Université de Montréal

Étude de la combinaison de la technique quasi-Monte Carlo randomisé vectoriel avec l'échantillonnage exact

par

Charles Sanvido

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la faculté des études supérieures

en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)

en informatique

Décembre 2006

Université de Montréal

Faculté des études supérieures

Ce mémoire de maîtrise intitulé

Étude de la combinaison de la technique quasi-Monte Carlo randomisé vectoriel avec l'échantillonnage exact

présenté par

Charles Sanvido

a été évalué par un jury composé des personnes suivantes :

Présidente-rapporteuse: Nadia El-Mabrouk

Directeur de recherche: Pierre L'Écuyer

Membre du jury: Patrice Marcotte

Mémoire accepté le 7 mars 2007

Résumé

Ce mémoire porte sur la combinaison de deux algorithmes. Le premier permet de produire des échantillons tirés de la loi de probabilité stationnaire d'une chaîne de Markov ergodique et le second de réduire la variance d'estimateurs du coût moyen par étape d'une chaîne de Markov. On aimerait pouvoir combiner ces deux algorithmes afin de produire des estimateurs du coût moyen par étape sur horizon infini ayant une variance plus faible que ceux produits seulement avec l'algorithme d'échantillonnage.

Ce mémoire présente un aperçu de l'échantillonnage utilisant les chaînes de Markov ainsi que les détails concernant l'algorithme d'échantillonage Coupling From The Past (CFTP) [35]. On peut aussi voir des combinaisons avec des techniques de réduction de la variance proposées dans la littérature.

La plus grande contribution de ce mémoire est de proposer de nouveaux algorithmes combinant les deux algorithmes CFTP et Quasi-Monte Carlo Randomisé Vectoriel (Array-RQMC) [21]. On propose quatre combinaisons différentes et on compare leurs performances à l'aide d'exemples numériques. On peut voir des exemples numériques où la performance est améliorée par un facteur de plusieurs milliers.

Mot-clés : Chaîne de Markov, réduction de la variance, simulation, loi de probabilité stationnaire, quasi-Monte Carlo randomisé, couplage.

Summary

This thesis relates to the combination of two algorithms. The first one makes it possible to produce samples drawn from the stationary probability law of an ergodic Markov chain and the second to reduce the variance of estimators of the average cost by step of a Markov chain. We would like to be able to combine these two algorithms in order to produce an estimator of the average cost by step on an infinite horizon having a lower variance than those only produced with the sampling algorithm.

This thesis presents an outline of sampling using the Markov chains as well as the details relating to the sampling algorithm Coupling From The Past (CFTP) [35]. We can also see combinations with reduction variance techniques proposed in the literature.

The greatest contribution of this thesis is to propose new algorithms combining two algorithms CFTP and Array Randomized Quasi-Monte Carlo (Array-RQMC) [21]. Four different combinations are proposed and we compare their performances using numerical examples. We can see numerical examples where the performance is improved by a factor of several thousands.

Key-words : Markov chains, variance reduction, simulation, stationnary law, randomized quasi-Monte Carlo, coupling.

Table des matières

Table des figures

Liste des tableaux

Liste des algorithmes

Liste des sigles et abréviations

Ω	Espace de probabilité
$\mathcal{P}$	Mesure de probabilité
E	Espérance mathématique
μ	Valeur de l'espérance mathématique
d	Dimension de l'espace Ω
$\mathbb{R}$	Espace des nombres réels
$\mathbf{U}$	Vecteur aléatoire suivant la loi Uniforme$(0,1)^d$
π	Loi de probabilité stationnaire
β	Biais
σ^2	Variance
n	Nombre de valeurs dans un échantillon
σ_n^2	Variance calculée avec un ensemble de n valeurs
X_t	État de la chaîne de Markov au moment t
$\mathcal{S}$	Espace d'états de la chaîne de Markov
m	Taille de l'espace d'états $\mathcal{S}$
μ_∞	Coût moyen par étape sur horizon infini
$\bar{\mu}_\infty^n$	Estimateur Monte Carlo de μ_∞ calculé avec n valeurs
$\bar{\sigma}_\infty^2$	Estimateur de la variance de $\bar{\mu}_\infty^n$
P^n	Ensemble de n points
$V_{(i)}$	Point i de P^n
U_i^t	Coordonnée t du point i
$\widehat{\mu}$	Estimateur quasi-Monte Carlo randomisé de μ_∞

$\widehat{\sigma}^2$	Estimateur de la variance de $\widehat{\mu}$
r	Nombre de macro-réplications des algorithmes
MSE	Erreur Quadratique Moyenne
Eff	Efficacité d'un estimateur
VRF	Facteur de Réduction de la Variance
MC	Monte Carlo
QMC	Quasi-Monte Carlo
RQMC	Quasi-Monte Carlo Randomisé
Array-RQMC	Quasi-Monte Carlo Randomisé Vectoriel
CFTP	Coupling From The Past
ROCFTP	Read Once Coupling From The Past

Chapitre 1

Introduction

1.1 L'estimation comme méthode de calcul

Avant d'entreprendre l'étude la combinaison de la technique quasi-Monte Carlo randomisé et de l'échantillonnage exact, on doit définir quelques notions au préalable. D'une part on veut pouvoir tirer des valeurs selon une loi de probabilité stationnaire des états d'une chaîne de Markov et de l'autre on veut réduire la variance des estimateurs produits à partir de ces valeurs. Commençons tout d'abord par le cadre général de l'estimation.

Supposons que l'on a une variable aléatoire Y définie sur un espace de probabilité $(\Omega, \mathcal{P})$ et que l'on s'intéresse à calculer l'espérance mathématique μ de Y,

$$\mu = E[Y] = \int_{\Omega} Y d\mathcal{P}.$$

L'espace de probabilité peut être défini de bien des façons et le calcul algébrique de la valeur de μ peut être très difficile à effectuer. Même si on échoue à trouver ls valeur exacte de μ, on peut être intéressé à en estimer la valeur. On appelle estimation le calcul approximatif d'un résultat qui est utilisable même si les données d'entrées sont incomplètes, incertaines ou bruitées. À partir de maintenant, l'ensemble d'événements sera $\Omega = [0,1)^d$, l'hypercube en d dimensions et on suppose que la variable Y peut être évaluée par une fonction $f : [0,1)^d \to \mathbb{R}$ construite selon la mesure de probabilité $\mathcal{P}$. L'espérance de Y peut se réécrire comme étant

$$E[Y] = E[f(\mathbf{U})] = \int_{[0,1)^d} f(\mathbf{u}) d\mathbf{u} \quad = \quad \int_0^1 \cdots \int_0^1 f(u_1, \ldots, u_d) du_1 \ldots du_d \quad (1.1)$$

où $\mathbf{U}\sim\text{Uniforme}(0,1)^d$. Pour couvrir tous les cas possibles, on laisse beaucoup de flexibilité au niveau de la complexité de f et de la valeur de d. Typiquement, on considérera que d est non borné. Pour faire ces estimations on peut avoir recourt, par exemple, à l'intégration Monte Carlo.

1.1.1 Intégration Monte Carlo

Les méthodes de Monte Carlo sont une classe largement répandue d'algorithmes pour simuler le comportement de divers systèmes physiques et mathématiques. Elles se distinguent d'autres méthodes de simulation (tel que la dynamique moléculaire) en étant stochastiques, habituellement en employant des nombres aléatoires (ou plus souvent des nombres pseudo-aléatoires).

Le but de la plupart des simulations stochastiques est d'estimer l'espérance mathématique d'une fonction, de la forme de l'équation (1.1). On parlera alors d'intégration Monte Carlo. Aussi longtemps que la fonction en question $f(\cdot)$ à intégrer est raisonnablement facile à calculer, on peut estimer la valeur de l'intégrale en choisissant aléatoirement des points dans l'espace à d dimensions et en prenant la moyenne des valeurs de la fonction en ces points. Pour un estimateur basé sur un échantillon de n valeurs $\{Y_0, Y_1, \ldots, Y_{n-1}\}$ évaluées sur les points indépendants et identiquement distribués $\{U_0, U_1, \ldots, U_{n-1} | U_i \in [0,1)^d$ pour tout $i = 0, \ldots, n-1\}$, tel que $Y_i = f(U_i)$ pour tout $i = 0, 1, \ldots n-1$, on obtient la valeur suivante

$$\bar{Y} = \frac{1}{n}\sum_{i=0}^{n-1} Y_i = \frac{1}{n}\sum_{i=0}^{n-1} f(U_i) \tag{1.2}$$

comme estimateur de μ. L'estimateur est sans biais

$$E[\bar{Y}] = E[Y_i] = E[f(U_i)] = \int_{[0,1)^d} f(\mathbf{u})d\mathbf{u} = \mu.$$

Pour pouvoir faire un tel calcul avec un ordinateur, on aura recourt à un générateur de nombres aléatoires. Ces nombres générés au hasard permettront, sous certaines conditions, de pouvoir échantillonner au hasard l'espace d'événements du modèle. Il existe une multitude de générateurs de nombres aléatoires. Ceux utilisés durant les expériences de ce mémoire sont implantés dans Stochastic Simulation in Java (SSJ) [18].

1.2 Échantillonnage et chaîne de Markov

Dans ce mémoire, on s'intéressera aux problèmes liés à certaines chaînes de Markov. On considère le problème de l'approximation de la valeur de l'espérance d'une fonction de coût par unité de temps pour les états d'une chaîne de Markov évoluant sur horizon infini. Pour cela, on ne peut pas simplement simuler la chaîne pendant très longtemps et ensuite estimer la valeur de l'espérance d'après ces valeurs prises très loin dans le futur de l'évolution de la chaîne car on produit des estimateurs biaisés.

Si la chaîne possède une loi de probabilité stationnaire π, on peut alors simuler la chaîne jusqu'au moment où les valeurs qu'elle prend sont générées par π pour ensuite calculer un estimateur selon l'équation (1.2). Il faut tout d'abord être capable de produire des valeurs selon π, moment où la chaîne de Markov est à l'état stationnaire. Pour ce faire, on aura recourt à des algorithmes d'échantillonnage parfait.

1.2.1 Échantillonnage parfait

L'échantillonnage parfait est une classe d'algorithmes permettant de générer des échantillons non biaisés suivant la loi de probabilité stationnaire d'une chaîne de Markov. Cette classe regroupe une foule d'algorithmes ; un résumé des travaux sur le sujet est disponible sur le site internet de David Wilson (`http://dbwilson.com/exact/`). Dans le cadre de ce travail, la technique d'échantillonnage étudiée est celle développée par Propp et Wilson [35]. Leur méthode appelée *Coupling From The Past* ou CFTP, permet d'éviter le biais introduit par la simulation d'une chaîne sur un horizon de temps très grand, mais fixe.

La méthode est basée sur la notion de couplage [25, 38]. On se sert du couplage pour entre autre étudier, de manière probabiliste, la convergence à l'équilibre de processus Markovien. Propp et Wilson [35] ont proposé une manière simple de coupler des chaînes de Markov de manière à produire, lorsque l'on échantillonne le processus selon une méthode de Monte Carlo, des valeurs tirées selon la loi de probabilité stationnaire d'une chaîne de Markov ergodique. Au lieu de simuler des chaînes couplées du présent vers le futur, ils le font à partir d'un moment éloigné dans le passé jusqu'au temps présent,

où la distance dans le passé est déterminée par l'algorithme lui-même. L'algorithme CFTP débute à une certaine étape dans le passé et évolue jusqu'au moment présent. S'il n'atteint pas le critère d'arrêt, une même valeur pour toutes les chaînes, il recommence plus loin dans le passé du processus. Au moment où il atteint le critère d'arrêt à l'instant présent, l'état du processus obéit alors à la loi de probabilité stationnaire de la chaîne avec probabilité 1 [35].

1.3 L'importance d'un bon échantillonnage

La simulation par ordinateur n'est pas sans rappeler la collecte de données statistiques réelles, à la différence que le calcul de l'estimateur se fait à partir d'échantillons produits aléatoirement selon un modèle choisi. Comme on ne fait qu'estimer la valeur, il y a donc une différence par rapport à la valeur réelle que l'on tente d'approximer. Une mesure de l'erreur faite par rapport à la vraie valeur recherchée peut s'exprimer par l'erreur quadratique moyenne (MSE) de notre estimateur $\bar{Y}$ défini par :

$$\mathrm{MSE}[\bar{Y}] = \beta^2 + \sigma^2$$

où β représente le biais et σ^2 la variance de $\bar{Y}$. Pour diminuer l'imprécision, on veut minimiser, voir annuler cette erreur lors de la recherche d'un estimateur. Dans le contexte qui nous intéresse, si les échantillons ne sont pas produits selon la vraie répartition, on induit alors un biais dans notre estimateur.

Le biais est une distorsion des propriétés d'un échantillon par rapport à ce qui est attendu, c'est-à-dire qu'il n'est pas tiré selon les lois que nous voulons échantillonner. On ne peut ni savoir ni estimer la vraie valeur de ce biais. Comme on ne sait rien sur celui-ci, même avec un très grand budget de simulation, on produit une mauvaise estimation de ce que l'on cherche. C'est pourquoi on recherche des méthodes pour produire des résultats non biaisés.

La deuxième partie de l'erreur quadratique moyenne est la variance. C'est une mesure arbitraire servant à caractériser la dispersion des échantillons recueillis. La variance de

l'estimateur est

$$\sigma_n^2 = \frac{1}{n^2}\sum_{j=0}^{n-1}\mathrm{Var}[Y_j] + \frac{2}{n^2}\sum_{i=0}^{n-1}\sum_{j=0}^{i-1}\mathrm{Cov}(Y_i, Y_j) \tag{1.3}$$

où $\mathrm{Var}[Y_j]$ est la variance unitaire de la variable aléatoire Y_j et est inconnue. Mais, en considérant que les Y_i sont indépendants et identiquement distribués, alors toutes les covariances sont nulles et toutes les variances unitaires sont égales et peuvent s'approximer par la variance empirique [15]

$$\mathrm{Var}[Y_j] \approx \frac{1}{n-1}\sum_{i=0}^{n-1}(Y_i - \bar{Y})^2, \text{ pour tout } j = 0, 1, \ldots, n-1. \tag{1.4}$$

On peut alors approcher la valeur de σ^2 par :

$$\sigma_n^2 = \frac{\mathrm{Var}[Y_j]}{n} \approx \frac{1}{n(n-1)}\sum_{i=0}^{n-1}(Y_i - \bar{Y})^2. \tag{1.5}$$

La variance des estimateurs est généralement de l'ordre $O(n^{-1})$, pour ceux calculés à partir de la moyenne empirique tel que l'équation (1.2) pour approximer une valeur tel que μ. Si on réussit à produire un estimateur non biaisé, on peut alors conclure que $\mathrm{MSE}[\bar{Y}] \to 0$ quand $n \to \infty$.

1.4 Performance d'un estimateur

Comme il existe plus d'une façon de faire de l'échatillonnage on doit déterminer un critère pour discriminer différents estimateurs. La simulation Monte Carlo, dans le cas présent, évalue empiriquement l'intégrale d'une fonction sur n points dans un espace à d dimensions pour produire une solution approximative. Pour calculer notre estimateur $\bar{Y}$, il y a un coût de calcul, généralement le temps de calcul et on notera son espérance mathématique par $\mathrm{Coût}[\bar{Y}]$. On définit l'*efficacité* de cet estimateur, avec les mêmes notations que L'Écuyer [19], comme étant

$$\mathrm{Eff}[\bar{Y}] \quad = \quad \frac{1}{\mathrm{Coût}[\bar{Y}] \cdot \mathrm{MSE}[\bar{Y}]}. \tag{1.6}$$

C'est avec ce critère que l'on comparera les différents estimateurs que l'on retrouve dans ce mémoire. On dira que l'estimateur $\bar{Y}_1$ est plus efficace qu'un autre estimateur $\bar{Y}_2$ si $\mathrm{Eff}[\bar{Y}_1] > \mathrm{Eff}[\bar{Y}_2]$. Pour augmenter la performance de l'estimateur $\bar{Y}$, on cherchera à

trouver un autre estimateur ayant une efficacité plus grande. Dans le cadre de ce présent travail, l'efficacité des estimateurs sera augmentée en réduisant leur erreur quadratique moyenne. Pour ce faire, en considérant que nous voulons des estimateurs non biaisés, on utilisera des techniques de réduction de variance.

Ces techniques sont employées pour augmenter la précision des évaluations qui peuvent être obtenues pour un nombre donné d'itérations ou un temps de calcul fixé. Il existe plusieurs techniques de réduction de variance employées afin de rendre une simulation statistiquement efficace : nombres aléatoires communs [19], variables aléatoires antithétiques [10, 9], variables de contrôle [5, 16], importance sampling [13] et échantillonnage stratifié [19].

1.5 Combinaison de diverses techniques

Bien qu'il existe plusieurs techniques de réduction de la variance, celles présentées dans ce mémoire utilisent les variables aléatoires antithétiques. Différents auteurs [2, 24] ont proposé des méthodes de générations de points pour la simulation avec l'algorithme CFTP. Il s'agit, de manière grossière, de corréler négativement plusieurs variables aléatoires suivant la loi Uniforme(0,1) utilisées par l'algorithme pour faire évoluer les différents processus durant la simulation par ordinateur. On espère ainsi obtenir une dépendance négative entre les différentes réalisations d'une même chaîne afin que la variance de l'estimateur final soit plus faible en comparaison au cas où les réalisations sont indépendantes.

Craiu et Meng [2] proposent trois techniques de génération de points : la méthode de déplacement permuté, la copule normale et l'échantillonnage par hypercube latin itératif. Ils démontrent des bornes sur le facteur de réduction de la variance (VRF) pour certains cas et donnent des exemples numériques.

Lemieux et Sidorsky [24] proposent d'utiliser 3 autres techniques Quasi-Monte Carlo (QMC) : règle de Korobov, Korobov par polynômes et ensemble de Sobol. Ces méthodes permettent, sous certains critères, de produire des estimateurs plus efficaces que les méthodes proposées par Craiu et Meng [2].

1.6 Combinaison avec Array-RQMC

Plus récemment, L'Écuyer, Lécot et Tuffin [20] ont développé un algorithme, appelé quasi-Monte Carlo randomisé vectoriel (Array-RQMC), permettant de réduire la variance d'un estimateur lors de la simulation d'une chaîne de Markov. L'idée derrière l'algorithme est de faire une simulation de plusieurs réalisations d'une chaîne de Markov, en parallèle. À chaque étape, on trie les réalisations selon la valeur de leurs états. Le tri permet une meilleure association des uniformes aux réalisations de la chaîne pour chacune des transitions, dans l'optique de la réduction de la variance, en comparaison aux méthodes RQMC standards [20]. La transition d'une réalisation se fait par inversion de la fonction de répartition de la valeur de l'état appliqué à une variable aléatoire d'un ensemble de points RQMC.

Durant ce travail, différentes combinaisons de cette technique à celle d'échantillonnage CFTP de Propp et Wilson [35] seront proposées. Il existe des bornes théoriques, sous certaines hypothèses, sur la convergence de variance de la méthode Array-RQMC pour un estimateur obtenu à partir d'une simulation de chaînes de Markov. Les mêmes conclusions ne peuvent s'appliquer quand on combine la méthode Array-RQMC avec l'algorithme CFTP car le contexte de simulation est différent. Bien que l'on cherche encore à faire un échantillonnage à partir d'une chaîne de Markov, la manière de le faire est différente.

Nous proposerons plusieurs algorithmes permettant de combiner ces deux techniques dans le but d'obtenir des échantillons non biaisés tout en ayant une variance plus faible. À première vue, des similitudes semblent relier les deux algorithmes : simulation de chaînes de Markov, simulation de processus en parallèle. Cependant, le mariage des deux techniques ne se fait pas si facilement que cela. Les nouvelles combinaisons avec Array-RQMC impliquent d'autres manipulations sur les différents processus qui évoluent en parallèle. Par exemple, on effectue une association des uniformes après avoir trié les processus durant l'exécution de l'algorithme. Pour certaines combinaisons, il est difficile de déterminer un tri adéquat afin d'obtenir une bonne efficacité.

Le cas où l'espace d'états de la chaîne de Markov est fini et petit est traité différem-

ment au cas où il est très grand. On analysera les forces et faiblesses des différents algorithmes. Certains d'entre eux permettent d'obtenir, pour certains exemples, de très grands facteurs de réduction de la variance.

Pour tenter d'améliorer l'efficacité de nos nouvelles méthodes de génération d'échantillons, on explore des heuristiques pour augmenter la valeur du facteur de réduction de la variance. On verra, dans les différents exemples que nous proposons, que le facteur de réduction de la variance dépend de cette heuristique.

Bien que d'un point de vue empirique, ces diverses combinaisons semblent donner d'excellents résultats, il faut regarder l'efficacité de ceux-ci comme définie par l'équation (1.6) en tenant compte du temps d'exécution des algorithmes. L'implantation des différents algorithmes a habituellement un coût de calcul plus grand que celle de l'échantillonnage exact avec le CFTP. Il y a donc un compromis à faire entre l'augmentation du coût de calcul et l'augmentation du facteur de réduction de la variance. On verra des manières de trouver des implantations adéquates permettant d'obtenir de bons facteurs d'augmentation de l'efficacité des différents estimateurs obtenus.

1.7 Aperçu du mémoire

Dans le chapitre 2, des explications plus en profondeur sur ce qu'est une chaîne de Markov seront données, ainsi que la manière d'utiliser cette structure mathématique pour faire la simulation de systèmes. Le cas particulier de générer des valeurs selon la loi de probabilité stationnaire des états d'une chaîne de Markov sera abordé. On survolera différentes techniques déjà proposées dans la littérature en ce qui a trait à ce type de problème.

Il sera question, dans le chapitre 3, de diverses méthodes de réduction de la variance proposé par Craiu et Meng, puis Lemieux et Sidorsky [2, 24]. On verra la manière dont ils abordent le problème ainsi que la définition des ensembles de points qu'ils utilisent pour arriver à obtenir une réduction de la variance.

La dernière partie de ce présent travail est consacrée à la manière d'introduire effi-

cacement une dépendance négative entre les différentes variables aléatoires. Plus particulièrement, c'est au niveau de la génération et de l'association des nombres aléatoires que l'effort est fait.

Le chapitre 4 contient de nouvelles combinaisons de l'algorithme Array-RQMC [20] avec le CFTP [35]. On y traitera séparément du cas où l'espace d'états est petit et celui où il est grand (ou infini). On va définir des algorithmes permettant d'obtenir un estimateur non biaisé ayant une variable plus faible que l'estimateur Monte Carlo et même que l'estimateur obtenu par Craiu et Meng [3] ou même Lemieux et Sidorsky [24]. Ce chapitre contient aussi l'analyse d'une heuristique permettant d'augmenter encore plus le facteur de réduction de variance de l'estimateur résultant de la combinaison de Array-RQMC et de CFTP.

Dans le chapitre 5, on se concentrera sur l'application des algorithmes proposés dans le chapitre 4 sur certains exemples numériques. On comparera leurs efficacités à ceux proposés dans la littérature [3, 24]. On verra aussi, de manière empirique, comment augmenter l'efficacité de nos estimateurs et comment on peut obtenir une augmentation considérable de l'efficacité.

Chapitre 2

Simulation Monte Carlo pour les chaînes de Markov

2.1 Définitions préliminaires

Comme mentionné préalablement, la recherche effectuée porte sur des problèmes pouvant être résolus à l'aide de chaînes de Markov. Ce chapitre est dédié à l'introduction des notations utilisées dans ce mémoire ainsi que l'énoncé des hypothèses de base sur les chaînes de Markov utilisées. On définit comme état de la chaîne de Markov au moment t la variable aléatoire X_t. On définit l'espace d'états $\mathcal{S}$ comme étant un ensemble de valeurs tel que X_t prend des valeurs dans $\mathcal{S}$, pour tout t.

