Table des matières

Table des figures

Liste des algorithmes

Nomenclature

Symboles	Définitions
F	Distribution cible de densité f
G	Distribution instrumentale de densité g
M	Moment d'une distribution
i, j	Indices généraux
n, m	Indices pour les éléments des vecteurs et des matrices
t	Indice des itérations d'un algorithme
T	Nombre total d'itérations
X	Variable aléatoire
x	Réalisation de la variable aléatoire X
χ	Domaine de la variable aléatoire X
Y	Vecteur des observations
$\{x_i\} = \{x_1, ..., x_n\}$	Série de réalisations d'une variable aléatoire
$U_{[0,1]}$	Distribution uniforme défini sur l'intervalle unité
u	Variable aléatoire distribuée selon $U_{[0,1]}$
c	Constante
Θ	Espace des états d'une chaîne de Markov
$\theta \in \Theta$	État d'une chaîne de Markov
$K(\theta, \theta')$	Noyau d'une chaîne de Markov
w_i	Pondération du $i^{\text{ème}}$ élément dune série
h	Fonction de mise à jour
H	Coupleur
MCMC	Abréviation méthodes de Monte-Carlos par chaînes de Markovs
CFTP	Abréviation du couplage par le passé
SMC	Abréviation méthodes de Monte-Carlos séquencielles
BIC	Abréviation du critère d'information bayésien

Chapitre 1

Introduction

Stanislaw Ulam venait de perdre un bout de cerveau lorsqu'il eut l'idée des méthodes Monte-Carlo. Alité depuis plusieurs jours, se remettant d'une encéphalite qu'il avait contractée au début de l'année 1946, il se divertissait en jouant au solitaire lorsqu'il se demanda quelle était la probabilité qu'il complète les séries selon les paquets de cartes à jouer qu'il avait devant lui. Après avoir tenté, sans succès, de répondre à cette question avec les méthodes combinatoires usuelles, il tenta de trouver une solution originale à ce problème qui semblait pourtant bien simple.

Mathématicien d'origine polonaise, Ulam avait émigré aux États-Unis pour terminer ses études universitaires à Harvard, peu de temps avant la Deuxième Guerre mondiale. En plus de lui sauver la vie, cette migration lui permit de rencontrer et créer des liens d'amitié avec John Von Newmann avant que celui-ci devienne une figure de proue de l'effort de guerre de la communauté scientique américaine. Lorsque ce dernier devient conseiller scientifique pour la U.S. navy, il invita Ulam à se joindre à l'équipe du projet Manhattan regroupé au laboratoire national de Los Alamos. Là-bas, il contribua à la résolution de problèmes de dynamique des fluides liés à l'implosion de l'uranium des premières bombes atomiques, mais surtout, il fût témoin de la construction d'ENIAC, le premier ordinateur américain.

Ulam remarqua qu'il était possible d'approximer la probabilité de réussite d'une main en observant plusieurs joueurs de solitaires jouer une partie débutant avec cette main. puis en calculant la proportion de joueurs réussissant à terminer leur partie. De plus, il savait que la qualité de cette approximation était proportionnelle à la quantité de joueurs observés et qu'ultimement, avec une infinité de joueurs. l'approximation serait égale à la vraie probabilité de réussite. Or cette méthodologie est inutilisable. Pour s'en convaincre, il suffit de considérer la quantité de ressources nécessaire à l'organisation d'une petite expérience d'une centaine de joueurs, comme la quantité de personnel à employer à trouver les participants. louer les locaux, acheter les cartes, placer des tables, etc. Ulam le savait. Ayant été impressionné par la puissance de calcul du premier ordinateur, Ulam naturellement considéra son utilisation pour résoudre ce problème probabiliste qu'il s'était posé en jouant aux cartes.

L'idée d'utiliser l'échantillonnage statistique pour résoudre des problèmes quantitatifs fût utilisée bien avant Ulam. Par exemple, en 1733, Laplace proposa une expérience inusitée, appelée l'aiguille de Buffon, permettant d'estimer la valeur de π. Sa méthodologie consistait à laisser tomber, une à une, un certain nombre n d'aiguilles de longueurs l entre deux lignes parallèles, séparées par une distance $d > l$, puis à compter le nombre m d'aiguilles ayant traversées une des lignes, dans le but d'estimer la probabilité d'un tel évènement selon l'expression $p = \frac{m}{n}$. Georges Louis Leclerc dit comte de Buffon, peu de temps auparavant, avait utilisé le calcul différentiel pour déterminer que cette probabilité était égale à $\frac{2l}{d\pi}$. En conséquence, l'expérience de l'aiguille de Buffon permet d'estimer la valeur de π, en utilisant l'approximation $\pi = \frac{2ln}{dm}$. De même, en 1908, William Sealy Gosset, surtout connu sous son pseudonyme de Student, fit un sondage parmi 3000 prisonniers pour obtenir deux bases de données ; l'une enregistrant leurs tailles, l'autre la longueur de leur

majeur. Puisque ces deux données sont proportionnelles au gabarit du prisonnier, Gosset put les utiliser pour simuler deux lois normales corrélées. Ces deux exemples ont en commun d'utiliser des phénomènes physiques pour générer des échantillons de nombres aléatoires et d'utiliser des méthodologies difficilement transférables à d'autres situations que celle pour laquelle elles ont été pensées.

Après sa convalescence, Stanislaw Ulam travaillait, entre autres, à systématiser et à prouver l'efficacité de sa méthode d'échantillonnage lorsqu'il eut l'occasion de la tester. En 1947, les scientifiques de Los Alamos commençaient à se pencher sur le concept de la bombe à neutron. John Von Newmann qui était toujours directeur scientifique du laboratoire à cette époque et était au courant des travaux de Ulam, lui demanda de venir utiliser sa méthode pour résoudre des équations de diffusions des neutrons dans des matériaux en fission. Ulam accepta et c'est grâce au travail collectif de ces mathématiciens de Los Alamos que cette méthode d'utilisation de l'échantillonnage pour résoudre des problèmes quantitatifs fût raffinée. D'abord, Von Newmann et Ulam ont défini le cadre théorique rigoureux de cette méthode, facilitant ainsi son utilisation avec la plupart des problèmes à résoudre. Ensuite, Von Newmann eut l'idée d'utiliser des algorithmes pour générer les nombres aléatoires nécessaires à ces méthodes. Ces algorithmes, nommés générateurs de nombres pseudoaléatoires, ont l'avantage d'être plus rapides et moins coûteux que l'utilisation de listes de nombres aléatoires générés par des phénomènes physiques. Finalement, pour répondre aux différents problèmes théoriques liés à la bombe à neutron, d'autres mathématiciens comme Nicholas Metropolis et Stanley Phillips Frankel, ont développé des versions particulières des méthodes Monte-Carlo démontrant ainsi la versatilité de cette nouvelle approche à la résolution de problème.

Il n'existe pas de définition formelle des méthodes de Monte-Carlo. D'une référence à l'autre, ce terme est employé autant pour désigner les méthodes utilisant la simulation de phénomènes aléatoires pour résoudre des problèmes mathématiques, comme des problèmes d'optimisations ou d'intégrations, que les méthodes utilisant l'échantillonnage répété de variables aléatoires pour déterminer le comportement d'une distribution ou bien des techniques de simulations. Dans ce document, une distinction est faite entre les techniques de simulation permettant de générer un échantillon de variables aléatoires distribuées selon une densité donnée et les méthodes de Monte-Carlo qui utilisent ces échantillons. De plus, l'étude des méthodes Monte-Carlo est faite sous l'angle de l'utilisation de ces méthodes pour estimer des moments d'une distribution donnée. Ce choix se justifie de par le fait que la méthodologie utilisée pour résoudre ce type de problèmes peut-être utiliser pour résoudre plusieurs autres types de problématiques, comme l'estimation d'intégrale ou l'estimation de statistique de test par exemple et qu'elle peut facilement être modifiée pour résoudre des problèmes d'autre nature, comme des problèmes d'optimisations ou de prise de décisions.

Chaque méthode présentée dans ce document utilise une technique de simulation particulière pour créer un échantillon, puis estime le moment à l'aide d'une méthode empirique. Donc, chacun des chapitres constituant ce document présente des techniques de simulation spécifique au contexte à étudier et discute des performances des méthodes de Monte-Carlo qui leur sont associées. En particulier,

- Le chapitre 2 présente les principes à la base des méthodes de Monte-Carlo, ainsi que les méthodes dites classiques. Les méthodes présentées dans ce chapitre sont des extensions naturelles de méthodes de simulations simples à implémenter, mais inutilisables lorsque la

distribution dont nous voulons déterminer la valeur des moments est multivariée.

- Le chapitre 3 introduit une classe de méthodes de Monte-Carlo utilisant les chaînes de Markov pour contourner les problèmes inhérents aux algorithmes présentés précédemment. Les méthodes de Monte-Carlo par chaînes de Markov, notée MCMC selon l'acronyme anglais, consiste à utiliser une chaîne de Markov ayant pour distribution stationnaire la fonction à simuler pour générer les points nécessaires à l'approximation.

- Le chapitre 4 discute des différents aspects de l'utilisation des méthodes de Monte-Carlo par chaînes de Markov. La première partie de ce chapitre énumère, via des exemples concrets, des différentes variantes de l'implémentation de ces méthodes et comment celles-ci influencent la vitesse d'exécution de l'algorithme et la qualité des estimations. La deuxième partie énumère les différents tests pouvant être utilisés pour évaluer la qualité de la simulation.

- Le chapitre 5 est une introduction aux méthodes de Monte-Carlo séquentielles, méthodes conçues pour modéliser des situations où le processus à simuler évolue en temps réel. Ce chapitre s'ouvre sur une présentation des hypothèses nécessaires à l'utilisation des méthodes de Monte-Carlo en contexte séquentiel. Ensuite, les filtres de Kalman sont introduits comme des méthodes permettant d'obtenir des estimations optimales dans un tel contexte et le chapitre se termine par l'étude des méthodes de Monte-Carlo séquentielles les plus connues.

Chapitre 2

Méthodes Monte-Carlo

Ce chapitre se veut une introduction aux méthodes de Monte-Carlo simples. La première section présente la définition des méthodes de Monte-Carlo utilisée tout au long de ce document. Ensuite, les problématiques liées à ces techniques sont présentées à l'aide de l'introduction à la méthode de la transformée inverse. Le restant du chapitre est dédié à l'étude de diverses méthodes de Monte-Carlo simples, soit les méthodes d'acceptation-rejet, l'échantillonnage adaptatif, l'échantillonnage préférentiel et la méthode de ré-échantillonnage.

2.1 Définition générale

Dans ce document, les méthodes de Monte-Carlo sont définies comme étant des techniques statistiques ayant pour but d'estimer la valeur des différents moments d'une distribution à l'aide d'un échantillon aléatoires $\{X_{1:n}\} = \{X_1, X_2, \ldots, X_i, \ldots, X_n\}$ et identiquements distribuées selon la fonction de répartition F définie dans le domaine χ, à étudier. En supposant que F possède une densité f, ses moments sont de la forme,

$$M = \int_{\chi} u(x) f(x) dx = \mathbb{E}\left[u(X)\right], \tag{2.1}$$

et si $f(x)$ est une fonction régulière, par la loi forte des grands nombres nous savons que la moyenne empirique converge asymptotiquement vers M,

$$\tilde{M} = \frac{1}{n} \sum_{i=1}^{n} u(X_i) \rightarrow \int_{\chi} u(x) f(x) dx = \mathbb{E}\left[u(X)\right] = M. \tag{2.2}$$

En générant, à l'aide d'une méthode de simulation de nombres pseudo-aléatoires, un échantillon de réalisations de variables aléatoires indépendantes identiquement distribuées selon $f(x)$, de taille n suffisament élevée, il est possible obtenir une estimation de M à l'aide de la moyenne empirique. Cette démarche est partagée par toutes les méthodes de Monte-Carlo étudiées dans ce document. L'expression de la variance de l'estimation étant dépendante de la méthode de simulation utilisée pour générer l'échantillon, pour obtenir la précision minimum permettant l'inférence, il est nécessaire d'utiliser une méthode adaptée à la situation à étudier. Ce fait justifie la multiplication des méthodes de Monte-Carlo. L'exemple le plus direct de cette méthodologie utilise le théorème de la réciproque pour créer l'échantillon.

2.2 Méthode de simulation par inversion

Soit $F(x)$, une fonction de répartition. Nous définissons l'inverse, $F^{-1}(x)$, de cette fonction par

$$F^{-1}(x) = \inf\{y \text{ telle que } F(y) \geq x\} \tag{2.3}$$

Théorème 1 (Théorème de la réciproque). *Si U est une variable aléatoire distribuée selon la loi uniforme $U_{[0,1]}$, alors la variable aléatoire*

$$\theta = F^{-1}(U)$$

est distribuée selon la loi F.

Ce résultat justifie la technique de simulation nommée méthode de la transformée inverse, dont le pseudo-code est résumé dans l'algorithme 2.2.1.

Algorithme 2.2.1 Simulation par inversion

Entrées: Le nombre d'itération T.
Poser t=1.
pour $t \leq T$ **faire**
Générer un nombre pseudo-aléatoire u distribué selon $U_{[0,1]}$.
Poser $x_t = F^{-1}(u)$.
$t = t + 1$.
fin

Lorsque les variables à simuler sont des variables vectorielles de forme $X = (X_1, ..., X_n)$ où les X_i sont des variables aléatoires univariées, cet algorithme est inutilisable. Dans ce cas, si les composantes de X sont indépendantes, il suffit de simuler chacune des composantes X_i individuellement selon sa loi marginale, puis de former le vecteur X par concatenation de ses composantes. Si les composantes de X sont dépendantes en probabilité, alors il est nécessaire de déterminer la distribution marginale de X_1, noté $F(X_1)$, ainsi que la série d'équations conditionnelles $F(X_2|X_1), F(X_3|X_2, X_1), ..., F(X_n|X_{n-1}, ..., X_1)$ et leurs inverses. À l'aide de ces données il est possible d'utiliser l'algorithme suivant :

Algorithme 2.2.2 Méthode de la transformée inverse multivariée

Entrées: Le nombre d'itération T.
Poser t=1.
pour $t \leq T$ **faire**
Générer un nombre pseudo-aléatoire u_1 distribué selon $U_{[0,1]}$.
Poser $x_1 = F_1^{-1}(u_1)$
Générer un nombre pseudo-aléatoire u_2 distribué selon $U_{[0,1]}$.
Poser $x_2 = F_1^{-1}(u_2|x_1)$
...
Générer un nombre pseudo-aléatoire u_n distribué selon $U_{[0,1]}$.
Poser $x_n = F_1^{-1}(u_n|x_{n-1}, ..., x_1)$.
Poser $X_t = [x_1, ..., x_n]$.
$t = t + 1$.
fin

Puisque le théorème de la transformée inverse permet de générer des échantillons parfaitement distribués selon une loi donnée, chacun des candidats générés par l'algorithme est utilisé pour le calcul de M. Malheureusement, ce rendement optimal est atteint au détriment de la souplesse de cette méthode. En effet, cette méthode suppose que la distribution F est connue et qu'il est facile d'inverser la fonction de répartition ce qui est rarement le cas en pratique.

2.2.1 Précision de l'estimation

Puisque les méthodes de Monte-Carlo utilisent la moyenne empirique pour estimer la valeur du moment M, les résultats classiques liés à cette statistique peuvent être utilisés pour caractériser les résultats obtenus par ces méthodes. En particulier, le théorème central limite nous assure que,

$$\frac{\sqrt{n}}{\sigma}\left[\tilde{M} - M\right] = \frac{\sqrt{n}}{\sigma}\left[\frac{1}{n}\sum_{i=1}^{n} u(X_i)f(X_i) - \mathbb{E}(u(X))\right] \rightarrow N(0,1), \tag{2.4}$$

où σ est l'écart-type de la fonction F. Lorsque la méthode de la transformée inverse est utilisée comme technique de simulation dans une méthode de Monte-Carlo, la variance de l'estimation ainsi obtenue peut être estimée par l'expression,

$$\sigma^2(\tilde{M}) = \frac{1}{n}\sum_{i=1}^{n} u(X_i)^2 - \left(\frac{1}{n}\sum_{i=1}^{n} u(X_i)\right)^2 = \frac{\sigma^2(u(X_i))}{n}. \tag{2.5}$$

Ce résultat permet d'établir un intervalle de confiance pour notre estimation et de déterminer la vitesse de convergence de cet algorithme. En effet, ce résultat montre que la variance de $\sigma^2(\tilde{M})$ diminue proportionnellement à $\frac{\sigma^2}{N}$, donc l'erreur d'estimation est proportionnelle à,

$$\tilde{M} - M \sim \frac{\sigma^2(M)}{\sqrt{n}}. \tag{2.6}$$

En conséquence, les méthodes de Monte-Carlo produisent des approximations des moments à une vitesse indépendante de la dimension de l'objet à estimer $O(\frac{\phi^2(M)}{\sqrt{n}})$. Cette propriété, rends ces méthodes particulièrement intéressantes lorsqu'on les compare aux méthodes analytiques, dont la complexité est directement liée à la dimension de la statistique à estimer.

Puisque l'échantillon obtenu par la méthode de simulation par inversion est parfaitement distribuée selon la fonction $f(x)$, l'utilisation de cette méthode ne fait pas augmenter la valeur de $\phi^2(M)$. Cette propriété n'est généralement pas partagée par les autres méthodes de simulation, ce qui rend la méthode de simulation par inversion la méthode à privilégier. Cependant, cette méthode a deux grandes limitations. D'abord, plusieurs distributions ne possèdent pas de distribution inverse ce qui restreint considérablement sa possible utilisation. Par exemple, la loi normale ne peut pas être étudiée en utilisant directement cette méthode [1]. Ensuite, l'algorithme de simulation par inversion est relativement lent, ce qui le rend pratiquement inutilisable pour étudier des distributions de hautes dimensionnalités. Pour que la méthode de Monte-Carlo soit utilisable dans ces situations, d'autres algorithmes de simulations doivent être utilisés à la place de la méthode de la transformée inverse. La méthode d'acceptation-rejet est un de ces algorithmes.

2.3 Méthode d'acceptation-rejet

Soit la fonction cible $f(x)$, la densité associée à la variable aléatoire dont nous voulons simuler une suite de réalisations. Supposons qu'il existe une fonction arbitraire g de domaine identique à

1. Il est possible de manipuler la méthode de la transformée inverse pour simuler une loi normale centrée réduite. Pour ce faire, il suffit de simuler une variable aléatoire $Y = cos(2\pi u_1)\sqrt{2\log(1/u_2)}$ où $u_1, u_2 \sim U_{[0,1]}$

f, noté χ, et une constante $C \geq 1$ telle que pour tout $x \in \chi$

$$\frac{f(x)}{g(x)} \leq C \quad et \quad g(x) \neq 0. \tag{2.7}$$

L'algorithme d'acceptation-rejet est le suivant :

Algorithme 2.3.1 Algorithme d'acceptation-rejet

Entrées: Le nombre d'itération T.
Poser t=1.
pour $t \leq T$ **faire**
Générer $\tilde{x}_t$ selon $g(x)$.
Générer u selon $U_{[0,1]}$.
si $u \leq \frac{f(\tilde{x}_t)}{Cg(\tilde{x}_t)}$ **alors**
$\theta_t = \tilde{x}_t$.
fin
$t = t + 1$.
fin

L'existance d'une borne supérieure sur $\frac{f(x)}{g(x)}$ nous assure que la loi distribuant les candidats acceptés est bien $f(x)$. En effet, si u est une variable aléatoire distribuée selon une loi uniforme $U_{[0,1]}$, la probabilité qu'un candidat x_i, soumis à l'étape i, soit accepté égale,

$$\begin{aligned}
\mathbb{P}(\hat{x}_i \text{ est accepté}) &= \mathbb{P}\left(U < \frac{f(\hat{x}_i)}{Cg(\hat{x}_i)}\right), \\
&= \mathbb{E}\left[\mathbb{1}_{\{U < \frac{f(\hat{x}_i)}{Cg(\hat{x}_i)}\}}\right], \\
&= \mathbb{E}\left[\mathbb{E}\left[\mathbb{1}_{\left\{U < \frac{f(\hat{x}_i)}{Cg(\hat{x}_i)}\right\}}\right] \mid \hat{x}_i\right], \\
&= \mathbb{E}\left[\frac{f(\hat{x}_i)}{Cg(\hat{x}_i)}\right], \\
&= \int_{-\infty}^{\infty} \frac{f(x)}{Cg(x)} g(x) dx, \\
&= \frac{1}{C}.
\end{aligned} \tag{2.8}$$

Donc, si $\tilde{x_i}$ est accepté.

$$\begin{aligned}
\mathbb{P}(\tilde{x_i} < \theta \mid \tilde{x_i} \text{ est accepté}) &= \frac{\mathbb{P}(\tilde{x_i} < \theta, \tilde{x_i} \text{ est accepté})}{\frac{1}{C}}, \\
&= C\mathbb{P}\left(\tilde{x_i} < \theta, U < \frac{f(\tilde{x_i})}{Cg(\tilde{x_i})}\right), \\
&= C\mathbb{E}\left[\mathbb{1}_{\left\{\tilde{x_i} < x, U < \frac{f(\tilde{x_i})}{Cg(\tilde{x_i})}\right\}}\right], \\
&= C\mathbb{E}\left[\mathbb{E}\left[\mathbb{1}_{\left\{\tilde{x_i} < x, U < \frac{f(\tilde{x_i})}{Cg(\tilde{x_i})}\right\}}\right] \mid \tilde{x_i}\right], \\
&= C\mathbb{E}\left[\mathbb{1}_{\{\tilde{x_i} < x\}} \frac{f(\tilde{x_i})}{Cg(\tilde{x_i})}\right], \\
&= \int_{-\infty}^{x} \frac{f(t)}{g(t)} g(t)dt, \\
&= F(x).
\end{aligned} \tag{2.9}$$

L'identitée 2.8, a pour conséquence de rendre cette méthode inutilisable lorsque le rapport $\frac{f(x)}{g(x)}$ n'est pas borné par C sur tout l'interval χ. Dans cette perspective, les fonctions à queue importantes sont particulièrement difficile à simulées par cette méthode puisqu'elles nécessitent l'utilisation d'une fonction $g(x)$, elle aussi à queue importante, facile à simuler et qui "enveloppe" complètement $f(x)$. Une telle fonction est difficile à construire lorsque la quantité d'information concernant la fonction cible $f(\theta)$ est limitée. En effet, dans ce cas, il est aisé de sous-estimer l'importance de la queue de la fonction et d'utiliser une fonction $g(x)$ ne couvrant pas l'entièreté du domaine de la fonction $f(x)$ à simuler. Les éléments ainsi générés ne serons pas distribuées selon $f(x)$. La fonction $g(x)$ est nommée *fonction instrumentale* et dans le reste du document, nous allons assumer que les fonctions instrumentales et les fonctions cibles sont des densités.

Pour simuler une distribution à l'aide de l'algorithme d'acceptation-rejet, il est seulement nécessaire de connaitre $f(x)$ à une constante près. Cela rend cette technique beaucoup plus versatile que la méthode de simulation par inversion. Cette flexibilité à un prix : comme la probabilité d'acceptation d'un candidat est égal à $\frac{1}{C} < 1$, le nombre moyen de couples (x, u) à générer sera C fois plus élevé que par la méthode de simultion par inversion. Pour que cette méthode soit performante, il faut choisir une fonction instrumentale $g(x)$ facile à simuler et qui soit sensiblement "près" de la fonction à simuler. Cette condition nous assure que la probabilité d'acceptation, égale à $\frac{1}{C}$ soit environ égale à 1.

2.3.0.1 Exemple d'application

La méthode d'acceptation-rejet est ici utilisée pour simuler un mélange de deux lois normales bivariées respectivement de moyenne $\mu_1 = 4$ et $\mu_2 = 8$, d'écart-type $\sigma_1 = 1$ et $\sigma_2 = 2$, avec une pondération $\alpha = 0.4$. Pour ce faire, les candidats générés sont distribués selon une fonction instrumentale sinusoïdale d'équation,

$$g(x) = 0.23 \sin\left(\frac{x + 0.7}{5}\right). \tag{2.10}$$

La figure 2.1 présente les deux fonctions d'intérêts dans cette situation. Comme le montre ce graphique, la fonction $g(x)$ enveloppe bien la distribution cible et que la distance entre ces deux fonctions est pratiquement nulle dans le voisinage de $x = 4$, ce qui nous assure que la borne $M \approx 1$.

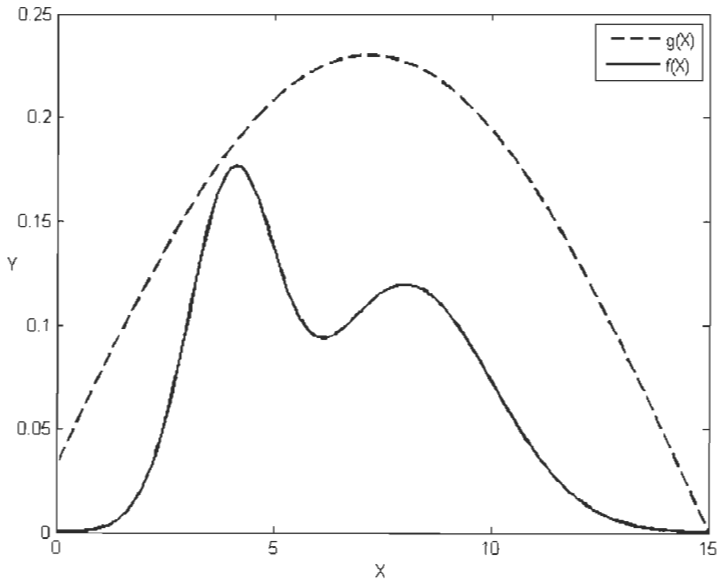
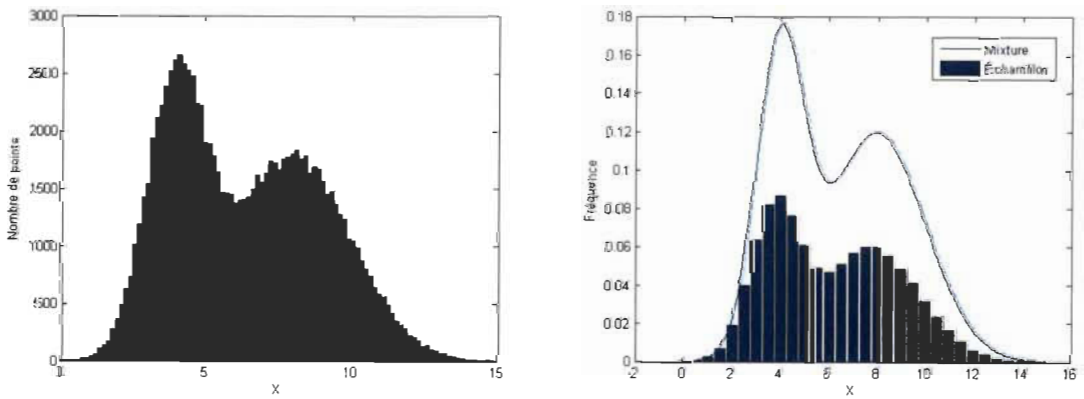


FIGURE 2.1 – Fonctions cible f(x) et instrumentale g(x)

Les figures 2.2a et 2.2b présente un échantillon de 100000 points, ainsi qu'une estimation des fréquences de la distribution. Un examen visuel de ces deux graphiques montre la validité de cette méthode.



(a) Distribution de l'échantillon (b) Estimation des fréquences de la distribution

FIGURE 2.2 – Distribution de l'échantillon généré par la méthode d'acceptation-rejet

2.3.1 Méthode d'acceptation-rejet approchée

Lorsque la fonction $f(x)$ est difficile à évaluer, il est préférable d'utiliser des fonctions intermédiaires plus simples pour ainsi rendre le critère d'acceptation $\frac{f(x)}{g(x)} \leq C$ plus facile à évaluer. Soit $a(x)$ et $b(x)$ de telles fonctions satisfaisant $b(x) \leq f(x) \leq a(x), \quad \forall x \in \chi$. L'utilisation de ces fonctions plus faciles à calculer permet de discriminer rapidement des candidats $\tilde{x}$, sans avoir nécessairement à calculer $f(x)$ et d'ainsi améliorer la performance de l'algorithme. Après avoir implanté cette stratégie l'algorithme d'acceptation-rejet devient,

Algorithme 2.3.2 Méthode d'acceptation-rejet approchée

Entrées: Le nombre d'itérations T.
Poser $t = 1$.
pour $t \leq T$ **faire**
  Générer $\tilde{x}_t$ selon $g(x)$.
  Générer u selon $U_{[0,1]}$.
  si $u \geq \frac{a(\tilde{x}_t)}{g(\tilde{x}_t)}$ **alors**
    Rejeter $\tilde{x}_t$.
  sinon si $u \leq \frac{b(\tilde{x}_t)}{g(\tilde{x}_t)}$ **alors**
    Poser $x_t = \tilde{x}_t$.
  sinon si $u \leq \frac{f(\tilde{x}_t)}{g(\tilde{x}_t)}$ **alors**
    Poser $x_t = \tilde{x}_t$.
  fin
  $t = t + 1$.
fin

L'utilisation des fonctions auxiliaires $a(x)$ et $b(x)$ ne modifie pas les caractéristiques statistiques de l'algorithme d'acceptation-rejet, comme le nombre moyen de candidats à générer. Les gains en efficacité liés à leurs utilisations se limitent à éviter le calcul fastidieux du critère d'acceptation.

Il est intéressant de constater que ces deux derniers algorithmes ont été concus pour générer une suite de variables aléatoires indépendantes des candidats dont l'acceptation est indépendantes de la variable x_{t-1}. Pour ce faire, la décision d'accepter un candidat $\tilde{x}_t$ est prise seulement à l'aide des données générées à l'étape t ; en conséquence, si le candidat est rejeté, ces données sont rejetées et doivent être regénérées à l'étape suivante ce qui alourdit considérablement la tâche à effectuer. D'autres méthodes de simulations, dont les méthodes de Monte-Carlo par chaînes de Markov, présenté au chapitre 3, laissent tomber la contrainte d'indépendance des variables simulées. Cela rend possible la récupération de ces données dans le but de limiter la quantité totale d'opérations à effectuer.

2.4 Échantillonnage adaptatif

L'algorithme d'acceptation-rejet est économique et rapide à condition que la fonction instrumentale $g(x)$ soit une bonne approximation de la fonction cible $f(x)$. En fait, cet algorithme est vraiment efficace lorsque la distance entre les deux fonctions est négligeable, i.e. $|f(x) - g(x)| \sim 0, \quad \forall x \in \chi$.

Or en pratique, il est généralement difficile de trouver une telle fonction. L'algorithme d'échantillonnage adaptatif utilise les propriétés des fonctions log-concaves pour construire une fonction enveloppant supérieurement $f(x)$, ce qui permet de minimiser le nombre de candidats rejetés. Commençons par définir les concepts de fonction log-concave et d'enveloppe d'une fonction.

Définition 1 (Fonction logarithmiquement concave). *Une fonction $f : \mathcal{R}^n \to \mathcal{R}$ est dite logarithmiquement concave, ou plus simplement log-concave, si son domaine est convexe, qu'elle est striquement positive et qu'elle satisfait,*

$$\theta \log f(x) + (1-\theta) \log f(y) \leq \log f(\theta x + (1-\theta)y),$$

pour tout $x, y \in$ domaine de f et $0 < \theta < 1$.

La plupart des distributions usuelles, comme les lois normales, exponentielles ou gamma sont log-concaves, bien que cette propriété puisse dépendre de leurs paramètres. De plus, il est à noter que la fonction de répartition d'une densité log-concave est elle aussi log-concave.

Définition 2 (Enveloppe). *L'enveloppe g d'une fonction f est une fonction satisfaisant,*

$$\log g(x) \geq \log f(x) \quad \forall x \in \chi.$$

La construction de la fonction instrumentale utilisée dans l'échantillonnage adaptatif est basée sur un résultat présenté par Gilks et Wild [1] stipulant que lorsqu'une variable x est univariée et que $E = \{a_1, a_2, ..., a_n\}$ est un ensemble ordonné de points pris sur l'axe des abscisses, alors il est possible de construire une enveloppe $\log g(x)$ de $\log f(x)$ à l'aide de E. Leur méthode consiste à dessiner la tangente t_i de $\log f(x)$ en chaque point $x = a_i$ de E, puis de définir la fonction $\log g(x)$ comme étant l'union des segments des tangentes t_i comprise entre deux éléments a_{i-1} et a_i de E. Par exemple, en utilisant cette technique, une enveloppe de la loi normale centrée réduite est créé en définissant la fonction $y = \log x$ où la variable aléatoire x est distribuée selon $\mathcal{N}(0,1)$. Donc $y = \log \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} = -\frac{1}{2}\log 2 - \frac{1}{2}\log \pi - \frac{x^2}{2}$ et sa dérivée d'ordre un est $y' = -x$. Pour obtenir les segments constituant l'enveloppe, il suffit de calculer l'équation de la tangente à chacun des points de E. La figure 2.3 illustre l'enveloppe, en vert, obtenue lorsque $E = \{-5, 2, 7\}$, ainsi que la courbe à approximer y, dessiné en rouge,
En ajoutant deux points à l'ensemble E, l'enveloppe illustrée par la figure 2.4 est obtenue.

Cet exemple illustre deux caractéristiques importantes de cet algorithme. Premièrement, l'enveloppe contruite par l'algorithme converge vers la fonction $\log f(x)$ lorsque le cardinal de E tend vers l'infini. Deuxièmement, nous voyons qu'aucun des segments de droites formant l'enveloppe ne pénètrent l'aire sous la courbe $\log f(x)$. Cela garantie que les valeurs générées selon $\log g(x)$ sont distribuées selon une distribution calquant $\log f(x)$, même lorsque le cardinal de E est bas. L'algorithme 2.4.1 présente le pseudo-code de cette technique.

1. Gilks, W. R. (1992) Derivative-free adaptive rejection sampling for Gibbs sampling. Bayesian Statistics 4, (eds. Bernardo, J., Berger, J., Dawid, A. P., and Smith, A. F. M.) Oxford University Press.

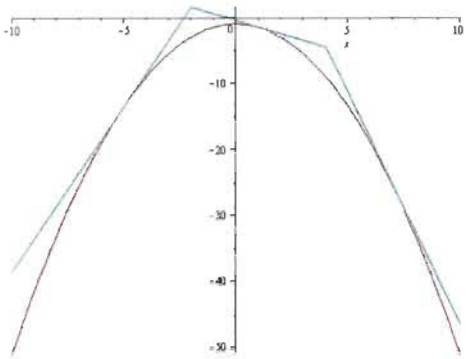


FIGURE 2.3 – Enveloppe construite avec trois tangentes

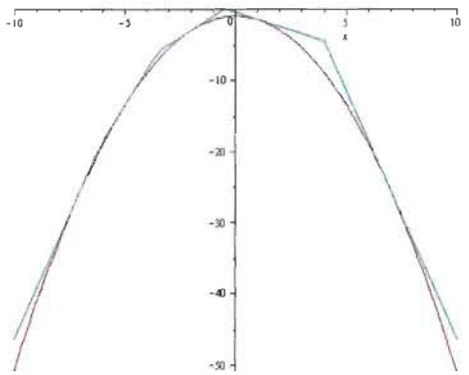


FIGURE 2.4 – Enveloppe construite avec cinq tangentes

Algorithme 2.4.1 Construction de l'enveloppe $\log g(x)$ par la méthode des tangentes.

Entrées: L'ensemble ordonné $E = \{a_1, a_2, ..., a_n\}$.

Poser $i = 1$
pour $i \leq n$ **faire**
  si $i = 1$ **alors**
    Définir s_1 comme étant le segment de droite sur la tangente t_1 compris sur le domaine de $f(x)$ et se terminant au point d'intersection avec la tangente au point $f(a_2)$, notée t_2.
  sinon
    Définir s_i comme étant le segment de droite sur la tangente t_i entre le point d'intersection avec la tangente au point $f(a_{i-1})$, notée t_{i-1}, et le point d'intersection avec la tangente au point $f(a_{i+1})$, t_{i+1}.
  fin
  $i = i + 1$;
fin
Poser $\log g(x) = s_i$ si $\theta \in [a_{i-1}, a_{i+1}]$.

Une méthode alternative, ayant l'avantage de ne pas nécessiter le calcul de la différentielle de f, suit sensiblement la même démarche, mais utilise certaines séquentes aux points $f(a_i)$ pour définir l'enveloppe $\log g(x)$. Cette variante possède les mêmes propriétés de la technique originale et a

pour avantage d'éviter le calcul de l'équation des tangentes formant l'enveloppe, ce qui accélère le processus. La figure 2.5 présente une enveloppe créée à l'aide de la méthode des séquentes et un ensemble de quatre points.

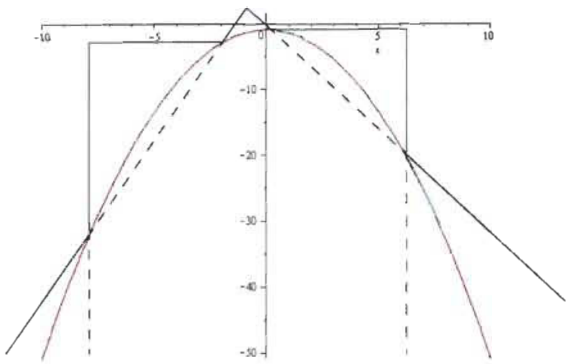


FIGURE 2.5 – Enveloppe construite avec la méthode de la séquente

Algorithme 2.4.2 Construction de l'enveloppe $\log g(x)$ par la méthode des séquentes.

Entrées: L'ensemble ordonné $E = \{a_1, a_2, ..., a_n\}$.
Poser $i = 1$
pour $i \leq n$ **faire**
si $i = 1$ ou $i = n$ **alors**
Définir V_1 comme étant la droite verticale passant par le point $f(a_1)$.
Définir S_1 comme étant la séquente passant par les point $f(a_1)$ et $f(a_2)$.
sinon
Définir V_i comme étant la droite verticale passant par le point $f(a_i)$.
Définir S_i comme étant la séquente passant par les point $f(a_i)$ et $f(a_{i+1})$.
Définir H_i comme étant la droite horizontale passant par le point $f(a_i)$.
fin
Définir s_1 comme étant le segment de droite sur S_1 sur l'interval $[-\infty, a_1]$.
Définir s_2 comme étant l'union du segment de droite sur V_1 compris entre $f(a_1)$ et $f(a_2)$ et du segment de droite sur H_2 compris entre a_1 et a_2.
Définir s_3 comme étant l'union du segment de droite sur S_2 compris entre $f(a_2)$ et l'intersection avec S_3, le segment de droite sur S_3 compris entre le point d'intersection avec S_2 et $f(a_3)$.
...
Définir s_n comme étant l'union du segment de droite sur V_i compris entre $f(a_{n-1})$ et $f(a_n)$ et du segment de droite sur S_n sur l'interval $[a_n, \infty]$.
fin
Poser $\log g(x) = \cup_{i=1}^{n} s_i$.

L'utilisation de cette méthode pour construire la fonction instrumentale de l'algorithme d'acceptation-rejet permet d'incorporer au fur et à mesure à l'ensemble E les points acceptés par l'algorithme. Ces quelques opérations supplémentaires permettent d'améliorer l'approximation de $\log f(x)$ par $\log g(x)$, de limiter radicalement le nombre de candidats rejetés et d'ainsi réduire la quantité de calculs nécessaires pour générer l'échantillon. Donc en utilisant la méthode de la tangente ou de

la séquente, l'algorithme d'acceptation-rejet peut parfois être amélioré. L'algorithme 2.4.3 résume cette technique sous forme de pseudo-code.

Algorithme 2.4.3 Méthode d'échantillonnage adaptatif

Initialiser l'ensemble E avec un minimum de 4 points.
Poser $t = 1$
pour $t \leq T$ **faire**
  Construire l'enveloppe $\log g(x)$ à l'aide de l'algorithme de la séquente ou de la tangente.
  Générer $\tilde{x}_t$ selon $\log g(x)$.
  Générer u selon $U_{[0,1]}$.
  si $u \leq \frac{\log f(\tilde{x}_t)}{\log g(\tilde{x}_t)}$ **alors**
    poser $\theta_t = \exp \tilde{x}_t$.
  sinon
    ajouter $\tilde{x}_t$ à l'ensemble E.
  fin
fin

Bien que l'échantillonnage adaptatif permet d'éviter les problèmes liés à la fonction instrumentale, cette technique nécessite beaucoup d'espace mémoire et de puissance de calculs pour obtenir une enveloppe de qualité. De plus, bien qu'une extension au cas multivariée soit imaginable, aucun algorithme efficace n'a été implémenté jusqu'ici. Ces limitations rendent l'échantillonnage adaptatif rarement utilisable en pratique.

