begin{bmatrix} 0.3518 & 0.0754 & 0.2161 & 0.3568 \\ 0.3518 & 0.0754 & 0.2161 & 0.3568 \\ 0.3518 & 0.0754 & 0.2161 & 0.3568 \\ 0.3518 & 0.0754 & 0.2161 & 0.3568 \end{bmatrix}$$

### Question 3

Cette question est résolue par la fonction `distrib_station`. La distribution stationnaire peut être obtenue en multipliant une distribution initiale  $\pi(0)$  par la matrice de transition jusqu'à convergence (jusqu'à ce que la différence entre deux itérations soit inférieure à une certaine valeur, qui se doit d'être suffisamment faible). Nous avons utilisé une autre solution équivalent qui consiste à multiplier la matrice de transition par elle-même jusqu'à convergence. La matrice obtenue aura toute ses lignes ayant la même valeur et cette valeur sera la

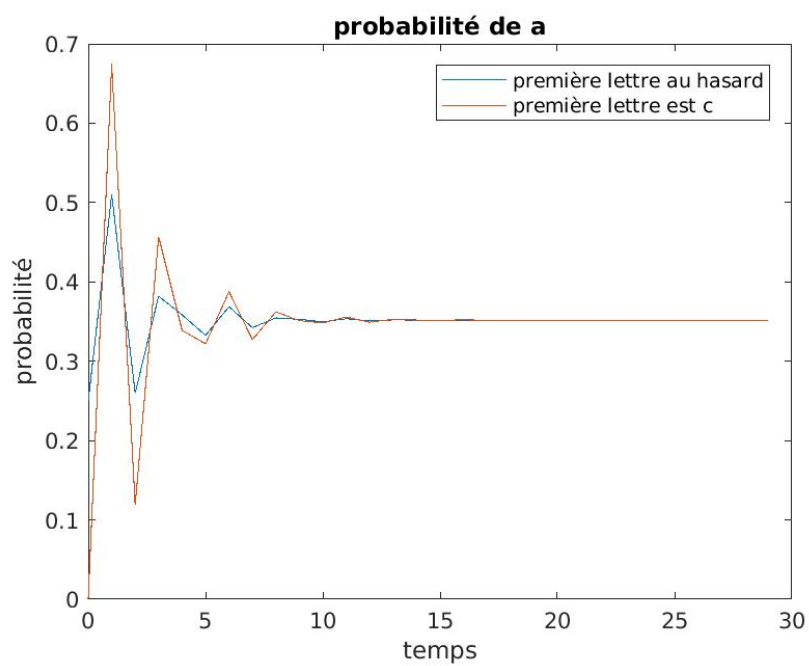


FIGURE 2

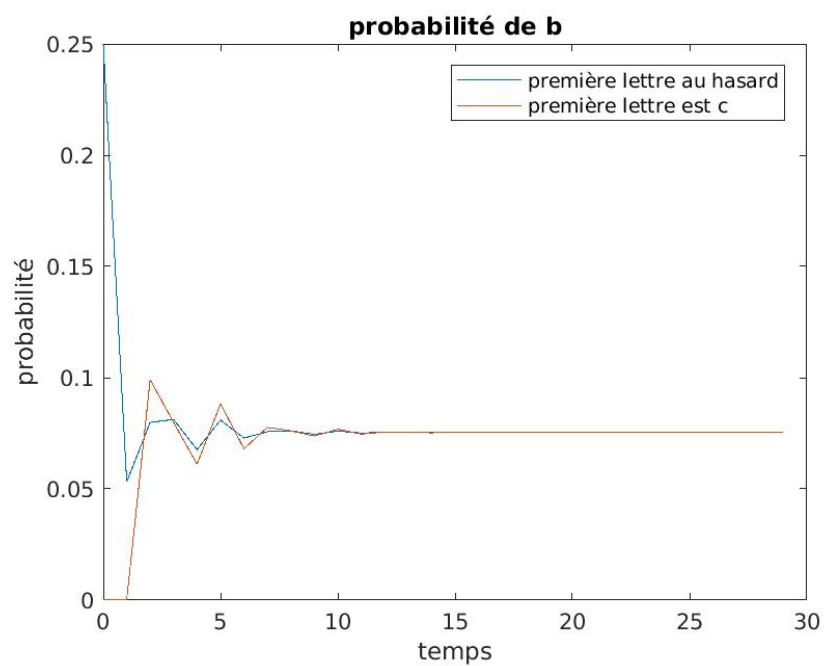


FIGURE 3

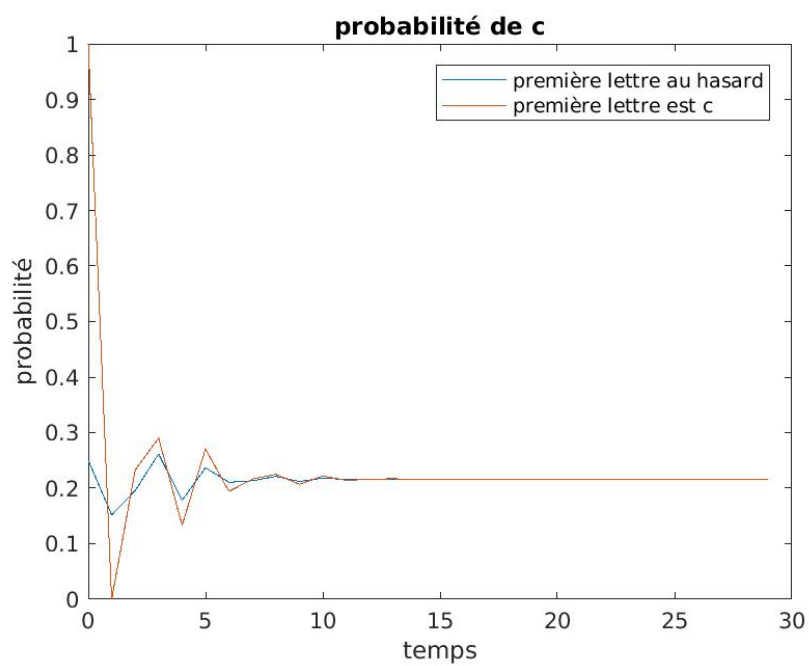


FIGURE 4

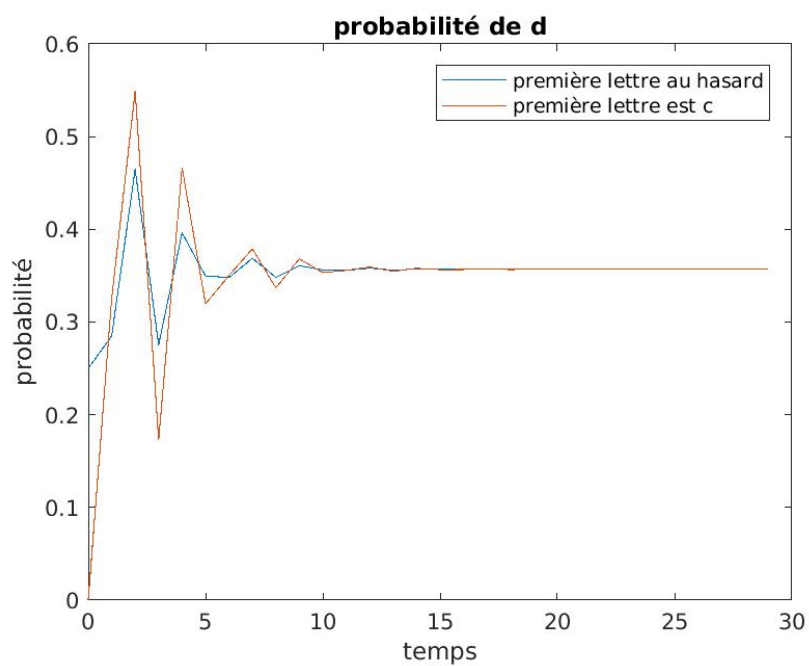


FIGURE 5

distribution stationnaire. La distribution stationnaire obtenue est

$$\pi_{\infty} = (0.3518 \ 0.0754 \ 0.2161 \ 0.3568)$$

#### Question 4

La fonction générant la réalisation de la chaîne de Markov est `realisation`. Elle est implémentée de cette façon, la première lettre est tirée selon la distribution stationnaire et les suivantes selon la ligne de la matrice `Q` correspondant à la lettre précédente.

