Université de Liège

Eléments de processus stochastiques

Méthodes de Monte Carlo par chaines de Markov - Application à la cryptanalyse

3ème bachelier en ingénieur civil

Auteurs :
Tom Crasset
Antoine Louis
Romain Vaneukem

Professeur :
V. Denoël

Assistant :
L. Duchesne

Année académique 2017-2018

Table des matières

1 Chaînes de Markov pour la modélisation du langage et MCMC

1.1 Chaîne de Markov pour la modélisation du langage

1.1.1 Matrice de transition et distribution de probabilité initiale

La méthode de vraisemblance a pour but de trouver la valeur du paramètre θ qui maximise la probabilité de trouver l'échantillon D_n qui, ici, correspond la séquence donnée par `seq1`. D'un point de vue théorique, nous avons

$$\hat{\theta}_{MV} = arg(max_\theta \mathcal{L}(\chi_i, \theta))$$

où la fonction de vraisemblance $\mathcal{L}(\chi_i, \theta)$ est telle que

$$\mathcal{L}(\chi_i, \theta) = P(D_n \mid \theta) = P((\chi_1, ..., \chi_n) = (x_1, ..., x_n))$$

Les événements étant indépendants, on a

$$\mathcal{L}(\chi_i, \theta) = P((\chi_1 = x_1), ..., (\chi_2 = x_n)) = \prod_i P(\chi_i = x_i)$$

Pour trouver la matrice de transition par cette méthode, nous nous basons sur la séquence fournie. Prenons l'exemple de l'élément $\mathbf{Q}(2,3)$ qui représente la probabilité de passer de la lettre 'b' à la lettre 'c'. Nous cherchons le nombre total des séquences 'bc' et divisons ce total par le nombre de 'b', ce qui nous donne alors la probabilité recherchée. Cette même méthode peut être appliquée pour trouver la distribution initiale π_0. Dans ce cas, la fréquence d'apparition d'une lettre est simplement divisée par le nombre total de lettres. Nous trouvons alors comme résultats

$$\mathbf{Q} = \begin{bmatrix} 0 & 0.0857 & 0.1 & 0.8143 \\ 1 & 0 & 0 & 0 \\ 0.6744 & 0 & 0 & 0.3256 \\ 0.3662 & 0.1268 & 0.507 & 0 \end{bmatrix}$$

$$\pi_0 = \begin{pmatrix} 0.355 \\ 0.075 \\ 0.215 \\ 0.355 \end{pmatrix}$$

En analysant la matrice de transition $\mathbf{Q}$, nous remarquons certains résultats intéressants.Tout d'abord, nous voyons que la diagonale de la matrice est nulle. Nous en concluons que la probabilité qu'une séquence ait deux fois la même lettre qui se suit est nulle. Chaque élément nul de cette matrice correspond ainsi à un événement impossible. Par exemple, on remarque qu'il est impossible que la lettre "b" précède la lettre "c". A contrario, un élément de la matrice égal à l'unité exprime une probabilité certaine qu'un événement se produise. Ainsi, on voit par exemple que la lettre "b" sera à chaque fois suivie par la lettre "a".

Nous pouvons observer à la Figure 1, le diagramme d'états de la chaîne de Markov de la séquence en question.

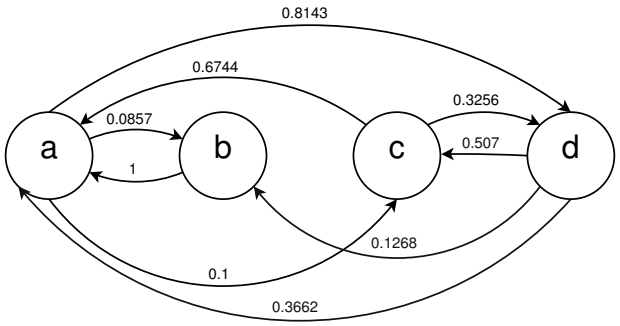

FIGURE 1 – Diagramme d'états de la chaîne de Markov

1.1.2 Calcul de probabilités sur base de la matrice de transition

Maintenant que la matrice de transition a pu être calculée, il nous est possible de calculer certaines quantités à l'aide de celle-ci. Dans un premier temps, nous supposons que la première lettre est choisie aléatoirement, ce qui correspond à une distribution de probabilité initiale uniforme :

$$\pi_0 = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

Pour obtenir les probabilités successives $P(X_t = i)$, il suffit de multiplier chaque densité par la matrice de transition. Cela revient à effectuer

$$\pi_{n+1} = \pi_n \times \mathbf{Q}$$

pour n allant de 0 à $t - 1$. Graphiquement, nous pouvons observer nos résultats sur la Figure 2 ci-dessous.

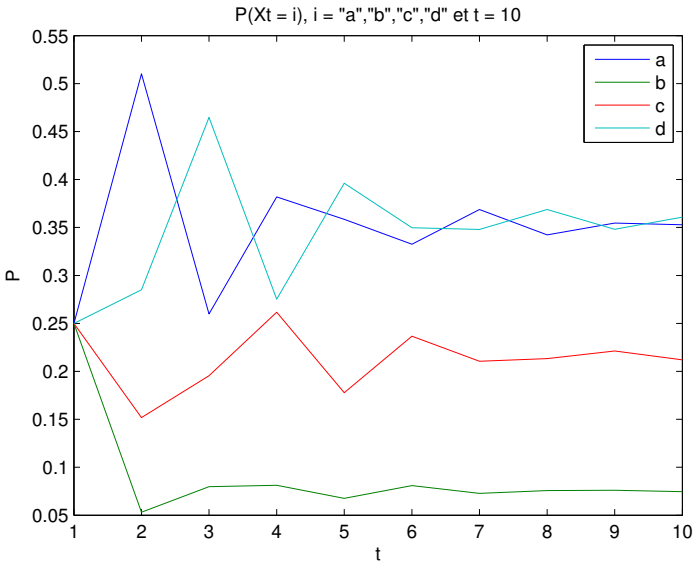

FIGURE 2 – Représentation graphique de $P(X_t = i)$ avec i = 'a','b','c','d' partant d'une lettre aléatoire

Ensuite, il nous est demandé de représenter cette probabilité $P(X_t = i)$ en supposant que la première lettre tirée est toujours la lettre 'c'. La différence ici se trouve dans la distribution de probabilité initiale. Nous avons dans ce cas :

$$\pi_0 = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}$$

Les résultats trouvés peuvent être observés à la Figure 3.