2.5 Échantillonnage préférenciel (Importance sampling)

L'échantillonnage préférentiel consiste à générer un échantillon distribué selon la fonction $g(x)$ puis de le transformer pour qu'il possède les propriétés caractéristiques de la fonction $f(x)$. Pour ce faire, une pondération des candidats à l'aide de poids notés w est utilisé dans le but de compenser la disparité entre des échantillons tirés des fonctions $f(x)$ et $g(x)$. Précisément, pour estimer la fonction $f(x)$, une fonction instrumentale quelconque $g(x)$ est d'abord définie, puis un échantillon de référence $\{x_1, x_2, \ldots, x_i, \ldots, x_n\}$ obtenue par simulation de $g(x)$, sans aucun rejet, est généré. Ensuite, M est estimé à l'aide de la moyenne pondérée,

$$\tilde{M} = \frac{1}{n}\sum_{i=1}^{n} u(x_i)w(x_i) \qquad \text{où } w(x_i) = \frac{f(x_i)}{g(x_i)}. \tag{2.11}$$

La loi des grands nombres nous assure que cette estimation converge asymptotiquement vers la véritable valeur de M, en effet,

$$\begin{aligned} \frac{1}{n}\sum_{i=1}^{n} u(x_i)w(x_i) &= \frac{1}{n}\sum_{i=1}^{n} u(x_i)\frac{f(x_i)}{g(x_i)}, \\ &\to \int u(x)\frac{f(x)}{g(x)}g(x)dx, \\ &= \int u(x)f(x)dx = M. \end{aligned} \tag{2.12}$$

L'effet de la pondération est évidente. Par exemple, supposons que la fonction instrumentale $g(x)$ sur-représente un certain point x, dans l'échantillon utilisé pour estimer M, par rapport à sa représentation dans un échantillon tiré de la loi f. Alors, le poids associé à ce point sera $w(x) = \frac{f(x)}{g(x)} < 1$ et sa représentation dans l'estimation sera importante pour compenser sa sur-représentation dans l'échantillon. Dans le cas contraire, $w(x) = \frac{f(x)}{g(x)} > 1$ et l'importance du point x augmente dans l'estimation du moment de f.

L'estimateur $\tilde{M}$ est non biaisé. tant que $f(x)$ est inclu dans le support de la fonction instrumentale $g(x)$, car,

$$E_g(\tilde{M}) = E_g(u(x)w(x)) = \int_\chi u(x)\frac{f(x)}{g(x)}g(x)dx = \int_\chi u(x)f(x)dx = M. \tag{2.13}$$

L'échantillonnage préférentiel a pour avantages de nous permettre d'atteindre une précision prédeterminée dans l'estimation de la statitique M. En effet, l'estimateur de la variance d'échantillonnage est donné par l'expression,

$$\sigma^2(\tilde{M}) \approx \frac{1}{n}\sum_{i=1}^{n}(u(x_i)w_i - M)^2. \tag{2.14}$$

Le théorème centrale limite implique que la distribution d'échantillonnage de l'estimateur sans biais $\tilde{M}$ peut être approximé par une distribution normale $N\left(M, \sigma^2(\tilde{M})\right)$, en autant que la taille n de l'échantillon soit assez élevée et que $\mathbb{E}_g\left(u(x)\frac{f(x)}{g(x)}\right)$ existe. En utilisant ce résultat, il est aisé de déterminer le nombre d'éléments à échantillonner pour que l'estimation soit incluse dans un intervalle de confiance donné.

Il est aussi possible de trouver la fonction instrumentale minimisant $\sigma^2(\tilde{M})$. Puisque l'estimation du moment $\tilde{M}$ est faite au moyen de la moyenne empirique, $\tilde{M}$ est une somme et sa variance peut être minimisée en minimisant l'expression,

$$\begin{aligned}\sigma_g^2(u(x_i)w(x_i)) &= \mathbb{E}_g\left(u(x_i)^2w(x_i)^2\right) - \left[E_g\left(u(x_i)\right)\right]^2, \\ &= \mathbb{E}_g\left(u(x_i)^2\left(\frac{f(x_i)}{g(x_i)}\right)^2\right) - M^2.\end{aligned} \tag{2.15}$$

Puisque M^2 est indépendante de $g(x)$, cette composante de la variance n'influence pas la valeur du minimum recherché. En appliquant l'inégalité de Jensen sur la première composante de l'égalité 2.15, nous obtenons la borne inférieure,

$$\mathbb{E}_g\left(u(x_i)\left(\frac{f(x)}{g(x)}\right)\right)^2 \leq \mathbb{E}_g(u(x_i)^2\left(\frac{f(x)}{g(x)}\right)^2). \tag{2.16}$$

De par la définition de l'espérance mathématique,

$$E_g\left(u(x_i)^2\left(\frac{f(x)}{g(x)}\right)^2\right) = \int\left(u(x)^2\left(\frac{f(x)}{g(x)}\right)^2 g(x)\right)dx. \tag{2.17}$$

En conséquence, la variance est minimisée lorsque,

$$\int u(x)^2 \left(\frac{f(x)}{g(x)}\right)^2 g(x)dx = \int \left(|u(x)| \left(\frac{f(x)}{g(x)}\right) g(x)\right)^2 dx = \mathbb{E}_g \left(u(x_i) \left(\frac{f(x)}{g(x)}\right)\right)^2. \quad (2.18)$$

Quelques manipulations algébriques appliquées sur cette dernière égalité permettent de trouver la fonction instrumentale minimisant la variance de l'estimation, appelée en anglais *optimal importance distribution*, définie par l'expression

$$g^*(x) = \frac{u(x)f(x)}{\int u(x)f(x)dx}. \quad (2.19)$$

Cette distribution n'est généralement pas directement utilisable. En effet, son utilisation suppose la connaissance de la valeur de $\int u(x)f(x)$, ce qui est essentiellement l'intégrale à calculer. Par contre, cette distribution nous donne une information importante sur le rendement de l'algorithme : celui-ci sera très performant lorsque l'accent est mis sur l'échantillonnage des régions de $g(x)$ qui engendre des valeurs de $|u(x)f(x)|$ relativement large. En pratique, cela signifie qu'il est possible d'obtenir un estimateur de M ayant une variance moins importante que celle de l'estimateur obtenu avec l'échantillonnage simple, si l'accent est mis sur l'échantillonnage de certaines régions d'importances. Cette tactique sera utilisée par les méthodes de ré-échantillonnage. L'algorithme d'échantillonnage préférentiel s'écrit en pseudo-code,

Algorithme 2.5.1 Algorithme d'échantillonnage préférentiel

Entrées: La fonction instrumentale g, le nombre d'itérations T.
Poser $t = 1$
pour $t \leq T$ **faire**
Générer un candidat x_t distribuée selon $g(x)$.
Calculer le poids $w_t = \frac{f(x_t)}{g(x_t)}$.
$t = t + 1$.
fin

2.5.1 Choix de la fonction instrumentale

Puisque la définition de g^* suppose généralement la connaissance de M, cette densité est rarement utilisable et il faut se résoudre à chercher empiriquement une fonction instrumentale minimisant $\phi^2(\tilde{M})$. Heureusement, comme nous l'avons vu précédemment, l'estimateur de la variance échantillonnage est facile à calculer pour chaque fonction candidate, donc il est aisé de choisir parmi plusieurs fonctions potentielles, celle qui semble minimiser la variance. De plus, certaines caractéristiques permettent de discriminer facilement des fonctions candidates. Entre-autre si pour une certaine fonction candidate g, le rapport $\left\{\frac{f(x)}{g(x)}\right\}$ n'est pas borné supérieurement, donc que $sup\left\{\frac{f(x)}{g(x)}\right\} = \infty$, elle doit être écartée. Finalement, avant de débuter la simulation il est important de comparer l'efficacité de cette forme particulière de l'échantillonnage préférentiel avec les méthodes d'échantillonnage d'acceptation-rejet, ainsi qu'avec la méthode d'échantillonnage pondéré approché qui peuvent donner des résultats plus précis et ceci plus rapidement.

TABLE 2.1 – Erreur d'estimation d'un mélange de loi normales par échantillonnage préférentiel

	Moyenne	Erreur relative	Erreur quadratique	Taux d'acceptation
Mélange	6,4			
Échantillonnage pondéré	6,4522	0,0081	0.0400	1

2.5.1.1 Exemple d'application

Dans la présente section, l'échantillonnage préférentiel est utilisé pour générer 100000 points distribués selon le mélange de lois normales présentées à la section 2.3.0.1. La fonction instrumentale utilisée pour générer des candidats est une distribution normale de moyenne $\mu = 7$ et d'écart-type $\sigma = 4$, tel qu'illustré par la figure 2.6. Cette fonction à pour avantage d'être facile à simuler et de couvrir l'ensemble du domaine de la fonction cible.

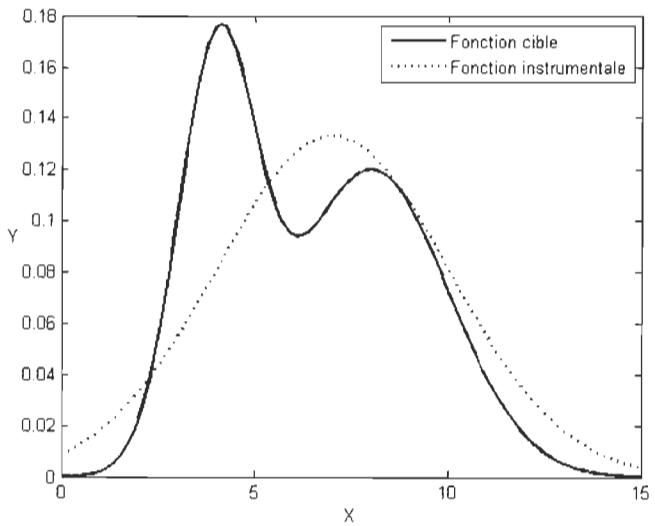


FIGURE 2.6 – Fonctions cible f(X) et instrumentale g(X)

La figure 2.7 présente une estimation des fréquences de la distribution via l'échantillon simulé. Un examen visuel de ce graphique indique que l'échantillon généré par cette technique est distribué selon la distribution d'intérêt. Cette intuition est confirmée par l'estimation de la moyenne de la distribution par l'échantillon tel que présenté dans le tableau 2.1.

2.5.2 Méthode d'échantillonnage préférentiel approché

Généralement, la connaissance de la fonction $f(x)$ à étudier est trop limitée pour nous permettre d'utiliser les méthodes précédentes. En particulier, il est commun de vouloir faire de l'inférence sur une distribution dont la constante de normalisation est inconnue. La méthode d'échantillonnage préférentiel approchée permet d'estimer les propriétés d'une fonction lorsque nous la connaissons à une constante près. Cette variante de l'échantillonnage préférentiel exploite le fait que $f(\theta)$ est une

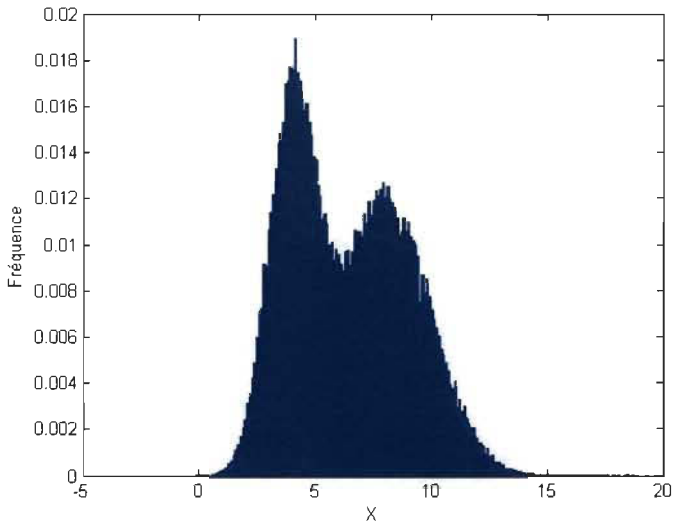


FIGURE 2.7 – Estimation des fréquences de l'échantillon

densité et donc que

$$\mathbb{E}(w_i) = \int \frac{f(x_i)}{g(x_i)} g(x_i) dx_i = \int f(x_i) dx_i = 1. \tag{2.20}$$

En conséquence, en posant $u = 1$ dans la définition de l'estimateur M

$$\tilde{M} = \frac{1}{n} \sum_{i=1}^{n} w_i \sim \mathbb{E}(w_i) = 1, \tag{2.21}$$

donc $\frac{1}{n} \sum_{i=1}^{n} w_i \to 1$ lorsque $n \to \infty$. Cela entraine que l'estimateur approché de M défini par

$$\hat{M} = \frac{\sum_{i=1}^{n} u(x_i) w_i}{\sum_{i=1}^{n} w_i}, \tag{2.22}$$

est un estimateur convergent, mais biaisé, de M lorsque $n \to \infty$. En utilisant les poids w_i dans le calcul de l'estimateur $\tilde{M}$ est donc parfois possible de substituer à l'évaluation des trois fonctions f, g et $\frac{f}{g}$ le calcul d'une seule valeur w_i, ce qui peut non seulement accélérer considérablement la vitesse d'exécution de l'algorithme, mais aussi permettre l'utilisation de l'algorithme d'échantillonage préférentiel lorsque l'information disponible sur f est limitée.

Algorithme 2.5.2 Algorithme d'échantillonnage préférentiel approché

Entrées: $g(x) \sim f(x)$, nombre d'itérations T.
Poser $t = 1$
pour $t \leq T$ **faire**
Générer un candidat x_t distribuée selon $g(\theta)$.
Calculer le poids $w_t = \frac{f(x_t)}{g(x_t)}$.
Poser $t = t + 1$.
fin

2.6 Méthode de ré-échantillonnage pondéré

Supposons, à titre d'exemple, que l'échantillonage préférentiel est utilisé pour créer un échantillon de taille n, $E = \{x_1, x_2, ..., x_n\}$. Remarquons que les éléments de E ne sont pas distribués selon la loi cible $f(x)$, mais selon une approximation de $\hat{f}(x)$ obtenue par une pondération de la fonction intrumentale $g(x)$. Pour obtenir un échantillon de variables aléatoires distribuées selon $f(x)$, il est possible d'utiliser la méthode de ré-échantillonnage pondérées, aussi appelé *Bootstrapping* selon son nom anglais. Cette méthode consiste à utiliser l'ensemble E comme une population de taille finie et à associer à chaque élément de E une probabilité égale à

$$w_r = \frac{f(x_r)/g(x_r)}{\sum_{i=1}^{n} f(x_i)/g(x_i)}. \tag{2.23}$$

Ensuite, ces probabilités sont utilisées pour tirer m éléments, $m < n$, formant l'échantillon recherché. L'algorithme général de cette méthode est le suivant :

Algorithme 2.6.1 Méthode de ré-échantillonnage pondéré

Entrées: n,m et $g(x) \sim f(x)$.
  Poser $t = 1$
  pour $t \leq n$ **faire**
    Générer un candidat $\hat{x}_t$ distribué selon $g(x)$.
    Calculer la probabilité $w_t = \frac{f(x_t)/g(x_t)}{\sum_{t=1}^{m} f(x_t)/g(x_t)}$.
    Poser $t = t + 1$.
  fin
  Poser $t = 1$
  pour $t \leq m$ **faire**
    Tirer un candidat x_j parmis les $\hat{x}_i$ selon les probabilités w_i.
  fin

Les candidats générés par cet algorithme sont distribués selon la loi $f(x)$. Pour s'en convaincre, il suffit de considérer la probabilité conditionnelle,

$$\mathbb{P}(x \in \chi \mid \tilde{x}_1, ..., \tilde{x}_m) = \frac{\sum_{i=1}^{m} w_{\tilde{x}_i} \mathbb{1}_{[\tilde{x}_i \in E]}}{\sum_{i=1}^{m} w_{\tilde{x}_i}}, \tag{2.24}$$

correspondant à la probabilité qu'un élément x généré par cet algorithme soit un élément d'un sous-ensemble quelconque $E \subset \chi$, conditionnellement aux m éléments générés par l'échantillonnage pondéré. Lorsque le nombre d'éléments m tend vers l'infini, la probabilité $\mathbb{P}(x \in \chi \mid \tilde{x}_1, ..., \tilde{x}_m)$ se simplifie de la manière suivante.

$$\begin{aligned}
\lim_{m\to\infty} \mathbb{P}(x \in \chi \mid \tilde{x}_1, ..., \tilde{x}_m) &= \lim_{m\to\infty} \mathbb{P}(x \mid \tilde{x}_1, ..., \tilde{x}_m), \\
&= \lim_{m\to\infty} \frac{m}{m} \frac{\sum_{i=1}^{m} \mathbb{1}_{\{\tilde{x}_i \in E\}} w_{\tilde{x}_i}}{\sum_{i=1}^{m} w_{\tilde{x}_i}}, \\
&= \lim_{m\to\infty} \frac{\mathbb{E}\left(\mathbb{1}_{\{\tilde{x}_i \in E\}} w_{\tilde{x}_i}\right)}{\mathbb{E}\left(w_{\tilde{x}_i}\right)}, \\
&= \lim_{m\to\infty} \frac{\int_E w(\tilde{x}) g(\tilde{x}) d\tilde{x}}{\int w(\tilde{x}) g(\tilde{x}) d\tilde{x}}, \\
&= \lim_{m\to\infty} \frac{\int_E f(\tilde{x}) d\tilde{x}}{\int f(\tilde{x}) d\tilde{x}}, \\
&= \lim_{m\to\infty} \frac{\int_E f(\tilde{x}) d\tilde{x}}{1}. \qquad (2.25)
\end{aligned}$$

Puisque f est une distribution, $\int_E f(\tilde{x}) d\tilde{x}$ est fini et constante. L'espérance d'une fonction constante étant égale à cette constante, i.e. $\mathbb{E}\left(\mathbb{P}(x \mid \tilde{x}_1, ..., \tilde{x}_m)\right) = \int_E f(\tilde{x}) d\tilde{x}$, et que

$$\begin{aligned}
\mathbb{E}\left(\mathbb{P}(x \mid \tilde{x}_1, ..., \tilde{x}_m)\right) &= \int_{\tilde{x}} \int_{x \in E} \mathbb{P}(x \mid \tilde{x}_1, ..., \tilde{x}_m) \mathbb{P}(\tilde{x}_1, ..., \tilde{x}_m) d\tilde{x}, \\
&= \int_{x \in E} \mathbb{P}(x) d\theta = \mathbb{P}(x \in E). \qquad (2.26)
\end{aligned}$$

Donc $\mathbb{P}(x \in E) = \int_E f(x) dx$ ce qui montre que l'algorithme génère des valeurs distribuées selon la distribution cible $f(x)$.

L'estimateur $\tilde{M}$, obtenue en calculant la moyenne empirique de l'échantillon obtenu par la méthode de ré-échantillonnage est non biaisée, mais sa variance est supérieure à celle obtenue par la méthode d'échantillonnage pondérée. En effet, l'étape de ré-échantillonnage consiste à éliminer les candidats ayant une pondération faible et à multiplier la représentation des candidats ayant une pondération élevée. Cette modification de l'échantillon introduit une perturbation dans la distribution des données. Puisque ce bruit ne peut pas être réduit directement par l'algorithme, elle fait augmenter la variance de l'estimateur obtenu par cette méthode.

Les méthodes de Monte-Carlo présentées jusqu'ici sont basées sur des hypothèses très restrictives et peuvent avoir des rendements plutôt médiocres selon le choix de la fonction instrumentale $g(x)$. Comme nous l'avons souligné précédemment, cette lourdeur est due principalement au fait que ces méthodes ont été construites pour générer des suites de réalisations indépendantes. En effet, cette propriété est assurée en régénérant les données nécessaires à la prise de décision au début de chaque nouvelle itération. Donc, lorsque $g(x)$ n'est pas une bonne approximation de la fonction cible, les candidats sont généralement rejetés et la quantité d'informations à simuler devient imposante. Or les données générées à une certaine itération peuvent parfois contenir assez d'informations pour guider le candidat de l'itération suivante vers la fonction cible. Il peut donc être économique de recycler les données d'une itération à l'autre en renonçant à l'indépendance des candidats et en conditionnant la probabilité d'acceptation d'une valeur par rapport à la dernière valeur générée. En particulier, il est possible de sélectionnés le candidat à l'itération i selon les probabilités conditionnelles $\mathbb{P}(\hat{\theta}_i \mid \theta_{i-1})$, donc d'utiliser des chaînes de Markov.

Chapitre 3

Méthodes Monte-Carlos par chaînes de Markov

Les méthodes étudiées dans le présent chapitre utilisent les chaînes de Markovs. Ce choix n'est pas innocent. En effet, l'utilisation des chaînes de Markovs a comme avantage de facilité l'échantillonnage de distribution dont la forme explicite est difficile à obtenir ou bien de haute dimension. La section 3.1 présente la démarche générale derrière les méthodes de Monte-Carlo par chaînes de Markovs. Le restant du chapitre est dédié à la l'étude des algorithmes de Metropolis-Hasting et de Gibbs, ainsi que certaines variantes de ces méthodes.

3.1 Méthodes de Monte-Carlo par chaînes de Markovs

L'approche développée dans cette section consiste à créer une chaîne de Markov dont l'unique distribution stationnaire est celle à étudier, puis de simuler une suite de réalisations de cette chaîne de Markov, notée $\{\theta_1, ..., \theta_n\}$. Puisque la distribution stationnaire de la chaîne $\pi(\theta, \theta')$ est $f(\theta)$ et que la chaîne est apériodique et récurrente, à partir d'un certain N, la suite $\{\theta_t \mid t > N\}$ est distribuée selon $f(\theta)$. De plus, le théorème ergodique assure que l'estimation d'un moment à l'aide d'un échantillon par cette chaîne converge vers la valeur à estimer. Les méthodes de Monte-Carlo tirant avantages de cette démarche sont réunies dans la classe des Monte Carlo Markov Chain ($MCMC$), terme parfois traduit en français par méthodes de Monte-Carlo par chaînes de Markov. Cette section présente la démarche générale de construction de la chaîne convergente désirée, puis les algorithmes les plus populaires basés sur des cas particuliers de ce raisonnement.

3.1.1 Algorithme d'Hastings

La chaîne de Markov que nous voulons construire doit posséder une unique distribution stationnaire π et posséder la propriété ergodique. En conséquence, la chaîne de matrice de transition $M(\theta, \theta')$ et la distribution π doivent idéalement satisfaire la propriété de balance globale $\sum_{\theta \in \Theta} \pi(\theta) M(\theta, \theta') = \pi(\theta')$. Or la sommation de cette équation est longue à évaluer lorsque le cardinal de Θ est élevé. Il est donc préférable d'utiliser la propriété de réversibilitée.

$$\pi(\theta) M(\theta, \theta') = \pi(\theta') M(\theta', \theta) \quad \theta, \theta' \in \Theta. \tag{3.1}$$

Cette propriété implique la balance global, donc que π est bien une distribution stationnaire de la chaîne. De plus, pour que π soit unique, la chaîne doit être apériodique et irréductible. Généralement, une telle chaîne existe pour chaque distribution π ; la difficulté est de la construire rapidement.

En 1970 W. Keith Hastings [1] montra qu'il existe une méthode simple permettant de créer une telle chaîne dans le cas où l'espace des états Θ est fini. Sa méthode, généralisant les travaux de E. Teller, N. Metropolis et A Rosenbluth [2], consiste à définir une matrice de transition d'une chaîne quelconque $A(\theta, \theta')$, servant de fonction instrumentale, ayant un espace d'état Θ et une matrice $B(\theta, \theta')$, servant de matrice d'acceptation dans l'algorithme, défini par

$$B(\theta, \theta') = \begin{cases} 0 & \text{si } A(\theta, \theta') = 0, \\ \min\left\{1, \frac{\pi(\theta')A(\theta', \theta)}{\pi(\theta)A(\theta, \theta')}\right\} & \text{sinon} \end{cases} \tag{3.2}$$

Ensuite, la matrice de transition $M(\theta, \theta')$ de la chaîne de Markov désirée est construite en deux étapes. D'abord, les éléments de M qui ne sont pas sur la diagonale principale sont obtenus en multipliant les matrices A et B,

$$M(\theta, \theta') = A(\theta, \theta')B(\theta, \theta') \quad \text{pour } \theta \neq \theta'. \tag{3.3}$$

Les éléments sur la diagonale principale sont obtenus par soustraction pour s'assurer que la somme des chaque colonne et chaque ligne de la matrice de transition soit égale à 1. Il est facile de vérifier que la matrice ainsi obtenue respecte la propriété d'équilibre détaillé. En effet,

$$\text{si } \theta = \theta' \rightarrow \pi(\theta)M(\theta, \theta') = \pi(\theta')M(\theta', \theta') = \pi(\theta')M(\theta', \theta). \tag{3.4}$$

Dans le cas contraire,

$$\begin{aligned} \pi(\theta)M(\theta, \theta') &= \pi(\theta)A(\theta, \theta') \min\left\{1, \frac{\pi(\theta')A(\theta', \theta)}{\pi(\theta)A(\theta, \theta')}\right\}, \\ &= \pi(\theta)A(\theta, \theta') \min\left\{\pi(\theta)A(\theta, \theta'), \pi(\theta')A(\theta', \theta)\right\}, \\ &= \pi(\theta)A(\theta, \theta') \min\left\{\pi(\theta')A(\theta', \theta), \pi(\theta)A(\theta, \theta')\right\}, \\ &= \pi(\theta')M(\theta', \theta). \end{aligned}$$

Donc π est une distribution stationnaire de $M(\theta, \theta')$. Pour que la chaîne de Markov déterminée par M soit ergodique et que π soit unique, une restriction est imposée, soit que la matrice de transition respecte,

$$M(\theta', \theta) \neq 0 \quad \forall \theta, \theta' \in \Theta. \tag{3.5}$$

Cette caractéristique implique l'irréductibilité et l'apériodicité de la chaîne, donc qu'elle est ergodique. Clairement, les réalisations de la chaîne de Markov ayant pour matrice de transition M suivent un processus ayant π comme loi limite ergodique invariante. Traduite en un algorithme, la démarche de Hastings s'écrit ;

1. W.K. Hastings, Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57, 97-109, (1970).

2. E Teller, N Metropolis, A Rosenbluth - J. Chem. Phys, (1953)

Algorithme 3.1.1 Algorithme d'Hastings

Entrées: La graine de l'algorithme θ_0 ; Le nombre d'itérations T.
Poser $t = 1$.
pour $t \leq T$ **faire**
À l'étape t, il est supposé que la chaîne est dans l'état $X_t = \theta_t$.
Générer un candidat $x^{'}$ pour l'état subséquent selon les probabilités de la fonction instrumentale $A(\theta_t, \theta^{'})$.
Générer $u \approx U_{[0,1]}$.
si $u \leq B(\theta_t, x^{'})$ **alors**
Accepter $x_{t+1} = \theta^{'}$.
sinon
$x_{t+1} = x_t$.
fin
fin

Dans la littérature, cet algorithme est parfois écrit sous une forme plus générale qui inclue le cas où les états sont constitués de plusieurs composantes. Les états sont alors notés sous forme de vecteurs $\theta = (\theta_1, \theta_2, ..., \theta_i, ...\theta_j)$ et la notation,

$$\theta_{-i} = (\theta_1, \theta_2, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_j) \tag{3.6}$$

est utilisée pour désigner le vecteur des états sans la composante θ_i. La généralisation s'opère en assignant des matrices A_i et B_i, telles que définies précédemment, à chacune des composantes du vecteur. À chaque étape de l'algorithme, chacune des composantes est alors mises à jour indépendamment des autres, par leur paire de matrices respective.

Algorithme 3.1.2 Algorithme d'Hastings par composante

Entrées: La graine de l'algorithme θ_0 ; Le nombre d'itérations T.
Poser $t = 0$
pour $t \leq n$ **faire**
À l'étape t, il est supposé que la chaîne est dans l'état $\theta_t = (\theta_1^t, \theta_2^t, ..., \theta_i^t, ...\theta_j^t)$.
Poser $i = 1$
pour $i \leq j$ **faire**
Générer un candidat $\theta_i^{'}$ pour l'état subséquent selon la fonction instrumentale marginale $A_i(\theta_i^t, \theta^{'})$.
Générer $u \approx U_{[0,1]}$.
si $u \leq B(\theta_i^t, \theta^{'})$ **alors**
Accepter $X_i^t = \theta^{'}$
sinon
$X_{i+1}^t = \theta_i^t$
fin
fin
fin

De plus, en utilisant la densité conditionnelle complète, définie par,

$$\pi(\theta_i \mid \theta_{-i}) = \frac{M(\theta_i, \theta_{-i})}{\sum M(\theta_i, \theta_{-i})}, \tag{3.7}$$

il est possible de réécrire les éléments constituant la matrice B. c'est-à-dire les probabilités d'acceptation d'un candidat,

$$B(\theta, \theta') = \min\left\{1, \frac{\pi(\theta_i' \mid \theta_{-i}) A_i(\theta, \theta_i')}{\pi(\theta_i \mid \theta_{-i}) A_i(\theta', \theta)}\right\}. \tag{3.8}$$

Cette approche est développée dans l'algorithme de Gibbs présenté dans la section 3.3.

En terminant, il est à noter qu'outre sa souplesse, une méthode de Monte-Carlo basée sur cette construction comporte deux avantages notables par rapport aux méthodes vues précédemment. D'abord, l'absence de restriction imposée sur la matrice de transition A donne beaucoup de flexibilité à l'utilisateur dans le choix de la fonction instrumentale lors de son application. De plus, puisque la chaîne peut rester dans le même état durant plusieurs étapes, le "gaspillage" de candidats que nous retrouvions dans les méthodes d'échantillonnage par rejet est minimisé. Cela entraine des gains considérables du point de vue du temps de calcul lorsque la fonction instrumentale est très près de la fonction cible $f(x)$. La généralisation de ces algorithmes au cas où l'espace des états Θ est infini est appelée algorithme de Metropolis-Hastings et est présenté dans la prochaine section.

3.2 Algorithme de Metropolis-Hastings

Premier algorithme utilisant les chaînes de Markovs pour rendre plus versatile les méthodes de Monte-Carlo, l'algorithme de Metropolis-Hasting correspond à un algorithme de Hastings transposé au cas où la chaîne simulée se déplace dans un espace des états Θ infini. La démarche menant au choix du noyau de la chaîne de Markov à simuler est sensiblement la même que dans le cas où Θ est fini : pour une certaine fonction instrumentale $g(\theta \mid \theta')$ est défini la fonction $h(\theta, \theta') = \pi(\theta) g(\theta \mid \theta')$ déterminant les probabilités d'acceptations. Pour que la chaîne soit ergodique, son noyau doit respecter la condition de réversibilité détaillée.

$$\pi(\theta)\, g\left(\theta' \mid \theta\right) h\left(\theta, \theta'\right) = \pi\left(\theta'\right) g\left(\theta \mid \theta'\right) h\left(\theta', \theta\right) \quad \forall\theta, \forall\theta', \theta \neq \theta'. \tag{3.9}$$

En manipulant cette équation, nous trouvons la condition suivante sur h,

$$\frac{h\left(\theta, \theta'\right)}{h\left(\theta', \theta\right)} = \frac{\pi\left(\theta'\right) g\left(\theta \mid \theta'\right)}{\pi(\theta)\, g\left(\theta' \mid \theta\right)}. \tag{3.10}$$

Cette condition est suffisante pour assurer que π est une loi stationnaire de h. Dans l'algorithme, la probabilité d'acceptation est donnée par l'expression,

$$h\left(\theta, \theta'\right) = \min\left\{\frac{\pi\left(\theta'\right) g\left(\theta \mid \theta'\right)}{\pi(\theta)\, g\left(\theta' \mid \theta\right)}, 1\right\}, \tag{3.11}$$

où le minimum permet de garantir que h génère des probabilités. De plus, pour respecter les conditions d'irréductibilité et d'apériodicité, donc assurer l'unicité de π, il suffit de construire le noyau différentiel tel que,

$$k\left(\theta,\theta^{'}\right)=g\left(\theta^{'}\mid\theta\right)h\left(\theta,\theta^{'}\right)+h\left(\theta,\tilde{\theta}\right)g\left(\tilde{\theta}\mid\theta\right)\mathbb{1}_{\theta}(d\theta^{'})\neq 0\quad\forall\theta,\theta^{'}\in\Theta. \tag{3.12}$$

Cette condition est respectée en utilisant la fonction minimum sur la probabilité d'acceptation. Sous forme de pseudo-code, l'algorithme s'écrit comme suit :

Algorithme 3.2.1 Algorithme de Metropolis-Hastings

Entrées: La graine de l'algorithme θ_0 ; Le nombre d'itération T.
t=1
pour $t\leq T$ **faire**
Générer le candidat $\tilde{\theta}_t$ selon la loi $g\left(\tilde{\theta}_t\mid\theta_{t-1}\right)$.
Générer le critère d'acceptation a selon la loi $\min\left\{\frac{\pi(\tilde{\theta}_t)g(\theta_{t-1}|\tilde{\theta}_t)}{\pi(\theta_{t-1})g(\tilde{\theta}_{t-1}|\theta_{t-1})},1\right\}$.
Générer $u\approx U_{[0,1]}$.
si $u\leq a$ **alors**
$\theta_i=\tilde{\theta}_i$.
sinon
$\theta_i=\theta_{i-1}$.
fin
fin

La série $\{\theta\}_{i\geq 1}$ ainsi générée suit une chaîne de Markov de noyau,

$$k\left(\theta,d\theta^{'}\right)=g\left(\theta^{'}\mid\theta\right)h\left(\theta,\theta^{'}\right)+\left(1-\int_{\tilde{\theta}}h\left(\theta,\tilde{\theta}\right)g\left(\tilde{\theta}\mid\theta\right)d\tilde{\theta}\right)\mathbb{1}_{\theta}(d\theta^{'}), \tag{3.13}$$

où $\mathbb{1}_{d\theta^{'}}=$ est la fonction indicatrice définie sur l'interval $\left[\theta^{'},\theta^{'}+d\theta^{'}\right]$, ie :

$$\mathbb{1}_{d\theta^{'}}=\begin{cases}1 & \text{si } \theta\in\left[\theta^{'},\theta^{'}+d\theta^{'}\right]\\ 0 & \text{sinon.}\end{cases}$$

3.2.1 Choix de la fonction instrumentale $g(\theta,\theta^{'})$

Bien que cette méthode soit beaucoup plus efficace du point de vue computationnel que la méthode d'acceptation-rejet, sa vitesse de convergence vers la fonction à simuler est directement liée au choix de la fonction instrumentale g. En pratique, trois classes de fonctions instrumentales sont utilisées :

1. $g\left(\theta\mid\theta_{t-1}\right)=g\left(\theta\right)$. Dans ce cas, le tirage se fait indépendament du point de départ comme dans le cas de la méthode d'acceptation-rejet.
2. $g\left(\theta\mid\theta_{t-1}\right)=g\left(\theta-\theta_{t-1}\right)$, i.e. une marche aléatoire homogène.
3. Une fonction majorante $M\left(t\right)$ log-concave par morceaux, telle que définie à la section 2.4.

3.2.2 Cas particuliers

Il existe plusieurs variations de l'algorithme de Metropolis-Hasting, chacune se différenciant par le type de fonction instrumentale choisie. Les méthodes les plus souvent rencontrées sont les suivantes :

3.2.2.1 Algorithme de Metropolis

L'algorithme de Metropolis a pour objectif de générer des candidats à partir d'une fonction instrumentale facile à simuler, puis de les discriminer très rapidement. Pour ce faire, elle utilise des fonctions instrumentales symétriques tel que $g(\theta \mid \theta') = g(\theta^{'} \mid \theta)$ pour tout $\theta, \theta^{'} \in \Theta$. Dans ce cas, le critère de discrimination h est simplifié et s'écrit

$$h(\theta \mid \theta^{'}) = \min \left\{ 1, \frac{\pi(\theta^{'})}{\pi(\theta)} \right\}. \tag{3.14}$$

Une caractéristique de cette égalité est que la partie non constante du minimum est réduite à un rapport entre les deux distributions cibles, ce qui rend son calcul beaucoup plus rapide. En effet, il est généralement plus aisé de calculer le ratio de deux distributions que d'évaluer une fonction en deux points différents puis de faire le ratio, à moins que la distribution soit particulièrement simple. Un exemple couramment utilisé en pratique est l'algorithme de Metropolis simulant une marche aléatoire utilisant la fonction instrumentale

$$g(\theta^{'} \mid \theta) = g(\mid \theta - \theta^{'} \mid). \tag{3.15}$$

3.2.2.2 Algorithme de Metropolis indépendant

Ce type d'algorithme comprend toutes les méthodes dont la fonction instrumentale conditionnelle $g(\theta^{'} \mid \theta)$ ne dépend pas de la variable θ, donc $g(\theta^{'} \mid \theta) = g(\theta^{'})$. Dans ce cas, le critère de discrimination des éléments à ajouter à l'échantillon h s'écrit, en utilisant la notation $w(\theta) = \frac{\pi(\theta)}{g(\theta)}$

$$h(\theta^{'}, \theta) = \min \left\{ 1, \frac{w(\theta^{'})}{w(\theta)} \right\}. \tag{3.16}$$

Le rendement de cette méthode, varie grandement selon la qualité de l'approximation de la fonction $\pi(\theta)$ par la fonction instrumentale $g(\theta)$. De plus, pour éviter que certaines régions de $\pi(\theta)$ soient surreprésentées dans l'échantillon, il est préférable de s'assurer que la queue de la distribution $g(\theta)$ soit importante pour assurer un déplacement de la chaîne sur tout le support de la distribution.

3.2.2.3 Algorithme de Metropolis-Hastings par composante

En général, les éléments à générer sont de dimension $k > 1$. Dans ce cas, tout dépendant de la forme de la distribution cible, il peut être avantageux de séparer les candidats en k composantes $\theta = (\theta_1, \theta_2, ..., \theta_k)$ et de générer de nouveaux candidats en mettant à jour successivement chacune des composantes. Cette méthode est obtenue en modifiant l'algorithme de Metropolis-Hastings de sorte que chaque itération est séparée en k sous étapes et en réécrivant la probabilité d'acceptation sous la forme

$$h(\theta_{-i}, \theta_i, \theta_i^{'}) = \min \left\{ 1, \frac{\pi(\theta_i^{'} \mid \theta_{-i}) g_i(\theta_i \mid \theta_i^{'}, \theta_{-i})}{\pi(\theta_i \mid \theta_{-i}) g_i(\theta_i^{'} \mid \theta_i, \theta_{-i})} \right\}. \tag{3.17}$$

où θ_{-i} correspond au vecteur $\theta = \{\theta_1, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_n\}$.

Algorithme 3.2.2 Algorithme de Metropolis-Hastings par composante

Entrées: La graine de l'algorithme θ_0 ; Le nombre d'itérations T.

Poser $t = 1$
pour $t \leq n$ **faire**
  Poser $i = 1$
  pour $i \leq k$ **faire**
    Générer le candidat $\tilde{\theta}_t^i$ selon la loi $g_i\left(\tilde{\theta}_t^i \mid \theta_{t-1}^i\right)$.
    Générer le critère d'acceptation a selon la loi $h(\theta_{-t}, \theta_t, \tilde{\theta}_t^i) = \min\left\{1, \frac{\pi(\tilde{\theta}_t^i|\theta_{-t}) g_t(\theta_t|\tilde{\theta}_t^i, \theta_{-t})}{\pi(\theta_t|\theta_{-t}) g_t(\tilde{\theta}_t^i|\theta_t, \theta_{-t})}\right\}$.
    Générer $u \approx U_{[0,1]}$.
    si $u \leq a$ **alors**
      $\theta_t = \tilde{\theta}_t^i$.
    sinon
      $\theta_t = \theta_{t-1}^{i-1}$.
    fin
  fin
fin

3.2.3 Exemple d'application

La distribution à étudier est un mélange de deux lois normales bivariées respectivement de moyenne $\mu_1 = (4, 4)$ et $\mu_1 = (8, 4)$ et de matrice de covariance $\left(\begin{smallmatrix} 1 & 0.3 \\ 0.3 & 1 \end{smallmatrix}\right)$ et $\left(\begin{smallmatrix} 2 & 0.4 \\ 0.4 & 2 \end{smallmatrix}\right)$, avec un coefficient de mixage $\alpha = 0.4$. La figure 3.1 illustre la projection de la densité sur le plan $X1 \times X2$.

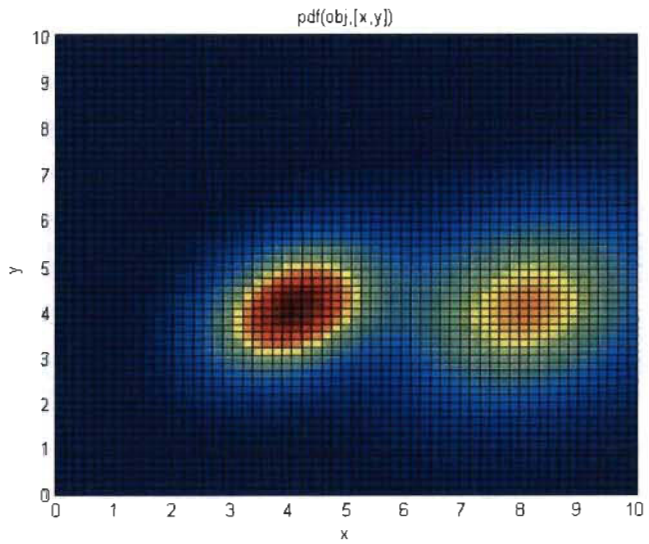


FIGURE 3.1 – Projection de la distribution à étudier.