Hypothèse 2.1.1. *Les chaînes de Markov employées posséderont les propriétés suivantes :*

a) Un espace d'états fini $\mathcal{S} = \{0, 1, \ldots, m-1\}$.

b) Un paramètre de temps t entier, $t = 0, 1, \ldots$.

c) Des probabilités de transition stationnaires : *Les probabilités de transition sont identiques pour toutes valeurs de t et notées par $p_{ij} = P\{X_{t+1} = j | X_t = i\}$ $\forall$ $t = 0, 1, \ldots$*

d) Un ensemble initial de probabilités : *$P\{X_0 = i\}$ pour tout $i \in \mathcal{S}$ notées par le vecteur $\lambda = \{\lambda_0, \ldots, \lambda_{m-1}\}$, avec $\lambda_i = P\{X_0 = i\}$ et on doit avoir $\sum_{j=0}^{m-1} \lambda_j = 1$*

On peut énumérer exhaustivement les probabilités de transition p_{ij} pour tous $i, j \in \mathcal{S}$. Une manière facile de représenter les probabilités de transition p_{ij} est sous la forme de la matrice

$$P = \begin{pmatrix} p_{00} & \cdots & p_{0\ m-1} \\ \vdots & & \vdots \\ p_{m-1\ 0} & \cdots & p_{m-1\ m-1} \end{pmatrix}.$$

On définit $p_{ij}^{(t)} = P\{X_{s+t} = j | X_s = i\}$ la probabilité conditionnelle de passer de l'état i à l'état j en t étapes. Comme les probabilités sont stationnaires alors $p_{ij}^{(t)} = P\{X_t = j | X_0 = i\}$. L'ensemble des $p_{ij}^{(t)}$ est obtenu en élevant la matrice de transition P à la puissance t [31]

$$P^t = \begin{pmatrix} p_{00}^{(t)} & \cdots & p_{0\ m-1}^{(t)} \\ \vdots & & \vdots \\ p_{m-1\ 0}^{(t)} & \cdots & p_{m-1\ m-1}^{(t)} \end{pmatrix}. \tag{2.1}$$

2.1.1 Exemple simple

Le graphe suivant illustre les états et les probabilités de transition d'une chaîne de Markov $\{X_t\}, X_0 = 0$, ayant les propriétés précédemment énumérées,

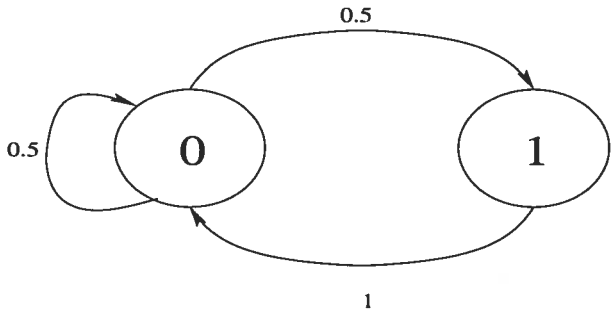

FIG. 2.1 – Exemple d'une chaîne de Markov en graphe

dont la suite des différents états $\{X_t\}$ s'exprime par

$$X_0 = 0, \qquad X_{t+1} = \begin{cases} X_t + a_t & \text{si } X_t = 0 \\ X_t - 1 & \text{si } X_t = 1 \end{cases}, a_t = \begin{cases} 0 & \text{avec probabilité } \frac{1}{2} \\ 1 & \text{avec probabilité } \frac{1}{2} \end{cases}.$$

L'exemple de la figure 2.1 est maintenant caractérisable de la manière suivante : l'espace d'états est $\mathcal{S} = \{0, 1\}$, le vecteur initial $\lambda = (1, 0)$ et les lois de probabilités de transition sont données par la matrice

$$P = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 1 & 0 \end{pmatrix}.$$

2.2 Problèmes d'intérêt

2.2.1 Coût moyen par étape

Supposons que l'exemple de la figure 2.1 modélise le fonctionnement d'un certain système et qu'il y ait un coût, $C : \mathcal{S} \mapsto \mathbb{R}$, associé à l'état dans lequel se trouve celui-ci. Un problème intéressant est de déterminer le coût moyen du système par unité de temps pour un nombre fini d'étapes. On calcule ce coût par la moyenne des valeurs des T premiers états de la chaîne

$$Y = \frac{1}{T}\sum_{t=1}^{T} C(X_t). \tag{2.2}$$

Pour ce système, le coût moyen espéré par unité de temps, μ_T, pour T unités s'exprime par :

$$\mu_T = E[Y] = E\left[\frac{1}{T}\sum_{t=1}^{T} C(X_t)\right]. \tag{2.3}$$

Pour calculer une telle espérance, on utilise les probabilités conditionnelles pour plus d'une étape (2.1). Il est alors possible de calculer la valeur exacte de μ_T de la manière suivante

$$\begin{aligned} \mu_T = E\left[\frac{1}{T}\sum_{t=1}^{T} C(X_t)\right] &= \frac{1}{T}\sum_{t=1}^{T}\sum_{j=0}^{m-1} P\{X_t = j\}C(X_j) \\ &= \frac{1}{T}\sum_{t=1}^{T}\sum_{j=0}^{m-1}\sum_{i=0}^{m-1} P\{X_0 = i\}P\{X_t = j|X_0 = i\}C(X_j). \end{aligned} \tag{2.4}$$

2.2.2 La loi de probabilité stationnaire

Une deuxième classe de problèmes que l'on retrouve dans la littérature est de déterminer la répartition des états d'une chaîne de Markov à l'équilibre. Il s'agit de déterminer la probabilité de se retrouver dans un certain état si la chaîne évolue pendant un très grand nombre d'étapes, et ce pour tous les états de $\mathcal{S}$. Pour cela, on laisse la valeur de t devenir très grande. Il s'agit de déterminer la limite suivante :

$$\lim_{t\to\infty} p_{ij}^{(t)} = \lim_{t\to\infty} P\{X_t = j|X_0 = i\}. \tag{2.5}$$

Même sous les hypothèses déjà posées on ne peut pas déterminer la limite (2.5) pour n'importe quelles chaînes de Markov. On restreint donc les chaînes considérées à celles apériodiques et irréductibles.

Définition 2.2.1. Une chaîne de Markov est dite apériodique si et seulement tous les états de la chaîne sont de période 1.

Définition 2.2.2. Une chaîne de Markov est dite irréductible si et seulement si pour tout couple $i, j \in \mathcal{S}$ il existe une valeur de t tel que $p_{ij}^{(t)} > 0$.

Hypothèse 2.2.1. *Les chaînes de Markov utilisées dans ce mémoire sont supposées apériodiques et irréductibles.*

Théorème 2.2.2. *[11] : Si une chaîne de Markov est apériodique et irréductible alors la limite (2.5) existe et elle est indépendante de la valeur de l'état initial i.*

On pose $\lim_{t\to\infty} p_{ij}^{(t)} = \pi_j$ pour tout $j \in \mathcal{S}$. L'ensemble des π_j forme un vecteur des probabilités stationnaires des états (à l'équilibre) $\pi = (\pi_0, \pi_1, \ldots, \pi_{m-1})$ s'il satisfait à [8] :

$$\begin{array}{rcll} \pi_j & \geq & 0, & \text{pour } j = 0, \ldots, m-1 \\ \pi P & = & \pi & \text{pour } j = 0, \ldots, m-1 \\ \sum_{j=0}^{m-1} \pi_j & = & 1. & \end{array}$$

La valeur de la limite, équation (2.5), est indépendante des probabilités à priori λ_i puisque

$$\begin{aligned} \lim_{t\to\infty} P\{X_t = j\} &= \lim_{t\to\infty} \sum_{i=0}^{m-1} \lambda_i p_{ij}^{(t)} = \sum_{i=0}^{m-1} \lambda_i \lim_{t\to\infty} p_{ij}^{(t)} \\ &= \sum_{i=0}^{m-1} \lambda_i \pi_j = \pi_j \underbrace{\sum_{i=0}^{m-1} \lambda_i}_{=1} = \pi_j. \end{aligned} \tag{2.6}$$

Le vecteur des probabilités stationnaires des états π d'une chaîne de Markov n'est pas sans lien avec le problème précédent du coût moyen par unité de temps. Il existe une relation lorsque le calcul de l'espérance est sur un horizon infini. Le calcul du coût moyen

espéré par étape sur horizon infini μ_∞ peut s'exprimer par la limite suivante :

$$\begin{aligned}
\mu_\infty = \lim_{T\to\infty} \mu_T &= \lim_{T\to\infty} \left\{ E\left[\frac{1}{T}\sum_{t=1}^{T} C(X_t)\right]\right\} \\
&= \lim_{T\to\infty} \left\{ \frac{1}{T}\sum_{t=1}^{T} E\left[C(X_t)\right]\right\} \qquad (2.7) \\
&= \lim_{T\to\infty} \left\{ \frac{1}{T}\sum_{t=1}^{T}\sum_{j=0}^{m-1} P\{X_t = j\}C(j)\right\} \\
&= \sum_{j=0}^{m-1} C(j) \lim_{T\to\infty} \left\{ \frac{1}{T}\sum_{t=1}^{T} P\{X_t = j\}\right\} \\
&= \sum_{j=0}^{m-1} \pi_j C(j). \qquad (2.8)
\end{aligned}$$

On n'a donc qu'à résoudre le système d'équations pour π avec la matrice de transition P et on peut calculer l'espérance de manière exacte avec l'équation (2.8). On peut alors calculer plus facilement cette espérance pour différentes fonctions de coût. Cette méthode fonctionne bien pour les chaînes ayant un petit nombre fini d'états. Il existe plusieurs logiciels permettant de résoudre le système d'équations pour π, par exemple *Cplex* [12]. La complexité du système d'équations augmente rapidement selon la dimension de la matrice et peut devenir rapidement difficile à résoudre même avec des logiciels spécialisés.

Si l'espace d'états $\mathcal{S}$ n'est pas dénombrable, on utilise un noyau de transition $K(s, B)$ qui donne la probabilité de se retrouver dans le sous-ensemble B de $\mathcal{S}$ si la chaîne est dans l'état s. Le vecteur π représentant la loi de probabilité à l'équilibre de la chaîne dans le cas où $\mathcal{S}$ est non-dénombrable devient une mesure qui satisfait la relation suivante

$$\int_{\mathcal{S}} K(s, B)d\pi(s) = \pi(B) \qquad \forall B \subset \mathcal{S} \qquad \text{et} \qquad \pi(S) = 1.$$

Il n'est plus toujours possible de résoudre facilement cette équation et ainsi d'appliquer la technique de calcul de l'espérance décrite par l'équation (2.8). On discutera plus précisément de ce cas plus loin dans la sous-section 2.3.4. Pour l'instant, on ne considère que le cas où l'espace d'états est fini.

2.3 La simulation Monte Carlo

2.3.1 Estimation sur horizon infini

Si on ne peut pas ou si on peut difficilement évaluer l'expression de l'équation (2.7), on peut être intéressé à tout le moins d'en approximer la valeur. Pour cela, on peut générer des échantillons produits par simulation par ordinateur afin de construire un estimateur du coût moyen par étape de la chaîne.

Définition 2.3.1. Une réalisation d'une chaîne de Markov est une suite de valeurs, où la première valeur x_0 est générée selon la loi de probabilité initiale et la valeur du $(t+1)^{ieme}$ état est générée en appliquant les lois de transition du t^{ieme} état.

Soit $x = \{X_0, X_1, \ldots, X_{T-1}\}$ une réalisation des T premiers états d'une chaîne de Markov où $X_t \in \mathcal{S}$ pour tout $t = 0, 1, \ldots, T-1$. On peut alors calculer

$$Y = \frac{1}{T} \sum_{t=0}^{T-1} C(X_t)$$

qui nous donne la moyenne des coûts par unité de temps pour la réalisation x. On répète ensuite l'expérience avec plusieurs réalisations, indépendamment les unes des autres, afin d'obtenir un ensemble de n valeurs $\{Y_0, \ldots Y_{n-1}\}$. En faisant la moyenne des résultats on obtient $\bar{\mu}_T^n$, l'estimateur de μ_∞

$$\bar{\mu}_T^n = \frac{1}{n} \sum_{i=0}^{n-1} Y_i.$$

On a bien sûr que

$$E[\bar{\mu}_T^n] = E[Y_i] = E\left[\frac{1}{T} \sum_{t=1}^{T} C(X_t)\right] = \mu_T.$$

On doit donc construire plusieurs réalisations indépendantes de la chaîne de Markov, chacune des réalisations correspondant à une observation possible de celle-ci. Pour cela, on aura recourt à la simulation par ordinateur à l'aide d'une fonction $\varphi_t : \mathcal{S} \times [0,1) \mapsto \mathcal{S}$ et la récurrence stochastique

$$X_0 = x, \qquad X_t = \varphi_t(X_{t-1}, U_t), \qquad t \geq 1.$$

La fonction φ_t prend en paramètre un élément dans $\mathcal{S}$, l'espace d'états de la chaîne de Markov, ainsi qu'une valeur dans l'ensemble $[0,1)$, et renvoie une valeur dans $\mathcal{S}$ que l'on

associera au prochain état de la réalisation à partir des probabilités de transition de la chaîne pour l'état à l'instant t. Avec une valeur de départ x_0 donnée et des éléments $u_1, \ldots, u_t$ générés au hasard selon la loi Uniforme(0,1), elle permettra de générer, par récurrence $x_{t+1} = \varphi_t(x_t, u_t), u_t \in [0,1)$, $x_t \in \mathcal{S}$, une suite de valeurs $x_0, x_1, \ldots$ correspondant à une réalisation.

Hypothèse 2.3.1. *φ_t est homogène par rapport au temps, $\varphi_t(i,u) = \varphi_{t'}(i,u)$ $\forall t, t' \in \mathbb{N}, i \in \mathcal{S}, u \in [0,1)$ et on la notera simplement par φ*

Comme la fonction de transition $\varphi(\cdot)$ prend en entrée une valeur dans l'intervalle [0,1) on pose l'hypothèse suivante :

Hypothèse 2.3.2. *On utilisera des générateurs de nombres aléatoires produisant des valeurs selon la loi Uniforme(0,1).*

Cette fonction n'est pas évaluée de n'importe quelle façon.

Hypothèse 2.3.3. *La fonction $\varphi : \mathcal{S} \times [0,1) \mapsto \mathcal{S}$ utilise la fonction de répartition inverse de x_{t+1} selon x_t en utilisant une seule uniforme pour générer le prochain état de la chaîne.*

Étant donné $X_t \in \mathcal{S}$, la fonction de répartition de X_{t+1} conditionnelle à X_t, $F_{X_t} : \mathcal{S} \mapsto [0,1)$, est définie par $F_{X_t}(x) = P\{X_{t+1} \leq x | X_t\}$. Pour une valeur $u \in [0,1)$ donné, alors la prochaine valeur est

$$\begin{aligned} X_{t+1} = F_{X_t}^{-1}(u) &= \min\{x | F_{X_t}(x) \geq u\} \\ &= \min\{x | \sum_{l=1}^{x} p_{x_t l} \geq u\} \stackrel{\text{def}}{=} \varphi(X_t, u) \end{aligned} \tag{2.9}$$

où $F_{X_t}^{-1}(\cdot)$ est l'inverse de la fonction de répartition de X_{t+1} conditionelle à X_t [19]. Donc, pour une suite donnée de valeurs $\{u_1, u_2, \ldots\}$, $u_i \in [0,1)$ et une valeur de départ x_0 on obtient une suite d'éléments qui correspond à une réalisation possible pour cette chaîne de Markov. Par exemple, pour le modèle de la figure 2.1, soit la suite d'uniformes $\{0.75, 0.83, 0.32, \ldots, 0.19, 0.65, 0.46, \ldots\}$ et $x_0 = 0$. On obtient la réalisation $\{x_0 = 0, x_1 = 1, x_2 = 0, x_3 = 0, \ldots, x_{T-2} = 0, x_{T-1} = 1, x_T = 0, \ldots\}$ que l'on peut représenter graphiquement comme à la figure 2.2 où l'abscisse indique le temps et l'or-

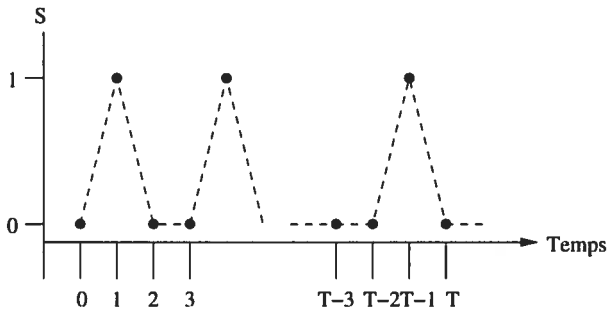

FIG. 2.2 – Exemple graphique d'évolution d'une chaîne de Markov

donnée l'espace d'états du processus. Cela se représente relativement bien lorsque T est fini, mais le problème qui nous intéresse à la base est d'estimer la valeur de l'équation (2.8), celle du coût moyen sur un horizon infini. On n'a pas à calculer le coût à chacune des étapes. Si on peut déterminer la fonction de répartition des états à l'équilibre, on pourrait ensuite échantillonner selon cette répartition pour calculer la valeur de notre estimateur.

Si le premier élément est généré selon la loi de probabilité stationnaire de la chaîne $X_0 \sim \pi$, alors $\bar{\mu}_T^n$ est un estimateur non biaisé de μ_∞ pour toute valeur de $T > 0$. On va alors estimer la valeur de μ_∞ par la moyenne des valeurs calculées sur un ensemble de réalisations possibles de la chaîne. Pour cela, on devra donc s'assurer que $X_0 \sim \pi$ pour les échantillons générés

2.3.2 Génération d'échantillons

Supposons que l'on tente d'estimer le coût moyen par étape sur horizon infini d'une certaine chaîne de Markov $\{X_t\}$ répondant aux hypothèses 2.1.1, 2.2.1, 2.3.1, 2.3.3. On doit générer des valeurs selon la loi de probabilité stationnaire π que l'on utilisera durant la simulation pour produire un ensemble de n valeurs afin de construire l'estimateur $\bar{\mu}_T^n$. De manière naïve, on peut initialiser la chaîne de Markov dans un certain état, selon une répartition à priori 2et on laisse le processus évoluer pendant un certain temps T selon l'algorithme 1. On considère ensuite la valeur du dernier état de la réalisation (X_T) comme une observation tirée approximativement selon la loi de probabilité stationnaire. Pour les transitions, à l'aide de la fonction $\varphi(s, U)$, $s \in \mathcal{S}$ et $U \sim$ Uniforme$(0, 1]$ définie dans la section 2.3, il faut générer aléatoirement des valeurs pour les différents u. On

ne discutera pas de la manière de le faire pour l'instant. On supposera seulement que l'on se sert d'une fonction Unif(), qui fait appel à un générateur de nombres aléatoires pour donner au hasard des valeurs aux variables u_t suivant la loi Uniforme(0,1].

Algorithme 1 Algorithme de simulation Monte Carlo d'une chaîne, T fini [8]

$t \leftarrow 0$ (instant de départ)

$x_t \leftarrow s \in \mathcal{S}$ un état généré selon une répartition à priori

répéter

$t \leftarrow t+1$

$u_t \leftarrow$Unif()

$x_t \leftarrow \varphi(x_{t-1}, u_t)$

jusqu'à $t = T$ (instant d'arrêt)

retourner la valeur de x_T

On obtient donc une valeur et on peut maintenant refaire l'algorithme 1 n fois indépendamment afin d'obtenir assez de données pour construire notre estimateur. Soit l'ensemble $\{X_T^0, \ldots, X_T^{n-1}\}$ un échantillon de n valeurs produites avec l'algorithme 1. On suppose ensuite que la valeur de μ_∞ s'approxime par

$$\bar{\mu}_\infty^n = \frac{1}{n}\sum_{i=0}^{n-1} C(X_T^i). \tag{2.10}$$

Ici, on ne considère que le dernier état de la réalisation, au temp T, dans le calcul de la valeur de notre estimateur $\bar{\mu}_\infty^n$. Il est possible de considérer d'autres manières de construire $\bar{\mu}_\infty^n$. Par exemple la moyenne des résultats sur un ensemble de T valeurs générées par une même réalisation pour approximer le corps de la limite dans l'équation (2.7). Cependant, pour approximer μ_∞ à l'aide de l'équation (2.8), il faut s'assurer de bien générer les valeurs selon la loi de probabilité stationnaire. En supposant qu'après T étapes les valeurs sont produites approximativement selon π, on doit répondre à plusieurs questions : comment choisir l'instant T où l'on doit s'arrêter ? Quel est l'effet de l'état initial choisi sur la valeur obtenue à la fin de la simulation ?

Dans l'exemple de la figure 2.2 la valeur au temps 0 n'est pas nécessairement générée selon la loi de probabilité stationnaire. Si elle ne l'est pas, après une évolution de T étapes, on ne se retrouve pas obligatoirement avec une valeur de la loi de probabilité stationnaire. En fait, si on arrête la simulation après un nombre T fixe et que l'on

échantillonne à cet instant, on se retrouve à échantillonner le vecteur de probabilités à l'étape T défini par λP^T. On sait, par l'équation (2.6), que

$$\lim_{T\to\infty} \lambda P^T = \pi \ .$$

Mais pour $T < \infty$, il y a une différence correspondant au biais $\beta \geq 0$. On obtient bien sûr que si une telle limite existe alors pour toute valeur de $\epsilon > 0$ il existe une valeur de T qui satisfait

$$||\lambda P^T - \pi|| \ \leq \ \epsilon \tag{2.11}$$

où la norme est définie comme la racine carrée de la somme des éléments au carré du vecteur. Mais il n'y a pas d'indication pour savoir la valeur à donner à T pour satisfaire cette relation. De manière générale, on ne peut pas dire que $X_T \sim \pi$ (où X_T est la valeur retournée par l'algorithme).

2.3.3 Le couplage

Une manière d'annuler l'effet de l'initialisation à priori est ce qu'on appelle le couplage. Pour plus de détails sur le couplage de chaînes de Markov, voir [25, 38]. On entend par couplage la construction conjointe de deux variables aléatoires (ou processus) ou plus, afin de déduire des propriétés des variables prises individuellement.

Définition 2.3.2. Une copie d'une variable aléatoire X est une variable aléatoire X' possédant la même loi de probabilité que X [25, 38]. On notera

$$X' \stackrel{loi}{=} X$$

pour indiquer que X' est une copie de X.

Définition 2.3.3. Un couplage d'une collection de variables aléatoires $\{X_t\}$ est une famille de variables aléatoires $\{X'_t\}$ telle que [38]

$$X'_t \stackrel{loi}{=} X_t, \text{ pour tout } t \ .$$

Notons ici que seulement les variables X'_t sont des copies des variables X_t prises deux à deux pour chaque t, mais la famille $\{X'_t\}$ n'est généralement pas une copie de la famille $\{X_t\}$.

Plus précisément, pour les chaînes de Markov couplées, on dirat que les chaînes $\{X_t\}, \{X'_t\}$ sont des copies, si pour tout couple (X_t, X'_t), X'_t est une copie de X_t. Chacune des variables X_t et X'_t est régie par la loi de la matrice de transition P, mais n'a pas nécessairement la même valeur initiale. Lors d'une simulation, la manière de coupler deux chaînes de Markov est d'utiliser des variables aléatoires communes. Supposons que l'on ait deux copies d'une même chaîne de Markov $\{X_t\}, \{X'_t\}$ et que l'on fait évoluer les deux chaînes avec des variables aléatoire communes $\{U_1, U_2, \ldots U_t, \ldots\}$ selon la fonction de récurrence

$$\begin{aligned} X_t &= \varphi(X_{t-1}, U_t), \\ X'_t &= \varphi(X'_{t-1}, U_t), \qquad X_0 \neq X'_0. \end{aligned}$$

On s'assure ainsi que les différentes copies de la chaîne auront les mêmes réalisations des variables servant à faire avancer la chaîne d'une étape et cela aux mêmes étapes. L'évolution des deux processus est en parallèle selon les mêmes uniformes U_t. Si à un instant T^* on obtient une même valeur pour les deux états $X_{T^*} = X'_{T^*}$ des deux réalisations, alors les deux copies partagent les mêmes valeurs pour la suite de leur évolution puisque l'on utilise des variables aléatoires communes. C'est à ce moment que l'on dit que les deux copies sont fusionnées et on dira que T^* est l'instant de fusion si $T^* = \min\{t | X_t = X'_t\}$. Pour les chaînes qui satisfont aux hypothèses 2.1.1 et 2.2.1, T^* est fini avec probabilité 1 [6]. Prises individuellement, chacune des copies agira exactement comme le processus original, mais quand les copies sont considérées conjointement, des informations supplémentaires peuvent en être déduites.

Supposons que l'on puisse définir une version stationnaire de la chaîne $\{\tilde{X}_t\}$ tel que $\tilde{X}_0 \sim \pi$. Si on couple $\{\tilde{X}_t\}$ avec une autre copie de la chaîne de Markov $\{X_t\}$ tel que l'état de départ X_0 n'est pas généré par la loi de π, alors à l'instant de couplage, la deuxième copie de la chaîne sera aussi régie par la loi de probabilité stationnaire [6, 25, 38]. L'idée est de trouver cet instant T^* où le processus est dans cette situation.

