Université de Montréal

Construction d'ensembles de points basée sur des récurrences linéaires dans un corps fini de caractéristique 2 pour la simulation Monte Carlo et l'intégration quasi-Monte Carlo

par

François Panneton

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures

en vue de l'obtention du grade de

Philosophiae Doctor (Ph.D.)

en informatique

Août 2004

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée:

Construction d'ensembles de points basée sur des récurrences linéaires dans un corps fini de caractéristique 2 pour la simulation Monte Carlo et l'intégration quasi-Monte Carlo

présentée par:

François Panneton

a été évaluée par un jury composé des personnes suivantes:

Victor Ostromoukhov

(président-rapporteur)

Pierre L'Écuyer

(directeur de recherche)

Felisa J. Vázquez-Abad

(membre du jury)

Henri Faure

(examinateur externe)

Thèse acceptée le:

25 octobre 2004

Sommaire

Dans cette thèse, nous nous intéressons au développement de nouveaux générateurs de nombres pseudo-aléatoires qui utilisent une récurrence linéaire dans un corps fini de caractéristique 2. Nous utilisons aussi ces récurrences pour définir de nouveaux ensembles de points qui peuvent être utilisés par les méthodes quasi-Monte Carlo.

Tout d'abord, nous définissons les critères avec lesquels nous choisissons les paramètres de nos générateurs. Parmi ces critères, on retrouve l'équidistribution et la distance minimale entre deux points d'un ensemble P_n de n points dans $[0,1)^t$. Pour ce dernier, nous développons un nouvel algorithme qui permet de trouver la distance minimale entre deux points de P_n en temps moyen $O(n)$.

Ensuite, nous définissons de nouveaux générateurs qui utilisent une récurrence linéaire dans $\mathbb{F}_{2^w}$, le corps fini avec 2^w éléments. Ces récurrences sont aussi utilisées pour définir de nouveaux ensembles de points qui peuvent être utilisés dans des applications quasi-Monte Carlo. La grande majorité des projections en deux dimensions des nouveaux ensembles de points construits avec ces récurrences ont la propriété d'être parfaites par rapport à l'équidistribution.

Nous développons également une autre famille de générateurs, les générateurs WELL, qui possèdent d'excellentes propriétés. En effet, l'équidistribution est presque parfaite pour plusieurs générateurs et le nombre de coefficients non nuls du polynôme caractéristique est près de $k/2$ où k est son degré. La vitesse d'exécution de ces

nouveaux générateurs est comparable aux plus rapides des générateurs disponibles.

Nous concluons cette thèse en analysant une famille de générateurs proposée par G. Marsaglia [60]. Nous effectuons une analyse théorique de ces générateurs et cherchons les meilleurs du point de vue de l'équidistribution. Malgré que ces générateurs ont, en général, une bonne équidistribution, la récurrence est trop simple pour produire de bons générateurs comme le démontrent leurs nombreuses faiblesses aux différents tests statistiques que nous leurs avons fait passer.

Mots-clés : Générateurs de nombres aléatoires, récurrence linéaire, Monte Carlo, quasi-Monte Carlo, simulation, ensembles de points uniformes, équidistribution, réseau polynômial, uniformité.

Summary

In this thesis, we are interested in the development of new random number generators (RNG) that use a linear recurrence in a finite field of characteristic 2. We also use these recurrences to define new point sets for quasi-Monte Carlo applications.

We first define the criteria used to choose the parameters of our generators. Among them, we find the equidistribution and the minimal distance between two points from a point set. For this last criterion, we developed a new algorithm that computes the minimal distance in average time $O(n)$ where n is the cardinality of the point set.

We then define new RNGs that use a linear recurrence in $\mathbb{F}_{2^w}$, the finite field with 2^w elements. With these generators, we also define new point sets for quasi-Monte Carlo applications. The vast majority of the two-dimensional projections of these new point sets have a perfect equidistribution.

Also, we define a new family of generators known as WELL RNGs. The equidistribution of these generators is almost perfect and the number of non zero coefficients in the characteristic polynomial is close to $k/2$ where k is its degree. The speed of these generators is comparable to other fast generators.

We conclude this thesis by analyzing a family of generators proposed by Marsaglia [60], known as "xorshift generators". We analyze their theoretical properties and look for the best generators. Even though they have, in general, good equidistribution

properties, the recurrence they use is too simple to produce good generators. We show this by testing many generators empirically.

Keywords : Random number generators, linear recurrence, Monte Carlo, quasi-Monte Carlo, simulation, uniform point sets, equidistribution, polynomial lattice, uniformity.

Table des matières

Liste des figures

Liste des tableaux

Liste des sigles et abréviations

$\mathbf{a}$	Vecteur $\mathbf{a}$.
$a^{(i)}$	L'élément numéro i de $\mathbf{a}$. Les éléments sont numérotés de gauche à droite en partant de 0.
k	Degré du polynôme caractéristique du générateur.
S	Espace des états du générateur.
U	Espace des sorties du générateur.
$\mathbf{x}_n$	État du générateur à la n-ième itération.
$\mathbf{z}_n$	Résultat de la multiplication de $\mathbf{x}_n$ par B à la n-ième itération.
$\mathbf{y}_n$	Vecteur de sortie du générateur à la n-ième itération.
u_n	Sortie du générateur à la n-ième itération comprise dans l'intervalle $[0, 1)$.
X	Matrice de transition du générateur.
Y	Matrice multipliant $\mathbf{z}_n$ pour obtenir $\mathbf{y}_n$.
B	Matrice représentant une transformation linéaire appliquée au vecteur d'état. Multiplie $\mathbf{x}_n$ pour obtenir $\mathbf{z}_n$.
L	Résolution de la sortie du générateur.

$\mathbf{a}^\mathsf{T}$	Transposée du vecteur $\mathbf{a}$.
B^T	Transposée de la matrice B.
B^{-1}	Inverse de la matrice B.
$\text{pgcd}(a, b)$	Plus grand commun diviseur de a et b.
$\text{ppcm}(a, b)$	Plus petit commun multiple de a et b.
$\mathbb{F}_2$	Corps fini à deux éléments.
$\mathbb{F}_{2^k}$	Corps fini à 2^k éléments.
$\mathbb{F}_2[z]$	Ensemble des polynômes ayant des coefficients dans $\mathbb{F}_2$.
$\mathbb{Z}_k$	L'ensemble $\{0, 1, \ldots, k-1\}$.
$\deg(p)$	Degré du polynôme p.
I_k	Matrice identité $k \times k$ dans $\mathbb{F}_2$.
$I_{L\times k}$	Matrice $L \times k$ ne contenant que les L premières lignes de I_k.
$\mathbf{a} \oplus \mathbf{b}$	Ou-exclusif bit par bit entre $\mathbf{a}$ et $\mathbf{b}$.
$\mathbf{a} \ll j$	Décalage de j bits vers la gauche du vecteur $\mathbf{a}$.
$\mathbf{a} \gg j$	Décalage de j bits vers la droite du vecteur $\mathbf{a}$.
$\lfloor x \rfloor$	Plus grand entier inférieur ou égal à x.
$\lceil x \rceil$	Plus petit entier supérieur ou égal à x.
$\text{trunc}_w(\mathbf{x})$	Opération qui retourne un vecteur de w bits contenant les w premiers bits de $\mathbf{x}$.

À mes amours,

Mélanie, Bébé et Charles-Émile.

Remerciements

Je voudrais remercier tous ceux qui m'ont aidé tout au long de mes études de doctorat. Tout d'abord, je voudrais remercier le CRSNG et le FCAR qui m'ont supporté financièrement tout au long de mes études de doctorat et de maîtrise.

Je tiens aussi à remercier mon directeur de recherche, Pierre L'Écuyer, pour son support financier et sa disponibilité tout au long de mes études de doctorat et de maîtrise. Il m'a fait connaître les générateurs de nombres aléatoires et a su me transmettre sa passion pour ce champ d'étude. Il m'a aussi fait découvrir d'autres cultures en me permettant de participer à des conférences en Asie et en Europe. Les discussions que nous avons eues ensemble m'ont permis de découvrir plusieurs résultats importants pour la rédaction de cette thèse et d'approfondir les différentes avenues de recherche qui se présentaient à moi.

Je ne peux passer sous le silence le soutient de ma copine, Mélanie, qui m'a toujours appuyé dans mes études, malgré les moments de doute et de découragement. Avec l'aide de mon fils Charles-Émile, elle m'a permis de penser à d'autres choses que « des uns et des zéros ».

Je voudrais aussi souligner la contribution de M. Alexander Keller qui, lors de son passage à notre laboratoire de simulation, m'a fait part de l'intérêt d'avoir des ensembles de points dont la distance minimale est grande pour les applications en infographie. Grâce à ces discussions, j'ai pu aboutir à un nouvel algorithme pour le

calcul de la distance minimale.

Je remercie M. Makoto Matsumoto pour les discussions que nous avons eues ensemble pendant l'atelier sur les nombres aléatoires qui a eu lieu à Montréal à l'été 2002. Ces discussions ont mené au développement des WELLRNG.

Également, M. Takuji Nishimura, lors de son passage a Montréal, m'a fait part de son implantation de l'algorithme, spécialement adapté au Mersenne twister, qui permet de trouver le plus court vecteur dans un réseau polynômial. Cette discussion m'a encouragé à revoir ma propre implantation. Sans cette révision, le calcul de l'équidistribution des WELLRNG de période $2^{19937} - 1$ et plus aurait été très long et peu efficace. Pour tout cela, je le remercie.

Je remercie tous les gens (dont quelques noms sont mentionnés dans la thèse) que j'ai contactés par internet et qui ont bien voulu répondre à mes questions. Également, je remercie M. Étienne Marcotte qui a révisé ma thèse et qui a écrit le programme qui a permis de tracer les graphiques de la section 7.7.

Finalement, je voudrais remercier tous les membres de ma famille (au sens large) qui m'ont encouragé à « mettre des points dans un carré ». Leur support moral a été primordial et m'a permis de mener à bien mes études doctorales.

Chapitre 1

Introduction

1.1 Comment imiter le hasard

Lors de simulation de systèmes stochastiques sur ordinateur, il est essentiel de disposer d'une source de hasard. Celle-ci consiste en une suite de valeurs $\{u_i\}_{i\geq 0}$ qui imite une suite de variables aléatoires uniformes indépendantes dans l'intervalle [0,1). Souvent, on utilise des algorithmes déterministes afin de produire de telles suites. Les principales propriétés désirées qui permettent de choisir les méthodes à utiliser sont expliquées dans [34]. Voici un résumé de celles-ci :

1. Bonnes propriétés statistiques : on désire obtenir une séquence de nombres $\{u_i\}_{i\geq 0}$ qui passe avec succès la plupart des tests statistiques raisonnables.
2. Longue période : supposons que l'on ait besoin de N valeurs aléatoires pour une simulation, alors il faut que la période (le nombre de valeurs produites avant que la séquence ne se répète) des valeurs produites soit beaucoup plus grande que N.
3. Efficacité : le temps nécessaire pour produire les valeurs doit être petit par rapport au temps de la simulation. Ce facteur devient important dans des si-

mulations qui prennent plusieurs jours d'exécution.

4. Répétabilité : l'utilisateur doit être capable de reproduire la même séquence de nombres facilement. Ceci est important pour la vérification des programmes et certaines techniques de réduction de variance.
5. Facilité d'implantation et séparabilité : l'implantation du générateur doit pouvoir être exécutée sur tous les types d'ordinateurs standards. Également, il doit être possible de « sauter » d'une valeur u_n à une autre u_{n+s} facilement, même quand s est grand. De cette manière, on peut utiliser plusieurs sous-séquences de la séquence $\{u_n\}_{n\geq 0}$ et considérer chaque sous-séquence comme un générateur indépendant (ceci exige que la période de la séquence $\{u_n\}_{n\geq 0}$ soit assez grande pour le permettre).

Au critère 1, un test statistique raisonnable est un test qui peut être effectué dans un temps raisonnable. Par exemple, s'il faut 200 ans avant de trouver un défaut sur un générateur à l'aide d'un test statistique, alors on peut dire que le test n'est pas raisonnable. Il existe plusieurs batteries de tests qui peuvent être utilisées dont DIEHARD et TestU01 [31, 19, 29, 61].

Il existe deux types de méthodes permettant de reproduire le hasard et qui tentent de répondre à ces critères. Il y a les méthodes physiques et les méthodes mathématiques.

Les méthodes physiques produisent des valeurs « vraiment aléatoires ». Par exemple, on pourrait utiliser une pièce de monnaie et enregistrer les résultats des tirages successifs de celle-ci. On obtiendrait alors une suite binaire aléatoire. D'autres méthodes peuvent être utilisées et les résultats observés seraient aléatoires dans le sens que ceux-ci suivent une distribution de probabilité qui est propre à l'expérience choisie.

Ces méthodes ont plusieurs désavantages qui les rendent inintéressantes pour la simulation. Le critère 3, l'efficacité, est difficilement respecté puisque le nombre de valeurs disponibles est limité par le temps nécessaire pour les produire et/ou l'es-

pace de stockage de l'information. Si le nombre de valeurs produites est trop faible, alors celles-ci auront vite été utilisées lors d'une simulation. Également, ces méthodes vérifient rarement le critère 1 étant donné la difficulté d'obtenir des valeurs successives vraiment indépendantes les unes des autres et aussi par le fait qu'il est difficile de s'assurer que les valeurs produites suivent une distribution uniforme dans l'intervalle [0,1). Ces méthodes ne sont pas souvent utilisées de nos jours pour la simulation. Les références [3] et [30] traitent plus en détails des méthodes physiques.

Le deuxième type de méthodes, les méthodes mathématiques, permettent de produire une suite de nombres qui ressemble à celle que produirait un système parfaitement aléatoire. Les nombres produits sont alors appelés *nombres pseudo-aléatoires* car ils ne sont pas générés par une méthode aléatoire, mais plutôt par une méthode déterministe.

Les méthodes mathématiques ont l'avantage de ne pas nécessiter d'espace de stockage permanent et de produire des valeurs pseudo-aléatoires rapidement, ce qui répond au critère 3. Aussi, le critère 2 est respecté puisque les méthodes actuelles ont de longues périodes, c'est-à-dire, que les simulations n'utilisent jamais toutes les valeurs disponibles si le générateur est bon. C'est à cause de leur rapidité et de leur facilité d'utilisation que l'on utilise des méthodes mathématiques pour la majorité des simulations. Le modèle mathématique qui permet de produire la séquence de nombres aléatoires est appelé *générateur de nombres pseudo-aléatoires.* Afin d'alléger le texte, on parlera plutôt d'un *générateur de nombres aléatoires* ou *générateur.* Une propriété des méthodes mathématiques est que les séquences qu'elles produisent ont toujours une période finie.

Idéalement, en plus du critère 1, on voudrait que le générateur soit imprévisible, c'est-à-dire qu'on voudrait qu'il soit impossible de faire la différence entre une séquence produite par un générateur donné et une suite de variables aléatoires indépendantes identiquement distribuées uniformes dans l'intervalle [0,1) avec une probabilité

significativement supérieure à 1/2 [32]. Évidemment, si on dispose d'un temps infini, ceci n'est pas possible puisqu'en examinant la séquence des nombres générés assez longtemps, il est possible de déterminer de façon exacte la prochaine valeur du générateur, puisque la séquence est périodique. Pour certains générateurs, il est même possible de déterminer tous les paramètres du modèle, même si on ne connaît qu'une petite fraction de la période [27, 89]. Quelques générateurs sont pratiquement imprévisibles [47, 2], mais ceux-ci sont trop lents pour les besoins pratiques de la simulation [32].

Une limitation des générateurs de nombres aléatoires est que les valeurs produites ne sont pas aléatoires, ils ne font qu'imiter des variables aléatoires. Ceci engendre des structures entre les nombres générés. C'est pour cette raison qu'il faut être très prudent lorsqu'on conçoit un générateur avec une méthode mathématique. Dans le passé, des gens ont conçu des générateurs sans en étudier à fond leurs propriétés. Un générateur célèbre pour ses lacunes est le générateur RANDU qui a été implanté sur les ordinateurs d'IBM dans les années 1960. Voir [24] et [28] pour plus de détails sur ce générateur et ses défaillances.

Pour cette thèse, nous développons des générateurs de nombres aléatoires qui ne sont pas imprévisibles, c'est-à-dire qu'on peut prédire exactement la séquence des futurs nombres à générer en n'observant que quelques valeurs consécutives. Ainsi, aucun des générateurs contenus dans cette thèse ne peut être utilisé tel quel dans une application cryptographique. Par contre, ils sont très utiles pour les besoins de la simulation numérique à cause de leur rapidité et de leurs bonnes propriétés statistiques.

1.2 Définition du générateur de nombres aléatoires

Il existe plusieurs types d'algorithmes déterministes pour la génération de nombres pseudo-aléatoires. Dans [32], on trouve un excellent compte-rendu des principaux algorithmes de génération de nombres aléatoires. Également, on y explique une structure commune à tous les générateurs qui se résume à la définition suivante.

Définition 1.1. (L'Ecuyer [32])
Soit un espace fini d'états S, une loi de probabilité P qui permet de choisir un élément de S, une fonction $f : S \to S$, appelée *fonction de transition*, et une fonction $g : S \to U \subset \mathbb{R}$, appelée *fonction de sortie*. On appelle *générateur de nombres pseudo-aléatoires* un algorithme qui choisit un élément $s_0 \in S$, appelé *germe*, grâce à la loi de probabilité P, et qui produit une séquence $u_n = g(s_n) \in U, n \geq 0$, où $s_n = f(s_{n-1})$, qui tente d'imiter une séquence de variables aléatoires indépendantes et uniformes sur U. On note ce générateur $G = (S, P, f, U, g)$.

Le générateur produit, à chacune des itérations, un nouvel état $s_n \in S$ avec f et une sortie $u_n \in U$, obtenue avec g. Pour la très grande majorité des générateurs de nombres aléatoires, $U = [0, 1)$. Ceci n'est pas une restriction, puisqu'il est possible d'obtenir n'importe quelle variable aléatoire avec une ou plusieurs variables aléatoires uniformes dans $[0, 1)$ [28]. Remarquons que, puisque le nombre d'états est fini, la séquence des nombres en sortie (les u_n) est périodique. La *période* d'une séquence $\{u_n\}_{n\geq 0}$ est la plus petite valeur de s telle qu'il existe un n_0 pour lequel $u_n = u_{n+s}$ pour tout $n \geq n_0$. La période maximale de la séquence produite par un générateur est $|S|$.

Malheureusement, le simple fait de choisir une structure compatible avec cette définition ne garantit pas que la séquence des valeurs produites soit de bonne qualité, c'est-à-dire qu'elle imite bien une séquence de nombres vraiment aléatoires. Pour que la séquence $\{u_n\}_{n\geq 0}$ soit de qualité, il faut choisir S, f et g judicieusement. Ce

choix doit s'appuyer sur des arguments théoriques et empiriques solides. Il existe des générateurs contenus dans des programmes et des bibliothèques informatiques couramment utilisés qui sont de piètre qualité [38]. La difficulté de trouver de bons S, f et g justifie de continuer la recherche de bons générateurs et de travailler à mieux comprendre leurs propriétés théoriques.

1.3 Notation

Dans cette section, nous introduisons la notation qui sera utilisée tout au long de cette thèse. Tout d'abord, mentionnons que la transposée est notée par $^\mathsf{T}$ (par exemple, la transposée de A est A^T). Pour représenter un vecteur en k dimensions, nous utilisons la notation $\mathbf{x} = (x^{(0)}, \ldots, x^{(k-1)})^\mathsf{T}$. Ainsi, le symbole pour le vecteur est en caractère gras, mais non ses éléments. L'indexation des éléments d'un vecteur commence à zéro. Pour les matrices, on utilise une lettre en majuscule, par exemple A.

Dans cette thèse, nous utiliserons beaucoup la notion de corps fini. Le lecteur peu familier avec cette notion est invité à lire l'annexe A qui passe en revue les propriétés de base de ceux-ci. On note le corps fini à b éléments, où b est une puissance d'un nombre premier, par $\mathbb{F}_b$ et l'espace des vecteurs en k dimension dont les éléments sont dans $\mathbb{F}_b$ par $\mathbb{F}_b^k$. L'ensemble des polynômes à coefficients dans $\mathbb{F}_b$ est noté $\mathbb{F}_b[z]$.