La simulation de cette distribution, telle qu'illustrée par la figure 3.2 a été faite à l'aide d'un échantillon de 500 points obtenus par l'algorithme de Metropolis-Hasting avec comme fonction d'exploration une marche aléatoire $\mathcal{N}(0,1)$, une période de chauffe de 500 itérations, une ventilation de 5 itérations et avec $x_0 = (6,6)$.

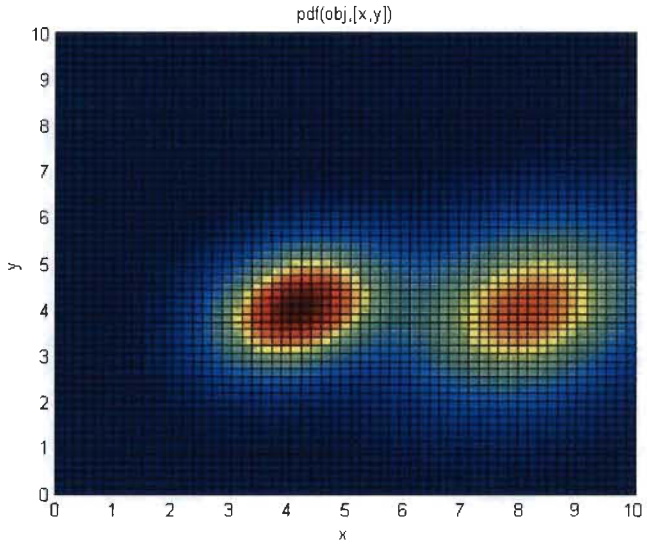


FIGURE 3.2 – Projection de la distribution de l'échantillon.

Une comparaison visuelle des deux densités suggère que l'échantillonnage préférentiel génère un échantillon distribué selon le mélange de lois normales. Cette intuition est confirmée par l'application du test de Kolmogorov-Smirnov sur l'estimation de la distribution, ainsi que des distributions marginales des variables X_1 et X_2. Les p-values de ces tests, ainsi que la valeur de l'estimation de la moyenne de la distribution sont résumées dans le tableau 3.2.3.

	Moyenne	Erreur	Test marge x1	Test marge x2	Test loi jointe
Mélange	(6.4,4)				
Metropolis-Hasting	(6.4250, 4.0486)	0.05472	$p < 0.001$	$p < 0.001$	$p < 0.001$

3.3 Algorithme de Gibbs

En posant $A(\theta, \theta') = \pi(\theta_i' \mid \theta_{-i})$ dans l'algorithme d'Hasting par composante 3.1.1, nous obtenons une version particulière de l'algorithme appelé algorithme de Gibbs d'après la distribution du même nom, utile en physique et qui fut l'objet des premières simulations faites avec cette méthode. La principale conséquence du choix de la matrice $A(\theta, \theta')$ est que la probabilité d'acceptation d'un candidat quelconque est toujours égale à

$$B(\theta, \theta') = \min\left\{1, \frac{\pi(\theta_i' \mid \theta_{-i})\pi(\theta_i \mid \theta_{-i})}{\pi(\theta_i \mid \theta_{-i})\pi(\theta_i' \mid \theta_{-i})}\right\} = 1. \tag{3.18}$$

Donc cette méthode tire des candidats de la distribution conditionnelle complète $\pi(\theta_i \mid \theta_{-i})$ et accepte chacun des candidats potentiels. La généralisation au cas où θ est infini dénombrable est immédiate : il suffit de poser $g(\theta_i, \theta_i^{'}) = \pi(\theta_i \mid \theta_i^{'})$ dans l'algorithme de Metropolis-Hastings.

Lorsque chacun des états possède deux composantes $\theta = (\theta_1, \theta_2)$, en supposant que les composantes sont distribuées selon la distribution jointe $P(\theta_1, \theta_2)$ et que les distributions marginales de θ_1 et θ_2 sont respectivement $P(\theta_1)$ et $P(\theta_2)$ algorithme s'écrit sous la forme suivante.

Algorithme 3.3.1 Algorithme de Gibbs

H

Entrées: La graine de l'algorithme θ_2^0 ; Le nombre d'itérations T.

Poser $t = 1$

pour $t \leq T$ **faire**

Générer θ_1^{t+1} en simulant la loi conditionnelle $P\left(\theta_1 \mid \theta_2^t\right)$.

Générer θ_2^{t+1} en simulant la loi conditionnelle $P\left(\theta_2 \mid \theta_1^{t+1}\right)$.

Poser $\theta^{t+1} = \left[\theta_1^{t+1}\theta_2^{t+1}\right]$.

fin

La suite des couples $\{(\theta_1, \theta_2)\}$ ainsi générés, suivent une chaîne de Markov de noyau :

$$k\left((\theta_1, \theta_2), \left(\theta_1^{'}, \theta_2^{'}\right)\right) = P\left(\theta_1^{'} \mid \theta_2\right) P\left(\theta_2^{'} \mid \theta_1\right).$$

De plus, il est possible de trouver la distribution stationnaire des suites de composantes. Pour ce faire, les équations marginales $P(\theta_1)$ et $P(\theta_2)$ sont réécrite sous la forme,

$$P(\theta_1) = \int P(\theta_1, \theta_2) d\theta_2 = \int P(\theta_1 \mid \theta_2) P(\theta_2) d\theta_2 \quad \text{et} \quad P(\theta_2) = \int P(\theta_2 \mid \theta_1) P(\theta_1) d\theta_1.$$

En combinant ces deux équations, nous obtenons,

$$\begin{aligned} P(\theta_1) &= \int P(\theta_1 \mid \theta_2) \int P(\theta_2 \mid \theta_1) P(\theta_1) d\theta_1 d\theta_2, \\ &= \int_{\theta_1} \int_{\theta_2} P(\theta_1 \mid \theta_2) P(\theta_2 \mid \theta_1) P(\theta_1^{'}) d\theta_1^{'} d\theta_2, \\ &= \int k_1(\theta_1, \theta_1^{'}) P(\theta_1^{'}) d\theta_1^{'}. \end{aligned} \tag{3.19}$$

avec $k_1(\theta_1, \theta_1^{'}) = \int_{\theta_2} P(\theta_1 \mid \theta_2) P(\theta_2 \mid \theta_1^{'}) d\theta_2$. L'équation 3.19 étant l'équation de balance globale, chaque suite de composantes θ_i a la distribution marginale $P(\theta_i)$ comme densité stationnaire.

3.3.1 Conditions de convergences

En écrivant le paramètre à estimer θ sous la forme d'un vecteur à deux dimensions de composantes θ_1, θ_2,

$$\begin{aligned}\int_{\theta} \theta k\left(\theta,\theta^{'}\right) d\theta &= \int_{\theta_1,\theta_2} P\left(\theta_1^{'} \mid \theta_2\right) P\left(\theta_2^{'} \mid \theta_1^{'}\right) P\left(\theta_1,\theta_2\right) d\theta_1 d\theta_2, \\ &= \int_{\theta_1} \left(\int_{\theta_2} P\left(\theta_1^{'} \mid \theta_2\right) P\left(\theta_2^{'} \mid \theta_1^{'}\right) P\left(\theta_2\right) d\theta_2\right) P\left(\theta_1 \mid \theta_2\right) d\theta_1,\end{aligned} \tag{3.20}$$

mais,

$$\int_{\theta_2} P\left(\theta_1^{'} \mid \theta_2\right) P\left(\theta_2^{'} \mid \theta_1^{'}\right) P\left(\theta_2\right) d\theta_2 = P\left(\theta_2^{'} \mid \theta_1^{'}\right) \underbrace{\int_{\theta_2} P\left(\theta_1^{'},\theta_2\right) d\theta_2}_{\text{densité marginale de } \theta_1^{'}} = P\left(\theta_1^{'} \mid \theta_2^{'}\right) P\left(\theta_1^{'}\right).$$

Donc,

$$\int_{\theta} P\left(\theta\right) k\left(\theta,\theta^{'}\right) d\theta = P\left(\theta_2^{'} \mid \theta_1^{'}\right) P\left(\theta_1^{'}\right) \int_{\theta_1} P\left(\theta_1 \mid \theta_2\right) d\theta_1 = P\left(\theta_1^{'},\theta_2^{'}\right) = P\left(\theta^{'}\right). \tag{3.21}$$

Ce qui montre que la balance globale est respectée. De plus, puisque l'algorithme génère des composantes successives selon les probabilités $P\left(\theta \mid \theta^{'}\right) \geq 0$, le noyau respecte la propriété de positivité $k\left(\theta,\theta^{'}\right) \neq 0, \quad \forall\left(\theta,\theta^{'}\right)$. Donc, cet algorithme génère une suite de réalisations suivant une trajectoire Markovienne dont $P\left(\theta_1,\theta_2\right)$ est la distribution stationnaire.

3.3.2 Cas général

La généralisation à des composantes de dimensions k quelconque est immédiate. Il suffit de séparer chaque itération en k sous-étapes et d'y générer la composante θ_i selon la loi conditionnelle complète $\pi(\theta_i \mid \theta_{-i}) = \pi(\theta_i \mid \theta_1, \theta_1, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_k)$ où les $\theta_1, ..., \theta_i$ sont les composantes mises à jour.

Algorithme 3.3.2 Algorithme de Gibbs général

Entrées: la graine de l'algorithme θ_2^0 ; Le nombre d'itérations T.
Poser $t = 1$
pour $t \leq T$ **faire**
Générer θ_1^{t+1} en simulant la loi conditionnelle $\pi\left(\theta_1 \mid \theta_2^t, ..., \theta_k^t\right)$.
Générer θ_2^{t+1} en simulant la loi conditionnelle $\pi\left(\theta_2 \mid \theta_1^{t+1}, \theta_3^{t+1}, ..., \theta_k^t\right)$.
...
Générer θ_k^{t+1} en simulant la loi conditionnelle $\pi\left(\theta_k \mid \theta_1^{t+1}, \theta_2^{t+1}, ..., \theta_{k-1}^{t+1}\right)$.
Poser $\theta^t = \left[\theta_1^{t+1}, \theta_2^{t+1}, ..., \theta_k^{t+1}\right]$.
fin

La chaîne de Markov créée par cet algorithme est déterminée par le noyau $k(\theta,\theta^{'}) = \prod_{i=1}^{k} \pi\left(\theta_i \mid \theta_{-i}\right)$. Il a été prouvé [1] que la suite des candidats ainsi générés forment une chaîne de Markov dont la

1. GO Roberts, AFM Smith - Stochastic Processes and their Applications, 1994 - Elsevier, par exemple.

distribution stationnaire de dimension k est $\pi(\theta) = P(\theta_1, \theta_2, ..., \theta_k)$. En conséquence, la distribution marginale des θ_i est la distribution stationnaire des sous-suite $\{\theta_i^j\}$, mais ces sous-suites ne forment pas une chaîne de Markov. Par mesure d'économie ces résultats ne seront pas prouvés dans le présent document.

3.3.3 Algorithme de Metropolis-Hastings dans l'algorithme de Gibbs

Lors de l'utilisation de l'algorithme de Gibbs, il est souvent difficile de calculer directement ou bien de simuler la fonction conditionnelle complète. Une manière de contourner ce problème est d'utiliser l'algorithme de Metropolis-Hastings indépendante, pour échantillonner des candidats distribués selon la fonction conditionnelle complète $\pi(\theta_t \mid \theta_{-t})$.

Algorithme 3.3.3 Algorithme de Metropolis-Hastings dans l'algorithme de Gibbs

Entrées: La graine de l'algorithme θ_0, $g(\theta_i \mid \theta_{-i})$; Le nombre d'itérations T ; le nombre d'itérations T_1 de la composante de Metropolis.
Poser $t = 1$.
pour $t \leq T$ **faire**
  Poser $i = 1$.
  pour $i \leq k$ **faire**
    $j = 1$.
    pour $j \leq T_1$ **faire**
      Générer le candidat $\tilde{\theta}_j$ selon la loi $g\left(\tilde{\theta}_j\right)$.
      Générer le critère d'acceptation a selon la loi $\min\left\{\frac{\pi(\theta_j|\theta_{-j})g(\theta_{j-1})}{\pi(\theta_{j-1}|\theta_{-(j-1)})g\left(\tilde{\theta}_j\right)}, 1\right\}$.
      Générer $u \approx U_{[0,1]}$.
      si $u \leq a$ **alors**
        $\theta_j = \tilde{\theta}_j$.
      sinon
        $\theta_j = \theta_{j-1}$.
      fin
    fin
    return $\tilde{\theta}$.
    Poser $\theta_i^t = \tilde{\theta}$.
  fin
  Poser $\theta^t = \left[\theta_1^{t+1}, \theta_2^{t+1}, ...\theta_k^{t+1}\right]$.
fin

Utilisé seul, l'algorithme de Metropolis-Hastings permet d'échantillonner à partir de la distribution conditionnelle complète, qu'elle soit univariée ou multivariée. Utilisé avec l'algorithme de Gibbs, il permet de créer une nouvelle chaîne de Markov ayant $f(\theta)$ comme distribution stationnaire.

3.3.4 Exemple d'application

Tout comme pour l'algorithme de Métropolis-Hasting, nous allons utiliser l'algorithme de Gibbs pour générer un échantillon de 500 points distribués selon un mélange de deux lois normales bivariées

décrites à la section 3.2.3. La figure 3.2 illustre le résultat d'une simulation faite avec une période de chauffe de 500 itérations, une ventilation de 5 itérations et $x_0 = (6, 6)$ comme point d'entré.

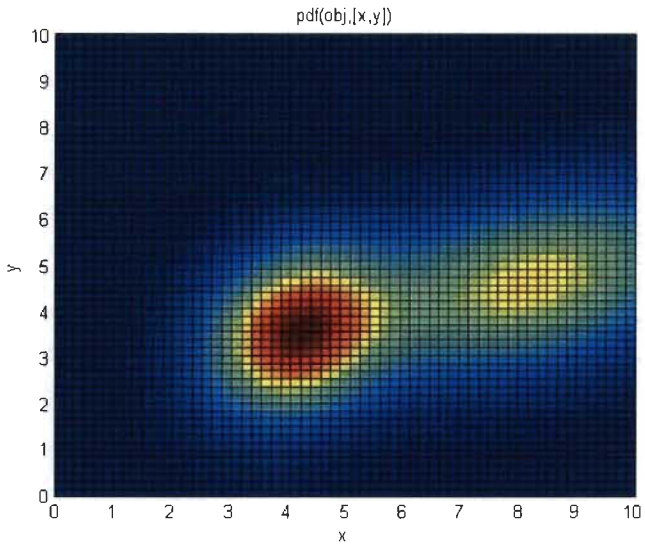


FIGURE 3.3 – Projection de l'échantillon

Puisque la fonction d'exploration de l'algorithme de Gibbs se déplace parallèlement aux axes et que cet algorithme accepte tous les candidats, cette dernière a une nette propension à accepter les points sur la frontière de la distribution. Cette caractéristique rend l'algorithme particulièrement performant pour l'échantillonnage de distribution à queue épaisse, mais dans le cas qui nous intéresse cela ce traduit par un échantillon plus diffus que l'échantillon obtenu par l'algorithme de Métropolis-Hasting. Le tableau 3.3.4 présente l'estimation de la moyenne, ainsi que les p-values des résultats du test de Kolmogorov-Smirnov appliqués aux distributions marginales, ainsi qu'à la distribution bivariée.

	Moyenne	Erreur	Test marge x1	Test marge x2	Test loi jointe
Mélange Gibbs	(6.4,4) (6,3940,4,0776)	0.0778	$p < 0.001$	$p < 0.001$	$p < 0.001$

3.4 Échantillonnage directionnel

Le rôle des chaînes de Markovs dans les algorithmes de Monte-Carlo est de parcourir le support de la distribution cible pour générer les candidats potentiels de l'échantillon. Selon leurs constructions, certaines méthodes auront plus de difficultés à parcourir le support de certaines fonctions cibles ayant une forme particulière. Par exemple, l'algorithme de Gibbs est basé sur la mise à jour successive de chacune des composantes des candidats. En conséquence, durant les mises à jour, l'algorithme se déplace parallèlement aux axes de coordonnées. Si la fonction cible à un support majoritairement parallèle aux axes, la chaîne la parcourra beaucoup plus rapidement que si la fonction ayant une forme plus exotique. Dans ces situations, l'utilisation d'une méthode s'adaptant à la forme de π sera plus performante que les méthodes vues jusqu'ici.

En pratique, bien que la capacité de calculs des ordinateurs a considérablement augmentée depuis l'introduction des *MCMC* dans l'industrie dans les années 1990, certains modèles, particulièrement les modèles hiérarchiques, demandent beaucoup de simulations. En conséquence, un ralentissement du taux de mixage de la chaîne peut facilement ralentir l'algorithme au point qu'il soit inutilisable. Les algorithmes d'échantillonnages directionnels suivants permettent de limiter le ralentissement de la convergence liée à la forme de la distribution ciblée.

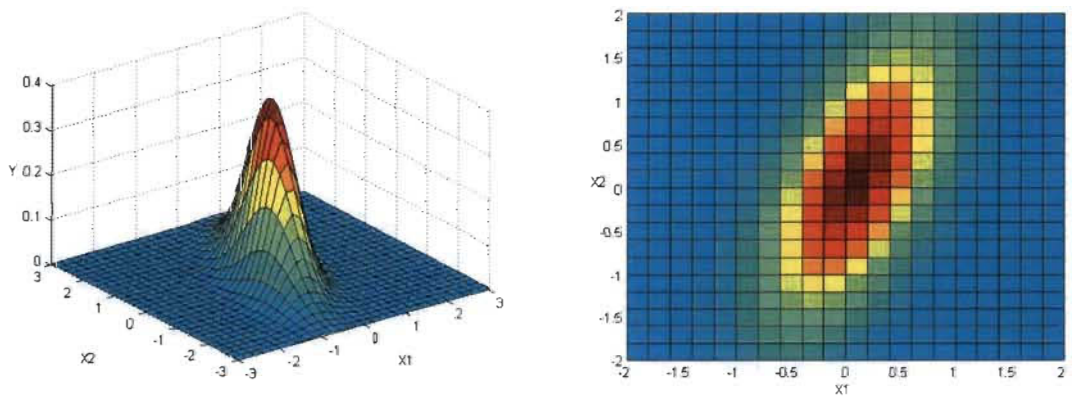
3.4.1 Algorithme hit and run

Cette méthodes est simplement un algorithme d'échantillonnage basé sur une marche aléatoire dans l'espace $\mathbb{R}^k$.

Algorithme 3.4.1 Algorithme hit and run

Entrées: La graine $\theta_0 \in \Theta$; Le nombre d'itérations T.
t=1
pour $t \leq n$ **faire**
Générer un vecteur unitaire e_t de dimension k.
Générer un scalaire s selon une loi uniforme défini sur le support de $f(\theta)$.
Générer un point p_t selon la loi conditionnelle complète $\pi(p_t \mid \theta_{t-1} + se_t)$.
Poser $\theta_t = \theta_{t-1} + p_t e_t$.
fin

L'algorithme consiste à tirer une direction dans $\mathbb{R}^k$ au hasard, puis de tirer le nouveau candidat selon les probabilités de la distribution conditionnelle complète se situant sur une droite passant par θ_{t-1}, $\theta_{t-1} + s_t e_t$. La figure 3.5 illustre comment l'algorithme hit and run génère un point distribué selon une loi normale fortement corrélée, illustrée par les figures 3.4a et 3.4.



(a) Distribution normale bivariée corrélée. (b) Projection sur le plan $X1 \times X2$

FIGURE 3.4 – Graphique de la loi normale bivariée à simuler.

Si la distribution $\pi(\theta_{t-1} + ae_t)$ est difficile à manipuler, une fonction tampon $h(\theta) \propto \pi(\theta_{t-1} + ae_t)$ peut être utilisée sans affecter les résultats. De plus, le vecteur e_t est généralement tiré selon deux méthodes simples. La première consiste à générer k réalisations d'une loi normale centrée réduite

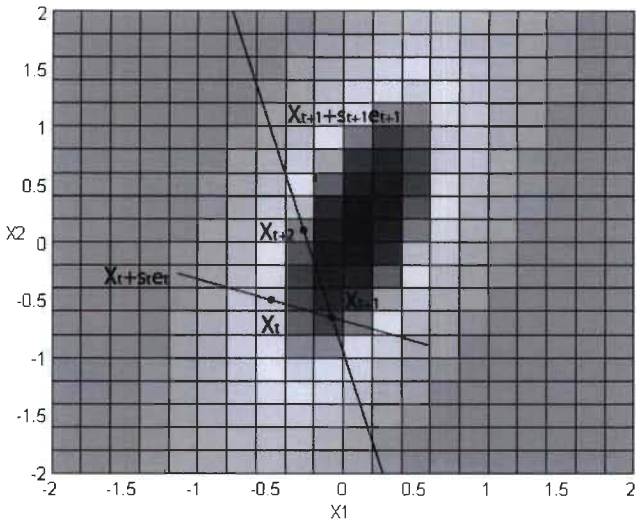


FIGURE 3.5 – Parcours de l'algorithme hit and run

z_i où $i = 1, ..., k$ puis de poser $e_t = \left\{ \frac{z_1}{\sqrt{\sum z_1^2}}, ..., \frac{z_n}{\sqrt{\sum z_i^2}} \right\}$. Cette méthode à l'avantage de centrer le vecteur e_t sur la sphère unité de dimension k. La deuxième méthode suit le même raisonnement, mais utilise une loi uniforme défini sur $[0, 1]^k$ dans le but d'accélérer la génération du vecteur e_t. L'utilisation de cette méthode entraine que certaines directions ont plus de chance d'être empruntées par l'algorithme, ce qui ralentit le taux de mixage de la chaîne.

Bien que cet algorithme soit simple à implémenter et donne d'excellents résultats pour des espaces de petites dimensions, il est à éviter lorsque la distribution à échantillonner présente des variations importantes ou lorsque la dimensionnalité est élevée.

3.4.2 Algorithme du snooker

Cette variation de l'algorithme hit-and-run a pour particularité de stocker k valeurs déjà générées dans un ensemble $E = \left\{\theta_i^1, ..., \theta_i^k\right\}$, puis de les utiliser pour déterminer le vecteur directionnel. L'algorithme consiste à tirer deux éléments de E au hasard, puis d'échantillonner un point se trouvant sur la droite reliant ces deux éléments. La figure 3.6 montre comment un point peut être généré à l'aide de l'algorithme du snooker utilisant un ensemble de cinq points. Puisque les éléments de E sont des points distribués selon la fonction cible π, la probabilité que les éléments utilisés dans l'algorithme fassent partie d'une région de haute densitée augmente au fur et à mesure que l'algorithme est utilisé. En conséquence, la droite reliant ces éléments à tendance à s'enligner avec la forme de la distribution et les points ainsi générés ont plus de chance d'être acceptés.

Dans certains cas, cet algorithme est plus efficace que l'algorithme de Gibbs ou l'algorithme de Metropolis-Hastings. Malheureusement, les calculs supplémentaires requis pour obtenir des candidats le rend peu approprié pour l'échantillonnage de distribution simple et sa performance varie grandement d'une situation à une autre. Il est donc recommandé d'utiliser l'algorithme du snooker seulement dans les situations où les méthodes plus traditionnelles sont peu efficaces. De plus, il est

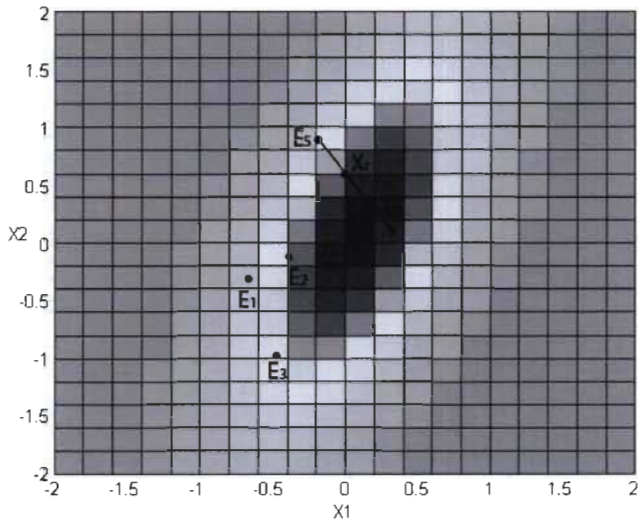


FIGURE 3.6 – Parcours de l'algorithme du snooker.

Algorithme 3.4.2 Algorithme du snooker

Entrées: L'ensemble de k points choisi au hasard sur le support de π ; $E = \left\{\theta_0^1, \theta_0^2, ..., \theta_0^k\right\}$; Le nombre d'itérations T.

Poser $t = 1$

pour $t \leq T$ **faire**

Sélectionner au hasard deux éléments θ_t^a et θ_t^b de l'ensemble E.

Poser $e_t = \theta_t^a - \theta_t^a$.

Générer un point p_t selon la loi $\pi(\theta_t^a + se_t)|1 - s^{k-1}|$.

Poser $\theta_{t+1}^a = \theta_t^a + p_t e_t$.

Poser $\theta_{t+1}^b = \theta_t^b$ pour $b \neq a$.

fin

impératif que le cardinal de l'ensemble E soit fini pour garantir que l'algorithme converge vers π. Cette condition s'explique facilement en considérant de la généralisation de cet algorithme, appelé algorithme de Metropolis adaptatif.

3.4.3 Algorithme de Metropolis adatatif

Cette méthode d'échantillonnage est une méthode adaptive, comme l'algorithme du snooker, mais qui utilise l'algorithme de Metropolis-Hastings pour mieux choisir les éléments faisant partie de l'ensemble de points E au temps t. Cette généralisation à comme particularité d'avoir comme distribution stationnaire la distribution de k valeurs indépendantes distribuées selon π.

Algorithme 3.4.3 Algorithme adaptatif de Metropolis

Entrées: L'ensemble de k points choisis au hasard sur le support de π, $E = \left\{\theta_0^1, \theta_0^2, ..., \theta_0^k\right\}$; La graine θ_0 ; Le nombre d'itérations T.
Poser $t = 1$
pour $t \leq T$ **faire**
Générer k points selon la fonction intrumentale $g\left(\{\theta_t^{(i)}\}_{i=1}^k = \theta_t^{(1)}, ..., \theta_t^{(k)} | \{\theta_{t-1}^{(i)}\}_{i=1}^k = \theta_{t-1}^{(1)}, ..., \theta_{t-1}^{(k)}\right)$
Calculer la probabilité d'acceptation $a = \min\left\{1, \frac{g(\{\theta_{t-1}^{(i)}\}_{i=1}^k | \{\theta_t^{(i)}\}_{i=1}^k) \prod_{i=1}^k \pi(\{\theta_{t-1}^{(i)}\}_{i=1}^k)}{g(\{\theta_t^{(i)}\}_{i=1}^k | \{\theta_{t-1}^{(i)}\}_{i=1}^k) \prod_{i=1}^k \pi(\{\theta_t^{(i)}\}_{i=1}^k)}\right\}$.
Générer $u \approx U_{[0,1]}$.
si $u \leq a$ **alors**
Poser $E_t = \{\theta_t^{(i)}\}_{i=1}^k$.
sinon
Poser $E_t = \{\theta_{t-1}^{(i)}\}_{i=1}^k$.
fin
fin

La fonction instrumentale $g\left(\{\theta_t^{(i)}\}_{i=1}^k | \{\theta_{t-1}^{(i)}\}_{i=1}^k\right)$ est la distribution de k points conditionnés sur les points de l'étape précédente. La figure 3.7 illustre comment l'algorithme opère à l'aide d'un ensemble de cinq points. Dans cet exemple, la fonction instrumentale est une loi normale centrée sur la moyenne des cinqs points de l'étapes précédente. Le cercle représente la région ayant une haute probabilité de contenir les points générés à cette itération.

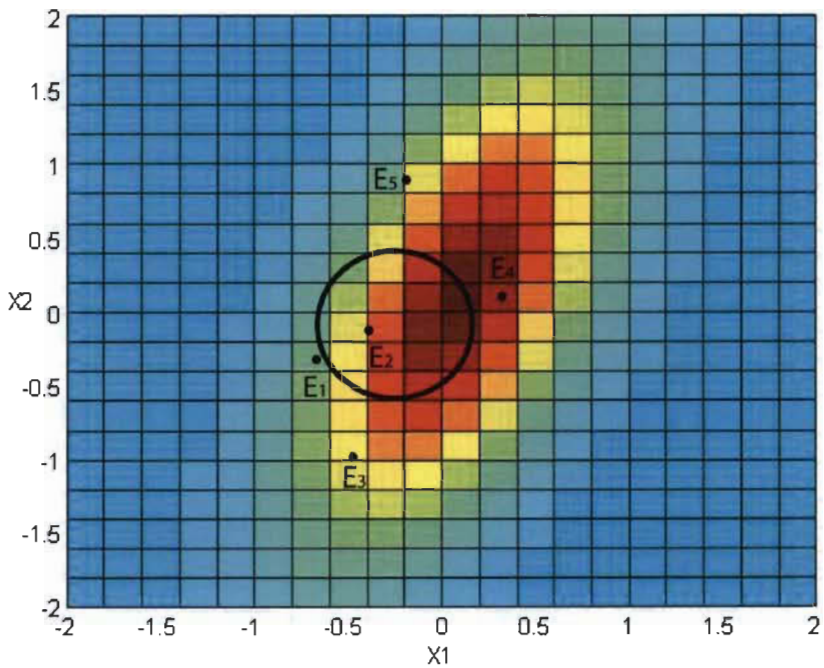


FIGURE 3.7 – L'algorithme de Metropolis adaptatif.

Lorsque le cardinal de l'ensemble E est très grand ou infini, il est possible de prouver que cet algorithme n'est pas convergent. Cela s'explique par le fait que, lors des premières itérations, les candidats générés ne sont pas distribués selon la fonction cible. Donc en s'adaptant à ces informations, l'algorithme génère d'autres valeurs mal distribuées et les incorpores dans E ce qui perpétue le problème et nuit à la convergence.

Chapitre 4

Utilisation des MCMC

Lors de l'utilisation des méthodes Monte-Carlo par chaînes de Markov, plusieurs contingences peuvent influencer leurs performances ainsi que la qualité des estimations. Ce chapitre est dédié à l'étude des difficultés rencontrées en implantant les MCMCs et aux diverses techniques permettant de les éviter. En particulier, les sujets abordés sont les paramètres de l'implémentation ainsi que les méthodes permettant de déterminer les valeurs à donner à ces variables, ainsi que les divers tests pouvant être utilisés pour juger de la qualité de la simulation.

4.1 Détails de l'implémentation

Les performances des méthodes de Monte-Carlo par chaîne de Markov sont grandement influencées par les choix faits par l'utilisateur. En effet, outre le choix de la méthode la plus apte à simuler la fonction d'intérêt et de la fonction instrumentale, l'utilisateur doit aussi choisir les paramètres de la simulation. Cette section présente les variantes d'implémentation des méthodes de Monte-Carlo par chaînes de Markov ayant une influence sur les performances des algorithmes. Pour illustrer l'effet de ces variantes, des échantillons de variables aléatoires distribuées selon la distribution mixte,

$$f(x) = \frac{2}{3}\frac{1}{2\sqrt{2\pi}}e^{\left(\frac{-1}{2}\left(\frac{x-7}{2}\right)^2\right)} + \frac{1}{3}(1.5)e^{-1.5x}, \tag{4.1}$$

simulés par l'algorithme de Metropolis-Hasting avec les diverses modalités seront présentés à titre d'exemple.

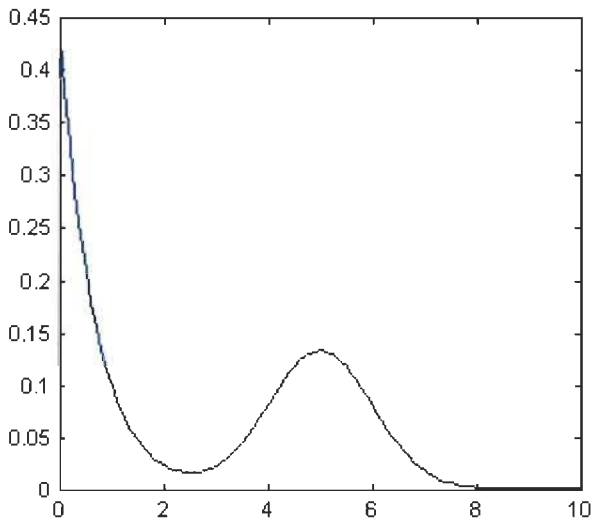


FIGURE 4.1 – Mixture d'une loi $\mathcal{N}(7,2)$ et d'une loi exponentielle de paramètre $\lambda = 1.5$.

4.1.1 Période de transition

Toute simulation MCMC peut être divisée en deux périodes, soit la période de transition, où les états visités par la chaîne sont dépendants de l'état initial de la chaîne, et la période de simulation où la chaîne à atteint sa distribution stationnaire π. Les candidats générés lors de l'étape de transition, appelée *burn-in* en anglais, ne peuvent pas être utilisés lors de l'échantillonnage, puisqu'ils ne sont pas distribués selon π. Or le nombre d'itérations formant l'étape de transition, dont la durée est noté n_0, n'est pas une valeur constante et facile à déterminer a priori. C'est donc à l'utilisateur de l'algorithme de vérifier empiriquement si la période de transition est terminée et s'il est donc possible de commencer l'échantillonnage de π. Une erreur dans l'estimation de t_0 entraine une augmentation de l'erreur d'estimation, si l'échantillonnage commence durant la période de transition, ou bien une augmentation du temps de calcul, la période de transition est prolongée inutilement. La figure 4.2 présente les résultats de deux simulations de la distribution 4.1, la première faite sans période de transition, l'autre avec une période de transition de 500 itérations. Il est clair qu'après 1000

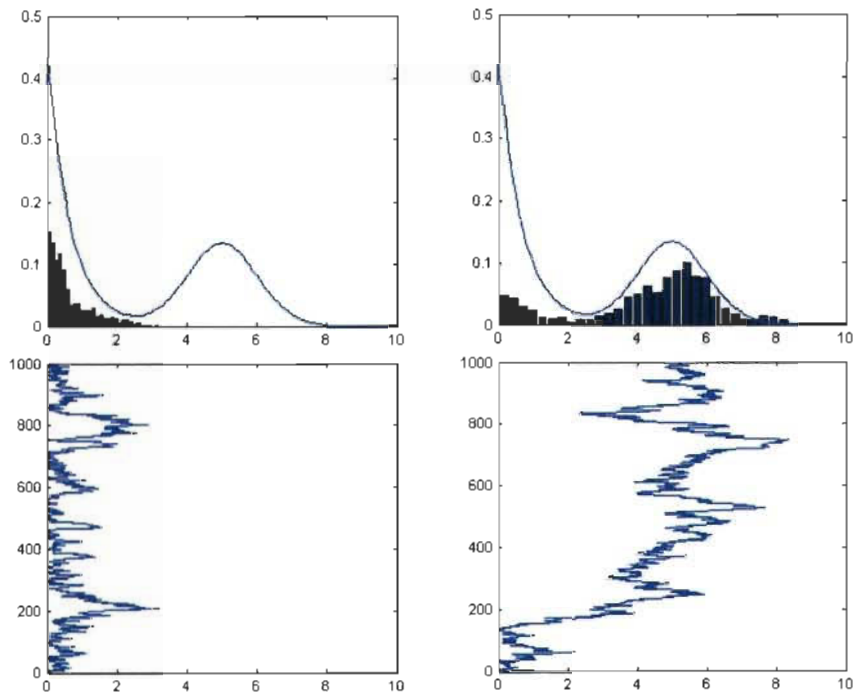


FIGURE 4.2 – Fréquences et série chronologiques de deux simulations de 4.1 avec respectivement $n_0 = 0$ et $n_0 = 500$.

itérations, dans les deux cas, la chaîne n'a toujours pas atteint son état stationnaire. Cependant, cet exemple illustre bien l'importance de la période de chauffe. En effet, le deuxième échantillon enveloppe bien tout le support de l'équation 4.1 et ses fréquences correspondent mieux à celles de cette dernière que l'échantillon obtenu sans rejet des itérations de la période de transition.

4.1.2 État initial

À chaque itération, les algorithmes MCMCs tirent des candidats conditionnellement aux états précédents de la chaîne. Pour démarrer l'algorithme, il est donc nécessaire de choisir le premier état dans lequel se trouve la chaîne. Puisqu'en théorie l'échantillonnage débute lorsque la chaîne a atteint son état stationnaire, la valeur initiale ne devrait pas influencer les résultats de l'algorithme. Par contre, cette valeur initiale, aussi appelée graine de l'algorithme et noté x_0, peut tout de même avoir de l'influence sur la vitesse de convergence de l'algorithme et nécessite donc d'être choisie judicieusement. Pour ce faire, il est conseillé de générer plusieurs chaînes de Markovs, chacune ayant leur propre valeur initiale et de conserver la chaîne ayant sélectionner des points dans le plus grand nombre de sous-ensemble du support de la fonction cible. Les valeurs initiales à tester sont généralement choisies au hasard, ou bien uniformément sur le support de π, mais lorsque certaines informations sont disponibles sur la fonction cible, il peut être préférable de choisir ces valeurs initiales en fonction de ces informations. Par exemple, si les observations d'un phénomène distribuée selon f nous porte à croire que le mode de f est inclu dans un intervalle I, le choix d'une valeur $x \in I$ comme valeur de départ peut être très avantageux, à condition que l'hypothèse concernant le mode soit vrai.

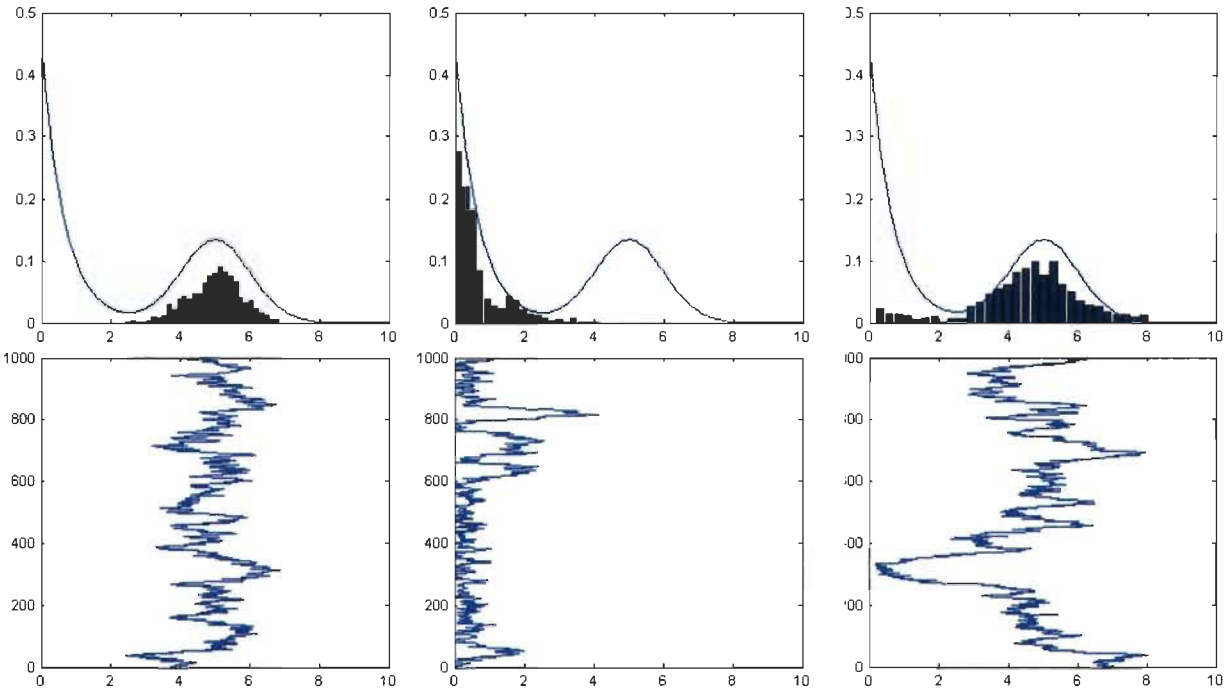


FIGURE 4.3 – Simulations de 1000 itérations avec respectivement $x_0 = 4$, $x_0 = 0$ et $x_0 = 7$.