On peut regarder la simulation de plus de deux copies du processus. Soit $m = |\mathcal{S}|$ copies $\{X_{0,t}\}, \{X_{1,t}\}, \ldots, \{X_{m-1,t}\}$ de la même chaîne de Markov. À l'instant où toutes les copies seront fusionnées, les différentes valeurs qu'avaient les copies à l'initialisation n'auront pas d'importance. Peu importe la valeur d'initialisation, les m copies ont la

même valeur à l'instant de fusion. Alors, si on se trouve dans le cas où l'espace d'états $\mathcal{S}$ de la chaîne est fini, on peut donner à chacune des réalisations de ces chaînes un état initial différent, de manière à ce que tout l'espace d'états à l'initialisation est couvert. Dans ce cas, la valeur retournée à la fin du processus ne dépendra plus de l'état initial, peu importe la valeur de celui-ci, la valeur de fin est identique pour tous les états initiaux possibles.

Algorithme 2 Algorithme de simulation Monte Carlo avec couplage [35]

$t \leftarrow 0$

pour $i = 0, \ldots, m-1$ **faire**

$x_{i,t} \leftarrow i$, l'état i de $\mathcal{S}$ que l'on associe à la i^{ieme} copie de la chaîne

fin pour

répéter

$t \leftarrow t+1$

$u_t \leftarrow$Unif()

pour $i = 0 \ldots m-1$ **faire**

$x_{i,t} \leftarrow \varphi(x_{i,t-1}, u_t)$

fin pour

jusqu'à $x_{0,t} = x_{1,t} = \ldots = x_{m-1,t}$

retourner l'unique valeur $x = x_{0,t} = x_{1,t} = \ldots = x_{m-1,t}$

On construit ensuite $\bar{\mu}_\infty^n$ en prenant la moyenne des valeurs retournées par n répétitions indépendantes de l'algorithme 2. Cependant, si on arrête la simulation au premier instant où toutes les copies de la chaîne sont fusionnées, l'estimateur $\bar{\mu}_\infty^n$ restera biaisé. Si on se réfère à l'exemple de la figure 2.1, il existe une probabilité non nulle de se retrouver dans l'état (1), mais l'algorithme 2 ne retournera pas cet état comme valeur. Les valeurs retournées par simulation sont (0) avec probabilité 1 et l'état (1) avec proba- bilité 0, tandis que les vraies probabilités stationnaires sont (0) avec probabilité $\frac{2}{3}$ et (1) avec probabilité $\frac{1}{3}$ [14]. On ne peut donc pas conclure que la valeur x est générée selon la loi π à l'instant où les chaînes fusionnent.

Le graphique de la figure 2.3 illustre une évolution de chaîne de Markov selon l'algorithme 2 appliqué à l'exemple de la figure 2.1. Il y a deux réalisations de la chaîne

de Markov, l'une d'elle ayant comme état initial (0) et l'autre (1). Les deux réalisations ont comme instant de fusion T^* et partagent donc les mêmes valeurs pour la suite de leur évolution. On ne peut pas considérer la valeur des chaînes à l'instant T^* comme

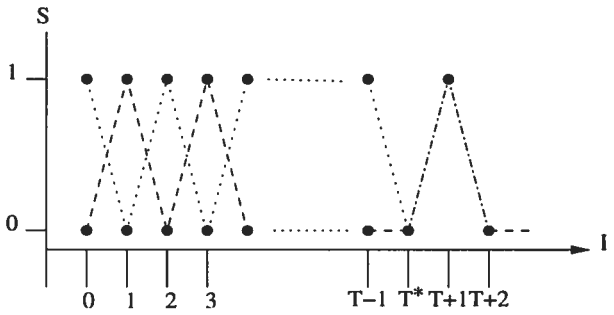

FIG. 2.3 – Exemple d'évolution avec couplage

étant générée selon la loi de probabilité stationnaire de la chaîne. On peut alors penser faire évoluer le processus pendant un certain temps $\delta \geq 0$ après que toutes les copies se soient fusionnées et échantillonner à $T^* + \delta$. On se retrouve malheureusement encore avec le même problème qu'auparavant, de déterminer la valeur de δ pour être assuré que la valeur générée soit bel et bien selon la vraie loi de probabilité stationnaire. C'est comme prendre l'algorithme 1 et d'utiliser les valeurs à l'instant de fusion plutôt que de les générer selon le vecteur d'initialisation à priori λ.

2.3.4 L'échantillonnage parfait

En 1996, Propp et Wilson [35] ont proposé une méthode qui permet, sans calcul préalable sur le temps de simulation, de générer des échantillons qui sont tirés selon la vraie loi de probabilité stationnaire d'une chaîne de Markov apériodique et irréductible. Leur méthode, *Coupling From The Past* (CFTP), est aussi basée sur le principe de couplage, mais pas comme nous l'avons déjà vu préalablement. Lorsque l'on fait la simulation selon l'algorithme 2, les différentes copies de la chaîne de Markov évoluent d'un temps initial 0 jusqu'à T. Pour calculer la valeur d'une copie à cet instant à partir du temps 0, il suffit de faire T itérations à partir d'un état initial $x_0 \in \mathcal{S}$ avec un ensemble de variables aléatoires uniformes indépendantes $\{U_t, t \in \mathbb{N}\}$ selon la récurrence $X_t = \varphi(X_{t-1}, U_t)$, $X_0 = x_0$. Cela nous donne une valeur d'état au temps t s'exprimant

comme :

$$X_t = \varphi(\varphi(...\varphi(\varphi(\varphi(X_0, U_1), U_2), U_3)..., U_{t-1}), U_t). \quad (2.12)$$

Pour l'instant, avec l'algorithme 2, le couplage de séquences comme $\{X_t\}$ selon l'équation (2.12) ne nous permet pas de trouver de temps d'arrêt adéquat. Pourtant, d'après Thorisson [38], s'il existe une loi de probabilité stationnaire pour notre processus, alors il existe un couplage ayant la même limite (loi stationnaire) et on atteint cette limite dans un temps fini. On construit maintenant la "*time-backward sequence*" de la chaîne [3]

$$\overleftarrow{X}_0 \stackrel{\text{def}}{=} \varphi(\varphi(...\varphi(\varphi(\overleftarrow{X}_{-t}, U_{-t}), U_{-t+1})..., U_{-2}), U_{-1}) \quad (2.13)$$

avec $\overleftarrow{X}_{-t} = X_0$. C'est avec cette séquence que les simulations sont effectuées. On trouve un temps $-t$ tel que pour toutes valeurs $\overleftarrow{X}_{-t} \in \mathcal{S}$ on obtient la même valeur de $\overleftarrow{X}_0$.

Théorème 2.3.4. *Le processus aura la même loi de probabilité stationnaire que la chaîne originale. Le processus ainsi créé suit la même loi π à l'équilibre, mais n'est pas markovien.*

Preuve : Pour démontrer que le processus n'est pas markovien, nous allons prendre le même exemple que Craiu et Meng [3]. Supposons que les réalisations évoluent selon la fonction de transition $\varphi(x, u) = u$ pour toute valeur de x dans $\mathcal{S}$, alors $X_t = U_t$, mais $\overleftarrow{X_0} = U_{-1}$ pour tout $t \geq 1$. Les deux processus 2.12 et 2.13 convergent en distribution vers la loi Uniforme(0,1]. Mais on ne peut, à priori, savoir que l'on a atteint la limite pour n'importe quelle valeur de t même si l'état au temps t=1 suivait déjà la bonne loi. Au contraire, la valeur de $\overleftarrow{X}_0$ est toujours égale à u_1 pour tout $t \geq 1$. Comme Foss et Tweedie [6] l'ont démontré, la loi de $\overleftarrow{X}_0$ converge vers la même limite que celle de X_t avec probabilité 1. Mais comment être sûr que le processus selon l'équation (2.13) possède la même fonction de répartition des états à l'état stationnaire de la chaîne ? Pour le processus markovien $\{X_t\}$, si on fait t étapes pour toutes les chaînes alors

$$\begin{aligned} P\{X_t = j | X_0\} &= E[I[\varphi(\varphi(...\varphi(\varphi(X_0, U_1), U_{2)})..., U_{t-1}), U_t) = j]] \\ &= E[I[\varphi(\varphi(...\varphi(\varphi(\overleftarrow{X}_{-t}, U_{-t}), U_{-t+1)})..., U_{-2}), U_{-1}) = j]] \\ &= P\{\overleftarrow{X}_0 = j | \overleftarrow{X}_{-t}\} \end{aligned}$$

car a priori les différentes variables aléatoire U_i sont indépendantes identiquement distribuées. Nous obtenons alors que les deux processus convergent bien vers la même limite $\lim_{t\to\infty} P\{\overleftarrow{X}_0 = j|\overleftarrow{X}_{-t}\} = \lim_{t\to\infty} P\{X_t = j|X_0\} = \pi_j$, le j^{ieme} élément du vecteur π. Cela signifie que la probabilité d'être dans un certain état après t étapes est la même pour les deux processus. ■

Mais la question demeure la même, comment trouver le temps d'arrêt T permettant de générer des valeurs selon la loi de probabilité stationnaire de la chaîne ? Les algorithmes et processus que nous avons utilisés jusqu'à maintenant ne permettaient pas d'obtenir un tel temps d'arrêt. Avec la nouvelle séquence de valeurs générées à partir de l'équation (2.13), la différence fondamentale se situe au niveau des réalisations des chaînes. Si les réalisations de la séquence (2.13) ont toutes fusionné à un instant, on conclut alors que le processus à atteint la limite et que le résultat retourné est bien tiré selon la loi de π.

2.3.5 Méta-chaîne

On veut donc simuler m copies de la chaîne en parallèle selon l'équation (2.13), chacune avec une valeur d'état initial différente $0, 1, ..., m-1$. On simule toutes les copies en même temps en utilisant, pour chacune des étapes, une uniforme servant à faire avancer toutes les copies d'une seule étape. Soit $\{X_t\}$ une chaîne de Markov que l'on veut échantillonner selon sa loi de probabilité stationnaire.

Définition 2.3.4. Nous appellerons $\{\mathcal{X}_t\}$ une méta-chaîne si c'est une chaîne de Markov dont l'état au temps t, $\mathcal{X}_t = (\mathcal{X}_{t,0}, \ldots, \mathcal{X}_{t,m-1})$, est constitué de m copies de l'état de la chaîne de Markov $\{X_t\}$. On obtient alors que $\mathcal{X}_{t,i}$ est une copie de X_t pour tout $t = 0, 1, \ldots$ et pour tout $i = 0, \ldots, m-1$.

Dans le cas présent, on a une chaîne de Markov $\{X_t\}$ ayant un espace d'états $\mathcal{S} = \{0, 1, \ldots, m-1\}$ et on veut simuler m copies. L'état de la méta-chaînes $\{\mathcal{X}_t\}$ au temps t est alors défini par la récurrence :

$$\mathcal{X}_0 = \{0, \ldots, m-1\} \qquad \mathcal{X}_t = f(\mathcal{X}_{t-1}, U_t) \qquad t \geq 1.$$

La fonction de transition de la méta-chaîne $f : \mathcal{S}^m \times U \to \mathcal{S}^m$ est définie à partir de la

fonction φ de la chaîne originale de la manière suivante :

$$\begin{aligned} f(\mathcal{X}_{t-1}, U_t) &= f(\{\mathcal{X}_{t,0}, \mathcal{X}_{t,1}, \ldots, \mathcal{X}_{t,m-1}\}, U) \\ &\stackrel{\text{def}}{=} \{\varphi(\mathcal{X}_{t,0}, U), \varphi(\mathcal{X}_{t,1}, U), \ldots, \varphi(\mathcal{X}_{t,m-1}, U)\} \end{aligned} \tag{2.14}$$

en utilisant des variables aléatoires U_t ~Uniforme(0,1). Dans le cas de la technique CFTP, on doit évaluer l'équation 2.13. Alors, pour la simulation de la t^{ieme} étape, du temps $-t$ à 0, on débute la simulation d'une méta-chaîne dans son état initial $\mathcal{X}_{-t} = \{0, 1, ..., m-1\}$ et on applique t fois la fonction de transition $f(\cdot)$ de la méta-chaîne d'après la suite $\{U_{-t}, U_{-t+1}, \ldots, U_{-1}\}$. Si les m éléments de la méta-chaîne ont tous le même état au temps 0, pour $\mathcal{X}_0$, on arrête la simulation et on considère cet état comme une observation tirée selon la loi de probabilité stationnaire. Sinon, on continue à l'étape suivante. On recommence ceci jusqu'à ce que pour une certaine valeur de t on obtienne une méta-chaîne dont tous les états la constituant soient de même valeur au temps 0. Quand toutes les valeurs d'un état de la méta-chaîne sont identiques, pour une étape donnée t, on peut alors considérer que celle-ci est le temps d'arrêt.

2.3.6 Algorithme CFTP

Voici l'algorithme général de la simulation de chaîne de Markov en utilisant la méthode du CFTP.

Algorithme 3 Algorithme CFTP

$t \leftarrow 0$

répéter

  $t \leftarrow t + 1$

  $u_{-t} \leftarrow$Unif()

  $\mathcal{X}_{-t} \leftarrow \{0, \ldots, m-1\}$

  pour $i = -t$ à -1 **faire**

    $\mathcal{X}_{i+1} \leftarrow f(\mathcal{X}_i, u_i)$

  fin pour

jusqu'à $\mathcal{X}_{0,0} = \mathcal{X}_{0,1} = \ldots = \mathcal{X}_{0,m-1}$

retourner l'unique valeur de $\mathcal{X}_0$

Le graphique de la figure 2.4 illustre, de gauche à droite, trois étapes de l'algorithme. À

chacune d'elle on initialise, à chacune des étapes la méta-chaîne à $\{0,1\}$. Les différentes

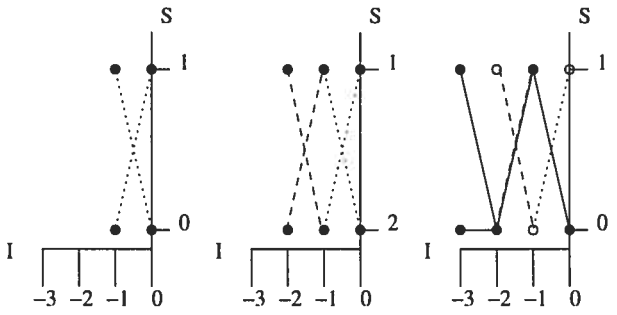

FIG. 2.4 – Exemple d'évolution avec CFTP pour le modèle de la figure 2.1

lignes pointillées et tiretées représentent les transitions des états de la méta-chaîne sans qu'il y ait fusion. Les lignes pleines représentent les transitions de la dernière étape, à ce moment il y a fusion des deux états au temps 0.

Théorème 2.3.5. *[35] Avec probabilité 1 l'algorithme 3 retourne une valeur et cette valeur est générée selon la loi de probabilité stationnaire de la chaîne.*

Preuve [35] : Sous l'hypothèse 2.2.1, il existe un entier l tel que pour toute paire d'états i,j dans $\mathcal{S}$ il y a une chance positive de passer de i à j en l étapes. Donc il existe une probabilité non nulle que la méta-chaîne $\mathcal{X}_0$ soit composée d'une seule valeur en l étapes. Donc pour chaque intervalle de temps $[-l,0]$, $[-2l,-l]$, ... il y a aussi une probabilité non nulle que pour chacun des intervalles si $\mathcal{X}_{-(i+1)\cdot l} = \{0,1,\ldots,m-1\}$ alors $\mathcal{X}_{-i\cdot l,j} = X^*$ pour tout $j = 0,\ldots,m-1$. Comme les événements sont indépendants, cela va arriver avec probabilité 1 pour une certaine valeur i.

Dans ce cas, pour une valeur assez grande i^*, si $\mathcal{X}_{-(i^*+1)\cdot l} = \{0,1,\ldots,m-1\}$ alors $\mathcal{X}_{0,j} = X^*$ pour tout $j = 0,\ldots,m-1$. Quand l'algorithme aura fait $(i^*+1)\cdot l$ étapes dans le passé, il va terminer et retourner la valeur X^*. Pour toute autre itération de l'algorithme on obtient cette même valeur. Ce résultat découle du fait qu'avec l'algorithme CFTP la méta-chaîne à comme valeur au temps $-(i^*+1)\cdot l$ tous les états de $\mathcal{S}$. Alors pour n'importe quelle réalisation d'une chaîne débutant dans un temps inférieur $-t < -(i^*+1)\cdot l$ elle va, au temps $-(i^*+1)\cdot l$, fusionner avec une des réalisations déjà calculées et sera égale, au temps 0, à X^*. Il en va de même pour la limite quand $-t \to -\infty$. On a déjà démontré que la chaîne de Markov et le couplage que l'on a défini ont la même loi de probabilité stationnaire, donc on conclut que $X^* \sim \pi$■

On peut, avec ce résultat, produire par simulation des valeurs générées selon la loi de probabilité stationnaire. On recommence la simulation n fois indépendamment pour obtenir un échantillon de n valeurs $\{X_0^*, X_1^*, \ldots X_{n-1}^*\}$ et on construit l'estimateur, équation (2.10), avec ces valeurs

$$\bar{\mu}_\infty^n = \frac{1}{n}\sum_{i=0}^{n-1} C(X_i^*) \tag{2.15}$$

qui est non biaisé pour μ_∞. Il y a deux détails très importants à noter, car il faut s'assurer de respecter l'évaluation de l'équation (2.13). Même si tous les états d'une méta-chaîne ont fusionné avant le temps 0, on simule quand même jusqu'au temps 0 et c'est seulement à ce moment là que l'on regarde si toutes les chaînes ont fusionné. La deuxième chose est qu'il faut toujours réutiliser les mêmes uniformes, pour les mêmes étapes, lorsque la simulation est réinitialisée plus loin dans le passé. Si on ne réutilise pas les mêmes uniformes aux mêmes temps de simulation, on aura un biais. Si on regarde l'exemple de la figure 2.1, l'utilisation de nouvelles valeurs à chaque étape de l'algorithme 3 introduira un biais en faveur de l'état (0).

Quand l'espace d'états $\mathcal{S}$ est fini, ible de ne pas recalculer toutes les transitions pour chacune des étapes antérieures. On fait une composition de fonctions d'après les fonctions de transition déjà calculées. Soit la fonction de transition $f_{-t} = f(\cdot, u_{-t})$: $\mathcal{S}^m \to \mathcal{S}^m$ qui définit la transition de chacun des états de la méta-chaîne selon $u_{-t} \in [0,1)$. La fonction de transition pour plusieurs étapes, du temps $-t$ au temps 0, est donc composée de plusieurs fonctions de transition pour une étape $f_{-t} \circ f_{-t+1} \circ \cdots \circ f_{-1}$. On note cette composition par F_{-t}^0, avec comme règle de mise à jour $F_{-t}^0 = F_{-t+1}^0 \circ f_{-t}$.

Pour une méta-chaîne, la fonction F_{-t}^0 que l'on construit représente la transition de cette méta-chaîne du temps $-t$ au temps 0. Elle utilise implicitement une suite d'uniformes déjà déterminée d'après la construction de la fonction F_{-t+1}^0 et une nouvelle valeur $u_{-t} \in [0,1)$ au temps $-t$ et ce pour toutes valeurs de t. Au temps $-t+1$, on a la transition pour toutes les initialisations possibles F_{-t+1}^0.

$$F_{-t+1}^0(x) = \varphi(\varphi(\ldots\varphi(x, u_{-t+1})\ldots, u_{-2}), u_{-1}) \qquad \forall x \in \mathcal{S}.$$

Comme l'espace d'états est fini, il est alors possible de calculer la valeur de la fonction pour chaque $x \in \mathcal{S}$. On peut mémoriser ces valeurs dans un vecteur à m éléments.

L'indice d'un élément correspondant à la valeur de l'état de départ au temps $-t+1$ et la valeur de l'élément celle de la valeur au temps 0. Quand on recommence à la prochaine étape $-t$, on calcule les évolutions f_{-t} pour une seule étape avec u_{-t}. La fonction du passage $-t$ à $-t+1$ est définie par

$$f_{-t}(x) = \varphi(x, u_{-t}) \qquad \forall x \in \mathcal{S}.$$

On combine alors les nouvelles évolutions avec les anciennes correspondantes pour obtenir notre nouvelle fonction de transition $F^0_{-t} \leftarrow F^0_{-t+1} \circ f_{-t}$.

$$\begin{aligned} F^0_{-t}(x) &= F^0_{-t+1}(f_{-t}(x)) \\ &= F^0_{-t+1}(\varphi(x, u_{-t})) \qquad \forall x \in \mathcal{S}. \end{aligned} \tag{2.16}$$

Comme on a déjà mémorisé la valeur de la fonction $F^0_{-t+1}(\cdot)$ pour chaque $x \in \mathcal{S}$ dans un vecteur on l'utilise pour faire la mise à jour avec la fonction $f_{-t}(\cdot)$. La mise à jour est alors plus rapide que l'ancienne car on peut la calculer en ordre du nombre d'éléments des états de la méta-chaîne. Avec cette manière de procéder on obtient l'algorithme 4.

Algorithme 4 Algorithme CFTP, $\mathcal{S}$ fini [35]

$t \leftarrow 0$

$F^0_t \leftarrow$ fonction identité sur $\{0,\ldots,m-1\}$

répéter

  $t \leftarrow t+1$

  $u_{-t} \leftarrow$Unif()

  $f_{-t}(\cdot) \leftarrow f(\cdot, u_{-t})$

  $F^0_{-t}(\cdot) \leftarrow F^0_{-t+1} \circ f_{-t}$

jusqu'à $F^0_{-t}(.)$ soit constant

retourner l'unique valeur de l'image de $F^0_{-t}(.)$

La valeur de l'image de $F^0_{-t}(\cdot)$ correspond à la valeur des différents éléments de la méta-chaîne de l'algorithme 3 au temps 0 si on débute la simulation au temps $-t$ avec comme état $\mathcal{X}_{-t} = \{0, 1, \ldots, m-1\}$.

Malheureusement, lorsque $\mathcal{S}$ est très grand ou infini on ne peut pas appliquer cette méthode telle quelle. La raison étant que l'on ne peut garder la trace que d'un nombre

fini d'états. Mais on peut quand même s'accommoder lorsque la fonction de transition φ nous donne une évaluation monotone.

Hypothèse 2.3.6. *Supposons que l'espace d'états $\mathcal{S}$ possède un ordre partiel et que la fonction de transition φ a comme propriété que pour tout i, j dans $\mathcal{S}$, si $i \leq j$, alors $\varphi(i, U) \leq \varphi(j, U)$ presque sûrement par rapport à $U \in [0, 1)$.*

Définition 2.3.5. On dira qu'une chaîne de Markov respectant l'hypothèse 2.3.6 possède la propriété d'avoir une évaluation Monte Carlo monotone.

On peut alors se contenter de suivre la trace de deux états, soit le plus petit $\widehat{0}$ et le plus grand $\widehat{1}$. Avec la propriété d'avoir une évaluation Monte Carlo monotone, toute les valeurs des images des fonctions de transition des autres états $\varphi(i, u), i \in \mathcal{S}$, seront comprises entre celles des bornes supérieure et inférieure $\varphi(\widehat{0}, u) \leq \varphi(i, u) \leq \varphi(\widehat{1}, u)$ pour tout $u \in (0, 1]$. La même conclusion s'applique sur la récursion de plusieurs étapes. Lorsque les deux chaînes initialisées dans ces deux états seront fusionnées, alors on peut conclure que tout l'espace d'états se serait fusionné avec la même valeur d'état et au même instant. À chaque passage de t à $t + 1$ on doit recalculer toutes les fonctions de transition pour la méta-chaîne du temps $-t$ à 0.