Pour les vecteurs dans $\mathbb{F}_2^k$, nous définissons les opérations suivantes :

opération	explication
$\mathbf{x} \ll v$	décalage de v bits vers la gauche
$\mathbf{x} \gg v$	décalage de v bits vers la droite
$\mathbf{y} \oplus \mathbf{x}$	addition bit à bit de $\mathbf{y}$ et $\mathbf{x}$
$\mathbf{y}$ & $\mathbf{x}$	et-exclusif bit à bit de $\mathbf{y}$ et $\mathbf{x}$

Une séquence infinie de valeurs $a_1, a_2, \ldots$ est représentée par $\{a_i\}_{i\geq 1}$.

1.4 Cadre général des générateurs étudiés dans cette thèse

Dans cette thèse, on se concentre sur les générateurs de nombres pseudo-aléatoires qui utilisent une récurrence linéaire modulo 2. L'espace des états est $\mathbb{F}_2^k$ et un état peut être représenté par un vecteur de k bits. La représentation par vecteurs de bits est utilisée afin de passer d'un état à l'autre, puisque celle-ci est facilement manipulée par les ordinateurs actuels. Les générateurs connus de ce type sont les générateurs de Tausworthe ou « linear feedback shift register » (LFSR) [100, 33, 37], les « generalized feedback shift register » (GFSR) [55, 22], les « twisted GFSR » (TGFSR) [66, 67], les générateurs à congruence linéaire polynômiaux (GCLP) [45] et les « Mersenne Twister » (MT) [69, 79]. Tous ces générateurs utilisent, sous des formes différentes, le même type de récurrence linéaire et peuvent être représentés sous une forme matricielle que nous allons maintenant décrire.

En partant de la définition des générateurs de nombres aléatoires introduite à la section 1.2, on définit l'espace d'états S, la fonction de transition f et la fonction de sortie g utilisés pour les générateurs à récurrence linéaire modulo 2.

L'espace d'états S est $\mathbb{F}_2^k$. Un élément de $\mathbb{F}_2^k$ est un vecteur de k bits. Le n-ième état est noté $\mathbf{x}_n = (x_n^{(0)}, , \ldots, x_n^{(k-1)})^\mathsf{T}$. La fonction de transition $f : S \to S$, est représentée par la matrice X, dans $\mathbb{F}_2$. La transition se fait par

$$\mathbf{x}_n = X\mathbf{x}_{n-1} \tag{1.1}$$

où X, qu'on appelle *matrice de transition*, est une matrice de dimension $k \times k$ dont les éléments sont dans $\mathbb{F}_2$. Un polynôme important relié à la matrice X est le *polynôme*

caractéristique de X qui est défini par $P_X(z) = \det(X - Iz)$. Ce polynôme sera utilisé fréquemment tout au long de cette thèse.

Pour obtenir la sortie, on a besoin d'un vecteur de L bits $\mathbf{y}_n$ que l'on définit par

$$\mathbf{z}_n = B\mathbf{x}_n \tag{1.2}$$

$$\mathbf{y}_n = Y\mathbf{z}_n \tag{1.3}$$

où $\mathbf{z}_n = (z_n^{(0)}, \ldots, z_n^{(k-1)})^\mathsf{T}$, $\mathbf{y}_n = (y_n^{(0)}, \ldots, y_n^{(L-1)})^\mathsf{T}$, Y est une matrice $L \times k$, B est une matrice $k \times k$.

On remarque que $\mathbf{y}_n = YB\mathbf{x}_n$. Cette décomposition est pratique lorsqu'on désire appliquer du tempering à la sortie du générateur. Habituellement, la matrice B est une matrice de plein rang k. La matrice Y est la matrice qui détermine le nombre de bits de résolution que le générateur aura à sa sortie. Si $L \leq k$, alors la matrice Y, multipliée par un vecteur, tronquera celui-ci pour ne garder que les L premiers bits. Par contre si $L > k$, alors la matrice Y aura la forme

$$Y = (I_k \; ; \; C)^\mathsf{T}$$

où I_k est la matrice identité $k \times k$ et C est une matrice $(L-k) \times k$ quelconque qui permet d'avoir plus que k bits à la sortie. La sortie $u_n \in [0, 1)$ associée à $\mathbf{x}_n$ est

$$u_n = \sum_{i=1}^{L} y_n^{(i-1)} 2^{-i}. \tag{1.4}$$

La valeur de L est dite la *résolution de sortie du générateur*. Ce terme est approprié puisque plus la valeur de L est grande plus les valeurs générées pourront avoir de la précision, c'est-à-dire qu'elles auront plusieurs chiffres après la virgule. La résolution représente le niveau de discrétisation de l'ensemble $[0, 1)$. Par exemple, si $L = 3$, alors l'ensemble des valeurs qui peuvent être produites par le générateur est $\{0, 1/8, 2/8, 3/8, 4/8, 5/8, 6/8, 7/8\}$. Par contre si L est grand, on aura une meilleure approximation de $[0, 1)$. On pourrait faire une analogie avec la résolution d'une photo numérisée : plus celle-ci est grande, plus on observe de détails au grossissement.

Les différents générateurs qui seront traités dans cette thèse ont tous la même structure ; seules leurs matrices X, Y et B respectives les différencient. On doit remarquer que, dans la définition donnée à la section 1.2, la fonction de sortie $g : S \to U$ est la combinaison des équations (1.2), (1.3) et (1.4).

En examinant les équations (1.1), (1.2), (1.3) et (1.4), on observe que l'on peut obtenir différentes suites de nombres $\{u_n\}_{n\geq 0}$ avec la même matrice X en utilisant différentes matrices B. Cette observation se révèle importante lorsqu'on désire améliorer ou changer les propriétés des générateurs. Souvent, les conditions qu'on impose à la matrice X sont plus restrictives que celles imposées à la matrice B, ce qui rend la matrice B plus facile à changer que la matrice X. Pour une matrice X donnée, on sélectionnera les matrices B selon un critère qui mesure la qualité du générateur obtenu. Un des critères qui sert souvent de mesure de la qualité des générateurs de nombres aléatoires à récurrences linéaires modulo 2 est l'équidistribution, que nous définissons dans la prochaine section.

1.5 Bref aperçu du critère d'équidistribution

Dans la section 1.1, on énumère différents critères permettant de juger de la qualité des générateurs de nombres aléatoires. Le premier critère est que le générateur doit avoir de bonnes propriétés statistiques et d'uniformité. L'équidistribution est une mesure de l'uniformité des points générés dans l'hypercube $[0,1)^t$.

Considérons l'ensemble $\Omega_{k,t}$ de tous les points à t dimensions produits par les valeurs successives d'un générateur, à partir de tous les 2^k états initiaux possibles $\mathbf{x}_0$, défini par

$$\Omega_{k,t} = \{\mathbf{u}_{0,t} = (u_0, \ldots, u_{t-1}) : \mathbf{x}_0 \in \mathbb{F}_2^k\} \subset [0,1)^t.$$

Supposons que l'on partitionne l'hypercube $[0,1)^t$ en $2^{t\ell}$ petits hypercubes de même grandeur. La plus grande valeur de ℓ pour laquelle chacun des petits hypercubes contient le même nombre de points de $\Omega_{k,t}$ est appelée la *résolution* en dimension t et est notée ℓ_t. Similairement, pour une résolution ℓ, la plus grande valeur de t telle que chacun des petits hypercubes contient le même nombre de points de $\Omega_{k,t}$ est notée t_ℓ. On dit que $\Omega_{k,t}$ (et par le fait même, le générateur) est (t,ℓ)-*équidistribué* si tous les petits hypercubes contiennent le même nombre de points, soit $2^{k-t\ell}$ points. Ceci est possible seulement si $k \geq t\ell$, parce que la cardinalité de $\Omega_{k,t}$ est au plus 2^k. La séquence $\{u_i\}_{i\geq 0}$ (ainsi que le générateur qui la produit) est dite *équidistribuée au maximum* (ou ME, de l'anglais "maximally equidistributed"), si ℓ_t atteint sa borne supérieure pour toutes les valeurs de t possibles, c'est-à-dire $\ell_t = \ell_t^*$ où $\ell_t^* \stackrel{\text{def}}{=} \min(L, \lfloor k/t \rfloor)$ pour $t = 1, \ldots, k$. On définit aussi *l'écart en dimension* t par $\Lambda_t \stackrel{\text{def}}{=} \ell_t^* - \ell_t$. On dit que le générateur a une *résolution maximale en dimension* t si $\Lambda_t = 0$. Selon ces définitions, on peut dire que le générateur est équidistribué au maximum (ME), si $\Lambda_t = 0$ pour $t = 1, \ldots, k$.

Dans la littérature, lorsque vient le temps de vérifier la qualité des générateurs de nombres aléatoires, il existe deux types de méthodes : les méthodes empiriques et les méthodes théoriques. Dans [35], on conclut qu'il est préférable de considérer les propriétés théoriques en premier lieu et ensuite, si les résultats sont satisfaisants, de vérifier les propriétés empiriques. Aussi, on affirme que si un générateur satisfait à un critère théorique exigeant, les chances d'échouer des critères empiriques diminuent considérablement. Par contre, étant donné le caractère déterministe des générateurs de nombres aléatoires, il est toujours possible de faire échouer un générateur à un test statistique.

L'équidistribution est souvent utilisée pour sélectionner les générateurs à récurrence linéaire modulo 2 parce qu'elle est facile à calculer. Les algorithmes de calculs de l'équidistribution tirent avantage de la linéarité des générateurs. Une description de

ces algorithmes est donnée dans la section 4.2. Un autre critère, qui s'apparente à l'équidistribution, est la q-valeur du (q, k, t)-réseau digital que l'ensemble $\Omega_{k,t}$ décrit. Cette notion, et d'autres mesures d'uniformité, seront expliquées plus en détail au chapitre 4.

1.6 Simulation Monte Carlo et intégration quasi-Monte Carlo

Beaucoup de simulations numériques sur ordinateur utilisant des variables aléatoires uniformes peuvent être considérées comme des intégrations. Soit μ, la valeur qu'on désire estimer par la simulation et $f(\mathbf{u})$, le résultat d'une simulation étant donné le vecteur de t variables aléatoires uniformes $\mathbf{u}$ que l'on donne en entrée à la simulation. On peut envisager la simulation comme une méthode d'approximation de l'intégrale d'une fonction réelle f sur l'hypercube unitaire $[0,1)^t$ donnée par

$$\mu = \int_{[0,1)^t} f(\mathbf{u})\, d\mathbf{u}.$$

De façon générale, la simulation Monte Carlo fonctionne de la manière suivante. On prend un ensemble de points $P_n = \{\mathbf{u}_1, \ldots, \mathbf{u}_n\}$, un ensemble de n vecteurs $\mathbf{u}_i$ indépendants et uniformément distribués sur $[0,1)^t$ et on calcule la moyenne de f sur tous les points de P_n,

$$R_n = \frac{1}{n} \sum_{i=1}^{n} f(\mathbf{u}_i),$$

comme une approximation de μ. Il est possible de calculer un intervalle de confiance sur μ avec cet estimateur à l'aide du théorème de la limite centrale.

Il existe un autre type d'algorithme qui tente d'évaluer la même intégrale, mais avec un ensemble de points P_n plus uniforme que l'ensemble de points uniformément

distribués de la simulation Monte Carlo. Cette technique est l'intégration quasi-Monte Carlo.

Une manière d'obtenir cet ensemble très uniforme est d'utiliser un générateur de nombres pseudo-aléatoires qui a un petit ensemble d'états S, par exemple $|S| = 2^{12}$ (on nomme un générateur qui possède cette propriété *petit générateur*) et pour lequel les points sont distribués uniformément dans l'hypercube. L'ensemble de points est

$$P_n = \{\mathbf{u}_i = (u_0, u_1, \ldots, u_{t-1}) : s_0 \in S\}$$

qui est l'ensemble des vecteurs de t valeurs successives obtenues à partir de tous les états initiaux.

Les générateurs trouvés et décrits dans cette thèse peuvent être utilisés pour les deux types d'applications. Les générateurs à longues périodes (ou grands générateurs) peuvent être utilisés pour la simulation Monte Carlo et les petits générateurs trouvent leur utilité pour l'intégration quasi-Monte Carlo. On peut choisir les générateurs selon différents critères d'uniformité de P_n. Parmi ceux-ci, on retrouve l'équidistribution. D'autres critères sont mentionnés au chapitre 4.

1.7 Aperçu de la thèse et principales contributions

Dans cette section, nous décrivons la manière dont cette thèse est divisée et discutons de ses principales contributions.

Au prochain chapitre, nous passons en revue les principaux générateurs de nombres aléatoires qui sont les plus utilisés et leurs propriétés de base. Il y a deux sections qui traitent des techniques utilisées pour déterminer l'irréductibilité ou la primitivité du polynôme caractéristique de la matrice X de l'équation (1.1). Il s'agit de techniques importantes pour déterminer la période d'un générateur. Ces techniques sont implantées dans la bibliothèque informatique REGPOLY [84] dont une description

est donnée à la section 4.8. À la fin du chapitre, on explique brièvement comment vérifier la qualité d'un générateur à l'aide d'un test statistique.

Au chapitre 3, nous discutons des propriétés des réseaux polynômiaux (traduction de l'anglais « polynomial lattice »). Dans cette section, on définit les deux types de réseaux que peuvent engendrer les générateurs qui entrent dans le cadre des équations (1.1)–(1.4), soit le réseau en dimension et le réseau en résolution. Pour des raisons qui seront expliquées, nous privilégierons l'étude du réseau en résolution et la majeure partie du chapitre sera consacrée à celle-ci. Nous illustrons ses propriétés à l'aide d'un exemple concret afin de guider le lecteur vers une meilleure compréhension. Le but de ce chapitre est d'offrir une base pour l'introduction de théorèmes et de propriétés dans les chapitres subséquents. Une contribution de ce chapitre est de démontrer comment trouver une base d'un réseau basé sur le corps fini $\mathbb{F}_2$ à partir d'un réseau basé sur $\mathbb{F}_{2^w}$. Ceci permettra de démontrer le théorème 5.3 qui donne une borne sur l'équidistribution pour certains générateurs introduits au chapitre 5.

Au chapitre 4, nous présentons les critères d'uniformité, avec lesquels on juge un ensemble de points $P_n \subset [0,1)^t$, qui seront utilisés dans les chapitres subséquents. Ceux-ci sont l'équidistribution, la q-valeur (qui est semblable, dans une certaine mesure, à l'équidistribution) et la plus courte distance entre deux points de P_n (qu'on appelle distance minimale). La principale contribution de cette section est une adaptation d'un algorithme de Khuller et Matias [23]. Cette adaptation permet de trouver la distance minimale pour les ensembles de points qui sont construits à l'aide d'un réseau digital (cette notion est expliquée dans ce chapitre). Elle profite de certaines propriétés des ensembles de points considérés (les réseaux digitaux) pour accélérer le calcul de la distance minimale. La méthode est particulièrement efficace quand l'ensemble de points est en deux dimensions. Les ensembles de points ayant une grande distance minimale sont importants pour certaines applications, par exemple, en graphisme [26]. Finalement, on donne une brève description de REGPOLY [84], qui est

une librairie de fonctions informatiques qui permet d'évaluer l'uniformité de tous les générateurs avec tous les critères dont il est question dans cette thèse. Bien que peu de lignes soient utilisées pour décrire REGPOLY, celui-ci est une des principales contributions de cette thèse. Le développement de REGPOLY a commencé lors de ma maîtrise et se continue depuis. Au cours de mes études au doctorat, j'ai revu la conception et beaucoup de choses ont changé depuis la version de la fin de ma maîtrise. En plus de la conception générale de la bibliothèque, plusieurs algorithmes ont été améliorés, dont celui qui permet de trouver le plus court vecteur dans un réseau polynômial. Six implantations différentes ont été testées afin d'arriver à l'implantation actuelle dans le but de minimiser le temps de calcul. Pour donner un ordre de grandeur de l'amélioration de la dernière implantation par rapport à la première, pour calculer le plus court vecteur du réseau en résolution décrit par le fameux générateur MT19937 [69], le temps de calcul est passé de plusieurs heures à 2 minutes. En outre, plusieurs modules ont été ajoutés pour tenir compte des générateurs introduits dans cette thèse et des nouveaux générateurs de Marsaglia [60] (voir chapitre 8).

Le chapitre 5 présente une des plus importantes contributions de cette thèse. On y introduit deux types de générateurs qui sont basés sur des récurrences linéaires dans le corps $\mathbb{F}_{2^w}$: soit le générateur à congruence linéaire polynômial dans $\mathbb{F}_{2^w}[z]/P(z)$ et le registre à décalage à rétroaction linéaire dans $\mathbb{F}_{2^w}$. Le théorème 5.3, qui donne une borne sur l'équidistribution pour certains choix de paramètres, vient généraliser un résultat déjà connu sur les limitations de l'équidistribution d'un type de TGFSR sans tempering [67]. À la section 5.6, nous énumérons certaines propriétés qui concernent l'équidistribution de ces générateurs lorsqu'on ne considère que certaines projections. Une des contributions importantes de ce chapitre est le corollaire 5.7 (qui découle du théorème 5.5) qui prédit que seulement une très petite fraction des projections en deux dimensions ne présentent pas une parfaite équidistribution (jusqu'à une certaine résolution) quelque soit le choix des paramètres.

Une recherche pour de bons générateurs a été faite et les résultats sont donnés sous formes de tableaux. Le critère de la recherche est basé sur l'équidistribution. Les générateurs pour lesquels nous avons fait des recherches ont des périodes de $2^{96} - 1$, $2^{256} - 1$, $2^{416} - 1$ et $2^{800} - 1$. Étant donné que l'arithmétique dans $\mathbb{F}_{2^w}$ n'est pas simple, il a fallu trouver des méthodes efficaces pour implanter ces types de générateurs sur ordinateur. Nous proposons deux techniques d'implantations efficaces qui utilisent des tableaux de valeurs pré-calculées et donnons également le code en langage C de générateurs qui utilisent ces techniques. Les temps d'exécution des générateurs résultants sont semblables à ceux d'autres générateurs courants (par exemple, le MT19937). Certains générateurs ont été testés à l'aide de batteries de tests et les ont tous passés avec succès.

Nous avons aussi développé la bibliothèque informatique F2wStreams [83]. Celle-ci implante les générateurs basés sur des récurrences linéaires dans $\mathbb{F}_{2^w}$. Elle a été écrite en langage C et peut être utilisée dans n'importe quelle application de simulation stochastique. Une caractéristique intéressante de cette bibliothèque est que la séquence produite par les générateurs implantés peut être séparée en sous-séquences, où chaque sous-séquence représente une source indépendante de nombres aléatoires. Ceci est très utile pour plusieurs types d'applications où la synchronisation [28] des nombres aléatoires et la disponibilité de plusieurs sources de hasard sont importantes [49]. Le guide d'utilisation de F2wStreams peut être consulté à l'annexe C.

Au chapitre 6, nous prenons les récurrences sur lesquelles on a construit les générateurs du chapitre 5 pour construire des ensembles de points de petite cardinalité (moins que 2^{18} points) très uniformes dans l'hypercube unitaire $[0, 1)^t$. Ce type d'ensembles de points est utile pour l'intégration quasi-Monte Carlo. Pour arriver à notre but, il a fallu modifier quelque peu la récurrence et la sortie par rapport à ce qui a été défini au chapitre 5. Ces modifications permettent d'obtenir des ensembles de points plus uniformes et une sortie qui n'est pas limitée dans sa résolution L qui,

en principe, pourrait être infinie. Parmi les contributions de ce chapitre, on retrouve deux théorèmes qui justifient les modifications des récurrences. La justification se fait en démontrant une borne sur l'équidistribution de l'ensemble de points produits.

Grâce à REGPOLY, on trouve plusieurs ensembles de paramètres pour de bons ensembles de points. Les critères de recherche sont basés sur l'équidistribution, la q-valeur (une généralisation de l'équidistribution) et la distance minimale. Les critères considérés observent plusieurs projections en faible dimension de l'ensemble de points. Des tables des meilleurs ensembles de points sont données.