Les résultat de la proportion de chaque lettre dans la chaîne de Markov pour des longueurs données sont repris au tableau 1. On constate que pour une longueur de chaîne qui augmente, la proportion de chaque lettre tend vers la distribution stationnaire.

longueur	a	b	c	d
1	0	0	0	1
10	0.3	0	0.2	0.5
50	0.36	0.04	0.2	0.4
100	0.35	0.08	0.22	0.35
200	0.345	0.075	0.2	0.38
500	0.37	0.076	0.208	0.346
1000	0.347	0.08	0.215	0.358
10000	0.3546	0.791	0.2125	0.3538
100000	0.35248	0.07425	0.2164	0.35687

Tableau 1 – proportion dans réalisation de chaîne de Markov

#### Question 5

Cette expérience permet de montrer que la variance d'échantillon diminue avec la taille de l'échantillon. En effet, plus la taille augmente et plus les résultats trouvés tendent vers l'espérance (la distribution stationnaire).

En plus de cette expérience, nous avons testé de faire la moyenne d'un nombre élevé de réalisations de longueur 1. Les proportions tendent également vers la distribution stationnaire. Le fait que ces deux expériences amènent aux mêmes résultats montre que la génération d'un élément dans la chaîne de Markov est indépendante du temps. On est donc en présence d'un processus ergodique. Attention, cette dernière remarque n'est valable que car on démarre de la distribution stationnaire.

## 1.2 Algorithme MCMC

#### Question 1

Pour prouver que  $\pi_0$  est une distribution stationnaire de la chaîne de Markov, il suffit de prouver que  $\pi_0 = \pi_0 * Q$ . On sait que les équations de balances détaillées  $\pi_0(i)Q_{i,j} = \pi_0(j)Q_{j,i}$

sont satisfaites. Passant en notation indicelle, on doit donc montrer que :

$$\begin{aligned}
 \pi_0 * Q &= \sum_{k=0}^N \pi_0(k) * Q_{k,i} \\
 &= \sum_{k=0}^N \pi_0(i) * Q_{i,k} \\
 &= \pi_0(i) \sum_{k=0}^N Q_{i,k} \\
 &= \pi_0(i)
 \end{aligned}$$

Cette distribution stationnaire est unique si la matrice de transition  $Q$  est irréductible.

## Question 2

Cette démonstration a été faite avec l'aide du pdf nommé MetropolisExplanation.pdf (inclus dans l'archive) trouvé sur internet.

Étudions d'abord la probabilité de transition.

La probabilité d'obtenir un élément  $x_j$  sachant que l'élément précédent de la chaîne de Markov est  $x_i$  est pour  $i \neq j$  la probabilité que cet élément soit généré selon la loi  $q$  et accepté.

$$P(x_j|x_i) = \alpha(x_j, x_i)q(x_j|x_i)$$

avec

$$\begin{aligned}
 \alpha(x_j, x_i) &= \min \left\{ 1, \frac{f(x_j)}{f(x_i)} \frac{q(x_i|x_j)}{q(x_j|x_i)} \right\} \\
 &= \min \left\{ 1, \frac{cP_X(x_j)}{cP_X(x_i)} \frac{q(x_i|x_j)}{q(x_j|x_i)} \right\}
 \end{aligned}$$

La probabilité d'obtenir à nouveau l'élément  $x_i$  sachant que l'élément précédent de la chaîne de Markov est également  $x_i$  est la somme de la probabilité que l'élément  $x_i$  soit généré selon la loi  $q$  et accepté et de la probabilité que tout autre élément soit généré et refusé.

$$P(x_i|x_i) = \alpha(x_i, x_i)q(x_i|x_i) + \sum_k (1 - \alpha(x_k, x_i))q(x_k|x_i)$$