Algorithme 5 Algorithme CFTP, $\mathcal{S}$ très grand [35]

$t \leftarrow 0$
répéter
  $t \leftarrow t + 1$
  $u_{-t} \leftarrow$Unif()
  $\mathcal{X}_{-t} \leftarrow \{\widehat{0}, \widehat{1}\}$
  pour $i = -t$ à -1 **faire**
    $\mathcal{X}_{i+1} \leftarrow f(\mathcal{X}_i, u_i)$
  fin pour
jusqu'à $\mathcal{X}_{0,0} = \mathcal{X}_{0,1}$
retourner l'unique valeur $\mathcal{X}_{0,0}$

On peut voir dans le graphique de la figure 2.5 un exemple d'évolution de l'algorithme où l'on garde la trace seulement du plus grand et du plus petit des états. Les

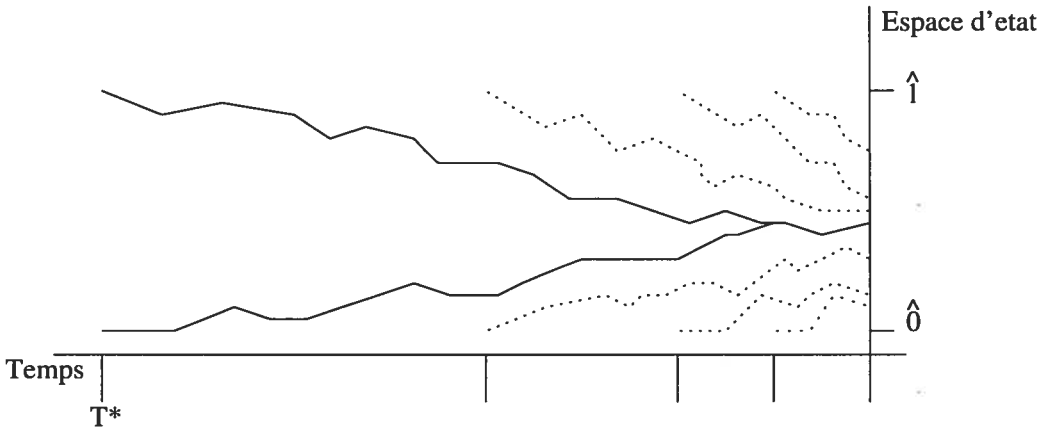

FIG. 2.5 – Exemple d'évolution avec CFTP, sous hypothèse 2.3.6

lignes pointillées représentent des trajectoires pour des cycles antérieurs. Les deux lignes pleines débutant au temps T^* dans le plus petit et le plus grand des états représentant le dernier cycle de l'algorithme 5. On voit dans le graphique de la figure 2.5 que chacune des nouvelles évaluations donne un résultat compris entre les précédentes, c'est la conséquence de la propriété d'évaluation Monte Carlo monotone.

Comme on doit tout recalculer à chaque passage où on augmente la valeur de t, on utilise généralement un incrément différent de 1. On peut étudier le problème, pour un exemple particulier, de trouver des incréments permettant de réduire le coût total d'évalution de l'algorithme. Dans le cas général, pour cet algorithme, on suggère d'utiliser une séquence de puissances de 2 pour faire les calculs comme suggérée dans l'article de Propp et Wilson [35]. Pour une valeur t, le temps de départ est -2^t. Pour plus de détails sur la méthode du "Coupling From The Past", voir [35, 4, 6, 25, 38].

2.4 Conclusion

Dans ce chapitre on a défini certaines hypothèses sur les chaînes de Markov et comment faire de la simulation par ordinateur de celles-ci. On a vu comment faire de l'échantillonnage selon la loi de distribution stationnaire des états d'une chaîne de Markov apériodique et irréductible. On s'est familiarisé avec des algorithmes permettant de produire des estimateurs non biaisés de μ_∞. Maintenant on va voir comment réduire la variance de ces estimateurs.

Chapitre 3

Réduction de la variance

3.1 Le but

La technique du CFTP offre un moyen de produire des échantillons non biaisés $\{X_0, \ldots, X_{n-1} | X_i \sim \pi, i = 0, \ldots, n-1\}$, tirés de la loi de probabilité stationnaire des états de certaines chaînes de Markov. Avec ces outils, il est possible de produire, par simulation par ordinateur, un estimateur de μ_∞, défini à l'équation (2.7). Maintenant que l'on a une méthode permettant d'estimer par simulation de manière non biaisée, que faire de plus ? Nous nous efforçons toujours d'augmenter l'efficacité de nos simulations et c'est dans ce but, que l'on s'attaque à la réduction de la variance. Une fois le biais annulé, la seule manière de réduire l'erreur quadratique moyenne est de réduire la variance. Le taux de convergence de la variance s'exprime de manière générale par $O(n^{-1})$, dans le cas du calcul de (2.10). On peut bien sûr la réduire en augmentant le nombre de simulations. On désire plutôt obtenir une variance inférieure pour un nombre équivalent de simulations. Une des manières d'y parvenir, est de considérer les variables aléatoires utilisées lors de la simulation. Nous verrons dans ce chapitre comment l'utilisation de différentes techniques peut permettre de réduire la variance de nos estimateurs. Comme il est très difficile de comprendre la dynamique qui règne, la majorité des résultats illustrés dans ce chapitre sont des résultats d'expériences.

3.2 Variables antithétiques

Supposons que l'on essaie d'estimer un certain paramètre μ par la moyenne calculée à partir d'un échantillon de taille n, $\{Y_0, Y_1, \ldots, Y_{n-1}\}$. Chacun des Y_i s'exprime comme étant $Y_i = C(X_T)$ (équation 2.2 pour $T = 1$). Pour que ces variables soient dites antithétiques, il suffit que la corrélation de chacune des paires $(Y_i, Y_j), i \neq j$ de variables soit négative et qu'elles soient tous égaux en espérance à μ, c'es-à-dire $E[Y_i] = \mu$ [10]. On obtient ainsi que leur moyenne,

$$\widehat{Y} = \frac{1}{n}\sum_{i=0}^{n-1} Y_i, \tag{3.1}$$

a la même espérance et sa variance est donnée par :

$$\widehat{\sigma}_n^2 = \frac{1}{n^2}\sum_{j=0}^{n-1}\mathrm{Var}[Y_j] + \frac{2}{n^2}\sum_{i=0}^{n-1}\sum_{j=0}^{i-1}\mathrm{Cov}(Y_i, Y_j). \tag{3.2}$$

Comme la valeur des corrélations des couples $(Y_i, Y_j), i \neq j$ est négative pour chacun des couples, la somme des covariance le sera aussi. Si la variance unitaire des variables est la même qu'avec l'estimateur Monte Carlo (1.4), alors la variance de cet estimateur $\widehat{Y}$ est inférieure à celle de l'estimateur calculé à l'aide de variables indépendantes [33, 34, 17, 22].

3.2.1 Paires antithétiques

On commence par regarder le cas où l'on a simplement deux variables aléatoires. On va définir quelques propriétés permettant d'obtenir une réduction de la variance avec deux variables aléatoires.

Définition 3.2.1. [19] Les variables aléatoires X_1 et X_2 sont dites négativement dépendantes par quadrant si pour tous nombres réels x_1 et x_2

$$P(X_1 \leq x_1, X_2 \leq x_2) \leq P(X_1 \leq x_1) \cdot P(X_2 \leq x_2). \tag{3.3}$$

Si le signe de l'inégalité est inversé, on dira que les variables sont positivement dépendantes par quadrant.

Théorème 3.2.1. *[19] Soit (X_1, X_2) une paire de variables aléatoires, X_1 et X_2 sont négativement dépendantes par quadrant si et seulement si $Cov[f(X_1), g(X_2)] \leq 0$ pour toutes fonctions non-décroissantes f, g pour lesquelles une telle covariance existe.*

Dans le cadre de la simulation de chaînes de Markov, voici comment les paires antithétiques permettent d'obtenir une réduction de la variance. On va introduire une corrélation négative à travers les uniformes générées durant la simulation. Supposons que l'on a deux chaînes de Markov $\{X_t^{(1)}\}$,$\{X_t^{(2)}\}$ et que l'on veut générer un couple d'uniformes permettant de minimiser la correlation entre les réalisations de celles-ci. Une stratégie est de générer les deux valeurs selon le procédé suivant [19, page 336] : on a une séquence de variables aléatoires $V^{(1)} = (U_1^{(1)}, U_2^{(1)}, \ldots)$ et on définit la séquence antithétique $V^{(2)} = (1-U_1^{(1)}, 1-U_2^{(1)}, \ldots)$. Les paires $(U_t^{(1)}, 1-U_t^{(1)})$ sont négativement dépendantes par quadrant. À chaque étape, la nouvelle valeur est obtenue grâce à la récurrence stochastique :

$$X_0^{(1)} = x_0^{(1)}, X_{t+1}^{(1)} = \varphi(X_t^{(1)}, U_t^{(1)}) \text{ et } X_0^{(2)} = x_0^{(2)}, X_{t+1}^{(2)} = \varphi(X_t^{(2)}, 1-U_t^{(1)}).$$

On obtient alors que pour une certaine étape t la valeur des deux processus à l'étape suivante est $X_{t+1}^{(1)} = \varphi(x_t^{(1)}, U_t^{(1)})$ et $X_{t+1}^{(2)} = \varphi(x_t^{(2)}, 1-U_t^{(1)})$, où φ est la fonction définie par l'équation (2.9). Le couple $(U_t^{(1)}, 1-U_t^{(1)})$ est négativement dépendant par quadrant pour tout $j \geq 1$ et la fonction φ est croissante. Alors, par le théorême 3.2.1, le couple $(X_{t+1}^{(1)}, X_{t+1}^{(2)})$ a une covariance négative. Cela reste vrai pour les états finaux des deux chaînes ($X_{\tau_1}^{(1)}$ et $X_{\tau_2}^{(2)}$), où τ_1 et τ_2 représente le temps d'arrêt respectivement pour la chaîne 1 et 2. Si la fonction de coût $C(\cdot)$ est aussi croissante, les deux variables aléatoires $Y_1 = C(X_{\tau_1}^{(1)}), Y_2 = C(X_{\tau_2}^{(2)})$ sont aussi négativement dépendantes par quadrant. Si on calcule un estimateur de μ_∞, défini à l'équation (2.10), en utilisant ce type de génération de variables aléatoires, alors les couples de la forme (Y_1, Y_2) servant au calcul de l'estimateur $\bar{\mu}_\infty^n$ auront aussi une covariance négative. Cela aura pour effet de réduire la variance de l'estimateur $\bar{\mu}_\infty^n$, d'après l'équation (1.4).

Par contre, pour $n > 2$, il n'est pas évident de générer les différentes variables tout en conservant leurs covariances négatives. Plusieurs méthodes ont été proposées pour générer de manière corrélée des variables aléatoires. L'utilisation de différentes techniques est suggérée par Craiu et Meng [3] et Lemieux et Sidorsky [24] : Permuted displacement, Multivariate Normal, Iterative Latin Hypercube sampling et méthodes Quasi Monte Carlo Randomisées.

3.2.2 Variables antithétiques généralisées

Dans l'équation (3.2), on peut voir que si la somme des covariances est négative, alors on obtient une réduction de la variance par rapport à l'estimateur Monte Carlo standard. On veut maintenant définir des manières d'y parvenir en utilisant plus de deux variables. L'idée est d'induire une dépendance entre les différentes variables utilisées durant la simulation, mais pas nécessairement pour chacun des couples de variables. Plusieurs méthodes sont envisageables, on va se concentrer sur seulement six d'entre elles pour le moment. Le but de cette section n'est pas de donner des preuves que les ensembles définis par Craiu et Meng [3] et Lemieux et Sidorsky [24] permettent d'obtenir une réduction de la variance. Cette section est consacrée au survol des techniques qu'ils proposent ainsi que de donner une intuition permettant de croire que les ensembles RQMC permettent effectivement d'obtenir une réduction de la variance. On va regarder la définition de plusieurs ensembles.

3.2.3 Définition d'ensembles de points

Maintenant on tente de trouver des ensembles de variables aléatoires de $n \geq 3$ valeurs possédant ces propriétés. On va avoir recourt à ce que l'on appelle des ensembles de points.

Définition 3.2.2. Un ensemble de points P^n en dimension d est un ensemble de n vecteurs $\{V_{(0)}, \ldots, V_{(n-1)}\}$ en dimension d, $V_{(i)} = (U_{(i)}^0, \ldots, U_{(i)}^{d-1})$. Par définition, on appelle le vecteur $V_{(i)}, i = 0, \ldots n-1$, le point i, et l'élément $U_{(i)}^t \in [0,1), t = 0, \ldots, d-1$ du vecteur i, la coordonnée t du point i.

Pour s'assurer que l'on pourra utiliser ces ensembles pour produire des estimateurs sans biais par simulation, on pose les hypothèses 3.2.2 et 3.2.3.

Hypothèse 3.2.2. *Les points $\{V_{(0)}, \ldots, V_{(n-1)}\}$ de P^n en dimension d sont tous définis sur l'hypercube unitaire $[0,1)^d$ et recouvrent de manière uniforme cet hypercube.*

Hypothèse 3.2.3. *Pour le point $V_{(i)} = (U_{(i)}^0, \ldots, U_{(i)}^{d-1})$, chacune des coordonnées obéit à la loi Uniforme(0,1) et les coordonnées sont indépendantes les unes des autres pour un même vecteur $i = 0, \ldots, n-1$.*

Une fois un tel ensemble P^n défini, on réalise n simulations via l'un des algorithmes 3, 4 ou 5. Chacune des simulations utilisant un point (ou vecteur) différent de l'ensemble P^n. Chacun des points sera donc utilisé une seule fois et tous les points de l'ensemble seront utilisés. Pour une simulation, on itère sur les coordonnées en ordre croissant de leurs indices à chaque appel de la fonction Unif().

3.2.4 Ensembles négativement associés

Cette sous-section contient les définitions des ensembles de points que Craiu et Meng [3] ont utilisés pour leurs expériences.

La méthode de déplacement permuté

Pour obtenir la t^{ieme} coordonnée de tous les points, on génère tout d'abord $U^t_{(0)} \sim$ Uniforme$(0,1)$ et on construit ensuite

$$U^t_{(i)} = \left(2^{i-1}U^t_{(0)} + \frac{1}{2}\right) \mod 1, \qquad i = 1,\ldots,n-2, \text{ et } U^t_{(n-1)} = 1-(2^{n-1}u_0 \mod 1).$$

On obtient ainsi l'ensemble des t^{ieme} coordonnées de tous les vecteurs $P^n_t = \{U^t_{(0)}, U^t_{(1)}, \ldots, U^t_{(n-1)}\}$ servant à faire avancer toutes les méta-chaînes d'une seule étape.

La méthode par copule normale

Cette manière de faire est un cas particulier de la méthode NORTA [1, 7, 19] où on définit chacun des élément (i,t) de la matrice de covariance Σ comme étant

$$\Sigma_{it} = -(n-1)^{-1} \qquad \text{si} \qquad i \neq t \qquad \text{et} \qquad \Sigma_{ii} = 1.$$

On génère tout d'abord un ensemble $(Z_0,\ldots,Z_{n-2}) \sim N(0,\Sigma)$ et $Z_{n-1} = -(Z_0 + Z_1 + \ldots + Z_{n-2})$. Finalement, pour générer les uniformes, on utilise la relation $U^t_{(i)} = \Phi(Z_i)$ pour tout $i \in \{0,\ldots,n-1\}$ pour construire $P^n_t = \{U^t_{(0)},\ldots,U^t_{(n-1)}\}$ l'ensemble des t^{ieme} coordonnées de tous les points, où Φ est la fonction de répartition Normale$(0,1)$.

Hypercube Latin Itératif

Cette définition est une version itérative de la méthode de l'Hypercube Latin bien connue [28, 37, 32, 26]. Cette méthode peut être considérée comme un cas particulier d'ensemble quasi-Monte Carlo randomisé.

1. Initialiser $P_0^n = (U_{(0)}^0, \ldots, U_{(n-1)}^0)$ où $\{U_{(i)}^0\}_{0 \leq i \leq n-1}$ est i.i.d. Uniforme(0,1)
2. Pour $t = 0, 1, 2, \ldots$ et $\mathcal{K}_t = (\zeta_t(0), \ldots, \zeta_t(n-1))$ une permutation aléatoire de $\{0, 1, \ldots, n-1\}$ indépendante des tirages précédents, on pose

$$P_{t+1}^n = \frac{1}{n}(\mathcal{K}_t + P_t^n)$$

Des exemples de l'effet sur la variance de certains estimateurs de ces trois dernières manières de génération de nombres aléatoires sont donnés dans les articles de Craiu et Meng [2, 3].

3.2.5 Règles de réseaux et réseaux digitaux

Les méthodes quasi-Monte Carlo randomisées sont une vaste classe de méthodes d'analyse numérique utilisant la génération de variables antithétiques généralisées. Lemieux et Sidorsky [24] ont choisi d'utiliser une règle de réseaux [30] et deux réseaux digitaux [36] pour tenter de réduire la variance des estimateurs calculés à partir de l'équation (2.10).

Règle de Korobov

Pour obtenir un ensemble de n points, la règle dépend d'un paramètre a appelé le multiplicateur

$$P^n = \left\{ \frac{1}{n} \left(i, i \cdot a \bmod n, i \cdot a^2 \bmod n, \ldots, i \cdot a^{d-1} \bmod n \right), \qquad i = 0, \ldots, n-1 \right\}$$

Le choix du multiplicateur est très important pour obtenir de bons ensembles de points. Pour choisir une bonne valeur de a, le lecteur est référé à l'article de L'Écuyer et Lemieux [22].

Ensemble de Sobol et Korobov polynomial

La manière de générer les points est semblable à la règle de Korobov initale, mais au lieu de prendre des entiers, on utilise des polynômes sur $\mathbb{F}_2$. Pour $n = 2^l$ un ensemble de Korobov polynomial P_n est obtenu en choisissant un polynôme $P(z)$ de degré l dans $\mathbb{F}_2[z]$ (l'anneau des polynômes sur $\mathbb{F}_2$), un polynôme générateur $a(z)$ dans $\mathbb{F}_2[z]/(P)$

(l'anneau des polynômes sur F_2 de degré inférieur à l) et on prend $P_n = \phi(P^n(z))$ où

$$P^n(z) = \left\{ \frac{q(z)}{P(z)}(1, a(z), a^2(z), \ldots, a^{d-1}(z)) : q(z) \in \mathbb{F}_2[z]/(P)) \right\}$$

et la fonction $\phi(\cdot)$ consiste à remplacer z par 2. Une description plus formelle sur ces règles est disponible dans [22].

Ces derniers ensembles, règle de Korobov, ensemble de Sobol et Korobov polynomial, sont dit déterministes car on donne une valeur fixe aux points dans l'ensemble. Pour obtenir un ensemble non déterministe on va randomiser chacun des points. La méthode proposée est le décalage aléatoire modulo 1 pour les ensemble de Korobov et Korobov polynômial. Pour ce faire on va simplement générer un point V suivant la loi Uniforme$(0,1]^d$ (d peut être infini) et l'ajouter à chacun des points de P^n, coordonnée par coordonnée, modulo 1. Dans le cas d'une règle de réseaux, la structure des points est préservée par le décalage [22, 19]. Pour les ensembles de Sobol, on utilisera une randomisation par matrice brouillante *(matrix scrambling)* et on fait ensuite un décalage aléatoire modulo 1 [27, 23].

3.3 Utilisation d'ensembles de points

Pour la simulation avec la technique CFTP, on peut voir dans [2, p.107] des résultats démontrant la réduction de variance de leurs trois techniques de génération de points. De manière conceptuelle, si on regarde l'évolution conjointe de toutes les réalisations des méta-chaînes, on peut voir la simulation avec la technique CFTP comme la simulation en parallèle de n méta-chaînes. Les différentes méthodes de génération antithétiques offrent un moyen de les corréler, lors des simulations, afin de réduire la dispersion de nos réalisations et d'espérer réduire la variance de notre estimateur. Si une telle corrélation peut être introduire entre les différentes valeurs retournées par les algorithmes, on espère obtenir une somme de corrélations négative des $\{(C(X_i), i = 0, \ldots, n-1\}$, $C(\cdot)$ étant la fonction de coût définie dans l'équation (2.10). La variance obtenue sera alors plus faible pour notre estimateur. Le graphique de la figure 3.1 illustre l'idée d'introduire une corrélation négative entre les différentes méta-chaînes. Comme à l'intérieur d'une méta-chaîne toutes les chaînes utilisent les mêmes variables aléatoires, la corrélation est positive. Mais entre elles, il devrait y avoir une corrélation négative.

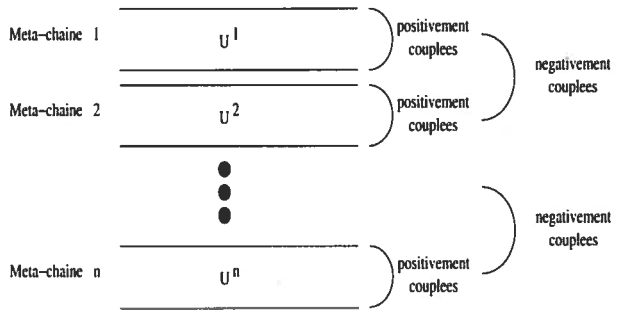

FIG. 3.1 – Évolution de n méta-chaînes couplées pour CFTP [2]

Il existe des différences d'efficacité entre les différents ensembles de points définis par Craiu et Meng [3], sous-section 3.2.4, et Lemieux et Sidorsky [24] sous-section 3.2.5. Ceux proposés par Lemieux et Sidorsky [24] présentent une manière rapide et efficace de générer des variables aléatoires négativement corrélées. Pour les trois méthodes, règle de Korobov, Korobov polynômial et ensemble de Sobol, l'efficacité des points, par rapport à la réduction de la variance, est relié au choix du multiplicateur ou du polynôme générateur. Il faut donc trouver des multiplicateurs/polynômes générateurs en tentant de maximiser cette efficacité. C'est avec cette vision que Lemieux et Sidorsky [24] ont proposé d'utiliser ce qu'ils appellent des ensembles de points très uniformes (HUPS[1]) pour générer les variables aléatoires utilisées par un algorithme CFTP. On dira qu'un ensemble de points P_n en d-dimensions est un HUPS si :

(i) Chaque point de P_n est uniformément distribué sur $[0,1)^d$

(ii) P_n couvre l'hypercube $[0,1)^d$ de manière plus homogène (selon certains critères) qu'un ensemble de points aléatoires indépendants.

Il existe plusieurs manières de mesurer l'homogénéité d'un ensemble de points. Ce mémoire ne se veut pas une analyse des ensembles de points, c'est pourquoi on ne regardera, à titre d'exemple, que le cas particulier de deux ensembles en dimensions deux. Le graphique de la figure 3.2 illustre à gauche un ensemble de Sobol et à droite un ILHS.

On a partitionné l'espace $[0,1]^2$ en 128 boîtes de taille 2^{-4} par 2^{-3}. Pour l'ensemble de Sobol, on retrouve un seul point par boîte. Par contre, pour les points générés par

[1]Nom tiré de l'atelier de recherche international intitulé "Random number generators and highly-uniform point sets, au Centre de Recherche Mathématiques, à Montréal, en juin 2002."

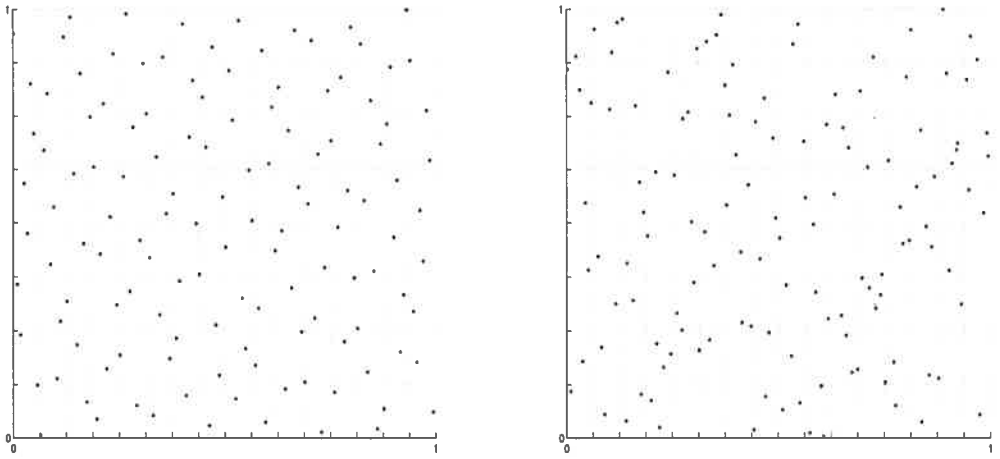

FIG. 3.2 – Deux ensembles de 128 points en 2 dimensions, division taille 2^{-4} par 2^{-3}

hypercube latin itératif il en est tout autre. Certaines boîtes ne contiennent aucun point tandis que d'autres peuvent en contenir jusqu'à quatre. On n'a donc pas une répartition très uniformae des points dans les différentes boîtes. De même si on utilise des boîtes rectangulaires de différentes tailles, alors pour l'ensemble de Sobol il y a toujours un seul point par boîte. On retrouve les même points dans la figure 3.3 avec une partition de boîtes de taille 2^{-2} par 2^{-5}

La figure 3.4 contient encore les même points mais avec une partition de boîtes de taille 2^{-5} par 2^{-2}.