La figures 4.3 illustre le résultats de trois simulations sans rejet de la période de transition avec avec respectivement $x_0 = 4$, $x_0 = 0$ et $x_0 = 7$ comme état initial. Ces graphiques montre le comportement de la fonction instrumentale en fonction de l'état initiale : cette dernière se déplace moins rapidement sur le support de la distribution à simuler lorsqu'elle est initialement dans un intervalle de haute probabilité.

4.1.3 Dilution de l'échantillon

Lorsque les candidats acceptés par un algorithme MCMC sont très corrélés entre eux, chaque nouveau candidat accepté apporte très peu d'information à l'échantillon. Par mesure d'économie, il peut être utile de conserver en mémoire seulement une fraction des candidats acceptés par l'algorithme. Cette opération de dilution de l'échantillon consiste à multiplier le nombre de valeurs à générer par i et d'enregistrer chaque $i^{\text{ème}}$ valeur générée par l'algorithme. La figure 4.4 montre l'effet de la dilution de l'échantillonnage sur trois simulations de 1000 itérations ; la première avec une période de transition de 500 itérations et une dilution de 5 itérations, la deuxième avec une période de transition de 4500 itérations et aucune dilution et la troisième avec une période de transition de 3500 itérations et une dilution de 2 itérations. Ces résultats montrent qu'en diluant l'échantillon, la fonction instrumentale a l'occasion de parcourir plus souvent le support de la distribution cible, donc les points conservés reflètent mieux la densité de π.

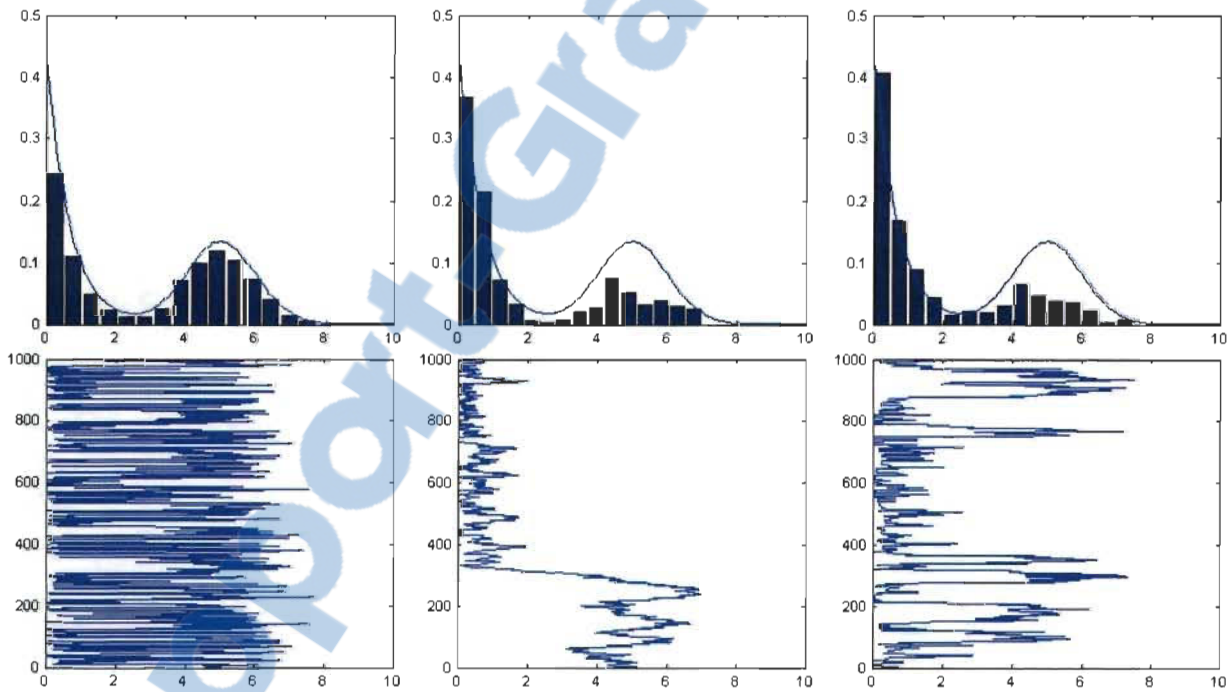


FIGURE 4.4 – Simulations de 1000 itérations avec respectivement une dilution de 5, 0 et 2 itérations.

4.1.4 Nombre de chaînes simulées en parallèle

Pour s'assurer que la chaîne simulée soit bien en régime stationnaire et que l'échantillonnage soit valide, deux stratégies sont couramment utilisées. La première consiste à simuler longtemps une seule chaîne, d'observer ses caractéristiques pour s'assurer qu'elles soient stables et coïncident avec les caractéristiques connues de la fonction cible. La deuxième consiste à effectuer parallèlement plusieurs simulations d'une même chaîne de Markov et de considérer le régime stationnaire atteint lorsque toutes les simulations partagent les mêmes caractéristiques de π. Avec la montée en popularité des architectures informatiques mettant a profit les processeurs travaillant en parallèle, qui

a grandement facilité la simulation de plusieurs chaînes de Markovs simultanément, la deuxième stratégie est de plus en plus utilisée, mais cette dernière ne présente pas d'avantage théorique par rapport à la première. Il est donc recommandé de faire son choix en fonction du modèle à analyser et d'effectuer des simulations préliminaires.

4.1.5 Ordre de mise à jour des composantes

Jusqu'à présent il fut supposé que la mise à jour de l'algorithme de Gibbs et de l'algorithme de Metropolis-Hastings par composantes se fait selon l'index donné aux composantes. Donc le premier composant est mis à jour en premier, la deuxième composante en deuxième, etc. Or bien que la convergence de la chaîne vers π n'est pas affectée par un changement de l'ordre des mises à jour, sa vitesse de convergence, ainsi que la vitesse d'exploration du support de la fonction f peut en être grandement améliorée. Par exemple, si les composantes de $\theta = \{\theta_1, ..., \theta_i, ..., \theta_n\}$ sont corrélées avec une certaine composantes θ_i, alors les valeurs de θ aurons tendance à être aussi corrélées, ce qui vient ralentir la convergence de la chaîne vers sa distribution stationnaire. Dans ce cas, il est préférable d'alterner des cycles où seule la composante θ_i est mise à jour lors d'une itération avec des cycles où toutes les composantes sont mises à jour selon un ordre quelconque. Lorsque l'information sur la corrélation des composantes n'est pas disponible, lors de l'implémentation d'une méthode MCMC générale par exemple, un processus aléatoire est généralement utilisé pour déterminer l'ordre de mise à jour des composantes.

4.2 Diagnostic de convergence

La justesse des estimations obtenues par les méthodes de Monte-Carlo par chaîne de Markov repose sur le comportement asymptotique des chaînes. Donc, en plus d'avoir à déterminer si l'estimateur du moment $\hat{M}$ obtenu par moyenne empirique a convergé près de M, dans un contexte MCMC, il primordiale de déterminer si la chaîne à bel et bien convergé vers sa distribution stationnaire. Les tests présentés dans cette section permettent de vérifier cette hypothèse.

4.2.1 Calcul a priori du nombre d'itérations

La méthode suivante permet de déterminer le nombre d'itérations nécessaires pour que l'estimation de la distribution a posteriori d'un certain quantile q ait une précision prédéterminée. Cette application est particulièrement importante dans le contexte où les quantiles sont souvent une des seules caractéristiques connues de la fonction cible et où il est important de s'assurer que l'échantillon généré par la méthode MCMC partage ces caractéristiques de la fonction qu'elle simule. Accessoirement, cette démarche rend possible l'estimation de bornes inférieures pour la longueur de l'étape de transition, pour la longueur des simulations préliminaires, ainsi que pour la simulation en entier.

Supposons que nous disposons des résultats d'une simulation préliminaire, que nous voulons estimer la valeur d'un quantile q avec une marge d'erreur maximale de ϵ et une probabilité de p et que nous sommes interessés à déterminer la longueur de l'étape de transition n_0 et de la simulation T. Supposons aussi que les informations disponibles sur la fonction cible indiquent que le quantile

$q = x$. En posant $X_t = f(\theta_t)$, à chaque itération la valeur z_t est calculée,

$$z_t = \begin{cases} 1 & \text{si} \quad X_t < x \\ 0 & \text{sinon.} \end{cases} \tag{4.2}$$

La série $\{Z_t \mid t = 1, 2, ..., T\}$ est une série binaire, mais n'est pas elle-même une chaîne de Markov, car ses éléments lorsque pris successivement sont trop dépendant. En effet, lorsqu'un candidat θ_t est accepté, de par la construction des algorithmes MCMCs les candidats suivants sont généralement échantillonnés dans un voisinage de θ_t. Donc, si $X_t \leq x$, la probabilité que les X_{t+i} suivant soient plus petit que x est non négligeable. Les éléments z_{t+i} sont donc dépendent de la valeur de z_t, même lorsque $i > 1$.

Ce problème de dépendance peut être contourné en diluant suffisamment l'échantillon, donc en créant la nouvelle série des $k^{\text{ème}}$ éléments de $\{Z_t\}$, noté $\left\{Z_t^k \mid Z_t^k = Z_{1+(t-1)k}\right\}$. Pour déterminer la valeur de k à utiliser pour obtenir une série adéquate, il est conseillé de générer les séries $\left\{Z_t^k\right\}$ pour $k = 1, 2, ...T/2$. Ensuite pour chaque valeur de k, le test des ratios de vraisemblances, noté G_k^2, des modèles de chaînes de Markov de premiers ordres par rapport à celui d'une chaîne de second ordre est calculé. Le critère d'information bayésien est utilisé pour comparer les modèles,

$$BIC_k = G_k^2 - 2\log T. \tag{4.3}$$

En général, le plus petit k qui favorise le modèle de la chaîne de Markov de premier ordre est sélectionné. Ainsi, en posant $\left\{Z_t^k \mid Z_t^k = Z_{1+(t-1)k}\right\}$, la nouvelle série des $k^{\text{ème}}$ éléments de $\{Z_t\}$ se comportera approximativement comme une chaîne de Markov. La série $\left\{Z_t^k\right\}$ a pour matrice de transition,

$$M = \begin{bmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{bmatrix}, \tag{4.4}$$

où α est la probabilité que la chaîne passe du premier état au deuxième état et β est la probabilité que la chaîne passe du deuxième état au premier état. Les résultats classiques concernant le comportement asymptotique des chaînes de Markov nous assurent que la distribution stationnaire de cette chaîne est donnée par l'expression,

$$\pi = (\pi_0, \pi_1) = (\alpha + \beta)^{-1}(\beta, \alpha), \tag{4.5}$$

où $\pi_0 = P(X \leq x)$ et $\pi_1 = 1 - \pi_0$. En conséquence, la matrice de transition de i étape est donnée par l'expression,

$$M^i = \begin{bmatrix} \pi_0 & \pi_1 \\ \pi_0 & \pi_1 \end{bmatrix} + \frac{\lambda^i}{\alpha+\beta} \begin{bmatrix} \alpha & -\alpha \\ -\beta & \beta \end{bmatrix}, \tag{4.6}$$

où $\lambda = (1 - \alpha - \beta)$. Pour que l'estimation du quantile ait une marge d'erreur ϵ, il faut qu'il existe un certain entier b tel que,

$$|P(z_b^k = i \mid z_0^k = j) - \pi_i| \leq \epsilon \quad i, j = 0, 1. \tag{4.7}$$

En assumant que $\lambda > 0$, cette condition se traduit par,

$$\alpha^b \leq \frac{(\alpha+\beta)\epsilon}{\max(\alpha,\beta)} \Longleftrightarrow b = \frac{\log\left\{\frac{(\alpha+\beta)\epsilon}{\max(\alpha,\beta)}\right\}}{\log((1-\alpha-\beta))}. \tag{4.8}$$

Donc, en tenant compte de la dilution de l'échantillon $\{Z_t^k\}$, nous constatons que cette simulation nécessite une période de transition de longueur $n_0 = bk$ pour générer un échantillon représentatif. De plus, en remanquant qu'il est possible d'utiliser la moyenne empirique pour estimer la valeur du quartile, par la loi des grands nombres, pour un n assez grand, l'estimateur

$$\hat{Z}^k = \frac{1}{n}\sum_{t=1}^{n} z_t^k, \tag{4.9}$$

est distribué selon une loi normale de moyenne q et de variance $\frac{(2-\alpha-\beta)\alpha\beta}{n(\alpha+\beta)^2}$. Donc pour obtenir une estimation contenue dans l'intervalle $[q-\epsilon, q+\epsilon]$ avec une probabilité de p et en notant la fonction de répartition normale usuelle Φ, le nombre d'itérations à effectuer T est égal à,

$$T_0 = \frac{(2-\alpha-\beta)\alpha\beta}{n(\alpha+\beta)^2}\left\{\frac{\Phi^{-1}(\frac{1}{2}(p+1))}{\epsilon}\right\}^2. \tag{4.10}$$

En conséquence, $T = T_0 k$ si l'échantillon a été dilué. Cette estimation de T ne tient pas compte des différentes contingences qui ralentiessent la vitesse de convergence de la chaîne vers sa distribution stationnaire. En conséquence, il est préférable de considérer T_0 et n_0 respectivement comme une borne inférieure du nombre d'itérations à faire et une borne inférieure de la durée de la période de transition. De plus, cette formule permet d'estimer la durée minimale de la simulation préliminaire. Cette estimation est basée sur le fait que la valeur de T est minimisée lorsque les éléments de la série z_t sont indépendants. En effet. dans ce cas $\alpha = 1-\beta = \pi_1 = 1-q$ et la formule de n se simplifie

$$n_0 = \left\{\Phi^{-1}\left(\frac{p+1}{2}\right)\right\}^2 \frac{q(1-q)}{\epsilon^2} \tag{4.11}$$

Donc en fixant les valeurs de p, ϵ et q, l'utilisateur peut avoir une bonne estimation de la durée minimale de la simulation $T_{min} = n_0 k$. Cette valeur est une estimation adéquate de la durée de la simulation préliminaire.

4.2.2 Test de Geweke

Ce test a pour but de comparer les caractéristiques des candidats générés par la chaîne au début de la simulation avec ceux générés à partir d'une certaine itération pour savoir si la chaîne à atteint sa distribution limite. Pour ce faire, deux séquences de points sont créées, soit $\{\theta_i^1 : i = 1, 2, ..., t_1\}$ et $\{\theta_i^1 : i = t_2, ..., t\}$ où $\{1 < t_1 < t_2 < t\}$, puis les moyennes de chacune de ces séquences sont calculées,

$$\bar{\theta}^1 = \frac{1}{t_1}\sum_{i=1}^{t_1}\theta_i \quad \text{et} \quad \bar{\theta}^2 = \frac{1}{t_3}\sum_{i=t_2}^{t}\theta_i \quad \text{où } t_3 = t - t_2 + 1.$$

Si $\frac{t_1+t_3}{t} < 1$, que les rapports $\frac{t_1}{t}$ et $\frac{t_3}{t}$ sont fixes et que la chaîne est en état stationnaire, alors la statistique,

$$Z_t = \frac{\bar{\theta}_1 - \bar{\theta}_2}{\sqrt{\frac{\hat{s}_1(0)}{t_1} + \frac{\hat{s}_2(0)}{t_3}}}, \tag{4.12}$$

est distribuée selon une loi normale centrée réduite lorsque $n \to \infty$. Dans la définition précédente, $\hat{s}_1(0)$ et $\hat{s}_2(0)$, sont les estimations de la fréquence 0 de la densité spectrale de chacune des séquences.

La statistique Z_t permet de tester l'hypothèse d'égalité des moyennes des deux séquences. Cette hypothèse est rejetée lorsque $|Z_t|$ est plus grand qu'une valeur arbitraire. Puisqu'une chaîne ayant atteint l'état stationnaire produit des séquences de réalisations ayant sensiblement la même moyenne, le rejet de cette hypothèse peut indiquer que la période de transition n'est pas terminée et que la simulation doit être continuée.

4.2.3 Diagnostic de Heidelberger et Welch

Soit une séquence de réalisations de la chaîne de Markov à étudier $\{\theta_t\}$, la somme des éléments de cette séquence $S_T = \sum_{t=1}^{T} \theta_t$ et la moyenne des valeurs de la séquence $\bar{\theta} = (\frac{1}{T}) \sum_{t=1}^{T} \theta_t$. En posant $S_0 = 0$, nous construisons la suite,

$$B_t(x) = \frac{\left(S_{[Tx]} - [Tx]\bar{\theta}\right)}{(T\hat{p}(0))^{\frac{1}{2}}} \quad x \in \left\{ \frac{1}{T}, \frac{2}{T}, ..., \frac{T-1}{T}, 1 \right\}, \tag{4.13}$$

où $[Tx]$ signifie la partie entière par excès de Tx et $\hat{p}(0)$ est une estimation de la densité spectrale évaluée à la fréquence 0. Puisque B_T converge vers un pont Brownien lorsque $n \to \infty$, il est possible de baser un test sur la borne de Cramer-Von Mises

$$\int_0^1 B_T^2(x)dx, \tag{4.14}$$

vérifiant la convergence de la chaîne. La procédure proposée par Heidelberg consiste à utiliser successivement ce test sur des sous-ensembles de réalisations de plus en plus petit dans le but de trouver le plus grand sous-ensemble pouvant exclure la période de transition. En particulier, la procédure consiste d'abord à choisir un facteur de précision α, à utiliser l'ensemble de la séquence $\{\theta_t\}$ pour calculer la statistique $B_T(x)$, puis à estimer $\int_0^1 B_T^2(x)$. Si cette dernière valeur est plus petite que la valeur paginée d'une statistique de Cramer-Von Mise avec une marge d'erreur $1 - \alpha$, nous concluons que cet échantillon est distribué selon la fonction cible $f(\theta)$; dans le cas contraire, nous retranchons 10% de l'échantillon, jusqu'à un maximum cumulatif de 50% et recommençons le processus. Lorsque 50% de l'échantillon a été retranché, nous concluons que l'échantillon est tiré d'un processus n'ayant pas encore atteint son régime stationnaire et nous continuons la simulation. Lorsque le nombre théorique d'itérations nécessaire pour la période de transition N_{min} a été calculées au préalable, le nombre d'itérations supplémentaires à effectuer avant de refaire le test est généralement admis comme étant égale à N_{min}, sinon le nombre d'éléments formant la séquence est doublée.

Les procédures de Geweke et de Heidelberger-Welch sont construites pour être utilisées parallèlement à la simulation de la chaîne. Pour minimiser la durée de la simulation, il peut être tentant d'organiser la simulation uniquement en fonction de l'utilisation de ces tests. Or, puisque chaque utilisation de ce test comporte un risque d'erreur de type 2, l'utilisation répétée de ces tests entraîne une augmentation de la probabilité d'accepter l'hypothèse nulle, alors qu'elle est fausse. Il est donc préférable de baser les durées de la simulation sur les résultats des simulations préliminaires et d'utiliser les tests uniquement comme outils de diagnostics.

4.2.4 Diagnostic de Gelman et Rubin

Gelman et Rubin ont développé une méthode permettant de déterminer numériquement si des chaînes évoluant en parallèle ont atteint leur régime stationnaire. En notant θ_{ij} la i^{me} observation

de la j^{me} simulation et en supposant qu'il y a m simulations de longueur n, nous définissons les statistiques,

$$G = \frac{n}{m-1}\sum_{i=1}^{m}(\hat{\theta}_{i.} - \hat{\theta}_{..})^2 \quad \text{et} \quad S = \frac{1}{m}\sum_{i=1}^{m} s^2, \tag{4.15}$$

où $\hat{\theta}_{i.} = \frac{1}{n}\sum_{j=1}^{n}\theta_{ij}$ et $s_i^2 = \frac{1}{n-1}\sum_{j=1}^{n}(\theta_{ij} - \hat{\theta}_{i.})^2$. La satistique G est une estimation de la variance globale entre les simulations et S est une estimation de la variance dans chacune des simulations. S a la particularité de sous-estimer la valeur de $\phi^2(\theta)$, car pour toute séquence de longueur finie n, la chaîne n'ayant pas parcourru tout son support, la variabilité de l'échantillon est nécéssairement plus petite que celle de la distribution cible. Il est aussi possible d'utiliser ces deux statistiques pour estimer la variance marginale a posteriori $\sigma^2(\theta \mid Y)$ à l'aide de la statistique,

$$\hat{\sigma}^2(\theta) = \frac{n-1}{n}S + \frac{m+1}{nm}G. \tag{4.16}$$

Cette statistique a pour particularité de surestimer la valeur de $\sigma^2(\theta)$, lorsque n est fini et que l'hypothèse que les points de départs soient bien répartis sur le support de la fonction f est vérifiée. De plus, $\hat{\sigma}^2(\theta)$ est non biaisé sous l'hypothèse que la chaîne ait atteint la stationnarité. Par la construction de la chaîne, cela est nécessairement vrai lorsque n est assez grand.

Si $n \to \infty$, les statistiques $\hat{\sigma}^2(\theta)$ et S converge vers $\sigma^2(\theta)$ chacun selon leur propre direction. Il est donc possible de définir la statistique

$$\sqrt{\hat{R}} = \sqrt{\frac{\hat{\sigma}^2(\theta \mid Y)}{S}}, \tag{4.17}$$

appellé "l'estimation de la réduction d'échelle potentielle" correspondant à la racine carrée du rapport de la borne supérieure sur la borne inférieure de la variance des θ. Lorsque les simulations sont toutes en régime permanent, il est impossible de les distinguer et $\hat{R} = 1$. Au contraire, si les simulations ne sont pas en régime stationnaire et $\hat{R} = a, a \in \mathbb{R}$, alors il est potentiellement possible de réduire l'intervalle de confiance de $\sigma^2(\theta)$ d'un facteur de $\sqrt{a}$ en retardant l'échantillonnage.

En 1997 par Brooks et Gelman [1] ont utilisé un estimateur plus raffiné de la variance marginale a posteriori

$$\tilde{\phi}^2(\theta) = \left(\frac{n-1}{n}\right)^2 \frac{1}{m}\hat{\phi^2}(s_i^2) + \left(\frac{m+1}{nm}\right)^2 \frac{2}{m-1}S^2 + 2\frac{(m+1)(n-1)}{n^2 m}\frac{n}{m}\left(\hat{cov}(s_i^2, (\hat{\theta}_{i.})^2) - 2\hat{\theta}\hat{cov}(s_i^2, (\hat{\theta}_{i.}))\right) \tag{4.18}$$

pour créer une version alternative de l'estimation de la réduction d'échelle potentielle $\hat{R}$. En posant $\hat{d} = \frac{2\hat{\phi^2}(\theta)^2}{\hat{\phi^2}\hat{\phi^2}\theta}$, ce nouvel estimateur est défini comme suit,

$$\hat{R}_c = \sqrt{\frac{\hat{d}+3}{\hat{d}+1}\frac{\hat{\phi^2}\theta}{S}} = \sqrt{\frac{\hat{d}+3}{\hat{d}+1}\left(\frac{n-1}{n} + \frac{m+1}{nm}\frac{G}{S}\right)}. \tag{4.19}$$

1. Brooks, S. P. and Gelman, A. (1997), "General Methods for Monitoring Convergence of Iterative Simulations," Journal of Computational and Graphical Statistics, 7, 434–455.

Cet estimateur a le même comportement que $\hat{R}$, mais permet une meilleure estimation des gains en précision pouvant être obtenu en prolongeant la période de transition. L'utilisation de cette statistique, et par extension de cette démarche, demande beaucoup de puissance de calcul. De plus, lors de l'utilisation de ce diagnostic, trois chaînes sont simulées en parallèle (une servant exclusivement à l'inférence et les deux autres sont utilisés pour faire le diagnostique sur la convergence) ce qui alourdit d'autant plus les calculs.

4.3 Diagnostic de la qualité de la simulation

4.3.1 Diagnostique d'autocorrélation

Pour un certain décalage h, tel que $0 \leq h < T$, l'autocovariance d'une séquence de réalisations d'une chaîne de Markov est estimée par,

$$\hat{\gamma}(h) = \frac{1}{T-h} \sum_{i=1}^{T-h} \left(\theta_{i+h} - \bar{\theta}\right)\left(\theta_i - \bar{\theta}\right). \tag{4.20}$$

En conséquence, l'autocorrélation de $\{\theta_t\}$ avec un décalage $0 \leq h < T$ est estimé par

$$\hat{\rho}(h) = \frac{\hat{\gamma}(h)}{\hat{\gamma}(0)}. \tag{4.21}$$

4.3.2 Temps d'autocorrélation

Le temps d'autocorrélation τ, est une mesure du taux de convergence de la moyenne d'une fonction définie par une chaîne de Markov stationnaire, de variance bornée et géométriquement ergodique. Plus précisément, pour une telle chaîne de Markov $\{\theta_t\}_{t=1}^{T}$, tel que $E\{\theta_t\} = \mu$ et $\phi^2(\theta_i) = \sigma^2$ et soit $\bar{\theta}_T$ la moyenne empirique d'une suite de réalisations $(\theta_1, ..., \theta_T)_{t=1}^{T}$ de $\{\theta_t\}_{t=1}^{T}$, alors le temps d'autocorrélation correspond à la valeur de τ tel que,

$$\sqrt{\frac{T}{\tau}} \frac{\bar{\theta}_T - \mu}{\sigma} \rightarrow N(0,1). \tag{4.22}$$

Il existe plusieurs méthodes générales d'approximation de τ comme le calcul par lots de la moyenne où la régression linéaire basé sur la fréquence $f(0)$ du spectre logarithmique de la chaîne, mais puisque l'autocorrélation $\rho(h)$ est généralement estimée lors d'une simulation MCMC, dans le but de faire des diagnostiques graphiques sur le taux de mixage de la chaîne, cette valeur est estimée par la méthode des séquences initiales (*Initial sequence estimators*). Cette méthode exploite l'identité utilisée dans la définition de taille d'échantillon réel, présenté dans la prochaine section, et consiste à fixer un seuil α, à identifier le plus petit I tel que $\rho_I(\theta) < \alpha$ et à calculer l'approximation,

$$\tau = 1 + 2\sum_{i-1}^{I} \rho_i^2(\theta). \tag{4.23}$$

Un seuil généralement utilisé est $\alpha = 1\%$. Un autre est $\alpha = 2s_i$ où s_i correspond à l'estimation de l'écart-type,

$$s_i = 2\sqrt{\left(\frac{1}{T}\left(1 + 2\sum_{i-1}^{I-1} \rho_i^2(\theta)\right)\right)}. \tag{4.24}$$

4.3.2.1 Taille d'échantillon réel

Si l'échantillon est grandement autocorrélé, la quantité d'information contenue dans l'échantillon est moins grande que si les observations sont indépendantes. Cette observation est à la base du concept de *taille d'échantillon réel* correspondant à l'estimation du nombre maximal d'observations indépendantes d'un échantillon. La statistique utilisée pour estimer cette quantité est nommée *Effective sample size(ESS)* en anglais et correspond au ratio de la taille totale T de l'échantillon et de τ le temps d'autocorrélation,

$$ESS = \frac{T}{\tau} = \frac{T}{1 + 2\sum_{i-1}^{\infty} \rho_i^2(\theta)}. \tag{4.25}$$

Plus la valeur de cette statistique est élevée, plus le mixage est de bonne qualité. De plus. puisque l'ESS est inversement proportionnelle au temps d'autocorrélation, seul le calcul de τ est nécessaire pour juger de la qualité du mixage de la chaîne. Donc, il est possible de conclure que la chaîne de Markov a atteint son état stationnaire si la valeur de τ est basse.

La statistique ESS peut être utilisée pour estimer a posteriori le nombre d'itérations à retrancher pour obtenir un échantillon distribué selon la distribution stationnaire de la chaîne simulée. Pour ce faire, il suffit de calculer l'ESS des sous-ensembles d'observations obtenu en retranchant successivement les premières $n = 1, 2, ..., T$ observations de l'échantillon. En comparant les valeurs ainsi obtenues, il est possible de déterminer la valeur de n maximisant la taille réelle de l'échantillon. Cette valeur est alors utilisée comme durée de la période de transition.

4.3.3 Test de stabilité

Ce test consiste à fixer une constante arbitraire α et de vérifier si cette constante borne le rapport de l'erreur d'estimation de la moyenne sur la valeur de l'estimation de la moyenne,

$$\frac{\sigma_\mu}{\hat{\mu}} \leq \alpha. \tag{4.26}$$

Lorsque cette relation est vérifiée, cela indique que le nombre d'observations est suffisant pour estimer la moyenne de la distribution. Puisque ce test vérifie l'erreur relative, la valeur de la borne α doit être ajustée pour compenser la norme de la moyenne ; une petite moyenne doit être testé avec une grande borne et inversement.

4.3.4 Diagnostique de Raftery et Lewis

Cette statistique utilise les résultats des calculs faits à l'aide des simulations préliminaires,

$$I = \frac{n_0 + T}{T_{min}}, \tag{4.27}$$

où n_0 est la longueur, en itération, de la période de transition, T est la longueur de la simulation et T_{min} est la longueur de la simulation d'une fonction indépendante tel que calculée avec l'équation 4.11. Cette statistique mesure le niveau de dépendance entre les éléments générés durant la simulation. En effet, plus les éléments d'une séquence sont corrélés plus la chaîne parcours le support de la distribution cible lentement et plus le nombre d'itérations $n_0 + T$ sera élevé. Bien qu'un rapport I près de 1 soit souhaité, il est communément admis dans la littérature qu'un rapport $I > 1$ est une preuve de la présence de dépendance importante dans l'échantillon seulement lorsque $I > 5$.

4.4 Échantillonnage parfais

De par la construction des méthodes MCMC, les valeurs générées par ces algorithmes sont distribuées selon une chaîne de Markovs possédant la propriété de se comporter sensiblement comme la fonction cible $f(\theta)$ lorsqu'elle est itérée un grand nombre de fois. En conséquence, l'échantillon généré par ces méthodes est toujours une approximation d'un échantillon distribué selon la densité d'intérêt $f(\theta)$. Ce dernier chapitre montre qu'il existe des techniques permettant de déterminer si cette approximation est de qualité, mais elles sont souvent subjectives et demandent beaucoup de ressources de calcul. La présence de ce facteur d'erreur est le principal défaut de cette classe de méthodes, autrement intuitives, versatiles et faciles d'implémentation. Les méthodes de couplage par le passé, présentées en détail à l'annexe A.3.3 permettent de contourner ce problème.

Bien que leurs bases théoriques soient solides, une utilisation naïve des méthodes de Monte-Carlo par chaîne de Markovs peut facilement générer des résultats erronés. Les techniques qui furent présentées dans ce chapitre sont d'une importance capitale pour le bon déroulement des simulations et l'obtention d'estimation de qualité. Or, les simulations préliminaires, le monitorage et les différents tests à effectuer sont des tâches supplémentaires venant alourdir des méthodes demandant elles même beaucoup de temps de calcul. Pour ce qui est du couplage par le passé, cette technique est relativement nouvelle ; la majorité des recherches faite ce sur sujet se concentrent sur l'énumération des situations où cette technique peut aboutir en un temps fini et le développement de variation permettant des modélisations moins contraignantes. Pour l'instant, ces méthodes sont trop lourdes à implémenter et donnent des résultats dont la qualité varie trop selon les situations pour qu'elles soient utilisées plus souvent. Plus particulièrement, cette lenteur rend les MCMC peu aptes à simuler des situations réelles dont les caractéristiques évoluent en temps réels. Dans un tel contexte, un autre type de méthodes Monte-Carlo doit être utilisée, soit les méthodes de Monte-Carlo séquentielles.

Chapitre 5

Méthodes de Monte-Carlo séquentielles

5.1 Introduction

Les méthodes étudiées jusqu'à présent ont été développées, partiellement avec l'objectif de minimiser la quantité d'information nécessaire à l'application des méthodes de Monte-Carlo. Par exemple, la méthode de simulation par inversion nécessite une connaissance totale de la fonction $f(\theta)$ à simuler, alors que les méthodes *MCMC* nécessite la connaissance de $f(x)$ qu'à une constante près. Cette évolution a rendu les méthodes de Monte-Carlo beaucoup plus souples et attrayantes. Or, en pratique, il est commun d'obtenir progressivement de l'information supplémentaire sur le système à modéliser et les techniques permettant la polyvalence de ces méthodes les rendent peu aptes à intégrer cette information. Pour s'en convaincre, il suffit de considérer l'utilisation des méthodes Monte-Carlos par chaînes de Markovs pour estimer la distribution d'une variable aléatoire X à partir de la série de réalisations $\{x_{1:t}\}$. Dans un contexte Bayesien, cela revient à estimer la distribution a posteriori $p(x|x_1, ..., x_t)$ en échantillonnant le produit de la vraisemblance $p(x_1, ..., x_t)$ et du prior $p(x)$ à l'aide d'une méthode *MCMC*. Lorsqu'une nouvelle réalisation x_{t+1} est observée, la fonction cible devient $p(x|x_1, ..., x_t, x_{t+1}) \propto p(x_1, ..., x_t, x_{t+1})p(x)$, fonction qui n'est pas la fonction cible de la méthode *MCMC* utilisée. Donc, chaque nouvelle observation apportant de l'information significative oblige une nouvelle implémentation de l'algorithme. De plus, cet exemple montre que la dimensionnalité de la fonction cible augmente avec t, donc rapidement le temps de simulation exigé par ces méthodes devient trop important pour qu'elles soient utiles, surtout lorsque l'information est transmise en temps réel.

Ce chapitre présente plusieurs méthodes utilisant les propriétés des fonctions récursives pour contrer ces limitations. La section 5.2 présente le modèle d'espace états et traduit les objectifs des méthodes de Monte-Carlo dans un tel contexte. Ensuite, la section 5.3.1 introduit une classe de techniques donnant une approximation optimale dans ce type de situations, appelé les filtres de Kalmans. Finalement, les sections 5.4.2 et 5.5 présentent respectivement les méthodes de Monte-Carlo séquentielles, ainsi que des méthodes complémentaires permettant de les améliorer.

5.2 Méthodologie en contexte récursif

Le modèle d'espace état est composé par
- Un espace d'état Θ.
- Une distribution de l'état initial $p(x_0)$.
- Une fonction de transition des états $p(x_t \mid x_{t-1})$.
- Une distribution des observations $p(y_t \mid x_t)$.

Spécifiquement, les états du processus à estimer sont déterminés par un processus de Markov X_t ayant un espace d'état Θ, dont le premier état θ_1 est distribuée selon une densité $p(\theta_0)$ et dont les

transitions respectent,

$$p(X_t \mid X_{t-1} = x_{t-1}) = f(x_{t-1}, \epsilon_x), \tag{5.1}$$

où ϵ_x est un résidu de distribution quelconque. De plus, il est supposé que les réalisations de X_t sont perceptibles que par l'entremise d'observations bruités y_t, effectuée durant un interval de temps $t \in \{1, 2, ..., T\}$. Par analogie avec X, ces observations sont suposées elles-mêmes générées par un processus de Markov Y_t respectant $p(Y_t \mid X_t = x_t) = h(x_t, \epsilon_y)$.

Dans les pages qui suivent, pour simplifier l'écriture, il est aussi supposé que la fonction de transition $h(y_t \mid x_t)$ est homogène, c'est-à-dire qu'elle ne varie pas selon t et que le temps varie de façon discrète. La généralisation des résultats présentés dans ce document aux situations où $h(y_t \mid x_t)$ est non-homogène ou que le paramètre t varie de façon continue est souvent immédiate ; dans le cas contraire, des références vers de la documentation détaillant les méthodes de généralisations seront données en bas de page.

5.2.1 Filtrage

Les algorithmes de filtrages permettent l'approximation des séquences de distributions $p(X_{1:T})$ conditionnellement aux observations $\{y_{1:T}\}$ pour $t = 1, 2, ..., T$. Lorsque les distributions f et g sont linéaires, que ϵ_x et ϵ_y sont additifs et distribués selon une loi normale, les filtres de Kalman permettent une estimation optimale de la distribution $p(X_{1:T})$. Dès que ces hypothèses ne sont pas rencontrées, il est préférable d'utiliser les méthodes d'inférence Bayesienne avec $p(x_{1:T}|Y_{1:T})$ comme distribution a posteriori, la distribution des $x_{1:T}$, $p(x_{1:T})$, comme distribution a priori et $p(Y_{1:T}|x_{1:T})$ comme vraisemblance. En utilisant ces notations, il est possible de redéfinir la distribution a priori du processus à étudier X_t, soit,

$$p(x_{1:T}) = h(x_1)\Pi_{t=2}^{T} f(x_t \mid x_{t-1}), \tag{5.2}$$

ainsi que la vraisemblance bayésienne,

$$p(y_{1:T} \mid x_{1:T}) = \Pi_{t=1}^{T} h(y_t \mid x_t). \tag{5.3}$$

Selon la formule de Bayes, la distribution de l'ensemble des réalisations $x_{1:T}$ conditionnées sur celle des observations $y_{1:T}$ s'écrit

$$p(x_{1:T} \mid y_{1:T}) = \frac{p(x_{1:T})p(y_{1:T} \mid x_{1:T})}{\int p(x_{1:T})p(y_{1:T} \mid x_{1:T})dx_{1:T}}, \tag{5.4}$$

où $p(x_{1:T}, y_{1:T})$ correspond à la fonction a posteriori non normalisée utilisée dans le reste du document. Dans des situations de non-linéarité, de non-normalité ou de haute dimensionnalité, l'intégrale au dénominateur de l'équation 5.4 est difficile voir impossible à calculer analytiquement. Pour évaluer ces quantités, les méthodes Monte-Carlo sont généralement utilisées. Il est à noter que pour alléger le document, la notation suivante est utilisée,

$$p(x_{1:T}, y_{1:T}) = p(x_{1:T})p(y_{1:T} \mid x_{1:T}) \quad \text{et} \quad p(y_{1:T}) = \int p(x_{1:T}, y_{1:T})dx_{1:T}. \tag{5.5}$$

De plus, il est parfois préférable d'indexer la distribution a postériori selon un paramètre ϕ. Si ϕ est connu, alors la distribution a posteriori $p_\phi(x_{1:T} \mid y_{1:T})$ est proportionnelle à la distribution,

$$p_\phi(x_{1:T}, y_{1:T}) = g_\phi \Pi_{t=2}^{T} f_\phi(x_t \mid x_{t-1}) \Pi_{t=1}^{T} h_\phi(y_t \mid x_t). \tag{5.6}$$

Dans le cas contraire, cette distribution est proportionnelle à la distribution bivariée,

$$p(\phi, x_{1:T} \mid y_{1:T}) \propto p_\phi(x_{1:T}, y_{1:T}) p(\phi), \tag{5.7}$$

qui est alors utilisée comme distribution a posteriori dans l'inférence Bayesienne. Puisque l'ajout de l'indexation aux résultats généraux est direct, par mesure d'économie, l'indexation est omise des formulations utilisées dans ce document.

En résumé, le but est de déterminer la distribution des réalisations du processus X_t au fur et à mesure que nous disposons de nouvelles informations liées aux observations Y_t, donc au temps t, nous estimons $p(x_{1:t} \mid y_{1:t})$ et $p(y_{1:t})$. À partir de cette approximation, il est relativement simple de faire une estimation des moments de $p(x_{1:t})$ par échantillonnage de $p(x_{1:t})$ et l'utilisation de la moyenne empirique.