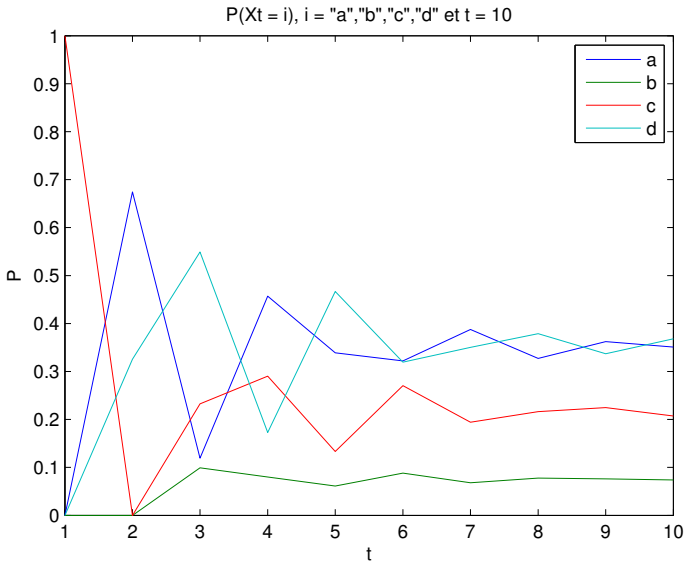

FIGURE 3 – Représentation graphique de $P(X_t = i)$ avec i = 'a','b','c','d' partant de la lettre 'c'

D'un point de vue théorique, la matrice **Q** reflète les probabilités de passer d'un état à un autre. Il s'agit en fait de la probabilité conditionnelle suivante

$$Q_n(i, j) = P(\chi_{n+1} = j | \chi_n = i_n)$$

A partir de la distribution initiale et de la matrice de transition, on peut caractériser entièrement la fonction de probabilité conjointe de la chaîne de Markov. En effet, par définition, on a

$$P(\chi_1 = i_1, X_0 = i_0) = P(X_1 = i_1 | X_0 = i_0) P(\chi_0 = i_0)$$

On peut aussi s'intéresser à la fonction de probabilité conjointe entre les trois premières lettres.

$$P(\chi_2 = i_2, \chi_1 = i_1, \chi_0 = i_0) = P(\chi_2 = i_2 | \chi_1 = i_1, \chi_0 = i_0) P(\chi_1 = i_1, \chi_0 = i_0)$$

Par définition d'une chaîne de Markov, un événement n vers un événement $n+1$ ne dépend pas de l'historique. Ainsi,

$$P(\chi_2 = i_2, \chi_1 = i_1, \chi_0 = i_0) = P(\chi_2 = i_2 | \chi_1 = i_1,) Q(i_1, i_0) \pi_0 = Q(i_1, i_2) Q(i_0, i_1) \pi_0$$

Et par récurrence,

$$P(\chi_n = i_n, ..., \chi_0 = i_0) = \pi_0 \prod_{k=1}^{n} Q(i_{k-1}, i_k)$$

On peut alors observer que, dans les évolutions représentées ici, les valeurs tendent, lorsque t devient de plus en plus grand, vers une distribution déterministe. Lorsque t tend vers l'infini, nous obtenons la distribution de probabilité stationnaire. Celle-ci sera étudiée plus en détails au point 1.1.3.

Finalement, nous étudions comment se comporte et à quoi correspond la matrice de transition élevée à la puissance t. De ce qui a été vu précédemment, cette puissance de Q est la matrice de transition pour une expérience ayant effectué t étapes. L'élément $Q(i_{k-1}, i_k)^t$ est la probabilité de passer à l'état i_{k-1} sachant i_k lorsque nous nous trouvons à la $t^{ème}$ étape. Calculons certaines puissances de cette matrice et reportons les résultats à la Table 1.

$$\mathbf{Q} = \begin{bmatrix} 0 & 0.0857 & 0.1 & 0.8143 \\ 1 & 0 & 0 & 0 \\ 0.6744 & 0 & 0 & 0.3256 \\ 0.3662 & 0.1268 & 0.507 & 0 \end{bmatrix} \quad \mathbf{Q}^3 = \begin{bmatrix} 0.3936 & 0.0428 & 0.0616 & 0.5019 \\ 0.4513 & 0.1032 & 0.4129 & 0.0326 \\ 0.4570 & 0.0798 & 0.2904 & 0.1728 \\ 0.2257 & 0.0989 & 0.2818 & 0.3936 \end{bmatrix}$$

$$\mathbf{Q}^{10} = \begin{bmatrix} 0.3591 & 0.0745 & 0.2147 & 0.3517 \\ 0.3424 & 0.0777 & 0.2239 & 0.3560 \\ 0.3482 & 0.0768 & 0.2218 & 0.3533 \\ 0.3487 & 0.0749 & 0.2123 & 0.3641 \end{bmatrix} \quad \mathbf{Q}^{20} = \begin{bmatrix} 0.3518 & 0.0754 & 0.2161 & 0.3567 \\ 0.3516 & 0.0754 & 0.2162 & 0.3568 \\ 0.3517 & 0.0754 & 0.2161 & 0.3568 \\ 0.3517 & 0.0754 & 0.2160 & 0.3569 \end{bmatrix}$$

TABLE 1 – Puissances de la matrice de transition

On observe premièrement que l'élévation de cette matrice à une certaine puissance à pour effet de faire disparaître les éléments nuls de la matrice. Cela montre bien que nous avons à faire à une chaîne irréductible. En effet, il est alors possible, après un certain temps, d'être passer par tout les états. Nous remarquons également que, plus l'exposant grandit, plus la matrice converge vers un état stationnaire, comme évoqué plus haut.

1.1.3 Distribution stationnaire de la chaîne de Markov

La distribution de probabilité stationnaire peut être trouvée relativement facilement par application directe de sa définition, c'est-à-dire

$$[\pi_\infty]_j = \lim_{t \to \infty} P(\chi_t = j)$$

Il suffit alors de multiplier la distribution de probabilité initiale par Q élevé à une certaine puissance, cette dernière étant suffisamment grande pour ne plus observer de changements au cours des itérations. On a dès lors

$$[\pi_\infty] = \pi_0 Q^t$$

En pratique, il est question de trouver un t suffisamment grand pour quitter la période transitoire, comme nous pouvons le constater à la Figure 4.

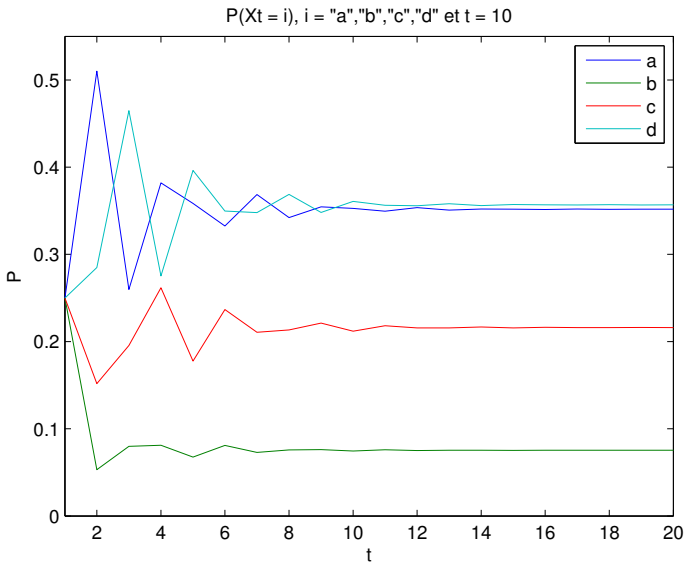

FIGURE 4 – Représentation graphique de la période stationnaire

On peut observer qu'un t allant jusqu'à 20 est suffisant pour atteindre la stationnarité avec une distribution initiale uniforme. On obtient ainsi

$$\pi_\infty = \begin{bmatrix} 0.3517 & 0.0754 & 0.2161 & 0.3568 \end{bmatrix}$$

1.1.4 Réalisation aléatoire de la chaîne de Markov

Il nous est demandé ici de générer une chaîne de Markov en démarrant d'une lettre choisie aléatoirement selon la distribution stationnaire. Chaque lettre est ensuite choisie selon la matrice de transition. Par exemple, si nous nous trouvons à la lettre 'a', la lettre suivante est choisie selon les probabilités : $P(\chi = a) = 0,\ P(\chi = b) = 0.0857,\ P(\chi = c) = 0.1,\ P(\chi = d) = 0.8143$. Voici un exemple de réalisation pour une longueur de chaîne de 20 : '`dcacadcadcacadbabadadcadadcada`'.