Pour l'ensemble de Sobol en deux dimensions, chacune des boîtes contiendra exactement un point et ce pour n'importe quel division de taille 2^{-q_1} par 2^{-q_2} tel que $n = 2^{q_1+q_2}$ [19]. Ce n'est donc pas n'importe quel ensemble de points RQMC qui est un HUPS. Il est important de porter une attention particulière au choix des ensembles de points car la performance de l'algorithme est liée au choix des ensembles.

3.3.1 Randomisation des ensembles de points

Sous l'hypothèse 3.2.3, on doit appliquer certaines tranformations sur nos ensemble de points. On doit les randomiser afin que les différentes coordonnées d'un même vecteur soient indépendantes. Pour les HUPS, l'ensemble de points que l'on utilise peut ne pas

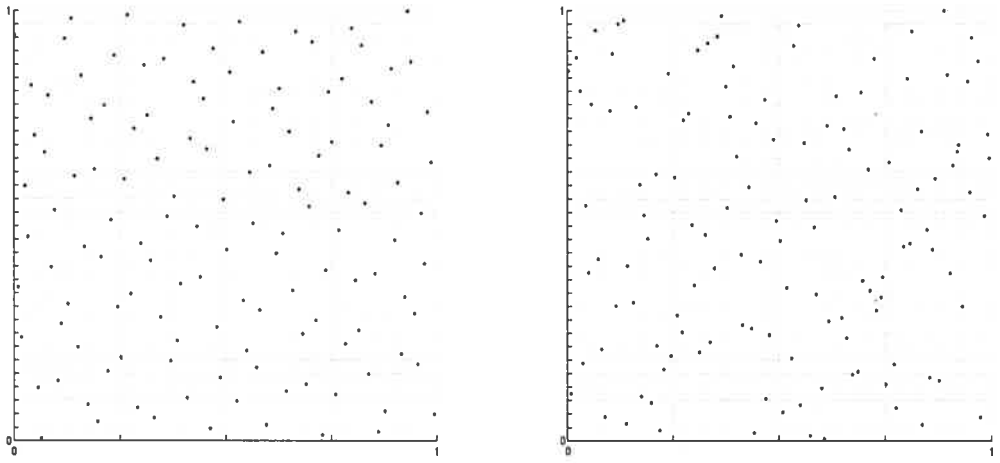

FIG. 3.3 – Deux ensembles de 128 points en 2 dimensions, division de taille 2^{-2} par 2^{-5}

être généré avant la simulation, mais pendant l'exécution de celui-ci. Il est possible de remplacer la valeur de l'uniforme de la i^{ieme} réalisation au temps $-t$, u^i_{-t} dans l'algorithme de la figure 6, par sa définition explicite [24]. En exemple, une règle de Korobov avec un décalage aléatoire modulo 1

$$u^i_{-t} = (((i \cdot a^t) \mod n)/n + \tilde{u}_t) \mod 1 \tag{3.4}$$

où $\tilde{u}_t$ est une variable obéissant à la loi Uniforme(0,1).

On ne fait donc plus la simulation d'une seule réalisation à la fois comme avec l'algorithme 4, mais plutôt une évolution de n réalisations en parallèle. On calcule ensuite une valeur possible pour $\bar{\mu}^n_\infty$ à partir de l'équation (2.10) et des n valeurs retournées par l'algorithme 6. Maintenant que l'on peut calculer un estimateur non biaisé de (2.8) et que l'on espère que sa variance soit inférieure à celui construit à partir de l'algorithme 4 on aimerait bien mesurer l'effet de nos ensembles de points. Pour cela on va estimer la variance de $\bar{\mu}^n_\infty$. Quand on utilise des ensembles de points qui ne sont pas indépendants, cela peut être fait en répétant l'algorithme plusieurs fois indépendamment. Pour avoir une répétition indépendante, on génère une nouvelle valeur pour le décalage aléatoire. On obtient ainsi un ensemble de points indépendant du précédent, donc une simulation indépendante. Soit r le nombre de fois que l'on répète l'algorithme 6 indépendamment. On obtiendra alors r estimateurs indépendants identiquement distribués $\widehat{\mu}_0, \ldots, \widehat{\mu}_{r-1}$

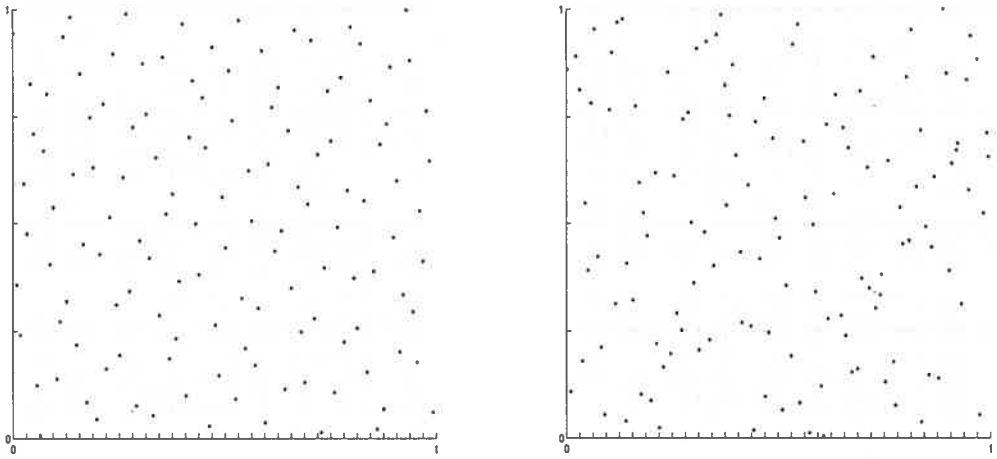

FIG. 3.4 – Deux ensembles de 128 points en 2 dimensions, division de taille 2^{-5} par 2^{-2}

définis par l'équation (2.10). Alors la variance de l'estimateur suivant [24]

$$\widehat{\mu} = \frac{1}{r}\sum_{v=0}^{r-1}\widehat{\mu}_v \tag{3.5}$$

peut être estimée sans biais par

$$\widehat{\sigma}^2 = \frac{1}{r}\sum_{v=0}^{r-1}\frac{(\widehat{\mu}_v - \widehat{\mu})^2}{r-1}. \tag{3.6}$$

L'estimateur obtenu à l'aide de l'équation (3.5) est aussi un estimateur non biaisé de μ_∞ et

$$E[\widehat{\mu}] = E[\widehat{\mu}_v] = \mu_\infty.$$

On a maintenant un nouvel estimateur $\bar{\mu}$ et un estimateur de sa variance $\widehat{\sigma}^2$. On aimerait avoir une idée de l'augmentation de l'efficacité de $\widehat{\sigma}^2$.

Définition 3.3.1. Le facteur de réduction de la variance est le rapport de la variance unitaire de l'estimateur Monte Carlo, $\mathrm{Var}[\bar{\mu}_\infty^n]$, et de celle du nouvel estimateur, $\mathrm{Var}[\widehat{\mu}]$:

$$\mathrm{VRF}[\bar{\mu}] = \frac{\mathrm{Var}[\bar{\mu}_\infty^n]}{\mathrm{Var}[\widehat{\mu}]}.$$

On peut approximer cette valeur en utilisant les estimateurs des différentes variances

$$\mathrm{VRF}[\bar{\mu}] = \frac{E[\bar{\sigma}_\infty^2]}{n \cdot E[\widehat{\sigma}^2]} \approx \frac{\bar{\sigma}_\infty^2}{n \cdot \widehat{\sigma}^2}.$$

Algorithme 6 Algorithme CFTP, $\mathcal{S}$ fini, points antithétiques

$t \leftarrow 0$

$F_t^{0,(i)} \leftarrow$ fonction identité sur $\{ 0,...,m-1\}, i = 0, \ldots n-1$

répéter

$t \leftarrow t+1$

Construire un ensemble $P_t^n = \{u_{(0)}^t, u_{(1)}^t, \ldots, u_{(n-1)}^t\}$ d'après une méthode génération de variables antithétiques généralisées

pour i de 0 à $n-1$ **faire**

$u_{-t,(i)} \leftarrow u_{(i)}^t$

$f_{-t}^{(i)}(\cdot) \leftarrow f(\cdot, u_{-t,(i)})$

$F_{-t}^{0,(i)} \leftarrow F_{-t+1}^{0,(i)} \circ f_{-t}^{(i)}$

fin pour

jusqu'à $F_{-t}^{(0,(i))}(.)$ soit constant pour tout $i = 0, \ldots, n-1$

retourner l'ensemble des valeurs des images des différents $F_{-t}^{0,(i)}(.)$, $i = 0, \ldots, n-1$

Le VRF servira de base de comparaison des estimateurs dans le chapitre 5. Ici on ne tient pas encore compte du temps de simulation, on y reviendra lors des exemples numériques.

3.4 Conclusion

On a pu voir dans ce chapitre comment la générations de nombres aléatoires utilisés durant une simulation par ordinateur peut influer sur la variance des estimateurs. On a combiné ces techniques de réduction de la variance avec l'algorithme CFTP. On a défini un critère de comparaison pour nos différents ensembles de points. On va maintenant aborder le sujet principal de ce mémoire, soit de la combinaison du CFTP et de la technique quasi-Monte Carlo randomizé vectoriel.

Chapitre 4

Combinaisons avec Array-RQMC

4.1 Array-RQMC

Nous allons maintenant étudier la combinaison d'une autre technique de réduction de variance avec la technique du CFTP. Il s'agit de la technique appelée Array-RQMC (RQMC vectoriel) introduite par L'Écuyer, Lécot et Tuffin [20] qui, pour certains exemples, permet de réduire significativement la variance pour l'échantillonnage de chaînes de Markov. Soit la chaîne de Markov $\{X_t\}$, avec espace d'états $\mathcal{S}$, évoluant selon la récurrence stochastique suivante

$$X_0 = x_0, \qquad X_t = \varphi_t(X_{t-1}, U_t), \qquad j \geq 1$$

où $U_t \sim$Uniforme(0,1) et $\varphi_t : \mathcal{S} \times [0,1) \to \mathcal{S}$.

Hypothèse 4.1.1. *On suppose que $\mathcal{S} \subseteq \mathbb{R}^l \cup \{\infty\}$ et qu'il y a une fonction $h : \mathcal{S} \to \mathbb{R} \cup \{\infty\}$ qui assigne un nombre réel à chaque état de la chaîne, avec $h(\infty) = \infty$.*

L'idée derrière Array-RQMC est de simuler des chaînes de Markov en parallèle. À chaque étape, on utilise un ensemble HUPS, randomisé indépendamment des autres étapes, pour faire la transition de chacune des chaînes. Après chaque transition, on trie les chaînes en ordre croissant de leurs valeurs de $h(\cdot)$. Un bon choix de cette fonction est important pour l'efficacité de l'algorithme. Voici l'algorithme de base pour Array-RQMC, tel que l'on peut le voir dans l'article de L'Écuyer, Lécot et Tuffin [20], lorsque l'on simule n chaînes de Markov en parallèle, numérotées de 0 à $n-1$, avec lesquelles

Algorithme 7 Algorithme Array-RQMC

pour $t = 1; X_{0,t-1} < \infty; t++$ **faire**

Randomiser l'ensemble de points P_n pour obtenir l'ensemble $\tilde{P}_n = (\tilde{U}_0, \ldots, \tilde{U}_{n-1})$

pour $i = 0; i < n$ and $X_{i,t-1} < \infty; i++$ **faire**

$X_{i,t} = \varphi(X_{i,t-1}, U_i)$

fin pour

Trier (et renuméroter) les chaînes pour lesquelles $X_{i,t} < \infty$ en ordre croissant de leur valeurs de $h(X_{i,t})$

fin pour

retourner la moyenne $\bar{\mu}_t^n$ des n réalisations de Y (équation (2.2)), $Y_i = C(X_{i,\tau_i})$ $i = 0, \ldots, n-1$ comme estimateur de μ_T

on cherche à estimer la valeur de μ_T, défini dans l'équation (2.4). La variable τ_i indique le temps d'arrêt, valeur aléatoire, pour la réalisation i. L'ensemble de points P_n doit respecter certaines propriétés pour être utilisé avec l'algortihme Array-RQMC [21]. Soit $P_n = (\tilde{U}_0, \ldots, \tilde{U}_{n-1})$ un ensemble de points RQMC sur $[0,1)^d$ construit de tel manière que l'ensemble de points $P'_n = (U'_i = ((i + 0.5)/n, U_i), 0 \leq i < n)$ possède les deux propriétés suivantes :

a) U_i est un vecteur aléatoire uniformément distribué sur $[0,1)^d$ pour chaque i.

b) P'_n est "très uniforme" sur $[0,1)^{d+1}$ (voir section 3.3 pour ce que l'on entend par très uniforme).

On utilise les équations (3.5) et (3.6) pour estimer la variance de cet estimateur. Chacune des chaînes se voit donc associer, pour chaque étape, un point d'un ensemble HUPS randomisé. Le tri des chaînes est défini par fonction $h(\cdot)$. Cela implique que le choix des uniformes qui seront assignées aux différentes chaînes sera dépendant de cette fonction $h(\cdot)$. Le tri effectué à chaque étape ne peut être déterminé à priori, puisque l'on ne connaît pas les valeurs des réalisations avant de débuter la simulation.

4.2 La combinaison du CFTP et de Array-RQMC

On a déjà vu dans le chapitre 3 que certaines personnes ont proposé d'utiliser des ensembles de points négativement corrélés pour réduire la variance d'échantillons produits

à l'aide de la technique CFTP. Dans tous les cas, on fait une simulation de processus en parallèle. La technique Array-RQMC permet parfois d'obtenir des gains importants au niveau du facteur de réduction de la variance lors de simulations utilisant des chaînes de Markov. Avec cette technique on simule aussi plusieurs processus en parallèle. Il est donc naturel de penser que l'on peut combiner la technique CFTP et Array-RQMC afin d'augmenter la précision de nos estimateurs. C'est précisément ce que l'on verra dans la suite de ce chapitre et des exemples numériques des différents algorithmes sont donnés dans le chapitre suivant.

Ce qui peut sembler évident au départ ne l'est pas toujours. Tout d'abord, il y a deux versions distinctes de l'algorithme CFTP : l'un avec un espace d'états fini petit et l'autre avec un espace d'états très grand. Il faudra aborder chacune d'elle différement. D'autre part, il y a plus d'indépendance entre les cycles des algorithmes et il y a d'autres contraintes de simulation comme la réutilisation des uniformes, que l'on n'a pas dans le cas de l'algorithme Array-RQMC de base. On utilise maintenant une fonction $h(\cdot)$ servant à trier les processus selon la valeur de leurs états à un temps donné. La performance de l'algorithme 7 est directement liée au choix de cette fonction. Plus l'état est complexe, plus il est difficile de choisir une fonction $h(\cdot)$ adéquate. Pour une chaîne de Markov donnée, les états des méta-chaînes, sont plus complexes que lors de la simulation utilisant le CFTP puisqu'ils sont constitués d'un ensemble de m éléments. Malgré toutes ces difficultés, on verra qu'il est possible d'obtenir de bons résultats.

On appellera CFTP-Array-RQMC les techniques utilisant à la fois la notion de couplage de chaînes de Markov selon le CFTP et de la technique de réduction de la variance Array-RQMC. Cette section aborde différentes façons de faire cette combinaison ainsi que les forces et faiblesses de chacune d'entre elles.

4.2.1 Combinaison directe

La première idée est d'appliquer l'algorithme 7 avec le CFTP. L'idée est la même que dans l'article de Lemieux et Sidorsky [24], mais en utilisant le nouvel algorithme pour produire les uniformes aux différentes étapes. L'état de la méta-chaîne i à une étape $-t$ est l'image de la fonction $F^0_{-t,i}$. À chaque cycle, passant de t à $t+1$, on recalcule l'ensemble des transitions à partir de la fonction $h(\cdot)$ aux différentes étapes de la méta-

chaîne du temps $-t+1$ à 0. La fonction $h(\cdot)$ proposée est la moyenne des valeurs de l'image de $F^0_{-t,i}$ pour la i^{eme} méta-chaîne au temps $-t$.

1. On initialise les n méta-chaînes.
2. On fait la simulation de toutes les méta-chaînes jusqu'au temps 0 en utilisant l'algorithme Array-RQMC et les ensembles de points déjà générés.
3. Si toutes les méta-chaînes sont composées d'une seule valeur d'état, on retourne le vecteur au temps 0 comme une observation tirée de la loi de probabilité stationnaire. On note ce temps d'arrêt T^*. Sinon, on recommence à l'étape 1 avec une valeur de départ inférieure.

Soit T^* l'étape où toutes les n méta-chaînes ont atteint le critère d'arrêt. Pour une fonction de coût $C(\cdot)$, on va approximer la valeur de μ_∞ par

$$\begin{aligned}\mu_\infty &= E_\pi[C(X)] \\ &= E[C(F^0_{-t}(\cdot))] \\ &\approx \frac{1}{n}\sum_{i=0}^{n-1} C(F^0_{-T^*,i}(\cdot)) = \widehat{\mu}.\end{aligned}$$

L'algorithme 8 effectue un tel échantillonnage pour un ensemble HUPS P_n et une chaîne de Markov à m états numérotés $0,\ldots,m-1$. Pour les méthodes RQMC, on a une définition de l'ensemble de points permettant de générer chacun des points au besoin à chacune des étapes pour chacune des chaînes. Pour Array-RQMC, une fois qu'un ensemble de points est associé à une étape, il devra le rester jusqu'à la fin de la simulation. On devra alors réutiliser les ensembles de points plus d'une fois, car on devra répéter certaines étapes plusieurs fois. On doit donc conserver une trace de ceux-ci. Pour ce faire, on n'est pas toujours dans l'obligation de garder chacun des ensembles de points intégralement. Par exemple, pour l'ensemble de Korobov, il suffit de sauvegarder le décalage aléatoire permettant de reconstruire l'ensemble, et non pas l'ensemble lui-même. On tire parti de la manière de générer les points pour ne pas trop utiliser d'espace mémoire comme avec les ensembles de points HUPS. Le nombre d'étapes n'étant pas connu à l'avance et pouvant être très grand, c'est une manière économique d'associer les ensembles $\tilde{P}_n$ aux différentes étapes.

Malgré toutes ces précautions, l'application directe de l'algorithme n'est pas adéquate. En effet, il y a un problème dans l'algorithme 8 dû à l'association dynamique des

Algorithme 8 Algorithme CFTP-Array-RQMC, mauvaise implantation

$t \leftarrow 0$

répéter

$t \leftarrow t+1$

$F^0_{-t,i} \leftarrow$ fonction identité sur $\{0,\ldots,m-1\}$ $\forall\, i = 0,\ldots,n-1$

Randomiser l'ensemble de points P_n pour obtenir l'ensemble $\tilde{P}_{n,-t} = (\tilde{U}_{0,-t},\ldots,\tilde{U}_{n-1,-t})$

pour $j = -t$ à -1 **faire**

pour i de 0 a $n-1$ **faire**

$f_{j,i}(\cdot) \leftarrow f(\cdot, \tilde{U}_{i,j})$

$F^{j+1}_{-t,i} \leftarrow f_{j,i} \circ F^{j}_{-t,i}$

fin pour

Trier (et renuméroter) les chaînes en ordre croissant de la valeur de $h(F^{j+1}_{-t,i})$

fin pour

jusqu'à $\forall\, i$ $F^0_{-t,i}(.)$ est constant

retourner l'ensemble des valeurs des images de $F^0_{-t,i}(.), i = 1,\ldots,n$ pour construire $\bar{\mu}^n_\infty$ l'estimateur de μ_∞

points. Lors de la simulation selon Array-RQMC, l'association des uniformes dépend des états des différentes méta-chaînes à chacune des étapes intermédiaires. Elle est donc déterminée par les tris effectués par l'algorithme. À chacun des passages $t \rightarrow t+1$ on recalcule toutes les étapes intermédiaires. Comme l'algorithme réinitialise l'ensemble des méta-chaînes et recalcule, en triant possiblement différemment à chacune des étapes intermédiaires, il n'est pas certain que les méta-chaînes réutilisent de la même manière les mêmes uniformes qu'aux cycles antérieurs $(t,\ldots,0)$. Voici un exemple où les uniformes sont utilisées différemment lorsque l'on change t.

Exemple problématique

Soit une chaîne de Markov définie par la matrice de transition suivante :

$$\begin{pmatrix} 0.5 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0.5 & 0 \\ 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0.5 \end{pmatrix}$$

La simulation de deux méta-chaînes (A et B) est effectuée en utilisant le couple de variables aléatoires $(U, 1-U)$, $U \sim$ Uniforme(0,1) à chaque étape. La fonction $h(\cdot)$ appliquée sur une méta-chaîne renvoie la moyenne des valeurs de ses différents états. Le dessin de la figure 4.1 illustre ce qui se passe pour les étapes $t = 1, 2, 3, 4$ avec l'algorithme 8.

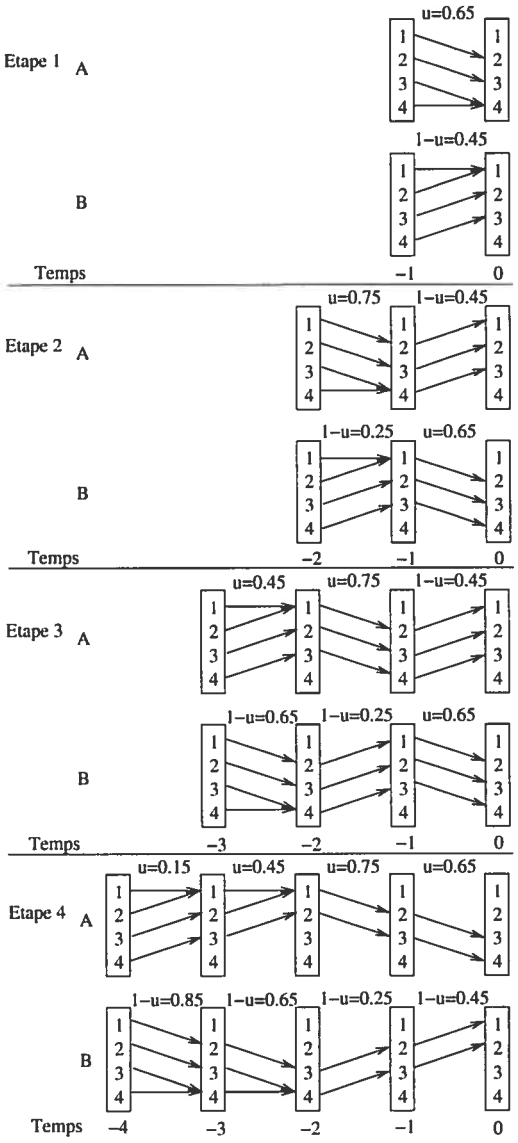

FIG. 4.1 – Illustration d'une mauvaise association des uniformes

Les quatre ensembles de points utilisés sont, en ordre croissant d'étape, (0.65, 0.45), (0.75, 0.25), (0.45, 0.65) et (0.15,0.85). La lettre à gauche des blocs représente le libellé de la méta-chaîne. L'uniforme est associée à la méta-chaîne selon la valeur de la fonction

$h(\cdot)$, la plus petite reçoit u et l'autre $1-u$. Les flèches représentent la fonction de transition $f(\cdot)$ appliquée selon la valeur au dessus.

Bien qu'à l'étape 2 on utilise les valeurs (0.75, 0.45) pour la méta-chaîne A et (0.25, 0.65) pour la B lorsque l'on passe à l'étape 4, les valeurs utilisées sont (0.15, 0.45, 0.75, 0.65) pour A et (0.85, 0.65, 0.25, 0.45) pour B. On ne retrouve donc pas les mêmes uniformes réutilisées pour faire les simulations des mêmes méta-chaînes. Il y a donc un biais dans notre estimateur final comme noté dans l'article de Propp et Wilson [35], car on ne réutilise pas les même uniformes aux même étapes pour les même méta-chaînes.