5.2.2 Estimation Bayesienne récursive

Cette méthode n'impose aucune restriction sur les distributions utilisées dans l'estimation de $p(x_{1:T})$ et peut donc être considéré comme la méthode récursive la plus générale. Comme son nom l'indique, cette méthode est basée sur le théorème de Bayes et les relations des fonctions récursives. Dans un tel contexte, la fonction a posteriori non normalisée satisfait,

$$\begin{aligned}
p(x_{1:t}, y_{1:t}) &= p(x_{1:t}) p(y_{1:t} \mid x_{1:t}) \\
&= g(x_1) \Pi_{i=2}^{t} f(x_i \mid x_{i-1}) \Pi_{i=1}^{t} h(y_i \mid x_i) \\
&= g(x_1) \Pi_{i=2}^{t-1} f(x_i \mid x_{i-1}) \Pi_{i=1}^{t-1} h(y_i \mid x_i) (f(x_t \mid x_{t-1}) h(y_t \mid x_t)) \\
&= p(x_{1:t-1}) p(y_{1:t-1} \mid x_{1:t-1}) (f(x_t \mid x_{t-1}) h(y_t \mid x_t)) \\
&= p(x_{1:t-1}, y_{1:t-1}) (f(x_t \mid x_{t-1}) h(y_t \mid x_t)).
\end{aligned} \tag{5.8}$$

La fonction a posteriori peut s'écrire sous la forme récursive suivante.

$$\begin{aligned}
p(x_{1:t} \mid y_{1:t}) &= \frac{p(x_{1:t}, y_{1:t})}{\int p(x_{1:t}, y_{1:t})) dx_{1:t}} \\
&= \frac{p(x_{1:t-1}, y_{1:t-1}) (f(x_t \mid x_{t-1}) h(y_t \mid x_t))}{\int p(x_{1:t-1}, y_{1:t-1}) (f(x_t \mid x_{t-1}) h(y_t \mid x_t)) dx_{t-1:t}} \\
&= \frac{p(x_{1:t-1}, y_{1:t-1}) (f(x_t \mid x_{t-1}) h(y_t \mid x_t))}{p(y_{1:t} \mid y_{1:t-1})}.
\end{aligned} \tag{5.9}$$

Cette équation nous permet d'atteindre le premier objectif, soit d'incorporer l'information provenant des Y_T à l'estimation de la distribution des réalisations du processus X_t. De plus, cette forme récursive offre des avantages considérables par rapport à la définition originale ; d'abord, puisque seule une mise à jour des données est nécessaire à chaque itération, elle nécessite beaucoup moins de calculs à effectuer que les méthodes vues précédemment. Ensuite puisque toute l'information sur les itérations précédentes est résumée dans le résultat d'une itération, la quantité d'espace mémoire

à utiliser est de beaucoup réduite. De plus, en intégrant la fonction 5.9 par rapport à $x_{1:t-1}$ permet d'obtenir la distribution marginale,

$$p(x_t \mid y_{1:t}) = \frac{h(y_t \mid x_t)p(x_t \mid y_{1:t-1})}{p(y_t \mid y_{1:t-1})}, \tag{5.10}$$

où

$$p(x_t \mid y_{1:t-1}) = \int (f(x_t \mid x_{t-1})p(x_{t-1} \mid y_{1:t-1}))dx_{t-1}. \tag{5.11}$$

Ces deux dernières équations peuvent être utilisées comme solution de rechange à l'estimation de $p(x_{1:t} \mid y_{1:t})$ dans cet algorithme. Finalement, la vraisemblance marginale $p(y_{1:t})$ peut être évaluée à chaque étape en utilisant les estimations vues précédemment et s'écrit,

$$p(y_{1:t}) = p(y_1)\Pi_{i=2}^{t} p(y_i \mid y_{1:i-1}). \tag{5.12}$$

À l'aide de ce cadre théorique générale, il est possible de développer des méthodes particulières adaptées à modéliser des situations spécifiques. Les méthodes présentées dans ce document peuvent être classées selon les hypothèses qui les soutiennent. Le filtre de Kalman, le filtre de Kalman étendu, le filtre de Kalman *unscented*, le filtre de Kalman *unscented* augmenté sont utilisés lorsque les résidus du processus et des observations sont distribués selon une loi normale additive. L'échantillonnage préférentiel séquentiel, le filtre particulaire, le filtre particulaire auxiliaire et le filtre particulaire *unscented* sont des méthodes de Monte-Carlo reposant sur des hypothèses moins restrictives.

5.3 Filtre de Kalman

Le filtre de Kalman a été initialement développé pour modéliser des processus ayant un espace des états linéaire et de cardinal fini. Sous ces hypothèses, le système à modéliser s'écrit,

$$\begin{aligned} x_t &= A_{t-1}x_{t-1} + q_{t-1}, \\ y_t &= H_{t-1}x_t + r_t, \end{aligned} \tag{5.13}$$

où $q_{t-1} \sim \mathcal{N}(0, Q_{t-1})$ est la valeur d'un bruit blanc à l'itération $t-1$, $r_t \sim \mathcal{N}(0, Q_{t-1})$ est la valeur du résidu des observations à l'itération t, A_{t-1} est la matrice de transition du modèle à estimer et H_{t-1} est la matrice de transition des observations. En d'autres termes,

$$\begin{aligned} p(x_t \mid x_{t-1}) &= \mathcal{N}(x_t \mid A_{t-1}x_{t-1}, Q_{t-1}), \\ p(y_t \mid x_t) &= \mathcal{N}(y_t \mid H_t, x_t, R_t). \end{aligned} \tag{5.14}$$

De plus, par convension, l'état initial x_0 est supposé distribuée selon une loi normale de paramètre arbitraire, idéalement correspondant aux paramètres généraux du système. De plus, le filtre de Kalman utilise les notations suivantes,

1. $\tilde{x}_t$ et $\tilde{P}_t$ sont respectivement la prédiction de la moyenne et de la matrice de covariance des états au temps t, avant observation.
2. x_t et P_t sont respectivement l'estimation de la moyenne et de la matrice de covariance des états au temps t, après observation.

3. v_t est la mesure du résidu au temps t.
4. S_t est la covariance de la prédiction au temp t.
5. K_t est le gain de Kalman.

Le filtre de Kalman se fait en deux temps. D'abord, l'étape de prédiction utilise les données observées à l'itération $t-1$ pour faire une estimation de l'état à l'itération t. Ensuite, l'étape de mise a jour utilise le gain de Kalman pour ajuster la prédiction aux observations. En pseudocode cette technique s'écrit,

Algorithme 5.3.1 Filtre de Kalman

Entrées: L'ensemble des observations $y_{1:t}$.
Entrées: $\hat{x}_0$ et P_0.
Initialisation de l'algorithme
Poser $x_0 \sim \mathcal{N}(\hat{x}_0, P_0)$.
$\tilde{x}_1 = A_0\tilde{x}_0$.
$\tilde{P}_1 = A_0 P_0 A_0^T + Q_0$.

Pour chaque observation au temps t,

Étape de mise à jour
$v_t = y_t - H_t\tilde{x}_t$.
$S_t = H_t\tilde{P}_t H_t^T + R_t$.
$K_t = \tilde{P}_t H_t^T S_t^{-1}$.
$x_t = \tilde{x}_t + K_t v_t$.
$P_t = \tilde{P}_t - K_t S_t K_t^T$.

Étape de prévision
$\tilde{x}_t = A_{t-1}\tilde{x}_{t-1}$.
$\tilde{P}_t = A_{t-1}P_{t-1}A_{t-1}^T + Q_{t-1}$.

Puisque l'étape de prévision ne dépend pas directement des observations, il est possible de prévoir la valeur de $\hat{x}_{t+n}$ pour une valeur de n arbitraire, mais puisque la précision de ces prévisions dépend de l'estimation des moyennes succesives $\hat{x}_t$, la qualité de la prévision diminue rapidement proportionnellement à n. De plus, lorsque les lois de transitions sont continues, il est tout de même possible d'utiliser ce filtre en appliquant les transformations appropriées [1].

5.3.1 Filtre de Kalman étendu

L'hypothèse de linéarité sous-tendant le filtre de Kalman est très restreignante en pratique. La classe des filtres de Kalman étendus évite cette limitation en utilisant les séries de Taylor pour faire une approximation de la distribution $p(x_t, y_t)$. Ces filtres sont classés selon le degré de l'approximation de Taylor utilisé ; le filtre de Kalman étendu de premier degré utilise une approximation linéaire ; le filtre de Kalman étendu de second degré utilise une approximation quadratique ;etc. Cette section présente le filtre de Kalman étendu de premier degré. Les processus pouvant être

1. Pour tous les détails, voir Optimal State Estimation - Kalman, H_∞, and Nonlinear Approaches, page 231-260

modélisés par ce filtre sont de la forme,

$$x_t = f(x_{t-1}, t-1) + q_{t-1}, \quad y_t = h(x_t, t) + r_t, \tag{5.15}$$

où les fonctions f et h ne sont pas nécessairement linéaires et les résidus sont des bruits blancs de distributions $q_{t-1} \sim \mathcal{N}(0, Q_{t-1})$ et $r_{t-1} \sim \mathcal{N}(0, R_t)$. En notant $G_x(\hat{x})$, le jacobien de la fonction non linéaire $p(y_t) = h(x_t)$ et en définissant $\mu_L = h(\hat{x})$, $S_L = G_x(\hat{x})PG_x^T(\hat{x})$ et $C_L = PG_x^T(\hat{x})$, l'approximation Gaussienne s'écrit,

$$\begin{pmatrix} x \\ y \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \hat{x} \\ \mu_L \end{pmatrix}, \begin{pmatrix} P & C_L \\ C_L^T & S_L \end{pmatrix}\right).$$

Ce résultat permet d'estimer la fonction $p(x_t \mid y_{1:t})$ par la loi normale $\mathcal{N}(x_t \mid \hat{x}_t, P_t)$ et en notant $F_x(\hat{x}, t-1)$ et $H_x(\hat{x}, t)$ respectivement les matrices Jacobiennes des fonctions f et h, les étapes de cette estimation sont résumées par l'algorithme 5.3.2.

Algorithme 5.3.2 Filtre de Kalman étendu

Entrées: $\hat{x}_0$ et P_0.

Initialisation de l'algorithme

Poser $x_0 \sim \mathcal{N}(\hat{x}_0, P_0)$.

$\tilde{x}_1 = f(\tilde{x}_0, 0)$.

$\tilde{P}_1 = F_x(\tilde{x}_0, 0)P_0F_x(\tilde{x}_0, 0)^T + Q_0$.

Pour chaque observation au temps t,

Étape de mise à jour

$v_t = y_t - h_t(\tilde{x}_t, t)$.

$S_t = H_t(\tilde{x}_t, t)\tilde{P}_tH_t(\tilde{x}_t, t)^T + R_t$.

$K_t = \tilde{P}_tH_t(\tilde{x}_t, t)^TS_t^{-1}$.

$x_t = \tilde{x}_t + K_tv_t$.

$P_t = \tilde{P}_t - K_tS_tK_t^T$.

Étape de prévision

$\tilde{x}_t = f(\tilde{x}_{t-1}, t-1)$.

$\tilde{P}_{t-1} = F_x(\tilde{x}_{t-1}, t-1)P_{t-1}F_x(\tilde{x}_{t-1}, t-1)^T + Q_{t-1}$

.

5.3.2 Filtre de Kalman *unscented*

L'utilisation des séries de Taylor dans le filtre de Kalman permet d'obtenir de bonnes estimations de la distribution a posteriori, mais cette méthode nécessite le calcule de la matrice Jacobienne, calcul qui est souvent difficile ou fastidieux. La version *unscented* de ce filtre utilise plutôt la transformation *unscented* pour effectuer l'approximation de la distribution jointe de x et y par une loi normale de la forme,

$$\begin{pmatrix} x \\ y \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \hat{x} \\ \mu_L \end{pmatrix}, \begin{pmatrix} P & C_U \\ C_U^T & S_U \end{pmatrix}\right).$$

La transformation *unscented* consiste à déterminer un ensemble de points, appelé points sigmas, partageant les caractéristiques de la distribution initiale de x, puis d'appliquer la distribution non linéaire g sur les points sigmas. L'ensemble résultant est un échantillon de points permettant d'estimer les moments de la distribution non linéaire.

Cette méthode consiste d'abord à fixer trois paramètres, α, β et γ, paramètre déterminant entre autres le paramètre d'échelle $\lambda = \alpha^2(n+\kappa) - n$. Ensuite, les $2n+1$ points sigma sont calculés à l'aide des colonnes de la matrice $\sqrt{(n+\lambda)P}$, selon la formule

$$\begin{aligned} x^{(0)} &= \hat{x}, \\ x^{(i)} &= \hat{x} + [\sqrt{(n+\lambda)P}]_i, \quad i = 1, 2, ..., n, \\ x^{(i)} &= \hat{x} - [\sqrt{(n+\lambda)P}]_i, \quad i = n+1, n+2, ..., 2n. \end{aligned} \tag{5.16}$$

La séquence des observations est obtenue en définissant chaque élément de la matrice Y de la manière suivante $Y^{(i)} = g(x^{(i)})$. Ces données permettent d'estimer la moyenne de Y grâce à la moyenne pondérée

$$\hat{Y}_U \approx \sum_{i=0}^{2n} w_{\hat{x}}^{(i)} Y^{(i)}, \tag{5.17}$$

où les poids sont définies par,

$$\begin{aligned} w_{\hat{x}}^{(0)} &= \lambda/(n+\lambda), \\ w_{\hat{x}}^{(i)} &= \lambda/(2(n+\lambda)) \quad i = 1, ..., 2n. \end{aligned} \tag{5.18}$$

ainsi que la covariance de Y,

$$S_U \approx \sum_{i=0}^{2n} w_c^{(i)} (Y^{(i)} - \hat{Y}_U)(Y^{(i)} - \hat{Y}_U)^T, \tag{5.19}$$

en utilisant les poids,

$$\begin{aligned} w_c^{(0)} &= \lambda/(n+\lambda) + (1 - \alpha^2 + \beta), \\ w_c^{(i)} &= \lambda/(2(n+\lambda)) \quad i = 1, ..., 2n. \end{aligned} \tag{5.20}$$

Finalement, la covariance de x et y est estimée par l'expression,

$$C_U \approx \sum_{i=0}^{2n} w_c^{(i)} (x^{(i)} - \hat{x})(x^{(i)} - \hat{x})^T, \tag{5.21}$$

où $(x^{(i)} - \hat{x})^T$ est la transposée du vecteur $(x^{(i)} - \hat{x})$. Ce processus peut être traduit sous forme matricielle, en posant,

$$W = (I - [w_{\hat{x}} ... w_{\hat{x}}]) diag(w_c^{(0)}, ..., w_c^{(2n)}) (I - [w_{\hat{x}} ... w_{\hat{x}}])^T, \tag{5.22}$$

où I est la matrice identité de dimension $2n+1 \times 2n+1$, $w_{\hat{x}}$ est la matrice colonne ayant le poids $W_{\hat{x}}^{(i)}$ comme élément à la ligne i et $diag(w_c^{(0)}, ..., w_c^{(2n)})$ est une matrice diagonale ayant $w^{(i)}$ comme

élément (i, i). La transformation *unscented* sous forme matricielle se résume à,

$$\begin{aligned} X &= [\hat{x} \quad ... \quad \hat{x}] + \sqrt{(n+\lambda)}[0 \ \ldots \ \sqrt{P} \ \ldots \ -\sqrt{P}], \\ Y &= g(X), \\ \hat{Y} &= Y w_{\hat{x}}, \\ S_U &= YWY^T, \\ C_U &= XWY^T. \end{aligned} \tag{5.23}$$

L'algorithme 5.6 du filtre de Kalman *unscented* utilise cette notation.

Algorithme 5.3.3 Filtre de Kalman *unscented*

H

Entrées: $\hat{x}_0$ et P_0.

Initialisation de l'algorithme

Poser $x_0 \sim \mathcal{N}(\hat{x}_0, P_0)$.

$\tilde{x}_1 = f(\tilde{x}_0, 0)$.

$\tilde{P}_1 = F_x(\tilde{x}_0, 0) P_0 F_x(\tilde{x}_0, 0)^T + Q_0$.

Pour chaque observation au temps t

Étape de mise à jour

$X_t = [\hat{x}_t \quad ... \quad \hat{x}_t] + \sqrt{(n+\lambda)}[0 \ \ldots \ \sqrt{P} \ \ldots \ -\sqrt{P}]$.

$\tilde{Y}_t = g(\tilde{X}_t, t)$.

$\mu_t = \tilde{Y}_t w_{\hat{x}}$.

$S_t = \tilde{Y}_t W \tilde{Y}_t^T + R_t$.

$C_t = \tilde{X}_t W \tilde{Y}_t^T$.

Étape de prévision

$X_{t+1} = [\hat{x}_t \quad ... \quad \hat{x}_t] + \sqrt{(n+\lambda)}[0 \ \ldots \ \sqrt{P} \ \ldots \ -\sqrt{P}]$.

$x_{t+1} = f(X_{t+11}, t)$.

$\tilde{\hat{X}}_t = x_{t+1} w_t$.

$\tilde{P}_{t+1} = x_{t+1} W x_{t+1}^T + Q_t$.

5.3.3 Filtre de Kalman *Unscented* augmenté

Cette version du filtre de Kalman utilise une version modifiées de l'espace état où chaque état x_t est défini par la concaténation de l'état X_t et des valeurs des résidus du processus et des observations ϵ_x et ϵ_y, donc $x_t = [X_t^T \quad \epsilon_x^T \quad \epsilon_y^T]$. L'utilisation de ces états augmentés offre essentiellement deux avantages ; premièrement, puisque les résidus sont des paramètres indépendants cela permet de modéliser des relations où le bruit n'est pas additif, ce qui était impossible dans les méthodes précédentes ; deuxièmement, puisque le bruit n'est pas simplement additionné aux états, l'effet de la non-linéarité est beaucoup plus marqué ce qui peut améliorer l'estimation [2]. En notant la moyenne

2. Dans leur article Unscented Kalman Filtering for Additive Noise Case : Augmented vs. Non-augmented, Yuanxin Wu et ses collègues font la démonstration que la principale différence entre le filtre de Kalman *unscented* et sa version augmentée est que la version augmentée ne nécessite pas l'échantillonnage de nouveaux points sigmas lors de l'étape de mise à jour et que les gains en informations sont seulement sur les moments d'ordre impair.

des états X_t, $\tilde{x}_{t-1} = [\hat{x}_{t-1}^T \quad 0 \quad 0]^T$, la matrice des covariances des états augmentés,

$$\bar{P}_{t-1} = \begin{pmatrix} P_{t-1} & 0 & 0 \\ 0 & Q_{t-1} & 0 \\ 0 & 0 & R_{t-1} \end{pmatrix}$$

et respectivement, la matrice des résidus du processus et des observations $\Sigma_t^{(x)}$ et $\Sigma_t^{(y)}$, l'algorithme 5.3.4 représente le pseudo-code du filtre de Kalman *unscented* augmenté.

Algorithme 5.3.4 Filtre de Kalman *unscented* augmenté

Entrées: $\hat{x}_0$ et P_0.

Initialisation de l'algorithme

Poser $x_0 \sim \mathcal{N}(\hat{x}_0, P_0)$.

$x_1 = f(\tilde{x}_0, 0)$.

$\tilde{X}_t = x_0 w_0$.

$\tilde{P}_1 = F_x(\tilde{x}_0, 0) P_0 F_x(\tilde{x}_0, 0)^T + Q_0$.

Pour chaque observation au temps t

Étape de mise à jour

$\tilde{Y}_t = g(x_t, \Sigma_{t-1}^{(y)}, t)$.

$\mu_t = \tilde{Y}_t w_{\hat{x}}$.

$S_t = \tilde{Y}_t W \tilde{Y}_t^T$.

$C_t = x_t W \tilde{Y}_t^T$.

Étape de prévision

$X_{t-1} = [\hat{x}_{t-1}, ..., \hat{x}_{t-1}] + \sqrt{c}[0\sqrt{P_{t-1}} - \sqrt{P_{t-1}}]$.

$x_t = f(X_{t-1}^x, \Sigma_{t-1}^{(x)}, t-1)$.

$\tilde{X}_t = x_t w_t$.

$\tilde{P}_t = x_t W x_t^T$.

$K_t = C_t S_t^{-1}$.

$\hat{x}_t = \tilde{x}_t + K_t[y_t - \mu_t]$.

$P_t = \tilde{P}_t - K_t S_t K_t^T$.

Il est à noter que d'autres versions du filtre de Kalman ont été développées, chacune utilisant un ensemble particulier d'hypothèses pour obtenir une estimation optimale de la distribution $p(X_t)$. Or ces hypothèses sont souvent restrictives et la performance de ces algorithmes diminue rapidement lorsqu'une de ces hypothèses n'est pas rencontrée. En conséquence, si l'information disponible à l'utilisateur préalablement à la simulation ne permet pas de vérifier avec certitude les hypothèses de ces méthodes, il est préférable d'utiliser les filtres particulaires.

5.4 Filtre particulaire

La classe des filtres particulaires ont pour objectif de faire une approximation discrète de la fonction $p(x_{1:T} \mid y_{1:T})$ qui servira par la suite à estimer les caractéristiques de cette dernière. Pour

ce faire, ces algorithmes génèrent n pairs de poids et d'états $\left\{w_t^{(i)}, x_t^{(i)}\right\}$, nommées particules, utilisées dans l'approximation de $p(x_{1:T} \mid y_{1:T})$ via la création de la fonction empirique pondérée,

$$p(x_{1:T} \mid y_{1:T}) \approx \sum_{i=1}^{N} w_t^i \delta_{x_t^{(i)}}, \tag{5.24}$$

où δ est la fonction delta de Dirac.

5.4.1 Échantillonnage préférentiel séquentiel

Une des premières méthodes de Monte-Carlo séquentielles développée est une transposition de la méthode d'échantillonnage préférentiel dans un contexte séquentiel. L'objectif de cette méthode est d'estimer la moyenne de $p(x_t)$ en utilisant la relation générale,

$$E(p(x_t) \mid y_t) = \frac{E(w_t p(x_t) \mid y_t)}{E(w_t \mid y_t)}, \tag{5.25}$$

où les poids sont le rapport de la distribution a posteriori et d'une fonction instrumentale arbitraire $w_t = p(x_t \mid y_t)/q(x_t \mid y_t)$. Similairement a l'échantillonnage préférentiel général, $E(p(x_t) \mid y_t)$ est estimé en calculant une moyenne pondérée sur un échantillon tiré de $q(x_t \mid y_t)$, donc,

$$E(p(x_t) \mid y_t) \propto \frac{\sum_{i=1}^{n} w_t^{(i)} p(x_t^{(i)})}{\sum_{i=1}^{n} w_t^{(i)}}. \tag{5.26}$$

De par la nature récursive des équations, il est possible de traduire les composantes de l'échantillonnage préférentiel en terme de mise à jour des données de l'itération précédente. La fonction instrumentale s'écrit en terme récursif,

$$\begin{aligned} q_t(x_{1:t}) &= q_{t-1}(x_{1:t-1}) q_t(x_t \mid x_{1:t-1}), \\ &= q_1(x_1) \Pi_{i=2}^{n} q_i(x_i \mid x_{1:i-1}). \end{aligned} \tag{5.27}$$

De même, les poids sont définis par l'expression,

$$\begin{aligned} w_t &= \frac{f_t(x_{1:t})}{g_t(x_{1:t})}, \\ &= \frac{f_{t-1}(x_{1:t-1})}{g_{t-1}(x_{1:t-1})} \frac{f_t(x_{1:t})}{f_{t-1}(x_{1:t-1}) g_t(x_t \mid x_{1:t-1})}, \end{aligned} \tag{5.28}$$

ou plus simplement, en définissant les poids préférentiels incrémentiels,

$$\alpha_t(x_{1:t}) = \frac{f_t(x_{1:t})}{f_{t-1}(x_{1:t-1}) g_t(x_t \mid x_{1:t-1})}, \tag{5.29}$$

les poids se réécrivent,

$$\begin{aligned} w_t &= w_{t-1}(x_{t-1}) \alpha_t(x_{1:t}), \\ &= w_1(x_1) \Pi_{i=2}^{t} \alpha_i(x_{1:i}). \end{aligned} \tag{5.30}$$

Algorithme 5.4.1 Échantillonnage pondéré séquentiel

Échantillonner $X_1^i \sim g(x_1)$.
Calculer les poids $w_1(X_1^i)$, ainsi que le poids pondéré W_1^i.
Poser $i = 2$.
pour $i \leq T$ **faire**
  Échantillonner $X_i^j \sim g_i(x_i \mid x_{1:i-1})$.
  Calculer les poids $w_i(X_{1:i}^j) = w_{i-1}(X_{1:i-1}^j)\alpha_i(X_{1:i}^j)$, ainsi que le poids pondéré W_i^j.
fin
Calculer l'estimateur $\hat{p}(x_{1:t} \mid y_{1:t}) = \sum_{i=1}^n W_t^i \delta_{x_t^n}$.
Calculer l'estimateur $\hat{p}(y_{1:n}) = \frac{1}{n}\sum_{i=1}^n w_n x_t^n$.

Ces redéfinitions permettent de traduire l'algorithme d'échantillonnage préférentiel en terme d'estimations séquentielles.

La performance de cet algorithme dépend de la qualité de l'approximation de $p(x_t \mid x_{1:t-1}, y_t)$ par la fonction instrumentale. Lorsque l'information disponible sur $p(x_t \mid x_{1:t-1}, y_t)$ est limité, une stratégie couramment utilisée est de choisir la fonction minimisant la variance des poids w_t, donc d'utiliser $p_t(x_t \mid x_{1:t-1}) = f(x_t \mid x_{1:t-1})$. Dans ce cas, les poids incrémentiels associés sont donnés par l'expression,

$$\alpha_t(x_{1:t}) = \frac{f_t(x_{1:t-1})}{f_{t-1}(x_{1:t-1})} = \frac{\int f_t(x_{1:t})dx_t}{f_{t-1}(x_{1:t-1})}, \tag{5.31}$$

et la variance des poids w_t est nulle. Lorsque la fonction instrumentale est mal choisie, la variance des poids augmente rapidement d'itération en itération. En pratique, cette augmentation de la variance a pour conséquence qu'après quelques itérations, seulement un petit nombre de particules ont un poids significatif, alors que les autres particules ont un poids nul. Ce phénomène appauvrit l'échantillon et eût entrainé des erreurs d'estimation. Ce problème de dégénérescence des particules est généralement évité en utilisant les méthodes de ré-échantillonnages.

5.4.2 Filtre particulaire

Le filtre particulaire, aussi appelé filtre bootstrap, est un cas particulier de l'échantillonnage préférentiel séquentiel utilisant une fonction instrumentale qui tient compte des observations $y_{1:T}$ et utilisant le ré-échantillonnage pour contrer la dégénérescence. Pour minimiser la variance des poids w_i l'utilisateur doit choisir une fonction instrumentale s'approchant de $p(x_t \mid x_{1:t-1})$, fonction définie par l'expression,

$$p(x_t \mid y_t, x_{t-1}) = \frac{h(y_t \mid x_t)f(x_t \mid x_{t-1})}{p(y_t \mid x_{t-1})} = \frac{h(y_t \mid x_t)f(x_t \mid x_{t-1})}{\alpha_t(x_{1:t})}, \tag{5.32}$$

où, $\alpha_t(x_{1:t}) = p(y_t \mid x_{t-1})$. Cette fonction est généralement difficile à évaluer et il est donc nécessaire d'utiliser des approximations. Puisque $p(x_t \mid x_{1:t-1})$ est supposé être un processus de Markov cette fonction est indépendante des états pris par x aux itérations précédents $t-1$. En conséquence, il est inutile d'utiliser une fonction instrumentale dépendante des états $t-2, t-3, ...$, il est préférable de construire une approximation tenant compte seulement des informations disponibles à l'étape t.

Donc $g(x_t \mid x_{1:t}) = q(x_t \mid y_t, x_{t-1})$ et

$$\alpha_t(x_{1:t}) = \alpha_t(x_{t-1:t}) = \frac{h(y_t \mid x_t) f(x_t \mid x_{t-1})}{g(x_t \mid y_t, x_{t-1})}. \tag{5.33}$$

Le filtre particulaire sous forme de pseudo-code est donné par l'algorithme 5.4.2[3].

Algorithme 5.4.2 Filtre particulaire

Échantillonner $X_1^i \sim g(x_1 \mid y_1)$.
Calculer les poids $w_1(X_1^i) = \frac{u(X_1^i) h(y_1 \mid X_1^i)}{g(X_1^i \mid y_1)}$.
Ré-échantillonner (W_1^i, X_1^i) pour obtenir N particules de poids égaux $(\frac{1}{N}, \bar{X}_1^i)$.
Poser $t = 2$.
pour $t \leq T$ **faire**
  Échantillonner $X_t^i \sim g(x_t \mid y_t, \bar{X}_{t-1}^i)$ et poser $X_{1:t}^i \leftarrow (\bar{X}_{1:t-1}^i, X_t^i)$.
  Calculer les poids $\alpha_t(X_{t-1:t}^i) = \frac{h(y_t \mid X_t^i) f(X_t^i \mid X_{t-1}^i)}{g(X_t^i \mid y_t, X_{t-1}^i)}$.
  Ré-échantillonner $(W_t^i, X_{1:t}^i)$ pour obtenir N particules de poids égaux $(\frac{1}{N}, \bar{X}_{1:t}^i)$.
fin

À chaque itération t, cet algorithme produit une approximation de la distribution a posteriori,

$$\hat{p}(x_{1:t} \mid y_{1:t}) = \sum_{i=1}^{n} W_t^i \delta_{X_{1:t}^i}(x_{1:n}). \tag{5.34}$$

et une approximation de la distribution de transition des observations,

$$\hat{p}(y_{1:t} \mid y_{1:t-1}) = \sum_{i=1}^{n} W_{t-1}^i \alpha_t(X_{t-1:t}^i). \tag{5.35}$$

De plus, bien que le ré-échantillonnage multinomial soit utilisé dans la présentation de cet algorithme, d'autres types de ré-échantillonnage peuvent être utilisés.

5.4.2.1 Exemple d'application

L'échantillonnage préférentiel séquentiel et le filtre particulaire ont été utilisés pour modéliser un modèle de Markov caché où le processus, se déroulant sur une période de temps $t = 100$, est une marche aléatoire et les observations sont bruitées par un bruit blanc. Dans les deux cas, la fonction instrumentale utilisée est une loi normale $\mathcal{N}(0, 2)$ et 50 particules ont été utilisées. Les résultats sont illustrés dans les figures 5.1 et 5.2, où la série à simuler est présentée sous forme d'un trait continu bleu et l'estimation par un trait pointillé noir. De plus, l'erreur quadratique des estimations est résumée dans le tableau 5.1.

3. Pour les preuves de la convergence de l'échantillonnage préférentiel séquentiel et du filtre particulaire, voir "Convergence of Sequential Monte Carlo Methods" par Dan Crisan et Arnaud Doucet.

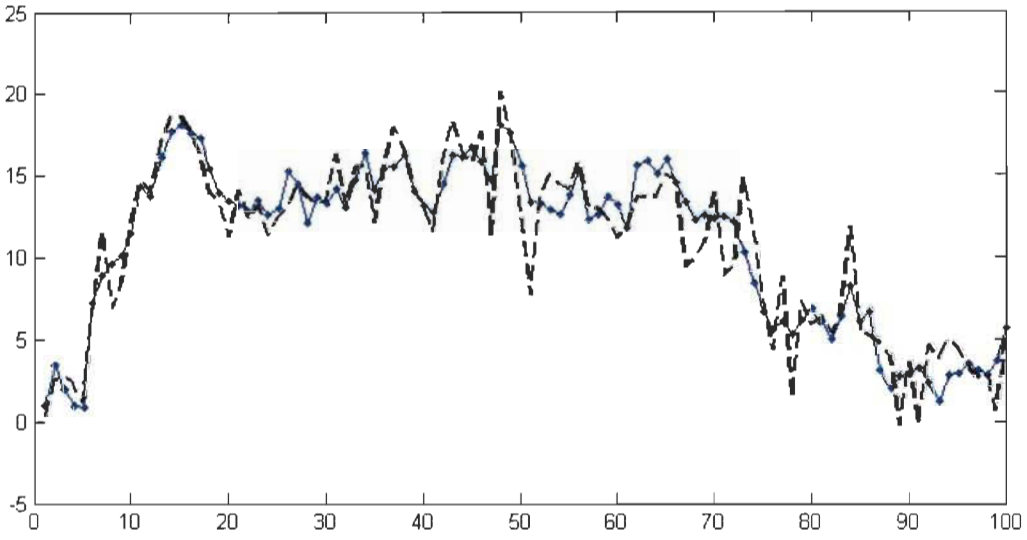


FIGURE 5.1 – Échantillonnage pondéré séquentiel

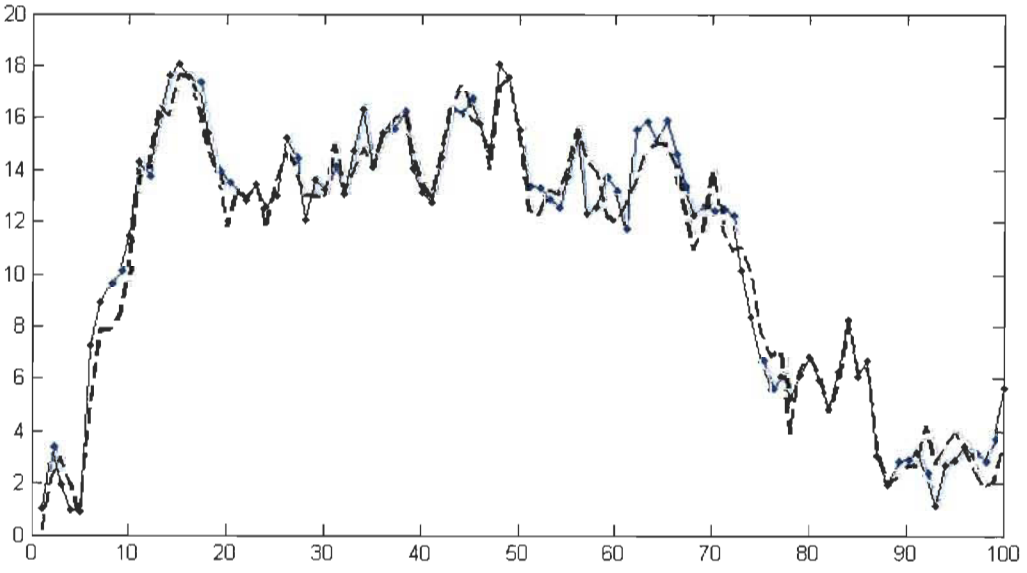


FIGURE 5.2 – Filtre particulaire

Ces données montrent que le filtre particulaire réagit plus rapidement aux rapides variations de la série que l'échantillonnage préférentiel séquentiel. Cela est une conséquence du plus grand nombre de poids non nul utilisés à chaque estimation.

TABLE 5.1 – Comparaison des estimations faites par FP et par EPS

	Erreur quadratique	Variance moyenne des poids
Échantillonnage pondéré séquentiel	1.04233	0.1283
Filtre particulaire	0.9045	0.4476

TABLE 5.2 – Nombre de poids plus grand que 0.01 selon l'itération t

Itération	1	10	50	75	100
Échantillonnage pondéré séquentiel	47	31	17	5	4
Filtre particulaire	49	49	50	39	49

5.4.3 Filtre particulaire auxiliaire

Le filtre particulaire auxiliaire est un algorithme qui utilise le ré-échantillonnage en début d'algorithme pour limiter les effets néfastes de l'utilisation du ré-échantillonnage, en particulier le bruitage des estimations. Pour atteindre cet objectif, chaque particule est indexée par une variable auxiliaire j servant de cible initiale du ré-échantillonnage. Dans ce contexte, les particules sont échantillonnées à partir de la distribution $p(x_{1:t}, j \mid y_{1:t})$, puis l'index j est ignoré pour que les particules soient distribuées selon la distribution cible.

L'algorithme consiste à calculer la moyenne de la densité $p(x_{1:t} \mid x_{1:t-1}^{(j)})$, puis à utiliser l'échantillonnage préférentiel séquentiel avec comme fonction instrumentale,

$$g(x_{1:t}, j \mid y_{1:t}) \propto p\left(y_{1:t} \mid \mu_t^{(j)}\right) p\left(x_{1:t} \mid x_{1:t-1}^{(j)}\right) w_{t-1}^{(j)}. \tag{5.36}$$

Ensuite, un ré-échantillonnage est appliqué sur les indices pour obtenir l'ensemble $\{j_i\}$ à l'aide des poids,

$$w_t(\mu_t^{(j)}) = \frac{p\left(y_{1:t} \mid \mu_t^{(j)}\right)}{\sum_{i=1}^{N} p\left(y_{1:t} \mid \mu_t^{(j)}\right)}. \tag{5.37}$$

Finalment les particules sont générées selon la distribution instrumentale $g(x_{1:t}, j \mid y_{1:t})$. Le pseudo-code de cette technique est donné par l'algorithme 5.4.3.

TABLE 5.3 – Comparaison des estimations faites par EPS, FP et FPA.

	Erreur quadratique	Variance moyenne des poids
Échantillonnage pondéré séquentiel	1.04233	0.1283
Filtre particulaire	0.9045	0.4476
Filtre particulaire auxiliaire	1.0780	0.0394

Algorithme 5.4.3 Algorithme de filtrage particulaire auxiliaire

Échantillonner $X_1^i \sim g(x_1 \mid y_1)$.
Calculer les poids $w_1(X_1^i) = \frac{u(X_1^i)h(y_1|X_1^i)}{g(X_1^i|y_1)}$.
Ré-échantillonner (W_1^i, X_1^i) pour obtenir N particules de poids égaux $(\frac{1}{N}, \bar{X_1^i})$.
Poser $t = 2$.
pour $t \leq T$ **faire**
  Poser $\left\{\mu_0^{(j)}\right\}_{j=1}^N$.
  Calculer les poids $w_t(\mu_0^j) = \frac{q(y_{1:t}|\mu_t^{(j)})}{\sum_{i=1}^N q(y_{1:t}|\mu_t^{(j)})}$.
  Ré-échantillonner $\{k\}_{k=1}^N$ pour obtenir $\{k_i\}_{i=1}^N$.
  pour $i = 1 : N$ **faire**
    Générer le candidats $x_t^{(i)} \sim g(x_t \mid x_{t-1}^{(k_i)})$.
  fin
fin

S'il est possible d'échantillonner les fonctions $p(x_{1:t} \mid y_{1:t}, x_{t-1})$ et $p(y_t \mid x_{t-1})$, alors en posant, $g_t(x_t \mid x_{1:t-1}) = p(x_{1:t} \mid y_{1:t}, x_{t-1})$ et $\tilde{p}(y_t \mid x_{t-1}) = p(y_t \mid x_{t-1})$ l'adaptation dite parfaite du filtre particulaire auxiliaire est obtenue. De plus, il est à noter que la qualité des estimations faites par le filtre particulaire auxiliaire est intimement liée à l'approximation $q(y_{1:t} \mid \mu_t^{(j)}) \sim p(y_{1:t} \mid x_t^{(j)})$. Lorsque l'approximation est médiocre, la qualité des estimations peut facilement être inférieure à celle obtenue par l'algorithme d'échantillonnage préférentiel séquentiel ou par le filtre particulaire. Donc lorsque l'information sur $p(y_{1:t} \mid x_t^{(j)})$ est limitée, il est préférable d'éviter d'utiliser cette méthode.

5.4.3.1 Exemple d'application

Le processus présenté à la section 5.4.2.1 a été simulé avec le filtre particulaire auxiliaire. Comme le montrent les résultats du tableau 5.3, cette méthode la précision des estimations faites par cette méthode se compare avantageusement avec celle des estimations faites par l'échantillonnage pondéré séquentiel et le filtre particulaire. De plus, la variance moyenne des poids utilisés lors de cette simulation est beaucoup moins importante. Comme le montre le tableau 5.4, le nombre de poids significatif reste élevé tout au long de la simulation. Cela, ainsi que la faible variance moyenne des poids, fait du filtre particulaire auxiliaire une méthode de choix pour obtenir des simulations fiables, surtout lors de longue simulation.

TABLE 5.4 – Nombre de poids plus grand que 0.01 selon l'itération t.

Itération	1	10	50	75	100
Échantillonnage pondéré séquentiel	47	31	17	5	4
Filtre particulaire	49	49	50	39	49
Filtre particulaire auxiliaire	49	50	50	50	50

La figure 5.3 présente un graphique de la simulation,où la série à simuler est présentée sous forme d'un trait continu bleu et l'estimation par un trait pointillé noir..

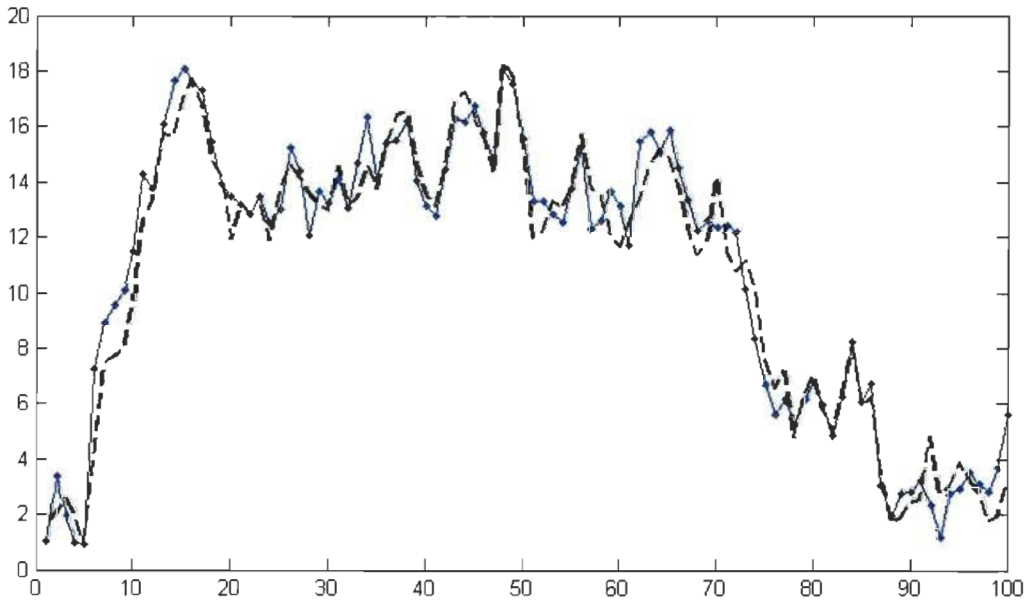


FIGURE 5.3 – Filtre particulaire auxiliaire.