On peut analyser la distribution de probabilité de cette chaîne générée avec la distribution stationnaire en regardant la fréquence de chaque lettre et en la divisant par la longueur de chaîne totale.

Les résultats en fonction de la longueur de chaîne sont disponibles à la Table 2.

T	$P(\chi = a)$	$P(\chi = b)$	$P(\chi = c)$	$P(\chi = d)$
20	0.3	0.05	0.2	0.45
50	0.34	0.08	0.22	0.36
100	0.37	0.11	0.18	0.34
170	0.3706	0.1	0.1765	0.3529
230	0.3435	0.0609	0.2478	0.3478
600	0.3517	0.0733	0.2217	0.3533
π_∞	0.3517	0.0754	0.2161	0.3568

TABLE 2 – Distribution de probabilités des chaînes générées, où T est la longueur de chaîne

La distribution de probabilité stationnaire a été rappelée pour permettre une comparaison plus aisée. On observe ainsi que, plus la longueur de la séquence est élevée, plus les probabilités se rapprochent de l'état stationnaire calculé précédemment.

1.1.5 Conclusion de l'expérience

Lors de cette expérience, nous sommes partis d'une séquence donnée et nous avons étudié sa répartition dans le but de pouvoir simuler une chaîne similaire. La méthode de vraisemblance nous a permis de caractériser la séquence donnée et de définir sa représentation probabiliste. A partir de cette représentation, il nous a été possible de générer une séquence similaire.

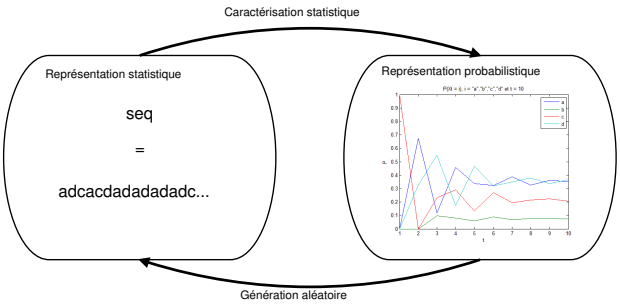

FIGURE 5 – Deux représentations équivalentes du processus aléatoire

Nous pouvons observer à la Figure 5 la démarche suivie lors des questions précédentes. Nous avons dès lors pu constater au point 1.1.4 que le résultat convergeait vers la distribution stationnaire de la matrice Q de départ. A une certaine chaîne de Markov, nous pouvons attribuer deux représentations : une représentation statistique et une représentation probabiliste. Ces deux représentations sont interchangeables, comme montré à la Fiqure 5. Lors des questions précédentes, nous sommes partis d'une représentation statistique, la chaîne initiale, et nous en avons déterminé la matrice de transition et la distribution de probabilités initiale. De là, nous avons simulé une chaîne similaire et le passage par la représentation statistique. De ces expériences, nous avons pu tirer des similitudes entre les deux approches. Toutes ces notions sont des principes clés de la stochastique, plus précisément des chaînes de Markov que nous avons donc pu mettre en pratique, ce qui nous a aidé à mieux les comprendre et mieux les utiliser.

1.2 Algorithme MCMC

1.2.1 Démonstration 1

Tout d'abord, nous savons qu'une distribution de probabilités est dite stationnaire lorsque

$$\pi_{stat} = \pi_{stat} Q \tag{1}$$

L'équation (1) est appelée l'équation de balance générale.

Soit $X = \{1, .., N\}$ un espace d'état discret. Une chaîne de Markov avec une distribution de probabilités initiale π de X et de matrice de transition Q sur X est dite réversible si elle satisfait

$$\pi(i)Q_{ij} = \pi(j)Q_{ji} \qquad , \forall i, j \in X \tag{2}$$

, où Q_{ij} est la probabilité que la chaîne se déplace à l'état j en partant de l'état i. Les équations (2) sont appelées équations de balance détaillée.

Si une chaîne de Markov est construite de manière à être réversible, alors il suit immédiatement de (2) que π sera son unique distribution stationnaire. Il est beaucoup plus facile de prouver que les équations de balance détaillé (2) sont d'application que de prouver l'équation générale de balance (1).

Si pour $\forall i, j \in X$, les équations de balance (2) sont satisfaites, alors π est une distribution stationnaire de Q. Si nous prenons une distribution initiale $\pi(i)$ et que nous utilisons l'égalité (2), on obtient

$$\begin{aligned} \sum_i \pi(i)Q_{ij} &= \sum_i \pi(j)Q_{ji} \\ &= \pi(j)\sum_i Q_{ji} \\ &= \pi(j) \qquad\qquad , \forall i, j \in \{1, .., N\} \end{aligned} \tag{3}$$

ce qui prouve la stationnarité de notre distribution initiale.

Finalement, la distribution initiale de la chaîne de Markov est unique si la chaîne est irréductible, c'est-à-dire si la probabilité d'atteindre l'état i partant de j est non nulle. On voit dans notre fichier `Q.mat` que certaines lettres ne pourront jamais immédiatement suivre une autre, mais cela ne veut pas dire qu'elles ne seront jamais accessibles. Ainsi, on peut dire que notre chaîne de Markov est irréductible et donc notre distribution unique.

1.2.2 Démonstration 2

Le principe majeur derrière les chaînes de Markov Monte Carlo est le fait que n'importe quelle chaîne de Markov à nombre fini d'états discrets, qui est apériodique et qui est irréductible, a une distribution unique vers laquelle la chaîne va converger à l'infini.