On peut penser que le problème se trouve au niveau du choix de la fonction de tri. Il existe bien sur des cas où cette manière de faire fonctionne quand même. On peut penser par exemple à une chaîne de Markov ayant une loi de probabilité stationnaire avec une fonction de transition de la forme $\varphi(x,u) = u$. En considérant une chaîne de Markov de cette forme, on obtient un estimateur non-biaisé. Mais la sous-famille de chaînes de Markov qui satisfont à cette condition est très restreinte. On veut trouver une combinaison permettant d'obtenir des échantillons non-biaisés peu importe la fonction de tri.

Tri unique par étape

On ne peut donc pas trier lors des étapes intermédiaires. On va conserver uniquement la transition totale du temps $-t$ à 0 et on va trier seulement après avoir effectué cette transition selon les valeurs des méta-chaînes au temps 0. L'état de la méta-chaîne i au cycle t est $F^0_{-t,i}$. On utilise la même définition pour la fonction $h(\cdot)$ que préalablement, soit la moyenne des valeurs de l'image de $F^0_{-t,i}$. La différence est que l'on ne l'évaluera pas pour toutes les étapes, mais simplement un fois par cycle au temps 0. Avec l'algorithme 9, on ne recalcule pas toutes les étapes précédentes, voir équation (2.16). On réduit ainsi le temps de simulation dans le cas où $\mathcal{S}$ est fini et on réutilise de la même façons les uniformes à chaque étape intermédiaire.

La version de l'algorithme pour l'évaluation des chaînes de Markov avec un espace d'états très grand, donné à l'algorithme 10,est légèrement différente. Comme on ne garde pas la trace de tout l'espace d'états, mais simplement celle du plus grand et plus petit état, on devra conserver toutes les fonctions de transition pour toutes les étapes in-

Algorithme 9 Algorithme CFTP-Array-RQMC, $\mathcal{S}$ fini, tri unique par cycle

$t \leftarrow 0$

$F^0_{t,i} \leftarrow$ fonction identité sur $\{0, \ldots, m-1\}$ $\forall\, i = 0, \ldots, n-1$

répéter

$t \leftarrow t+1$

Randomiser l'ensemble de points P_n pour obtenir l'ensemble $\tilde{P}_n = (\tilde{U}_0, \ldots, \tilde{U}_{n-1})$

pour i de 0 a $n-1$ **faire**

$f_{-t,i}(\cdot) \leftarrow f(\cdot, \tilde{U}_i)$

$F^0_{-t,i} \leftarrow F^0_{-t+1,i} \circ f_{-t,i}$

fin pour

Trier (et renuméroter) les chaînes en ordre croissant des valeurs de $h(F^0_{-t,i}(.))$

jusqu'à $\forall\, i$ $F^0_{-t,i}(.)$ est constant

retourner l'ensemble des valeurs des images de $F^0_{-t,i}(.)$ $i = 0, \ldots, n-1$ pour construire $\bar{\mu}^n_\infty$, l'estimateur de μ_∞

termédiaires. Pour un cycle t on va donc recalculer toutes les étapes intermédiaires des temps $-t$ à 0, comme avec l'algorithme 8, mais en évaluant la fonction $h(\cdot)$ seulement au temps 0. L'état de la méta-chaîne est composé de deux éléments comme dans l'algorithme 5 et la fonction $h(\cdot)$ est définie comme étant la moyenne entre des ces valeurs au temps 0.

Pour les algorithmes 9 et 10 on n'utilise pas des variables aléatoires indépendantes pour le calcul de l'estimateur. Par contre les uniformes servant au décalage aléatoire à chacune des étapes le sont. Pour une seule méta-chaîne la séquence de valeurs qu'elle utilise le sera aussi. Chacune des méta-chaînes évoluera donc à partir d'événements indépendants et les valeurs de retour seront générées selon la loi de probabilité stationnaire de la chaîne, voir théorème 2.3.5. L'estimateur calculé à partir des valeurs retournées par un des deux algorithmes sera sans biais.

Algorithme 10 Algorithme CFTP-Array-RQMC, $\mathcal{S}$ très grand, tri unique par cycle

$t \leftarrow 0$

répéter

$t \leftarrow t+1$

Randomiser l'ensemble de points P_n pour obtenir l'ensemble $\tilde{P}_{n,-t} = (\tilde{U}_{0,-t}, \ldots, \tilde{U}_{n-1,-t})$

$\mathcal{X}^i_{-t} \leftarrow (\widehat{0}, \widehat{1})$ pour tout $i = 0, \ldots, n-1$

pour i de 0 a $n-1$ **faire**

$f_{-t,i}(\cdot) \leftarrow f(\cdot, \tilde{U}_{i,-t})$

fin pour

pour $j = -t$ à -1 **faire**

pour i de 0 a $n-1$ **faire**

$\mathcal{X}^i_{j+1} \leftarrow f_{j,i}(\mathcal{X}^i_j)$

fin pour

fin pour

Trier (et renuméroter) les chaînes en ordre croissant des valeurs de $h(\mathcal{X}^i_0)$

jusqu'à $\mathcal{X}^i_{0,0} = \mathcal{X}^i_{0,1}$ pour tout $i = 0, \ldots, n-1$

retourner la moyenne $\bar{\mu}^n_\infty$ calculée à partir des valeurs des $\mathcal{X}^i_{0,0}$ pour $i = 0, \ldots, n-1$

4.2.2 Combinaison avec temps de départ

Difficulté des algorithmes 9 et 10

Une des difficultés retrouvées lorsque le nombre d'étapes devient très grand est que la dépendance négative entre les différentes réalisations n'est pas aussi forte que l'on aurait espéré. La figure 4.2 représente l'évolution d'un processus du temps $-t+1$ à 0. Les lignes représentent l'évolution du plus grand et plus petit état de l'espace d'états et elles bornent les valeurs possibles pour les autres états à chaque étape. Le couple ($\bar{a}$,$\underline{a}$) représente la valeur de la méta-chaîne au temps 0 pour l'étape $-t+1$ de l'algorithme 10. Supposons que l'on a plusieurs de ces processus en parallèle à un certain cycle t et que l'on veut les trier pour réduire la variance des valeurs obtenues dans la suite de l'exécution de l'algorithme 10. On sait, lors du calcul de l'évolution au cycle t, que le nouvel intervalle sera inclus dans l'ancien. Ce que l'on aimerait, c'est d'associer des

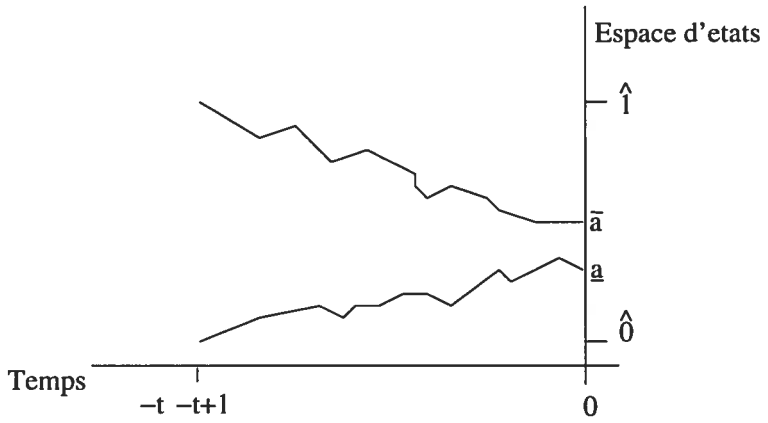

FIG. 4.2 – Exemple de l'évolution d'un processus d'un temps $-t+1$ à 0

uniformes au temps $-t$ tel que cela introduira une forte dépendance négative entre les processus au temps 0. Dans bien des cas plus t est grand, plus il est difficile de prévoir l'effet de l'uniforme associée au temps $-t$ sur les valeurs que nous allons obtenir au temps 0. Il est donc ardu de trouver une fonction de $h(\cdot)$ permettant une bonne association des uniformes lors d'un seul tri par cycle.

Par opposition, si on fait la simulation vers l'avant, l'intervalle des valeurs possibles à une certaine étape n'est pas nécessairement borné par l'intervalle de l'étape précédente. Il est plus facile de trouver une fonction de tri pour le choix des uniformes, même si le temps de simulation est très grand. Si on pouvait débuter la simulation assez loin dans le passé pour être assuré qu'au temps 0 toutes les méta-chaînes seraient composées d'une seule valeur on pourrait trier les méta-chaînes après chaque transition. Comme beaucoup de méta-chaînes seraient composées d'une seule valeur au temps 0, on obtiendrait beacoup de valeurs ayant une très grande dépendance négative. On espère ainsi définir plus facilement de bonne fonction de tri $h(\cdot)$ et parvenir à mieux introduire une dépendance négative entre les réalisations.

Temps de départ fixé

Soit un ensemble de n méta-chaînes dont on fait la simulation avec CFTP. On aimerait pouvoir déterminer une valeur T tel que toutes les n méta-chaînes auront atteint la limite au temps 0 si on débute la simulation au temps $-T$. On pourrait alors appliquer directement l'algorithme 8 pendant T étapes, stopper la simulation et conclure

que tous les échantillons obéissent à la bonne loi de probabilité stationnaire π.

Soit la variable aléatoire τ_i, l'instant de fusion de la simulation de la i^{eme} réalisation $i = 0, \ldots, n-1\}$. On définit $T^*_{\max} = \max\{\tau_i | i = 0, \ldots, n-1\}$ comme étant la valeur maximale de l'instant de fusion pour l'ensemble des n méta-chaînes. On ne peut pas déterminer une valeur de $T < \infty$ tel que $P(T^*_{\max} \leq T) = 1$ pour tout $i = 0, \ldots n-1$. Mais on peut se contenter d'une tolérance $\epsilon > 0$ telle que pour $T < \infty$ alors $P(T^*_{\max} \leq T) \geq 1 - \epsilon$. En acceptant cette tolérance sur le nombre de méta-chaînes n'ayant pas atteint leur état final, il peut y avoir une certaine proportion des réalisations qui n'auront pas fusionnés après T étapes. On ne peut pas abandonner la simulation de ces méta-chaînes, car on introduirait alors ce que l'on appelle le biais dû à l'impatience de l'observateur [35]. Par contre, on peut enchaîner avec l'algorithme 9 pour les méta-chaînes restantes. On obtiendra à la fin de la simulation un estimateur sans biais. Voici donc la nouvelle manière d'appliquer l'algorithme :

1. On fixe une valeur T de départ.
2. On fait la simulation du temps $-T$ jusqu'à 0 avec l'algorithme Array-RQMC standard sur les méta-chaînes pour obtenir un ensemble $\{F^0_{-T,0}, F^0_{-T,1}, \ldots, F^0_{-T,n-1}\}$
3. On reprend l'ensemble des valeurs obtenues à l'étape 2 comme ensemble initial de l'algorithme 9 et on applique celui-ci sur les méta-chaînes n'ayant pas atteint un état stationnaire.

Ceci nous donne l'algorithme 11 pour le cas où l'espace d'états est fini et l'algorithme 12 pour la cas où l'espace d'états est très grand et où la fonction $\varphi(\cdot)$ possède la propriété d'évaluation Monte Carlo monotone.

Pour le moment, le choix de la valeur de départ T n'est pas déterminée. Le prochain chapitre traitera de la manière de choisir la valeur de T et on illustrera à l'aide d'exemples l'effet de cette valeur sur le facteur de réduction de variance de l'algorithme.

4.3 Conclusion

On a vu les différentes combinaisons possible que l'on peut faire avec l'algorithme CFTP et quasi-Monte Carlo randomizé. Le mariage des deux techniques ne sait pas fait directement, mais on a pu obtenir quatre algortihmes. On pense que ces algorithmes vont

Algorithme 11 Algorithme CFTP-Array-RQMC, S fini, temps de départ fixé

$F_{-(T+1),i}^{-T} \leftarrow$ fonction identité sur $\mathcal{S}$ $\forall\ i = 0, \ldots, n-1$

pour $t = T$ à 1 **faire**

Randomiser l'ensemble de points P_n pour obtenir l'ensemble $\tilde{P}_n = (\tilde{U}_0, \ldots, \tilde{U}_{n-1})$

pour i de 0 a $n-1$ **faire**

$f_{-t,i}(\cdot) \leftarrow f(\cdot, \tilde{U}_i)$

$F_{-T,i}^{-t+1} \leftarrow f_{-t,i} \circ F_{-T,i}^{-t}$

Trier (et renuméroter) les chaînes en ordre croissant de la valeur de $h(F_{-T,i}^{-t})$

fin pour

fin pour

$t \leftarrow T$

répéter

$t \leftarrow t+1$

Randomiser l'ensemble de points P_n pour obtenir l'ensemble $\tilde{P}_n = (\tilde{U}_0, \ldots, \tilde{U}_{n-1})$

pour i de 0 a $n-1$ **faire**

$f_{-t,i}(\cdot) \leftarrow f(\cdot, \tilde{U}_i)$

$F_{-t,i}^{0} \leftarrow F_{-t+1,i}^{0} \circ f_{-t,i}$

fin pour

Trier (et renuméroter) les chaînes en ordre croissant des valeurs de $h(F_{-t,i}^{0}(.))$

jusqu'à $\forall\ i\ F_{-t,i}^{0}(.)$ est constant

retourner l'ensemble des valeurs des images de $F_{-t,i}^{0}(.)$ $i = 0, \ldots, n-1$ pour construire $\bar{\mu}_{\infty}^{n}$, l'estimateur de μ_{∞}

permettrent d'obtenir des estimateurs ayant une variance plus faible que ceux construits à l'aide des algorithmes du chapitre 2 et 3. Il est maintenant temps de l'illustrer avec des exemples numériques.

Algorithme 12 Algorithme CFTP-Array-RQMC, S très grand, temps de départ fixé

$\mathcal{X}_{-T}^i \leftarrow (\widehat{0}, \widehat{1})$ pour tout $i = 0, \ldots, n-1$

pour $t = T$ à 1 **faire**

Randomiser l'ensemble de points P_n pour obtenir l'ensemble $\tilde{P}_n = (\tilde{U}_0, \ldots, \tilde{U}_{n-1})$

pour i de 0 a $n-1$ **faire**

$f_{-t,i}(\cdot) \leftarrow f(\cdot, \tilde{U}_{i,-t})$

$\mathcal{X}_{-t+1}^i \leftarrow f_{-t,i}(\mathcal{X}_{-t}^i)$

Trier (et renuméroter) les chaînes en ordre croissant de la valeur de $h(\mathcal{X}_{-t+1}^i)$

fin pour

fin pour

$t \leftarrow T$

répéter

$t \leftarrow t+1$

Randomiser l'ensemble de points P_n pour obtenir l'ensemble $\tilde{P}_{n,-t} = (\tilde{U}_{0,-t}, \ldots, \tilde{U}_{n-1,-t})$

$\mathcal{X}_{-t}^i \leftarrow (\widehat{0}, \widehat{1})$ pour tout $i = 0, \ldots, n-1$

pour i de 0 a $n-1$ **faire**

$f_{-t,i}(\cdot) \leftarrow f(\cdot, \tilde{U}_{i,-t})$

fin pour

pour $j = -t$ à -1 **faire**

pour i de 0 a $n-1$ **faire**

$\mathcal{X}_{j+1}^i \leftarrow f_{j,i}(\mathcal{X}_j^i)$

fin pour

fin pour

Trier (et renuméroter) les chaînes en ordre croissant des valeurs de $h(\mathcal{X}_0^i)$

jusqu'à $\mathcal{X}_{0,0}^i = \mathcal{X}_{0,1}^i$ pour tout $i = 0, \ldots, n-1$

retourner l'ensemble des valeurs $\mathcal{X}_{0,0}^i$ pour $i = 0, \ldots, n-1$ pour construire $\bar{\mu}_\infty^n$, l'estimateur de μ_∞

Chapitre 5

Exemples numériques

5.1 Espace d'états fini

Le cas le plus simple à observer est bien sûr le cas où l'espace d'états est fini et petit. Pour comparer la différence d'efficacité avec les méthodes vues précédemment, on fera les simulations avec les mêmes matrices et fonctions que celles utilisées dans les articles de Craiu et Meng [3], et Lemieux et Sidosky [24]. Chaque matrice stochastique de la figure 5.1 définit un espace d'états et les probabilités de transition pour trois chaînes de Markov différentes.

Matrice 1

$$P^1 = \begin{pmatrix} 0.5 & 0.4 & 0.1 \\ 0.3 & 0.4 & 0.3 \\ 0.2 & 0.3 & 0.5 \end{pmatrix}$$

Matrice 2

$$P^2 = \begin{pmatrix} 0.7 & 0.0 & 0.3 & 0.0 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.4 & 0.0 & 0.6 \\ 0.0 & 0.2 & 0.0 & 0.8 \end{pmatrix}$$

Matrice 3

$$P^3_{i,j} = \begin{cases} 0.2 & \text{Si } i+1 = j \leq N \text{ ou } i = j = N \\ 0.8 & \text{Si } i-1 = j \leq N \text{ ou } i = j = 1 \\ 0 & \text{Sinon} \end{cases}$$

FIG. 5.1 – Exemples de matrices de transition

On évaluera, pour chacune des chaînes, les trois fonctions de coût par état

$$C_1(x) = x, \qquad C_2(x) = (x-2)(x-5) \qquad \text{et } C_3(x) = \sin(3x)$$

servant à calculer la valeur de l'estimateur défini par l'équation (2.10).

5.1.1 Algorithme 9 avec tri unique par cycle

Pour chacune des chaînes de Markov définies à la figure 5.1 on fera la simulation à l'aide de l'algorithme 9 et on calculera un estimateur de μ_∞, défini à l'équation (2.8), à partir de l'équation (2.10). La fonction $h(\cdot)$ est définie comme la moyenne des valeurs de l'état de la méta-chaîne. Le VRF a été choisie comme premier élément de comparaison. Le coût de calcul par ordinateur est dépendant de l'implanation et ne reflète pas toujours fidèlement le coût théorique des algorithmes. On donnera quand même une idée de la valeur de ce coût obtenu lors des simulations. On a calculé le VRF pour chaque estimateur, pour chacune des fonctions de coût avec chacune des trois chaînes de Markov. L'estimateur Monte Carlo de base $\bar{\mu}_\infty^n$ a été obtenu à partir des résultats de simulation de 2^{20} répétitions indépendantes. Les événements durant la simulation sont générés à partir des valeurs uniformes produites par la classe MRG32k3a implantée dans SSJ [18]. Les moyennes $\bar{\mu}_\infty^n$ et variances $\bar{\sigma}_\infty^2$ empiriques des estimateurs Monte Carlo pour les différents estimateurs calculés pour chacune des matrices et fonctions de coût sont résumées dans le tableau 5.1 :

		P^1	P^2	P^3
C_1	$\bar{\mu}_\infty^n$	0.9508	1.8009	0.3326
	$\bar{\sigma}_\infty^2$	0.6262	1.5606	0.4425
C_2	$\bar{\mu}_\infty^n$	4.8663	2.2018	8.2250
	$\bar{\sigma}_\infty^2$	16.0843	24.3650	10.4509
C_3	$\bar{\mu}_\infty^n$	-0.0289	0.1646	0.0170
	$\bar{\sigma}_\infty^2$	0.0292	0.0640	0.0104

TAB. 5.1 – Moyennes et variances empiriques pour l'estimateur Monte Carlo

Le tableau 5.2 contient les facteurs de réduction de variance obtenus avec l'algorithme 9. La valeur du nombre de méta-chaînes est indiquée en haut de chacune des colonnes ainsi que les multiplicateurs utilisés pour générer les différents ensembles de points. La

valeur de la variance du nouvel estimateur est estimée à partir de l'équation (3.6) avec une valeur de r=100.

On note une augmentation du facteur de réduction de variance pour les estimateurs produits par la combinaison avec array-RQMC, en comparaison avec la méthode RQMC classique.

	Nombre de méta-chaînes	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
	a Classical-Korobov-Baker	306	1397	5693	944	118068	802275
	a Array-Korobov-Baker	633	2531	10125	40503	162013	648055
	Nb chaînes Korobov	1021	4093	16381	65521	262139	1048573
Matrice P^1							
C_1	Classical-Korobov-Baker	23	50	34	68	106	339
	Array-Korobov-Baker	240	826	1951	4886	2638	21560
	Array-Sobol	155	378	1281	4882	16150	51470
C_2	Classical-Korobov-Baker	33	51	25	43	102	342
	Array-Korobov-Baker	199	720	1820	5467	2962	25260
	Array-Sobol	123	540	1483	3982	16910	41410
C_3	Classical-Korobov-Baker	17	30	36	26	25	40
	Array-Korobov-Baker	97	317	1067	2380	1464	5763
	Array-Sobol	63	156	503	1015	6496	19150
Matrice P^2							
C_1	Classical-Korobov-Baker	5	12	3	17	44	24
	Array-Korobov-Baker	32	59	186	607	774	3555
	Array-Sobol	22	46	104	285	812	2031
C_2	Classical-Korobov-Baker	5	10	2	12	31	20
	Array-Korobov-Baker	33	62	211	491	585	3165
	Array-Sobol	23	48	100	282	742	2228
C_3	Classical-Korobov-Baker	5	8	3	14	33	18
	Array-Korobov-Baker	31	78	230	596	673	3690
	Array-Sobol	15	68	101	273	673	1912
Matrice P^3							
C_1	Classical-Korobov-Baker	7	13	26	47	54	72
	Array-Korobov-Baker	39	84	155	345	598	1484
	Array-Sobol	26	49	102	220	559	1108
C_2	Classical-Korobov-Baker	12	16	16	38	37	68
	Array-Korobov-Baker	51	114	265	587	858	2140
	Array-Sobol	34	64	172	369	917	1718
C_3	Classical-Korobov-Baker	1	2	3	6	6	7
	Array-Korobov-Baker	4	8	15	22	53	101
	Array-Sobol	3	4	9	14	34	69

TAB. 5.2 – Facteurs de réduction de la variance avec l'algorithme 9

Pour la méthode RQMC classique, on utilise un ensemble de Korobov à dimension

infinie, noté Classical-Korobov-Baker. Le multiplicateur a se trouve en haut de chaque colonne à la ligne a Classical-Korobov-Baker. Ils sont tirés de la Table 1 de L'Écuyer et Lemieux [22]. On a ensuite appliqué un décalage aléatoire modulo 1 (voir la sous-section 3.2.5) suivi d'une transformation du pâtissier. Cette tranformation consiste à simplement remplacer la valeur des uniformes u par $2u$ si $u \leq 1/2$ et $2(1-u)$ si $u > 1/2$.

Pour la méthode array-RQMC, on utilise deux ensembles de points en une seule dimension. Le premier est un ensemble de Korobov en une dimension, noté Array-Korobov-Baker auquel on a aussi appliqué un décalage aléatoire modulo 1 et une transformation du pâtissier. Le multiplicateur a se trouve aussi en haut de chaque colonne à la ligne a Array-Korobov-Baker. Le deuxième et un ensemble de Sobol en une dimension, noté Array-Sobol. Il consiste à prendre les n premiers points de la séquence de Sobol randomisé par une matrice brouillante suivi d'un décalage aléatoire digital.

Les résultats obtenus pour l'algorithme 9 sont meilleurs que ceux obtenus par la méthode RQMC standard et ce par un très grand facteur. On obtient jusqu'à 150 fois plus de réduction de la variance avec notre nouvel algorithme.

5.1.2 Algorithme 11 avec temps de départ fixé

On a déjà abordé le sujet de débuter avec une valeur T de temps loin dans le passé lors du premier cycle afin d'augmenter le facteur de réduction de la variance de notre estimateur. Mais comment doit-on choisir la valeur de T ? Si, par exemple, le nombre d'étapes moyen avant la fusion est de 10, laquelle des deux valeurs $T = 1000$ ou $T = 10000$ est la meilleure ? Voici plusieurs graphiques où l'on a appliqué l'idée discutée à la sous-section 4.2.2 pour différentes valeurs de T sur les exemples précédents. Dans les graphiques des figures 5.2, 5.3, et 5.4 on peut voir la valeur du facteur d'augmentation de réduction de la variance en ordonnée, calculé selon la valeur initiale de T en abscisse. La courbe pleine représente les valeurs des VRF estimées en fonction de celle de T lorsque l'on fait 100 répétitions indépendantes de l'algorithme avec n=1021 méta-chaînes avec comme ensemble de points une règle de Korobov suivi d'une transformation du pâtissier. La ligne discontinue représente les valeurs des VRF en fonction de T calculées à partir de 100 répétitions indépendantes de l'algorithme avec n=1024 méta-chaînes avec comme ensemble de points celui de Sobol randomisé à l'aide d'une matrice brouillante et d'un

décalage aléatoire digital. Les graphiques sont regroupés en groupe de trois. Dans chaque figure on peut voir le graphique des trois fonctions de coût pour une seule matrice. La fonction de coût impliquée dans le calcul de l'estimateur est, de gauche à droite, C_1, C_2 et C_3. La valeur de T varie de 0 à 50 pour les matrice P^1 et P^2 et 0 à 100 pour la matrice P^3.

