Université de Liège

Ingénieur Civil - 3ème BAC

5 mai 2020

Éléments de processus stochastiques

Méthodes de Monte Carlo par chaines de Markov

LIONE Maxime - s175606
MUKOLONGA Jean-David - s170679
SIBOYABASORE Cédric - s175202

1 Première partie : chaines de Markov et algorithme MCMC

1.1 Chaines de Markov

1.1.1

Si le premier état est tiré dans une loi uniforme, alors la distribution de probabilité initiale π_0 est telle que

$$\pi_0(x) = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$$

Si l'état initial est toujours 3, la probabilité initiale de l'état 3 vaut 1 tandis que les probabilités initiales des autres états sont nulles. π_0 est telle que

$$\pi_0(x) = \delta_{x3}\,^1 = (0, 0, 1, 0)$$

En outre, la distribution de probabilité $P(X_t)$ à l'instant t se calcule telle que

$$P(X_t) = \pi_t = \pi_0 * Q^t$$

Grâce à cette formule, dans le script *q1_1_1.m*, on calcule dans les deux cas (π_0 uniforme et $\pi_0 = \delta_{x3}$) les distributions $P(X_t)$ en chaque instant t jusqu'en $t = 30$ dans une matrice 30 x 4 où les colonnes représentent les 4 états et les lignes représentent les distributions. Voici les représentations de l'évolution des distributions dans les deux cas, où on peut voir une courbe par état :

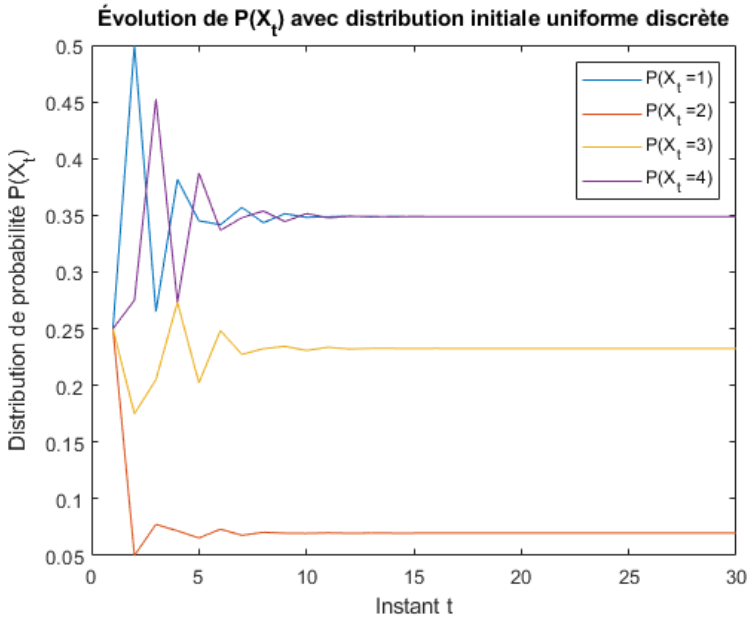


FIGURE 1

1. δ_{ij} représente le symbole de Kronecker

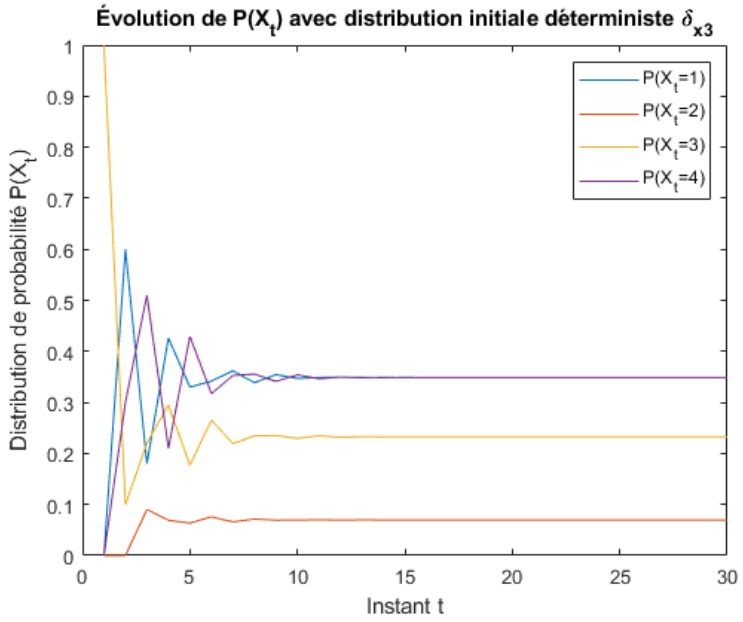


FIGURE 2

On remarque sur nos graphiques ci-dessus que peu importe la distribution initiale, la distribution de probabilités converge dans les deux cas vers une même distribution, une distribution où la chaîne a la même probabilité de se trouver dans l'état 1 que dans l'état 4. On observe sur les graphiques que la distribution de probabilité commence à converger vers $t = 12$. Lorsque $t = 12$, la t-ième puissance de la matrice de transition vaut

$$Q^{12} = \begin{pmatrix} 0.3496 & 0.0697 & 0.2329 & 0.3478 \\ 0.3474 & 0.0700 & 0.2328 & 0.3497 \\ 0.3483 & 0.0699 & 0.2328 & 0.3490 \\ 0.3488 & 0.0697 & 0.2320 & 0.3496 \end{pmatrix}$$

Lorsque $t = 30$,

$$Q^{30} = \begin{pmatrix} 0.3488 & 0.0698 & 0.2326 & 0.3488 \\ 0.3488 & 0.0698 & 0.2326 & 0.3488 \\ 0.3488 & 0.0698 & 0.2326 & 0.3488 \\ 0.3488 & 0.0698 & 0.2326 & 0.3488 \end{pmatrix}$$

La matrice Q ne varie presque pas de $t = 12$ à $t = 30$. De plus, ses lignes sont quasiment égales en $t = 12$ et elles le sont parfaitement en $t = 30$. Ceci explique pourquoi sur les graphiques, la distribution de probabilité $P(X_t) = \pi_t = \pi_0 * Q^t$ converge vers la même distribution.

Les lignes de Q^{30} correspondent en fait à la distribution stationnaire π_∞, distribution vers laquelle la chaîne tend quand $t \to \infty$.

1.1.2

Dans le script *q1_1_2.m*, en calculant la 30-ième puissance de la matrice de transition, on a pu trouver la distribution stationnaire

$$\pi_\infty = (0.3488, 0.0698, 0.2326, 0.3488)$$

On peut aussi trouver cette distribution d'une manière un peu plus théorique. En effet, par définition, la distribution stationnaire π_∞ d'une chaîne de Markov est un vecteur propre à gauche de la matrice de transition Q associé à une valeur propre unitaire $\lambda = 1$. Grâce à la fonction *eig* de Matlab utilisée sur la matrice Q, on trouve 4 valeurs propres dont la valeur propre unitaire $\lambda = 1$ et le vecteur propre à gauche qui y est associé

$$(0.6344, 0.1269, 0.4230, 0.6344).$$

En normalisant ce vecteur, on obtient le vecteur

$$\pi_\infty = (0.3488, 0.0698, 0.2326, 0.3488)$$

qui correspond bien à la distribution stationnaire trouvée précédemment. Elle vérifie

$$(0.3488, 0.0698, 0.2326, 0.3488) * Q = (0.3488, 0.0698, 0.2326, 0.3488).$$

1.1.3

Comme il vient d'être précisé à la question précédente, la distribution stationnaire vérifie

$$\pi_\infty = \pi_\infty * Q$$

Par ailleurs, on sait qu'une propriété importante d'une distribution stationnaire est que, si la chaîne de Markov est initialisée selon cette distribution, alors elle y restera pour tous les instants suivants. Ainsi, générer une réalisation aléatoire de longueur T de la chaîne de Markov en démarrant d'un état choisi aléatoirement selon la distribution stationnaire revient à générer chaque état de la chaîne selon cette distribution.

Pour ce faire, dans le script *q1_1_3*, on commence par établir une échelle allant de 0 à 1, divisée en 4 parties reflétant la distribution stationnaire π_∞ (où les variables *first*, *second*, *third* et *forth* déterminent les gradutations de l'échelle) :

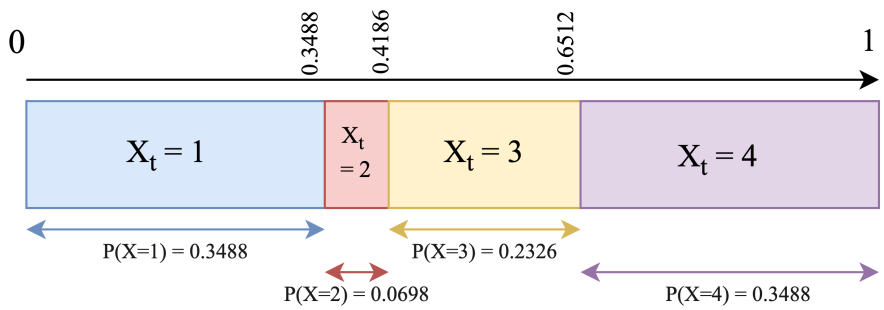


FIGURE 3

Pour générer un état, il suffit alors de générer un nombre aléatoire entre 0 et 1 et lui associer l'état correspondant sur l'échelle. Les états de la réalisation aléatoire de la chaine de Markov ainsi obtenus suivent la distribution π_∞.

En calculant les fréquences d'apparition des différents états pour une réalisation, on remarque qu'elles tendent vers la distribution stationnaire et ce, d'autant plus précisément lorsque T croît. Par exemple, pour une réalisation aléatoire de longueur T = 1000, on obtient les fréquences suivantes :

$$[freq_{x_1} = 0.3170,\ freq_{x_2} = 0.0790,\ freq_{x_3} = 0.2520,\ freq_{x_4} = 0.3520\ ]$$

Pour une réalisation aléatoire de beaucoup plus grande longueur, par exemple T = 3000000, on obtient la distribution de fréquence visible sur la Figure suivante. On constate en effet que la distribution de fréquences tend vers la distribution stantionnaire.