Une implantation des constructions introduites dans ce chapitre a été programmée dans SSJ [39, 44]. À partir de celle-ci, on a pu mettre à l'épreuve les ensembles de points trouvés avec REGPOLY. Pour des fonctions « test » , on a estimé la variance de l'estimateur quasi-Monte Carlo randomisé (σ^2_{QMC}) et la variance de l'estimateur Monte Carlo standard (σ^2_{MC}). Le rapport $\sigma^2_{\text{MC}}/\sigma^2_{\text{QMC}}$, qu'on appelle le *facteur de réduction de variance*, est utilisé comme critère de performance. Les résultats obtenus par nos nouveaux ensembles de points sont comparés avec ceux obtenus par des ensembles de points de Sobol [98, 99]. Les nouveaux ensembles de points performent mieux que ceux de Sobol pour certaines fonctions et, pour d'autres, obtiennent des résultats qui sont semblables. Il arrive rarement qu'un de nos ensembles de points fasse beaucoup moins bien qu'un ensemble de points de Sobol.

Au chapitre 7, nous introduisons une nouvelle construction de générateurs de nombres aléatoires. Ces nouveaux générateurs tentent de profiter de l'architecture des ordinateurs afin de construire des récurrences rapides. Pour la plupart des générateurs courants, on définit un récurrence sur un corps fini et, ensuite, on tente d'obtenir une implantation qui soit efficace du point de vue de la vitesse d'exécution et de la mémoire. Souvent on y réussit en n'utilisant qu'un sous-ensemble de tous les paramètres possibles de la récurrence. Par exemple, pour les générateurs à récurrences multiples, dans [50], on propose d'utiliser des coefficients de la forme $a = \pm 2^q \pm 2^r$ pour

obtenir une implantation rapide. Ainsi, ces restrictions sur le choix des paramètres peuvent faire en sorte qu'on n'obtient pas les meilleurs générateurs possibles avec le type de récurrence choisi. Pour les nouveaux générateurs introduits dans ce chapitre, on fait plutôt le raisonnement inverse : on définit la récurrence à partir des opérations qui sont rapides sur un ordinateur.

Cette stratégie a été payante. Nous avons trouvé des générateurs rapides et qui sont d'une qualité exceptionnelle du point de vue de l'équidistribution. Voici quelques bonnes caractéristiques de ces générateurs.

- Ils sont rapides : la vitesse d'exécution de l'implantation est comparable aux bons générateurs les plus rapides (MT19937 [69] et TT800 [67]) et est nettement mieux que celle des générateurs à récurrences multiples (notamment celle du MRG32k3a [36]).
- Il est possible d'identifier des générateurs qui ont une très longue période : nous avons trouvé des générateurs qui ont des périodes de $2^{19937}-1$, $2^{21701}-1$, $2^{23209}-1$ et même $2^{44497}-1$.
- Ils ont une excellente équidistribution : l'équidistribution est quasiment parfaite et peut le devenir en transformant quelque peu la sortie du générateur.
- Ils ont réussi tous les tests statistiques qu'on leur a fait passer.

Nous présentons l'implantation d'un générateur dont la période est $2^{44497}-1$. Nous avons fait des tests statistiques et des tests de vitesse d'exécution avec ce générateur et les résultats démontrent qu'il s'agit, statistiquement, d'un générateur de qualité et que la vitesse d'exécution de l'implantation est comparable à celle des meilleurs.

Un défaut important des générateurs comme le MT19937 et le TT800 est leur sensibilité à la mauvaise initialisation du générateur. Si l'état initial comporte plusieurs bits nuls, alors les états successifs auront aussi cette propriété pour plusieurs itérations ce qui fait que la sortie, qui est supposée être uniformément distribuée sur $[0,1)$, a un fort biais vers 0. Ceci implique que si on génère deux séquences obtenues

avec deux initialisations très similaires, alors les deux séquences seront très fortement corrélées (pour ne pas dire identiques) pour un grand nombre de valeurs. La capacité d'un générateur de contrer les effets d'une mauvaise initialisation est appelée *la capacité de diffusion.* Il se trouve que les nouveaux générateurs qu'on introduit au chapitre 7 ont une excellente capacité de diffusion. Une expérience empirique est conduite et démontre clairement le peu de sensibilité à une mauvaise initialisation de nos nouveaux générateurs. La capacité de diffusion est importante pour un utilisateur qui n'a aucune idée de la structure des générateurs de nombres aléatoires. Souvent, l'utilisateur n'est pas sensibilisé à la nécessité de bien choisir l'initialisation. Si la capacité de diffusion est grande, l'utilisateur néophyte a moins de chance d'être pénalisé pour le biais dû à la mauvaise initialisation.

Au dernier chapitre, nous analysons une famille de générateurs introduite par Marsaglia [60]. Ces générateurs utilisent une récurrence très simple : l'implantation nécessite seulement trois décalages de bits et quelques additions de vecteurs de bits. La stratégie employée par Marsaglia pour obtenir des générateurs rapides est semblable à celle des générateurs introduits au chapitre 7 : il utilise des opérations rapides pour définir une récurrence. Nous analysons les récurrences proposées par Marsaglia et démontrons des relations de similarité entre certaines récurrences qui nous permettent de déduire l'équidistribution et la période de générateurs « similaires ». Nous trouvons les meilleurs générateurs possibles à l'aide d'un critère basé sur l'équidistribution et nous vérifions leurs qualités statistiques avec la batterie de tests TESTU01 [48]. On montre, par ces tests, que les récurrences sont trop simples pour produire des générateurs de nombres aléatoires de qualité.

Chapitre 2

Théorie des générateurs utilisant une récurrence linéaire modulo 2

Dans ce chapitre, il est question de la théorie des générateurs utilisant des récurrences linéaires modulo 2. On discute de différents générateurs qui existent présentement dans la littérature scientifique : les GFSR, les générateurs de type GCL polynômial, les générateurs de Tausworthe, les TGFSR, les Mersenne twister et les automates cellulaires. Également, on explique deux transformations linéaires applicables à la sortie des générateurs qui permettent d'améliorer la qualité des nombres produits du point vue de l'uniformité.

De plus, nous avons deux sections qui discutent de la détermination de la primitivité du polynôme caractéristique de la matrice X. Finalement, nous discutons brièvement de la manière de vérifier la qualité d'un générateur de nombres aléatoires à l'aide de tests statistiques.

2.1 Récurrence de base pour les générateurs utilisant une récurrence linéaire modulo 2

Dans cette section, afin de mieux comprendre les générateurs qui entrent dans le cadre des équations (1.1)-(1.4), nous examinons plus en détail la récurrence dans $\mathbb{F}_2$

$$x_j = (a_1 x_{j-1} + \cdots + a_k x_{j-k}) \bmod 2, \tag{2.1}$$

où k est appelé *l'ordre de la récurrence* si $a_k \neq 0$ et $a_i \in \mathbb{F}_2$ pour $i = 1, \ldots, k$. Cette récurrence est très utile puisqu'elle nous permettra d'analyser tous les générateurs à récurrence linéaire modulo 2.

À cette récurrence est associé un polynôme dans $\mathbb{F}_2[z]$ qu'on appelle *polynôme caractéristique de la récurrence.* Ce polynôme est

$$P(z) = z^k - a_1 z^{k-1} - \cdots - a_k. \tag{2.2}$$

Grâce à celui-ci, on peut déduire des informations importantes sur la récurrence, dont sa période. En effet, il est bien connu que la récurrence (2.1) est de période maximale $2^k - 1$ si et seulement si son polynôme caractéristique est primitif [56].

À une initialisation donnée $x_0, \ldots, x_{k-1}$ de la récurrence (2.1), on associe la *fonction génératrice*

$$G(z) = x_0 z^{-1} + x_1 z^{-2} + \cdots + x_n z^{-n} + \cdots = \sum_{n=1}^{\infty} x_{n-1} z^{-n}.$$

Cette fonction n'en est pas une au sens propre, mais simplement une expression formelle. Ce $G(z)$ fait partie du corps des séries formelles de Laurent sur $\mathbb{F}_2$. Ce corps est noté $\mathbb{L}_2$ et comprend toutes les séries de la forme

$$\pi(z) = \alpha_n z^n + \alpha_{n-1} z^{n-1} + \cdots,$$

où $n \in \mathbb{Z}$ et $\alpha_i \in \mathbb{F}_2, i \leq n$.

On peut démontrer, en adaptant la preuve du théorème 6.40 de [56], le théorème suivant.

Théorème 2.1. *Soit $\{x_n\}_{n\geq 0}$ une séquence qui suit la récurrence (2.1), $P(z) \in \mathbb{F}_2[z]$ le polynôme caractéristique de cette récurrence et $G(z) \in \mathbb{L}_2$ sa fonction génératrice. L'identité*

$$G(z) = \frac{g(z)}{P(z)} \tag{2.3}$$

est valide avec

$$g(z) = -\sum_{j=0}^{k-1} \sum_{i=0}^{k-1-j} a_{k-i-j-1} x_i z^j \in \mathbb{F}_2[z]$$

où l'on pose $a_0 = -1$. De manière opposée, si $g(z)$ est un polynôme quelconque sur $\mathbb{F}_2$ avec $deg(g(z)) < k$ et si $P(z)$ est donné par (2.2), alors la série de Laurent $G(z)$ définie par (2.3) est une fonction génératrice de la récurrence linéaire modulo 2 donnée par (2.1).

Ce théorème nous permet d'associer, pour la récurrence (2.1), à une initialisation donnée, un polynôme $g(z)$ dans $\mathbb{F}_2[z]/P(z)$, l'anneau des polynômes modulo $P(z)$.[1] Il existe donc une bijection entre chacune des initialisations possibles et chacun des éléments de $\mathbb{F}_2[z]/P(z)$.

2.2 Propriétés des générateurs utilisant une récurrence linéaire modulo 2

Dans cette section, nous étudions les propriétés des générateurs qui entrent dans le cadre défini par les équations (1.1)–(1.4). Le polynôme caractéristique des générateurs à récurrence linéaire modulo 2 est le polynôme caractéristique de la matrice X. Si on connaît ce polynôme caractéristique, il est possible de vérifier si la période des états

[1] Tout comme $\mathbb{Z}_p = \mathbb{Z}/p$ est l'anneau des entiers modulo p.

visités (les $\mathbf{x}_n$) par le générateur est maximale. En effet, la période est maximale, c'est-à-dire de $2^k - 1$, si et seulement si le le polynôme caractéristique est primitif dans $\mathbb{F}_2$ et est de degré k.

Ce qui suit tente de démontrer l'importance de la récurrence donnée par l'équation (2.1) pour l'analyse des générateurs de nombres aléatoires qui suivent une récurrence linéaire modulo 2. Pour ceci, on suppose que la matrice X est de plein rang ainsi que la matrice B. Dans ce cas, il est trivial de démontrer que la séquence des $\mathbf{z}_n, n \geq 0$ suit la récurrence

$$\mathbf{z}_n = BXB^{-1}\mathbf{z}_{n-1} \tag{2.4}$$

car $\mathbf{x}_n = X\mathbf{x}_{n-1}$ et $\mathbf{z}_n = B\mathbf{x}_n$ ce qui implique que $\mathbf{x}_n = B^{-1}\mathbf{z}_n$ et que $B^{-1}\mathbf{z}_n = XB^{-1}\mathbf{z}_{n-1}$. Cette dernière implique (2.4).

Soit $P_X(z) = \det(X - zI)$, le polynôme caractéristique de la matrice X. La proposition suivante permet de caractériser la récurrence décrite par (2.4).

Proposition 2.1. *[76]*
Soit $\{\mathbf{z}_n\}_{n\geq 0}$, une séquence de vecteurs qui suit la récurrence donnée par l'équation (2.4). Également, soit $\gamma_i = \{z_n^{(i)}\}_{n\geq 0}$, la séquence de bits observée au bit i des vecteurs $\mathbf{z}_n, n \geq 0$, pour un $\mathbf{z}_0$ donné. Chaque séquence γ_i, $0 \leq i < k$, suit une récurrence linéaire donnée par l'équation (2.1) où le polynôme caractéristique de celle-ci est $P_X(z)$, mais avec une initialisation différente.

Puisque chaque séquence γ_i, $0 \leq i < k$, suit une récurrence linéaire du type (2.1), alors il est possible d'associer à chaque γ_i une fonction génératrice $\tilde{G}_i(z) = \tilde{g}_i(z)/P_X(z)$. De plus, si la matrice Y est telle que chaque ligne est non nulle (comme c'est le cas tout au long de cette thèse), alors, pour $i < L$, on a que $\beta_i = \{y_n^{(i)}\}_{n\geq 0}$ suit également une récurrence linéaire (2.1). On peut aussi associer à chaque séquence β_i, $0 \leq i < L$, une fonction génératrice $G_i(z) = g_i(z)/P_X(z)$.

Ainsi, pour une initialisation donnée $\mathbf{x}_0$ d'un générateur qui suit le cadre donné

par les équations (1.1)-(1.4), on peut identifier, à l'aide du théorème 2.1, un unique vecteur de fonctions génératrices

$$(g_0(z), \ldots, g_{L-1}(z))/P_X(z)$$

qui caractérise entièrement la séquence des vecteurs $\{\mathbf{y}_n\}_{n\geq 0}$. Ce vecteur de fonctions génératrices sera important pour l'étude du réseau en résolution que décrivent tous les générateurs qui entrent dans le cadre défini par (1.1)-(1.4). Cet aspect des générateurs à récurrence linéaire modulo 2 est vu plus en détails au chapitre 4.

2.3 Générateurs à récurrence linéaire existants

Dans cette section, on décrit les principaux générateurs que l'on retrouve dans la littérature scientifique. Principalement, ce qui les différencie est leur matrice X.

2.3.1 Générateur GFSR

Un GFSR [55, 22] suit une récurrence de la forme

$$\mathbf{v}_n = a_1\mathbf{v}_{n-1} + \cdots + a_r\mathbf{v}_{n-r} \tag{2.5}$$

où les $\mathbf{v}_n$, $n \geq 0$ sont des vecteurs de w bits et les $a_i \in \mathbb{F}_2$, $i = 1, \ldots, r$. Habituellement, la sortie est donnée par $\mathbf{y}_n = \mathbf{v}_n$.

L'état du générateur est le vecteur $\mathbf{x}_n = (\mathbf{v}_n^\mathsf{T}, \cdots, \mathbf{v}_{n-r+1}^\mathsf{T})^\mathsf{T}$. Il est facile de voir que chaque bit de $\mathbf{v}_n$ (et par le fait même, chaque bit de $\mathbf{x}_n$) suit la récurrence de base (2.1) avec $k = r$ et les mêmes valeurs de a_i.

Ainsi, à la lumière de la section 2.2, on remarque que tout générateur qui suit le cadre général décrit dans la section 1.4 peut être exprimé par une récurrence de type GFSR. Soit $P_X(z) = z^k - a_1 z^{k-1} - \cdots - a_k$, le polynôme caractéristique de X et

$G_i(z) = g_i(z)/P_X(z) = \sum_{j=0}^{\infty} g_{i,j} z^{-j}$, la fonction génératrice du i-ième bit des $\mathbf{x}_n$, $n \geq 0$. En posant $r = k$, en initialisant les $\mathbf{v}_n = (g_{0,n}, \ldots, g_{k-1,n})$ pour $n = 0, \ldots, k-1$ et en prenant les coefficients de $P_X(z)$ comme valeurs de a_i dans (2.5), on obtient $\mathbf{v}_n = \mathbf{x}_n$, $n \geq 0$.

Pour un générateur à récurrence linéaire d'ordre k, dont la sortie possède L bits de résolution, l'état du générateur est de kL bits, si on la représentait par un GFSR. Optimalement, pour un générateur d'ordre k, l'état est de k bits. Ceci fait en sorte que l'implantation d'un générateur par un GFSR n'est pas efficace au plan de la mémoire. De plus, pour des raisons d'efficacité, on doit se restreindre à des GFSR dont le polynôme caractéristique $P_X(z)$ est un trinôme ou un pentanôme. Les générateurs basés sur un trinôme sont reconnus pour avoir de mauvaises propriétés statistiques [68, 67, 10]. En général, pour avoir de bonnes propriétés statistiques, le polynôme caractéristique doit avoir beaucoup de coefficients non nuls [9, 10]. Pour ces raisons, le développement de générateurs implantés par des GFSR n'est pas l'objet de cette thèse.

2.3.2 Générateur de Tausworthe

Le générateur de Tausworthe utilise explicitement la récurrence de base (2.1). Soit $\{x_j\}_{j \geq 0}$, une séquence dans $\mathbb{F}_2$ qui suit la récurrence (2.1). L'état du générateur de Tausworthe à la n-ième itération est

$$\mathbf{x}_n = (x_{ns}, x_{ns+1}, \ldots, x_{ns+k-1})^{\mathsf{T}}$$

où s est un paramètre entier. Habituellement, le vecteur de bits de sortie associé à cet état du générateur est

$$\mathbf{y}_n = (x_{ns}, x_{ns+1}, \ldots, x_{ns+L-1}) \tag{2.6}$$

et on utilise la sortie définie par (1.4).

On peut facilement voir que ce générateur entre bien dans le cadre général défini antérieurement. La récurrence peut être écrite sous la forme

$$\mathbf{x}_n = \bar{X}^s_{\text{taus}} \mathbf{x}_{n-1}$$

où

$$\bar{X}_{\text{taus}} = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ a_k & a_{k-1} & a_{k-2} & \cdots & a_1 \end{pmatrix}.$$

Pour obtenir la sortie (sans autre transformation linéaire), si $L > k$, on définit la matrice $Y = (I_k \quad C)^\mathsf{T}$ où C permet d'obtenir les $L - w$ bits les moins significatifs de $\mathbf{y}_n$, sinon $Y = I_{L \times k}$, la matrice identité $k \times k$ à laquelle on a amputé les $k - L$ dernières lignes. La matrice B, pour l'instant, est la matrice I_k.

Pour ce générateur, il n'est pas suffisant que le polynôme caractéristique $Q(z)$ de la matrice $\bar{X}_{\text{taus}}$ soit primitif pour obtenir une période maximale. Pour une valeur de s donnée, il est également nécessaire de satisfaire la condition $\text{pgcd}(s, 2^k - 1) = 1$ [33].

Ce générateur peut se concevoir comme une récurrence dans le corps $\mathbb{L}_2$ des séries formelles de Laurent [12]. Soit

$$\pi(z) = \alpha_n z^n + \alpha_{n-1} z^{n-1} + \cdots, \tag{2.7}$$

un élément de $\mathbb{L}_2$ où n est un entier et soient les opérateurs

$$\text{frac}(\pi(z)) = \alpha_{-1} z^{-1} + \alpha_{-2} z^{-2} + \cdots, \tag{2.8}$$

$$\text{trunc}_l(\pi(z)) = \alpha_n z^n + \alpha_{n-1} z^{n-1} + \cdots + \alpha_{-l} z^{-l} \tag{2.9}$$

où $l > 0$ est un entier.

Si on initialise correctement le générateur de Tausworthe avec $\mathbf{x}_0 = (0, 0, \ldots, 0, 1)^\mathsf{T}$ et si la sortie est donnée par (2.6) et (1.4), alors on peut obtenir la même séquence

$\{u_n\}_{n\geq 0}$ comme suit :

$$q_n = \text{trunc}_L(\text{frac}(a(z)^n/Q(z) = \alpha_{n,1}z^{-1} + \cdots + \alpha_{n,L}z^{-L}, \quad n = 0, 1, 2, \ldots \tag{2.10}$$

$$\tilde{u}_n = \sum_{j=1}^{L} \alpha_{n,j}2^{-j}, \quad n = 0, 1, 2, \ldots \tag{2.11}$$

où $a(z) = z^s \bmod Q(z) \in \mathbb{F}_2[z]/Q(z)$ et L est le nombre de bits requis à la sortie. Cette manière de voir le générateur de Tausworthe permet de définir un réseau polynômial sur un sous-ensemble de $\mathbb{L}_2^t$. Ce réseau est utile pour calculer l'équidistribution de ce type de générateur[12]. Ceci est discuté un peu plus en détail au chapitre 4.

2.3.3 Générateur à congruence linéaire polynômial

Le générateur à congruence linéaire polynômial (ou GCL polynômial) peut être construit à partir d'une récurrence dans $\mathbb{F}_2[z]/Q(z)$ où $Q(z) = z^k - a_1z^{k-1} - \cdots - a_k$ est un polynôme primitif de degré k. L'état du générateur, $p_n(z)$, est un élément de $\mathbb{F}_2[z]/Q(z)$ et suit la récurrence

$$p_n(z) = a(z)p_{n-1}(z) \bmod Q(z) \tag{2.12}$$

où $a(z) = z^s$ est un élément de $\mathbb{F}_2[z]/Q(z)$. Tout comme le générateur de Tausworthe, celui-ci est de période maximale si et seulement si $\text{pgcd}(2^k - 1, s) = 1$.