Dans le cas où l'élément généré est différent du précédent, on a :

$$\begin{aligned}
 P(x_j|x_i)\pi_0(x_i) &= \alpha(x_j, x_i)q(x_j|x_i)\pi_0(x_i) \\
 &= \min \left\{ 1, \frac{cP_X(x_j)}{cP_X(x_i)} \frac{q(x_i|x_j)}{q(x_j|x_i)} \right\} q(x_j|x_i)\pi_0(x_i) \\
 &= \frac{\pi_0(x_i)}{cP_X(x_i)} \min \{ cP_X(x_i)q(x_j|x_i), cP_X(x_j)q(x_i|x_j) \} \\
 &\text{en posant } x_i \leftarrow x_j \text{ et } x_j \leftarrow x_i \\
 &= \frac{\pi_0(x_j)}{cP_X(x_j)} \min \{ cP_X(x_j)q(x_i|x_j), cP_X(x_i)q(x_j|x_i) \} \\
 &= \min \left\{ 1, \frac{cP_X(x_i)}{cP_X(x_j)} \frac{q(x_j|x_i)}{q(x_i|x_j)} \right\} q(x_i|x_j)\pi_0(x_j) \\
 &= \alpha(x_i, x_j)q(x_i|x_j)\pi_0(x_j) \\
 &= P(x_i|x_j)\pi_0(x_j)
 \end{aligned}$$

Dans le cas où l'élément généré est le même que le précédent,  $x_i$  étant égal à  $x_j$  il est évident que

$$P(x_i|x_j)\pi_0(x_j) = P(x_j|x_i)\pi_0(x_i)$$

car

$$P(x_i|x_i)\pi_0(x_i) = P(x_i|x_i)\pi_0(x_i)$$

La chaîne de Markov générée satisfait donc les équations de balance détaillée et au vu de la question 1,  $p_X(x)$  est donc la distribution stationnaire.

Une autre condition est que la probabilité  $p_X(x^{(t-1)})$  ne soit pas nulle afin d'éviter une division par 0. Pour ce faire, il faut que chaque état soit probable. Dans le cadre de l'implémentation pour le décryptage, cette condition se traduit par le fait que chaque chaîne résultant d'une permutation de la chaîne initiale soit possible et donc qu'il soit possible d'avoir n'importe quel caractère suivi de n'importe quel autre caractère ; ce qui se traduit par une matrice de transition ne comportant pas d'éléments nuls.

$q(y^{(t)}|x^{(t-1)})$  ne peut pas être nulle car  $y^{(t)}$  aura été généré avant selon cette même loi et un élément qui a une probabilité nulle d'être généré ne saurait pas être généré.

## 2 Deuxième partie : décryptage d'une séquence codée

### Question 1

Pour déterminer la cardinalité de l'ensemble  $\Theta$ , il suffit de calculer le nombre possibles de permutations des caractères disponibles. On considère la langue anglaise avec 40 caractères. Pour calculer le nombre de permutations possibles, il suffit de considérer qu'on a 40 permutations possibles pour la première lettre, 39 pour la deuxième, ...

On a donc :

$$|\Theta| = 40 * 39 * 38 * \dots * 1 = 40!$$

.



## Question 2

Dans le modèle  $\pi_0, Q$ , on peut trouver la vraisemblance de la séquence  $T'$  en multipliant les probabilités d'avoir la lettre  $n$  de  $T'$  en partant de la lettre  $n - 1$  de  $T'$ . Nos probabilités sont calculées comme suit :

- $P(\text{lettre1} = T'(1))$  est simplement sa probabilité d'avoir cette lettre dans  $\pi_0$ .
- $P(\text{lettre2} = T'(2))$  : est la probabilité d'avoir cette lettre dans  $\pi_0 * Q$ .
- $P(\text{lettre3} = T'(3))$  : est la probabilité d'avoir cette lettre dans  $\pi_0 * Q^2$ .
- etc