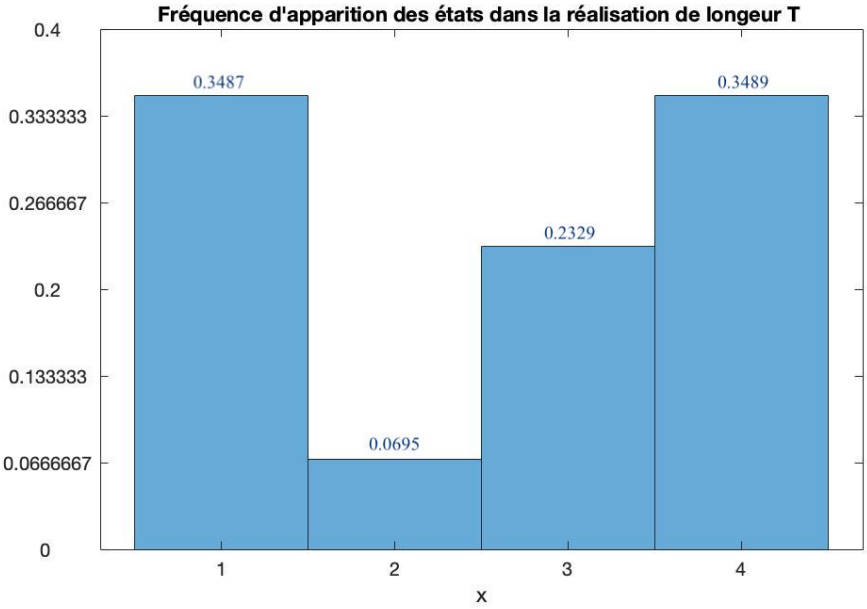


FIGURE 4

1.1.4

On a remarqué que la distribution tendait vers la même distribution stationnaire π_∞ aussi bien pour une distribution initiale uniforme que pour une distribution initiale déterministe. On en conclut que la limite $\lim_{n\to+\infty} \pi_n = \lim_{n\to+\infty} \pi_0 Q^n$ tend uniformément vers π_∞ quelque soit la distribution initiale.

Cette propriété a également été vérifiée lorsqu'on a généré des réalisations de la chaîne après avoir initialisé la distribution initiale selon la distribution stationnaire : l'histogramme des états occupés rapporté à la longueur de la chaîne est conforme à la distribution stationnaire. Ceci est dû au fait que si la chaîne de Markov est initialisée selon la distribution stationnaire π_∞, alors elle y restera pour tous les instants suivants.

1.2 Méthode MCMC : analyse théorique dans le cas fini

1.2.1

Une distribution est stationnaire si en appliquant la distribution à la matrice de transition on obtient la même distribution. Montrons que la distribution π_0 est stationnaire si elle satisfait aux équations de balance détaillée. C'est a dire si

$$\pi_0(i)[Q]_{i,j} = \pi_0(j)[Q]_{j,i} \forall i, j \in \{1, \ldots, N\}$$

Commençons par calculer la distribution après avoir appliquer la matrice de transition une fois. On obtient

$$\pi_1(i) = \sum_{j=1}^{N} \pi_0(j)[Q]_{j,i}$$

En utilisant les équations de balance détaillée, π_1 devient :

$$\pi_1(i) = \sum_{j=1}^{N} \pi_0(i)[Q]_{i,j}$$

On s'aperçoit qu'on peut faire sortir $\pi_0(i)$ car il ne depend plus de j :

$$\pi_1(i) = \pi_0(i) \sum_{j=1}^{N} [Q]_{i,j}$$

Vu la nature de la matrice de transition, la somme des éléments d'une ligne est égale à 1. Ainsi, nous obtenons :

$$\pi_1(i) = \pi_0(i)$$

La distribution π_0 est donc bien une distribution stationnaire si elle satisfait aux équations de balance détaillée. De plus, on peut remarquer que la distribution π_0 est un vecteur propre de la matrice de transition.

$$\pi_0(i) = \pi_0(i)[Q]_{i,j}$$

Elle est donc unique si la valeur propre associée est non-dégénérée.

1.2.2

La probabilité de transition entre l'état x (en t-1) et y (en t) peut s'écrire comme étant le produit entre la densité de proposition et la probabilité α d'accepter l'état y, c'est-à-dire tel que :

$$Q(x, y) = P(y|x) = \alpha\, q(y|x) = \min\left\{1, \frac{f_X(y)}{f_X(x)} \frac{q(x|y)}{q(y|x)}\right\} q(y|x)$$

Similairement, la probabilité de transition de l'état y à l'état x s'écrit :

$$Q(y, x) = P(x|y) = \alpha\, q(x|y) = \min\left\{1, \frac{f_X(x)}{f_X(y)} \frac{q(y|x)}{q(x|y)}\right\} q(x|y)$$

• Si $f_X(y)\,q(x|y) > f_X(x)\,q(y|x)$,

$$Q(x,y) = q(y|x)$$

$$Q(y,x) = \frac{f_X\,(x)}{f_X\,(y)}\frac{q\,(y|x)}{q\,(x|y)}q(x|y)$$

On a donc $Q(y,x) = \frac{f_X(x)}{f_X(y)}q\,(y|x) = \frac{f_X(x)}{f_X(y)}Q(x,y) \Leftrightarrow f_X\,(x)\,Q(x,y) = f_X\,(y)\,Q(y,x)$.

• Si $f_X(x)\,q(y|x) > f_X(y)\,q(x|y)$,

$$Q(x,y) = \frac{f_X\,(y)}{f_X\,(x)}\frac{q\,(x|y)}{q\,(y|x)}q(y|x)$$

$$Q(y,x) = q(x|y)$$

On a donc $Q(x,y) = \frac{f_X(y)}{f_X(x)}q\,(x|y) = \frac{f_X(y)}{f_X(x)}Q(y,x) \Leftrightarrow f_X\,(x)\,Q(x,y) = f_X\,(y)\,Q(y,x)$.

Dans les deux cas, on finit avec la même équation. Si on remplace f_X par cp_X, l'équation devient

$$cp_X(x)Q(x,y) = cp_X(y)Q(y,x)$$

On remarque qu'on peut simplifier la constante c dans les deux membres. On arrive donc à l'équation de balance détaillée

$$p_X(x)Q(x,y) = p_X(y)Q(y,x)$$

avec p_X qui est donc la distribution stationnaire de la chaîne de Markov générée.

Pour que l'algorithme de Metropolis-Hastings fonctionne, il faut que la chaîne soit ergodique, c'est-à-dire irréductible et apériodique, afin que la distribution stationnaire p_X soit unique et afin d'avoir une convergence vers celle-ci.

1.3 Méthode MCMC : illustration sur le modèle d'Ising unidimensionel

1.3.1

La méthode de Metropolis-Hastings est censée être capable de simuler une chaine de Markov ergodique dont la distribution stationnaire est égale à P_{S^N}. Pour ce faire, il faut que la distribution de proposition q soit telle que la chaîne de Markov vérifie les conditions explicitées à la section précédente :

Tout d'abord, la chaîne de Markov doit être à espace d'états fini. On sait que c'est le cas pour le modèle d'Ising car le système est composé de N particules pouvant prendre 2 orientations possibles.

De plus, on sait que la probabilité de passer d'un état à un état voisin pour lequel une seule particule a changé d'orientation vaut $\frac{1}{N}$. Par conséquent, il est possible d'atteindre n'importe quel état à partir d'un autre en un nombre fini de pas. Ainsi, la distribution de proposition q induit bien que la chaîne de Markov est irréductible.

Par ailleurs, elle est apériodique aussi vu qu'il y a toujours une probabilité non nulle de rester dans le même état d'une itération à l'autre, étant donné que la matrice de transition de l'algorithme ne correspond pas parfaitement à la densité de proposition. Étant irréductible et apériodique, la chaîne de Markov générée sera donc ergodique.

Finalement, vu la section **1.2.2**, on sait que l'algorithme de Metropolis-Hastings ne nécessite que la connaissance de p_x à une constante multiplicative près et qu'il permet de simuler une chaîne de Markov ergodique qui satisfait les équations de balance détaillée. Vu que la distribution de proposition q est bien telle que la chaîne de Markov vérifie les conditions données en **1.2.2** pour que l'algorithme de Metropolis-Hastings fonctionne, on sait que ce dernier permettra bien de générer des échantillons selon la distribution P_{S^N}.

1.3.2

La probabilité α d'acceptation d'un nouvel état $(s'_1, \ldots, s'_N)$ s'écrit :

$$\alpha = \frac{P_{S'^N}\left(s'_1, \ldots, s'_N\right)}{P_{S^N}\left(s_1, \ldots, s_N\right)} \frac{q\left(s'_1, \ldots, s'_N | s_1, \ldots, s_N\right)}{q\left(s_1, \ldots, s_N | s'_1, \ldots, s'_N\right)}$$

Or, pour passer de l'état $(s_1, \ldots, s_N)$ à l'état $(s'_1, \ldots, s'_N)$, seule une particule i a changé de valeur. La densité de proposition est donc symétrique :

$$q\left(s_1, \ldots, s_N | s'_1, \ldots, s'_N\right) = q\left(s'_1, \ldots, s'_N | s_1, \ldots, s_N\right) = \frac{1}{N}$$

La probabilité d'acceptation peut alors se réécrire telle que :

$$\begin{aligned}
\alpha &= \frac{P_{S'^N}\left(s'_1, \ldots, s'_N\right)}{P_{S^N}\left(s_1, \ldots, s_N\right)} \\
&= \frac{\frac{1}{Z}\exp\left(-\beta E\left(s'_1, \ldots, s'_N\right)\right)}{\frac{1}{Z}\exp\left(-\beta E\left(s_1, \ldots, s_N\right)\right)} \\
&= \exp\left(-\beta E\left(s'_1, \ldots, s'_N\right) + \beta E\left(s_1, \ldots, s_N\right)\right) \\
&= \exp\left(-\beta \Delta E\right)
\end{aligned}$$

où on note $\Delta E = E\left(s'_1, \ldots, s'_N\right) - E\left(s_1, \ldots, s_N\right)$ la différence d'énergie entre les deux états.

Les termes $s_{l(i)}s_i$ et $s_{r(i)}s_i$ font partie de la somme exprimée dans le premier terme de

$$E\left(s_1, \ldots, s_N\right) = -J\left(\sum_{j=1}^{N-1} s_j s_{j+1} + s_1 s_N\right) - H\sum_{j=1}^{N} s_j$$

et le terme s_i fait partie de la somme exprimée dans le deuxième terme (avec $1 \leq i \leq n$).