À chaque $p_n(z) = c_1z^{k-1} + \cdots + c_k$, associons le vecteur de bits $\mathbf{x}_n = (c_k, \ldots, c_1)$. La récurrence (2.12) peut maintenant être exprimée sous forme matricielle par

$$\mathbf{x}_n = \bar{X}_{\text{gcl}}^s \mathbf{x}_{n-1} \tag{2.13}$$

où

$$\bar{X}_{\text{gcl}} = \begin{pmatrix} & & & a_k \\ 1 & & & a_{k-1} \\ & 1 & & a_{k-2} \\ & & \ddots & \vdots \\ & & 1 & a_1 \end{pmatrix}.$$

Fait intéressant à remarquer, $\bar{X}_{\text{gcl}} = \bar{X}^{\mathsf{T}}_{\text{taus}}$ pour un polynôme $Q(z)$ donné. La différence entre les deux générateurs se trouve dans la manière de générer le prochain $\mathbf{x}_n$ quand $s = 1$. Dans le cas du générateur de Tausworthe, on décale les bits de $\mathbf{x}_{n-1}$ de 1 position vers la gauche et on met à jour le dernier bit, $x_n^{(k-1)}$, avec une combinaison linéaire des bits de $\mathbf{x}_{n-1}$. Dans le cas du GCL polynômial, on décale les bits de $\mathbf{x}_{n-1}$ de 1 position vers la droite et on modifie plusieurs bits selon la valeur du dernier bit de $\mathbf{x}_{n-1}$. Le point important est que dans le cas du générateur de Tausworthe, on modifie 1 bit en examinant plusieurs bits et dans le cas du GCL polynômial, on modifie plusieurs bits en tenant compte d'un seul bit. Nous reviendrons sur cette propriété lorsqu'on présentera les générateurs introduits au chapitre 5. Ces nouveaux générateurs se veulent une généralisation des GCL polynômiaux et des générateurs de Tausworthe : au lieu d'utiliser le corps fini $\mathbb{F}_2$, ils utilisent les corps fini $\mathbb{F}_{2^w}$. Au chapitre 7, nous introduisons d'autres nouveaux générateurs qui tentent de généraliser cette propriété. Ces derniers utilisent l'information contenue dans plusieurs bits afin d'en modifier plusieurs autres. Il en résulte des générateurs de très bonne qualité.

2.3.4 Générateur TGFSR

Le TGFSR [66, 67] est basé sur la récurrence

$$\mathbf{v}_n = \mathbf{v}_{n+m-r} + A\mathbf{v}_{n-r}, \tag{2.14}$$

où A est une matrice avec ses éléments dans $\mathbb{F}_2$, de dimension $w \times w$, et les $\mathbf{v}_i \in \mathbb{F}_2^w$ sont des vecteurs colonnes. En choisissant de façon adéquate les valeurs de w, r, m et A, il est possible d'atteindre une période de $2^{rw} - 1$, qui est la période maximale pour ce type de générateur.

On constate que ce générateur entre dans le cadre général décrit à la section 1.4,

puisque

$$\begin{pmatrix} \mathbf{v}_n \\ \mathbf{v}_{n-1} \\ \vdots \\ \mathbf{v}_{n-r+2} \\ \mathbf{v}_{n-r+1} \end{pmatrix} = X \begin{pmatrix} \mathbf{v}_{n-1} \\ \mathbf{v}_{n-2} \\ \vdots \\ \mathbf{v}_{n-r+1} \\ \mathbf{v}_{n-r} \end{pmatrix} \tag{2.15}$$

où

$$X = \begin{pmatrix} & & & I_w & & A \\ I_w & & & & & \\ & I_w & & & & \\ & & I_w & & & \\ & & & \ddots & & \\ & & & & I_w & \end{pmatrix} \tag{2.16}$$

et l'élément $(0,0)$ de la matrice I_w de la première ligne de X est l'élément $(0,(r-m-1)w)$ de X. L'état du générateur est $\mathbf{x}_n = (\mathbf{v}_n^\mathsf{T}, \mathbf{v}_{n-1}^\mathsf{T}, \ldots, \mathbf{v}_{n-r+1}^\mathsf{T})^\mathsf{T}$. La sortie est habituellement donnée par $\mathbf{y}_n = \mathbf{v}_n$ et $L = w$.

Les auteurs de [66] décrivent les avantages des TGFSR par rapport aux GFSR. On y retrouve aussi le théorème suivant sur les conditions nécessaires et suffisantes pour atteindre la période maximale.

Théorème 2.2. *(Matsumoto et Kurita [66])*
Soit $\phi_A(t)$, le polynôme caractéristique de la matrice A. Le générateur a une période maximale de $2^{rw}-1$ si et seulement si $\phi_A(t^r+t^m)$ est un polynôme primitif de degré rw. Dans ce cas, chaque bit du vecteur $\mathbf{v}_i$ suit une récurrence de la forme (2.1), avec polynôme caractéristique $\phi_A(t^r+t^m)$.

Dans [66], les auteurs utilisent la matrice

$$\bar{X}_{\text{gcl}} = \begin{pmatrix} & & & a_k \\ 1 & & & a_{k-1} \\ & 1 & & a_{k-2} \\ & & \ddots & \vdots \\ & & 1 & a_1 \end{pmatrix} \tag{2.17}$$

comme matrice A.

2.3.5 Générateur Mersenne twister

Le Mersenne twister a été introduit pour la première fois dans [69] et est une généralisation du TGFSR. La récurrence a la forme

$$\mathbf{v}_n = \mathbf{v}_{n+m-r} + A(\mathbf{v}_{n-r}^h | \mathbf{v}_{n-r+1}^b). \tag{2.18}$$

Cette récurrence dépend de cinq paramètres : les entiers r et w, un entier p, $0 \leq p \leq w-1$, un entier m, $0 \leq m \leq r$, et une matrice A dont les entrées sont dans $\mathbb{F}_2$. Les vecteurs $\mathbf{v}_n$ sont des vecteurs colonnes. Le symbole $\mathbf{v}_{n-r}^h$ représente les $w-p$ premiers bits de $\mathbf{v}_{n-r}$ tandis que $\mathbf{v}_{n-r+1}^b$ représente les p derniers bits de $\mathbf{v}_{n-r+1}$. L'opération $|$ représente la concaténation des opérandes. De plus, si $p = 0$, alors la récurrence (2.18) devient (2.14), d'où la généralisation du TGFSR.

On montre maintenant que ce générateur entre dans le cadre général défini à la section 1.4. L'état du Mersenne twister est représenté sur $rw-p$ bits, ce qui implique que la période maximale de ce type de générateur est de $2^{rw-p}-1$. L'état est

$$\mathbf{x}_n = (\mathbf{v}_n^\mathsf{T}, \mathbf{v}_{n-1}^\mathsf{T}, \ldots, \mathbf{v}_{n-r+2}^\mathsf{T}, \text{trunc}_{w-p}(\mathbf{v}_{n-r+1})^\mathsf{T})^\mathsf{T}$$

où $\text{trunc}_{w-p}((\mathbf{v}_{n-r+1})^\mathsf{T})$ est le vecteur de $w-p$ bits composé des $w-p$ premiers bits

de $(\mathbf{v}_{n-r+1})^{\mathsf{T}}$. La matrice X de dimension $(rw-p) \times (rw-p)$ est

$$X = \begin{pmatrix} & & I_w & & S \\ I_w & & & & \\ & I_w & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & I_{w-p} \end{pmatrix} \tag{2.19}$$

où

$$S = A \begin{pmatrix} & I_{w-p} \\ I_p & \end{pmatrix}$$

est une matrice $w \times w$. La sortie est habituellement donnée par $\mathbf{y}_n = \mathbf{v}_n$ et $L = w$.

Le polynôme caractéristique de la récurrence dépend de la matrice A et une formule permettant de déterminer celui-ci est donnée dans [69]. Dans cette dernière référence, ils utilisent une matrice de forme (2.17) comme matrice A.

2.3.6 Méthode matricielle à récurrences multiples

La méthode matricielle à récurrences multiples (MMRM) [75, 76, 77] est une autre généralisation de la récurrence utilisée par les TGFSR. Par abus de langage, on parlera de *générateur MMRM* pour tout générateur qui utilise cette méthode. La récurrence est

$$\mathbf{v}_n = A_1\mathbf{v}_{n-1} + A_2\mathbf{v}_{n-2} + \cdots + A_r\mathbf{v}_{n-r} \tag{2.20}$$

où les $\mathbf{v}_n$ sont des vecteurs de w bits et les matrices A_i, $i = 1, \ldots, r$, sont de dimensions $w \times w$. L'état du générateur est $\mathbf{x}_n = (\mathbf{v}_n^{\mathsf{T}}, \cdots, \mathbf{v}_{n-r+1}^{\mathsf{T}})^{\mathsf{T}}$ et la matrice de transition

de ce type de générateur est donnée par

$$X = \begin{pmatrix} A_1 & A_2 & \dots & A_{r-1} & A_r \\ I_w & & & & \\ & I_w & & & \\ & & \ddots & & \\ & & & I_w & \end{pmatrix}. \tag{2.21}$$

La sortie est habituellement donnée par $\mathbf{y}_n = \mathbf{v}_n$ et $L = w$. Le théorème suivant permet de déterminer si un générateur MMRM a la période maximale.

Théorème 2.3. *(Niederreiter [75])*
N'importe quelle séquence générée par (2.20) a une période inférieure ou égale à $2^{rw} - 1$. *De plus, la période est* $2^{rw} - 1$ *si et seulement si*

$$\det\left(z^r I_w + \sum_{i=1}^{r} z^{r-i} A_i\right) \tag{2.22}$$

est un polynôme primitif sur $\mathbb{F}_2$.

Ce type de générateur est aussi une généralisation d'une récurrence décrite au chapitre 5. Cette dernière est basée sur une récurrence linéaire dans $\mathbb{F}_{2^w}$ similaire à la récurrence (2.1) qui est dans $\mathbb{F}_2$.

Mentionnons que, dans [77], on utilise la récurrence (2.20) afin d'obtenir des vecteurs pseudo-aléatoires dans $[0,1)^w$, mais dans le contexte où $\mathbb{F}_2$ est remplacé par $\mathbb{F}_p$ où p est un entier premier assez grand. Soit la fonction $h : \mathbb{F}_p^w \to [0,1)^w$ définie par

$$h(\mathbf{v}) = (v^{(0)}, \dots, v^{(w-1)})/p \tag{2.23}$$

où $\mathbf{v} = (v^{(0)}, \dots, v^{(w-1)})$ et on prend la valeur de chaque coordonnée du vecteur comme un nombre réel. Le n-ième vecteur en w dimensions est obtenu par

$$\mathbf{u}_n = h(\mathbf{v}_n) \in [0,1)^w.$$

Il s'agit d'une approche bien différente de celle qu'on utilise dans le cadre défini par les équations (1.1)-(1.4). Dans le cas où $p = 2$, avec cette méthode, on ne peut générer que des vecteurs qui ont pour coordonnées les valeurs 0 et 1/2. Ceci n'est pas acceptable pour un générateur de nombres aléatoires, mais si $p > 2$, cette méthode est susceptible de construire de bons générateurs.

2.3.7 Générateurs basés sur un automate cellulaire

Un automate cellulaire correspond à un tableau $M(n)$ de dimension $p \times m$, avec ses éléments dans $\mathbb{F}_2$, et qui évolue dans le temps représenté par n. On appelle chaque élément $m_{i,j}(n), 0 \leq i < p, 0 \leq j < m$ du tableau $M(n)$ une *cellule*. Pour ce système, le temps évolue de façon discrète et à chaque incrément du temps, la valeur de chacune des cellules est modifiée selon la valeur des cellules voisines et selon des règles préétablies. Ce type de générateurs est surtout utilisé dans la vérification de composantes électroniques. Ils sont implantés de façon matérielle plutôt que logicielle.

Dans le cas où $p = 1$ et $m = k$, on obtient un automate cellulaire en 1 dimension. Pour les besoins de cette thèse, nous ne considérerons que les règles qui correspondent à une transformation linéaire et pour lesquelles le prochain état d'une cellule ($m_{1,j}(n+1)$) est déterminé à partir de l'état de ses voisins les plus proches. On spécifie par v la taille du voisinage. Si on note $x_n^{(j)} = m_{1,j}(n)$, ce type d'automate cellulaire suit la récurrence

$$x_n^{(j)} = \sum_{s=-v}^{v} a_{j,s} x_{n-1}^{(j+s)}, \quad j = 0, \ldots, k-1, \tag{2.24}$$

où les $a_{j,s} \in \mathbb{F}_2$ sont fixes, $x_n^{(\ell)} = 0$ pour $\ell \notin \{0, 1, \ldots, k-1\}$ et $a_{j,s} = 0$ si $(j,s) \notin \{(j,s) : 0 \leq j < k, -v \leq s \leq v\}$.

Par exemple, supposons que $v = 1$. Dans ce cas, l'état suivant d'une cellule $(x_{n+1}^{(j)})$

est déterminé à partir de l'état de ses voisins immédiats ($x_n^{(j-1)}$ et $x_n^{(j+1)}$) et de son état actuel ($x_n^{(j)}$). Il est facile de voir que ce type d'automate cellulaire entre bien dans le cadre d'un générateur à récurrence linéaire modulo 2 défini à la section 1.4. L'état de ce générateur est le vecteur de bits $\mathbf{x}_n = (x_n^{(0)}, \ldots, x_n^{(k-1)})$ et la matrice de transition X est

$$X = \begin{pmatrix} a_{0,0} & a_{0,1} & & & & & & \\ a_{1,-1} & a_{1,0} & a_{1,1} & & & & & \\ & a_{2,-1} & a_{2,0} & a_{2,1} & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & & a_{k-2,-1} & a_{k-2,0} & a_{k-2,1} & \\ & & & & & a_{k-1,-1} & a_{k-1,0} & \end{pmatrix} \tag{2.25}$$

qui est de dimension $k \times k$. Le polynôme caractéristique de ce type de générateur est le polynôme caractéristique de la matrice X.

La manière dont une cellule est mise à jour est appelée *règle de mise à jour*. Des exemples de règles linéaires courantes (quand $v = 1$) sont les règles 90 et 150. Si une cellule j suit la règle 90, alors celle-ci sera mise à jour par l'équation

$$x_n^{(j)} = x_{n-1}^{(j-1)} + x_{n-1}^{(j+1)}, \tag{2.26}$$

donc $a_{j,-1} = 1, a_{j,0} = 0, a_{j,1} = 1$ pour cette cellule. Si c'est la règle 150 qui est appliquée à la cellule j, alors celle-ci sera mise à jour par l'équation

$$x_n^{(j)} = x_{n-1}^{(j-1)} + x_{n-1}^{(j)} + x_{n-1}^{(j+1)}, \tag{2.27}$$

donc $a_{j,-1} = 1, a_{j,0} = 1, a_{j,1} = 1$ pour cette cellule. Cette manière d'affecter un numéro à une règle a été introduite par Wolfram [107].

Dans la littérature, il existe beaucoup de types d'automates cellulaires à 1 dimension dont les règles sont linéaires. Certains ont pleine période $2^k - 1$ [104, 65].

Dans le cas où $p > 1$, chaque cellule évolue selon une récurrence de la forme

$$m_{i,j}(n) = \sum_{(v,w)\in V} a_{i,j,v,w} m_{i+v,j+w}(n-1), \quad 0 \leq i < p, 0 \leq j < m \tag{2.28}$$

où $V = \{(0,0),(0,1),(1,0),(-1,0),(0,-1)\}$, les $a_{i,j,v,w} \in \mathbb{F}_2$, $m_{i,j}(n) = 0$ pour $i \notin \{0,1,\ldots,p-1\}$ ou $j \notin \{0,1,\ldots,m-1\}$ et $a_{i,j,v,w} = 0$ si $(i+v) \notin \{0,1,\ldots,p-1\}$ ou si $(j+w) \notin \{0,1,\ldots,m-1\}$. Dans cette récurrence, la taille du voisinage d'une cellule est 5 cellules, soit la cellule de gauche, de droite, du haut, du bas et elle-même. On appelle ce type d'automate cellulaire *automate cellulaire en 2 dimensions* [81], puisqu'il correspond à un tableau de valeurs en 2 dimensions qui évolue dans le temps.

Plus récemment, on a introduit des automates cellulaires dont la valeur que peut prendre chacune des cellules n'est pas dans $\mathbb{F}_2$, mais dans $\mathbb{F}_{2^w}$ pour une valeur de $w > 1$ [95, 87, 86].

2.4 « Tempering » permettant d'améliorer l'équidistribution

À la section 2.3, on a discuté de différents types de générateurs de nombres aléatoires qui existent dans la littérature (c'est-à-dire de différents types de matrices X). Lors de l'implantation de ces générateurs, la matrice B est très souvent la matrice identité. Dans [67], les auteurs utilisent pour la première fois une matrice B différente de l'identité sur leurs TGFSR. Les matrices A des TGFSR qu'ils utilisent sont de la

forme

$$A = \begin{pmatrix} & & & a_0 \\ 1 & & & a_1 \\ & 1 & & a_2 \\ & & \ddots & \vdots \\ & & 1 & a_{w-1} \end{pmatrix}. \tag{2.29}$$

Le problème avec cette matrice A (avec $B = I_w$) est que l'équidistribution est telle que $\ell_t \leq 2$ pour tout $t \geq r$, bien loin de la borne supérieure $\ell_t^* = \lfloor rw/t \rfloor$. Afin d'améliorer l'équidistribution, ils introduisent une transformation linéaire, le tempering de Matsumoto-Kurita. Malgré cette transformation linéaire, l'équidistribution de ce type de générateur est bornée par

$$t_\ell \leq r\lfloor w/\ell \rfloor.$$

On trouve un exemple d'utilisation d'un autre type de tempering dans [69] dans lequel on applique une transformation linéaire similaire au tempering de Matsumoto-Kurita. Mais encore une fois, les auteurs n'arrivent pas à atteindre la borne supérieure pour plusieurs valeurs de ℓ.

Dans [82], on applique diverses transformations linéaires sur un générateur de type GCL polynômial et on obtient un générateur qui est équidistribué au maximum (ME). Mais ce générateur est deux fois plus lent que la version sans tempering.

2.4.1 Tempering de Matsumoto-Kurita

Le tempering de Matsumoto-Kurita a été introduit dans [67] et a été modifié dans [69] et [45]. Il s'agit d'opérations qui s'effectuent rapidement sur un ordinateur et qui ont démontré une bonne capacité à produire de bons générateurs. En combinant les

idées de toutes ces références, on pourrait le définir par les opérations suivantes :

$$\begin{aligned}
\overline{\mathbf{x}}_n &= \text{trunc}_w(\mathbf{x}_n) \\
\mathbf{s}_n &= \overline{\mathbf{x}}_n \oplus (\overline{\mathbf{x}}_n \gg u) \\
\mathbf{r}_n &= \mathbf{s}_n \oplus ((\mathbf{s}_n \ll s_1) \,\&\, \mathbf{b}) \\
\mathbf{t}_n &= \mathbf{r}_n \oplus ((\mathbf{r}_n \ll s_2) \,\&\, \mathbf{c}) \\
\mathbf{z}_n &= \mathbf{t}_n \oplus (\mathbf{t}_n \gg l)
\end{aligned}$$

où w, u, s_1, s_2 et l sont des entiers et $\overline{\mathbf{x}}_n$, $\mathbf{s}_n$, $\mathbf{r}_n$, $\mathbf{t}_n$, $\mathbf{b}$, $\mathbf{c}$ et $\mathbf{z}_n$ sont des vecteurs de w bits. Puisque normalement $L \leq w$, on obtient le vecteur $\mathbf{y}_n$ en tronquant le vecteur $\mathbf{z}_n$ pour ne garder que les L premiers bits.

Dans [67, 69], on donne des recommandations sur les choix de u, s_1, s_2 et l. On décrit également un algorithme qui permet de choisir les vecteurs $\mathbf{b}$ et $\mathbf{c}$ de façon à obtenir une bonne équidistribution pour l'ensemble de points $\Omega_{k,t}$ générés par les valeurs successives. Cet algorithme est implanté dans REGPOLY [84] et est utilisé plus loin dans cette thèse pour trouver de bons générateurs.