Soit $X = \{1, .., N\}$ un espace d'état discret et $\forall i, j \in X$ on a

$$Q_{ij} = q(i, j)\alpha(i, j) \qquad \forall\ i \neq j$$

où Q_{ij} est la matrice de transition sur X d'une chaîne de Markov quelconque, $q(i, j)$ la distribution de proposition et $\alpha(i, j)$ la probabilité d'acceptation telle que

$$\alpha(i, j) = \frac{s(i, j)}{1 + t(i, j)}$$

où $s(i, j)$ est une fonction symétrique de i et j choisie de telle façon que pour tout i et j, $0 \leq \alpha(i, j) \leq 1$ et

$$t(i, j) = \frac{\pi_X(i)q(i, j)}{\pi_X(j)q(j, i)}$$

Soit une fonction $f(x) = c\pi_X(x)$ et remplaçons π_X par cette fonction dans la formule ci-dessus. Dans ce cas, montrons que la chaîne de Markov générée satisfait les équations de balance détaillée en respect avec la distribution π_X.

$$\begin{aligned} f(i)Q_{ij} &= c\pi_X(i)q(i,j)\alpha(i,j) \\ &= \frac{c\pi_X(i)q(i,j)s(i,j)}{1+\frac{c\pi_X(i)q(i,j)}{c\pi_X(j)q(j,i)}} \\ &= \frac{c\pi_X(i)q(i,j)s(j,i)\pi_X(j)q(j,i)}{\pi_X(j)q(j,i)+\pi_X(i)q(i,j)} \\ &= \frac{c\pi_X(j)q(j,i)s(j,i)}{1+\frac{c\pi_X(j)q(j,i)}{c\pi_X(i)q(i,j)}} \\ &= c\pi_X(j)q(j,i)\alpha(j,i) \\ &= c\pi_X(j)Q_{ji} \end{aligned} \tag{4}$$

La constante c étant présente des deux côtés, si $f(x)$ satisfait les équations de balance, π_x le fait tout autant. Il s'ensuit qu'on a donc bien une distribution stationnaire en π_X. Comme dans la démonstration 1, une contrainte supplémentaire est que la chaîne de Markov doit être irréductible.

2 Décryptage d'une séquence codée

2.1 Cardinalité de l'ensemble Θ

Il nous est demandé ici de calculer la cardinalité de l'ensemble Θ. L'ensemble Θ comprend toutes les permutations des caractères utilisés dans la langue anglaise.

Nous avons un total de $N = 40$ symboles. De l'analyse combinatoire, nous savons que le nombre de permutations pour des éléments tous différents est égal à la factorielle du nombre d'éléments. Nous obtenons alors

$$\#(\Theta) = N! = 8.16 \times 10^{47}$$

2.2 Calcul de la vraisemblance

A partir de la matrice de transition Q, de la loi de distribution initiale π_0 et d'un texte non crypté T', nous cherchons sa vraisemblance, c'est-à-dire la probabilité d'observer cette séquence. pour ce faire, on compare le premier symbole avec la probabilité de l'observer grâce à π_0, puis on observe le deuxième symbole de la chaîne. Sachant que l'on a le premier symbole, on regarde quel est la probabilité d'avoir le deuxième symbole par $\pi_0\mathbf{Q}$. De même pour le troisième symbole mais avec $\pi_0 \times \mathbf{Q}^2$ et ainsi de suite jusqu'au dernier symbole. On multiplie chaque probabilité des symboles pour obtenir la vraisemblance de T'.

Pour la vraisemblance d'un texte crypté D, cela revient à appliquer le même procédé que précédemment mais en ayant au préalable décrypté le texte par $T' = \theta^{-1}(D)$ vu la relation biunivoque du cryptage.

2.3 Description d'un algorithme

Nous disposons de 10 codes de déchiffrement possibles, qu'on peut choisir de manière équiprobable. Pour chaque code, on va calculer la vraisemblance associée. Ensuite, nous allons utiliser la probabilité a posteriori $p_{\theta|D,\pi_0,Q}(\theta)$ pour chaque code de déchiffrement. Dans cette formule, la distribution de probabilité à priori $p_\theta(\theta)$ se factorise car cette dernière peu se sortir de la somme vu qu'elle est uniforme. Le code de déchiffrement ayant la probabilité à posteriori la plus élevée sera le code que nous privilégierons.

2.4 Algorithme de Metropolis-Hastings

L'algorithme de Metropolis-Hastings est assez simple. Il consiste a prendre un état de départ x_0 et de tirer, à partir d'une loi de distribution q, l'état suivant y. On va par après choisir d'accepter ou de rejeter ce dernier en se basant sur la valeur alpha, qui joue le rôle le plus important de l'algorithme. Ce dernier se calcule comme suit :

$$\begin{aligned} \alpha &= min\{1, \frac{p_\chi(y^{(t)})}{p_\chi(x^{(t-1)})} \frac{q(x^{(t-1)}|y^{(t)})}{q(y^{(t)}|x^{(t-1)})}\} \\ &= min\{1, \frac{p_\chi(y^{(t)})}{p_\chi(x^{(t-1)})}\} \end{aligned} \tag{5}$$

De plus, il est nécessaire de préciser que ceci se passe en moyenne, car dans l'algorithme de Metropolis-Hastings, le taux d'acceptation est choisi aléatoirement dans l'intervalle $[0, 1]$. Du coup, il est parfois possible de régresser quelque peu, mais ceci fait la force de Metropolis-Hastings, car cela empêche d'être coincé dans des maximas locaux alors qu'il existe un maximum global.

Notons que la simplification que nous effectuons dans la formule (5) ne peut être faite que si la distribution de proposition q est symétrique. Dans ce cas, le rapport des deux distributions est égal à 1 et notre α sera alors le minimum entre 1 et le rapport des probabilités à posteriori. Nous tombons alors simplement sur l'algorithme de Metropolis. Il conviendra à présent de choisir minutieusement cette distribution. Ce choix sera discuté lors des questions suivantes.

Revenons maintenant au calcul de notre taux d'acceptation α. Il est tout d'abord intéressant de remarquer que les probabilités à posteriori (6) peuvent être simplifiées :

$$\begin{aligned} P_{\theta|D,\pi_0,Q}(\theta) &= \frac{\mathcal{L}(\theta^{-1}(D), \pi_0, Q) p_\theta(\theta)}{\sum_{\theta \in \Theta} \mathcal{L}(\theta^{-1}(D), \pi_0, Q) p_\theta(\theta)} \\ &= \frac{\mathcal{L}(\theta^{-1}(D), \pi_0, Q)}{\sum_{\theta \in \Theta} \mathcal{L}(\theta^{-1}(D), \pi_0, Q)} \end{aligned} \tag{6}$$

De plus, lorsqu'on effectue le rapport des probabilités à posteriori, les sommes des vraisemblances se factorisent car elles sont identiques. En effet, il s'agit en fait de la somme des vraisemblances de chaque permutation possible de l'ensemble θ, elle reste donc constante pour chaque $\theta \in \Theta$. Par conséquent, le calcul des probabilités à posteriori

se réduit au simple calcul de la vraisemblance du texte, dont le principe a déjà été abordé à la question 2.

Par ailleurs, en calculant la vraisemblance d'un texte très grand, il se peut qu'on obtienne des erreurs d'arrondi dû aux limites du calcul par virgules flottantes. On va alors procéder au calcul de la vraisemblance par logarithme pour se débarrasser de ces erreurs. Toutefois, nous devons quand même faire attention aux zéros de la matrice de transition Q à cause du rapport effectué lors du calcul d'α. Nous avons donc également choisi d'ajouter une valeur minuscule à chaque élément de la matrice Q. Cela laisse les nombres non-nuls quasi inchangés mais permet de transformer un élément nul en un élément infiniment petit mais non nul.