Pour passer du premier état au deuxième, on a uniquement changé la valeur de la particule i telle que $s'_i = -s_i$ (vu que -1 et 1 sont les deux seules valeurs possibles), les autres particules ont gardé leurs valeurs. Les termes $s_{l(i)}s'_i$ et $s_{r(i)}s'_i$ font donc parties du premier terme de l'énergie du nouvel état et le terme s'_i fait partie du deuxième terme.

Dans le calcul de la différence d'énergie entre les deux états, les termes restés identiques d'un état à l'autre se soustraient et on obtient :

$$\Delta E = -J\left(s_{l(i)} * s'_i + s'i * s_{r(i)} - (s_{l(i)} * s_i + si * s_{r(i)})\right) - H\left(s_i - (s'_i)\right)$$

Or, $s'_i = -s_i$. La différence d'énergie entre les deux états devient donc :

$$\Delta E = -J(-2s_{l(i)} * s_i - 2si * s_{r(i)}) + 2Hs_i$$
$$\Leftrightarrow \Delta E = 2 * s_i\left(J(s_{l(i)} + s_{r(i)}) + H\right)$$

On a donc : $\alpha = \exp\left(-2\beta s_i\left(J(s_{l(i)} + s_{r(i)}) + H\right)\right)$.

Finalement, vu que α représente une probabilité, elle doit être comprise entre 0 et 1. Or, si $\Delta E < 0$ (si l'énergie diminue d'un état à l'autre), $\alpha = \exp(-\beta\Delta E)$ sera plus grand que 1, c'est-à-dire qu'on acceptera avec certitude le nouvel état. Pour être sûr que α ne soit jamais plus grand que 1, on assignera 1 à α si $\Delta E < 0$.
On exprime finalement la probabilité d'acceptation du nouvel état telle que :

$$\alpha = \min\left\{1, \exp\left(-2\beta s_i\left(J\left(s_{l(i)} + s_{r(i)}\right) + H\right)\right)\right\}$$

1.3.3

Dans le script *q1_3_3.m*, on initialise d'abord un vecteur ligne s de N=10 colonnes, représentant le système d'Ising à une dimension contenant 10 particules. On remplit ce vecteur par des valeurs aléatoires entre 0 et 1 grâce à la fonction *rand* de Matlab (qui génère aléatoirement de manière uniforme un nombre entre 0 et 1) puis on remplace les valeurs de ce vecteur plus petites que 0.5 par -1 et les autres par 1. On fixe arbitrairement l'état initial x(t=1) de la réalisation de la chaîne de Markov à ce vecteur s. Ce choix a peu d'importance dans la suite car la chaîne est suffisamment longue (de longueur 10000).

Dans une boucle itérant $10000 - 1 = 9999$ fois (vu qu'on a initialisé la première réalisation à s), on sélectionne aléatoirement un indice i correspondant à une particule grâce à la fonction *randsample* qui pioche un nombre entre 1 et $N = 10$ de manière uniforme ($q = \frac{1}{N}$). Ensuite, on calcule la probabilité α d'acceptation d'un nouvel état dans lequel on aurait modifié la valeur de la particule i de s tel que démontré au point **1.3.2**. Grâce à la fonction *rand* de Matlab, on obtient le nombre u qu'on compare avec α. Si u est plus petit qu'α, on accepte le nouvel état proposé ; on change l'orientation de la particule en multipliant sa valeur par -1 et la réalisation $x(t)$ au temps t est ce nouvel état. Sinon, on ne change rien et $x(t)$ est la même qu'au temps $t - 1$. À chaque itération t, on calcule la nouvelle magnétisation moyenne qu'on sauvegarde dans un vecteur $magn_m$. On affiche le contenu de ce vecteur une fois la boucle terminée.

Sur la *Figure 5*, on peut voir que la magnétisation moyenne converge, dès $t \sim= 2000$, autour de sa valeur exacte 0.6887 retournée par $ising1Dexact$, ce qui confirme la théorie.

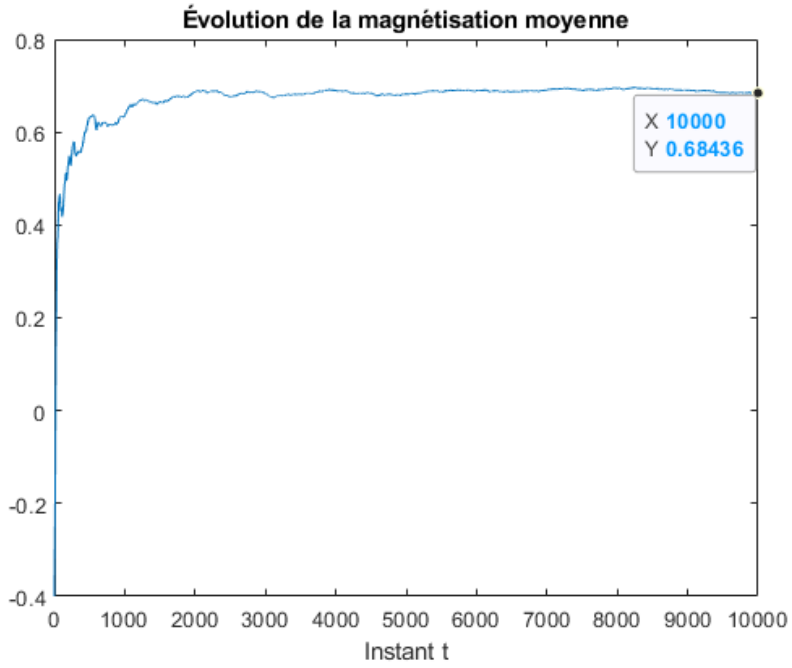


FIGURE 5

1.3.4

Dans le script *q1_3_4.m*, on répète 100 fois l'expérience du point **1.3.3**. À chaque fin d'expérience, on additionne les magnétisations moyennes à un vecteur *mean_m* contenant la somme des magnétisations moyennes des expériences précédentes puis, après la 100ème expérience, on divise ce vecteur par 100 pour obtenir la moyenne des magnétisations moyennes sur les 100 expériences.
En affichant le contenu de ce vecteur, on observe sur la *Figure 6* (à la page suivante) qu'en moyenne, sur les 100 expériences, l'estimation de la magnétisation moyenne converge vers la valeur exacte 0.6887.

En effet, en moyenne, seuls 5 % des magnétisations moyennes se trouvent en-dessous de 0.6860 (5ème centile, retourné par la fonction *prctile* de Matlab), ce qui montre que l'estimation de la magnétisation moyenne atteint tôt les alentours de 0.6887. De plus, en moyenne, 95 % des magnétisations moyennes se trouvent en-dessous de 0.6899 (95ème centile), ce qui signifie que, pour t allant de 1 à 10000, 90 % des magnétisations moyennes se trouvent entre 0.6860 et 0.6899, ce qui montre que l'estimation de la magnétisation moyenne converge en moyenne vers 0.6887, et ce, assez tôt.

Avoir des résultats similaires pour une seule expérience (voir point **1.3.3**) et pour la moyenne de 100 expériences nous permet de conclure que l'estimation de la magnétisation moyenne obtenue à l'aide de l'algorithme de Metropolis-Hasting converge avec certitude vers la valeur exacte 0.6887 de la magnétisation moyenne.

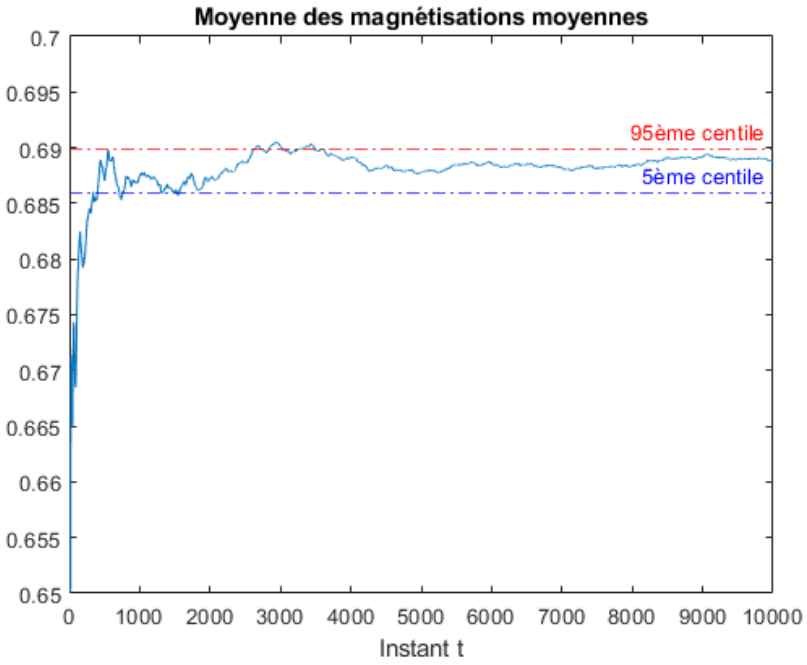


FIGURE 6

1.3.5

Pour chaque H appartenant à l'intervalle [-20, 20], on effectue comme au point **1.3.3** une réalisation de la chaîne de Markov de longueur 10000 avec $\beta = 0.2$ et on calcule la magnétisation moyenne à chaque itération. Puis, en $t = 10000$, instant où la magnétisation moyenne se rapproche de/converge vers sa valeur exacte, on met la magnétisation moyenne dans un vecteur $magn\_m1$. Ce dernier sera de taille 41 car il contiendra les magnétisations moyennes à l'instant $t = 10000$ pour chaque H.

On calcule ensuite grâce à $ising1Dexact$ les valeurs exactes de magnétisation moyenne pour chaque H qu'on met toutes dans un vecteur $exact\_magn1$.

On observe sur la *Figure 7* que pour tout H appartenant à l'intervalle [-20, 20], la courbe correspondant aux magnétisations moyennes estimées (du vecteur $magn\_m1$) se superpose presque complètement avec la courbe exacte calculée par $ising1Dexact$. Quand on soustrait les vecteurs $magn\_m1$ (estimations) et $exact\_magn1$ (valeurs exactes), la plus grande différence n'est que de 0.0197, ce qui appuie la qualité des estimations de l'algorithme de Metropolis-Hasting.