2.4.2 Permutation de bits

Un autre type de tempering est la permutation de bits [45, 82]. Ce tempering est défini par les opérations suivantes :

$$\overline{\mathbf{x}}_n = \text{trunc}_w(\mathbf{x}_n) \tag{2.30}$$

$$z_n^{(i)} = \overline{x}_n^{(\pi(i))}, \quad 0 \leq i < w \tag{2.31}$$

$$\mathbf{y}_n = \text{trunc}_L(\mathbf{z}_n) \tag{2.32}$$

où w est un entier, $L \leq w$ et $\pi : \mathbb{Z}_w \to \mathbb{Z}_w$ est une permutation.

Une permutation qui peut être implantée efficacement pour certains types de

générateurs (voir [45, 82]) est

$$\pi(i) = pi + q \mod w \text{ pour } 0 \leq i < w.$$

Les deux paramètres p et q sont des entiers. Si $p = 1$, alors la permutation est en fait une rotation de bits.

Soit $\mathbf{G}(z) = (G_0(z), \ldots, G_{w-1}(z))$, le vecteur des fonctions génératrices des w premiers bits de $\mathbf{x}_n$ et $\mathbf{H}(z) = (H_0(z), \ldots, H_{w-1}(z))$, le vecteur des fonctions génératrices des w bits de $\mathbf{z}_n$. On remarque les vecteurs $\mathbf{G}(z)$ et $\mathbf{H}(z)$ ont les même composantes, mais permutées. Cette permutation des fonctions génératrices peut modifier assez significativement l'équidistribution d'un générateur donné.

2.5 Détermination du polynôme caractéristique de X

Le polynôme caractéristique de la récurrence $\mathbf{x}_n = X\mathbf{x}_{n-1}$ peut être déterminé en calculant le polynôme caractéristique $P_X(z) \in \mathbb{F}_2[z]$ sur $\mathbb{F}_2$ de la matrice X par $P_X(z) = \det(X - Iz)$. Si X est de rang k, alors $P_X(z)$ est de degré k. Le calcul de ce déterminant peut s'avérer fastidieux quand k est grand. La méthode utilisée dans REGPOLY n'est pas basée sur le calcul d'un déterminant. Elle est plutôt basée sur l'algorithme de Berlekamp-Massey [62]. Cet algorithme permet de calculer la plus courte relation linéaire du type (2.1) qui génère une séquence d'éléments dans un corps fini. Soit $Q(z)$, le polynôme caractéristique de cette plus courte relation linéaire. Celui-ci s'appelle le *polynôme minimal* de la séquence.

Pour une valeur de i, $0 \leq i < k$, considérons la séquence $\{x_n^{(i)}\}_{n\geq 0}$ produite par la récurrence $\mathbf{x}_n = X\mathbf{x}_{n-1}$ où $\mathbf{x}_0 \neq 0$. Soit $Q_i(z) \in \mathbb{F}_2[z]$, le polynôme minimal de cette séquence tel que calculé par l'algorithme de Berlekamp-Massey. On sait que la

séquence $\{x_n^{(i)}\}_{n\geq 0}$ suit aussi une récurrence linéaire dont le polynôme caractéristique est $P_X(z)$. Ceci implique que $Q_i(z)$ divise $P_X(z)$. Si le polynôme $P_X(z)$ est irréductible, alors l'algorithme Berlekamp-Massey donne $Q_i(z) = P_X(z)$ pour $i = 0, \ldots, k-1$. Ainsi, pour déterminer si le polynôme caractéristique de la matrice X est primitif (ou irréductible) ou non, on utilise l'algorithme 2.1 qui suit. Celui-ci détermine si le polynôme minimal de la séquence $\{x_n^{(0)}\}_{n\geq 0}$ est primitif (ou irréductible) ou non. Pour cet algorithme, on suppose que la matrice X est de plein rang.

Algorithme 2.1. Détermination de la primitivité (ou de l'irréductible) du polynôme caractéristique de X.

1. Fixer une initialisation non nulle $\mathbf{x}_0$.
2. Calculer $Q_0(z)$, le polynôme minimal de la séquence $\{x_n^{(0)}\}_{n\geq 0}$.
3. Si $\deg(Q_0(z)) = k$, alors faire un test de primitivité (irréductibilité) complet sur $Q_0(z)$ et retourner le résultat.
4. Sinon retourner que le polynôme minimal n'est pas primitif (irréductible).

Pour un générateur de nombres aléatoires, on désire une matrice X dont le polynôme caractéristique est primitif (ou seulement irréductible dans certains cas). Pour trouver une telle matrice, il est nécessaire de vérifier que le polynôme minimal de la séquence $\{x_n^{(0)}\}_{n\geq 0}$ est de degré k. Si ce n'était pas le cas, ceci signifierait que le polynôme caractéristique n'est pas irréductible. Dans l'algorithme, on fait référence à un test de primitivité (irréductibilité) complet. Ce type de test est le sujet de la prochaine section.

2.6 Détermination de la primitivité d'un polynôme

Pour déterminer si un générateur a une période de $2^k - 1$, il faut démontrer que le polynôme caractéristique

$$P_X(z) = z^k - \sum_{i=1}^{k} a_i z^{k-i} \in \mathbb{F}_2[z]$$

de X est primitif sur $\mathbb{F}_2$ où $a_i \in \mathbb{F}_2$, $1 \leq i \leq k$. L'algorithme classique pour déterminer la primitivité sur $\mathbb{F}_2$ d'un polynôme dans $\mathbb{F}_2[z]$ est l'algorithme 2.2.

Algorithme 2.2. ([24]) Détermination de la primitivité d'un polynôme $Q(z) \in \mathbb{F}_2[z]$

1. Factoriser $r = 2^k - 1$ en nombres premiers distincts : $p_1^{e_1} \cdots p_b^{e_b}$.
2. Vérifier si $z^r \equiv 1 \bmod Q(z)$. Si ce n'est pas le cas, alors $Q(z)$ n'est pas primitif.
3. Soit $r_i = r/p_i$, $1 \leq i \leq b$. Pour chaque r_i, vérifier si $z^{r_i} \not\equiv 1 \bmod Q(z)$. Si ce n'est pas le cas pour au moins un seul r_i, alors $Q(z)$ n'est pas primitif.
4. Si $Q(z)$ passe toutes ces vérifications avec succès, alors il est primitif sur $\mathbb{F}_2$.

Quand k est grand, les étapes 2 et 3 de cet algorithme sont très coûteuses. Dans REGPOLY, nous utilisons certaines propriétés des polynômes irréductibles afin d'éviter d'effectuer les étapes 2 et 3 inutilement. La première propriété utilisée est que si $P_X(z)$ est irréductible, alors $a_k = 1$. Si ce n'était pas le cas, alors $P_X(z)$ serait divisible par z. La deuxième propriété est que le nombre de coefficients non nuls de $P_X(z)$ est impair. Si ce n'était pas le cas, alors $P_X(1) = 0$ ce qui implique que $P_X(z)$ serait divisible par $(z + 1)$. Ces deux propriétés fort simples permettent d'éliminer, sans grand effort, plusieurs polynômes qui ne sont pas irréductibles.

Le théorème suivant caractérise les polynômes irréductibles d'un degré donné.

Théorème 2.4. *[56, 5] Soit $\Phi_{d,1}(z), \Phi_{d,2}(z), \ldots$, les polynômes irréductibles de degré d dans $\mathbb{F}_2[z]$. Alors, pour $n \geq 1$,*

$$\prod_{d|n} \prod_{j} \Phi_{d,j}(z) = z^{2^n} + z.$$

Tester l'irréductibilité d'un polynôme est similaire au test de primalité d'un entier : on doit déterminer si le polynôme a au moins un diviseur. Par le théorème 2.4, on peut vérifier si un polynôme $Q(z)$ est divisible par un autre polynôme de degré $d|n$ en calculant $\text{pgcd}(z^{2^n} + z, Q(z))$. Soit $T_s(n)$, le temps nécessaire pour calculer $\text{pgcd}(z^{2^n} + z, Q(z))$ et $T_f(k)$, le temps nécessaire pour faire un test d'irréductibilité complet. L'algorithme 2.3, qui est introduit dans [5], permet de déterminer assez facilement si $Q(z)$ a des facteurs.

Algorithme 2.3. Détermination de l'irréductibilité de $Q(z) \in \mathbb{F}_2[z]$.
Calculer $\text{pgcd}(z^{2^n} + z, Q(z))$ pour $n = 1, 2, \ldots$ jusqu'à ce qu'une des situations suivantes soit rencontrée :

1. On trouve un pgcd non trivial. Dans ce cas, $Q(z)$ n'est pas irréductible.
2. $nT_s(n) \geq T_f(k)$. Dans ce cas, on applique un test d'irréductibilité complet.
3. $n > k/2$. Dans ce cas, le polynôme est irréductible.

L'algorithme suppose que $Q(z)$ est un trinôme dans le critère défini à l'étape 2, mais fonctionne quand même très bien dans le cas où $Q(z)$ a plusieurs coefficients non nuls (du moins, il a permis de déterminer l'irréductibilité de polynômes de très grand degré dans nos applications).

L'algorithme calcule $\text{pgcd}(z^{2^n} + z, Q(z))$ pour plusieurs valeurs de n jusqu'à ce qu'il trouve un pgcd non trivial. Dans [5], on a constaté empiriquement que ce test élimine un polynôme $Q(z)$ sur n, si $Q(z)$ est un trinôme.

Ainsi, pour un trinôme pris au hasard, pour l'algorithme 2.3, on définit le *coût* d'un test qui détecte la réductibilité d'un polynôme $Q(z)$ par le temps nécessaire pour ef-

fectuer le test multiplié par l'inverse de la probabilité d'éliminer un polynôme qui n'est pas irréductible. Ainsi, le coût de l'étape n est $nT_s(n)$ et le coût du test d'irréductibilité complet est $T_f(k)$. L'algorithme s'arrête lorsque le coût du test d'irréductibilité complet est moins grand que celui du test du pgcd (voir [5]).

Si $n > k/2$, alors $Q(z)$ est nécessairement irréductible puisque s'il ne l'était pas, alors il faudrait qu'il soit divisible par un polynôme $g(z)$ de degré $r > k/2$. Dans ce cas, il serait aussi divisible par $h(z) = Q(z)/g(z)$. On remarque que $h(z)$ est de degré inférieur à $k/2$, ce qui amène une contradiction, puisque $n > k/2$.

Le test d'irréductibilité complet utilisé dans REGPOLY est celui de la fonction `ZENPolyIsNotIrreducible()` de la bibliothèque ZENFact qui est basée sur la bibliothèque ZEN [8]. Cette fonction implante l'algorithme de Berlekamp [56].

Une fois qu'un polynôme est prouvé irréductible, alors il faut tester sa primitivité. Dans le cas où $2^k - 1$ est premier, l'irréductibilité implique la primitivité [5]. Par contre, si $2^k - 1$ n'est pas premier, pour tous les diviseurs premiers p_i de $2^k - 1$, il faut vérifier si $z^{r/p_i} \not\equiv 1 \bmod Q(z)$. Dans REGPOLY, on utilise un algorithme présenté dans [92]. Celui-ci tente de réduire le temps nécessaire pour faire une exponentiation en recyclant des résultats déjà obtenus. Cet algorithme remplace l'algorithme 2.2 et est le suivant.

Algorithme 2.4. ([92]) Algorithme 2.2 amélioré

1. Factoriser $r = 2^k - 1$ en nombres premiers distincts : $p_1^{e_1} \cdots p_b^{e_b}$.
2. $q \leftarrow \prod_{i=1}^{b} p_i^{e_i - 1}$ et $q_b(z) \leftarrow z^q \bmod Q(z)$.
3. Pour $i = b-1, \ldots, 0$, $q_i(z) \leftarrow q_{i+1}(z)^{p_{i+1}} \bmod Q(z)$.
4. Pour $i = b, \ldots, 2$, calculer

$$t_i(z) \leftarrow q_i(z)^{p_{i-1} \times \cdots \times p_1} \bmod Q(z).$$

5. $t_1(z) \leftarrow q_1(z)$ et $t_0(z) \leftarrow q_0(z)$.

6. Si $t_0(z) = 1$ et $t_i(z) \neq 1$ pour $i = 1, \ldots, b$, alors $Q(z)$ est primitif. Sinon, il ne l'est pas.

L'algorithme 2.4 est essentiellement le même que l'algorithme 2.2. Par contre l'algorithme 2.4 spécifie la manière de faire les exponentiations. À la fin de l'algorithme, on a $t_i(z) = z^{r_i}$, donc les deux algorithmes calculent les mêmes valeurs.

Un problème sérieux qui limite souvent la détermination de la primitivité d'un polynôme est qu'on ne connaît pas toujours la factorisation de $2^k - 1$. Quand on sait que k est un exposant de Mersenne, alors le temps de calcul est beaucoup moindre puisqu'un simple test d'irréductibilité suffit. Si ce n'est pas le cas, alors il faut soit calculer la factorisation, soit consulter des tables où cette factorisation est donnée. Quand REGPOLY détermine la primitivité d'un polynôme de degré k, il consulte une table pour la factorisation de $2^k - 1$. Celle utilisée par REGPOLY provient du site internet du projet Cunningham, `http://www.cerias.purdue.edu/homes/ssw/cun/index.html`, maintenu par Sam Wagstaff.

2.7 Test statistiques

Nous avons testé la plupart des générateurs proposés dans cette thèse à l'aide de tests statistiques contenus dans les batteries de tests Smallcrush, Crush et Bigcrush de la bibliothèque TestU01 [48]. Chacun des tests fonctionne sur le même principe. Soit $u_0, u_1, \ldots$, une suite de nombre contenus dans $[0, 1) \subset \mathbb{R}$ produite par un générateur. On essaie de trouver des arguments qui permettent de rejeter l'hypothèse nulle

$\mathcal{H}_0 : u_0, u_1, \ldots$ sont des variables aléatoires indépendantes et uniformes sur $[0, 1)$.

Pour ce faire, on conçoit une expérience statistique qui produit une variable aléatoire continue Y (pour le cas où l'expérience produit une variable aléatoire Y discrète, voir le guide de TestU01), à partir de variables aléatoires uniformes, pour laquelle on

connaît la distribution en supposant l'hypothèse nulle. Avec le générateur de nombres aléatoires à tester, on exécute l'expérience qui donne un résultat y. Avec celui-ci, on peut calculer la *p-valeur* qui est

$$p = \Pr[Y \geq y | \mathcal{H}_0].$$

La distribution de p devrait être uniforme sur $[0, 1)$. Si la p-valeur est trop petite (par exemple, $< 10^{-5}$) ou trop grande (par exemple, $> (1 - 10^{-5})$), alors on rejette l'hypothèse nulle puisque le générateur a produit une valeur y qui est très peu probable. Lorsque nous reportons les résultats de tests statistiques qui ont été échoués par certains générateurs, nous donnons la p-valeur afin de donner une idée du degré de l'échec.

Pour les générateurs qui passent tous les tests statistiques, on n'a aucun argument qui permettent de rejeter l'hypothèse nulle. Ceci ne prouve pas que les nombres produits sont des variables aléatoires indépendantes et uniformes sur $[0, 1)$, mais nous réconforte dans notre choix de les utiliser tout comme s'ils en étaient.

Pour les générateurs qui échouent des tests statistiques, nous donnons les paramètres du test et nous faisons référence au guide de TestU01 pour l'explication des tests. Ceci permettra de présenter les résultats aux tests le plus clairement et succinctement possible.

Remarque. La batterie Crush contient plusieurs tests statistiques dont celui du rang de la matrice (`smarsa_MatrixRank`). Dans ce test, on rempli une matrice de bits $g \times h$ avec les bits produits par le générateur et on calcule le rang de la matrice. Pour les générateurs qui entrent dans le cadre défini par les équations (1.1)-(1.4), la matrice,

qui est remplie ligne par ligne, est

$$M = \begin{pmatrix} \tilde{\mathbf{y}}_0 & \tilde{\mathbf{y}}_1 & \cdots & \tilde{\mathbf{y}}_{c-1} \\ \tilde{\mathbf{y}}_c & \tilde{\mathbf{y}}_{c+1} & \cdots & \tilde{\mathbf{y}}_{2c-1} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{\mathbf{y}}_{(g-1)c} & \tilde{\mathbf{y}}_{(g-1)c+1} & \cdots & \tilde{\mathbf{y}}_{gc-1} \end{pmatrix}$$

où $\tilde{\mathbf{y}}_i$ est $\text{trunc}_s(\mathbf{y}_i \ll r)$, s et r sont des paramètres du test et $c = \lceil h/s \rceil$. Pour les vecteurs $\tilde{\mathbf{y}}_{ic-1}$, on laisse tomber quelques uns des bits les moins significatifs pour que la matrice ait h colonnes de bits. Si $g > k$, on remarque que les $k - g$ dernières lignes ne sont pas linéairement indépendantes des k premières lignes. En effet, on a toujours

$$\mathbf{y}_n = \sum_{j=1}^{k} a_j \mathbf{y}_{n-j}$$

pour $n > k$ et pour certains $a_i \in \mathbb{F}_2$, $1 \leq i \leq k$. Cette récurrence peut être facilement déduite du cadre général défini par (1.1)-(1.4). Ainsi, si $g > k$, la matrice M n'a jamais toutes ses lignes linéairement indépendantes.

En supposant que la matrice M est produite de façon vraiment aléatoire où chaque bit est uniformément distribué sur $\{0, 1\}$ et indépendant des autres, alors le rang de la matrice devrait suivre une certaine loi de probabilité connue (voir le guide de TestU01). Pour les générateur utilisés dans cette thèse, si on applique le test avec $g > k$, alors, le test échouera toujours parce qu'il n'est pas normal de toujours obtenir des matrices tels que les $g - k$ dernières lignes ne sont pas linéairement indépendantes des g premières lignes. Par contre, quand $k < g$, il n'y a aucune excuse pour que le test soit échoué (à moins que $h > sk$, auquel cas, les $h - sk$ dernières colonnes sont des combinaisons linéaires de k des sk premières colonnes).

Pour un générateur défini par les équations (1.1)-(1.4), à cause de sa structure, nous ne mentionnerons que les échecs aux tests auxquels il est sensé réussir. Par exemple, la batterie de tests Crush contient un test `smarsa_MatrixRank` qui construit des matrices 300×300. Si le générateur testé est tel que $k = 128$ et que la batterie

Crush détecte une p-valeur suspecte, nous n'y ferons pas mention, puisque ce test ne peut pas être réussi. La batterie Crush contient aussi le test `scomp_LinearComp`, qui tente de détecter des relations linéaires entre les bits. Puisque chacun des bits produits par les générateurs dont il est question dans cette thèse sont produits à partir d'une composition linéaire de bits produits précédemment, on ne s'attend pas non plus que ces générateurs passent ce test.

Par contre, aucun des autres tests contenus dans Smallcrush, Crush et Bigcrush ne devrait poser de problèmes aux bons générateurs basé sur les équations (1.1)-(1.4).

Chapitre 3

Théorie des réseaux polynômiaux

Dans ce chapitre, on définit les réseaux polynomiaux et on explique comment ceux-ci sont reliés aux récurrences linéaires dans un corps fini. Les résultats rapportés dans ce chapitre, sauf pour la section 3.4, sont connus car ils sont tous issus des propriétés des réseaux [96], plus particulièrement des réseaux polynômiaux [40]. Par contre, la manière de présenter ceux-ci est originale, puisqu'elle prend en considération tous les types de récurrences linéaires dans un corps fini qui utilisent la représentation matricielle. Cette présentation permet d'arriver au résultat de la section 3.4 qui démontre comment un réseau polynômial basé sur $\mathbb{F}_q$ peut en engendrer un autre sur $\mathbb{F}_p$ si $q = p^w$ pour un entier $w > 1$ et p est un nombre premier. Ce résultat est important puisqu'il nous permettra de mieux comprendre la structure des générateurs introduits au chapitre 5 et de démontrer certaines de leurs propriétés. Tout au long de ce chapitre, les propriétés exposées sont illustrées à l'aide d'un générateur « jouet » qu'on définit à la prochaine section.

Soit $\mathbb{L}_q$, le corps des séries formelles de Laurent sur le corps à q éléments $\mathbb{F}_q$.