Finalement, un dernier élément important dans l'algorithme de Metropolis-Hastings est le choix de l'état de départ x_0. En effet, si ce dernier est déjà assez vraisemblable avec la clé que l'on doit trouver, cela nous permettra d'effectuer moins d'itérations et de converger plus vite.

2.5 Convergence de l'algorithme

Pour étudier la convergence de notre algorithme, nous choisissons une distribution de proposition symétrique : une permutation de deux symboles de notre clé de cryptage à chaque itération. Ce choix est expliqué un peu plus en détails à la question 2.6.

Nous allons dans un premier temps étudier le nombre d'itérations nécessaire pour obtenir une convergence satisfaisante. Dans un second temps, nous testerons l'influence de la taille du texte à décrypter. Finalement, nous parlerons de l'influence d'un bon état initial x_0 pour commencer l'algorithme. Ces tests seront effectués à l'aide d'un texte connu que nous crypterons grâce à une clé d'encodage déterminée. On définit la précision comme étant le nombre de caractères correctement déchiffrés sur le nombre total de caractères. Le nombre de caractères trouvés est une moyenne sur 10 expériences étant donnée la fluctuation des résultats d'un décryptage à l'autre.

Nombre d'itérations Le texte choisi pour effectuer ces tests est une partie du roman de Tolstoy, 'War and Peace' version traduite en Anglais bien évidement. Ce texte a été choisi du fait de son ancienneté (publication en 1899 de la version anglaise) et donc de l'utilisation d'un vocabulaire anglais traditionnel. La longueur de l'extrait est de 1500 caractères. Nous faisons varier le nombre d'itérations de l'algorithme de 1 à 5000. Les résultats peuvent être observés à la Table 3 ci-dessous.

Itérations	Précision (%)
500	44.7
1000	61.4
1500	70.7
2000	78.0
2500	83.8
3000	85.8
3500	97.2
4000	97.8
4500	97.8
5000	97.8

TABLE 3 – Précision de décryptage en fonction du nombre d'itérations

Nous portons à présent ces données sous forme de graphiques. Ceux-ci peuvent être observés à la Figure 6.

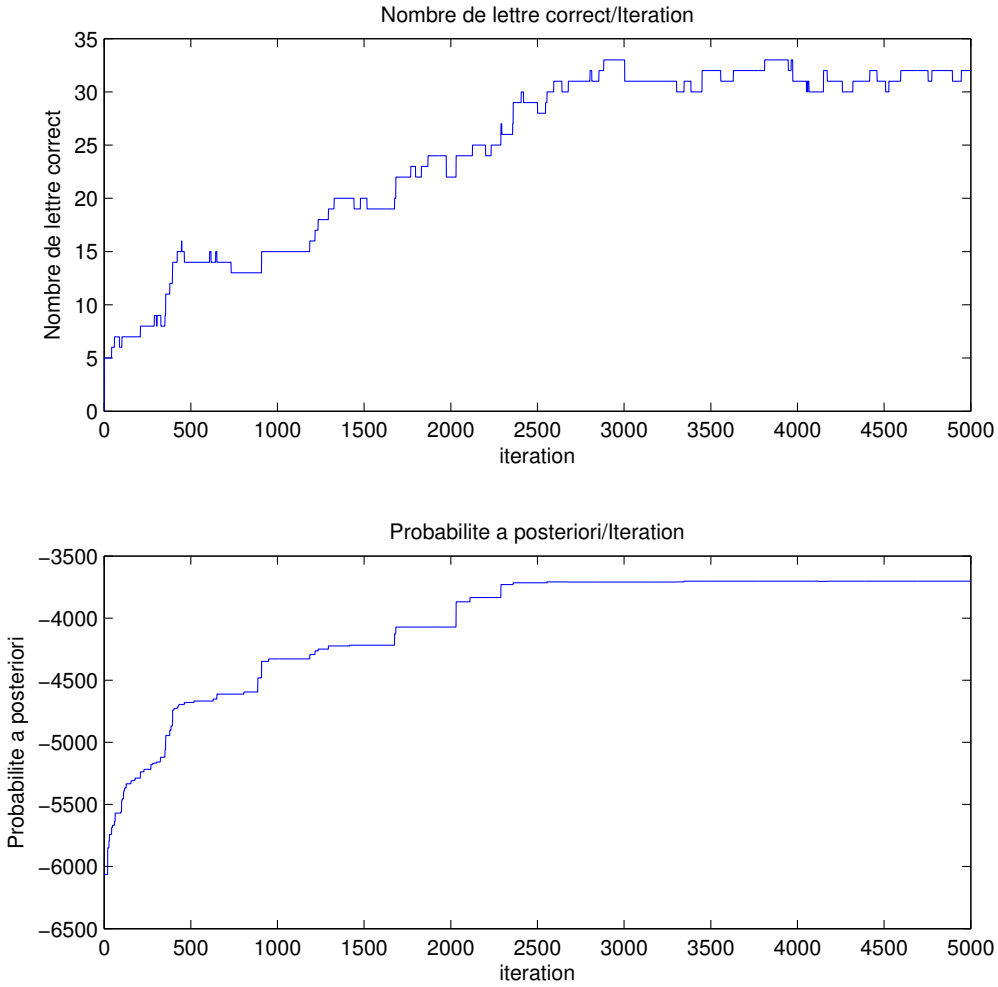

FIGURE 6 – Deux graphiques représentant la convergence de l'algorithme avec le nombre d'itération

D'une manière générale, la précision augmente avec le nombre d'itérations, comme attendu. Cependant, à partir d'environ 3500 itérations, il semblerait que l'algorithme ait fourni son résultat presque définitif. Nous pouvons observer à la Figure 6 que le nombre de symboles correctement identifiés varie encore quelque peu. En effet, certains symboles sont rarement voire pas du tout utilisés, par exemple la parenthèse qui n'est pas présente dans ce texte. L'algorithme peut alors permuter indéfiniment la parenthèse ouvrante et fermante sans modifier la probabilité à posteriori. Cela semble indiquer qu'il n'est pas nécessaire d'effectuer un nombre d'itérations trop important étant donné le faible gain comparé à l'augmentation du temps d'exécution.

Longueur du texte Nous allons à présent fixer notre nombre d'itérations à 5000 et tester la convergence de textes de longueurs différentes. Les résultats sont repris à la Table 4 et affichés à la Figure 7. Commençons par analyser cette dernière.