5.4.4 Filtre particulaire *unscented*

La qualité des estimations faites par échantillonnage préférentiel séquentiel est grandement influencée par le choix de la distribution instrumentale. Puisque la distribution instrumentale minimisant la variance des poids est connue, mais impossible à utiliser, l'utilisation d'une approximation de celle-ci faite par le filtre de Kalman comme distribution instrumentale permet de faire diminuer considérablement l'erreur d'estimation. Cette constatation entraina la création des filtres particulaires hybride utilisant les diverses variantes du filtre de Kalman. Puisque les comportements asymptotiques de ces filtres sont sensiblement identiques, ces filtres hybrides ne se différencient qu'au niveau de la quantité de calculs requis et des performances des filtres de Kalman selon la forme de la fonction instrumentale à échantillonner. Par mesure d'économie, seul le filtre particulaire *unscented* est présenté dans ce document. Ce choix est basé sur le fait que le filtre de Kalman

unscented est versatile et propage la moyenne et la covariance des approximations d'une itération à l'autre beaucoup mieux que les autres techniques ce qui rend son utilisation particulièrement intéressante.

Le filtre particulaire *unscented* est équivalent à l'algorithme d'échantillonnage préférentiel séquentiel utilisant UKF pour générer des particules et se déroule en trois étapes principales. D'abord, l'algorithme est initialisé en échantillonnant le prior $p(x_0)$ pour obtenir n états initials $x_0^{(i)}$. À l'aide de ces états les quantités suivantes sont calculée :

$$\begin{aligned}
&\bar{x}_0^{(i)} = E(x_0^{(i)}),\\
&P_0^{(i)} = E\left[(x_0^{(i)} - \bar{x}_0^{((i))})(x_0^{(i)} - \bar{x}_0^{(i)})^T\right],\\
&\bar{x}_0^{(i)a} = E(x_0^{(i)a}) = [(\bar{x}_0^{(i)a})^T \quad 0 \quad 0],\\
&P_0^{(i)a} = E\left[(x_0^{(i)a} - \bar{x}_0^{(i)a})(x_0^{((i)a)} - \bar{x}_0^{(i)a})^T\right] = \begin{bmatrix} P_0^{(i)} & 0 & 0\\ 0 & Q & 0\\ 0 & 0 & R \end{bmatrix}.
\end{aligned} \tag{5.38}$$

Ensuite, chaque itération commence par mettre à jour les n particules à l'aide du filtre particulaire *unscented*. Donc, pour chaque particule $x_{t-1}^{(i)}$ la série de points sigmas est calculée,

$$X_{t|t-1}^{(i)a} = \left[x_{t-1}^{(i)a} x_{t-1}^{(i)a} \pm \sqrt{(n_a + \lambda)P_{t-1}^{(i)a}}\right], \tag{5.39}$$

avant d'être projetée dans le futur à l'aide des fonctions,

$$\begin{aligned}
X_{t|t-1}^{(i)x} &= f\left(X_{t-1}^{(i)x}, X_{t-1}^{(i)v}\right),\\
\bar{x}_{t|t-1}^{(i)} &= \sum_{j=0}^{2n_a} W_j^m X_{j|t-1}^{(i)x},\\
P_{t|t-1}^{(i)} &= \sum_{j=0}^{2n_a} W_j^c \left[X_{j|t-1}^{(i)x} - \bar{x}_{t|t-1}^{(i)}\right]\left[X_{j|t-1}^{(i)x} - \bar{x}_{t|t-1}^{(i)}\right]^T,\\
Y_{t|t-1}^{(i)x} &= h\left(X_{t-1}^{(i)x}, X_{t-1}^{(i)n}\right),\\
\bar{y}_{t|t-1}^{(i)} &= \sum_{j=0}^{2n_a} W_j^m Y_{j|t-1}^{(i)x}.
\end{aligned} \tag{5.40}$$

Les nouvelles observations sont utilisées pour mettre à jour les paramètres du système, soit

$$
\begin{aligned}
P_{\bar{y}_t|\bar{y}_t-1} &= \sum_{j=0}^{2n_a} W_j^c \left[Y_{j,t|t-1}^{(i)} - \bar{y}_{t|t-1}^{(i)}\right]\left[Y_{j,t|t-1}^{(i)} - \bar{y}_{t|t-1}^{(i)}\right]^T, \\
P_{x_t|y_t-1} &= \sum_{j=0}^{2n_a} W_j^c \left[X_{j,t|t-1}^{(i)} - \bar{x}_{t|t-1}^{(i)}\right]\left[X_{j,t|t-1}^{(i)} - \bar{x}_{t|t-1}^{(i)}\right]^T, \\
K_t &= P_{x_t|y_t-1}^T P_{\bar{y}_t|\bar{y}_t-1}^{-1}, \\
\hat{x}_t^{(i)} &= \hat{x}_{t|t-1}^{(i)} + K_t(y_t - y_{t|t-1}^{(i)}), \\
\hat{P}_t^{(i)} &= P_{t|t-1}^{(i)} - K_t P_{\bar{y}_t|\bar{y}_t-1} K_t^T.
\end{aligned} \tag{5.41}
$$

Ces paramètres permettent d'approximer la distribution a posteriori par la loi normale $\mathcal{N}(\hat{x}_t^{(i)}, \hat{P}_t^{(i)})$ et donc d'échantillonner le nouvel état $x_t^{(i)}$ et de calculer son poids $w_t^{(i)}$. Finalement, le ré-échantillonnage est effectué pour obtenir n particules $\left\{W_t^{(i)}, x_t^{(i)}\right\}$ de poids égaux $W_t^{(i)} = \frac{1}{n}$. Le filtre particulaire *unscented* sous forme de pseudocode est résumé par l'algorithme 5.4.4.

Algorithme 5.4.4 Filtre particulaire *unscented*

Initialise l'algorithme à l'aide des équations 5.38.
pour $1 < t < T$ **faire**
  pour $1 < i < n$ **faire**
    Calculer les points sigmas selon l'équation 5.39.
    Projeter les points sigmas dans le futur selon les équations 5.40.
    Incorporer les observations via la mise à jour des composantes de l'approximation Gaussienne 5.41.
    Échantillonner $\hat{x}_t^i \sim g(x_t \mid x_{0:t-1}^{(i)}, y_{1:t}) = \mathcal{N}(\hat{x}_t^{(i)}, \hat{P}_t^{(i)})$.
    Poser $\hat{x}_{0:t}^{(i)} = (x_{0:t-1}^{(i)}, x_t^{(i)})$ et $\hat{P}_{0:t}^{(i)} = (P_{0:t-1}^{(i)}, P_t^{(i)})$.
  fin
  Calculer les poids $w_t^{(i)} = \frac{p(y_t|\hat{x}_t^i)p(\hat{x}_t^i|\hat{x}_{t-1}^i)}{g(\hat{x}_t^i|x_{0:t-1}^i, y_{1:t})}$, puis les normaliser.
  Ré-échantillonner.
fin

L'utilisation du filtre de Kalman *unscented* dans l'échantillonnage préférentiel offre plusieurs avantages. D'abord, cet algorithme permet d'obtenir de bonnes approximations de la distribution instrumentale même si cette dernière n'est pas linéaire. De plus, ce filtre se distingue des autres variantes du filtre de Kalman par sa capacité à toujours englober le domaine de la distribution à simuler, propriété qui découle directement de l'utilisation de la transformation *unscented*. Finalement, l'ensemble des points sigmas est construit pour que ces derniers représentent certaines caractéristiques spécifiques de la distribution à simuler. En particulier, il est possible d'ajuster l'algorithme pour qu'il simule de manière optimale les distributions à queues importantes, distributions souvent utilisées comme fonction instrumentale de l'échantillonnage préférentiel séquentiel.

5.4.4.1 Exemple d'application

Le filtre particulaire *unscented* a été utilisé[4] pour estimer les états de 100 itérations d'un modèle de croissance univariée non stationnaire défini par les équations,

$$x_t = \alpha x_{t-1} + \frac{\beta x_{t-1}}{1 + x_{t-1}^2} + \gamma \cos((t-1)) + u_t \tag{5.42}$$

$$y_t = \frac{x_t^2}{20} + v_t \tag{5.43}$$

où u_t et v_t sont des bruits blancs. De plus, les valeurs $\alpha = 0.5$, $\beta = 25$ et $\gamma = 8$ sont supposées inconnues lors de l'implémentation du filtre. La figure 5.4 présente les résultats de l'estimation.

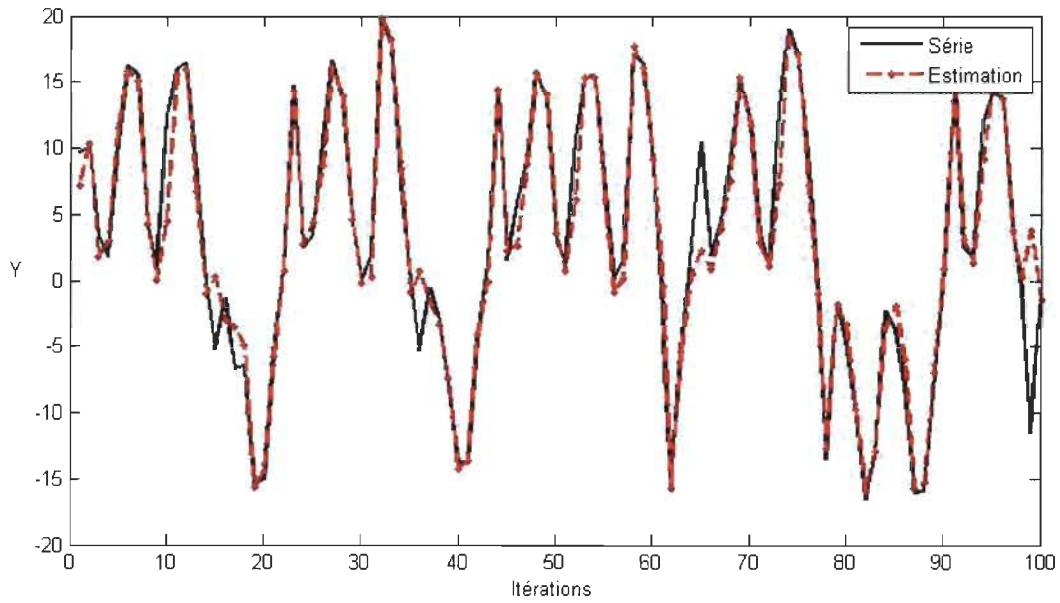


FIGURE 5.4 – Estimation faite avec le filtre particulaire *unscented*.

Le filtre particulaire, avec 50 particules, à été utilisé pour simuler la même série d'observations. La figure 5.5 présente les résultats de cette simulation et le tableau 5.5 présente l'erreur quadratique des deux simulations.

4. Pour la simulation, nous avons modifié le programme EKF/UKF toolbox for Matlab 7.x implenté par le département de bio-ingénérie et d'informatique de l'université Aalto, disponible à l'adresse http ://becs.aalto.fi/en/research/bayes/ekfukf/#

TABLE 5.5 – Comparaison des estimations faites par filtre particulaire et Filtre particulaire auxiliaire.

	Erreur quadratique
Filtre particulaire	26.0192
Filtre particulaire unscented	6.7074

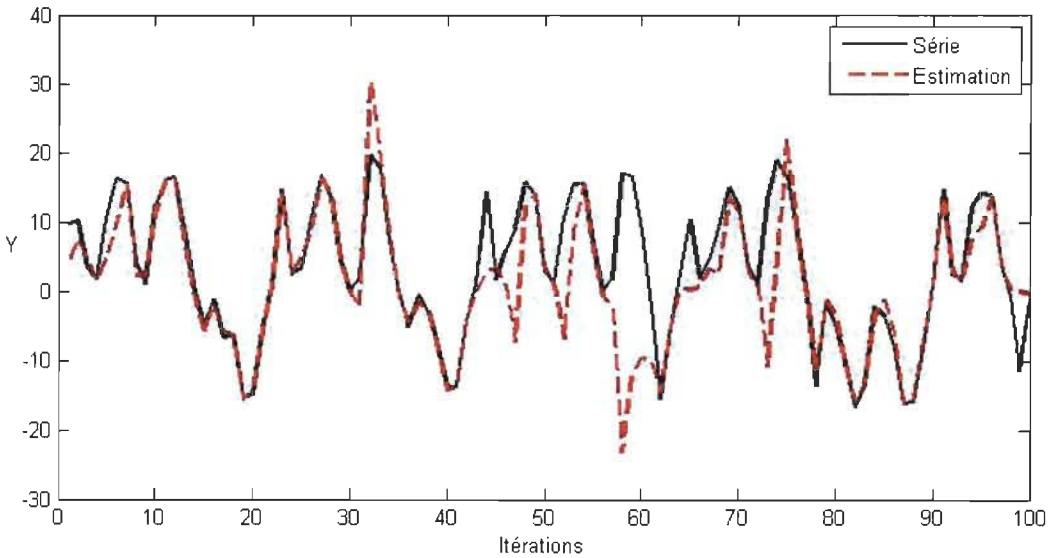


FIGURE 5.5 – Estimation faite avec le filtre particulaire.

Nous pouvons constater que les importantes variations de la série sont mieux simulées par le filtre particulaire *unscented* que le filtre particulaire. Il est à noter qu'en augmentant radicalement le nombre de particules utilisé par le filtre particulaire, ce dernier peut obtenir des résultats sensiblement identiques à ceux obtenus au filtre particulaire *unscented*.

TABLE 5.6 – Comparaison des estimations faites par EPS, FP, FPA et avec le filtre particulaire auxiliaire.

	Erreur quadratique	Variance moyenne des poids
Échantillonnage pondéré séquentiel	1.04233	0.1283
Filtre particulaire	0.9045	0.4476
Filtre particulaire auxiliaire	1.0780	0.0394
Filtre particulaire unscented	0.9252	0.3359

TABLE 5.7 – Nombre de poids plus grand que 0.01 selon l'itération t.

Itération	1	10	50	75	100
Échantillonnage pondéré séquentiel	47	31	17	5	4
Filtre particulaire	49	49	50	39	49
Filtre particulaire auxiliaire	49	50	50	50	50
Filtre particulaire unscented	50	48	49	42	49

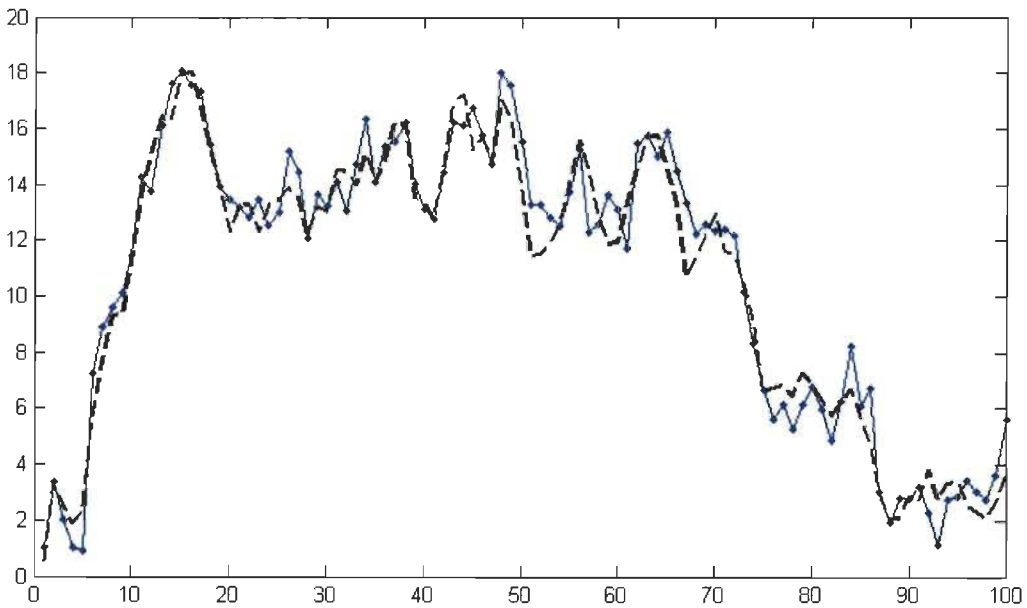


FIGURE 5.6 – Filtre particulaire *unscented*.

Dans le cas où le modèle à étudier est linéaire, les performances de ces deux algorithmes sont sensiblement identiques. Par exemple, la figure 5.6 montre le résultat de la simulation de la situation présenté à la section 5.4.2.1. Dans ce cas, les gains de performance obtenue par l'implémentation du filtre particulaire *unscented*, plutôt que le filtre particulaire, sont marginaux.

5.5 Méthodes contrant l'appauvrissement des échantillons

Le problème de la dégénérescence des échantillons n'est pas la seule problématique pouvant rendre les méthodes de Monte-Carlo séquentielles inutilisables. En effet, puisqu'à chaque itération, seule la variable $x_t \sim X_t^i$ conditionnelle aux observations $y_{1:t}$ est simulée puis est ajoutée à $\{X_{1:t-1}^i\}$ les particules générées ont un co-domaine beaucoup plus restreint qu'en simulant les parcours entiers $\{X_{1:t-1}^i\}$ et ultimement collapse après plusiseurs itérations. De plus, l'utilisation du ré-échantillonnage entraîne qu'à chaque itération seules les particules ayant un poids important sont utilisées pour l'approximation de l'itération suivante. Cette sélection des particules a pour conséquence qu'un petit nombre de particules peuvent avoir une surreprésentation lors de la génération des particules à l'itération suivante. Les méthodes présentées dans ce chapitre offrent des solutions au problème d'appauvrissement des échantillons causé par ces deux phénomènes.

5.5.1 Filtres particulaires adaptatifs

Lors d'une simulation par échantillonnage préférentiel séquentiel avec ré-échantillonnage, il est fréquent qu'à une itération donnée la variance des poids W_t^i soit minime. Dans ce cas, le coût de l'utilisation du ré-échantillonnage dépasse grandement ses bénéfices. Il est donc avantageux d'implanter des méthodes utilisant le ré-échantillonnage avec parcimonie.

Le qualificatif d'adaptatif est apposé à tout filtre particulaire fixant un seuil arbitraire sur la variance des poids non normalisés à partir duquel le ré-échantillonnage est appliqué sur les particules. Plusieurs méthodes peuvent être utilisées pour juger de la variance des poids non normalisés, mais une statistique particulièrement performante et simple à calculer est la taille d'échantillon réelle qui dans ce contexte est définie par

$$ESS = \frac{1}{\sum_{i=1}^{n} (W_t^n)^2}. \tag{5.44}$$

Cette statistique prend des valeurs $ESS = 1, 2, ..., n$ et estime le nombre de particules offrant réellement de l'information sur la distribution à estimer. En conséquence, une valeur basse de ESS reflète un appauvrissement de l'échantillon de particules et suggère la nécessité d'utiliser le ré-échantillonnage. Dans ce cas, le filtre particulaire adaptatif ré-échantillonnera lors que l'ESS est plus bas que le seuil arbitraire fixé par l'utilisateur, seuil communément fixé à $s = n/2$.

5.5.1.1 Exemple d'application

Le filtre utilisé pour la simulation utilisait 50 particules et le rééchantillonnage était effectué lorsque que la taille d'échantillon réel, ESS, était plus petite que 25. Comme le montre la figure 5.7 la simulation du processus présenté à la section 5.4.2.1 faite avec le filtre particulaire adaptatif donne de bonnes estimations des états réels du processus.

TABLE 5.8 – Comparaison des estimations faites par EPS, FP, FPA, FPU et avec le FPRA.

	Erreur quadratique	Variance moyenne des poids
Échantillonnage pondéré séquentiel	1.04233	0.1283
Filtre particulaire	0.9045	0.4476
Filtre particulaire auxiliaire	1.0780	0.0394
Filtre particulaire unscented	0.9252	0.3359
FP avec ré-échantillonnage adaptatif	0.9575	0.0482

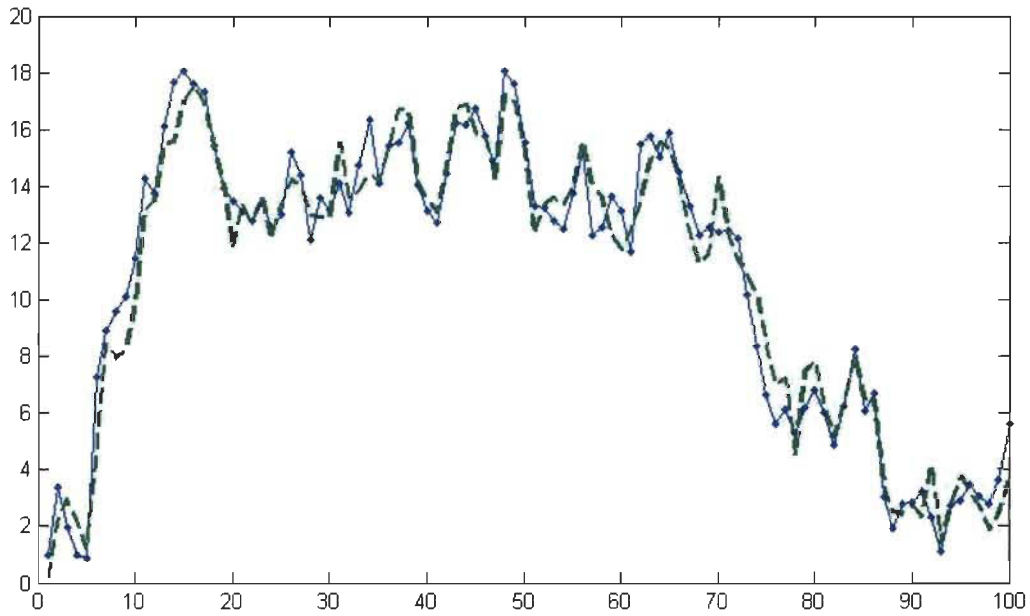


FIGURE 5.7 – Filtre particulaire adaptatif.

De plus, les données du tableau 5.8 montrent que le protocole d'utilisation du rééchantillonnage implanté dans cette méthode permet de faire diminuer considérablement la variance moyenne des poids utilisés par le filtre. Le tableau 5.9 montre comment le nombre de poids significatifs varie grandement d'une itération à l'autre selon l'utilisation ou non du rééchantillonnage.

TABLE 5.9 – Nombre de poids plus grand que 0.01 selon l'itération t.

Itération	1	10	50	75	100
Échantillonnage pondéré séquentiel	47	31	17	5	4
Filtre particulaire	49	49	50	39	49
Filtre particulaire auxiliaire	49	50	50	50	50
Filtre particulaire unscented	50	48	49	42	49
FP avec ré-échantillonnage adaptatif	49	27	26	33	25

5.5.2 Méthode de Monte-Carlo séquenciel avec MCMC

Une astuce couramment utilisée pour diversifier l'échantillon après l'utilisation du ré-échantillonnage. est de multiplier les particules $p(\hat{x}_{1:t} \mid y_{1:t})$ par le noyau d'une chaîne de Markov ayant $p(x_{1:t} \mid y_{1:t})$ comme distribution stationnaire. Le chapitre 3 a montré qu'il est relativement simple de construire une chaîne de Markov distribuée selon un noyau de transition $k(x \mid x^{'})$ ayant $p(x_{t-L:t} \mid y_{t-L:t})$ comme distribution stationnaire. De plus, lorsque c'est le cas,

$$\int k(x \mid x^{'}) p(\hat{x}_{1:t} \mid y_{1:t}) = p(x_{1:t} \mid y_{1:t}), \tag{5.45}$$

donc la particule résultante de cette étape supplémentaire est toujours distribuée selon la distribution cible. Les travaux académiques sur ce sujet [5] ont prouvés que la variance des estimations ne peut que diminuées avec l'ajout de cette étape, car le produit du noyau de la chaîne déplace les particules vers les zones de haute probabilité de la distribution.

La technique la plus souvent utilisée dans les algorithmes séquentiels est l'algorithme de Metropolis-Hasting avec une loi instrumentale $g(x_t \mid y_t, x_{t-1}, x_{1:t})$ et une probabilité d'acceptation

$$\beta = \min\left(1, \frac{h(y_k \mid x_k^{'}) f(x_{k+1} \mid x_k^{'}) f(x_k^{'} \mid x_{k-1}^{'}) g(x_k \mid y_k, x_{k-1}^{'}, x_k^{'}, x_{k+1})}{h(y_k \mid x_k) f(x_{k+1} \mid x_k) f(x_k \mid x_{k-1}^{'}) g(x_k^{'} \mid y_k, x_{k-1}^{'}, x_{k:k+1})}\right) \tag{5.46}$$

Cette technique bien qu'efficace, est à éviter s'il est possible d'échantillonner directement la loi conditionnelle complète $p(x_k^{'} \mid y_{1:t}, x_{1:t-1}^{'}, x_{k+1:t})$. Dans ce cas, il est préférable d'utiliser l'échantillonneur de Gibbs qui sera plus rapide. Dans un contexte séquentiel, cette technique consiste à poser $x_{1:t-L}^{'} \sim x_{1:t-L}$ puis à générer successivement $x_{1:t-L+k}^{'} \sim p(x_{t-L+k} \mid y_{1:k}, x_{1:t-L+k-1}^{'}, x_{t-L+k+1:t})$ pour $k < L$. Conséquement avec les résultats présentés au chapitre 3, les noyaux créés par ces méthodes sont invariants si la séquences $x_{1:t}$ est simulée en entier. Or pour répondre aux contraintes de temps imposées par les méthodes séquentiels. seul une partie des séries de durée $L < t$ peut être simulée.

Puisque le ré-échantillonnage uniformise l'ensemble des particules, l'étape correspondante à l'échantillonnage faite par *MCMC* est toujours placée à la fin d'une itération dans le but de maximiser son effet. Les poids utilisés pour le ré-échantillonnage étant calculés avant l'étape de *MCMC*, ils ne sont pas influencés par l'ajout de cette étape, donc, si l'étape *MCMC* est utilisé avec le filtre particulaire adaptatif, la fréquence d'utilisations du ré-échantillonnage reste sensiblement la même.

5. Pour plus de détails, voir Sequential MCMC for Bayesian Model Selection, par Christophe Andrieu, Nando de Freitas, Arnaud Doucet, IEEE Signal Processing Workshop on Higher Order Statistics. Ceasarea, Israel

TABLE 5.10 – Comparaison de la performances des méthodes MCS.

	Erreur quadratique	Variance moyenne des poids
Échantillonnage pondéré séquentiel	1.04233	0.1283
Filtre particulaire	0.9045	0.4476
Filtre particulaire auxiliaire	1.0780	0.0394
Filtre particulaire unscented	0.9252	0.3359
FP avec ré-échantillonnage adaptatif	0.9575	0.0482
Filtre particulaire avec MCMC	1.0733	0.5270

5.5.2.1 Exemple d'application

La figure 5.8 illustre une simulation du processus présenté à la section 5.4.2.1 faite par la méthode de Monte-Carlo séquentiel avec MCMC.

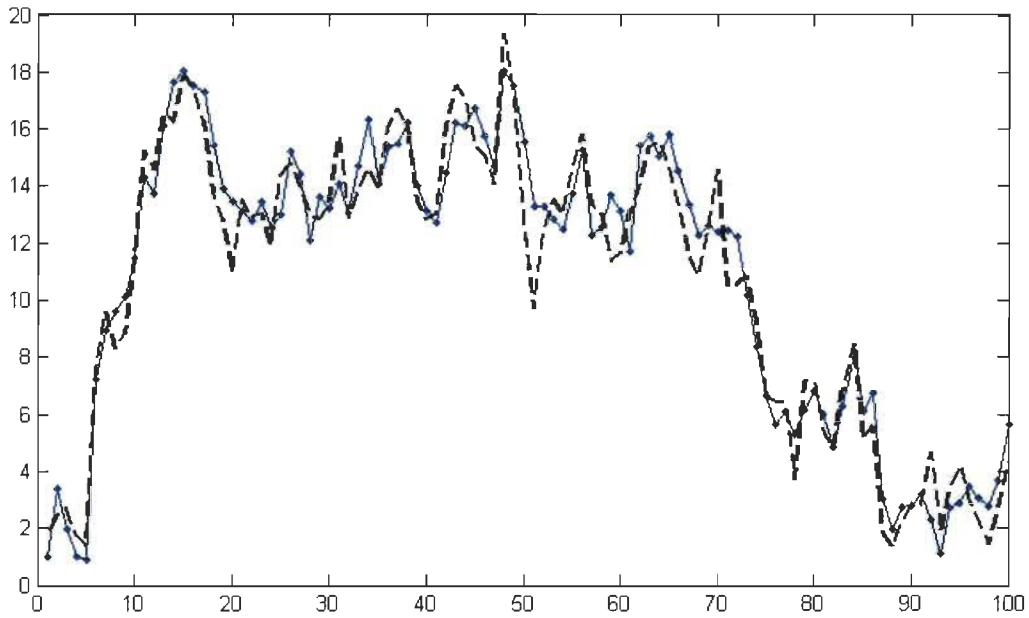


FIGURE 5.8 – Filtre particulaire avec MCMC.

Comme le montre les tableaux 5.10 et 5.11, les estimations faites par cette technique sont un peu plus précise que celles faites par les autres filtres présentés précédemment, mais, puisque le processus à simuler est linéaire, la différence est marginale. De plus, les résultats du tableau 5.11 suggèrent que l'addition de l'étape *MCMC* influence légèrement le nombre de particules ayant un poids significatif lors de l'estimation.

TABLE 5.11 – Nombre de poids plus grand que 0.01 selon l'itération t.

Itération	1	10	50	75	100
Échantillonnage pondéré séquentiel	47	31	17	5	4
Filtre particulaire	49	49	50	39	49
Filtre particulaire auxiliaire	49	50	50	50	50
Filtre particulaire unscented	50	48	49	42	49
FP avec ré-échantillonnage adaptatif	49	21	22	33	25
Filtre particulaire avec MCMC	49	37	42	50	41

Chapitre 6

Conclusion et discussion

Les méthodes de Monte-Carlo présentées dans ce document permettent de modéliser un grand nombre de situations pouvant être rencontrées par les chercheurs et les praticiens. Ce document se voulant une présentation des méthodes de Monte-Carlo les plus utiles, plusieurs techniques développées dans les dernières années pour simuler efficacement des processus plus exotiques ont dû être laissées de côté. Certaines méthodes de Monte-Carlo par chaînes de Markov particulièrement intéressantes sont,

- La méthode *Reverse-jump*. Cette méthode a comme particularité de permettre de simuler des distributions dont la dimension peut varier d'itération en itération.
- L'échantillonneur *Metropolis-Adjusted Langevin*, une méthode qui utilise le gradient ou la dérivée deuxième d'une distribution pour générer des échantillons.
- Méthode de Monte-Carlo hybride, une variante de l'algorithme de Métropolis-Hasting se servant de l'opérateur de Hamilton de la distribution cible pour calculer la probabilité d'acceptation d'un candidat.
- Algorithme de Monte-Carlo avec plusieurs essaies, un algorithme permettant de faire diminuer l'autocorrélation des échantillons obtenus lors de la simulation.
- L'échantillonnage par tranche, qui est un algorithme facilitant l'échantillonnage d'une distribution connue et qui est très performant lorsque la dimension de cette dernière est élevée.
- *Annealed Importance Sampling*. Cette méthode permet de mieux parcourir le support d'une distribution ayant des régions isolées.

De même, plusieurs méthodes de Monte-Carlo séquentielles très performantes ont dû être omise de ce document, par mesure d'économie. Parmi celles-ci, les plus intéressantes sont,

- Le filtre particulaire Rao–Blackwell qui utilise le théorème de Rao–Blackwell pour réduire la variance de l'estimation.
- Le filtre particulaire hiérarchique qui est particulièrement efficace pour simuler des modèles où les valeurs à estimer sont dépendantes.
- Échantillonnage séquentiel par bloc, un algorithme qui vise à simuler directement la suite des états $x_{t-s:t}$ pour $s \geq 1$ à chaque itération t.

Dans les dernières années, les recherches ayant comme sujet les méthodes de Monte-Carlo peuvent généralement être classées en trois types de recherches. D'abord, plusieurs articles parus dernièrement portaient sur l'étude d'applications de ces méthodes dans différents champs d'études. Cela s'explique par le fait que de plus en plus de chercheurs dans d'autres domaines que les mathématiques et l'informatique maîtrisent au moins un langage de programmation et que les milieux financiers ont adopté les techniques d'analyse statistique pour gérer le risque et faire des prévisions. Ces deux phénomènes font des méthodes de Monte-Carlo des techniques incontournables pour dans la plus part des domaines de recherches. Ensuite, plusieurs recherches portaient sur les conditions qui permettent d'accélérer la convergence des estimateurs ou de diminuer la variance des estimations. Ces recherches utilisent les propriétés des distributions, par exemple l'existence d'ordre partiel ou la présence de corrélation entre les différents paramètres d'un modèle hiérarchiques, à simuler

pour rendre ces méthodes plus efficaces. Le dernier type de recherche concerne les difficultés liées à l'inférence faite à partir des simulations Monte-Carlo.

Il est intéressant de constater que peu de recherche a été faite pour l'utilisation des copules dans l'échantillonnage par les méthodes de Monte-Carlo. L'incorporation des copules dans les méthodes Monte-Carlo offre potentiellement plusieurs avantages. Par exemple, l'efficacité de la plus part des méthodes de Monte-Carlo dépend du choix d'une fonction instrumentale. Puisque les copules permettent d'aisément simuler indépendamment les marges de la distribution et la distribution elle-même, leur utilisation dans les méthodes de Monte-Carlo pourrait permettre de choisir des fonctions instrumentales plus performantes en utilisant l'information disponible sur les marges. De plus, leurs implémentations facilitent les ajustements de l'algorithme suite à des simulations préliminaires. Finalement, dans certaines situations, les expressions des divers paramètres de la simulation, comme les probabilités d'acceptation et les poids, peuvent être simplifiées lorsque les distributions sont exprimées sous forme de copules. En conséquence, la vitesse d'exécution des algorithmes pourrait potentiellement être accélérée.

Appendices

Annexe A

Échantillonnage parfait

A.1 introduction

Cette section présente un ensemble de techniques permettant d'échantillonner directement la distribution stationnaire d'une chaîne de Markov. Cette technique appelée couplage par le passé, notée *CFTP* selon l'abréviation anglaise, est souvent difficile à implémenter, mais la possibilité de pouvoir générer un échantillon parfaitement distribué selon une distribution stationnaire donnée est trop avantageuse pour qu'elles soient ignorées.

Bien que, par mesure d'économie, ce chapitre s'attarde uniquement au cas où les chaînes évolueraient selon un temps discret, il est à noter que les prochains résultats peuvent être étendus aux cas où les chaînes évoluent selon un temps continu. De plus, puisque l'implémentation des algorithmes rencontre différentes difficultés selon le cardinal de Θ, l'espace des états de la chaîne de Markov à simuler, les versions du *CFTP* lorsque Θ est fini et lorsque Θ est infini dénombrable sont traités séparément.

Dans le cas où Θ est fini, l'étude du couplage par le passé consiste à observer k simulations de chaînes de Markovs effectuées en parallèle. Dans cette situation, l'état occupé par la chaîne n est noté $\theta_t^{(n)}$, où l'indice t commence à $t = 0$ et augmente au fil du temps. De plus, puisque ces méthodes sont présentées comme un complément des algorithmes MCMCs, pour chaque $i \in \{1, \ldots, k\}$ il est supposé que chaque chaîne de Markov $\mathbf{X}^{(i)} = \left(\mathbf{X}_0^{(i)}, \mathbf{X}_1^{(i)}, \ldots\right)$ est homogène et possède un espace d'état fini $\Theta_i = \{\theta_1^{(i)}, \ldots, \theta_k^{(i)}\}$. Il est à noter que le nombre de simulations en parallèle est égal au cardinal de Θ. Chaque chaîne est supposée posséder un état initial déterminé $\mathbf{X}_0^{(i)} = \theta_0^i$ et une matrice de transition notée $\mathbf{M}^i$. Évidemment, chaque matrice de transition $\mathbf{M}^i$ est supposée irréductible, apériodique et la distribution stationnaire de la chaîne $\mathbf{X}^{(i)}$ est notée π^i.

A.2 Couplage par le passé

Le couplage par le passé est basé sur les travaux de Propp et Wilson [1]. Ces derniers ont remarqué que l'état d'une hypothétique chaîne de Markov ayant un espace d'état Θ fini ayant évolué durant une période de temps infini jusqu'a un temps $t = 0$ est distribué selon la distribution limite π de la chaîne, à condition que cette dernière soit apériodique et irréductible. Puisque ce résultat est inutilisable en pratique, Propp et Wilson ont dû trouver une condition permettant d'utiliser ce résultat en un temps fini. Or si une telle chaîne de Markov évolue en fonction de nombres aléatoires observables et qu'il est possible de déterminer l'état au temps $t = 0$ en observant un nombre fini de ces nombres aléatoires, alors l'état au temps $t = 0$ est distribué selon π. Pour faciliter la

1. Propp, J. G. and Wilson, D. B. (1996), Exact sampling with coupled Markov chains and applications to statistical mechanics.

démonstration de cette préposition, il est nécessaire d'introduire les concepts d'innovation et de fonction de mise à jour.

Définition 3 (Innovation). *Soit $i \in \{1, \ldots, k\}$. Une innovation à l'étape t pour un état $\theta_i \in \Theta$ est une séquence de variables aléatoires $U_t^{(i)}$ i.i.d selon la loi uniforme de domaine $(0, 1]$, noté $\mathbf{U}^{(i)} = \left(U_1^{(i)}, U_2^{(i)}, \ldots\right)$. Deux cas nous intéressent : lorsque les séquences $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(k)}$ sont indépendantes et lorsqu'il existe une seule suite $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(k)}$. Dans ce dernier cas, nous définissons $\mathbf{U}^{(1)} = \ldots = \mathbf{U}^{(k)} = \mathbf{U}$.*

Définition 4 (Fonction de mise à jour). *Une fonction h défini par,*

$$h : \Theta \times (0, 1] \to \Theta,$$

est qualifiée de fonction de mise à jour si elle est constante sur chaque intervalle $\theta \in \Theta$ et que pour n'importe quel $\theta, \theta' \in \Theta$ tel que $M(\theta, \theta') > 0$, la dimension de l'ensemble $\{u \in (0, 1] : h(\theta, u) = \theta'\}$ est égale à $M(\theta, \theta')$.

Les concepts d'innovation et de fonction de mise à jour permettent de définir la chaîne de Markov $X^{(i)}$ en terme d'états et des nombres aléatoires. Cette définition est très intuitive puisqu'elle s'apparente au processus utilisé en pratique pour simuler des chaînes de Markovs : la génération d'un nombre aléatoire, ici nommé innovation, détermine la transition de la chaîne à chaque itération. Dans un contexte de simulation en parallèle, nous disons que les innovations sont dépendantes lorsqu'un seul et même nombre aléatoire est utilisé dans toutes les k simulations, et qu'elles sont indépendantes dans le cas contraire. Un autre avantage de cette définition alternative est qu'elle permet de réécrire les chaînes de Markovs de manière récursive, selon l'expression suivante,

$$X_t^{(i)} = h\left(\theta, U_t^{(i)}\right) \quad \text{si} \quad X_{t-1}^{(i)} = \theta.$$

Supposons qu'une série de k simulations en parallèle d'une même chaîne de Markov est en cours depuis un temps $t = -\infty$, que les innovations utilisées son dépendantes et que nous nous intéressons à leurs comportements au temps $t = 0$. Comme il est impossible de suivre le déroulement d'un processus quelconque durant une infinité de temps, l'étude est limitée aux parcours pris par les chaînes à un horizon fini $\{-i, -i + 1, ..., 1, 0\}$ précédent le temps $t = 0$. En utilisant la définition récursive des chaînes de Markovs, le chemin pris par la chaîne de Markov entre les itérations i et j si elle avait comme point de départ θ_i à l'itération i, est noté,

$$H_i^j(\theta_i, U_i) = h_{j-1} \circ ... \circ h_i(\theta_i, U_i).$$

Il est à noter que selon ce scénario, l'utilisation d'innovations dépendantes entraine que lorsque deux chaînes différentes se retrouvent dans un même état à une certaine itération t, alors ces deux chaînes partageront les mêmes chemins H_i^j peut importe la valeur prise par j. Dans ce cas, les chaînes sont dites couplées. La fonction H permet de définir le concept de temps de couplage par le passé.

Définition 5 (Temps de couplage par le passé). *La variable aléatoire*

$$\tau = \min\{-t \geq 1 \mid H_t^0(\theta) = H_t^0(\theta') \quad \forall \theta, \theta' \in \Theta\},$$

est appelée temps de couplage par le passé. De plus, s'il n'existe pas d'entier $-t$ satisfaisant cette relation, la notation $\tau = \infty$ est utilisée.