Définissons un *réseau polynômial* $\mathcal{L}_t$ sur $\mathbb{L}_q$ comme un ensemble de la forme

$$\mathcal{L}_t = \left\{ \mathbf{v}(z) = \sum_{j=0}^{t-1} q_j(z)\mathbf{v}_j(z) \text{ tel que } q_j(z) \in \mathbb{F}_q[z] \text{ pour chaque } j \right\}$$

où $\mathbf{v}_0(z), \ldots, \mathbf{v}_{t-1}(z)$ sont des vecteurs arbitraires dans $\mathbb{L}_q^t$, indépendants sur $\mathbb{L}_q$. L'ensemble $\{\mathbf{v}_0(z), \ldots, \mathbf{v}_{t-1}(z)\}$ est une *base du réseau polynômial* $\mathcal{L}_t$ et n'est pas nécessairement unique. Si les vecteurs de la base sont tels que l'ensemble $(\mathbb{F}_q[z])^t$ de tous les vecteurs en t dimensions de polynômes dans $\mathbb{F}_q[z]$ est inclus dans $\mathcal{L}_t$, alors on appelle ce réseau un *réseau polynômial d'intégration.* Les réseaux polynômiaux étudiés dans cette thèse sont tous des réseaux polynômiaux d'intégration.

Soit $v(z) = v_d z^d + v_{d-1} z^{d-1} + \cdots \in \mathbb{L}_q$ où $v_d \neq 0$ quand $v(z) \neq 0$ et $\mathbf{v}(z) = (v_0(z), \ldots, v_{t-1}(z)) \in \mathbb{L}_q^t$. Soit la fonction $\deg : \mathbb{L}_q \to \mathbb{Z}$ définie par

$$\deg(v(z)) = \begin{cases} d & \text{si } v(z) \neq 0 \\ -\infty & \text{sinon} \end{cases}$$

On définit les normes

$$|v(z)| = \begin{cases} 0 & \text{si } v(z) = 0 \\ q^{\deg(v(z))} & \text{sinon} \end{cases}$$

et

$$\|\mathbf{v}(z)\| = \max_{0 \leq i < t} |v_i(z)|.$$

Soit $\mathbb{L}_{q,0}$, l'ensemble des séries de Laurent $\mathbf{v}(z) \in \mathbb{L}_q$ telles que $\log_q \|\mathbf{v}(z)\| < 0$ (on note $\mathbb{L}_{q,0}$ par $\mathbb{L}_0$ lorsque le contexte ne laisse aucune ambiguïté). On définit l'ensemble

$$\bar{\mathcal{L}}_t = \mathcal{L}_t \cap \mathbb{L}_{q,0}^t = \left\{ \mathbf{v}(z) \in \mathcal{L}_t : \log_q \|\mathbf{v}(z)\| < 0 \right\}. \tag{3.1}$$

Cette définition sera utilisée un peu plus loin dans ce chapitre.

Soit

$$V = \begin{pmatrix} \mathbf{v}_0(z) \\ \vdots \\ \mathbf{v}_{t-1}(z) \end{pmatrix}$$

où $\{\mathbf{v}_0(z), \ldots, \mathbf{v}_{t-1}(z)\}$ est une base de $\mathcal{L}_t$ et soit W, la matrice

$$\left(\mathbf{w}_0(z)^\mathsf{T} : \ldots : \mathbf{w}_{t-1}(z)^\mathsf{T} \right)$$

telle que $VW = I$, la matrice identité. On définit le *dual du réseau polynômial* $\mathcal{L}_t$ par le réseau $\mathcal{L}_t^*$ généré par la base $\{\mathbf{w}_0(z), \ldots, \mathbf{w}_{t-1}(z)\}$.

Une base qui est utile pour déterminer l'équidistribution d'un générateur (on explique comment au chapitre 4) est une base réduite au sens de Minkowski. Pour définir celle-ci, le théorème suivant nous est utile.

Théorème 3.1. *(Mahler [57])*
Soit $\mathbf{h}_0(z), \ldots, \mathbf{h}_{t-1}(z)$, *des points dans le réseau polynômial* $\mathcal{L}_t \subset \mathbb{L}_q^t$ *avec les propriétés suivantes :*

1. $\mathbf{h}_0(z)$ *est un plus court vecteur dans* $\mathcal{L}_t$ *;*
2. *pour* $i = 1, \ldots, t-1$, $\mathbf{h}_i(z)$ *est un plus court vecteur parmi l'ensemble des vecteurs* $\mathbf{h}(z)$ *dans* $\mathcal{L}_t$ *tel que* $\mathbf{h}_0(z), \ldots, \mathbf{h}_{i-1}(z), \mathbf{h}(z)$ *sont linéairement indépendants sur* $\mathbb{L}_q$.

Les vecteurs $\mathbf{h}_0(z), \ldots, \mathbf{h}_{t-1}(z)$ *forment une base de* $\mathcal{L}_t$.

Un système $\mathbf{h}_0(z), \ldots, \mathbf{h}_{t-1}(z)$ tel que décrit dans le théorème 3.1 est une *base réduite* du réseau $\mathcal{L}_t$ au sens de Minkowski. Ce type de base n'est pas nécessairement unique. À partir de cette base, on définit les valeurs $\sigma_i = \|\mathbf{h}_i(z)\|$ qui sont uniquement déterminées par $\mathcal{L}_t$ et qui sont appelées *minima successifs*. Un algorithme de Lenstra [54] permet de calculer efficacement un ensemble de vecteurs $\mathbf{h}_i(z)$.

Ce type de réseau dans l'espace des séries formelles de Laurent est utile pour déterminer l'équidistribution. Il existe deux catégories de réseaux que décrivent les générateurs à récurrence linéaire modulo 2. Le premier réseau, dit réseau polynômial en dimension, n'est utile que pour certains types de générateurs, tels les générateurs de type Tausworthe. La raison étant que pour la plupart des générateurs qui entrent

dans le cadre des équations (1.1)-(1.4) le réseau en dimension $\mathcal{L}_t$ engendré par ces points est tel que $\bar{\mathcal{L}}_t$ contient beaucoup plus d'éléments que l'ensemble de points $\Omega_{k,t}$ produit par le générateur, ce qui le rend inutilisable pour nos fins. Le deuxième type de réseau, dit réseau polynômial en résolution, est produit par tous les générateurs à récurrence linéaire dans un corps fini et l'ensemble $\bar{\mathcal{L}}_t$ contient le même nombre de points que l'ensemble de points $\Omega_{k,t}$. Le réseau en résolution constitue le sujet de ce chapitre.

3.1 Exemple

Avant d'entrer dans le coeur du sujet de ce chapitre, examinons un générateur particulier qui est basé sur une récurrence linéaire dans $\mathbb{F}_{2^2}$. Cet exemple permettra de mieux comprendre ce chapitre et ses implications.

Soit

$$T = \begin{pmatrix} 0 & 0 & \zeta \\ 1 & 0 & \zeta \\ 0 & 1 & 1+\zeta \end{pmatrix},$$

une matrice 3×3, avec ses éléments dans $\mathbb{F}_4$, dont le polynôme minimal est $P(z) = z^3 + (1+\zeta)z^2 + \zeta z + \zeta$, $\zeta \in \mathbb{F}_4$ et le polynôme minimal de ζ est $R(x) = x^2 + x + 1$. Soit la récurrence

$$\mathbf{t}_n = T\mathbf{t}_{n-1}$$

où

$$\mathbf{t}_n = (t_n^{(0)}, t_n^{(1)}, t_n^{(2)}) \in \mathbb{F}_4^3,$$

$$t_n^{(j)} = t_n^{(j,0)} + t_n^{(j,1)}\zeta$$

et $t_n^{(j,i)} \in \mathbb{F}_2$ pour $n \geq 0$, $j = 0, 1, 2$ et $i = 0, 1$. Ainsi, on peut représenter l'état de la

récurrence par un vecteur de bits

$$\begin{aligned}\bar{\mathbf{t}}_n &= (t_n^{(0,0)}, t_n^{(0,1)}, t_n^{(1,0)}, t_n^{(1,1)}, t_n^{(2,0)} t_n^{(2,1)}) \\ &= (\bar{t}_n^{(0)}, \bar{t}_n^{(1)}, \bar{t}_n^{(2)}, \bar{t}_n^{(3)}, \bar{t}_n^{(4)}, \bar{t}_n^{(5)}) \in \mathbb{F}_2^6\end{aligned}$$

Le polynôme $P(z)$ étant primitif sur $\mathbb{F}_{64}$, ceci indique que la récurrence définie par T est de période $(2^2)^3 - 1 = 63$.

Pour illustrer la récurrence, nous donnons les valeurs que prennent $\mathbf{t}_1$, $\mathbf{t}_2$ et $\mathbf{t}_3$ à partir de l'initialisation arbitraire $\mathbf{t}_0 = (0, 1, 0) \in \mathbb{F}_4^3$:

$$\begin{aligned}\mathbf{t}_1 &= (\ 0\ ,\ \ 0\ \ ,\ \ 1\ \ ) \\ \mathbf{t}_2 &= (\ \zeta\ ,\ \ \zeta\ \ ,\ 1+\zeta\ ) \\ \mathbf{t}_3 &= (\ 1\ ,\ 1+\zeta\ ,\ \ 0\ \ ) \\ &\ldots.\end{aligned}$$

Dans ce cas, les vecteurs $\bar{\mathbf{t}}_0$, $\bar{\mathbf{t}}_1$, $\bar{\mathbf{t}}_2$ et $\bar{\mathbf{t}}_3$ sont

$$\begin{aligned}\bar{\mathbf{t}}_0 &= (0,0,1,0,0,0) \\ \bar{\mathbf{t}}_1 &= (0,0,0,0,1,0) \\ \bar{\mathbf{t}}_2 &= (0,1,0,1,1,1) \\ \bar{\mathbf{t}}_3 &= (1,0,1,1,0,0) \\ &\ldots.\end{aligned}$$

On peut définir un très petit générateur de nombres aléatoires avec la sortie

$$u_n = \bar{t}_n^{(0)} 2^{-1} + \bar{t}_n^{(1)} 2^{-2} + \bar{t}_n^{(2)} 2^{-3} + \bar{t}_n^{(3)} 2^{-4} + \bar{t}_n^{(4)} 2^{-5} + \bar{t}_n^{(5)} 2^{-6}$$

pour $n \geq 0$. Bien sûr, la période de ce générateur est beaucoup trop petite pour que celui-ci puisse être utilisé dans des applications réelles, mais il sera utile tout au long de ce chapitre pour illustrer les concepts présentés. Ce type de générateur est décrit plus en détails au chapitre 5.

Avec l'initialisation $\mathbf{t}_0 = (0, 1, 0)$, on obtient les sorties

$$\begin{aligned} u_0 &= 2^{-3} &&= 0.125 \\ u_1 &= 2^{-5} &&= 0.03125 \\ u_2 &= 2^{-2} + 2^{-4} + 2^{-5} + 2^{-6} &&= 0.359375 \\ u_3 &= 2^{-1} + 2^{-3} + 2^{-4} &&= 0.6875 \\ &\cdots \end{aligned}$$

Soit $\{t_n^{(i)}\}_{n\geq 0}$, la séquence d'éléments dans $\mathbb{F}_4$ produit par la $(i+1)$-ième coordonnée de la séquence $\{\mathbf{t}_n\}_{n\geq 0}$ à partir de l'initialisation $\mathbf{t}_0$. On verra dans ce chapitre que cette séquence peut être produite par une fonction génératrice $G_i(z) = g_i(z)/P(z)$ et que le polynôme $g_i(z) \in \mathbb{F}_4[z]/P(z)$ dépend de l'initialisation $\mathbf{t}_0$. Pour une initialisation $\mathbf{t}_0$ donnée, on peut définir un vecteur de fonctions génératrices

$$\mathbf{G}_3(T, \mathbf{t}_0) = (G_0(z), G_1(z), G_2(z)) = (g_0(z), g_1(z), g_2(z))/P(z)$$

où la $(i+1)$-ième coordonnée contient la fonction génératrice de la séquence $\{t_n^{(i)}\}_{n\geq 0}$. Nous verrons dans ce chapitre que l'ensemble

$$\tilde{\mathcal{L}}_3(T(\mathbb{F}_4)) = \{\mathbf{G}_3(T, \mathbf{t}_0) : \mathbf{t}_0 \in \mathbb{F}_4^3\}$$

est un sous-ensemble d'un réseau polynômial d'intégration $\mathcal{L}_3(T(\mathbb{F}_4))$, que l'on nomme réseau polynômial en $(\mathbb{F}_4, 3)$-résolution, produit par la matrice T.

De plus, soit $Q(z)$, le polynôme minimal de la matrice T sur $\mathbb{F}_2$ et $\{\bar{t}_n^{(i)}\}_{n\geq 0}$, la séquence d'éléments dans $\mathbb{F}_2$ produite par la $(i+1)$-ième coordonnée de la séquence $\{\bar{\mathbf{t}}_n\}_{n\geq 0}$ à partir de l'initialisation $\bar{\mathbf{t}}_0$. Soit $H_i(z) = h_i(z)/Q(z)$, la fonction génératrice de cette séquence. Soit le vecteur

$$\begin{aligned} \mathbf{H}_6(T, \bar{\mathbf{t}}_0) &= (H_0(z), H_1(z), H_2(z), H_3(z), H_4(z), H_5(z)) \\ &= (h_0(z), h_1(z), h_2(z), h_3(z), h_4(z), h_5(z))/Q(z) \end{aligned}$$

des fonctions génératrices des séquences $\{\bar{t}_n^{(i)}\}_{n\geq 0}$ produites à partir de l'état initial $\bar{\mathbf{t}}_0$. Nous verrons dans ce chapitre que l'ensemble

$$\bar{\mathcal{L}}_6(T(\mathbb{F}_2)) = \{\mathbf{H}_6(T, \bar{\mathbf{t}}_0) : \bar{\mathbf{t}}_0 \in \mathbb{F}_2^6\}$$

est aussi un sous-ensemble d'un réseau polynômial d'intégration $\bar{\mathcal{L}}_6(T(\mathbb{F}_2))$ et qu'il est possible de l'obtenir à partir de $\mathcal{L}_3(T(\mathbb{F}_4))$. Le réseau $\mathcal{L}_6(T(\mathbb{F}_2))$ se veut un « raffinement » du réseau $\mathcal{L}_3(T(\mathbb{F}_4))$ et est appelé réseau polynômial en $(\mathbb{F}_2, 6)$-résolution. Ceci sera vu à la section 3.4 et constitue la contribution originale de ce chapitre. Grâce à cette connexion entre les deux réseaux, on réussit, au chapitre 5, à démontrer des bornes sur l'équidistribution de certains générateurs.

Afin d'illustrer l'utilité de ces réseaux, considérons la figure 3.1 où on montre tous les points $(u_0, u_1) \in [0,1)^2$ construits à partir de toutes les initialisations possibles $\mathbf{t_0} \in \mathbb{F}_4^3$. L'ensemble de tous ces points peut être représenté de manière compacte par

$$\Omega_{6,2} = \{(u_0, u_1) : \mathbf{t}_0 \in \mathbb{F}_4^3\}.$$

Comme illustré à la figure 3.1, si on divise $[0,1)^2$ en 64 carrés disjoints de dimension $2^{-3} \times 2^{-3}$, alors on remarque que la moitié des carrés ont deux points et l'autre moitié n'en ont aucun. Quand on partitionne $[0,1)^2$ en 16 carrés de dimension $2^{-2} \times 2^{-2}$, on voit que chacun des carrés contient exactement quatre points. Ceci correspond à dire que le générateur n'est pas équidistribué pour la résolution $\ell = 3$, mais qu'il l'est pour $\ell = 2$, donc $\ell_2 = 2$.

On peut représenter l'ensemble de points de la figure 3.1 par un réseau en résolution. Dans cette représentation, comme il est expliqué dans [101], on inverse les notions de résolution et de dimension. Par exemple, à une initialisation $\bar{\mathbf{t}}_0 \in \mathbb{F}_2^6$, on peut associer un point en ℓ dimensions qui est donné par les fonctions génératrices $H_0(z), \ldots, H_{\ell-1}(z)$ associées à $\bar{\mathbf{t}}_0$. Ainsi, le point associé à $\mathbf{t}_0$ est $(H_0(2), \ldots, H_{\ell-1}(2))$. Le réseau en $(\mathbb{F}_2, 2)$-résolution associé à $\Omega_{6,2}$ est illustré à la figure 3.2. Nous insistons sur le fait que l'ensemble de points illustré à la figure 3.2 n'est pas $\Omega_{6,2}$, mais un

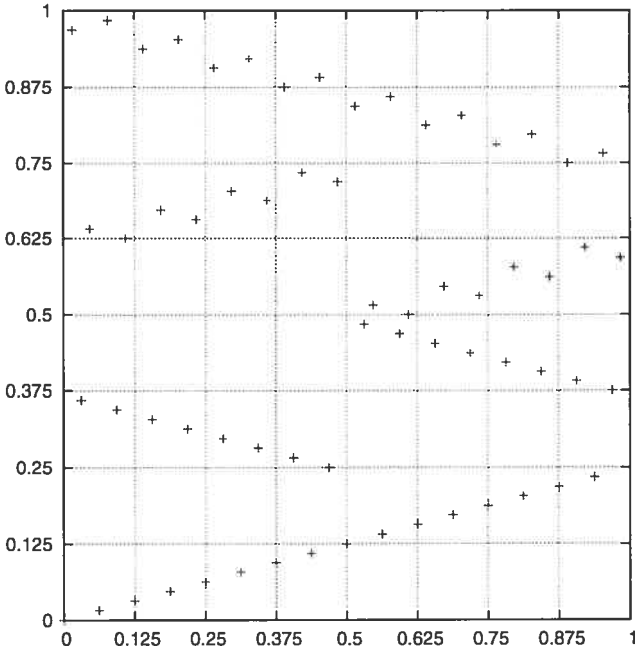


Figure 3.1 – Ensemble de points $\Omega_{6,2}$.

ensembles de points qui nous permet de visualiser le réseau en résolution décrit par $\Omega_{6,2}$. En considérant l'ensemble de points illustré, on peut déterminer la résolution à laquelle on a la $(2, \ell)$-équidistribution. On remarque que si on partitionne $[0,1)^2$ en carrés de dimension $2^{-3} \times 2^{-3}$ (tel qu'illustré), alors il y a un point dans chacun des carrés et l'ensemble de points illustré est tel que $\ell_2 = 3$. En se souvenant que pour le réseau en $(\mathbb{F}_2, 2)$-résolution, on a inversé les notions de dimension et de résolution par rapport à l'ensemble de points original, on obtient que ce dernier est tel que $t_2 = 3$. Il existe toute une théorie qui justifie l'utilisation du réseau en résolution et qui démontre la validité de ce qu'on vient de faire [101].

Quand vient le temps de calculer l'équidistribution de générateurs qui ont un état très grand (par exemple, l'espace des états est $\mathbb{F}_2^{800}$), il est impraticable de vérifier de manière visuelle l'équidistribution comme on vient de le faire. Pour y arriver autrement, on aurait recours au réseau en résolution $\mathcal{L}_\ell(T(\mathbb{F}_2))$. Un théorème dont on

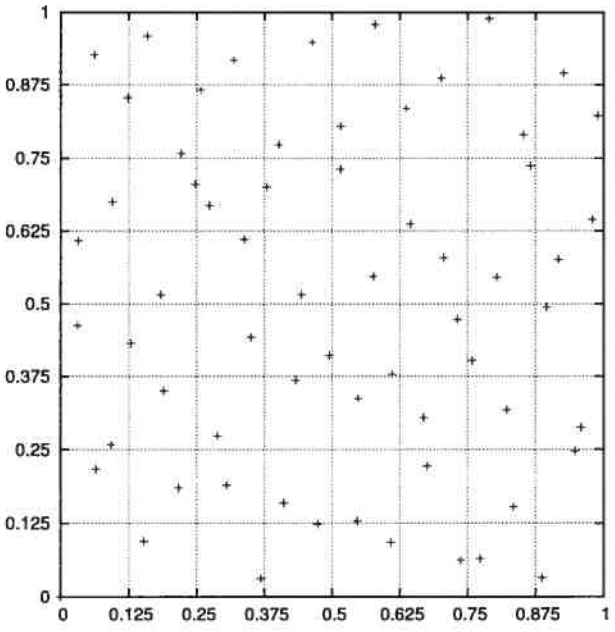


Figure 3.2 – Illustration graphique du réseau en résolution décrit par $\Omega_{6,2}$.

fait mention au chapitre 4 permet de déterminer la valeur de t_ℓ à l'aide du réseau en $(\mathbb{F}_2, \ell)$-résolution et de sa base réduite au sens de Minkowski.