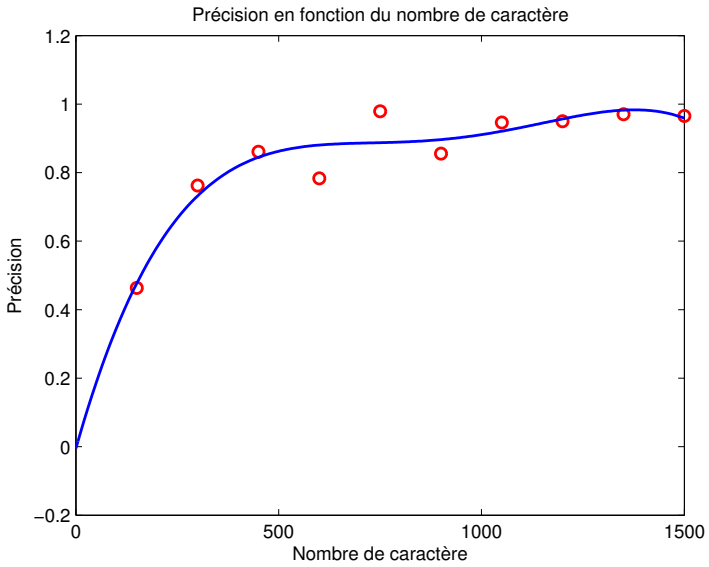

FIGURE 7 – Évolution de la précision sur le nombre de caractère

Les points rouges représentent les valeurs de la Table 4. La courbe est une régression polynomiale de ces points. En effet, le nombre de caractères correctement identifiés varie largement d'une itération à l'autre. Les points sont eux-même une moyenne de 10 exécutions. Pour aller plus loin, nous décidons de calculer l'écart-type de ces exécutions. Le résultat est présenté à la Figure 8.

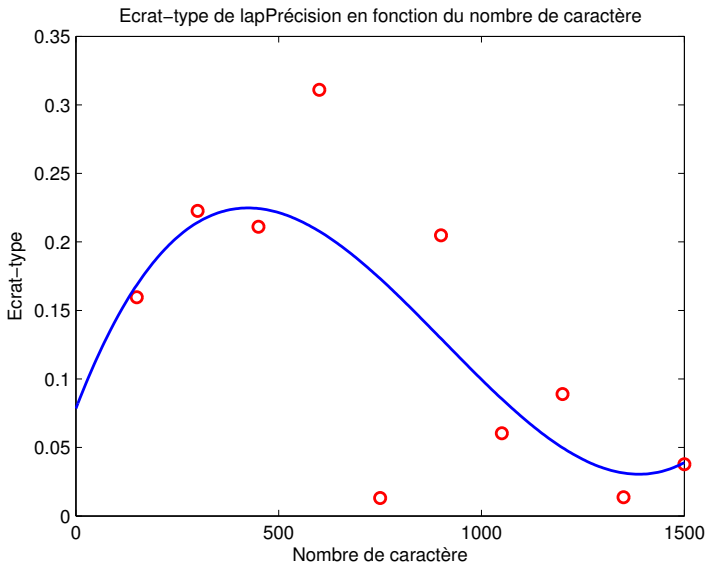

FIGURE 8 – Évolution de l'écart-type de la précision sur le nombre de caractère

Nous pouvons voir que les variations de la précision d'une exécution à une autre sont importantes, d'autant plus lorsque le nombre de symboles est petit. Revenons à présent à la table 4 et étudions plus en détails les résultats.

Nombre de symbole	Précision (%)
150	46.3
300	76.2
450	86.1
600	78.3
750	97.9
900	85.5
1050	94.6
1200	95.0
1350	97.0
1500	96.5

TABLE 4 – Précision de décryptage en fonction de la taille du texte

Comme attendu, on remarque une augmentation de la précision du décryptage avec la taille du texte. On voit également que, d'une façon générale, les textes plus courts ont plus de mal a être décryptés. Cela parait évident, plus il y a de lettres, plus la vraisemblance entre deux séquences de même taille est différente et donc plus la séquence en question est unique. Ainsi, on converge beaucoup plus facilement quand le texte à décrypter contient beaucoup de caractères.

A partir de 500 caractères, nous atteignons déjà un plateau avec un nombre important de décryptage satisfaisant, la précision n'augmente plus drastiquement avec la longueur des textes. Or, il est important de noter que plus le texte est long plus le temps d'exécution augmente. Pour des textes cryptés assez larges, par exemple plus de 5000 symboles, il serait préférable d'effectuer le décryptage par morceau en prenant des sous-textes de 1000 symboles.

État initial x0 Comme dit précédemment, le choix d'un bon état de départ est important et nous permet de converger plus vite. A la base, nous n'avions pas pris ce fait en compte. Nous nous étions dit que, comme la clé pouvait être n'importe quelle permutation aléatoire des 40 symboles, il suffisait d'en faire une pour commencer. Ce choix n'était pas des plus judicieux puisqu'on avait très peu de chance de tomber sur une clé plus ou moins vraisemblable du premier coup étant donné qu'il y avait 40 ! possibilités.

Finalement, nous avons décidé de procéder comme suit pour notre état initial : calculer les fréquences d'apparition de chaque symbole de notre texte crypté et les comparer avec celles de la langue anglaise. Lorsque la fréquence d'apparition d'un symbole du texte crypté se rapprochait de celle d'un symbole de la langue anglaise, on associait ce premier symbole au second, et ainsi de suite pour chacun des 40 symboles. Nous avons remarqué que, en partant de cette première approximation, les résultats étaient beaucoup plus vraisemblables et nécessitaient, comme attendu, moins d'itérations.

2.6 Choix de la distribution de proposition et résultats

Il est évident que le choix de la distribution de proposition a son importance. Le plus simple est, comme expliqué précédemment, de choisir une distribution de proposition symétrique de façon à simplifier le calcul du taux d'acceptation α. Nous avions premièrement décidé d'effectuer une permutation aléatoire de l'ensemble des symboles à chaque itération. Nous nous sommes vite rendus compte que ce choix n'était pas le plus judicieux puisque, si par chance nous tombions sur une clé vraisemblable, et bien cela était vain car nous régénérerions de façon complètement aléatoire une toute nouvelle clé sans prendre en compte les changements apportés à l'itération précédente. Avec cette solution, il faudrait, dans le pire des cas, parcourir toutes les permutations de Θ avant de trouver la séquence qui décrypte le texte fourni.

Nous avons alors choisi une distribution de proposition qui, immédiatement, fut beaucoup plus efficace : une substitution aléatoire, c'est-à-dire, une permutation de deux symboles choisis selon une loi uniforme. Chaque permutation a alors une probabilités de $\frac{1}{n^2}$ ou n est le nombre de symbole. On voit que cette distribution est bien symétrique. Cela nous permettait de repartir, à chaque itération, d'une clé pour laquelle la vraisemblance du texte avait,en moyenne, augmenté et donc de converger plus vite.

Le texte qui nous a été fourni comporte 1107 caractères, ce qui, comme discuté précédemment, est adapté à la convergence de la méthode. Après décryptage par notre algorithme, nous obtenons le texte suivant :

> about noon they saw a pretty snow-white bird sitting on a bough, and singing so sweetly that they stopped to listen. and when he had finished the bird spread his wings and flew before them, and they followed after him until they came to a little house, and the bird perched on the roof, and when they came nearer they saw that the house was built of bread, and roofed with cakes ; and the window was of transparent sugar. "we will have some of this," said hansel, "and make a fine meal. i will eat a piece of the roof, grethel, and you can have some of the window-that will taste sweet." so hansel reached up and broke off a bit of the roof, just to see how it tasted, and grethel stood by the window and gnawed at it. then they heard a thin voice call out from inside, "nibble, nibble, like a mouse, who is nibbling at my house ?" and the children

answered, "never mind, it is the wind." and they went on eating, never disturbing themselves. hansel, who found that the roof tasted very nice, took down a great piece of it, and grethel pulled out a large round window-pane, and sat her down and began upon it.