On trace la même courbe pour $\beta = 0.02$, représentée à la *Figure 8* . On observe qu'elle se superpose également avec la courbe exacte ; la plus grande différence entre les estimations et les valeurs exactes est de 0.0294.
On fait tout cela dans le script *q1_3_5.m*

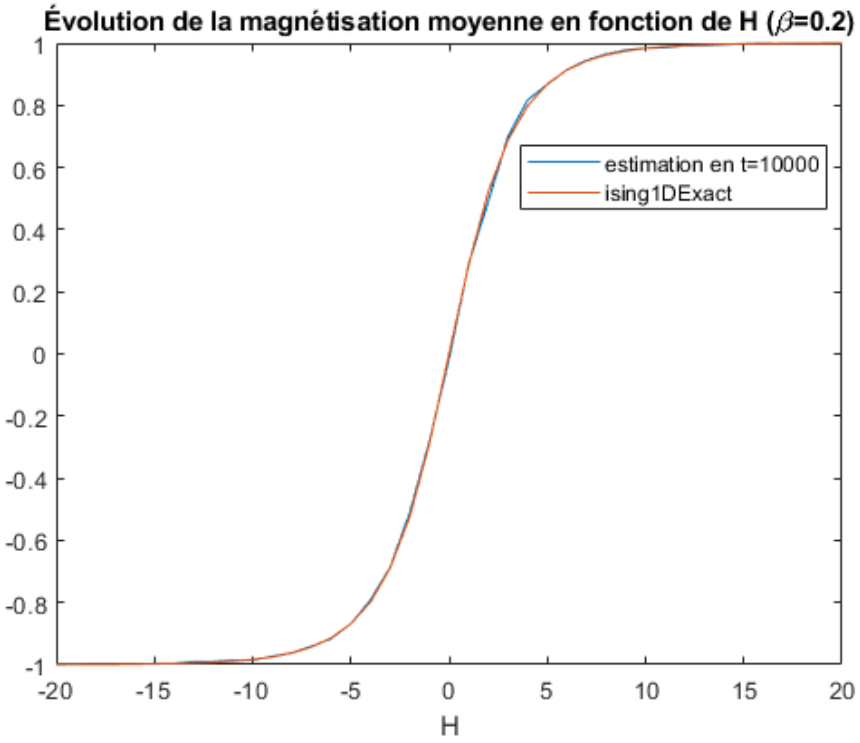


FIGURE 7

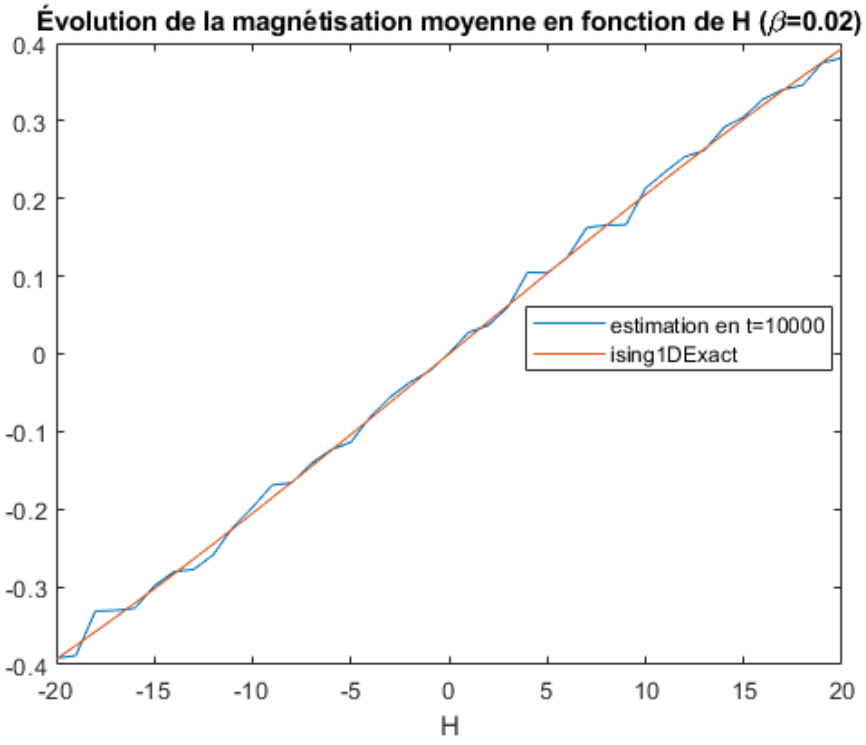


FIGURE 8

2 Deuxième partie : débruitage d'images binaires à l'aide de la méthode MCMC

2.1 Débruitage d'images binaires : description du problème

Le problème d'optimisation qu'on a choisi est celui du débruitage d'images binaires.

On a une image binaire I [2] de dimensions N x M représentée par une matrice M x N (voir *Figure 9*) , dont chacun des éléments est un pixel qui vaut soit 255 pour un pixel blanc, soit 0 pour un pixel noir. Une image I' (voir *Figure 6*) est obtenue en appliquant à I un bruit dont on connaît le processus, c'est-à-dire un bruit qu'on a construit artificiellement de la manière suivante :

Soit I_{ij} le pixel à la position (i, j) $(0 < i \leq M, 0 < j \leq N)$ dans l'image. Le pixel homologue I'_{ij} dans l'image bruitée est tel que

$$I'_{ij} = \begin{cases} \overline{I}_{ij} & \text{avec une probabilité } p \\ I_{ij} & \text{avec une probabilité } 1-p \end{cases}$$

où $\overline{I}_{ij}$ est la couleur complémentaire de I_{ij} ($\overline{I}_{ij}$ est noir si I_{ij} est blanc, blanc si I_{ij} est noir). Plus p est grand, plus l'image est bruitée. Dans la suite, on prendra $p = 0.1$. [3] Le but du problème est de reconstruire l'image originale non-bruitée I à partir de l'image bruitée I'.

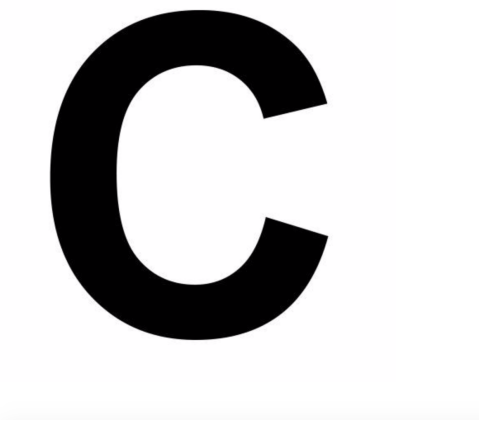


FIGURE 9 – Image non-bruitée I

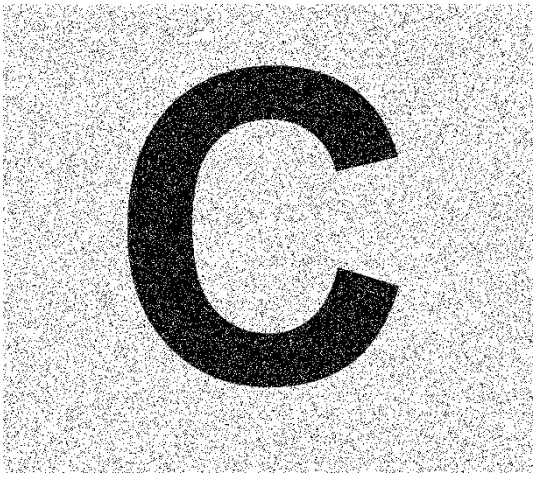


FIGURE 10 – Image bruitée I' avec p=0.1

2. Prise de https ://steamcommunity.com/sharedfiles/filedetails/ ?l=frenchid=903417513 . L'image n'est pas parfaitement binaire donc on a utilisé la fonction *im2bw* de Matlab pour qu'elle le soit parfaitement.

3. Bien entendu, un tel type de bruit, qui bruite une image binaire et la garde binaire, est peu réaliste. Dans la vraie vie, un bruit numérique est souvent modélisé comme étant un bruit additif Gaussien. Dans ce cas, l'image bruitée n'est plus binaire et le problème d'optimisation n'est plus combinatoire. Pour illustrer la théorie de la première partie du projet, on ne considère que le cas discret, donc on se limite à un bruitage binaire et à une image bruitée binaire.

En faisant une analogie avec un modèle d'Ising, On peut reformuler le problème :

Soit I l'image non-bruitée constituée par M x N variables aléatoires $\{I_{11}, ..., I_{MN}\}$ représentant les pixels (// *particules*) et prenant chacune leur valeur dans l'espace discret $D = \{-1, 1\}$ où -1 représenterait par exemple la couleur noire et +1 la couleur blanche (// *orientation des particules*).

Après application sur I du bruit décrit ci-dessus, on obtient l'image bruitée $I' = \{I'_{11}, ..., I'_{MN}\}$ dont les M x N variables aléatoires prennent également valeur dans $D = \{-1, 1\}$.

Similairement à un modèle de Markov caché, on essaie de reconstruire les variables cachées I en observant les variables I'. On essaie de trouver dans l'ensemble S fini contenant toutes les associations de M x N pixels binaires possibles l'unique association de pixels I^* correspondant à l'image originale non-bruitée. Cette dernière correspond à l'image I qui maximise la probabilité conditionnelle $P(I|I')$, soit

$$I^* = \arg_{I \in S} \max P(I|I').$$

Grâce au théorème de Bayes, on peut écrire

$$I^* = \arg_{I \in S} \max P(I|I') = \arg_{I \in S} \max \frac{P(I'|I)P(I)}{P(I')} = \arg_{I \in S} \max P(I'|I)P(I)$$

où $P(I'|I)$ est la probabilité p avec laquelle on change la couleur d'un pixel de I.

En tenant compte de l'analogie effectuée avec le modèle d'Ising et en postulant que l'image originale est telle qu'un pixel d'une couleur est généralement entouré de pixels de la même couleur, on peut utiliser une probabilité a priori $P(I)$ proche de celle du modèle d'Ising, c'est-à-dire telle que $P(I) = \frac{1}{Z} \exp\left(-\beta E\left(I = \{I_1, \ldots, I_{MxN}\}\right)\right)$.