Au chapitre 5, on utilise la notion de réseau en résolution afin de démontrer plusieurs propriétés dont celle concernant les bornes sur l'équidistribution des nouveaux générateurs introduits (pour certains choix de paramètres). Également, on démontre certaines propriétés sur le comportement de séquences de blocs de bits produites par ces nouveaux générateurs à l'aide des réseaux en résolution.

Ce petit exemple donne un avant-goût de ce dont traite ce chapitre. Le but de ce chapitre est de jeter de bonnes bases afin de présenter certains théorèmes connus et d'autres nouveaux résultats qui font appel à la notion de réseau en résolution.

3.2 Réseau en dimension

Dans le contexte des générateurs de Tausworthe, il est possible de caractériser l'ensemble

$$\{(u_0, u_1, \ldots, u_{t-1}) : \mathbf{x}_0 \in \mathbb{F}_2^k\}$$

par un réseau polynômial. À la section 2.3.2, on décrit la récurrence du générateur de Tausworthe par les équations (2.10) et (2.11). Si on considère tous les t-tuplets $(q_i, q_{i+1}, \ldots, q_{i+t-1}) \in \mathbb{L}_2^t, 0 \leq i \leq 2^k - 1$, alors ceux-ci font partie d'un réseau $\mathcal{L}_t$ qui a pour base

$$\begin{aligned}
\mathbf{v}_0(z) &= (1, a(z), a(z)^2 \bmod P(z), \ldots, a(z)^{t-1} \bmod P(z))/P(z), \\
\mathbf{v}_1(z) &= (0, 1, \ldots, 0), \\
&\ldots \\
\mathbf{v}_{t-1}(z) &= (0, 0, \ldots, 1).
\end{aligned}$$

Celui-ci est un réseau polynômial de type Korobov. Pour plus de détails sur ce type de réseau, voir [53] (en particulier la proposition 3.10). Ce réseau n'est pas utile pour les besoins de cette thèse puisque seul un ensemble restreint de générateurs, dont plusieurs sont décrits dans [103], sont tels que l'ensemble de points $\Omega_{k,t}$ produit par le générateur contient le même nombre de points que l'ensemble $\bar{\mathcal{L}}_t$ généré par l'ensemble de points. Dans le cas contraire, $\bar{\mathcal{L}}_t$ ne représente pas nécessairement le comportement de $\Omega_{k,t}$. Par contre, pour tous les générateurs qui entrent dans le cadre des équations (1.1)-(1.4), le réseau en résolution est très utile. Ce type de réseau est le sujet du reste de ce chapitre.

3.3 Réseau polynômial en $(\mathbb{F}_{p^w}, \nu)$-résolution

Dans cette section et celles qui suivent, nous analysons la récurrence

$$\mathbf{t}_n = T\mathbf{t}_{n-1}. \tag{3.2}$$

où T est une matrice non singulière de dimension $r \times r$ dont les éléments sont dans le corps fini $\mathbb{F}_q$ et $\mathbf{t}_n = (t_n^{(0)}, \ldots, t_n^{(r-1)})^\mathsf{T} \in \mathbb{F}_q^r$ où $q = p^w$ et p est un nombre premier. Cette récurrence est utile puisque nous l'utilisons afin de définir deux générateurs au chapitre 5. Dans ce cas précis, on a $p = 2$ et w est un entier relativement petit (plusieurs exemples sont donnés avec $w = 32$). Cette analyse nous permet de définir ce qu'est un réseau polynômial en $(\mathbb{F}_{p^w}, \nu)$-résolution. Cette notion a été introduite par Tezuka [101].

Soit $P(z) = z^r - \sum_{i=1}^{r} \alpha_i z^{r-i}$, où $\alpha_i \in \mathbb{F}_q$ pour $i = 1, \ldots, r$, le polynôme minimal de la récurrence (3.2). L'hypothèse suivante est utile puisqu'elle garantit que chaque élément de $\mathbb{F}_q[z]/P(z)$ a un inverse multiplicatif.

Hypothèse 3.1. *Pour le reste du chapitre, on suppose que $P(z)$ est irréductible sur $\mathbb{F}_q$.*

Il est bien connu que la séquence $S_j = \{t_n^{(j)}\}_{n \geq 0}$, pour $0 \leq j < r$, avec une initialisation appropriée, peut être reproduite par une récurrence linéaire de la forme

$$t_n^{(j)} = \sum_{i=1}^{r} \alpha_i t_{n-i}^{(j)},$$

puisque la récurrence (3.2) est linéaire. Ceci signifie qu'on peut associer à S_j une fonction génératrice $G_j(z)$ [56] pour $j = 0, \ldots, r-1$. Cette fonction est

$$G_j(z) = t_0^{(j)} z^{-1} + t_1^{(j)} z^{-2} + \ldots.$$

On l'appelle ainsi parce que les coefficients successifs de z^{-i} génèrent la séquence. Comme a l'a déjà affirmé au chapitre 2 dans le cas où $q = 2$, il peut être démontré [56]

qu'il existe un polynôme unique $g_j(z) \in \mathbb{F}_q[z]/P(z)$ tel que $G_j(z) = g_j(z)/P(z)$. Soit

$$\mathbf{G}_\nu(T, \mathbf{t}_0) = (G_0(z), \ldots, G_{\nu-1}(z)) = (g_0(z), \ldots, g_{\nu-1}(z))/P(z) \in \mathbb{L}_q^\nu,$$

le vecteur des $\nu \leq r$ premières fonctions génératrices de la récurrence (3.2) avec état initial $\mathbf{t}_0 \in \mathbb{F}_q^r$.

Exemple 3.1. L'exemple de la section 3.1 est repris afin d'illustrer les différents concepts que nous avons vus jusqu'à présent.

En choisissant l'initialisation arbitraire $\mathbf{t}_0 = (0, 1, 0)$, on a obtenu certains vecteurs $\mathbf{t}_1$, $\mathbf{t}_2$ et $\mathbf{t}_3$. On peut voir que, pour cette initialisation, les fonctions génératrices des séquences $S_j = \{t_{n,j}\}_{n\geq 0}$ relatives à T, pour $j = 1, 2, 3$, sont

$$\begin{aligned}
G_0(z) &= 0z^{-1} + 0z^{-2} + \zeta z^{-3} + z^{-4} + \cdots \\
G_1(z) &= z^{-1} + 0z^{-2} + \zeta z^{-3} + (1+\zeta)z^{-4} + \cdots \\
G_2(z) &= 0z^{-1} + z^{-2} + (1+\zeta)z^{-3} + 0z^{-4} + \cdots .
\end{aligned}$$

Les termes en z^{-i} de $G_j(z)$ où $i > 4$ sont déterminés par la récurrence

$$t_n^{(j)} = (1+\zeta)t_{n-1}^{(j)} + \zeta t_{n-2}^{(j)} + \zeta t_{n-3}^{(j)} \tag{3.3}$$

pour $j = 1, 2, 3$. On peut vérifier, en faisant la division par $P(z)$, que

$$\begin{aligned}
\mathbf{G}_3(T, \mathbf{t}_0) &= (G_0(z), G_1(z), G_2(z)) \\
&= (g_0(z), g_1(z), g_2(z))/P(z) \\
&= (\zeta, z^2 + (1+\zeta)z, z)/P(z).
\end{aligned}$$

3.3.1 Propriétés de $\mathbf{G}_\nu(T, \mathbf{t}_0)$

Le vecteur $\mathbf{G}_\nu(T, \mathbf{t}_0)$ est le vecteur des fonctions génératrices de chacune des séquences $S_j = \{t_n^{(j)}\}_{n\geq 0}$ à partir d'une initialisation donnée $\mathbf{t}_0$. Dans cette section, on donne les propriétés de ce vecteur, ce qui nous mènera, à la prochaine section, à la définition d'un réseau polynômial. Ces propriétés sont illustrées à l'exemple 3.2.

Proposition 3.1. *Supposons que $P(z)$ soit irréductible. Soit*

$$\mathbf{G}_\nu(T, \mathbf{t}_0) = (g_0(z), \ldots, g_{\nu-1}(z))/P(z) \in \mathbb{L}_q$$

et $\mathbf{t}_0, \tilde{\mathbf{t}}_0 \in \mathbb{F}_q^r$. On a les propriétés suivantes :

1. $\mathbf{G}_\nu(T, \mathbf{t}_0) + \mathbf{G}_\nu(T, \tilde{\mathbf{t}}_0) = \mathbf{G}_\nu(T, \mathbf{t}_0 + \tilde{\mathbf{t}}_0)$.
2. *Pour un entier $m \geq 0$,*

$$\begin{aligned} \mathbf{G}_\nu(T, T^m\mathbf{t}_0) &= \mathbf{G}_\nu(T, \mathbf{t}_m) \\ &= (z^m g_0(z) \bmod P(z), \ldots, z^m g_{\nu-1}(z) \bmod P(z))/P(z). \end{aligned}$$

3. *Pour tout $\mathbf{t}_0 \neq 0, \tilde{\mathbf{t}}_0 \neq 0 \in \mathbb{F}_q^r$, il existe un $f(z) \in \mathbb{F}_q[z]/P(z)$ tel que* $\mathbf{G}_\nu(T, \tilde{\mathbf{t}}_0) = \mathbf{G}_\nu(T, f(T)\mathbf{t}_0) = (f(z)g_0(z) \bmod P(z), \ldots, f(z)g_{\nu-1}(z) \bmod P(z))/P(z)$.

Démonstration.

1. Soit $\{\mathbf{t}_n\}_{n\geq 0}$ et $\{\tilde{\mathbf{t}}_n\}_{n\geq 0}$, les séquences produites par les initialisations $\mathbf{t}_0$ et $\tilde{\mathbf{t}}_0$, respectivement. Soit

$$g_j(z)/P(z) = \sum_{n\geq 0} t_n^{(j)} z^{-n-1}$$

et

$$\tilde{g}_j(z)/P(z) = \sum_{n\geq 0} \tilde{t}_n^{(j)} z^{-n-1},$$

les fonctions génératrices des séquences $\{t_n^{(j)}\}_{n\geq 0}$ et $\{\tilde{t}_n^{(j)}\}_{n\geq 0}$ respectivement. Puisque la récurrence (3.2) est linéaire, alors l'initialisation $(\mathbf{t}_0 + \tilde{\mathbf{t}}_0)$ produit la séquence $\{\mathbf{t}_n + \tilde{\mathbf{t}}_n\}_{n\geq 0}$. D'où on déduit que la fonction génératrice de $\{t_n^{(j)} + \tilde{t}_n^{(j)}\}_{n\geq 0}$ est $\sum_{n\geq 0}(t_n^{(j)} + \tilde{t}_n^{(j)})z^{-n-1} = (g_j(z) + \tilde{g}_j(z))/P(z)$. Ce qui démontre la propriété 1.

2. Soit $G_j(z) = g_j(z)/P(z) = \sum_{n\geq 0} t_n^{(j)} z^{-n-1}$, la fonction génératrice de la séquence $\{t_n^{(j)}\}_{n\geq 0}$ produite avec l'initialisation $\mathbf{t}_0$. La fonction génératrice de la séquence $\{t_n^{(j)}\}_{n\geq m}$ produite avec l'initialisation $\mathbf{t}_m$ est trivialement

$$\bar{G}_j(z) = \bar{g}_j(z)/P(z) = \sum_{n\geq 0} t_{n+m}^{(j)} z^{-n-1}.$$

Puisque

$$\bar{g}_j(z)/P(z) + \sum_{h=0}^{m-1} t_h^{(j)} z^h = z^m g_j(z)/P(z),$$

on vérifie directement que $\bar{g}_j(z) \equiv z^m g_j(z) \bmod P(z)$. Ceci démontre la propriété 2.

3. Soit $\mathbf{t}_0 \in \mathbb{F}_q^r$. Pour $\mathbf{t}_0 \neq 0$, les vecteurs $\mathbf{t}_0, T\mathbf{t}_0, \ldots, T^{r-1}\mathbf{t}_0$ sont linéairement indépendants sur $\mathbb{F}_q$ puisque s'ils ne l'étaient pas, il existerait des $a_i \in \mathbb{F}_q, 0 \leq i \leq r-1$ tels que

$$\begin{aligned} 0 &= a_0\mathbf{t}_0 + a_1 T\mathbf{t}_0 + \cdots + a_{r-1}T^{r-1}\mathbf{t}_0 \\ &= (a_0 + a_1 T + \cdots + a_{r-1}T^{r-1})\mathbf{t}_0 \\ &= (a_0 + a_1 T + \cdots + a_{r-1}T^{r-1}). \end{aligned}$$

Cette dernière équation est une contradiction avec le fait que $P(z)$ soit le polynôme minimal de T sur $\mathbb{F}_q$ puisque le polynôme $\bar{P}(z) = a_0 + a_1 z + \cdots + a_{r-1}z^{r-1} \in \mathbb{F}_q[z]$ est de degré inférieur à r et est tel que $P(T) = 0$. On peut donc en conclure que les vecteurs $\mathbf{t}_0, T\mathbf{t}_0, \ldots, T^{r-1}\mathbf{t}_0$ sont linéairement indépendants sur $\mathbb{F}_q$ et qu'ils engendrent $\mathbb{F}_q^r$ puisque celui-ci est un espace en r dimensions et que $\mathbb{F}_q$ est un corps. Ceci implique que, étant donné un $\mathbf{t}_0 \neq 0$, un élément $\tilde{\mathbf{t}}_0 \in \mathbb{F}_q^r$, tel que $\tilde{\mathbf{t}}_0 \neq 0$ et $\tilde{\mathbf{t}}_0 \neq \mathbf{t}_0$, peut être exprimé par une combinaison linéaire dans $\mathbb{F}_q$ des vecteurs $\mathbf{t}_0, T\mathbf{t}_0, \ldots, T^{r-1}\mathbf{t}_0$. On obtient

$$\tilde{\mathbf{t}}_0 = b_0\mathbf{t}_0 + b_1 T\mathbf{t}_0 + \cdots + b_{r-1}T^{r-1}\mathbf{t}_0 = (b_0 + b_1 T + \cdots + b_{r-1}T^{r-1})\mathbf{t}_0 = f(T)\mathbf{t}_0.$$

Ceci démontre l'égalité $\mathbf{G}_\nu(T, \tilde{\mathbf{t}}_0) = \mathbf{G}_\nu(T, f(T)\mathbf{t}_0)$.

L'égalité

$$\mathbf{G}_\nu(T, f(T)\mathbf{t}_0) = (f(z)g_1(z) \bmod P(z), \ldots, f(z)g_\nu(z) \bmod P(z))/P(z)$$

est déduite des propriétés 1 et 2.

□

Exemple 3.2. Soit $\mathbf{t}_0 = (0,1,0)$ et $\tilde{\mathbf{t}}_0 = (0,0,1)$, la matrice T étant la même que celle de la section 3.1. On sait que $\mathbf{G}_3(T,\mathbf{t}_0) = (\zeta, z^2+(1+\zeta)z, z)/P(z)$. On remarque que $\tilde{\mathbf{t}}_0 = \mathbf{t}_1$ et que, par la propriété 2 de la proposition 3.1, on a que $\mathbf{G}_3(T,\tilde{\mathbf{t}}_0) = \mathbf{G}_3(T, f(T)\mathbf{t}_0) = (\zeta z, \zeta z+\zeta, z^2)/P(z)$. Le polynôme de la propriété 3 est $f(z) = z$.

Par la propriété 1, on a que $\mathbf{G}_3(T,(0,1,1)) = \mathbf{G}_3(T,\mathbf{t}_0)+\mathbf{G}_3(T,\tilde{\mathbf{t}}_0) = (\zeta+\zeta z, z^2+ z+\zeta, z^2+z)/P(z)$.

Lemme 3.1. *Supposons que $P(z)$ soit irréductible sur $\mathbb{F}_q$. Pour tout $\mathbf{G}_\nu(T,\mathbf{t}_0) = (g_0(z),\ldots,g_{\nu-1}(z))/P(z)$ où $\mathbf{t}_0 \neq 0$, il existe des polynômes $r_j(z), j = 1,\ldots,\nu-1$, qui ne dépendent que de la matrice T et non de l'initialisation $\mathbf{t}_0$, tels que*

$$g_j(z) \equiv r_j(z) g_0(z) \bmod P(z). \tag{3.4}$$

Démonstration. Pour une initialisation $\mathbf{t}_0$ donnée, les polynômes $r_j(z)$ existent puisque $P(z)$ est irréductible. Soit

$$\mathbf{G}_\nu(T,\mathbf{t}_0) = (g_0(z),\ldots,g_{\nu-1}(z))/P(z)$$

et

$$\mathbf{G}_\nu(T,\tilde{\mathbf{t}}_0) = (\tilde{g}_0(z),\ldots,\tilde{g}_{\nu-1}(z))/P(z).$$

Soit, de plus, $f(z) \in \mathbb{F}_q[z]/P(z)$, un polynôme tel que $\mathbf{G}_\nu(T, f(T)\tilde{\mathbf{t}}_0) = \mathbf{G}_\nu(T,\mathbf{t}_0)$. Ce polynôme existe par la propriété 3 de la proposition 3.1 et l'irréductibilité de $P(z)$. On a

$$f(z)\tilde{g}_j(z) \equiv g_j(z) \bmod P(z),\ 0 \leq j < \nu.$$

On en conclut que

$$\begin{aligned} g_j(z) &\equiv r_j(z) g_0(z) \bmod P(z) \\ f(z)\tilde{g}_j(z) &\equiv r_j(z) f(z)\tilde{g}_0(z) \bmod P(z) \\ \tilde{g}_j(z) &\equiv r_j(z)\tilde{g}_0(z) \bmod P(z). \end{aligned}$$

À la troisième équation, l'inverse de $f(z)$ existe puisque le fait que $P(z)$ soit irréductible garantit que l'anneau $\mathbb{F}_q[z]/P(z)$ est un aussi un corps. On voit que les polynômes $r_j(z)$ ont la propriété désirée pour une initialisation arbitraire $\tilde{\mathbf{t}}_0$. Ceci implique que les $r_j(z)$ ne dépendent pas de l'initialisation. □

Exemple 3.3. Pour la matrice T de la section 3.1, on a $r_1(z) = \zeta z + (1+\zeta)z^2$ et $r_2(z) = (1+\zeta)z$. Avec l'initialisation $\mathbf{t}_0 = (0,1,0)$, on obtient

$$r_1(z)g_0(z) \equiv (\zeta z + (1+\zeta)z^2)(\zeta) \equiv (1+\zeta)z + z^2 \equiv g_1(z) \bmod P(z)$$

et

$$r_2(z)g_0(z) \equiv ((1+\zeta)z)(\zeta) \equiv z \equiv g_2(z) \bmod P(z).$$

Le lecteur peut vérifier que

$$\begin{aligned} r_2(z)g_0(z) &\equiv g_2(z) \bmod P(z) \\ \text{et } r_1(z)g_0(z) &\equiv g_1(z) \bmod P(z) \end{aligned}$$

avec l'initialisation $\mathbf{t}_0 = (0,1,1)$ ou $\mathbf{t}_0 = (0,0,1)$.

On remarque que $r_1(z) \equiv z^{57} \bmod P(z)$. La valeur 57 signifie que la séquence S_1 est 57 itérations en avant dans la récurrence (3.3) par rapport à la séquence S_0. De plus, $r_2(z) \equiv z^{43} \bmod P(z)$, ce qui signifie que la séquence S_2 est 43 itérations en avant par rapport à la séquence S_0. On peut en déduire que la séquence S_1 est 14 itérations en avant par rapport à la séquence S_2. À noter que si $P(z)$ n'est pas primitif, alors $r_j(z)$ n'a pas nécessairement la forme $z^s \bmod P(z)$ pour une valeur de s donnée. Dans ce cas, les séquences S_j et S_0 ne sont pas nécessairement la même séquence avec des points de départ différents.

Lemme 3.2. *Supposons que $P(z)$ est irréductible sur $\mathbb{F}_q$. Soit $0 \neq \mathbf{t}_0 \in \mathbb{F}_q^r$,*

$$\mathbf{G}_\nu(T, \mathbf{t}_0) = (g_0(z), \ldots, g_{\nu-1}(z))/P(z)$$

et

$$\bar{\mathbf{G}}_\nu(T, \mathbf{t}_0) = (1, g_1(z)g_0(z)^{-1} \bmod P(z), \ldots, g_{\nu-1}(z)g_0(z)^{-1} \bmod P(z))/P(z).$$

Le vecteur $\bar{\mathbf{G}}_\nu(T, \mathbf{t}_0)$ *est indépendant de l'initialisation* $\mathbf{t}_0 \in \mathbb{F}_q^r$.