Le code de décryptage est alors :

a b c d e f g h i j k l m n o p q r s t u v w x y z , . ' " - ! ? : ; ( ) [ ]
- y a g s ' o w n j i d k t f l ; e (q) ! : x p . c , m h b ? [" u] z v r

Table 5 – Code de décryptage

Il est important de noter que tous nos codes de décryptages trouvés ne sont pas parfaits, certains se trompant encore sur quelques caractères. Nous avons choisi le texte décrypté qui ressortait le plus fréquemment et qui était le plus vraisemblable.

3 Bonus

La méthode MCMC peut aussi être appliquée à d'autres types de chiffrage, comme la transposition par exemple. Le principe de la transposition est de permuter des blocs de tailles aléatoires du texte original. La transposition est bien plus difficile à décrypter. En effet, dans le pire des cas, la totalité du texte peut être permuté et la cardinalité vaut, dans ce cas, la factorielle du nombre de caractères. Pour pouvoir appliquer la méthode MCMC pour déchiffrer ce cryptage, il faut remplacer la distribution de proposition q. Nous avons essayé par ailleurs une méthode de "slide move" où un bloc de taille aléatoire est placé à un endroit aléatoire, tous les autres caractères étant simplement translatés. Malheureusement, par manque de temps, nous n'avons pas pu étendre le sujet.

A Codes

A.1 Code utilisé pour répondre au point 1.1.1

```
%[Q1.1.1] Prend en argument la chaine de caractere et renvoie la matrice
%de transition Q et la distribution de probabilte pi0

function [ Q, pi0 ] = Q1_1_1( seq )

    %Pour a
    Q(1,1) = length(strfind(seq,'aa'))/(length(strfind(seq,'a'))-1);
    Q(1,2) = length(strfind(seq,'ab'))/(length(strfind(seq,'a'))-1);
    Q(1,3) = length(strfind(seq,'ac'))/(length(strfind(seq,'a'))-1);
    Q(1,4) = length(strfind(seq,'ad'))/(length(strfind(seq,'a'))-1);
    %NB:on retire -1 car la premier letter est "a"

    %Pour b
    Q(2,1) = length(strfind(seq,'ba'))/length(strfind(seq,'b'));
    Q(2,2) = length(strfind(seq,'bb'))/length(strfind(seq,'b'));
    Q(2,3) = length(strfind(seq,'bc'))/length(strfind(seq,'b'));
    Q(2,4) = length(strfind(seq,'bd'))/length(strfind(seq,'b'));

    %Pour c
    Q(3,1) = length(strfind(seq,'ca'))/length(strfind(seq,'c'));
    Q(3,2) = length(strfind(seq,'cb'))/length(strfind(seq,'c'));
    Q(3,3) = length(strfind(seq,'cc'))/length(strfind(seq,'c'));
    Q(3,4) = length(strfind(seq,'cd'))/length(strfind(seq,'c'));

    %Pour d
    Q(4,1) = length(strfind(seq,'da'))/length(strfind(seq,'d'));
    Q(4,2) = length(strfind(seq,'db'))/length(strfind(seq,'d'));
    Q(4,3) = length(strfind(seq,'dc'))/length(strfind(seq,'d'));
    Q(4,4) = length(strfind(seq,'dd'))/length(strfind(seq,'d'));

    %Pour trouver pi0
    pi0 = zeros(1,4);
    pi0(1) = length(strfind(seq,'a'))/length(seq);
    pi0(2) = length(strfind(seq,'b'))/length(seq);
    pi0(3) = length(strfind(seq,'c'))/length(seq);
    pi0(4) = length(strfind(seq,'d'))/length(seq);

end
```

A.2 Code utilisé pour répondre au point 1.1.2

```
%[Q1_1_2] Represente les differents graphes de la question 1.1.2
%distribution uniforme et donc aleatoire

function Q1_1_2 ( Q )

    t=9;
    Ppit = zeros(t,4);

    %------La premiere lettre est choisie au hasard------
    ProbPi = [.25 .25 .25 .25];
    Ppit(1,:) = ProbPi;

    for i=1:t
        Ppit(i+1,:) = ProbPi*Q^i;
    end
    %Graphe
    figure(1);
    plot(Ppit);
    xlabel('t');
    ylabel('P');
    legend('a','b','c','d');
    title('P(Xt = i), i = "a","b","c","d" et t = 10');

    %------La premiere lettre est toujours 'c'------
    ProbPi = [0 0 1 0];
    Ppit(1,:) = ProbPi;

    for i=1:t
        Ppit(i+1,:) = ProbPi*Q^i;
    end
    %Graphe
    figure(2);
    plot(Ppit);
    xlabel('t');
    ylabel('P');
    legend('a','b','c','d');
    title('P(Xt = i), i = "a","b","c","d" et t = 10');

end
```

A.3 Code utilisé pour répondre au point 1.1.3

```
%[Q1_1_3] Calcul la distribution pi a l'etat n
%Rem: Si n = 20 on est a l'etat stationnaire.

function [ ProbStat ] = Q1_1_3( Q , n )

    ProbStat = [.25 .25 .25 .25];
    ProbStat = ProbStat*Q^n;

end
```

A.4 Code utilisé pour répondre au point 1.1.4

```
%[Q1_1_4] Cree une chaine du meme type que celle dictee par "seq1",
%pi0 est la distribution stationnaire.

function [ SimuSeq ] = Q1_1_4( Q, piStat, T )

    %La premiere lettre est choisie aleatoirement selon
    %la distribution uniforme
    rnd = rand;

    if rnd < piStat(1)
        letter = 1;
    elseif rnd < piStat(2)
        letter = 2;
    elseif rnd < piStat(3)
        letter = 3;
    else
        letter = 4;
    end

    SimuSeq = zeros(1,T);
    for k = 1:T
        rnd = rand;
        if rnd < Q(letter,1)
            letter = 1;
        elseif rnd < Q(letter,1)+Q(letter,2)
            letter = 2;
        elseif rnd < Q(letter,1)+Q(letter,2)+Q(letter,3)
            letter = 3;
        else
            letter = 4;
        end
        SimuSeq(k) = letter;
    end

    %NB:On verifie que le nombre de 'd' ou 'c' de la chaine creee est
    %proche de ce que l'on trouve dans la chaine donnee. Pour ca, on
    %reconvertit les nombres en lettres
    res = char(zeros(size(SimuSeq)));
    res(SimuSeq==1) = 'a';
    res(SimuSeq==2) = 'b';
    res(SimuSeq==3) = 'c';
    res(SimuSeq==4) = 'd';
    SimuSeq = res;

    %Comparaison des probabilites
    PSimu = [sum(SimuSeq == 'a') sum(SimuSeq == 'b') sum(SimuSeq == 'c')
                 sum(SimuSeq == 'd')];
    PSimu = PSimu./length(SimuSeq);
    disp(PSimu);

end
```