Le terme E, qui représente l'Énergie du système dans le modèle d'Ising va ici représenter ce qu'on appellera la fonction d'Équilibre des pixels. C'est notre fonction objectif. Le calcul de E va légèrement différer de sa définition dans le modèle d'Ising car dans le cas d'une image, un pixel (// *particule*) a 4 pixels voisins (sauf ceux aux coins de l'image qui en ont 2 et ceux sur les bords qui en ont 3) et non 2. De plus, une suite de pixels n'est pas circulaire comme l'est le modèle d'Ising donc il n'y a pas de terme dans E qui exprime la fermeture circulaire d'une séquence de pixels. On a alors :

$$E(I) = -J * \sum_{i=1}^{N-1} \sum_{j=1}^{N-1} I_{ij} * I_{voisins\,de\,ij} - H * \sum_{i=1}^{N} \sum_{j=1}^{N} I_{ij}$$

Or, un H non-nul signifierait que l'on veut une image à couleur uniforme : si $H < 0$, les pixels tendront à avoir la valeur -1 et l'image sera blanche tandis que si $H > 0$, les pixels tendront à valoir 1 et l'image sera noire. On choisit donc de nier l'existence de ce terme qui, dans le modèle d'Ising, exprime l'existence d'un champ magnétique extérieur (un tel phénomène n'aurait pas de sens dans le débruitage d'images). On a alors $H = 0$ et l'expression de notre fonction d'équilibre devient :

$$E(I) = -J * \sum_{i=1}^{N-1} \sum_{j=1}^{N-1} I_{ij} * I_{voisins\,de\,ij}$$

En utilisant une $P(I)$ proche de la distribution du modèle d'Ising, on a choisi une distribution de probabilité semblable à la distribution $p(I) = Ce^{-\beta f(I)}$ proposée dans l'énoncé où $C = \frac{1}{Z}$, les β se correspondent. La fonction objectif $f(I)$ est la fonction d'équilibre $E(I)$. On va alors utiliser l'algorithme de Metropolis-Hastings pour échantillonner selon cette distribution, c'est-à-dire l'algorithme de recuit simulé pour trouver une solution optimale à notre problème.

2.2 Cardinalité de l'ensemble S de solutions

Une image binaire de taille M x N possède M x N pixels qui prennent soit la valeur 0 (noir) soit la valeur 255 (blanc). L'ensemble S de solutions contient donc 2^{MxN} images-différentes possibles, parmi lesquels se trouve l'image originale non-bruitée. Une recherche exhaustive (qui consisterait à générer toutes les associations de pixels possibles jusqu'à tomber sur celle correspondant à l'image non-bruitée) ne serait pas envisageable. En effet, dans notre cas, l'image est de dimensions 620 x 545, ce qui veut dire que la cardinalité de S est de $2^{545x620} = 2^{337900}$! Même pour une version réduite de l'image à 250 x 220, une approche par force brute serait irréalisable !

On va donc utiliser l'algorithme de Metropolis-Hasting pour trouver une solution approchée au problème.

2.3 Implémentation de l'algorithme de Metropolis-Hastings

On implémente l'algorithme dans un script *denoising.m* sur Matlab.
On part de l'image originale I qu'on bruite avec une fonction *noise_generator.m* avec une probabilité $p = 0.1$ de changer la couleur d'un pixel. On obtient ainsi l'image bruitée I'. On convertit I' en système d'Ising en remplaçant tous ses pixels de valeur 0 (noirs) par -1 et tous ses pixels de valeur 255 (blancs) par 1 (on fait ça dans une fonction *model_conversion.m*).

Étant donné les nombreuses analogies du problème avec le modèle d'Ising, il semble naturel d'implémenter l'algorithme de Metropolis-Hastings en s'appuyant sur une distribution de proposition similaire à celle de la partie 1, c'est-à-dire telle que :

$$q\left(I'_{11}(t) = i'_{11}(t), .., I'_{MN}(t) = i'_{MN}(t) \mid I'_{11}(t-1) = i'_{11}(t-1), .., I'_{MN}(t-1) = i'_{MN}(t-1)\right)$$
$$= \begin{cases} \frac{1}{M*N} & \text{si } \exists k \in \{11, 12, \ldots, MN\} : i'_k(t) = -i'_k(t-1), i'_l(t) = i'_l(t-1) \forall l \neq k \\ 0 & \text{sinon} \end{cases}$$

qui consiste à générer un nouvel état de l'image à partir du précédent simplement en changeant la couleur d'un pixel tiré au hasard. Dans la première partie, cette densité de proposition avait permis à l'algorithme de Metropolis-Hastings de générer des échantillons d'une réalisation dont la magnétisation moyenne au cours du temps convergeait vers la valeur exacte. Étant donné les similitudes avec ce problème, ce choix de q pourrait permettre à l'algorithme de converger vers l'image I recherchée.

L'implémentation de l'algorithme semble ensuite assez aisée. Dans une boucle :

1. On sélectionne aléatoirement un pixel $I'_{ij}(t)$ selon une densité $q = \frac{1}{MxN}$ grâce à *randsample*. On propose un nouvel état de l'image en $t+1$ où le pixel $I'_{ij}(t+1)$ aurait une couleur opposée à celle qu'il a en t.

2. On calcule la probabilité d'acceptation α du nouvel état.

3. On change la couleur du pixel avec une probabilité α : on tire un nombre aléatoire u entre 0 et 1 avec la fonction *rand* :

$$\begin{array}{ll} \text{Si } u < \alpha, & I'_{ij}(t+1) = -I'_{ij}(t). \\ \text{Sinon,} & I'_{ij}(t+1) = I'_{ij}(t) \end{array}$$

On acceptera un nouvel état $I'(t)$ de l'image bruitée si ce nouvel état nous permet de se rapprocher de l'image originale I.

La probabilité α d'acceptation d'un nouvel état $I'(t)$ de l'image s'écrit donc :

$$\alpha = \min\left\{1, \frac{P(I'(t)|I')}{P(I'(t-1)|I')}\right\}.$$

Puis, en utilisant la formule de Bayes, on obtient :

$$\alpha = \min\left\{1, \frac{P(I'|I'(t))}{P(I'|I'(t-1))}\frac{P(I'(t))}{P(I'(t-1))}\right\}.$$

où la fonction min est utilisée pour être sûr que la valeur d'α ne dépasse jamais 1.

Comme le bruit est appliqué indépendamment à chaque pixel, on peut écrire

$$P(I'|I) = p^a * (1-p)^b = 0.1^a * 0.9^b$$

où a est le nombre de pixels ayant changé de couleur et b est le nombre de pixels étant restés les mêmes. De $t-1$ à t, on propose de changer un pixel. Donc, α peut s'écrire

$$\alpha = \min\left\{1, \frac{0.1^{a+1} * 0.9^{b-1}}{0.1^a * 0.9^b} \frac{P(I'(t))}{P(I'(t-1))}\right\} = \min\left\{1, \frac{0.1}{0.9} \frac{P(I'(t))}{P(I'(t-1))}\right\}$$

Similairement au point **1.3.2** de la Partie 1, on peut écrire

$$\frac{P(I'(t))}{P(I'(t-1))} = \frac{\frac{1}{Z}\exp\left(-\beta E\left(I'(t)\right)\right)}{\frac{1}{Z}\exp\left(-\beta E\left(I'(t-1)\right)\right)} = \exp\left(-\beta \Delta E\right))$$

où

$$\Delta E = E\left(I'(t)\right) - E\left(I'(t-1)\right)$$

$$\Leftrightarrow \Delta E = -J * \sum_{k=1}^{N-1}\sum_{l=1}^{N-1} I'_{kl}(t) * I'_{voisins\,de\,kl}(t) - (-J * \sum_{k=1}^{N-1}\sum_{l=1}^{N-1} I'_{kl}(t-1) * I'_{voisins\,de\,kl}(t-1))$$

Si on propose de changer le pixel $I'_{ij}(t)$ tel que $I'_{ij}(t) = -I'_{ij}(t-1)$ et que le reste de l'image ne change pas, on peut écrire

$$\Delta E = -J * I'_{ij}(t) * \sum I'_{voisins\,de\,ij}(t) - (-J * I'_{ij}(t-1) * \sum I'_{voisins\,de\,ij}(t-1))$$

$$\Leftrightarrow \Delta E = J * I'_{ij}(t-1) * \sum I'_{voisins\,de\,ij}(t-1) + J * I'_{ij}(t-1) * \sum I'_{voisins\,de\,ij}(t-1)$$

$$\Leftrightarrow \Delta E = 2J * I'_{ij}(t-1) * \sum I'_{voisins\,de\,ij}(t-1)$$

La probabilité α d'acceptation d'un nouvel état de l'image s'écrit donc

$$\alpha = \min\left\{1, \frac{0.1}{0.9}\exp(-2\beta J * I_{ij}(t-1) * \sum I_{voisins\,de\,ij}(t-1))\right\}$$

On va considérer comme dans la première partie une constante de couplage $J = 1$. Pour calculer la somme des valeurs des pixels voisins, on utilise les fonctions que nous avons implémentées *for_neighbors.m* ou *eight_neighbors.m*. Le choix de l'utilisation de l'une ou l'autre de ces deux fonctions est expliqué dans les paragraphes qui suivent.

En guise d'exemple, considérons dans une image bruitée un pixel -1 (noir) entouré de 4 pixels +1 (blanc) au temps t, tel que représenté sur la Figure suivante :

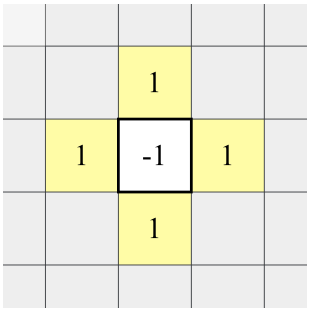


FIGURE 11 – Exemple 1

La probabilité α de changer ce pixel vaudra

$$\alpha = \min\left\{1, \frac{0.1}{0.9}\exp(-2\beta * (-1) * (1+1+1+1)\right\} = \min\left\{1, \frac{0.1}{0.9}\exp(8\beta)\right\}$$

Si on choisit $\beta = 0.2$ comme dans la Partie 1, on aurait une probabilité $\alpha = 55\%$. Si $\beta = 0.4$, le pixel serait changé avec certitude car $\alpha = 1$ (si le pixel sélectionné était également noir, la probabilité de le changer serait de $\alpha = 0.45\%$). Il semble qu'un grand β permettrait de passer d'une image bruitée à une image plus plausible où un pixel d'une certaine couleur serait entouré de pixels de la même couleur.

2.4 Application de l'algorithme pour une image de taille 250x220

On va d'abord tester notre implémentation de l'algorithme de Metropolis-Hastings sur une version réduite de notre image à 250 x 220, qu'on peut voir ci-dessous, et observer le résultat pour $\beta = 0.2$. On va effectuer $3*10^6$ itérations pour tenter de la débruiter.