La vraisemblance de la chaîne  $T'$  est donc, notant  $N$  la taille de la chaîne  $T'$  et  $|\pi|_{T'(k)}$  la probabilité d'avoir la  $k^{\text{ième}}$  lettre de  $T'$  selon la distribution  $\pi$  :

$$P(T') = \prod_{k=0}^N |\pi_0 * Q^k|_{T'(k)}$$

Pour trouver la vraisemblance de  $D$ , il faut d'abord calculer la matrice  $Q_\theta$  exprimant notre matrice  $Q$  dans le cas de la permutation  $\theta$ . Pour cela, il suffit de permuter les lignes et colonnes de la matrice  $Q$  pour correspondre à la distribution  $\theta$ . Ensuite, la vraisemblance de  $D$  se trouve de la même façon que celle de  $T'$ , à savoir :

$$P(D) = \prod_{k=0}^N |\pi_0 * Q_\theta^k|_{D(k)}$$

## Question 3

Si il n'a que 10 codes candidats possible qui ont la même probabilité, il est assez simple de retrouver le texte de base. Il suffit en effet de calculer la vraisemblance de notre texte avec chaque code  $\theta$  suivant l'algorithme décrit au point 2.2, et puis simplement sélectionner le  $\theta$  menant au texte ayant la plus grande vraisemblance.

## Question 4

L'algorithme ayant été implémenter en suivant le pseudo-code donné, nous n'allons discuter que des particularités différentes du pseudo-code et pas l'algorithme dans son ensemble. La fonction implémentant l'algorithme s'appelle **Metro\_Hast**.

Les seules particularités et difficultés de l'algorithme résident dans le calcul des probabilités permettant de calculer  $\alpha$ .

Premièrement, nous considérons  $q(x^{(t-1)}|y^{(t)}) = q(y^{(t)}|x^{(t-1)})$ . En effet, toutes les fonctions de proposition  $q$  que nous utilisons résultent de permutations aléatoires ou sont des fonctions indépendantes de l'état précédent et uniformément distribuées. Le deuxième cas implique logiquement que l'évaluation de tout élément généré par  $q$  est le même. Le premier cas implique également cette propriété car les permutations étant aléatoires, la probabilité de refaire la permutation arrière est la même que celle de faire la permutation qui a été faite pour arriver à cet état.

Deuxièmement, en ce qui concerne le calcul de  $P_x$ , nous avons simplement implémenté ce qui a été discuté dans la question 2.2. La seule subtilité a été ici d'utiliser les logarithmes des probabilités plutôt que les probabilités elles mêmes. En effet, puisque la chaîne à décoder

est assez longue, nous arrivons vite à des probabilités plus petites que l'épsilon machine et donc considérées comme nulles par la machine. L'utilisation des logarithmes évite ce problème. Aussi, les probabilités nulles de la matrice  $Q$  ont été remplacées par des probabilités très faible, afin de pouvoir différencier les probabilités de deux chaînes avec une transition "impossibles", mais dont une est plus probable que l'autre en ce qui concerne les autres éléments. Cela évite également des divisions par 0 qui seraient problématiques.

### Question 5

La distribution de proposition choisie est implémentée par la fonction `random_flip` et consiste à permuter la position de deux éléments dans la permutation de l'alphabet actuelle. L'expérience menée consiste à exécuter l'algorithme 20 fois pour chaque longueur traitée. L'algorithme est utilisé avec les caractéristiques suivantes :

- la permutation de départ est celle ne permutant aucun caractère de la chaîne.
- la convergence est considérée atteinte après avoir conservé l'ancienne valeur de  $x$  100 fois dans l'algorithme.
- Si la convergence n'est pas atteinte après 20000 itérations de l'algorithme, l'algorithme s'arrête et on considère qu'il n'y a pas convergence. La qualité de la chaîne produite est tout de même évaluée car on a remarqué expérimentalement que le fait de ne pas atteindre la convergence ne diminue pas la qualité de la chaîne et il n'y a donc pas d'intérêt à différencier ces métriques.

Afin d'évaluer la convergence, les métriques utilisées sont la proportion d'exécution amenant à une convergence (selon les critères définis ci-dessus) et le nombre moyen d'itérations nécessaires à la convergence si il y a convergence.

En ce qui concerne la qualité de la chaîne, le pourcentage de chaîne correctement déchiffrées à été obtenu en comparant à un résultat obtenu précédemment que l'on sait correct la chaîne décryptée, en permettant quelques erreurs. En effet, dans le cadre de ce projet, le but est de rendre un texte codé en texte intelligible, et quelques erreurs mineures (par exemple changer les "-" en ";") n'influent pas vraiment notre capacité à comprendre le texte décrypté. D'un point de vue pratique, les signes de ponctuation "rares" ( :;, ?, ...), sont problématiques puisque comme ils sont en petit nombres, ils influent beaucoup moins la probabilité de la chaîne par rapport à permuter deux lettres de l'alphabet, et l'algorithme a donc beaucoup plus de mal à converger vers la bonne valeur de ces symboles. Dans notre cas, nous avons par exemple "slow-moving", qui sera décrypté en "slow ;moving" ou "slow ?moving", mais nous considérons que ça n'affecte pas la compréhension et que ce n'est pas une erreur.