Le temps de couplage par le passé représente donc, le plus petit nombre d'itérations tel que des chaînes de Markovs évoluant en parallèle soient dans le même état à $t = 0$. Dans ce cas, nous disons que les chaînes sont entrées un état de coalescence. Une condition nécessaire au bon fonctionnement du couplage par le passé est que le temps de couplage par le passé soit fini indépendamment des états occupés par les chaînes au temps $t = 0$: cela assure que la chaîne soit apériodique. Le théorème suivant montre que si le temps de couplage vers le passé est fini pour une certaine série de chaînes de Markovs et que les innovations sont dépendantes, alors les chaînes au temps $t = 0$ sont distribuées selon la distribution stationnaire π.

Théorème 2. *Supposons que* $\mathbb{P}(\tau < \infty) = 1$ *et que* U *représente une série de variables aléatoires iid. selon une loi uniforme sur l'intervalle* $[0, 1]$, *alors pour tout* $k \leq -\tau$,

$$H_k^0(\theta) = H_k^0(\theta').$$

De plus, pour un $k \leq -\tau$ *quelconque,*

$$H_k^0(\theta) = H_{-\tau}^0(\theta') \approx \pi.$$

D'abord, de la définition par récurrence des chaînes de Markovs il découle que $H_k^0(\theta) = \ldots = H_k^0(\theta')$ et $H_t^0 = H_{-\tau}^0$ pour $t \leq -\tau$ quelconque. De plus, par hypothèse $\mathbb{P}(\tau < \infty) = 1$, alors pour $i \in \{1, \ldots, k\}$,

$$\begin{aligned}
P\left(H_0^{-\tau} = \theta\right) &= \lim_{t \to -\infty} \mathbb{P}\left(H_0^{-\tau} = \theta, \tau \leq -t\right), \\
&= \lim_{t \to -\infty} \mathbb{P}\left(H_0^{t} = \theta, \tau \leq -t\right), \\
&= \lim_{t \to -\infty} \mathbb{P}\left(H_0^{t} = \theta\right) - \underbrace{\lim_{t \to -\infty} \mathbb{P}\left(H_0^{t} = \theta, \tau > -t\right)}_{=0}, \\
&= \lim_{t \to -\infty} \mathbb{P}\left(H_0^{t} = \theta\right), \\
&= \lim_{t \to -\infty} \mathbb{P}\left(H_{-t}^{0} = \theta\right) \quad \text{par homogénéité}, \\
&= \pi_\theta. \qquad (A.1)
\end{aligned}$$

Ce résultat justifie l'élaboration d'une méthode générale de couplage par le passé, tel que présenté par l'algorithme A.2.1.

Algorithme A.2.1 Couplage par le passé

Entrées: T le nombre initial d'itérations.
  Poser $t = 0$
  tant que H_t^0 n'est pas une constante **faire**
    tant que $t < T$ **faire**
      Générer une nouvelle valeur de la fonction de mise à jour h_t.
      $H_t^0 = H_{t-1}^0 \circ h_t$.
    fin
    $T = 2T$.
  fin
  return La valeur de H_t^0.

Bien que l'algorithme vérifie le couplage des chaînes qu'au temps $t = 0$, ce phénomène peut se produire a un temps antérieur $t < 0$. En effet, la définition récursive des chaînes de Markovs met en évidence le fait que l'utilisation des mêmes innovations pour la mise à jour de toutes les chaînes entraine que les couplages sont permanents. En pratique, les algorithmes utilisent ce fait pour minimiser les calculs. De plus, pour assurer une bonne estimation de τ, il est nécessaire de simuler un nombre de chaînes de Markovs au cardinal de Θ. Or lorsque Θ est dénombrable, il est impossible de simuler chacun des parcours φ_k^0 : une autre démarche a dû être développée.

L'algorithme général, laisse à l'utilisateur le soin de choisir trois éléments nécessaires au fonctionnement du CPLP, soit la fonction de mise à jour, la méthode de composition de l'ensemble H_T^0, ainsi que la méthode de vérification de la coalescence. Bien qu'il soit simple de composer des procédures optimales et polyvalentes pour la composition de H_T^0 et la vérification de la coalescence, le choix de la fonction de mise à jour n'est jamais triviale et ce pour deux raisons. D'abord, il est souvent nécessaire d'ajuster la fonction de mise à jour pour rencontrer les hypothèses formulées précédemment. Ensuite, lorsque l'espace des états de la chaîne possède certaines propriétés combinatoires, il est préférable de créer une fonction de mise à jour utilisant pleinement ces propriétés pour améliorer la vitesse de coalescence. L'exemple suivant illustre bien les problèmes liés au choix de la fonction de mise à jour. Soit une chaîne de Markov ayant l'espace d'état $\Theta = \{1, 2, 3, 4\}$ ayant comme matrice de transition,

$$M = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

La fonction de mise à jour naturellement liée à cette chaîne peut s'écrire,

$$h^{(1)}(x, u) = \begin{cases} \min((x+1), 4) & \text{si } x = 1, 2, 3 \text{ et } u \leq 1/2 \\ \max((x-1), 1) & \text{si } x = 1, 2, 3 \text{ et } u > 1/2 \\ 3 & \text{si } x = 4 \end{cases}$$

Cette chaîne possède des états périodiques. Or la péridiocité de la chaîne empêche la convergence de la fonction composée $H_T^0 = h_{-1}^{(1)} \circ ... \circ h_t^{(1)}$ vers une fonction constante : il est donc impossible d'utiliser cette fonction avec le couplage par le passé. Pour contourner ce problème, il est généralement conseillé d'utiliser la fonction de mise à jour modifiée,

$$h^{(2)}(x, u_1, u_2) = \begin{cases} h^{(1)}(x, u_1) & \text{si } u_2 \leq 1/2 \\ x & \text{sinon} \end{cases}$$

Cette fonction est celle d'une chaîne partageant les propriétés de la chaîne précédente, mais dont chaque état a une probabilité non nulle d'avoir lui-même comme successeur. De plus, il est possible d'établir un ordre partiel $\preceq$ sur l'espace des états Θ de cet exemple et celui-ci est respecté par la fonction de mise à jour. En utilisant $\leq$, la relation d'ordre usuelle sur les entiers, il est évident que $\forall x, y \in \Theta$,

$$x \leq y \rightarrow f(x, u) \leq f(y, u).$$

Une fonction de mise à jour ayant cette propriété est qualifiée de monotone. Lorsque Θ possède des états x^{max} et x^{min} tel que $x^{min} \preceq x \preceq x^{max}$ pour tous les états de Θ, il est très avantageux d'utiliser une fonction de mise à jour monotone. En effet, lors d'une simulation, si $H_T^0(x^{min}) = H_T^0(x^{max})$

la monotonicité de la fonction de mise à jour nous assure que $H_T^0(\Theta)$ est constante et donc que la coalescence a eu lieu. L'avantage d'exploiter l'ordre partiel de l'espace des états est évident : seulement deux simulations de chaînes sont nécessaires pour obtenir un échantillon parfait, plutôt qu'une simulation par éléments de χ. Dans notre exemple, toujours en utilisant la relation d'ordre $\leq$, l'élément maximal de θ est 4, le minimum est 1 et $h^2(x, u_1, u_2)$ est clairement monotone, alors il est possible de faire le couplage par le passé avec seulement la moitié des calculs nécessaires pour l'utilisation de la méthode générale.

Algorithme A.2.2 Couplage par le passé : cas Θ fini

Entrées: Temps de simulation initial T
Initialiser l'ensemble vide U
tant que $H_T^0(\theta) \neq H_T^0(\theta')$ $\forall \theta, \theta' \in \Theta$ **faire**
Pour chaque état $\theta \in \Theta$, initialiser une chaîne de Markov ayant θ comme graine,
Poser $i = 1$.
pour $i + \#U \leq T$ **faire**
Générer un nombres aléatoires, noté $U_{i+\#U}$ distribués selon une loi uniforme sur $[0, 1]$ et l'ajouter à U.
fin
Poser $i = 1$.
pour $i \leq T$ **faire**
Mettre à jour les chaînes de Markovs à l'aide de U_i.
fin
si $H_T^0(\theta) \neq H_T^0(\theta')$ $\forall \theta, \theta' \in \Theta$ **alors**
$T = 2 * T$;
fin
fin

La méthode de couplage par le passé est basée sur un raisonnement pouvant être facilement traduit en terme d'espace des états continu. Malheureusement, puisqu'il est impossible d'énumérer chacun des états de Θ lorsque ce dernier est de cardinal infini, il est impossible d'utiliser les protocoles présentés jusqu'ici.

A.3 Cas Θ continu

Pour contourner l'impossibilité d'énumérer chacun des $H_k^0(\theta)$ possible, un ensemble B_t est défini comme étant l'ensemble des sous-ensembles contenant au moins un $H_t^0(\theta)$ pour $\theta \in \Theta$. En définissant la variable aléatoire $T = \min\{t \geq 1 \mid \#(B_t) = 1\}$, il est maintenant possible d'élaborer un algorithme de couplage par le passé simulant une loi continue. Cette version repose sur les mêmes hypothèses que dans le cas discret, ainsi que les deux hypothèses supplémentaires suivantes.

1. Il existe une variable aléatoire T tel que $\#(B_{-T}) = 1$ et $P(T \leq \infty) \approx 1$.
2. Pour deux itérations quelconques t et $t+1$, l'ensemble B_{t+1} contient tous les états des chaînes dont les états étaient contenus dans B_t.

Il est facile de prouver par induction qu'une suite d'ensembles B_t satisfaisant la dernière hypothèse la satisfait aussi pour des t_1 et t_2 quelconques. La première condition nous assure que la version du

temps de couplage par le passé dans le cas continu est presque sûrement fini. La deuxième hypothèse a pour conséquence que $H^0_{-T}(\theta) \subset B_1, \forall\theta$. Puisque la définition de T entraine que $\#(B_0) = 1$, alors $\#(H^0_{-T}(\Theta)) = 1$ et $H^0_{-T}(\theta) \equiv H^0_{\tau}(\theta)$. En conséquence, un algorithme de couplage par le passé basé sur ces hypothèses génère une chaîne ayant atteint sa loi stationnaire. L'algorithme A.3.1 suit cette démarche générale.

Algorithme A.3.1 Couplage par le passé général, cas continu

Entrées: Nombre d'itérations initial T.
  Poser $t = -T$.
  Poser $B_t = \Theta$.
  Poser $U = \emptyset$.
  tant que $\#(B) \neq 1$ **faire**
    Poser $i = 0$.
    pour $\#(U) + i \leq M$ **faire**
      Générer $U_{\#(U)+i}$ selon une loi uniforme sur $[0, 1]$.
      Ajouter $U_{\#(U)+i}$ à U.
    fin
    tant que $\#(B_t) = \infty$ et $t > 0$ **faire**
      Poser $t = t + 1$.
      Poser $B_t \leftarrow$ Ensemble contenant $h(B_{t-1}, U_t)$.
      tant que $t > 0$ **faire**
        Poser $t = t + 1$.
        Poser $B_t \leftarrow$ Ensemble contenant $h(B_{t-1}, U_t)$.
      fin
    fin
    si $\#(B_0) \neq 1$ **alors**
      $T = 2 * T$.
      $t = T$
    sinon
      Retourner B_0.
    fin
  fin

Le présent chapitre se termine par la présentation de trois différents algorithmes utilisant cette démarche. Par mesure d'économie, l'accent sera porté sur les cas univariés.

A.3.1 Couplage par le passé à lecture unique

Cette technique est une modification du couplage par le passé pour qu'il se comporte comme un algorithme dont l'index des itérations avance dans le temps. Pour ce faire, Wilson [1] redéfini la fonction H comme étant une composition de blocs indépendants identiquement distribuée de longueur t,

$$H^0_{-nt} = H^0_{-t} \circ H^{-t}_{-2t} \circ ... \circ H^{-(n-1)t}_{-nt}.$$

1. DB Wilson-How to Couple from the Past Using a Read-Once Source of Randomness - 1999 - en.scientificcommons.org.

Wilson a montré que s'il existe un t suffisamment grand tel pour la probabilité que H_{-t}^{0} soit en coalescence, alors l'algorithme de couplage à lecture unique est équivalent au *CFTP*. En définissant les variables,

$$\tau_a = \inf\{n \geq 0 \text{ tel que } \#B(\Theta) = 1\}, \quad X_a = H_{\tau_a}(\Theta),$$

et

$$\tau_z = \inf\{n \geq \tau_a \text{ tel que } \#B(\Theta) - \tau_a = 1\}, \quad X_z = H_{\tau_a+1}^{\tau_a+\tau_z}(\Theta),$$

le pseudo-code du couplage par le passé à lecture unique est donné par l'algorithme

Algorithme A.3.2 Couplage par le passé à lecture unique

A.3.2.

$t = 0$.
tant que $\#B(\Theta) \neq 1$ **faire**
  $t = t + 1$.
fin
si $\#B(\Theta) = 1$ **alors**
  $\tau_a = t$.
  $X_a = B_{\tau_a}(\Theta)$.
fin
$t = \tau_a + 1$.
tant que $\#Bt(\Theta) \neq 1$ **faire**
  $t = t + 1$.
fin
si $\#B_t(\Theta) = 1$ **alors**
  $\tau_z = t - \tau_a$.
  $X_z = B_{\tau_z+1}^{\tau_z+\tau_z} X_a$.
fin
return X_z

Cet algorithme présente plusieurs avantages par rapport au *CFTP* : d'abord, il est plus simple à implémenter. Ensuite, puisque l'algorithme débute au temps $t = 0$ et que chaque itération se projette dans l'avenir plutôt que le passé, n'importe quel nombre aléatoire peut être utilisé pour mettre la chaîne à jour. Donc l'algorithme ne nécessite pas l'utilisation des mêmes innovations à chaque itération des simulations et demande moins de ressource mémoire. Finalement, cet algorithme est beaucoup plus économique que le *CFTP* pour effectuer plusieurs tirages d'une distribution donnée. En effet, dans la deuxième boucle de l'algorithme, à l'itération t, si la coalescence est détectée, alors la valeur de X_{t-1} est retournée comme un échantillon distribué selon π. Pour obtenir un échantillon supplémentaire et indépendant de X_{t-1}, il est possible d'utiliser BX_{t-1} comme étant le point d'entrée de la seconde boucle de l'algorithme. Du coup, les calculs liés à la première boucle de l'algorithme sont évités et la coalescence est atteinte beaucoup plus rapidement que lors de la simulation du premier échantillon. De plus, lorsque Θ est de cardinal fini, l'algorithme se simplifie puisque le coupleur n'est pas à utiliser.

A.3.2 Coupleur multigamma

Supposons que l'espace des états Θ de la chaîne de Markov à simuler est un sous-ensemble de nombre réels. De plus, il est supposé qu'il existe une fonction g enveloppant inférieurement le noyau

de la chaîne, i.e, $g(\theta^{'}) \leq K(\theta^{'} \mid \theta), \quad \forall\theta \in \Theta$. Le noyau peut être réécrit sous forme d'une moyenne pondérée

$$K(\theta^{'} \mid \theta) = (1-\rho)\left[\frac{K(\theta^{'} \mid \theta) - g(\theta^{'})}{1-\rho}\right] + \rho\frac{g(\theta^{'})}{\rho}.$$

Cette redéfinition est vraie pour une fonction ρ quelconque, mais ici le cas intéressant est lorsque $\rho = \int g(\theta)d\theta$. Cette définition du noyau met en évidence le fait qu'il est possible d'utiliser la simulation gamma pour simuler cette chaîne. Cette technique consiste à générer une variable aléatoire U distribuée selon une loi uniforme $U_{[0,1]}$ et si $U < \rho$ de simuler une variable de loi,

$$p(\theta) = \rho^{-1}\int_{-\infty}^{\theta} g(x)dx.$$

Sinon il suffit de simuler une variable aléatoire de loi,

$$q(\theta \mid \theta^{'}) = (1-\rho)^{-1}\int_{\infty}^{\theta}\Big[K(x \mid \theta^{'}) - g(x)\Big]\,dx.$$

En d'autres termes, nous avons défini la fonction de mise à jour de la chaîne de la manière suivante,

$$h(\theta, U) = \begin{cases} p & \text{si } U < \rho \\ q & \text{sinon} \end{cases}$$

Puisque la fonction p ne dépend pas de l'état $\theta^{'}$ dans lequel se trouve la chaîne à chaque itération, si à une certaine itération la variable U est plus petite que ρ, la définition de h entraine que les chaînes simulées en parallèles se trouveront toutes dans le même état à l'itération suivante. Cela nous assure que l'état de coalescence est atteint par les chaînes simulées par cet algorithme, en autant que $U < \rho$ à une certaine itération.

L'algorithme A.3.3 respecte les hypothèses propres aux algorithmes de couplages par le passé généraux. Une preuve générale est formulée dans la section suivante portant sur le coupleur multigamma partitionné, une généralisation de l'algorithme précédent.

A.3.3 Coupleur multigamma partitionné

Pour facilité la recherche d'une fonction $g(\theta)$ facile à échantillonner, cette technique partitionne l'espace des états en m sous-ensemble E_i et applique le coupleur multigamma sur chacun de ceux-ci. Pour ce faire, m nombres réels $e_1, e_2, ..., e_{m-1}$ sont choisis de manière à bien couvrir l'espace des états. Puis les E_i sont définis récursivement selon les équations,

$$E_1 = \Theta \cap (-\infty, e_1),$$

$$E_2 = \Theta \cap [e_1, e_2),$$

$$\ldots$$

$$E_1 = \Theta \cap [e_{m-1}, \infty).$$

Il est supposé que pour chaque E_i, il existe une fonction g_i tel que $\forall\theta \in E_i$, $g_i(\theta^{'}) \leq K(\theta^{'} \mid \theta)$. Tout comme avec le coupleur multigamma, les fonctions suivantes doivent être définies,

$$\rho_i = \int g_i(\theta)d\theta \quad p_i(\theta) = \rho_i^{-1}\int_{-\infty}^{\theta} g_i(x)dx \quad q_i(\theta \mid \theta^{'}) = (1-\rho_i)^{-1}\int_{\infty}^{\theta}\Big[K(x \mid \theta^{'}) - g_i(x)\Big]\,dx.$$

Algorithme A.3.3 Coupleur multigamma

Entrées: Nombre d'itérations initial T
Poser $t = -T$.
Poser $B_t = \Theta$.
Poser $U = \emptyset$.
tant que $t \leq 0$ **faire**
si $B_t = \Theta$ **alors**
Générer les variables U_t^1 et U_t^2 distribuées selon $U_{[0,1]}$.
Calculer ρ.
si $U_t^1 < \rho$ **alors**
Générer θ_t selon $R^{-1}(U_t^2)$.
Ajouter θ_t à l'ensemble B_t.
sinon si $U_t^1 > \rho$ **alors**
Ajouter Θ à l'ensemble B_t.
$t = t + 1$.
fin
fin
si $\#B_t = 1$ **alors**
Générer les variables U_t^1 et U_t^2 distribuées selon $U_{[0,1]}$.
Calculer ρ.
si $U_t^1 < \rho$ **alors**
Générer θ_t selon $R^{-1}(U_t^2)$.
Ajouter θ_t à l'ensemble B_t.
sinon si $U_t^1 > \rho$ **alors**
Générer θ_t selon $Q^{-1}(U_t^2 \mid B_t)$.
Ajouter θ_t à l'ensemble B_t.
fin
$t = t + 1$.
fin
si $\#B_0 = 1$ **alors**
Retourner B_0.
sinon
$T = 2T$.
$t = -T$
fin
fin

Donc, en notant les variables aléatoires simulées $p \approx p_i(\theta)$ et $q \approx q_i(\theta \mid \theta')$ la fonction de mise à jour est définie par,

$$h(\theta, U) = \begin{cases} p & \text{si } U < \rho_i \\ q & \text{sinon} \end{cases}$$

Algorithme A.3.4 Coupleur multigamma partitionné

Entrées: Nombre d'itérations initial T
Poser $t = -T$.
Poser $B_t = \Theta$.
Poser $U = \emptyset$.
tant que t<0 **faire**
si $B_t = \Theta$ **alors**
Générer les variables U_t^1 et U_t^2 distribuées selon $U_{[0,1]}$.
Calculer les ρ_i.
si $U_t^1 < \min\{\rho_1, ..., \rho_i\}$ **alors**
Générer les θ_i selon $R_i^{-1}(U_t^2)$.
Ajouter les θ_i à l'ensemble B_t.
sinon
Ajouter Θ à l'ensemble B_t.
$t = t + 1$.
fin
fin
si $\#B_t = m$ **alors**
Générer les variables U_t^1 et U_t^2 distribuées selon $U_{[0,1]}$.
Poser $B_{t+1} = \emptyset$.
$i = 1$.
pour $i < m + 1$ **faire**
Calculer ρ_i.
si $U_t^1 < \rho_i$ **alors**
$B_{t+1} \leftarrow B_{t+1} \cup_{i=1}^{m} \{R_i^{-1}(U_t^2) : A_i \cap B_t \neq \emptyset\}$.
sinon si $U_t^1 > \rho_i$ **alors**
$B_{t+1} \leftarrow B_{t+1} \cup_{i=1}^{m} \{Q_i^{-1}(U_t^2 \mid \theta) : \theta \in A_i \cap B_t\}$.
$t = t + 1$.
fin
fin
si $\#B_0 = 1$ **alors**
Retourner B_0.
sinon
$T = 2T$.
$t = -T$
fin
fin
fin

Il reste à vérifier que cet algorithme est un algorithme de couplage par le passé.

Théorème 3. *L'algorithme du coupleur multigamma partitionné est un algorithme de couplage par le passé valide.*

La preuve de ce théorème se fait en trois parties. D'abord, il faut démontrer que l'algorithme se termine dans un temps fini. Notons,

$$\rho = \min\{\rho_1, ..., \rho_m\} \quad \tau = \inf\{t \in \mathbb{N} \mid U_t^1 \leq \rho\}.$$

Pour tous $\theta \in \Theta$, une chaîne qui est dans cet état au temps $t = -T$ se retrouvera, au temps $t = \tau - T$, dans un état inclue dans l'ensemble fini $B_{\tau-T} = \{\Theta_1, ..., \Theta_n\}$ dont les éléments peuvent être égaux. Puisque τ est distribuée selon une loi géométrique de paramètre $p = \rho$, nous déduisons que $P(\#(B_{-T+1}) = n) = P(\tau = 1) = \rho$. Supposons maintenant qu'il existe une partition de l'espace des états $\Theta = \cap_{i=1}^{n} E_i$ et un entier $k \in \{1, 2, ..., n\}$ tel que pour tout $i, j \in \{1, 2, ..., n\}$,

$$\int_{E_k} \min(g_i(\theta), g_j(\theta)) d\theta \geq \epsilon > 0,$$

pour un certain $\epsilon \in \mathbb{R}$. Cette intégrale conditionné sur n, implique qu'avec une probabilité ϵ non nulle des chaînes dans des états θ_i et θ_j au temps $t = -T + 1$ seront dans des états $\theta_i', \theta_j' \in E_k$ au temps $t = -T + 2$. De même, au temps $t = -T + 3$, $P(\#(E_k) = 1) = \rho$, donc la probabilité que les incréments respectent $\#(B_{-T+3}) - \#(B_{-T+1}) \leq -1$ est au moins égale à $\rho\epsilon$. Puisque les incréments sont indépendants, il est possible de prouver par induction que $P(\#(B_{-T+2n+1}) = 1) \geq \rho(\rho\epsilon)^{n-1} > 0$ et qu'ainsi l'algorithme se termine en un temps fini.

Ensuite, nous devons prouver que pour chaque itération t et $t + 1$ l'ensemble B_{t+1} contient tous les états des chaînes dont les états étaient contenus dans B_t. Posons $t_c = \min\{t > -T \mid U_t^1 < \rho_i\}$, alors pour tout $t < t_c$ $B_t = B_{t-1} = \Theta$. Pour $t = t_c$, à l'itération suivantes toutes les chaînes sont dans l'état $\theta_{t+1} = R_i^{-1}(U_t^2)$ pour $i = 1, 2, ..., m$, puisque par définition de t_c $U_t^1 < \rho_i$ pour $i = 1, 2, ..., m$ ce qui entraine que $\theta_{t_c+1} \in B_{t+1}$. De plus, par design de l'algorithme, il est immédiat que $\theta_{t_c+1} \in B_{t+1} \quad \forall t > t_c$.

Finalement, nous devons prouver que la chaîne créée par cet algorithme est la chaîne de Markov à simuler. Or de par la construction de la fonction de mise à jour h, nous savons que la suite des variables aléatoires $\theta_{t+1} = h(\theta_t, U_t^1, U_t^2)$ forment une chaînes de Markov de noyau f. Donc le coupleur multigamma partitionné est un *CFTP* valide.

Annexe B

Code Matlab

B.1 Échantillonnage préférentiel

```
function [I E]= testIS(t,fonct_instr,fonct_cible)
% Fonction utilisant l'échantillonnage préférentiel pour simuler une
% variable aléatoire. La fonction prend comme argument le nombre
% de point à simuler, ainsi que l'indice de la fonction instrumentale et
% de la fonction cible. Ces derniers correspond aux indice utilisé par les
% fonction serie_va et eval_distribution. La fonction retourne
% l'estimation de la moyenne (I) de la fonction cible et l'estimation de la
% distribution (E).

%mu=4;
%sigma=1;
%mu1=8;
%sigma1=2;
%mu_inst=4;
%sigma_inst=2;
%alpha=0.4;
%lambda=1.5;
%scale=1;
mu=5;
sigma=3.3;

borneinf=-5;
bornesup=20;
intervalle=250;


%Nombre d'itérations
num_iterations = t;

%Candidat distribuée selon la loi instrumentale
[candidat X]=serie_va(mu,sigma,fonct_instr,num_iterations,1);

W=zeros(num_iterations ,1);
I=zeros(num_iterations ,1);
for i=1 :num_iterations
    W(i,1)=eval_distribution(candidat(i,1),fonct_cible)/(eval_distribution(candidat(i,1),fonct_instr,mu,sigma))
    I(i,1)=W(i,1)*candidat(i,1);
end
```

```
E=zeros(intervalle,4) ;

for i=1 :intervalle
    E(i,1)=borneinf+(i-1)*((bornesup-borneinf)/intervalle) ;
    E(i,2)=borneinf+i*((bornesup-borneinf)/intervalle) ;
end
for i=1 :intervalle
    for j=1 :num_iterations
        if candidat(j,1)<E(i,2)
            E(i,3)=E(i,3)+W(j,1) ;
        end
    end
end
E( :,4)=E( :,3)./num_iterations ;
```

B.2 Fonction serieva

```
    function [candidatX,X] = serie_va(mu,sigma,d,m,n)
% Fonction générant une série de réalisation d'une variable aléatoire
% distribuéeselon une distribution donnée. La fonction prend comme
% argument les paramètres mu et sigma de la distribution, l'indice
% désignant la distribution choisie, ainsi que les dimensions m x n de la
% matrice contenant l'échantillon généré. La fonction retourne
% l'échantillon, ainsi que les nombres aléatoires utilisés pour les
% générer. Les indices des distributions sont :

%1-Loi normale centrée réduite
%2-Loi normale de paramètre mu et sigma
%3-loi exponentielle de paramètre lambda
%4-Mixture de deux lois normales
%5-Loi de Pareto
%6-Loi de Fréchet
%7-Loi de Gamma

% Paramètres de la deuxième loi normale utilisée dans la définition de la
% mixture.
mu2=6 ;
sigma2=1 ;
theta=1 ;

if nargin==3
    m=1 ;
    n=1 ;
```

```
end

if d==1
    candidatX=zeros(m,n);
    X=rand(m,n);
    for i=1 :m
        candidatX=normrnd(0,1,m,n);
    end
else
    if d==2
        candidatX=zeros(m,n);
        X=rand(m,n);
        for i=1 :m
            candidatX=normrnd(mu,sigma,m,n);
        end
    else
        if d==3
            candidatX=zeros(m,n);
            X=rand(m,n);
            for i=1 :m
                candidatX=exprnd(mu,m,n);
            end
        else
            if d==4
                candidatX=zeros(m,n);
                X=rand(m,n);
                for i=1 :m
                    if temp<=0.4
                        candidatX(i,1)=normrnd(mu,sigma);
                    else
                        candidatX(i,1)=normrnd(mu2,sigma2);
                    end
                end
            else
                if d==5
                    candidatX=zeros(m,n);
                    X=rand(m,n);
                    for i=1 :m
                        candidatX=gprnd(-1,mu,sigma,m,n);
                    end
                else
                    if d==6
                        candidatX=zeros(m,n);
                        X=rand(m,n);
                        for i=1 :m
```

```
                candidatX=gevrnd(theta^(-1),sigma,mu,m,n) ;
            end
        else
            if d==7
                candidatX=zeros(m,n) ;
                X=rand(m,n) ;
                for i=1 :m
                    candidatX=gamrnd(mu,sigma,m,n) ;
                end
            else
                if d==8
                    candidatX=zeros(m,n) ;
                    X=rand(m,n) ;
                    for i=1 :m
                        candidatX(i,1)=(-log(1-X(i,1))/2) ;
                    end
                end
            end
        end
    end
end
end
```

B.3 Fonction evaldistribution

```
function pX = eval_distribution(x,d,mu,sigma,mu1,sigma1,alpha)
% Fonction retournant la valeur de la probabilité p(X=x) selon une
% distribution données. La fonction prend comme argument, la valeur de x,
% l'indice de la distribution choisie, ainsi que les paramètres optionnels
% mu, sigma,mu1, sigma1 et alpha de la distribution. Les indices des
% distributions, sont :

%1-Loi normale centrée réduite
%2-Loi normale de paramètre mu et sigma
%3-loi exponentielle de paramètre lambda
%4-Mixture de deux loi normale
%5-Loi de Pareto
%6-Loi de Fréchet
%paramètre lambda

%Initialise les constantes selon la définition de la fonction utilisée
if nargin==3
```

```
    lambda=mu;
    alpha=mu;
else
    if nargin==4
        alpha=mu;
        t=sigma;
    else
        if nargin==5
            alpha=mu;
            t=sigma;
            scale=mu1;
        else
            if nargin==2
                mu=4;
                sigma=1;
                mu1=8;
                sigma1=2;
                alpha=0.4;
                lambda=1;
                t=4;
                scale=1;
            else
                if nargin==3
                    lambda=mu;
                    t=4;
                    scale=1;
                end
            end
        end
    end
end

pX=zeros(length(x),1);


if d==1
    for i=1 :length(x)
        pX(i,1) =(1/(sqrt(2*pi)))*exp((-x(i,1)^2)/2);
    end
else
    if d==2
        for i=1 :length(x)
            pX(i,1) =normpdf(x,mu,sigma);
        end
    else
```

```
        if d==3
            for i=1 :length(x)
                if x(i,1)<=0
                    pX(i,1)=0;
                else
                    pX(i,1)=((1/lambda)*exp(-lambda*x(i,1)));
                end
            end
        else
            if d==4
                for i=1 :length(x)
                    pX(i,1) =(alpha)*(1/(sigma*sqrt(2*pi)))*exp((-1/2)*(((x(i,1)-mu)/sigma)^2))+(1-
alpha)*(1/(sigma1*sqrt(2*pi)))*exp((-1/2)*(((x(i,1)-mu1)/sigma1)^2));
                end
            else
                if d==5
                    for i=1 :length(x)
                        pX(i,1)=gppdf(x,0.8,1,0);
                    end
                else
                    if d==6
                        for i=1 :length(x)
                            if x(i,1)<=t
                                pX(i,1)=0;
                            else
                                pX(i,1)=(alpha/scale).*((x(i,1)-t)./scale).^(-1-alpha).*exp(-((x(i,1)-t)./scale).^(-
alpha));
                            end
                        end
                    end
                end
            end
        end
    end
end
```

B.4 Fonction MHsimple

```
    function [X] = MHsimple(c,i,t,x0)
% Fonction utilisant l'algorithme de Metropolis-Hasting pour simuler une
% variable aléatoire distribuée selon une distribution donnée. Cette
% fonction utilise la fonction iterationMH pour effectuer l'étape MCMC.

%Période de chauffe
```

```
chauffe = c;
%Nombre de candidats écarté entre les itérations
interval = i;
%Taille de l'échantillon
n = t;
%Graine
x = x0;
% Mémoire
X = zeros(n,1); %échantillon
tauxAccep = [0 0]; %Taux d'acceptation
% Période de chauffe de la simulation
for i = 1 :chauffe
    [x,a] = iterationMH(x);
    tauxAccep = tauxAccep + [a 1];
end
%Étape Métropolis-Hasting
for i = 1 :n
    for j = 1 :interval
        [x,a] = iterationMH(x);
        tauxAccep = tauxAccep + [a 1];
    end
    %Ajoute le ieme candidat
    X(i) = x;
end
X=sort(X);
M=mean(X);
```

B.5 Fonction IterationMH

```
    function [x1,a] = iterationMH(x0)
% Fonction effectuant l'étape d'évaluation du critère d'acceptation et de
% selection de la valeur simulée. Cette fonction utilise la fonction
% serie_va pour générer les candidats et la fonction eval_distribution
% pour calculer la probabilité p(X=x) selon la distribution désirée.

%D est le choix de la distribution instrumentale
instr=2;
sigma_instr=1;

%Choix de la distribution cible
cible=4;
%Moyenne et écart-type de la distribution cible
mu_cible=4;
sigma_cible=1;
```

```
%Génère un nouveau candidat
xp = serie_va(x0,sigma_instr,instr,1,1);

probAcceptation =min((eval_distribution(xp,cible)*...
eval_distribution(xp,instr,x0,sigma_instr)) /...
(eval_distribution(x0,cible)*eval_distribution(x0,instr,xp,sigma_instr)),1);
u = rand;
if u <= probAcceptation
    x1 = xp;
    a = 1;
else
    x1 = x0;
    a = 0;
end
```

B.6 Fonction test kolmogorov-Smirnov

```
    function [X T]=Test_Kolmogorov(X0,D,Nombre_intervalle)
% Fonction calculant la valeur de la statistique du test de
% Kolmogorov-Smirnov pour une distribution et un échantillon donnée. La
% fonction prend comme argument la série de données à tester, l'indice de la
% distribution de référence et le nombre d'intervalle à utiliser pour
% calculer les valeurs de la fonction empirique. La fonction retourne les
% bornes des intervalles, ainsi que la valeur du test.

%Paramètre de la distribution cible
mu1=4;
sigma1=1;
mu2=8;
sigma2=2;

%Construction du domaine d'intérêt
Borne_inf=min(X0);
Borne_sup=max(X0);

%Construction des intervalles sur lesquels sont calculés les différentes
%fréquences
borne=zeros(Nombre_intervalle,3);

for i=1 :Nombre_intervalle
    borne(i,1)= Borne_inf+i*((Borne_sup-Borne_inf)/Nombre_intervalle);
end
```

```
%Calcul du nombres des points dans chaque itervalle
for i=1 :Nombre_intervalle
    for j=1 :length(X0)
        if X0(j,1)<=borne(i,1)
            borne(i,2)=borne(i,2)+1 ;
        end
    end
end

%Calcul des fréquences
borne( :,3)=borne( :,2)./length(X0) ;

%Détermine la valeur du test
stat=0 ;
fx=zeros(length(borne( :,3)),1) ;
if D==1 %Mixture de deux lois normales
    for i=1 :length(borne( :,3))
        x=borne(i,1) ;
        fx(i,1)=0.4*normcdf(x,mu1,sigma1)+0.6*normcdf(x,mu2,sigma2) ;
        stat=max([abs(borne(i,3)-fx(i,1)) stat]) ;
    end
else
    if D==2 %lois normale
        for i=1 :length(borne( :,3))
            x=borne(i,1) ;
            fx(i,1)=normcdf(x,mu1,sigma1) ;
            stat=max([abs(borne(i,3)-fx(i,1)) stat]) ;
        end
    end
end
T=stat*sqrt(length(X0)) ;
X=borne ;
```

B.7 Fonction MHBivarie

```
function X=MHBivarie(C,I,T)

%Graine de l'algorithme
theta_0=[4 4] ;

%Taille de l'échantillon
t=T ;

%Longueur de la période de chauffe
```

```
c=C;

%Ventillation de l'échantillon
lag=I;

%Moyenne et écart-type des composantes de la première loi normale de la
%mixture
mu1=4;
sigma1=1;

mu2=4;
sigma2=1;

%Moyenne et écart-type des composantes de la deuxième loi normale de la
%mixture
mu3=8;
sigma3=2;

mu4=4;
sigma4=2;

%Élément des matrices de covariance
p1=0.9;
p2=0.4;

%Matrice de covariance
s=[1 0;0 1];

%coefficient de mixage
alpha=1;

%variable
theta=theta_0;

%Initialisation du vecteur des observations
X=zeros(t,2);

%Période de chauffe
for i=1 :c

    g=mvnrnd(theta,s); %fonction instrumentale

    x1=alpha*mvnpdf(theta,[mu1 mu2],[sigma1 p1 ;p1 sigma3])+(1-alpha)*mvnpdf(theta,[mu3 mu4],[sigma2
p2 ;p2 sigma4]);
    x2=alpha*mvnpdf(g,[mu1 mu2],[sigma1 p1 ;p1 sigma3])+(1-alpha)*mvnpdf(g,[mu3 mu4],[sigma2
```

```
p2 ; p2 sigma4]) ;

    %Critère d'acceptation
    a=min(1,(x2/x1)*(mvnpdf(g,theta,s)/mvnpdf(theta,g,s))) ;
    u=rand() ;
    if u<=a
        theta(1,1)=g(1,1) ;
        theta(1,2)=g(1,2) ;
    end
end

%Étape MH
for i=1 :t
    for j=1 :lag
        %fonction instrumentale
        g=mvnrnd(theta,s) ;
        x1=alpha*mvnpdf(theta,[mu1 mu2],[sigma2 p1 ;p1 sigma1])+(1-alpha)*mvnpdf(theta,[mu3
mu4],[sigma3 p2 ;p2 sigma4]) ;
        x2=alpha*mvnpdf(g,[mu1 mu2],[sigma2 p1 ;p1 sigma1])+(1-alpha)*mvnpdf(g,[mu3 mu4],[sigma3
p2 ; p2 sigma4]) ;

        %Critère d'acceptation
        a=min(1,(x2/x1)*(mvnpdf(g,theta,s)/mvnpdf(theta,g,s))) ;

        u=rand() ;
        if u<=a
            theta(1,1)=g(1,1) ;
            theta(1,2)=g(1,2) ;
        end
    end
    X(i,1)=theta(1,1) ;
    X(i,2)=theta(1,2) ;

end
```

B.8 Fonction Gibbs2D

```
    function X= Gibbs2D(C,I,T)
% Fonction utilisant l'algorithme de Gibbs pour simuler une mixture de loi
% normale multivariée.

%Graine de l'algorithme
theta_0=[4 4] ;
```

```
%Taille de l'échantillon
t=T;

%Longueur de la période de chauffe
c=C;

%Ventillation de l'échantillon
lag=I;

%Moyenne et écart-type des composantes de la première loi normale de la
%mixture
mu1=4;
sigma1=1;

mu2=4;
sigma2=1;

%Moyenne et écart-type des composantes de la deuxième loi normale de la
%mixture
mu3=8;
sigma3=2;

mu4=4;
sigma4=2;

%Coefficient de corrélation
p1=0.3;
p2=0.4;


%coefficient de mixage
alpha=0.4;

%Variable d'échantillonnage
theta=theta_0;

%Initialisation du vecteur des observations
X=zeros(t,2);

%Période de chauffe
for i=1 :c
    %Loi conditionnelle P(X1|X2) de la première loi normale
    X1X2=(mu1+(sigma1/sigma2)*p1*(theta(1,2)-mu2));
    VX1X2=(1-p1^2)*sigma1^2;
    %Loi conditionnelle P(X1|X2) de la deuxième loi normale
```

```
    bX1X2=(mu3+(sigma3/sigma4)*p2*(theta(1,2)-mu4));
    bVX1X2=(1-p2^2)*sigma3^2;

    u=rand();
    if u<=alpha
        theta(1,1)=normrnd(X1X2,sqrt(VX1X2));
    else
        theta(1,1)=normrnd(bX1X2,sqrt(bVX1X2));
    end

    %Loi conditionnelle P(X2|X1) de la première loi normale
    X2X1=(mu2+(sigma2/sigma1)*p1*(theta(1,1)-mu1));
    VX2X1=(1-p1^2)*sigma2^2;
    %Loi conditionnelle P(X2|X1) de la deuxième loi normale
    bX2X1=(mu4+(sigma4/sigma3)*p2*(theta(1,1)-mu3));
    bVX2X1=(1-p2^2)*sigma4^2;

    u=rand();
    if u<=alpha
        theta(1,2)=normrnd(X2X1,sqrt(VX2X1));
    else
        theta(1,2)=normrnd(bX2X1,sqrt(bVX2X1));
    end
end

for i=1 :t
    for j=1 :lag
        %Loi conditionnelle P(X1|X2) de la première loi normale
        X1X2=(mu1+(sigma1/sigma2)*p1*(theta(1,2)-mu2));
        VX1X2=(1-p1^2)*sigma1^2;
        %Loi conditionnelle P(X1|X2) de la deuxième loi normale
        bX1X2=(mu3+(sigma3/sigma4)*p2*(theta(1,2)-mu4));
        bVX1X2=(1-p2^2)*sigma3^2;

        u=rand();
        if u<=alpha
            theta(1,1)=normrnd(X1X2,sqrt(VX1X2));
        else
            theta(1,1)=normrnd(bX1X2,sqrt(bVX1X2));
        end

        %Loi conditionnelle P(X2|X1) de la première loi normale
        X2X1=(mu2+(sigma2/sigma1)*p1*(theta(1,1)-mu1));
        VX2X1=(1-p1^2)*sigma2^2;
        %Loi conditionnelle P(X2|X1) de la deuxième loi normale
```

```
        bX2X1=(mu4+(sigma4/sigma3)*p2*(theta(1,1)-mu3));
        bVX2X1=(1-p2^2)*sigma4^2;

        u=rand();
        if u<=alpha
            theta(1,2)=normrnd(X2X1,sqrt(VX2X1));
        else
            theta(1,2)=normrnd(bX2X1,sqrt(bVX2X1));
        end
    end
    X(i,1)=theta(1,1);
    X(i,2)=theta(1,2);
end
```