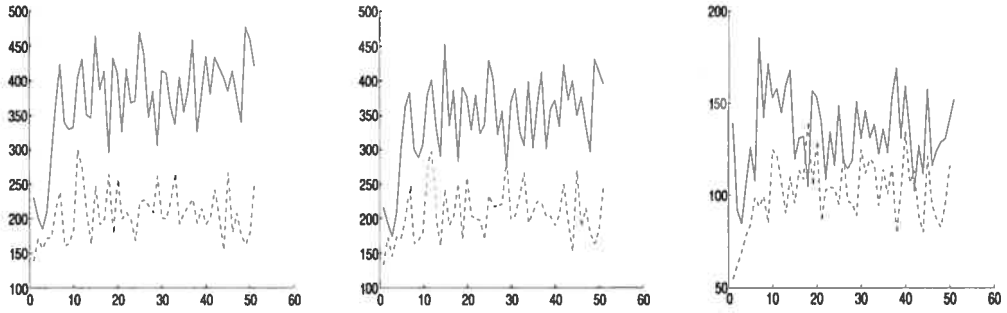

FIG. 5.2 – Graphique de la valeur du VRF par rapport à T pour P^1

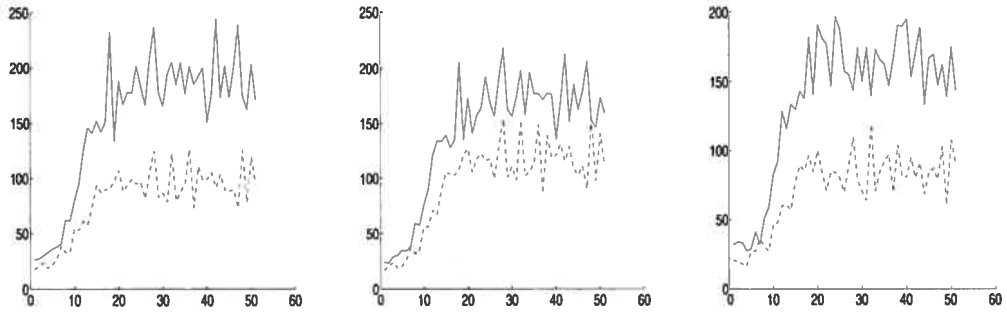

FIG. 5.3 – Graphique de la valeur du VRF par rapport à T pour P^2

On peut voir une tendance à l'augmentation du VRF pour les premières valeurs de T pour ensuite converger vers une certaine valeur. Le temps pour obtenir cette valeur semble dépendre de la matrice. En examinant les graphiques on peut déduire que, pratiquement, on atteint un maximum près des valeurs de T égales à 10, 20, 60, respectivement pour les matrice P^1, P^2 et P^3.

Si on augmente la valeur de départ T, certaines réalisations seront simulées plus

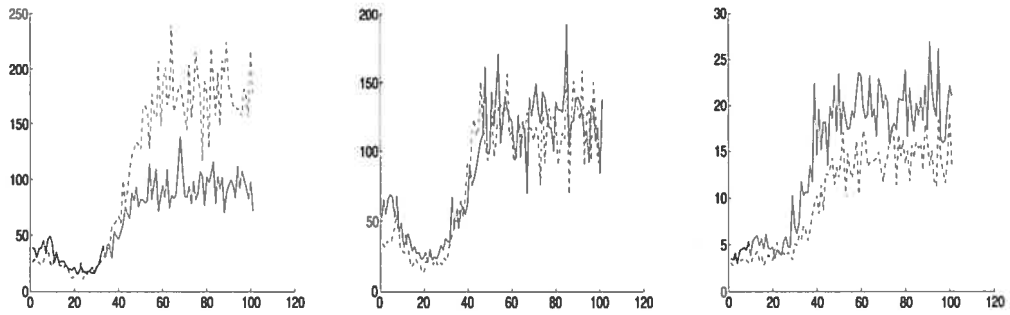

FIG. 5.4 – Graphique de la valeur du VRF par rapport à T pour P^3

longtemps que nécessaire. On calculera donc, pour ces réalisations, plus de fonctions de transition. On doit maintenant tenir compte de ce nouveau facteur, le temps de calcul, dans la comparaison de l'efficacité de notre algorithme. Pour ce faire on calcule le facteur suivant :

$$\frac{\text{Coût unitaire moyen pour Monte-Carlo}}{\text{Coût unitaire moyen pour Array-RQMC-CFTP}} \tag{5.1}$$

et multiplier le VRF par ce nombre. Dans les graphiques des figures 5.5, 5.6 et 5.7 on peut voir la valeur du facteur d'augmentation de l'efficacité en ordonnée, calculé selon la valeur initiale de T en abscisse. Pour calculer l'efficacité, on utilise l'équation (1.6), pour

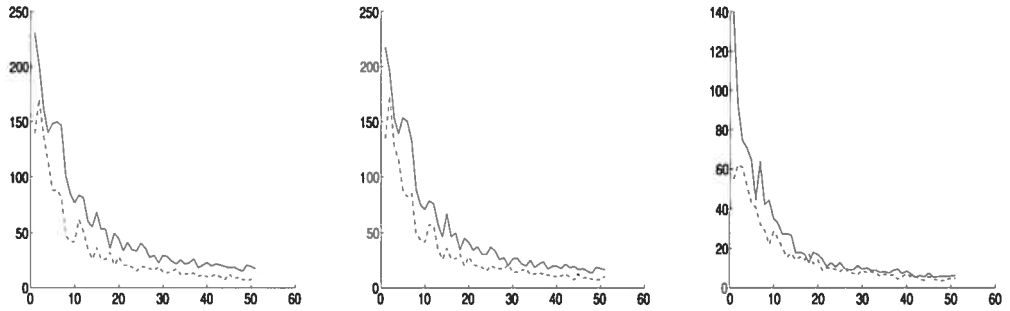

FIG. 5.5 – Augmentation de l'efficacité selon T pour P^1

tous les estimateurs avec comme fonction de coût le nombre de fonctions de transition calculées. Pour calculer le facteur d'augmentation on fait ensuite le ratio entre l'efficacité de l'estimateur Monte Carlo et celle calculée avec la nouvelle méthode. On a multiplié les valeurs des VRF obtenues précédemment par le rapport du nombre de fonctions de transition calculé par l'algorithme 11 et celui calculé en utilisant la méthode Monte

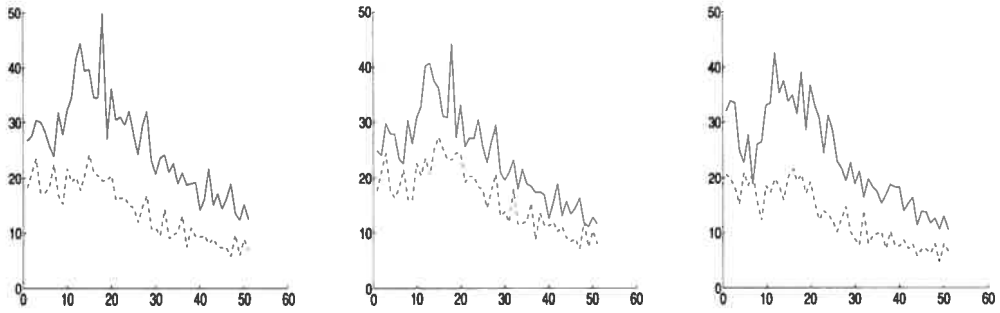

FIG. 5.6 – Augmentation de l'efficacité selon T pour P^2

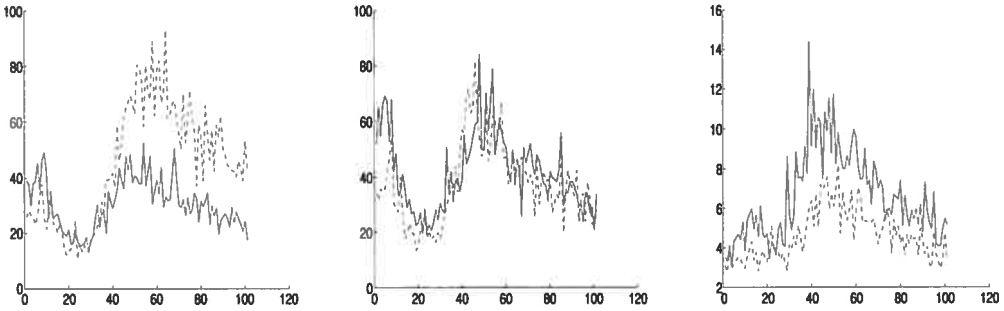

FIG. 5.7 – Augmentation de l'efficacité selon T pour P^3

Carlo standard. La courbe pleine représente les valeurs du facteur d'augmentation de l'efficacité selon T lorsqu'on utilise l'ensemble de Korobov. La ligne discontinue sont les valeurs selon T calculées à partir de l'ensemble de points de Sobol. Les intervalles dans lequels T prend ses valeurs sont les mêmes que pour les graphiques 5.2, 5.3, et 5.4.

Si le seul facteur de comparaison pour le coût de calcul supplémentaire est le nombre de fonctions de transitions, alors les gains observés dans les graphiques des figures 5.5, 5.6 et 5.7 ne reflètent pas fidèlement les véritables gains. Il y a d'autres opérations, comme le tri, qui ne sont pas prises en compte lors du calcul du coût de l'algortihme 11. Ce coût dépend de la complexité des états et de la fonction $h(\cdot)$ et il fait habituellement augmenter le coût de calcul, surtout pour les petites valeurs de T. Dans les figures 5.8, 5.9 et 5.10 on retrouve les graphiques du facteur d'augmentation de l'efficacité calculé avec les mêmes facteurs de réduction de variance qu'auparavant, mais en prenant le temps de calcul CPU moyen au lieu du nombre de fonctions de transition moyen calculé dans l'équation du coût de l'estimateur.

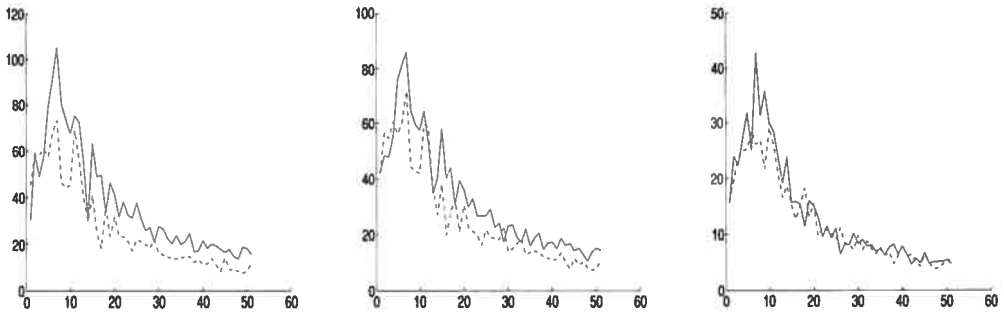

FIG. 5.8 – Augmentation de l'efficacité selon T pour P^1

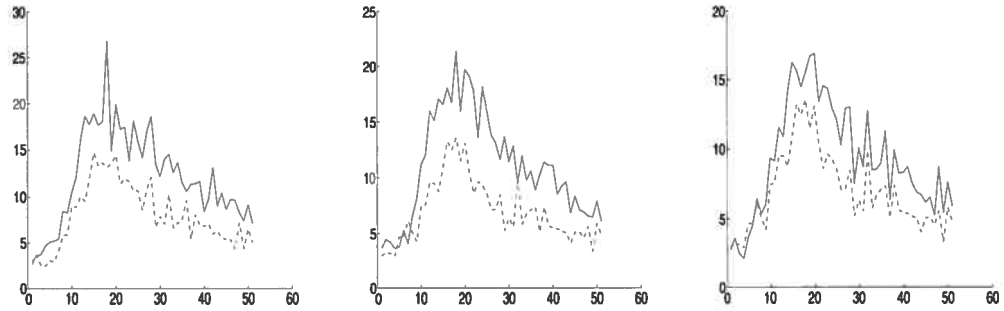

FIG. 5.9 – Augmentation de l'efficacité selon T pour P^2

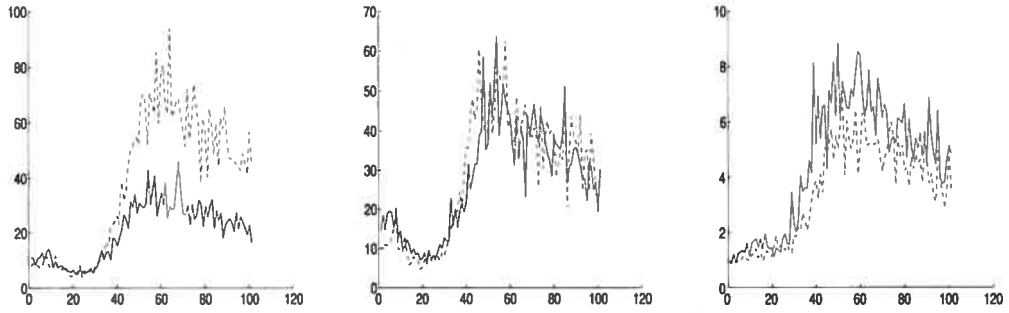

FIG. 5.10 – Augmentation de l'efficacité selon T pour P^3

Les valeurs où on atteint le maximum d'efficacité sont à peu près les même que dans les graphiques des figures 5.2, 5.3 et 5.4. Il y a une différence avec ceux prenant en compte le nombre de fonctions de transition calculées. Comme l'espace d'états est petit,

le coût de calcul des fonctions de transitions est relativement faible en comparaison du tri. La différence de la forme des courbes pour de petites valeur s'explique facilement par le coût d'initialisation qui est grand, même pour de petites valeurs de T. On observe encore une augmentation en fonction de la valeur de départ T pour les premières valeurs comme pour le calcul du VRF. Pour trouver une valeur adéquate pour T, on doit donc faire un compromis entre le coût d'exécution et le facteur de réduction de la variance peu importe le critère choisi (nombre de fonction de transition / temps CPU). On suggère de prendre une valeur de T tel qu'il ne reste en moyenne que 10^{-4} des chaînes à simuler. Voici des graphiques, à la figure 5.11, représentant le pourcentage de méta-chaînes ayant fusionnées selon la valeur de T. On y voit, de gauche à droite, les résultats pour les matrices P^1, P^2 et P^3. Les calculs ont été réalisés avec 2^{20} méta-chaînes pour chacune des matrices.

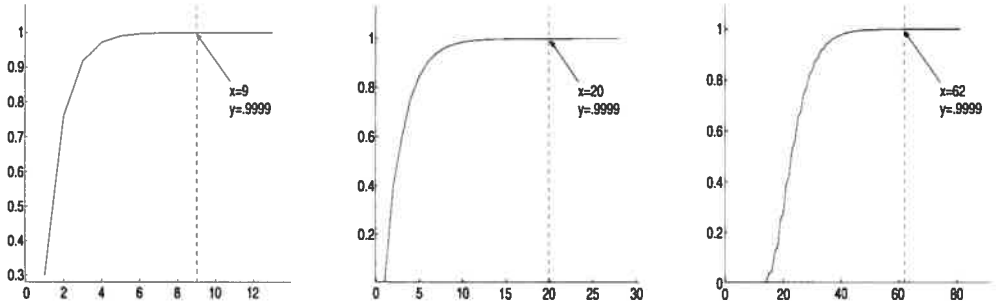

FIG. 5.11 – Graphiques du pourcentage de méta-chaînes fusionnées selon T

On remarque que pour les valeurs 9, 20, et 62, celles où l'on obtient un maximum d'efficacité, la presque totalité des chaînes sont fusionnées. En fait, dans les trois cas on s'approche du maximum d'augmentation de l'efficacité quand moins de 10^{-4} des chaînes ne se sont pas fusionnées.

Pour trouver une valeur adéquate de T pour nos exemples, on peut procéder de la manière suivante :

1. Supposons que l'on veut faire r répétitions de l'algorithme avec n méta-chaînes.
2. On va faire des expériences pilotes avec n méta-chaînes pour estimer la valeur de T nécessaire pour qu'à peu près moins de 10^{-4} des n méta-chaînes ne se soient pas fusionnées.

3. Ensuite, on fait r répétitions indépendantes avec comme valeur de T de départ celle trouvée à l'étape 2 pour construire un estimateur.

Ce que l'on recherche, c'est que la presque totalité des méta-chaînes se soient fusionnées après T étapes. La valeur 10^{-4} n'est pas absolue et doit être choisie en fonction de l'exemple. Pour les exemples de la figure 5.1 c'est environ cette valeur que l'on a obtenue. Après l'étape 2, on obtient une valeur de départ T qui permet de réduire considérablement la variance. On espère donc de ne pas perdre trop de performance au niveau du temps de simulation. On retrouve dans le tableau 5.3 les résultats obtenus lorsque l'on applique cette méthode. Les valeurs du tableau 5.3 sont celles du facteurs de réduction de la variance. Pour obtenir le véritable gain d'efficacité, en tenant compte du nombre de fonctions de transitions calculées, on doit calculer le travail supplémentaire que l'on fait avec l'algorithme 11 tel que définie par l'équation 5.1. On obtient alors les facteurs : 2/9 pour la première matrice, 3.8/20 pour la deuxième matrice et 24/62 pour la troisième matrice. Pour les facteurs obtenus avec l'algorithme RQMC classique, les facteurs sont tous de 1. Le temps de calcul CPU est plus difficile à évaluer et varie beaucoup d'un exemple à l'autre et aussi en fonction de la valeur de départ T ainsi que le nombre de méta-chaînes n. Ce temps de calcul rajoute un travail variant de l'ordre de 3 à 100 fois plus.

Les résultats de la table 5.4, l'efficacité quand l'on tient compte du temps de calculs CPU, peuvent sembler un peu moins intéressants que précédemment lorsque l'on utilise le nombre de fonctions de transitions calculées pour déterminer le travail supplémentaire accompli. Pour certains exemples, les résultats obtenus avec la méthode RQMC sont meilleurs que ceux produits à l'aide de la combinaison avec Array-RQMC. Cela arrive seulement dans les exemples utilisant la matrice P^1 ou la matrice P^2 et ce pour un petit nombre de chaînes. Mais quand le nombre de chaînes et le nombre d'étapes moyen avant fusion augmente, matrice P^3 par exemple, l'algorithme 11 produit de meilleurs estimateurs.

	Nombre de méta-chaînes	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
	a Classical-Korobov-Baker	306	1397	5693	944	118068	802275
	a Array-Korobov-Baker	633	2531	10125	40503	162013	648055
	Nb chaînes Korobov	1021	4093	16381	65521	262139	1048573
Matrice P^1							
C_1	Classical-Korobov-Baker	23	50	34	68	106	339
	Array-Korobov-Baker	318	670	3318	558	28720	37540
	Array-Sobol	190	627	2353	7469	18790	61600
C_2	Classical-Korobov-Baker	33	51	25	43	102	342
	Array-Korobov-Baker	378	468	1873	513	26300	45200
	Array-Sobol	225	705	2692	6733	28060	70970
C_3	Classical-Korobov-Baker	17	30	36	26	25	40
	Array-Korobov-Baker	178	105	1079	1364	7777	9406
	Array-Sobol	97	322	800	3954	14880	42110
Matrice P^2							
C_1	Classical-Korobov-Baker	5	12	3	17	44	24
	Array-Korobov-Baker	162	327	1304	373	6953	7530
	Array-Sobol	82	313	976	3254	10130	26340
C_2	Classical-Korobov-Baker	5	10	2	12	31	20
	Array-Korobov-Baker	224	288	1074	386	536	8640
	Array-Sobol	139	342	1312	2935	8959	20980
C_3	Classical-Korobov-Baker	5	8	3	14	33	18
	Array-Korobov-Baker	198	229	933	682	5632	11070
	Array-Sobol	74	276	852	2693	6397	22990
Matrice P^3							
C_1	Classical-Korobov-Baker	7	13	26	47	54	72
	Array-Korobov-Baker	84	149	619	3663	2380	5444
	Array-Sobol	168	463	1995	6583	13500	37020
C_2	Classical-Korobov-Baker	12	16	16	38	37	68
	Array-Korobov-Baker	119	208	567	2561	2431	3204
	Array-Sobol	107	3324	1106	3886	8486	23320
C_3	Classical-Korobov-Baker	1	2	3	6	6	7
	Array-Korobov-Baker	17	34	109	286	318	766
	Array-Sobol	14	38	151	371	1078	3144

TAB. 5.3 – Facteurs de réduction de la variance avec l'algorithme 11

5.2 Espace d'états très grand

La section précédente explore une manière de combiner l'algorithme Array-RQMC et CFTP pour les chaînes de Markov avec un espace d'états fini où les méta-chaînes sont composées d'un nombre d'éléments égal au nombre d'états possible de l'espace $\mathcal{S}$. Dans le cas où l'espace d'états est très grand, les choses sont différentes. Une autre approche a

	Nombre de méta-chaînes	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}
	a Classical-Korobov-Baker	306	1397	5693	944	118068	802275
	a Array-Korobov-Baker	633	2531	10125	40503	162013	648055
	Nb chaînes Korobov	1021	4093	16381	65521	262139	1048573
Matrice P^1							
C_1	Classical-Korobov-Baker	23	50	34	68	106	339
	Array-Korobov-Baker	28	59	126	18	556	536
	Array-Sobol	28	67	114	224	391	965
C_2	Classical-Korobov-Baker	33	51	25	43	102	342
	Array-Korobov-Baker	55	40	63	13	496	492
	Array-Sobol	31	68	111	210	584	795
C_3	Classical-Korobov-Baker	17	30	36	26	25	40
	Array-Korobov-Baker	23	8	51	50	198	158
	Array-Sobol	15	29	48	151	368	763
Matrice P^2							
C_1	Classical-Korobov-Baker	5	12	3	17	44	24
	Array-Korobov-Baker	14	18	37	8	99	72
	Array-Sobol	8	20	29	69	161	297
C_2	Classical-Korobov-Baker	5	10	2	12	31	20
	Array-Korobov-Baker	12	18	28	8	8	107
	Array-Sobol	17	26	41	66	149	197
C_3	Classical-Korobov-Baker	5	8	3	14	33	18
	Array-Korobov-Baker	22	15	29	16	100	156
	Array-Sobol	5	20	36	66	123	343
Matrice P^3							
C_1	Classical-Korobov-Baker	7	13	26	47	54	72
	Array-Korobov-Baker	30	33	106	412	210	343
	Array-Sobol	70	126	319	712	1242	2264
C_2	Classical-Korobov-Baker	12	16	16	38	37	68
	Array-Korobov-Baker	42	48	97	312	216	208
	Array-Sobol	46	942	189	482	827	1582
C_3	Classical-Korobov-Baker	1	2	3	6	6	7
	Array-Korobov-Baker	6	8	18	38	29	48
	Array-Sobol	6	10	25	52	98	208

TAB. 5.4 – Facteurs d'augmentation de l'efficacité avec l'algorithme 11, temps CPU

été précédemment explorée pour ce cas, comme on peut le voir dans l'algorithme 5. On demande simplement que la fonction de transition ait la propriété d'évaluation Monte Carlo monotone (définition 2.3.5). Cela permet de faire l'échantillonnage pour le cas infini et d'accélérer la simulation pour le cas fini avec un espace d'états très grand.