Le tableau 2 reprend ces différentes métriques. On remarque que plus la chaîne est longue, plus la convergence est rapide et plus la qualité du résultat obtenu est élevée. C'est assez logique car plus la chaîne est longue, plus on a d'information alors que le nombre d'inconnues reste constant (il y a toujours autant de caractères dans l'alphabet). On pourrait considérer que le nombre d'inconnues n'est pas tout à fait constant car plus la chaîne est courte, plus les chances que des caractères ne soient pas dans la chaîne sont élevées mais c'est négligeable par rapport à l'apport d'information.

longueur	% convergence	nombre d'itérations si convergence	% correct
10	0	pas applicable	0
100	0	pas applicable	0
500	0	pas applicable	45
750	0	pas applicable	50
1000	10	8955	60
1300	80	8690	60

Tableau 2 – convergence et qualité selon la longueur d'une chaîne

### Question 6

Pour implémenter la convergence dans notre algorithme, nous avons tout simplement regardé le nombre d'itération successives pour lesquelles l'état courant de notre algorithme de Metropolis-Hastings reste le même. Nous donnons ensuite une valeur  $x$  de convergence à l'algorithme, et quand l'état courant est resté  $x$  fois le même, nous considérons que la convergence est atteinte.

Nous avons alors sélectionné une série de densités de proposition pour l'algorithme de Metropolis-Hastings. Nous avons choisi :

- `random_flip` : flip deux éléments de la distribution actuelle
- `random_flip_4` : effectue deux `random_flip` successifs
- `random_flip_close` : flip deux éléments côte à côte du tableau
- `random_permutation` : génère une permutation aléatoire du tableau

Nous avons ensuite conduit une série de test pour regarder les résultats en fonction des différentes valeurs de convergence. Ces tests ont été effectués avec un nombre limite d'itération de l'algorithme de 20000. Si cette limite de 20000 itérations est dépassée, nous considérons que nous ne convergeons pas, en tout cas pas dans un délai raisonnable. Les statistiques ont été obtenus en avec 20 essais, ce qui n'est pas un très grand nombre mais est suffisant pour se donner une idée des résultats que l'on doit obtenir. Augmenter le nombre d'essais nous aurait permis d'augmenter la fiabilité des tests, mais pour des contraintes de temps de calcul (calculer tous les statistiques nous prenait environ 2h), nous nous sommes contentés de 20 essais.

Nous voyons dans les tableaux 3, 4, 5 et 6 que la seule méthode convergeant est `random_flip`. On remarque que si la valeur seuil de convergence est trop basse, on convergera trop vite et on n'obtiendra pas la bonne réponse. On voit aussi que même pour des valeurs de convergence assez élevées, les méthodes `random_flip_4` et `random_permutation` convergent assez vite, ce qui se comprend par le fait qu'elles sont "mauvaises", et donc que l'algorithme a beaucoup de mal d'améliorer la vraisemblance de la chaîne avec ces méthodes.

Le point le plus notable est que surprenamment la méthode `random_flip` a beaucoup de mal à converger pour des valeurs de convergence élevées. On voit en fait sur la figure 6 que la vraisemblance de la chaîne a depuis longtemps convergé vers une valeur, mais que

l'algorithme trouve de nouveaux code ayant presque la même vraisemblance mais pas tout à fait. Cela pourrait être par exemple des permutations sur la ponctuation rare évoquée précédemment. Cela montre que trouver une réponse "parfaite" avec notre algorithme est très compliquée, il faut se satisfaire de résultats très proches de la bonne réponse.

Nous choisissons donc évidemment la fonction `random_flip` pour décrypter notre chaîne, avec une valeur de convergence de 200, ce qui nous donne la bonne chaîne 50% du temps, ce qui est plus que correct.