B.9 Fonction hit and run

```
    function [ Probabilite X]= Hitandrun(C,I,T)
% Fonction utilisant l'algorithme hit and run pout simuler une loi normale
% bivariée fortement corrélée.

%Graine de l'algorithme
theta_0=[4 4];

%Taille de l'échantillon
t=T;

%Longueur de la période de chauffe
c=C;

%Ventillation de l'échantillon
lag=I;

%Coefficient de corrélation
p=0.9;

%Moyenne et écart-type des composantes de la loi normale
mu=[4 4];
sigma= [1 p; p 1];

%Multiple du vecteur d'échantillonnage(doit être un nombre rationnel)
pas=0.01;
```

```
%Longueur maximale du vecteur d'échantillonnage(doit être pair)
longueur=1;


%Tableau contenant les probabilitées le long du vecteur
Probabilite=zeros((1/pas)*longueur,4);

%Variable d'échantillonnage
theta=theta_0;

%Initialisation du vecteur des observations
X=zeros(t,2);

%Période de chauffe
for i=1 :c
    %Génère le vecteur déterminant l'échantillonnage.
    e1=normrnd(0,1);
    e2=normrnd(0,1);
    norme=sqrt(e1^2+e2^2);
    e1=e1/norme;
    e2=e2/norme;

    %Calcul des probabilités le long du vecteur e=[e1 e2]
    moitier=longueur/2;
    for j=1 :(1/pas)*moitier
        Probabilite(j,1)=theta(1,1)+(-j)*pas*e1;
        Probabilite(j,2)=theta(1,2)+(-j)*pas*e2;
        Probabilite(j,3)=mvnpdf([Probabilite(j,1) Probabilite(j,2)],mu,sigma);
        if Probabilite(j,3)<0.001
            Probabilite(j,3)=0;
        end
        Probabilite(j,4)=j;
    end
    for j=(1/pas)*moitier+1 :(1/pas)*longueur
        temp=j-((1/pas)*moitier);
        Probabilite(j,1)=theta(1,1)+temp*pas*e1;
        Probabilite(j,2)=theta(1,2)+temp*pas*e2;
        Probabilite(j,3)=mvnpdf([Probabilite(j,1) Probabilite(j,2)],mu,sigma);
        if Probabilite(j,3)<0.001
            Probabilite(j,3)=0;
        end
        Probabilite(j,4)=j;
    end
    if sum(Probabilite( :,3))>0
```

```
        indice= randsample(Probabilite( :,4),1,true,Probabilite( :,3)) ;
        theta(1,1)=Probabilite(indice,1) ;
        theta(1,2)=Probabilite(indice,2) ;
    else
        theta(1,1)=theta(1,1) ;
        theta(1,2)=theta(1,2) ;
    end

end
i=1 ;
while i<t
    for a=1 :lag
        %Génère le vecteur déterminant l'échantillonnage.
        e1=normrnd(0,1) ;
        e2=normrnd(0,1) ;
        norme=sqrt(e1^2+e2^2) ;
        e1=e1/norme ;
        e2=e2/norme ;

        %Calcul des probabilités le long du vecteur e=[e1 e2]
        for j=1 :(1/pas)*moitier
            Probabilite(j,1)=theta(1,1)+(-j)*pas*e1 ;
            Probabilite(j,2)=theta(1,2)+(-j)*pas*e2 ;
            Probabilite(j,3)=mvnpdf([Probabilite(j,1) Probabilite(j,2)],mu,sigma) ;
            if Probabilite(j,3)<0.00001
                Probabilite(j,3)=0 ;
            end
            Probabilite(j,4)=j ;
        end
        for j=(1/pas)*moitier+1 :(1/pas)*longueur
            temp=j-((1/pas)*moitier) ;
            Probabilite(j,1)=theta(1,1)+temp*pas*e1 ;
            Probabilite(j,2)=theta(1,2)+temp*pas*e2 ;
            Probabilite(j,3)=mvnpdf([Probabilite(j,1) Probabilite(j,2)],mu,sigma) ;
            if Probabilite(j,3)<0.00001
                Probabilite(j,3)=0 ;
            end
            Probabilite(j,4)=j ;
        end

        if sum(Probabilite( :,3))>0
            indice= randsample(Probabilite( :,4),1,true,Probabilite( :,3)) ;
            theta(1,1)=Probabilite(indice,1) ;
            theta(1,2)=Probabilite(indice,2) ;
        else
```

```
            theta(1,1)=theta(1,1);
            theta(1,2)=theta(1,2);
        end
    end

    X(i,1)=theta(1,1);
    X(i,2)=theta(1,2);
    i=i+1;
end

end
```

B.10 Fonction Snooker

```
    function [ Probabilite X]= Snooker(C,I,T)
% Fonction utilisant l'algorithme hit and run pout simuler une loi normale
% bivariée fortement corrélée.

%Graine de l'algorithme
theta_0=[4 4];

%Taille de l'échantillon
t=T;

%Longueur de la période de chauffe
c=C;
if c<10
    c=2;
end

%Ventillation de l'échantillon
lag=I;

%Coefficient de corrélation
p=0.9;

%Moyenne et écart-type des composantes de la loi normale
mu=[4 4];
sigma= [1 p; p 1];

%Multiple du vecteur d'échantillonnage(doit être un nombre rationnel)
```

```
pas=0.01;

%Longueur maximale du vecteur d'échantillonnage(doit être pair)
longueur=1;

%Tableau contenant les probabilitées le long du vecteur
Probabilite=zeros((1/pas)*longueur,4);

%Nombre de points dans l'ensemble de référence
m=10;
%Ensemble contenant les points de références
Ensemble=zeros(m,2);

%Variable d'échantillonnage
theta=theta_0;

%Initialisation du vecteur des observations
X=zeros(t,2);

%Initialisation de l'ensemble de référence par l'étape Hit and run
for i=1 :m
    %Génère le vecteur déterminant l'échantillonnage.
    e1=normrnd(0,1);
    e2=normrnd(0,1);
    norme=sqrt(e1^2+e2^2);
    e1=e1/norme;
    e2=e2/norme;

    %Calcul des probabilités le long du vecteur e=[e1 e2]
    moitier=longueur/2;
    for j=1 :(1/pas)*moitier
        Probabilite(j,1)=theta(1,1)+(-j)*pas*e1;
        Probabilite(j,2)=theta(1,2)+(-j)*pas*e2;
        Probabilite(j,3)=mvnpdf([Probabilite(j,1) Probabilite(j,2)],mu,sigma);
        if Probabilite(j,3)<0.001
            Probabilite(j,3)=0;
        end
        Probabilite(j,4)=j;
    end
    for j=(1/pas)*moitier+1 :(1/pas)*longueur
        temp=j-((1/pas)*moitier);
        Probabilite(j,1)=theta(1,1)+temp*pas*e1;
        Probabilite(j,2)=theta(1,2)+temp*pas*e2;
        Probabilite(j,3)=mvnpdf([Probabilite(j,1) Probabilite(j,2)],mu,sigma);
        if Probabilite(j,3)<0.001
```

```
            Probabilite(j,3)=0;
        end
        Probabilite(j,4)=j;
    end
    if sum(Probabilite( :,3))>0
        indice= randsample(Probabilite( :,4),1,true,Probabilite( :,3));
        theta(1,1)=Probabilite(indice,1);
        theta(1,2)=Probabilite(indice,2);
    else
        theta(1,1)=theta(1,1);
        theta(1,2)=theta(1,2);
    end

    Ensemble(i,1)=theta(1,1);
    Ensemble(i,2)=theta(1,2);
end


%Période de chauffe
for i=1 :c
    %Génère le vecteur déterminant l'échantillonnage.
    point=randsample(m,2);

    e1=Ensemble(point(1,1),1)-Ensemble(point(2,1),1);
    e2=Ensemble(point(1,1),2)-Ensemble(point(2,1),2);
    norme=sqrt(e1^2+e2^2);
    e1=e1/norme;
    e2=e2/norme;

    %Calcul des probabilités le long du vecteur e=[e1 e2]
    moitier=longueur/2;
    for j=1 :(1/pas)*moitier
        Probabilite(j,1)=theta(1,1)+(-j)*pas*e1;
        Probabilite(j,2)=theta(1,2)+(-j)*pas*e2;
        Probabilite(j,3)=mvnpdf([Probabilite(j,1) Probabilite(j,2)],mu,sigma);
        if Probabilite(j,3)<0.001
            Probabilite(j,3)=0;
        end
        Probabilite(j,4)=j;
    end
    for j=(1/pas)*moitier+1 :(1/pas)*longueur
        temp=j-((1/pas)*moitier);
        Probabilite(j,1)=theta(1,1)+temp*pas*e1;
        Probabilite(j,2)=theta(1,2)+temp*pas*e2;
        Probabilite(j,3)=mvnpdf([Probabilite(j,1) Probabilite(j,2)],mu,sigma);
```

```
        if Probabilite(j,3)<0.001
            Probabilite(j,3)=0;
        end
        Probabilite(j,4)=j;
    end
    if sum(Probabilite( :,3))>0
        indice= randsample(Probabilite( :,4),1,true,Probabilite( :,3));
        theta(1,1)=Probabilite(indice,1);
        theta(1,2)=Probabilite(indice,2);
    else
        theta(1,1)=theta(1,1);
        theta(1,2)=theta(1,2);
    end
    Ensemble(point(1,1),1)=theta(1,1);
    Ensemble(point(1,1),2)=theta(1,2);
end

i=1;
while i<t
    for a=1 :lag
        %Génère le vecteur déterminant l'échantillonnage.
        point=randsample(m,2);

        e1=Ensemble(point(1,1),1)-Ensemble(point(2,1),1);
        e2=Ensemble(point(1,1),2)-Ensemble(point(2,1),2);
        norme=sqrt(e1^2+e2^2);
        e1=e1/norme;
        e2=e2/norme;

        %Calcul des probabilités le long du vecteur e=[e1 e2]
        for j=1 :(1/pas)*moitier
            Probabilite(j,1)=theta(1,1)+(-j)*pas*e1;
            Probabilite(j,2)=theta(1,2)+(-j)*pas*e2;
            Probabilite(j,3)=mvnpdf([Probabilite(j,1) Probabilite(j,2)],mu,sigma);
            if Probabilite(j,3)<0.00001
                Probabilite(j,3)=0;
            end
            Probabilite(j,4)=j;
        end
        for j=(1/pas)*moitier+1 :(1/pas)*longueur
            temp=j-((1/pas)*moitier);
            Probabilite(j,1)=theta(1,1)+temp*pas*e1;
            Probabilite(j,2)=theta(1,2)+temp*pas*e2;
            Probabilite(j,3)=mvnpdf([Probabilite(j,1) Probabilite(j,2)],mu,sigma);
            if Probabilite(j,3)<0.00001
```

```
            Probabilite(j,3)=0;
        end
        Probabilite(j,4)=j;
    end

    if sum(Probabilite( :,3))>0
        indice= randsample(Probabilite( :,4),1,true,Probabilite( :,3));
        theta(1,1)=Probabilite(indice,1);
        theta(1,2)=Probabilite(indice,2);
    else
        theta(1,1)=theta(1,1);
        theta(1,2)=theta(1,2);
    end
    Ensemble(point(1,1),1)=theta(1,1);
    Ensemble(point(1,1),2)=theta(1,2);
end

X(i,1)=theta(1,1);
X(i,2)=theta(1,2);
i=i+1;
end

end
```

B.11 Fonction AMC

```
function X=AMS(C,I,T)

%Graine de l'algorithme
theta_0=[4 4];

%Nombre de points dans l'ensemble de référence
m=10;

%Taille de l'échantillon
t=T;

%Longueur de la période de chauffe
c=C;

%Ventillation de l'échantillon
```

```
lag=I;

%Variance de la fonction instrumentale
s=[1 0;0 1];

%Covariance des composantes
p=0.9;

%Moyenne et écart-type des composantes de la loi normale
mu=[4 4];
sigma= [1 p; p 1];

%Ensemble contenant les points de références
Ensemble=zeros(m,3);

%variable
theta=theta_0;

%Initialisation du vecteur des observations
X=zeros(t,2);

%Initialisation de l'ensemble de référence
for i=1 :m

    g=mvnrnd(theta,s); %fonction instrumentale

    x1=mvnpdf(theta,mu,sigma);
    x2=mvnpdf(g,mu,sigma);

    %Critère d'acceptation
    a=min(1,(x2/x1)*(mvnpdf(g,theta,s)/mvnpdf(theta,g,s)));
    u=rand();
    if u<=a
        theta(1,1)=g(1,1);
        theta(1,2)=g(1,2);
    end
    Ensemble(i,1)=theta(1,1);
    Ensemble(i,2)=theta(1,2);
end

for i=1 :m
    Ensemble(i,3)= mvnpdf([Ensemble(i,1) Ensemble(i,2)],mu,sigma);
end

%Période de chauffe
```

```
for i=1 :c
    moyenne= [mean(Ensemble( :,1)) mean(Ensemble( :,2))];
    g=mvnrnd(moyenne,s,m); %fonction instrumentale

    x1=1;
    x2=1;
    for j=1 :m
        x1=x1*mvnpdf(g(j, :),mu,sigma);
        x2=x2*mvnpdf([Ensemble(j,1) Ensemble(j,2)],mu,sigma);
    end


    %Critère d'acceptation
    a=min(1,(x1/x2));
    u=rand();
    if u<=a
        Ensemble( :,1)=g( :,1);
        Ensemble( :,2)=g( :,2);
    end

    for j=1 :m
        Ensemble(j,3)= mvnpdf([Ensemble(j,1) Ensemble(j,2)],mu,sigma);
    end

end

%Étape MH
i=1;
while i<t
    test=false;
    for j=1 :lag
        moyenne= [mean(Ensemble( :,1)) mean(Ensemble( :,2))];
        g=mvnrnd(moyenne,s,m); %fonction instrumentale

        x1=1;
        x2=1;
        for k=1 :m
            x1=x1*mvnpdf(g(j, :),mu,sigma);
            x2=x2*mvnpdf([Ensemble(j,1) Ensemble(j,2)],mu,sigma);
        end


        %Critère d'acceptation
        a=min(1,(x1/x2));
        u=rand();
```

```
        if u<=a
            Ensemble( :,1)=g( :,1) ;
            Ensemble( :,2)=g( :,2) ;
            test=true ;
        end

        for k=1 :m
            Ensemble(k,3)= mvnpdf([Ensemble(k,1) Ensemble(k,2)],mu,sigma) ;
        end
    end
    if test
        for k=1 :m
            X((i-1)*m+k,1)=Ensemble(k,1) ;
            X((i-1)*m+k,2)=Ensemble(k,2) ;
        end
        i=i+1 ;
    end

end
```

B.12 Fonction test kolmogorov2d

```
    function [T frequencexy]=Test_Kolmogorov2d(X0,D,Nombre_intervalle)
% Fonction calculant la valeur de la statistique du test de
% Kolmogorov-Smirnov pour une mixture de deux lois normales bivariées et un
% échantillon donnée. La fonction prend comme argument la série de données
% à tester, l'indice de la distribution de référence et le nombre d'intervalle
% à utiliser pour calculer les valeurs de la fonction empirique. La
% fonction retourne les fréquences, ainsi que la valeur du test.

%Construction des intervalles sur lesquels sont calculées les différentes
%fréquences
Borne_infx=min(X0( :,1)) ;
Borne_supx=max(X0( :,1)) ;

Borne_infy=min(X0( :,2)) ;
Borne_supy=max(X0( :,2)) ;

borne=zeros(Nombre_intervalle,2) ;

for i=1 :Nombre_intervalle
    borne(i,1)= Borne_infx+i*((Borne_supx-Borne_infx)/Nombre_intervalle) ;
```

```
    borne(i,2)= Borne_infy+i*((Borne_supy-Borne_infy)/Nombre_intervalle) ;
end

%Fonction empirique bivariée ; calcul du nombres des points dans chaque itervalle
frequencexy=zeros(Nombre_intervalle,Nombre_intervalle) ;
for i=1 :Nombre_intervalle
    for j=1 :Nombre_intervalle
        for k=1 :length(X0( :,1))
            if X0(k,1)<=borne(i,1)&&X0(k,2)<=borne(j,2)
                frequencexy(i,j)=frequencexy(i,j)+1 ;
            end
        end
    end
end

%Calcul des fréquences
frequencexy=frequencexy./length(X0) ;
FXY=frequencexy ;

if D==1
    stat=0 ;
    for i=1 :Nombre_intervalle
        for j=1 :Nombre_intervalle
            x=[borne(i,1) borne(j,2)] ;
            fx=0.4*mvnpdf(x,[4 4],[1 0.3 ; 0.3 1])+0.6*mvnpdf(x,[8 4],[2 0.4 ;0.4 2]) ;
            stat=max([(fx-FXY(i,j)) stat]) ;
        end
    end
end
T=stat ;
```

B.13 Fonction SIS2D

```
    function [E_SIS Erreur_estimation_SIS Variance_poids W] = SIS_2D(X,Y,N_Particules, N_Iterations)
% Implementation de l'échantillonnage préférentiel séquentiel. La fonction a comme
% argument les séries des états et des observations, ainsi que le nombre de particules
% et la durée de l'utilisation du filtre. La fonction retourne l'estimation des états à
% chaque itération, l'erreur quadratique moyenne des estimations,
% la variance des poids normalisés et les poids normalisés des particules.

%Paramètre du filtre
T=N_Iterations ;
N=N_Particules ;
```

```
%Données à estimer à chaque itérations
E_SIS=zeros(T,1);
Variance_poids=zeros(T,1);

%Simulation de la série des états
%[X Y]=HMM(T);

%Matrice des particules
P=zeros(N,1);
Part=zeros(N,1);

%Matrice des poids
poid_incr=zeros(N,T);

%Matrice des fonctions gamma et instrumentale incrémentielles
gamma_n=zeros(N,T);
qn=zeros(N,T);

%Fonction instrumentale est une loi normale de moyenne 0 et variance
sigma=4;

%Première itération
for i=1 :N
    P(i,1)= normrnd(Y(1,1),sigma);
    mu=mean(P( :,1));
    if mu<=0
        gamma_n(i,1)=normpdf(Y(1,1),P(i,1),1);
    else
        gamma_n(i,1)=normpdf(Y(1,1),P(i,1),1)*mu;
    end
    qn(i,1)=normpdf(P(i,1),Y(1,1),sigma);
    poid_incr(i,1)= gamma_n(i,1)/qn(i,1);
end

Variance_poids(1,1)=var(poid_incr( :,1)./sum(poid_incr( :,1)));

for i=2 :T
    for j=1 :N
        Part(j,1)= normrnd(P(j,1),sigma);
        gamma_n(j,i)=(normpdf(Y(i,1),Part(j,1),1)*normpdf(Part(j,1),P(j,1),1));
        qn(j,i)=normpdf(Part(j,1),P(j,1),sigma);
        poid_incr(j,i)=gamma_n(j,i)/(qn(j,i));
    end

    %Estimation de Xt par échantillonnage pondéré séquentiel
```

```
    total=sum(poid_incr( :,i)) ; %Calcul des poids normalisés
    W( :,i)=poid_incr( :,i)./total ;
    if W(j,i)>0
        temp=Part( :,1).*W( :,i) ;
    end
    E_SIS(i,1)=sum(temp) ;

    %Calcul de la variance des poids
    Variance_poids(i,1)=var(poid_incr( :,i)./sum(poid_incr( :,i))) ;

    %Préparation des matrices pour la nouvelle itération
    P=Part( :,1) ;

end
Erreur_estimation_SIS=(mean(X-E_SIS).^2) ;
```

B.14 Fonction SIR2D

```
    function [E_SIR Erreur_estimation_SIR Variance_poids W] = SIR_2D(X,Y,N_Particules, N_Iterations)
% Implementation du filtre particulaire. La fonction a comme argument les
% séries des états et des observations, ainsi que le nombre de particules
% et la durée de l'utilisation du filtre. La fonction retourne l'estimation
% des états à chaque itération, l'erreur quadratique moyenne des estimations,
% la variance des poids normalisés et les poids normalisés des particules.

%Paramètre du filtre
T=N_Iterations ;
N=N_Particules ;

%Données à estimer à chaque itérations
E_SIS=zeros(T,1) ;
E_SIR=zeros(T,1) ;
Variance_poids=zeros(T,1) ;
W=zeros(N,T) ;

%Matrice des particules
P=zeros(N,1) ;
Part=zeros(N,1) ;
re_part=zeros(N,1) ;

%Matrice des poids
poid_incr=zeros(N,T) ;
```

```
%Matrice des fonctions gamma et instrumentale incrémentielles
gamma_n=zeros(N,1);
qn=zeros(N,1);

%Fonction instrumentale est une loi normale de moyenne 0 et variance
sigma=2;

%Première itération
for i=1 :N
    P(i,1)= normrnd(Y(1,1),sigma);
    mu=mean(P( :,1));
    if mu<=0
        gamma_n(i,1)=normpdf(Y(1,1),P(i,1),1);
    else
        gamma_n(i,1)=normpdf(Y(1,1),P(i,1),1)*mu;
    end
    qn(i,1)=normpdf(P(i,1),Y(1,1),sigma);
    poid_incr(i,1)= gamma_n(i,1)/qn(i,1);
end
%Ré-échantillonnage
re_part( :,1)=randsample(P( :,1),N,true,poid_incr( :,1)); %Poids normalisés
P( :,1)=re_part( :,1);
Variance_poids(1,1)=var(re_part( :,1));
W( :,1)=poid_incr( :,1)./sum(poid_incr( :,1));

for i=2 :T
    for j=1 :N
        Part(j,1)= normrnd(P(j,1),sigma); %Particule avant ré-échantillonnage
        gamma_n(j,1)=(normpdf(Y(i,1),Part(j,1),1)*normpdf(Part(j,1),P(j,1),1));
        qn(j,1)=normpdf(Part(j,1),P(j,1),sigma);
        poid_incr(j,i)=gamma_n(j,1)/qn(j,1);
    end

    %Estimation de Xt par échantillonnage pondéré séquentiel
    total=sum(poid_incr( :,i)); %Calcul des poids normalisés
    W( :,i)=poid_incr( :,i)./total;
    temp=Part( :,1).*W( :,i);
    E_SIS(i,1)=sum(temp);

    %Estimation de Xt par échantillonnage pondéré séquentiel avec
    %ré-échantillonnage
    re_part( :,1)=randsample(Part( :,1),N,true,poid_incr( :,i)); %Ré-échantillonnage
    E_SIR(i,1)=mean(re_part( :,1));
```

```
    %Calcul de la variance des poids
    Variance_poids(i,1)=var(re_part( :,1)) ;

    %Préparation des matrices pour la nouvelle itération
    P=re_part( :,1) ;

end
%Erreur_estimation_SIS=mean((X-E_SIS).^2) ;
Erreur_estimation_SIR=mean((X-E_SIR).^2) ;
```

B.15 Fonction Adaptatif Sir2D

```
function [E_ASIR Erreur_estimation_ASIR Nombre_RS Variance_poids W] = Adapatif_SIR_2D(X,Y,N_Part
N_Iterations)
% Implementation du filtre particulaire adaptatif. La fonction a comme argument les
% séries des états et des observations, ainsi que le nombre de particules et la durée
% de l'utilisation du filtre. La fonction retourne l'estimation des états à
% chaque itération, l'erreur quadratique moyenne des estimations, le nombre
% de ré-échantillonnage appliqués, la variance des poids normalisés et les
% poids normalisés des particules.

%Paramètre du filtre
T=N_Iterations ;
N=N_Particules ;

%Données à estimer à chaque itérations
E_ASIR=zeros(T,1) ;
Variance_poids=zeros(T,1) ;

%Matrice des particules
P=zeros(N,1) ;
Part=zeros(N,1) ;
re_part=zeros(N,1) ;

%Matrice des poids
poid_incr=zeros(N,T) ;
W=zeros(N,T) ;

%Matrice des fonctions gamma et instrumentale incrémentielles
gamma_n=zeros(N,1) ;
qn=zeros(N,1) ;

%Fonction instrumentale est une loi normale de moyenne 0 et variance
```

```
sigma=2;

%COmpteur du nombre de ré-échantillonnage
Nombre_RS=0;

%Première itération
for i=1 :N
    P(i,1)= normrnd(Y(1,1),sigma);
    mu=mean(P( :,1));
    if mu<=0
        gamma_n(i,1)=normpdf(Y(1,1),P(i,1),1);
    else
        gamma_n(i,1)=normpdf(Y(1,1),P(i,1),1)*mu;
    end
    qn(i,1)=normpdf(P(i,1),Y(1,1),sigma);
    poid_incr(i,1)= gamma_n(i,1)/qn(i,1);
end

W( :,1)=poid_incr( :,1)./sum(poid_incr( :,1));

temp=poid_incr( :,i).^2;
ESS=1/sum(temp);

if ESS>=N/2
    E_ASIR(i,1)=P( :,1).'*W( :,i);
else
    re_part( :,1)=randsample(Part( :,1),N,true,poid_incr( :,i));
    Nombre_RS=Nombre_RS+1;
    E_ASIR(i,1)=mean(re_part( :,1));
    P( :,1)=re_part;
end

Variance_poids(1,1)=var(poid_incr( :,1));



for i=2 :T
    for j=1 :N
        Part(j,1)= normrnd(P(j,1),sigma); %Particule avant ré-échantillonnage
        gamma_n(j,1)=(normpdf(Y(i,1),Part(j,1),1)*normpdf(Part(j,1),P(j,1),1));
        qn(j,1)=normpdf(Part(j,1),P(j,1),sigma);
        poid_incr(j,i)=gamma_n(j,1)/qn(j,1);
    end

    %Calcul de la variance des poids
```

```
    W( :,i)=poid_incr( :,i)./sum(poid_incr( :,i));
    Variance_poids(i,1)=var(poid_incr( :,i));

    %Estimation de Xt par échantillonnage pondéré séquentiel avec
    %ré-échantillonnage
    temp=poid_incr( :,i).^2;
    ESS=1/sum(temp);

    if ESS>=N/2
        E_ASIR(i,1)=Part( :,1).'*W( :,i);
    else
        re_part( :,1)=randsample(Part( :,1),N,true,poid_incr( :,i));
        Nombre_RS=Nombre_RS+1;
        E_ASIR(i,1)=mean(re_part( :,1));
        Part( :,1)=re_part;
    end

    %Préparation des matrices pour la nouvelle itération
    P=Part( :,1);

end
Erreur_estimation_ASIR=mean((X-E_ASIR).^2);
```

B.16 Fonction AFP

```
function [variance_poids estimation Erreur1 estimation_reechantillonnage Erreur2 W]=AFP(X,Y,N_particu
N_Iteration)
% Implémentation du filtre particulaire auxiliaire. La fonction prend comme
% argument la série des état, la séries des observations, le nombre de
% particules et le nombre d'itérations sur lesques utiliser l'algorithme.
% La fonction retourne la variance des poids normalisés, l'estimation des états
% occupés par le processus faite avant ré-échantillonnage, l'erreur quadratique moyenne de
% cette estimation, l'estimation faite après le ré-échantillonnage, l'EQM de
% cette dernière et les poids des particules à chaque itération.


%Paramètres de la simulation
N=N_particule;
T=N_Iteration;


%Initialisation de l'algorithme
```

```
Part=zeros(N,T) ; %Matrice des particules temporaire
P=zeros(N,T) ; %Matrice des particules
mu=zeros(N,1) ; %Paramètre déterminant les indices
poid_mu=zeros(N,1) ; %Poids des paramètres
index=zeros(N,2) ; %Matrice des indices
estimation=zeros(T,1) ; %Matrice des estimations avant ré-échantillonnage
estimation_reechantillonnage=zeros(T,1) ;%Matrice des estimations après ré-échantillonnage
w=zeros(N,1) ; %Matrice des poids
W=zeros(N,T) ; %Matrice des poids normalisés
variance_poids=zeros(N,T) ; %Variance des poids normalisés

%Première itération
for i=1 :N
    mu(i,1)=normrnd(Y(1,1),1) ;
end
for i=1 :N
    poid_mu(i,1)=normpdf(mu(i,1),Y(1,1),1) ;
end
index( :,1)=randsample(mu,N,true,poid_mu) ;
for i=1 :N
    for j=1 :N
        if index(i,1)==mu(j,1)
            index(i,2)=j ;
        end
    end
end
for i=1 :N
    Part(i,1)=normrnd(index(i,1),1) ;
end
for i=1 :N
    w(i,1)=normpdf(Y(1,1),Part(i,1),1)/normpdf(Y(1,1),index(i,1),1) ;
end
W( :,1)=w( :,1)./sum(w( :,1)) ;
P( :,1)=Part( :,1) ;
estimation(1,1)=sum(P( :,1).*W( :,1)) ;
P( :,1)=randsample(Part( :,1),N,true,w) ;
estimation_reechantillonnage(1,1)=mean(P( :,1)) ;
variance_poids( :,1)=var(W( :,1)) ;
for t=2 :T
    for i=1 :N
        mu(i,1)=P(i,t-1) ;
    end
    for i=1 :N
        poid_mu(i,1)=normpdf(mu(i,1),Y(t,1),1) ;
```

```
    end
    index( :,1)=randsample(mu,N,true,poid_mu) ;
    for i=1 :N
        for j=1 :N
            if index(i,1)==mu(j,1)
                index(i,2)=j ;
            end
        end
    end
    for i=1 :N
        Part(i,t)=normrnd(P(index(i,2),t-1),1) ;
    end
    for i=1 :N
        w(i,1)=normpdf(Y(t,1),Part(i,t),1)/normpdf(Y(t,1),index(i,1),1) ;
    end
    W( :,t)=w./sum(w) ;
    P( :,t)=Part( :,t) ;
    estimation(t,1)=sum(P( :,t).*W( :,t)) ;
    P( :,t)=randsample(Part( :,t),N,true,w) ;
    estimation_reechantillonnage(t,1)=mean(P( :,t)) ;
    variance_poids( :,t)=var(W( :,t)) ;
end



Erreur1=mean((X(1 :T,1)-estimation).^2) ;
Erreur2=mean((X(1 :T,1)-estimation_reechantillonnage).^2) ;
```

B.17 Fonction PFUnscented

```
    function [variance estimation EQM]=PFUnscented(X,Y,N_Particules, N_Iterations)
% Implementation du filtre particulaire unscented. La fonction a comme argument les
% séries des états et des observations, ainsi que le nombre de particules et la durée
% de l'utilisation du filtre. La fonction retourne la variance des poids normalisés,
% l'estimation des états à chaque itération et r l'erreur quadratique moyenne
% de cette estimation. Cette méthode utilise les fonctions ukf_predict1 et
% ukf_update1 provenant de la suite EKF/UKF toolbox for Matlab 7.x disponible à l'adresse
    % http ://becs.aalto.fi/en/research/bayes/ekfukf/

    %Fonction utilisées par les méthodes ukf_predict1 et ukf_update1
    ident = @identite ;

    % États initaux du processus
    x_0 = .1 ;
```

```
P_0 = 1;

% Variance du bruit
u_n = 1;
v_n = 1;

% Valeurs initiales du filtre de Kalman unscented
M = x_0;
P = P_0;

%Paramètres de la simulation
X=X.';
Y=Y.';
t = N_Iterations;
n=N_Particules;

Particules=zeros(n,t); %Matrice des particules temporaire
Part=zeros(n,t); %Matrice des particules

covariance=zeros(n,t); %Matrices des covariances estimées par le filtres
cov=zeros(n,t); %Matrices des covariances utilisé à chaque itération

poids=zeros(n,t); %Matrice des poids
poids_normalise=zeros(n,t); %Matrice des poids normalisés
variance=zeros(t,1); %Matrice des variances des poids normalisés
estimation=zeros(t,1); %Matrice des estimations des états

for i=1 :n
    [M,P] = ukf_predict1(M,P,ident,u_n);
    [M,P] = ukf_update1(M,P,Y(1,i),ident,v_n);
    Particules(i,1)= normrnd(M,P);
    covariance(i,1)=P;
    poids(i,1)=(normpdf(Y(1,1),Particules(i,1),1)*normpdf(Particules(i,1),x_0,1))/...
    normpdf(Particules(i,1),M,P);
end

%Ré-échantillonnage
index=randsample((1 :n).',n,true,poids( :,1));
for i=1 :n
    Part(i, :)= Particules(index(i,1), :);
    cov(i, :)=covariance(index(i,1), :);
end
Particules=Part;
covariance=cov;
poids_normalise( :,1)=poids( :,1)./sum(poids( :,1));
```

```
variance(1,1)=var(poids_normalise( :,1)) ;

for k = 2 :t
    for i=1 :n
        [M,P] = ukf_predict1(Particules(i,k-1),covariance(i,k-1),ident,u_n) ;
        [M,P] = ukf_update1(M,P,Y(1,k),ident,v_n) ;

        Particules(i,k)= normrnd(M,P) ;
        covariance(i,k)=P ;
        poids(i,k)=(normpdf(Y(1,k),Particules(i,k),1)*normpdf(Particules(i,k),Particules(i,k-1),1))/...
        normpdf(Particules(i,k),M,P) ;
    end
    %Ré-échantillonnage
    index=randsample((1 :n).',n,true,poids( :,k)) ;
    for i=1 :n
        Part(i, :)= Particules(index(i,1), :) ;
        cov(i, :)=covariance(index(i,1), :) ;
    end
    Particules=Part ;
    covariance=cov ;
    poids_normalise( :,k)=poids( :,k)./sum(poids( :,k)) ;
    variance(k,1)=var(poids_normalise( :,k)) ;
end

%Estimation à l'aide des particules
for i=1 :t
    estimation(i,1)=mean(Particules( :,i)) ;
end
X=X.' ;

%Calcul de l'erreur quadratique moyenne
EQM = sum((X-estimation).^2)/t ;
```

B.18 Fonction PFMCMC

```
function [estimation MSE variance w]=PFMCMC(X,Y,N_Particules, N_Iterations)
% Implementation du filtre particulaire. La fonction a comme argument les
% séries des états et des observations, ainsi que le nombre de particules
% et la durée de l'utilisation du filtre. La fonction retourne l'estimation
% des états à chaque itération, l'erreur quadratique moyenne des estimations,
% la variance des poids normalisés et les poids des particules.

%Paramètre du filtre
T=N_Iterations ;
```

```
N=N_Particules;
estimation=zeros(T,1);

%Borne
L=5;

%Matrice des particules
P=zeros(N,T);
Part=zeros(N,T);

%Matrice des poids
w=zeros(N,T);
variance=zeros(T,1);

%Première itération
for i=1 :N
    P(i,1)= normrnd(Y(1,1),1);

    if mean(P( :,1))>0
        w(i,1)=(mean(P( :,1))*normpdf(Y(1,1),P(i,1),1))/(normpdf(P(i,1),Y(1,1),1));
    else
        w(i,1)=normpdf(Y(1,1),P(i,1),1)/(normpdf(P(i,1),Y(1,1),1));
    end
end

%Ré-échantillonnage
index=randsample((1 :N).',N,true,w( :,1));
for i=1 :N
    Part(i, :)= P(index(i,1), :);
end
P=Part;

variance(1,1)=var(w( :,1)./sum(w( :,1)));
%Itération 1<t<L
for j=2 :L-1
    for i=1 :N
        %temp=(normpdf(Y(j,1),Part(i,j-1),2));
        P(i,j)= normrnd(Y(j,1),1);
        w(i,j)=(normpdf(Y(j,1),P(i,j),1)*normpdf(P(i,j),Part(i,j-1),1))/...
        (normpdf(Y(j,1),P(i,j),1)*normpdf(P(i,j),Part(i,j-1),1)/normpdf(Y(j,1),Part(i,j-1),2));
    end

    variance(j,1)=var(w( :,j)./sum(w( :,j)));
    %Ré-échantillonnage
    index=randsample((1 :N).',N,true,w( :,j));
```

```
    for i=1 :N
        Part(i, :)= P(index(i,1), :) ;
    end

    %MCMC
    for i=1 :N
        x0=normrnd(Y(j,1),1) ;
        ProbAcceptation=(normpdf(x0,Y(j,1),1)*normpdf(x0,Part(i,j-1),1)*normpdf(Part(i,j),Part(i,j-
1),1))/...
        (normpdf(Part(i,j),Y(j,1),1)*normpdf(Part(i,j),Part(i,j-1),1)*normpdf(x0,Part(i,j),1)) ;

        CritereAccep=rand() ;
        if CritereAccep<=min(1,ProbAcceptation)
            Part(i,j)=x0 ;
        end
    end

    P=Part ;
end

%Itération L<=t<T
for j=L :T
    for i=1 :N
        P(i,j)= normrnd(Y(j,1),1) ;
        w(i,j)=(normpdf(Y(j,1),P(i,j),1)*normpdf(P(i,j),Part(i,j-1),1))/...
        (normpdf(Y(j,1),P(i,j),1)*normpdf(P(i,j),Part(i,j-1),1)/normpdf(Y(j,1),Part(i,j-1),2)) ;
    end

    variance(j,1)=var(w( :,j)./sum(w( :,j))) ;
    %Ré-échantillonnage
    index=randsample((1 :N).',N,true,w( :,j)) ;
    for i=1 :N
        Part(i, :)= P(index(i,1), :) ;
    end
    %MCMC
    for i=1 :N
        x0=normrnd(Y(j,1),1) ;
        ProbAcceptation=(normpdf(x0,Y(j,1),1)*normpdf(x0,Part(i,j-1),1)*normpdf(Part(i,j),Part(i,j-
1),1))/...
        (normpdf(Part(i,j),Y(j,1),1)*normpdf(Part(i,j),Part(i,j-1),1)*normpdf(x0,Part(i,j),1)) ;

        CritereAccep=rand() ;
        if CritereAccep<=min(1,ProbAcceptation)
            Part(i,j)=x0 ;
        end
```

```
    end

    P=Part;

end
for i=1 :T
    estimation(i,1)=mean(Part( :,i));
end

MSE=mean((X-estimation).^2);
```

Bibliographie

[1] A Doucet AM Johansen. A note on auxiliary particle filters. *Statistics and Probability Letters*, 2008.

[2] Galin L. Jones Andrew Gelman. *Handbook of Markov Chain Monte Carlo : Methods and Applications*. Chapman and Hall, 2011.

[3] Galin L. Jones Andrew Gelman. *Handbook of Markov Chain Monte Carlo : Methods and Applications*. Chapman and Hall, 2011.

[4] Stéphane Sénécal Arnaud Doucet, Mark Briers. Efficient block sampling strategies for sequential monte carlo. *Journal of Computational and Graphical Statistics*, pages 693–711, 2006.

[5] S. P. Brooks and A. Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7 :434–455, 1997.

[6] ZHE CHEN. Bayesian filtering : From kalman filters to particle filters, and beyond. *damas.ift.ulaval.ca*, 2003.

[7] Arnaud Doucet. Christophe Andrieu, Nando de Freitas. Sequential mcmc for bayesian model selection. *IEEE Signal Processing Workshop on Higher Order Statistics.*, 1999.

[8] Arnaud Doucet Dan Crisan. Convergence of sequential monte carlo methods. 2000.

[9] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10 :197–208, 2000. 10.1023/A :1008935410038.

[10] Dani Gamerman. *Stochastic simulation for Bayesian inference*. Chapman and Hall, 1997.

[11] W. R. Gilks. *Bayesian Statistics 4*. Oxford University Press, 1992.

[12] W. R. Gilks. *Derivative-free adaptive rejection sampling for Gibbs sampling*. Oxford University Press., 1992.

[13] Olle Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambrigde university press, 2002.

[14] Arnaud Doucet Neil Gordon, Nando de Freitas. *Sequential Monte Carlo in Practice*. Springer, 2001.

[15] Sujit Sahu. Tutorial lectures on mcmc. 2000.

[16] Ramon van Handel. Hidden markov models, lecture notes. *Princeton*, pages 1–71, 2008.

[17] David B. Wilson. How to couple from the past using a read-once source of randomness. *Random Structures and Algorithms*, 16 :85–113, 2000.