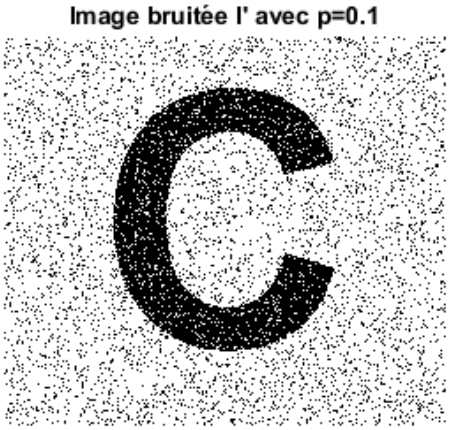


FIGURE 12 – Image bruitée de taille **250x220**

Au lieu de simplement évaluer la qualité de l'image débruitée par notre implémentation à l'oeil nu, on va quantifier la qualité du débruitage grâce à des outils que l'*Image Processing Toolbox* de Matlab met à notre disposition :

- la fonction de corrélation *corr*2 qui retourne le coefficient de corrélation 2D entre les deux matrices I et I'
- la fonction *ssim* qui renvoie le SSIM (*Structural Similarity Index*) :

$$\mathrm{SSIM}(I', I) = \frac{(2\mu_{I'}\mu_I + C_1)(2\sigma_{I'I} + C_2)}{(\mu_{I'}^2 + \mu_I^2 + C_1)(\sigma_{I'}^2 + \sigma_I^2 + C_2)}$$

où μ_I et $\mu_{I'}$ sont les moyennes respectives des deux images, σ_I et $\sigma_{I'}$ sont leurs variances, $\sigma_{II'}$ est leur covariance et C_1 et C_2 sont deux constantes. Cet index permet de mesurer la similitude entre l'image I' et l'image de référence $I/$

On va alors tracer l'évolution de ces deux grandeurs lors du débruitage. Par ailleurs, en testant une première implémentation, on a remarqué que calculer la corrélation et le SSIM à chacune des $3 * 10^6$ itérations augmente considérablement le temps de calcul. On va donc uniquement les calculer à chaque 100 000ème itération, ce qui reste suffisant pour permettre d'observer leur évolution. On obtient les résultats suivants :

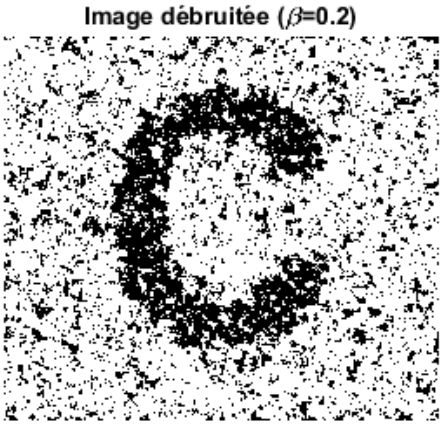


FIGURE 13

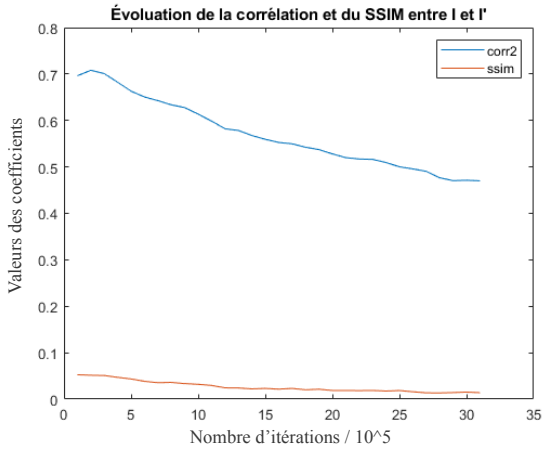
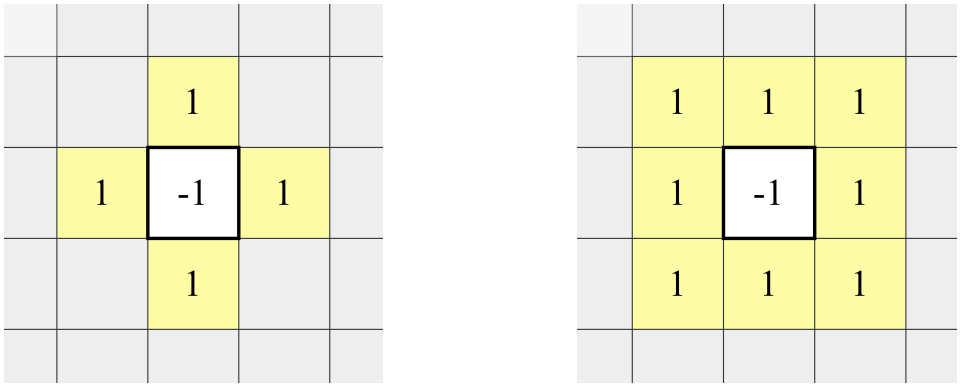


FIGURE 14

Donc, avec $\beta = 0.2$, on obtient l'image de la *Figure 13* qui n'est pas débruitée et qui est encore plus bruitée qu'avant ! Cette observation est confirmée par le fait que, sur la *Figure 14*, le coefficient de corrélation et le SSIM diminuent tous deux au cours des $3*10^6$ itérations. Cela veut dire qu'on a accepté des changements de pixels qui ont bruité l'image encore plus. Même après 3 000 000 d'itérations, aucune convergence de la corrélation et du SSIM avec l'image originale n'est observée ! On a fait le contraire de ce qu'on désirait !

L'hypothèse qu'un pixel d'une couleur est en général entouré de pixels de la même couleur n'en est pas pour autant fausse. Cependant, on a considéré dans notre implémentation qu'un pixel avait au plus 4 voisins. Alors, comme analysé dans l'exemple précédent, un pixel -1 sélectionné aléatoirement qui est entouré de 4 pixels +1 n'a que une probabilité $\alpha = \frac{0.1}{0.9} \exp(-2 * (-0.2) * (-1) * (1 + 1 + 1 + 1)) = 55\%$ de changer.

Pour augmenter cette probabilité, on a 2 choix : soit augmenter le paramètre β soit considérer qu'un pixel possède plus de 4 voisins. On va donc tester ces deux solutions. Commençons par la deuxième, qui semble prometteuse car, lorsqu'on considère qu'un pixel possède 4 voisins et qu'un de ces pixels voisins est lui-même bruité, α descend à 24.72%. Si on examine plus de voisins, on a une meilleure idée de la couleur entourant le pixel sélectionné et on a ainsi moins de chance de tirer des conclusions hâtives.



(a) 4 voisins considérés (b) 8 voisins considérés

FIGURE 15 – On passe de la conception (a) à la conception (b)

On relance alors le même algorithme pour l'image 250 x 220 et avec $\beta = 0.2$ sauf qu'à chaque fois qu'on propose de changer un pixel dans l'image, on considère ses 8 voisins dans le calcul de α : celui juste en bas, en haut, à gauche, à droite, en haut à gauche, en bas à gauche, en haut à droite, en bas à droite. Un pixel situé sur les bords de l'image aura 5 voisins sauf s'il est dans le coin de l'image auquel cas il en aura 4. On détermine les voisins d'un pixel grâce à la fonction $eight\_neighbors.m$ (au lieu de la fonction $four\_neighbors.m$). On obtient alors les résultats présentés à la page suivante.

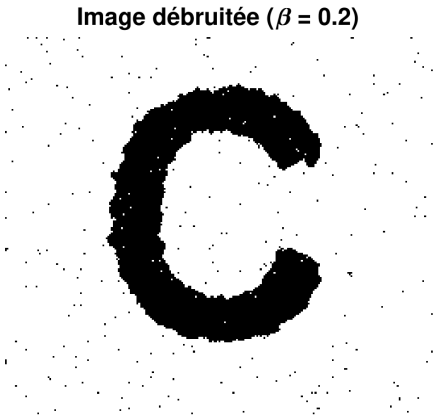


FIGURE 16

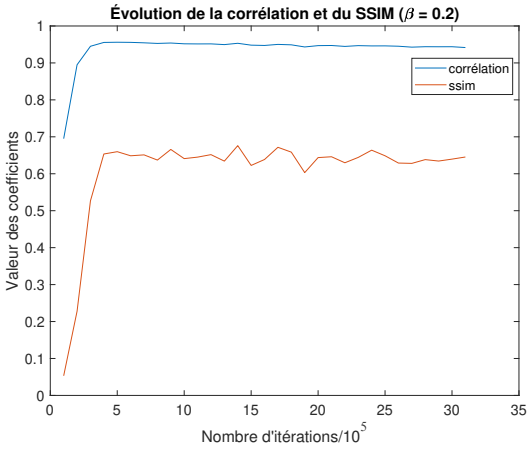


FIGURE 17

On observe sur la *Figure 16* que l'image obtenue se rapproche plus de l'image originale. De plus, on voit sur la *Figure 17* que vers 500 000 itérations, le coefficient de corrélation et le SSIM commencent à converger et atteignent respectivement les valeurs 0.94164 et 0.64536 à la dernière itération. Même si la corrélation est élevée, le SSIM reste assez faible.

Pour tenter d'obtenir d'encore meilleurs résultats, on va utiliser la seconde solution d'amélioration énoncée plus haut, c'est-à-dire qu'on va observer les résultats pour des valeurs de β plus élevées en l'augmentant par pas de 0.2 jusqu'à $\beta = 1$.