A.5 Code utilisé pour répondre au point 2.2

```
%[Q2_2] Fonction prenant en arguments un texte crypte, la matrice
%de transition Q, une loi de distribution initiale pi_0 et une cle
%d'encodage et renvoyant la vraisemblance (dans la langue anglaise)
%de ce texte crypte par rapport a la cle

function [ P ] = vraisemblance(T, pinit, Q, symb)

    %Convertit notre texte en tableau de caracteres minuscules
    T = char(lower(T));

    %On fait nos calculs en log pour ne pas avoir des erreurs du aux
    %limitations de l'ordinateur en terme de calcul numerique
    %NB: rajout de exp(-15) partout pour eviter les problemes avec -inf
    pi0 = log(pinit + exp(-15));
    Q = log(Q + exp(-15));

    %Initialement
    P = pi0(strfind(symb,T(1)));

    %Calcul de la vraisemblance
    for i = 2:numel(T)
        P = P +  Q(strfind(symb,T(i-1)), strfind(symb,T(i)));
    end

end
```

A.6 Codes utilisés pour répondre au point 2.4

```
%[Q2.4]Fonction prenant en argument un texte crypte, une loi de
%distribution initiale et une matrice de transition et renvoyant
%une cle de dechiffrement grace a l'algorithme de Metropolis-Hastings.

function [probPost, best_key] = Metropolis(T,pinit,Q)

   %Import symbols ('abcdefghi...')
   symbols;

   %Nombres d'iterations
   n = 5e3;

   %Cle initiale
   key = initialKey(T, pinit, Q);

   %Matrice des permutations de y
   keyList = zeros(n,length(symb));
   keyList(1,:) = key;

   %Matrice des probabilites a posteriori
   probPost = zeros(n,1);
   probPost(1) = vraisemblance(T,pinit,Q,key);

```

```
    for i=2:n

        %Reprendre la cle precedente
        key = keyList(i-1,:);

        %Permuter 2 lettres, choisies aleatoirement
        i1 = randi([1 40]);
        i2 = randi([1 40]);
        key([i1 i2]) = key([i2 i1]);

        %Probabilite avec permutation aleatoire de symb
        probKey = vraisemblance(T,pinit,Q,key);

        %Calcul converti car les prob sont des log
        alpha = exp(probKey - probPost(i-1));

        %Acceptation
        if(rand < alpha)
            probPost(i) = probKey;
            keyList(i,:) = key;

        %Rejet
        else
            probPost(i) = probPost(i-1);
            keyList(i,:) = keyList(i-1,:);
        end

    end

    %Cle de decryptage finale
    best_key = char(keyList(i,:));

end
```

```
%Fonction prenant en arguments le texte crypte, la matrice de
%transition Q de la langue anglaise et la distribution initiale, calcule
%les probabilites stationnaires de la langue anglaise et du texte crypte,
%les compare et fais une premiere approximation de la cle d'encodage en
%fonction des probabilites similaires.

function [ key ] = initialKey(Tcrypte,pinit,Q)

    symbols;
    key = char(symb);

    %Calcul des probabilites stationnaires
    ProbStatEnglish = englishFrequency(pinit,Q);
    ProbStatCrypte = letterFrequency(Tcrypte);

    for i=1:length(symb)

        %Frequence du symbole i
        freq = ProbStatEnglish(i);

        %Indice de la lettre dont la frequence d'apparition se rapproche
        %le plus de celle d'une autre lettre dans la langue anglaise
        closestFreq = abs(ProbStatCrypte - freq);
```

```
        index = find(closestFreq == min(closestFreq),1);

        %Echanger les deux lettres et les deux probas
        key([i index]) = key([index i]);
        ProbStatCrypte([i index]) = ProbStatCrypte([index i]);
    end

end
```

```
%Fonction prenant en arguments la distribution initiale pi_0 et la matrice
%de transition Q et renvoyant la distribution stationnaire pi_inf de la
%langue anglaise en general.

function [ PiInf ] = englishFrequency( pinit, Q )

    PiInf = pinit*Q^20;
    PiInf = transpose(PiInf);

end
```

```
%Fonction prenant en argument le texte crypte et renvoyant la frequence
%d'apparition de chaque symbole dans ce texte

function [frequency] = letterFrequency(Tcrypte)

    symbols;%load symb
    Tcrypte = char(lower(Tcrypte));

    T_length = length(Tcrypte);

    frequency = zeros(1,length(symb));

    for i=1:length(symb)
        nb = length(strfind(Tcrypte,symb(i)));
        frequency(i) = nb/T_length;
    end

end
```

A.7 Codes utilisés pour répondre au point 2.5

```
%Fonction prenant en arguments un texte et une cle de cryptage et
%renvoyant le texte crypte selon cette cle.

function [ Tcrypte ] = encrypt( T, key )

    symbols;
    T = char(lower(T));
    Tcrypte = '';

    for i=1: numel(T)
```

```
        index = strfind(symb,T(i));
        Tcrypte = [Tcrypte key(index)];
    end

end
```

```
%Fonction prenant en arguments un texte crypte et une cle de decryptage
%renvoyant le texte decrypte selon cette cle.

function [ Tdecrypt ] = decrypt( Tcrypte, key )

    symbols;
    Tcrypte = char(lower(Tcrypte));
    Tdecrypt = '';

    for i=1: numel(Tcrypte)
        index = strfind(key,Tcrypte(i));
        Tdecrypt = [Tdecrypt symb(index)];
    end

end
```

```
%Script de test de l'algorithme en fonction de textes de longueurs
%differentes

clc;
clear;
load('Q.mat');
load('pinit.mat');
symbols;

T = ' ... ';%Plusieurs textes testes
key = symb(randperm(length(symb)));
Tcrypte = encrypt(T,key);
[probPost, best_key] = Metropolis(Tcrypte,pinit,Q);
Tdecrypt = decrypt(Tcrypte,best_key);

%Affichage des resulats a la console
fprintf('\n**********RESULTATS**********\n');
fprintf('Code de chiffrement trouve= %s\n', best_key);
fprintf('Code de chiffrement attendu= %s\n', key);
fprintf('Texte decrypte= %s\n',  Tdecrypt);
fprintf('Texte original= %s\n', T);

%Graphique des prob a posteriori en fonction de l'iteration
plot(probPost);
xlabel('Iteration');
ylabel('Probabilite a posteriori');
title('Probabilite a posteriori/Iteration');
```

B Références

[1] Stephen Connor. *Simulation and Solving Substitution Codes*, mai 2001.

[2] Jian Chen and Jeffrey S. Rosenthal. *Decrypting Classical Cipher Text Using Markov Chain Monte Carlo*, mai 2010.

[3] https ://stats.stackexchange.com/posts/47691/revisions