Exemple

On va définir une chaîne de Markov sur un espace d'états infini. L'exemple choisi est une marche aléatoire dans un espace continu borné. L'espace d'états de la chaîne de Markov est $\mathcal{S} = [0, \overline{W}]$, $\overline{W} > 0$. Soit W_t la valeur du t^{eme} état de la chaîne, $t \geq 0$. Les W_t satisfont la récurrence stochastique :

$$W_0 = 0 \text{ et } W_t = \max(0, \min(W_{t-1} + Z_t, \overline{W})) \qquad \text{pour } t \geq 1.$$

$$Z_t \sim \text{Normal}(0, \sigma^2)$$

où Normal$(0, \sigma)$ représente la distribution normale de moyenne 0 et d'écart-type σ. La fonction de coût est définie comme étant $C(x) = x$. La méta-chaîne associée $\mathcal{X}$ est composé de deux éléments et évolue selon la récurrence

$$\mathcal{X}_0 = (\mathcal{X}_0(0), \mathcal{X}_0(1)), \qquad \text{où } \mathcal{X}_0(0) = 0 \text{ et } \mathcal{X}_0(1) = \overline{W},$$

$$\mathcal{X}_{t+1} = (\max(0, \min(\mathcal{X}_t(0) + Z_t, \overline{W})), \max(0, \min(\mathcal{X}_t(1) + Z_t, \overline{W}))), \qquad t \geq 1$$

$$Z_t \sim \text{Normal}(0, \sigma^2)$$

On va comparer l'application des deux algorithmes 10, avec un tri unique par cycle et 12 avec temps de départ fixé. On va changer la variance des Z_t pour obtenir 3 exemples. Les trois différentes valeurs de σ^2 employées sont 5, 1 et 1/2.

5.2.1 Algorithme 10 avec tri unique par cycle

Le tableau suivant contient la valeur du facteur de réduction de la variance obtenu avec l'algorithme 10. Les valeurs du multiplicateur a pour les ensembles de Korobov sont les mêmes que dans le tableau 5.1.1. Les valeurs dans la colonne de gauche représentent les paramètres de la chaîne de Markov. On obtient, avec l'algorithme 10, un estimateur non-biaisé. Mais, les résultats semblent être moins intéressants par rapport à RQMC classique, surtout lorsque le nombre de méta-chaînes augmente. Le nombre d'étapes est bien plus grand que dans les exemples précédents.

La fonction $h(\cdot)$ est encore définie comme étant la moyenne des valeurs des états de la méta-chaîne au temps 0, dans le cas ici la moyenne correspond aussi à la valeur médiane de l'intervalle. On peut donc avoir la même valeur de $h(\cdot)$ pour plusieurs intervalles

Paramètres	Nombre de méta-chaînes	2^{10}	2^{12}	2^{14}
σ^2=5	Classical-Korobov-Baker	17	42	14
$\overline{W}$=5	Array-Korobov-Baker	28	34	28
	Array-Sobol	34	32	31
σ^2=1	Classical-Korobov-Baker	11	13	18
$\overline{W}$=5	Array-Korobov-Baker	7	12	10
	Array-Sobol	10	9	10
σ^2=0.5	Classical-Korobov-Baker	5	2	9
$\overline{W}$=5	Array-Korobov-Baker	1	1	1
	Array-Sobol	1	1	1

TAB. 5.5 – Facteurs de réduction de la variance avec l'algorithme 10

différents. Il n'est donc pas évident de savoir comment trier les méta-chaînes aussi efficacement que dans la simulation d'une seule chaîne. La problématique de trouver une fonction de tri $h(\cdot)$ devient très complexe et c'est alors plus difficile d'introduire une corrélation négative entre les différentes réalisations.

5.2.2 Algorithme 12 avec temps de départ fixé

Pour tenter d'obtenir de meilleurs résultats de performance pour les chaînes de Markov ayant un espace d'états très grand, on va utiliser la même heuristique que dans la section 4.2.2. On va tenter de trouver une valeur initiale T permettant d'obtenir une augmentation du VRF. Si le pourcentage de méta-chaînes nécessitant encore du temps de simulation est faible, on peut alors s'attendre a réduire de manière significative le temps de caucul.

Le tableau 5.6 montre les résultats obtenus lorsque l'on fixe une valeur initiale T. Les résultats sont obtenus à l'aide de 100 répétitions de l'algorithme avec l'heuristique. On utilise des expériences pilotes pour tenter de trouver une valeur adéquate pour T. La valeur de T correspond au temps nécessaire pour que moins de 10^{-4} méta-chaînes n'aient pas fusionnées durant les expériences pilotes. Le facteur de réduction de la variance a augmenté considérablement par rapport au tableau 5.5. On se retrouve dans une situation semblable à celle des exemples précédents quand à la performance de l'algorithme. Bien qu'on obtienne de très bonnes valeurs de VRF pour notre algorithme par rapport à la méthode RQMC classique, on doit tenir compte du temps nécessaire de calcul pour produire nos estimateurs. Si on utilise comme temps de calcul le nombre

Paramètres	Nombre de méta-chaînes	2^{10}		2^{12}		2^{14}	
		T	VRF	T	VRF	T	VRF
σ^2=5	Classical-Korobov-Baker		17		42		14
$\overline{W}$=5	Array-Korobov-Baker	10	1816	10	11420	12	34790
	Array-Sobol	9	12770	11	90380	12	473800
σ^2=1	Classical-Korobov-Baker		11		13		18
$\overline{W}$=5	Array-Korobov-Baker	81	70	80	5818	98	22580
	Array-Sobol	86	3525	86	16840	92	95840
σ^2=0.5	Classical-Korobov-Baker		5		2		9
$\overline{W}$=5	Array-Korobov-Baker	303	107	319	10830	298	47320
	Array-Sobol	302	2969	326	18820	326	86480

TAB. 5.6 – Facteurs de réduction de la variance avec l'algorithme 12

de fonctions de transition, il faut simplement diviser par un facteur d'environ 2 pour obtenir le gain d'efficacité de l'algorithme 12 dans tous les cas.

La situation est légèrement différente de celle où l'espace d'états est fini et petit. Dans le cas présent, pour un certain cycle de l'algorithme, on recalcule toutes les fonctions de transitions intermédiaires jusqu'au temps 0 et ce même pour toutes les méthodes. Le temps de départ à donc une plus grande influence. Le nombres de fonctions de transition calculées durant l'exécution de l'algorithme n'augmente pas proportionnellement par rapport à la valeur de T comme dans l'algorithme 9. Pour certaines valeurs de T, on peut diminuer le nombres d'évaluations et ainsi réduire le coût de simulation. Cela laisse croire que les gains obtenus à l'aide la nouvelle combinaison avec Array-RQMC seront plus significatifs que dans le cas où l'espace d'états n'est pas ordonné.

On a suggéré dans la sous-section 2.3.4 d'utiliser une séquence de puissances de 2 comme l'incrément de la valeur de t dans l'algorithme 2.5. Cette méthode permet de réduire le coût d'évaluation, mais ne minimise pas le nombre de fonctions de transition calculées par l'algorithme 2.5. L'analyse sur le facteur d'augmentation de l'efficacité n'est donc pas exacte, car on pourrait augmenter l'efficacité de l'estimateur Monte Carlo et ainsi diminuer l'augmentation par rapport à la nouvelle technique. Elle permet tout de même d'avoir un aperçu de l'efficacité de l'algorithme 12 avec une valeur de départ.

On peut voir les résultats de la valeur du VRF en fonction de T dans les graphiques de la figure 5.12. La courbe pleine est la valeur du VRF, en fonction de la valeur de T, calculé à partir de 100 répétitions de l'algorithme avec n=1021 méta-chaînes avec comme ensemble de points une règle de Korobov suivi d'une transformation du pâtissier.

La ligne discontinue est la valeur du VRF, en fonction de la valeur de T, calculé à partir de 100 répétitions de l'algorithme avec n=1024 méta-chaînes avec comme ensemble de points celui de Sobol. Les résultats sont obtenus à partir de la chaîne définie à la section 5.2 paramétrée par $\mu = 0$, $\sigma = 1$, et $\overline{W}$=5. Le nombre de méta-chaînes utilisées est, de gauche à droite, 2^{10}, 2^{12} et 2^{14}.

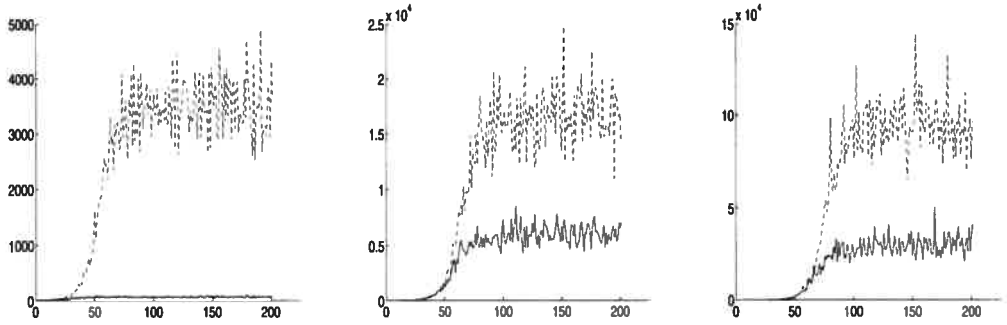

FIG. 5.12 – Graphique du VRF selon la valeur initiale de T

Les graphiques de la figure 5.2.2, sont ceux de l'efficacité en ordonnée par rapport à la valeur de T en abscisse.

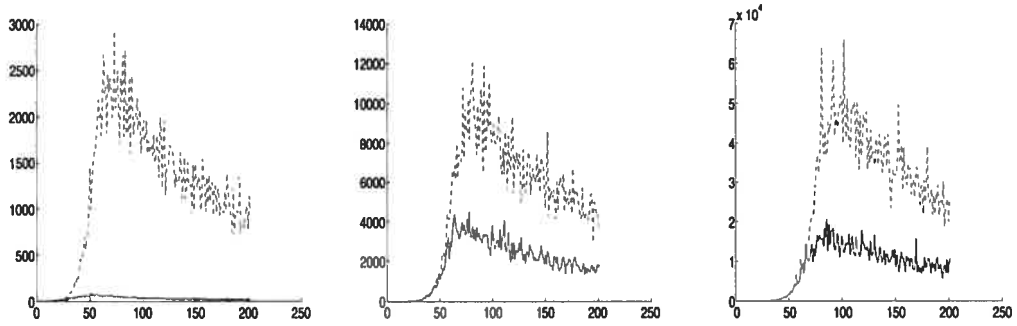

FIG. 5.13 – Graphique de l'efficacité selon la valeur initiale de T

En regardant les graphiques on remarque que les estimations de la valeur de départ T que l'on retrouve dans le tableau 5.6 étaient adéquates. On obtient, dans le tableau 5.7, les valeurs de l'efficacité de nos estimateurs en tenant compte du nombre de transitions calculées.

Quand on compare l'efficacité des estimateurs, avec comme critère le temps CPU, la charge suplémentaire de travail est de 6 à 30 fois plus élevée avec l'algorithme 12,

Paramètres	Nombre de méta-chaînes	2^{10}		2^{12}		2^{14}	
		T		T		T	
σ^2=5	Classical-Korobov-Baker		17		42		14
$\overline{W}$=5	Array-Korobov-Baker	10	817	10	5139	12	13046
	Array-Sobol	9	6385	11	36974	12	177675
σ^2=1	Classical-Korobov-Baker		11		13		18
$\overline{W}$=5	Array-Korobov-Baker	81	45	80	3782	98	11981
	Array-Sobol	86	2131	86	10182	92	54170
σ^2=0.5	Classical-Korobov-Baker		5		2		9
$\overline{W}$=5	Array-Korobov-Baker	303	63	319	6009	298	28106
	Array-Sobol	302	1740	326	10218	326	46954

TAB. 5.7 – Facteurs d'augmentation de l'efficacité avec l'algorithme 12

pour les trois exemples et les différentes valeurs du nombre de méta-chaînes. Si on regarde maintenant la valeur de l'efficacité de notre algorithme par rapport au temps de simulation on obtient le tableau 5.8.

Paramètres	Nombre de méta-chaînes	2^{10}		2^{12}		2^{14}	
		T		T		T	
σ^2=5	Classical-Korobov-Baker		17		42		14
$\overline{W}$=5	Array-Korobov-Baker	10	227	10	1428	12	2485
	Array-Sobol	9	2554	11	12911	12	36449
σ^2=1	Classical-Korobov-Baker		11		13		18
$\overline{W}$=5	Array-Korobov-Baker	81	11	80	727	98	1411
	Array-Sobol	86	588	86	2406	92	6846
σ^2=0.5	Classical-Korobov-Baker		5		2		9
$\overline{W}$=5	Array-Korobov-Baker	303	11	319	637	298	1753
	Array-Sobol	302	2969	326	1255	326	3195

TAB. 5.8 – Facteurs d'augmentation de l'efficacité avec l'algorithme 12, temps CPU

5.2.3 Discussion des résultats

On vient de voir jusqu'à maintenant plusieurs exemples de l'application des algorithmes 9, 11, 10 et 12. La simulation avec un espace d'états fini et petit semble moins intéressant, puisqu'il est possible de calculer la valeur de μ_∞ analytiquement dans ces cas. Mais il nous a servi de base de comparaison pour montrer que la combinaison de la technique CFTP et Array-RQMC permet d'obtenir de meilleurs résultats que ce que l'on retrouve dans la littérature [3, 24].

Dans le cas où l'espace d'états est très grand et que l'on doit faire beaucoup de

transitions (plus de 300 dans certains cas) l'efficacité des techniques RQMC décroit rapidement dans les exemples choisis. Par contre avec un bon choix de valeur pour le temps de départ dans le passé on obtient de très grands facteurs de réduction de la variance avec la combinaison des techniques CFTP et Array-RQMC. Bien que le temps de simulation, autant au niveau du nombre d'évaluations que du temps CPU, est plus élevé avec l'utilisation de Array-RQMC, le gain de performance reste quand même apréciable.

La performance des algorithmes est étroitement liée à leur implantation et au choix de la fonction de tri. Une meilleure implantation du tri permettrait donc de réduire le temps de simulation et ainsi avoir un algorithme plus performant. Il existe d'autres fonctions de tri que nous n'avons pas utilisées, il se peut qu'un autre choix de $h(\cdot)$ permette d'augmenter encore l'efficacité. Mais, même avec une implantation et un choix de $h(\cdot)$ plutôt simple, on obtient quand même de bons résultats. Cela laisse croire que l'efficacité pourrait être encore augmentée avec une meilleure implantation.

5.3 ROCFTP

Un autre coût important, surtout au niveau de l'utilisation mémoire, est la mémorisation et la réutilisation des valeurs générées par les différents processus. Une solution est proposée par Wilson [39] pour le cas où l'on utilise seulement une seule fois chacune des uniformes produites lors des simulations. Cette méthode est appelée *Read-Once Coupling From The Past* (ROCFTP). La combinaison de Array-RQMC avec cette technique permettrait d'éviter la mémorisation de milliers de nombres comme avec les algorithmes 10 et 12. Voici une variation de l'algorithme proposé par Wilson [29].

1. Choisir un bloc de taille T
2. Simuler des blocs de taille T vers l'avant dans le temps, où le nombre d'étapes dans chacun des blocs est aléatoire, jusqu'à ce que l'on trouve un bloc où il y a fusion à la dernière étape du bloc. Soit le bloc k, constitué des étapes entre le temps $(k-1)T$ jusqu'à kT. Conserver X_{kT}, la valeur à la fin de ce bloc comme étant le "chemin spécial".
3. Continer la simulation vers l'avant, en suivant les chemins à partir de tous les

états au début de chacun des blocs, mais en ayant une attention particulière au chemin spécial jusqu'à ce qu'une autre fusion soit détectée. Soit $l > k$ le second bloc ayant une fusion.

4. Retourner la valeur du chemin spécial au début du second bloc, au temp $(l-1)T$, où il y a fusion.

Cette idée est reprise par Murdoch et Takahara [29] pour des exemples de files d'attente et de modèles de réseaux. Ils ont démontré comment l'adapter afin d'obtenir des valeurs tirées de la loi de probabilité stationnaire d'une chaîne de Markov représentant une file d'attente ou un modèle de réseau. À première vue, cette méthode peut sembler être la solution au problème de mémorisation des uniformes produites car on ne les utilise seulement qu'une fois durant la simulation.

La combinaison entre Array-RQMC et ROCFTP donne des gains substanciels. Elle permet, dans les implantations que nous avons réalisées, d'obtenir des facteurs de réduction de la variance de 3. Des recherches plus poussées permettraient peut-être d'améliorer ce facteur.

Chapitre 6

Conclusion

La principale contribution de ce mémoire est d'avoir développé des manières de combiner l'algorithme Array-RQMC avec la méthode CFTP. Non seulement ces combinaisons permettent de produire des échantillons selon la loi de probabilité stationnaire d'une chaîne de Markov, mais elles permettent en plus d'obtenir des estimateurs ayant une plus grande efficacité que l'estimateur produit à l'aide de méthodes Monte Carlo ou quasi-Monte Carlo standards.

On peut voir dans ce présent travail quatres combinaisons distinctes :

1. Combinaison avec espace d'état fini et tri unique par cycle.
2. Combinaison avec espace d'état très grand et tri unique par cycle.
3. Combinaison avec espace d'état fini et temps de départ fixé.
4. Combinaison avec espace d'état très grand et temps de départ fixé.

Les combinaisons avec temps de départ fixé permettent d'obtenir les meilleurs gains de performance. Le gain de performance est de plusieurs milliers pour certains exemples. On a démontré ainsi qu'il est possible de combiner l'algorithme Array-RQMC et le CFTP et d'obtenir quand même des estimateurs ayant une plus grande efficacité que les méthodes RQMC standard même si le coût des estimateurs est plus grand.

Le choix de la fonction de tri $h(\cdot)$ et la manière de trouver la valeur du temps de départ ont été proposés de manière arbitraire. L'étude d'autres fonctions de tri pourrait permettre d'obtenir de meilleurs résultats. On a seulement démontré que l'utilisation d'un temps de départ fixé permettait d'obtenir de meilleurs résultats, mais sans donner

de conditions d'optimalité. Il reste encore du travail à faire pour l'élaboration de critères pour le choix de la valeur de départ.

On a certaines limitations avec l'algorithme 12. On ne peut utiliser de très grands ensembles de points, par exemple un ensemble avec 2^{18} points, car la mémorisation des uniformes demande trop d'espace mémoire. L'emploi d'autres implantations permettrait peut-être de réduire l'espace mémoire utilisé.

Pour l'instant, la combinaison avec ROCFTP permet d'obtenir des gains de performances, mais pas aussi importants que ceux donnés par l'algorithme 12. Il serait intéressant d'étudier la possibilité de développer un autre algorithme basé sur l'idée de ROCFTP afin d'améliorer ce gain dans les cas où on ne peut pas conserver toutes les valeurs générées durant la simulation.

Bibliographie

[1] M. C. Cario and B. L. Nelson. Modeling and generating random vectors with arbitrary marginal distributions and correlation matrix. working paper, 1997.

[2] R. V. Craiu and X.-L. Meng. Antithetic coupling for perfect sampling. In E. I. George, editor, *Bayesian Methods with Applications to Science, Policy, and Official Statistics (Selected Papers from ISBA 2000)*, pages 99–108, 2000.

[3] R. V. Craiu and X.-L. Meng. Multi-process parallel antithetic coupling for backward and forward Markov chain Monte Carlo. *Annals of Statistics*, 55(2) :661–697, 2003.

[4] X. K. Dimakos. A guide to exact simulation. *International Statistical Review*, 69(1) :27–48, 2001.

[5] E. C. Fieller and H. O. Hartley. Sampling with control variables. *Biometrika*, 1954.

[6] S. G. Foss and R. L. Tweedie. Perfect simulation and backward coupling. *Stochastic Models*, 14 :187–203, 1998.

[7] S. Ghosh and S. G. Henderson. Properties of the NORTA method in higher dimensions. In E. Yücesan, C.-H. Chen, J. L. Snowdon, and J. M. Charnes, editors, *Proceedings of the 2002 Winter Simulation Conference*, pages 263–269. IEEE Press, 2002.

[8] O. Häggström. *Finite Markov Chains and Algorithmic Applications.* Cambridge University Press, 2002.

[9] J. M. Hammersley and J. G. Mauldron. General principles of antithetic variates. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52 :476–481, 1956.

[10] J. M. Hammersley and K. V. Morton. A new Monte Carlo technique : antithetic variates. *Mathematical Proceedings of the Cambridge Philosophical Society*, 52 :449–475, 1956.

[11] F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research.* McGraw-Hill, fifth edition, 1990.

[12] ILOG, Inc. ILOG CPLEX : High-performance software for mathematical programming and optimization, 2006. See `http://www.ilog.com/products/cplex/`.

[13] H. Kahn and A. W. Marshall. Methods of reducing sample size in Monte Carlo computations. *Journal of the Operations Research Society of America*, 1953.

[14] W. S. Kendall. Perfect simulation for the area-interaction point process. 1998.

[15] L. Laurencelle. *Hasard, Nombres Aléatoires et Méthode Monte Carlo.* Presses de l'Université du Québec, 2001.

[16] S. S. Lavenberg and P. D. Welch. A perspective on the use of control variables to increase the efficiency of Monte Carlo simulations. *Management Science*, 27 :322–335, 1981.

[17] C. Lécot. Error bound for quasi-Monte Carlo integration with nets. *Mathematics of Computation*, 65(213) :179–187, 1996.

[18] P. L'Ecuyer. *SSJ : A Java Library for Stochastic Simulation*, 2004. Software user's guide, Available at `http://www.iro.umontreal.ca/~lecuyer`.

[19] P. L'Ecuyer. *Stochastic Simulation.* 2006. Notes for a graduate simulation course.

[20] P. L'Ecuyer, C. Lécot, and B. Tuffin. A randomized quasi-Monte Carlo simulation method for Markov chains. Submitted to *Operations Research*, under revision, 2005.

[21] P. L'Ecuyer, C. Lécot, and B. Tuffin. Randomized quasi-Monte Carlo simulation of Markov chains with an ordered state space. In H. Niederreiter and D. Talay, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2004*, pages 331–342, 2006.

[22] P. L'Ecuyer and C. Lemieux. Variance reduction via lattice rules. *Management Science*, 46(9) :1214–1235, 2000.

[23] P. L'Ecuyer and C. Lemieux. Recent advances in randomized quasi-Monte Carlo methods. In M. Dror, P. L'Ecuyer, and F. Szidarovszky, editors, *Modeling Uncer-*

tainty : An Examination of Stochastic Theory, Methods, and Applications, pages 419–474. Kluwer Academic, Boston, 2002.

[24] C. Lemieux and P. Sidorsky. Exact sampling with highly-uniform point sets. *Mathematical and Computer Modelling.* To appear. See `http://www.math.ucalgary.ca/~lemieux`.

[25] T. Lindvall. *Lectures on the Coupling Method.* Wiley Series in Probability and Mathematical Statistics. John Wiley, New York, 1992.

[26] W.-L. Loh. On Latin hypercube sampling. *The Annals of Statistics*, 24 :2058–2080, 1996.

[27] J. Matoušek. *Geometric Discrepancy : An Illustrated Guide.* Springer-Verlag, Berlin, 1999.

[28] M. D. McKay, R. J. Beckman, and W. J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21 :239–245, 1979.

[29] D. J. Murdoch and G. Takahara. Perfect sampling for queues and network models. *ACM Transations on Modeling and Computer Simulation*, 2006.

[30] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*, volume 63 of *SIAM CBMS-NSF Regional Conference Series in Applied Mathematics.* SIAM, Philadelphia, 1992.

[31] J. R. Norris. *Markov Chains.* Cambridge Series in Statistical and Probabilistic mathematics. Cambridge University Press, New York, 1997.

[32] A. B. Owen. A central limit theorem for Latin hypercube sampling. *Journal of the Royal Statistical Society B*, 54(2) :541–551, 1992.

[33] A. B. Owen. Scrambled net variance for integrals of smooth functions. *Annals of Statistics*, 25(4) :1541–1562, 1997.

[34] A. B. Owen. Latin supercube sampling for very high-dimensional simulations. *ACM Transactions on Modeling and Computer Simulation*, 8(1) :71–102, 1998.

[35] J. G. Propp and D. B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, 9(1&2) :223–252, 1996.

[36] I. H. Sloan and S. Joe. *Lattice methods for multiple integration.* Oxford Science Publications. The Clarendon Press Oxford University Press, New York, 1994.

[37] M. Stein. Large sample properties of simulations using Latin hypercube sampling. *Technometrics*, 29(2) :143–151, 1987.

[38] H. Thorisson. *Coupling, Stationarity, and Regeneration.* Probability and its applications. Springer-Verlag, New York, 2000.

[39] D. B. Wilson. How to couple from the past using a read-once source of randomness. *Random Structures and Algorithms*, 16, 2000.