On obtient alors les résultats repris dans le tableau suivant. On peut voir, pour les valeurs croissantes de β, les valeurs vers lesquelles le SSIM et le coéfficient de corrélation convergent :

β	0.2	0.4	0.6	0.8	1
coeff corré.	0.94164	0.96682	0.97247	0.97338	0.97365
SSIM	0.64536	0.92027	0.93639	0.93619	0.94007

TABLE 1 – Valeurs des coefficients en fonction de β en t = 3.10^6 pour l'image **250 x 220**

On remarque que le coefficient de corrélation et le SSIM augmentent de $\beta = 0.2$ à $\beta = 0.8$ mais au delà de 0.8, ils ne changent presque plus. Ci-dessous, on peut voir les images obtenues avec $\beta = 0.4$ et avec $\beta = 1$, ainsi que l'évolution des coefficients pour ces différents cas *(On ne présente pas les résultats pour $\beta = 0.6$ et $\beta = 0.8$ pour ne pas alourdir le rapport, étant donné qu'ils représentent simplement des résultats intermédiaires entre ceux présentés ci-dessous)* :

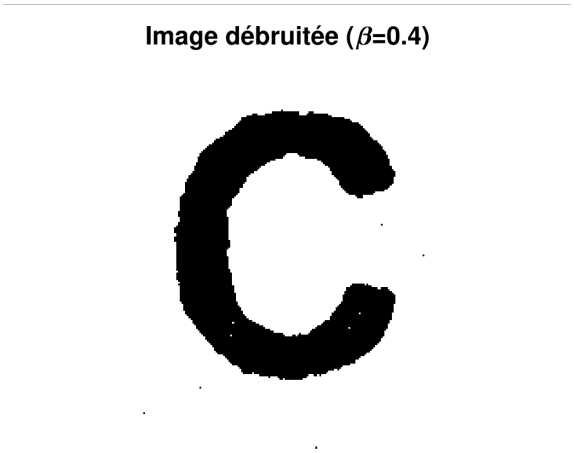


FIGURE 18

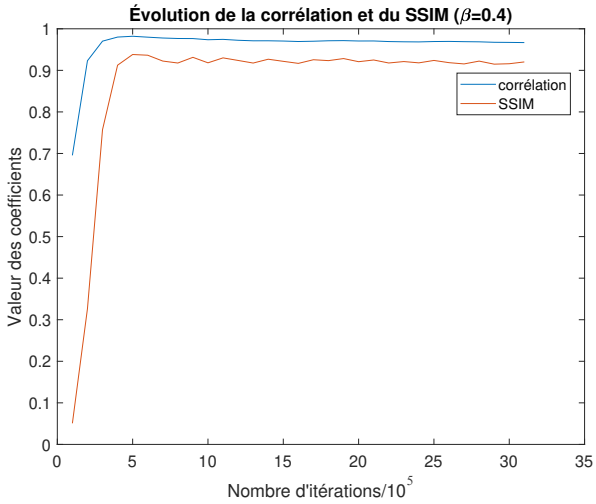


FIGURE 19

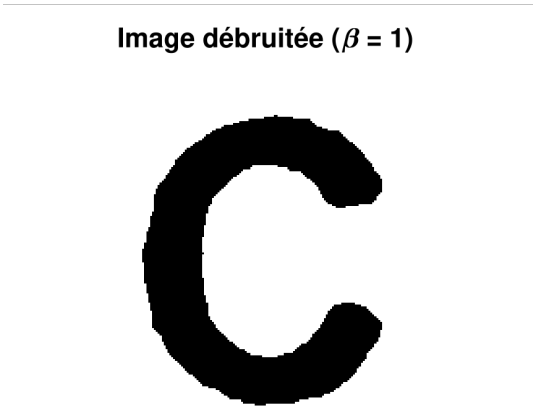


FIGURE 20

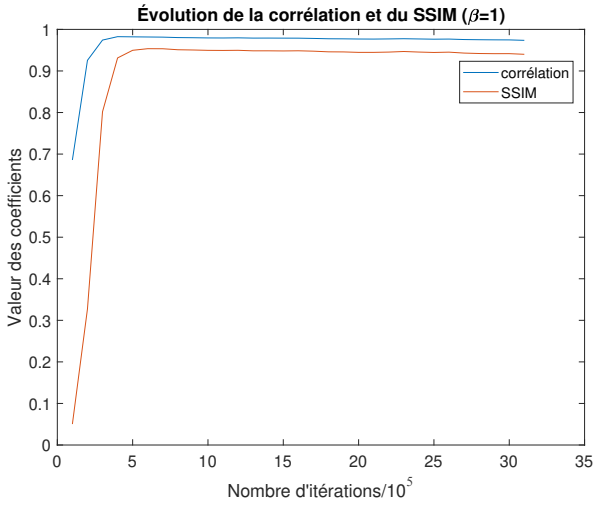


FIGURE 21

On constate bien que l'image débruitée se rapproche de plus en plus de l'image d'origine. On remarque aussi que, peu importe la valeur de β, les valeurs des coefficients commencent à converger à partir de 500 000 itérations, le paramètre β n'a pas d'influence sur le temps de convergence. Cependant, les valeurs vers lesquelles ils convergent sont d'autant plus proche de 1 que β augmente jusqu'à 0.8 (et plus beaucoup au-delà de cette valeur). En relançant l'algorithme pour des valeurs de β bien plus grandes, on voit en effet que la corrélation et le SSIM changent très peu et commencent aussi à converger vers 500 000 itérations :

β	10	100	1000	10000	100000
coeff corré.	0.97324	0.97415	0.97624	0.9776	0.97403
SSIM	0.93878	0.94163	0.94427	0.94713	0.94154

TABLE 2 – Valeurs des coefficients en t = 3.10^6 pour des grands β

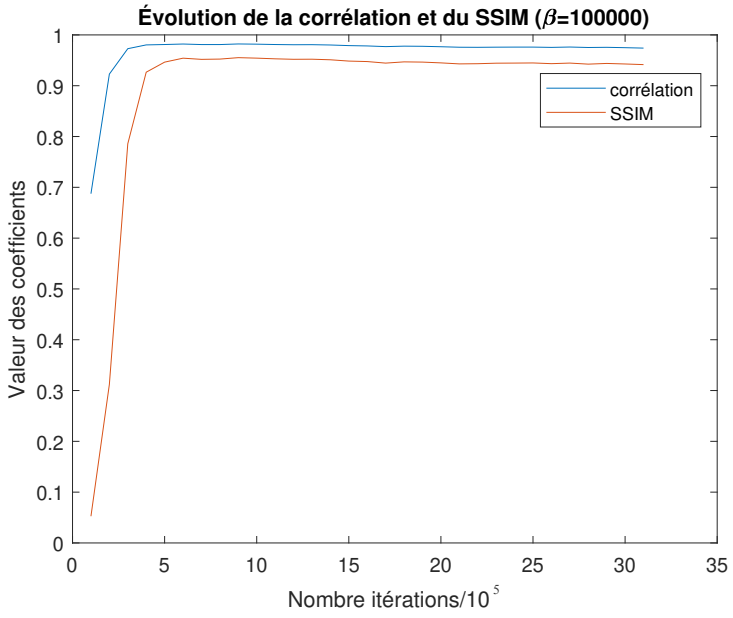


FIGURE 22

2.5 Application de l'algorithme pour l'image originale de taille 620x545

Dans cette partie, on va finalement utiliser notre implémentation de l'algorithme de Metropolis-Hastings sur la version originale de l'image, de dimension 620 x 545 et ce, toujours pour des valeurs de β variant par pas de 0.2 entre 0.2 et 1. Pour s'assurer la convergence du coefficient de corrélation et du SSIM, on effectue ici $4*10^6$ itérations. En outre, on considère 8 voisins pour chaque pixels (exceptés ceux sur les bords de l'image), et non 4 car on a constaté précédemment que cette considération menait à de meilleurs résultats.

On constate que, similairement au cas de l'image réduite, le débruitage est de meilleure qualité lorsque β augmente jusque 0.8. Au-delà de cette valeur, les valeurs du coefficient de corrélation et du SSIM n'augmentent plus. On a reporté dans le tableau suivant les valeurs de ces coefficients en fonction de β :

β	0.2	0.4	0.6	0.8	1
coeff corré.	0.9695	0.9918	0.993	0.9934	0.9934
SSIM	0.6696	0.9694	0.982	0.9828	0.9828

TABLE 3 – Valeurs des coefficients en fonction de β en t = 4.10^6 pour l'image **620 x 545**

Comme dans la cas de l'image réduite, on aperçoit clairement leur augmentation, et on comprend que l'image devient de mieux en mieux débruitée. Sur les 2 pages qui suivent, les résultats de l'image débruitée pour $\beta = 0.2$ et pour $\beta = 1$ sont affichés, ainsi que l'évolution des coefficient au cours des itérations. *Les résultats pour les valeurs intermédiaires de β ne sont pas affichés, car on comprend facilement qu'ils représentent des images "intermédiaires", de mieux en mieux débruitées.*

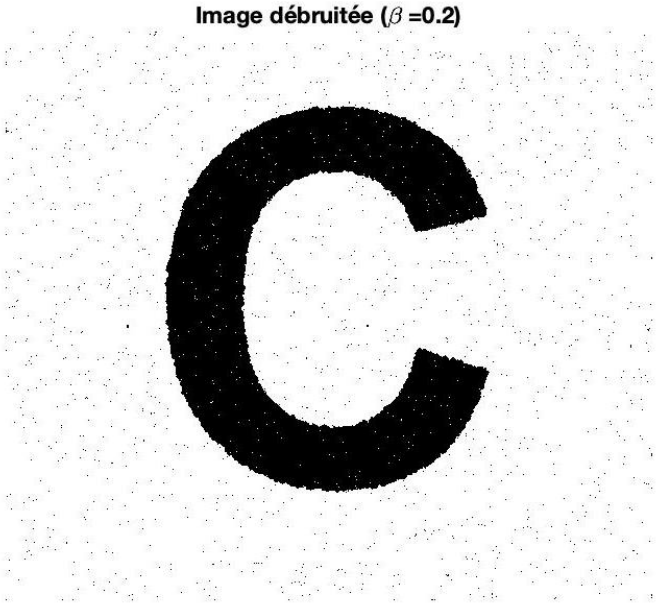


FIGURE 23 – Image débruitée

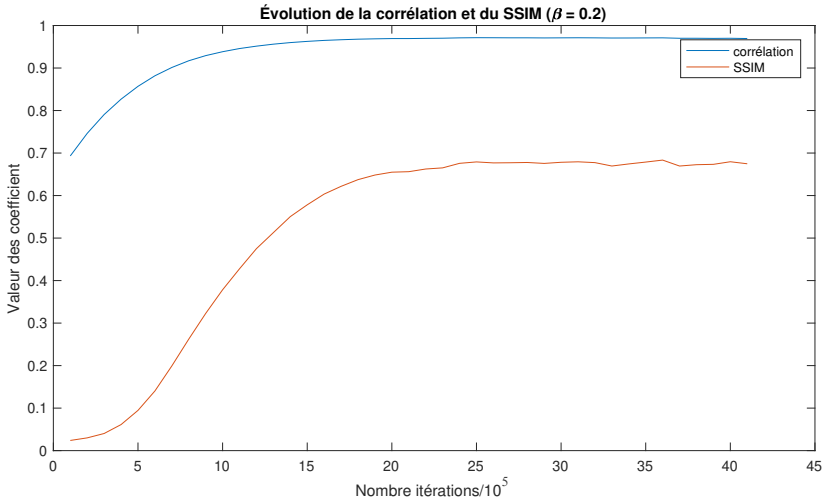


FIGURE 24 – Évolution de la valeur des coefficients pour l'image 620 x 545 avec $\beta = 0.2$