Finalement, nous trouvons la chaîne décodée suivante, issue de "A Song of Ice and Fire" de Georges R.R. Martin :

it stretched before them to the misty cast, a tranquil land of rich black soil, wide slow : moving rivers, and hundreds of small lakes that shone like mirrors in the sun, protected on all sides by its sheltering peaks. wheat and corn and barley grew high in its fields, and even in highgarden the pumpkins were no larger nor the fruit any sweeter than here. they stood at the western end of the valley, where the high road crested the last pass and began its winding descent to the bottomlands two miles below. the vale was narrow here, no more than a half day's ride across, and the northern mountains seemed so close that catelyn could almost reach out and touch them. looming over them all was the jagged peak called the giant's lance, a mountain that even mountains looked up to, its head lost in icy mists three and a half miles above the valley floor. over its massive western shoulder flowed the ghost torrent of alyssa's tears. even from this distance, catelyn could make out the shining silver thread, bright against the dark stone. when her uncle saw that she had stopped, he moved his horse closer and pointed. "it's there, beside alyssa's tears. all you can see from here is a flash of white every now and then, if you look hard and the sun hits the walls just right."

Convergence	Moyenne itérations	% convergence	% corrects
10	79.75	100	0
50	1137.15	100	0
100	9731.7	90	45
200	limite atteinte	0	45
500	limite atteinte	0	45

Tableau 3 – Statistique sur la convergence de l'algorithme `random_flip`

Convergence	Moyenne itérations	% convergence	% corrects
10	47.9	100	0
50	350.1	100	0
100	886.1	100	0
200	2025.7	100	0
500	7319.3	100	0

Tableau 4 – Statistique sur la convergence de l’algorithme random\_flip\_4

Convergence	Moyenne itérations	% convergence	% corrects
10	55.4	100	0
50	1708.3	100	0
100	4740.8	65	0
200	19902	5	0
500	limite atteinte	0	0

Tableau 5 – Statistique sur la convergence de l’algorithme random\_flip\_close

Convergence	Moyenne itérations	% convergence	% corrects
10	17.3	100	0
50	89.3	100	0
100	163.45	100	0
200	356.1	100	0
500	946.5	100	0

Tableau 6 – Statistique sur la convergence de l’algorithme random\_permutation

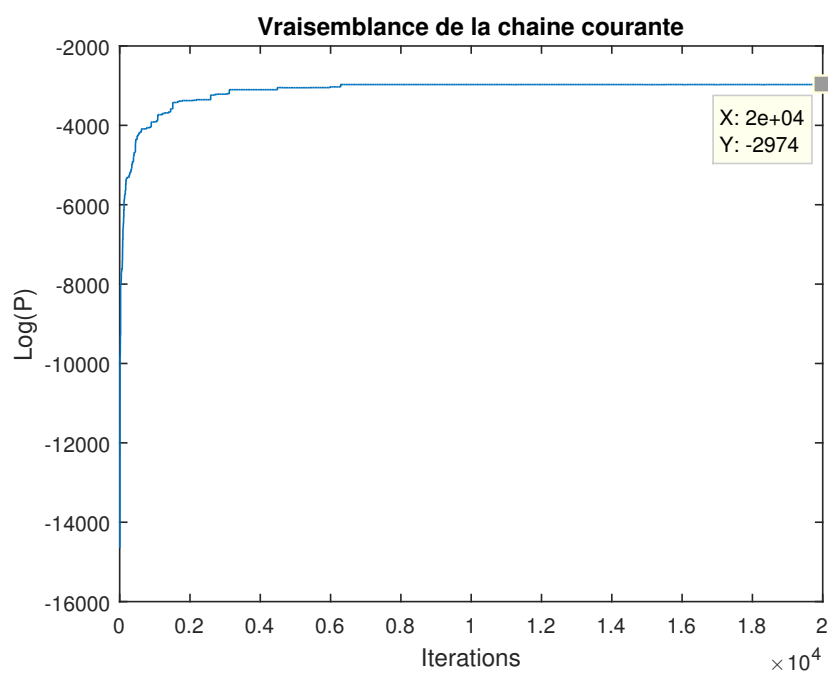


FIGURE 6