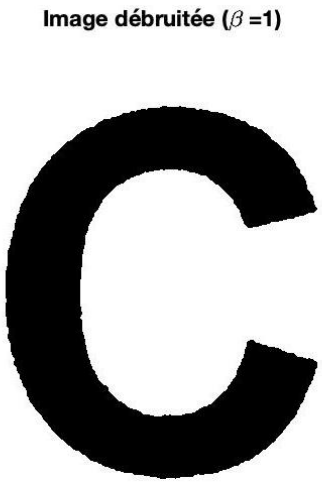


FIGURE 25 – Image débruitée

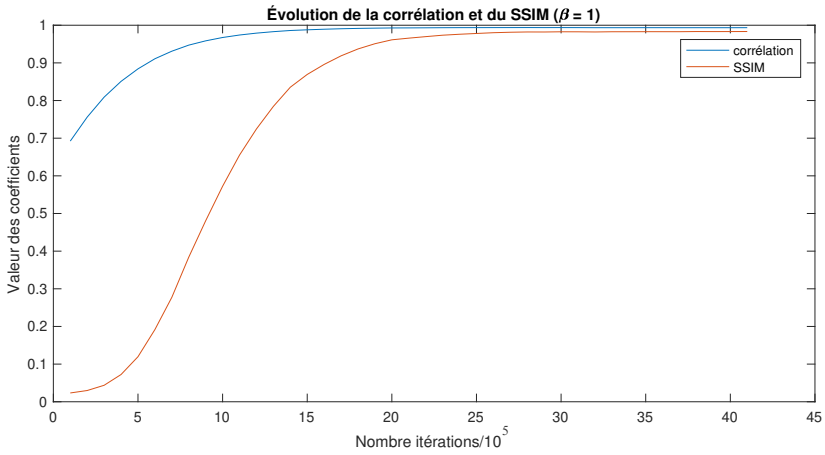


FIGURE 26 – Évolution de la valeur des coefficients pour l'image 620 x 545 avec $\beta = 1$

On observe tout de même sur les *Figures 24* et *26* que le coefficient de corrélation et le SSIM convergent bien plus tard que pour l'image réduite, ce qui est logique car il y a plus de pixels. Ils convergent vers $25 * 10^5$ itérations.

Les bords de la lettre semblent cependant plus correctement débruités que pour l'image réduite. On explique cela sur base du postulat qu'on a émis pour mettre en place l'algorithme qui était qu'une majorité de pixels voisins sont identiques dans l'image standard. Ceci est vrai partout sauf pour les pixels qui se trouvent sur les bords de la lettre, ce seront donc ceux pour lesquels il y aura plus d'ambiguïté lors du débruitage. Le fait d'avoir une image avec plus de pixels réduit la proportion de pixels qui posent problème (sur un bord), c'est pourquoi les valeur de la corrélation et du SSIM sont plus élevées. Les bords paraissent alors plus corrects car le plus grand nombre de pixels permet une meilleur atténuation des pixels mal débruités.

2.6 Conclusion

Que ce soit pour la version réduite de l'image ou pour sa version originale, on n'a pas pu complètement débruiter l'image. La meilleure reconstruction obtenue était celle pour $\beta = 1$ pour la grande image où on a eu un coefficient de corrélation de 0.9934 et un SSIM de 0.9828 . Le bruit restant se situe sur les contours de la lettre C.

Cela a du sens car, en choisissant une distribution de probabilité $P(I)$ proche de celle du modèle d'Ising, on postule que l'image standard est généralement telle qu'une majorité de pixels voisins sont identiques. Or, ce n'est pas le cas pour les pixel noirs en bordure de la lettre qui sont avoisinés par des pixels blancs du fond blanc.

Il existe de célèbres filtres, comme le filtre médian par exemple, qui reconstruisent bien les contours dans une image. Le filtre médian est un filtre qui remplace chaque pixel aberrant par la médiane de ses voisins [4]. Après avoir bruité l'image originale avec une probabilité $p = 0.1$ de changement de pixels , on applique un filtre médian (*Figure 27*).

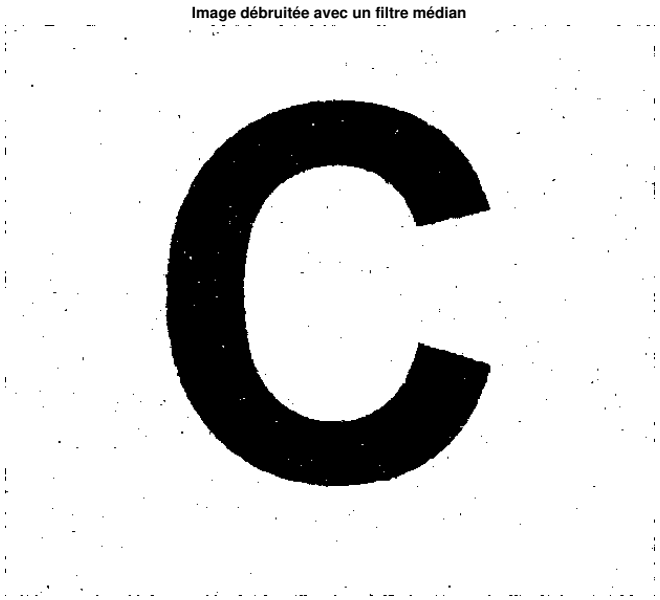


FIGURE 27

On observe que sur le résultat du filtrage médian, bien que les contours sont pratiquement comme ils l'étaient avant le bruitage, des pixels autre part que sur la lettre centrale sont noirs ! On n'avait pas ce problème après avoir débruité l'image avec l'algorithme de Metropolis-Hastings. De plus, le SSIM par rapport à l'image originale est de 0.9348 alors qu'il est de 0.9828 pour un $\beta = 1$ avec l'algorithme de Metropolis-Hastings.

Cela montre que chaque méthode de filtrage d'un bruit possède ses faiblesses. Dans certains cas (cas où on a une forme simple dans une image binaire), l'algorithme de Metropolis-Hastings peut s'avérer plus efficace que des filtres déjà existants pour le débruitage.

4. https ://fr.wikipedia.org/wiki/Filtre$_m$%$C3$%$A9dian$

Le choix de la distribution proche d'Ising et le choix qu'on a fait pour la distribution de proposition q a permis à l'algorithme de converger vers une solution approchée optimale, approche plus efficace que de parcourir les $2^{545x620}$ solutions possibles.

Pour finir, nous avons décidé de tester notre implémentation sur une autre grande image de taille **1200x1560**. Cette fois, l'image est bruitée avec une probabilité $p = 0.3$ de changement de pixels. On peut voir ci-dessous à gauche, l'image originale et à droite, l'image bruitée :

FIGURE 28 – Image originale I
source : https ://2ch.hk/b/arch/2019-11-26/res/208248221.html

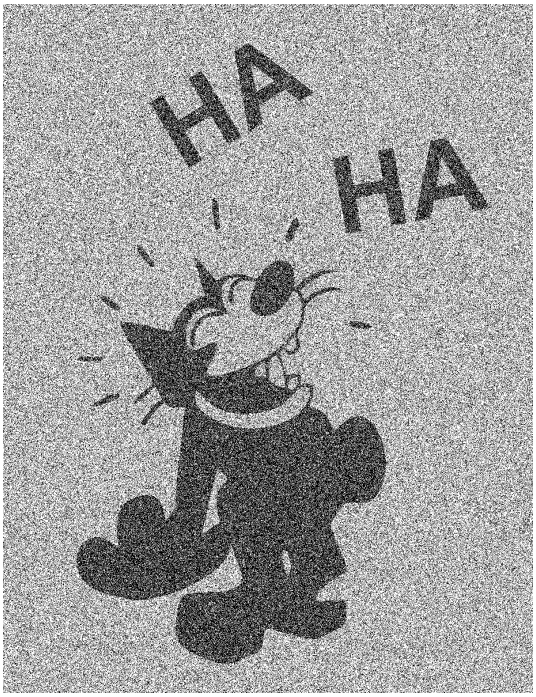


FIGURE 29 – Image bruitée I' avec p=0.3

Après débruitage de cette image binaire selon notre implémentation de l'algorithme de Metropolis-Hastings, on obtient les évolutions du coefficient de corrélation et du SSIM ci-dessous, ainsi que l'image à la page suivante :

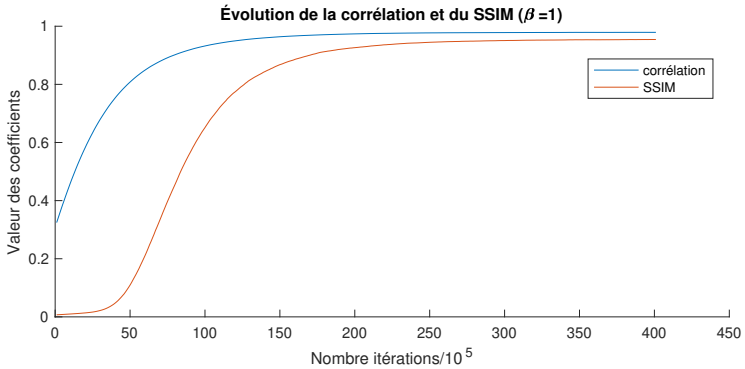


FIGURE 30 – Évolution de la valeur des coefficients pour l'image 1200x1560 avec $\beta = 1$

FIGURE 31 – Image débruitée

On aperçoit bien que l'image débruitée se rapproche beaucoup de l'image originale, malgré les pixels sur les contours des formes qui restent ceux pour lesquels le débruitage selon l'algorithme de Metropolis-Hastings est le plus difficile, pour les raisons explicitées plus haut. Les valeurs du coefficient de corrélation et du SSIM valent respectivement 0.979 et 0.9543, ce qui indique bien que une grande partie du bruit a été corrigé et qu'on retrouve ainsi une image similaire à plus de 95% de l'image originale.