Démonstration. Soit $\mathbf{t}_0 \neq 0$ et $\tilde{\mathbf{t}}_0 \neq 0$, deux initialisations. De plus, soit

$$\mathbf{G}_\nu(T, \mathbf{t}_0) = (g_0(z), \ldots, g_{\nu-1}(z))/P(z)$$

et

$$\mathbf{G}_\nu(T, \tilde{\mathbf{t}}_0) = (\tilde{g}_0(z), \ldots, \tilde{g}_{\nu-1}(z))/P(z).$$

Par la proposition 3.1, on sait qu'il existe un polynôme $f(z)$ tel que

$$\tilde{g}_j(z) = f(z)g_j(z) \bmod P(z)$$

pour $j = 0, \ldots, \nu - 1$. Soit

$$\bar{\mathbf{G}}_\nu(T, \tilde{\mathbf{t}}_0) = (1, \tilde{g}_1(z)\tilde{g}_0(z)^{-1} \bmod P(z), \ldots, \tilde{g}_{\nu-1}(z)\tilde{g}_0(z)^{-1} \bmod P(z))/P(z).$$

Ce vecteur est égal à

$$(1, f(z)g_1(z)(f(z)g_0(z))^{-1} \bmod P(z), \ldots, f(z)g_{\nu-1}(z)(f(z)g_0(z))^{-1} \bmod P(z))/P(z)$$

qui, lui-même, est égal à

$$\bar{\mathbf{G}}_\nu(T, \mathbf{t}_0) = (1, g_1(z)g_0(z)^{-1} \bmod P(z), \ldots, g_{\nu-1}(z)g_0(z)^{-1} \bmod P(z))/P(z).$$

Donc, indépendamment de $\mathbf{t}_0$, $\bar{\mathbf{G}}_\nu(T, \mathbf{t}_0)$ est toujours le même vecteur. □

Exemple 3.4. Soit T, la matrice définie à l'exemple 3.1, et les initialisations $\mathbf{t}_0 = (0, 1, 0)$ et $\tilde{\mathbf{t}}_0 = (0, 1, 1)$. On sait, à partir des exemples 3.1 et 3.2, que $\mathbf{G}_3(T, \mathbf{t}_0) = (\zeta, z^2 + (1 + \zeta)z, z)/P(z)$ et $\mathbf{G}_3(T, \tilde{\mathbf{t}}_0) = (\tilde{g}_0(z), \tilde{g}_1(z), \tilde{g}_2(z))/P(z) = (\zeta + \zeta z, z^2 + z + \zeta, z^2 + z)/P(z)$. L'inverse de ζ dans $\mathbb{F}_4[z]/P(z)$ (ou $\mathbb{F}_4$, peu importe) est $(1 + \zeta)$. Calculons $\bar{\mathbf{G}}(T, \mathbf{t}_0)$:

$$\begin{array}{lcccl}
g_0(z)g_0(z)^{-1} & \equiv & \zeta(1+\zeta) & \equiv & 1 \bmod P(z) \\
g_1(z)g_0(z)^{-1} & \equiv & (z^2 + (1+\zeta)z)(1+\zeta) & \equiv & (1+\zeta)z^2 + \zeta z \bmod P(z) \\
g_2(z)g_0(z)^{-1} & \equiv & z(1+\zeta) & \equiv & (1+\zeta)z \bmod P(z).
\end{array}$$

On obtient $\bar{\mathbf{G}}(T, \mathbf{t}_0) = (1, (1+\zeta)z^2 + \zeta z, (1+\zeta)z)/P(z)$. Vérifions qu'on obtient le même résultat avec l'initialisation $\tilde{\mathbf{t}}_0$. Dans ce cas-ci, $\tilde{g}_0(z)^{-1} \equiv (1+\zeta)z+\zeta z^2 \bmod P(z)$ et

$$\begin{aligned}
\tilde{g}_0(z)\tilde{g}_0(z)^{-1} &\equiv (\zeta+\zeta z)((1+\zeta)z+\zeta z^2) &&\equiv 1 && \bmod P(z)\\
\tilde{g}_1(z)\tilde{g}_0(z)^{-1} &\equiv (z^2+z+\zeta)((1+\zeta)z+\zeta z^2) &&\equiv (1+\zeta)z^2+\zeta z && \bmod P(z)\\
\tilde{g}_2(z)\tilde{g}_0(z)^{-1} &\equiv (z^2+z)((1+\zeta)z+\zeta z^2) &&\equiv (1+\zeta)z && \bmod P(z).
\end{aligned}$$

On voit, dans cet exemple, que le vecteur $\bar{\mathbf{G}}(T, \mathbf{t}_0)$ est le même.

Définition 3.1. Puisque $\bar{\mathbf{G}}_\nu(T, \mathbf{t}_0)$ est indépendant de $\mathbf{t}_0$, définissons $\bar{\mathbf{G}}_\nu(T)$ par le vecteur $\bar{\mathbf{G}}_\nu(T, \mathbf{t}_0)$ pour une initialisation $\mathbf{t}_0 \neq 0 \in \mathbb{F}_q^r$ arbitraire.

Exemple 3.5. Pour la matrice T définie à l'exemple 3.1 et l'initialisation $\mathbf{t}_0 = (0,1,0)$, on a calculé $\bar{\mathbf{G}}(T, \mathbf{t}_0)$ à l'exemple précédent. Par la définition 3.1, on a

$$\bar{\mathbf{G}}_\nu(T) = \bar{\mathbf{G}}(T, \mathbf{t}_0) = (1, (1+\zeta)z^2 + \zeta z, (1+\zeta)z)/P(z).$$

3.3.2 Réseau polynômial construit à partir de $\bar{\mathbf{G}}_\nu(T)$

Dans cette section, on définit le réseau polynômial $\mathcal{L}_\nu(T)$ engendré par la récurrence (3.2). On présente à la fin de cette section un résultat démontrant que l'ensemble

$$\Phi_\nu(T) = \{\mathbf{G}_\nu(T, \mathbf{t}_0) : \mathbf{t}_0 \in \mathbb{F}_q^r\}$$

de tous les vecteurs $\mathbf{G}_\nu(T, \mathbf{t}_0)$ sont des éléments du réseau polynômial $\mathcal{L}_\nu(T)$ et que chaque élément de $\bar{\mathcal{L}}_\nu(T)$, qui est défini à l'équation (3.1), est contenu dans $\Phi_\nu(T)$. Bref, on montrera que $\Phi_\nu(T) = \bar{\mathcal{L}}_\nu(T)$. Les résultats de cette section sont illustrés à l'aide de la matrice définie à la section 3.1.

Lemme 3.3. *Les vecteurs* $\mathbf{v}_0(z), \ldots, \mathbf{v}_{\nu-1}(z)$, *où*

$$\mathbf{v}_0(z) = \bar{\mathbf{G}}_\nu(T)$$

et $\mathbf{v}_i(z)$ *est le* $(i+1)$*-ième vecteur de la base canonique pour* $i = 1, \ldots, \nu-1$, *sont linéairement indépendants sur* $\mathbb{L}_q$.

Démonstration. Il suffit de démontrer que la première coordonnée de $\mathbf{v}_0(z)$ est non nulle. C'est bien le cas, puisque dans le cas contraire, cela signifierait que la première ligne de T est zéro, ce qui contredirait la non singularité de T. □

Définition 3.2. Soit $\mathcal{L}_\nu(T)$, le réseau polynômial généré par la base

$$\{\mathbf{v}_0(z), \ldots, \mathbf{v}_{\nu-1}(z)\}$$

où

$$\mathbf{v}_0(z) = \bar{\mathbf{G}}_\nu(T)$$

et $\mathbf{v}_i(z)$ est le $(i+1)$-ième vecteur de la base canonique pour $i = 1, \ldots, \nu - 1$. Ce réseau est appelé *réseau polynômial en* $(\mathbb{F}_q, \nu)$*-résolution généré par* T.

Proposition 3.2. *Soit* $\bar{\mathbf{G}}_\nu(T) = (1, g_1(z), \ldots, g_{\nu-1}(z))/P(z)$. *Une base du réseau dual* $\mathcal{L}^*_\nu(T)$ *du réseau polynômial* $\mathcal{L}_\nu(T)$ *est*

$$\begin{array}{rcl}
\mathbf{w}_0(z) & = & (\quad P(z) \quad , \; 0 \; , \; \ldots \; , \; 0 \;) \\
\mathbf{w}_1(z) & = & (\quad -g_1(z) \quad , \; 1 \; , \; \ldots \; , \; 0 \;) \\
\vdots & & \vdots \qquad\qquad \ddots \qquad \vdots \;) \\
\mathbf{w}_{\nu-1}(z) & = & (\quad -g_{\nu-1}(z) \quad , \; 0 \; , \; \ldots \; , \; 1 \;)
\end{array}$$

Démonstration. Ceci est vérifié directement à l'aide de la base donnée à la définition 3.2 et par la définition de la base duale. Le produit scalaire de $\mathbf{w}_i(z)$ et $\mathbf{v}_j(z)$ est tel que

$$\mathbf{w}_i(z).\mathbf{v}_j(z) = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{sinon} \end{cases}$$

□

Exemple 3.6. Soit T, la matrice définie à l'exemple 3.1. Le réseau polynômial $\mathcal{L}_3(T)$ est celui engendré par la base

$$\begin{array}{rcl}
\mathbf{v}_0(z) & = (& 1 \quad , \quad (1+\zeta)z^2 + \zeta z \quad , \quad (\zeta+1)z \quad)/P(z) \\
\mathbf{v}_1(z) & = (& 0 \quad , \quad 1 \quad , \quad 0 \quad) \\
\mathbf{v}_2(z) & = (& 0 \quad , \quad 0 \quad , \quad 1 \quad).
\end{array}$$

Une base possible du réseau dual est donnée par

$$\begin{aligned}
\mathbf{w}_0(z) &= (\ z^3+(1+\zeta)z^2+\zeta z+\zeta\ ,\ 0\ ,\ 0\ ) \\
\mathbf{w}_1(z) &= (\ \ (1+\zeta)z^2+\zeta z\ \ ,\ 1\ ,\ 0\ ) \\
\mathbf{w}_2(z) &= (\ \ (\zeta+1)z\ \ ,\ 0\ ,\ 1\ ).
\end{aligned}$$

Soit

$$\Phi_\nu(T) = \{\mathbf{G}_\nu(T,\mathbf{t}_0) : \mathbf{t}_0 \in \mathbb{F}_q^r\}$$

où $\nu \leq r$. Il s'agit de l'ensemble de tous les vecteurs $\mathbf{G}_\nu(T,\mathbf{t}_0)$ construits à partir de toutes les initialisations $\mathbf{t}_0$ possibles. L'ensemble $\Phi_\nu(T)$ est un sous-ensemble du réseau $\mathcal{L}_\nu(T)$ qui contient tous les éléments $\mathbf{v}(z)$ de $\mathcal{L}_\nu(T)$ tels que $\log_q \|\mathbf{v}(\mathbf{z})\| < 0$. C'est ce que démontre le prochain théorème. Pour celui-ci, rappelons que $\bar{\mathcal{L}}_\nu(T) = \mathcal{L}_\nu(T) \cap \mathbb{L}_{q,0}$.

Théorème 3.2. *On a* $\Phi_\nu(T) = \bar{\mathcal{L}}_\nu(T)$.

Démonstration. Soit

$$\mathbf{v}_0(z) = \bar{\mathbf{G}}_\nu(T) = (g_0(z),\ldots,g_{\nu-1}(z))/P(z)$$

où $g_0(z) = 1$ et $\mathbf{v}_j(z), 1 \leq j < \nu$, est le $(j+1)$-ième vecteur de la base canonique.

Démontrons que $\Phi_\nu(T) \subseteq \bar{\mathcal{L}}_\nu(T)$. Soit $\mathbf{G}_\nu(T,\tilde{\mathbf{t}}_0) = (\tilde{g}_0(z),\ldots,\tilde{g}_{\nu-1}(z))/P(z)$, un élément arbitraire de $\Phi_\nu(T)$. Soit $f(z)$, un polynôme tel que

$$\tilde{g}_j(z) \equiv f(z)g_j(z) \bmod P(z),$$

pour $j = 0,\ldots,r-1$. Ce polynôme existe par la proposition 3.1. Dans le cas qui nous concerne, on a $f(z) = \tilde{g}_0(z)$. Soit $C_j(z) = (g_j(z)\tilde{g}_0(z) \bmod P(z))/P(z) - g_j(z)\tilde{g}_0(z)/P(z)$ pour $1 \leq j < \nu$. Notons que $C_j(z) \in \mathbb{F}_q[z]$. Il est facile de vérifier la validité de l'équation

$$(\tilde{g}_0(z),\ldots,\tilde{g}_{\nu-1}(z))/P(z) = \tilde{g}_0(z)\mathbf{v}_0(z) + C_1(z)\mathbf{v}_1(z) + \cdots + C_{\nu-1}(z)\mathbf{v}_{\nu-1}(z).$$

Ceci implique que $\Phi_\nu(T) \subseteq \mathcal{L}_\nu(T)$. Étant donné que tout $\mathbf{G}_\nu(T, \tilde{\mathbf{t}}_0) \in \Phi_\nu(T)$ est tel que $\log_q \|\mathbf{G}_\nu(T, \tilde{\mathbf{t}}_0)\| < 0$, il en résulte que $\Phi_\nu(T) \subseteq \bar{\mathcal{L}}_\nu(T)$.

À présent, démontrons que $\bar{\mathcal{L}}_\nu(T) \subseteq \Phi_\nu(T)$. Soit

$$\mathbf{G}(z) = (G_0(z), \ldots, G_{\nu-1}(z)) = q_0(z)\mathbf{v}_0(z) + \cdots + q_{\nu-1}(z)\mathbf{v}_{\nu-1}(z),$$

un élément arbitraire de $\bar{\mathcal{L}}_\nu(T)$. Étant donné que $\mathbf{G}(z) \in \bar{\mathcal{L}}_\nu(T)$, alors $\log_q |G_j(z)| < 0$ pour $j = 0, \ldots, \nu - 1$. Puisque $\mathbf{v}_0(z) = (1, g_1(z), \ldots, g_{\nu-1}(z))/P(z)$ et que la première coordonnée des vecteurs $\mathbf{v}_1(z), \ldots, \mathbf{v}_{\nu-1}(z)$ est nulle, il est évident que $q_0(z) = P(z)G_0(z)$. Une fois $q_0(z)$ déterminé, on peut établir, sans aucune ambiguïté, que

$$G_j(z) \;=\; q_0(z)g_j(z)/P(z) + q_j(z)$$

pour $j = 1, \ldots, \nu - 1$.

Sans perte de généralité, posons $q_j(z) = -q_0(z)g_j(z) \bmod P(z) + q'_j(z)$ où $q'_j(z)$ est un polynôme dans $\mathbb{F}_q[z]$. Ceci implique que

$$G_j(z) = q_0(z)g_j(z)/P(z) - q_0(z)g_j(z) \bmod P(z) + q'_j(z).$$

De plus, puisque

$$\log_q |q_0(z)g_j(z)/P(z) - q_0(z)g_j(z) \bmod P(z)| < 0$$

et que $q'_j(z) \in \mathbb{F}_q[z]$, alors nécessairement $q'_j(z) = 0$. Dans ce cas,

$$q_j(z) = q_0(z)g_j(z) \bmod P(z)$$

et

$$G_j(z) = q_0(z)g_j(z)/P(z) - q_0(z)g_j(z) \bmod P(z) = (q_0(z)g_j(z) \bmod P(z))/P(z).$$

On obtient

$$\mathbf{G}(z) = (q_0(z), q_0(z)g_1(z) \bmod P(z), \ldots, q_0(z)g_{\nu-1}(z) \bmod P(z))/P(z).$$

On sait qu'il existe une initialisation $\mathbf{t}_0$ telle que

$$\mathbf{G}_\nu(T, \mathbf{t}_0) = (1, g_1(z), \ldots, g_{\nu-1}(z))/P(z) \in \Phi_\nu(T).$$

Soit l'initialisation $\tilde{\mathbf{t}}_0$ telle que $\tilde{\mathbf{t}}_0 = q_0(T)\mathbf{t}_0$. À partir de celle-ci, on obtient

$$\mathbf{G}_\nu(T, \tilde{\mathbf{t}}_0) = (q_0(z), q_0(z)g_1(z) \bmod P(z), \ldots, q_0(z)g_{\nu-1}(z) \bmod P(z))/P(z) = \mathbf{G}(z),$$

ce qui démontre que $\mathbf{G}(z) \in \Phi_\nu(T)$. Puisque que $\mathbf{G}(z)$ a été choisi arbitrairement dans $\bar{\mathcal{L}}_\nu(T)$, ceci implique que $\bar{\mathcal{L}}_\nu(T) \subseteq \Phi_\nu(T)$.

Finalement, puisque $\Phi_\nu(T) \subseteq \bar{\mathcal{L}}_\nu(T)$ et que $\bar{\mathcal{L}}_\nu(T) \subseteq \Phi_\nu(T)$, on peut conclure que $\bar{\mathcal{L}}_\nu(T) = \Phi_\nu(T)$. □

3.4 Réseau en $(\mathbb{F}_p, \ell)$-résolution

Dans cette section, nous partons du réseau polynômial $\mathcal{L}_\nu(T)$ pour en définir un nouveau, $\mathcal{L}_\ell(T(\mathbb{F}_p))$, qui se veut un raffinement du premier réseau. Les éléments de ce nouveau réseau sont des vecteurs en ℓ dimensions de fonctions génératrices de séquences dans $\mathbb{F}_p$. Ce raffinement est utile au chapitre 5 pour démontrer le théorème 5.3.

Pour pouvoir manipuler les éléments de $\mathbb{F}_q$ où $q = p^w$, $w > 1$ et p est une puissance d'un nombre premier, il est nécessaire de représenter ceux-ci dans une base prédéterminée de $\mathbb{F}_q$ sur $\mathbb{F}_p$ (pour le lecteur peu familier avec la notion de base, l'annexe B passe en revue cette notion). Soit $(\beta_1, \ldots, \beta_w)$, une base ordonnée de $\mathbb{F}_q$ sur $\mathbb{F}_p$. La séquence $\{t_n^{(j)}\}_{n\geq 0}$ peut être représentée par une séquence de vecteurs dans $\mathbb{F}_p^w$

$$\{(t_{n,j,1}, \ldots, t_{n,j,w})\}_{n\geq 0}$$

où $t_n^{(j)} = \beta_1 t_{n,j,1} + \cdots + \beta_w t_{n,j,w}$. Ainsi, à partir d'une séquence $\{t_n^{(j)}\}_{n\geq 0}$, on peut définir w nouvelles séquences $\{t_{n,j,i}\}_{n\geq 0}$ pour $i = 1, \ldots, w$. Ces nouvelles séquences

suivent une récurrence linéaire de la forme

$$t_{n,j,i} = \sum_{m=1}^{rw} \kappa_m t_{n-m,j,i} \tag{3.5}$$

dans $\mathbb{F}_p$ où $\kappa_m \in \mathbb{F}_p$ pour $m = 1, \ldots, rw$, et définissent chacune une fonction génératrice

$$h_{jw+i-1}(z) = t_{0,j,i} z^{-1} + t_{1,j,i} z^{-2} + \cdots .$$

Le polynôme caractéristique de cette récurrence est $Q(z) = z^{rw} - \sum_{m=1}^{rw} \kappa_m z^{rw-m}$. On remarque que $P(z)$ divise $Q(z)$, car la séquence $\{t_n^{(j)}\}_{n \geq 0}$ suit également la récurrence (3.5). Le fait que $Q(z)$ soit de degré rw devient évident à lumière de l'équation (3.6). Ainsi, le vecteur $\mathbf{t}_n = (t_n^{(0)}, \ldots, t_n^{(r-1)}) \in \mathbb{F}_q^r$ peut être représenté par le vecteur

$$(t_{n,0,1}, \ldots, t_{n,0,w}, t_{n,1,1}, \ldots, t_{n,1,w}, \ldots, t_{n,r-1,1}, \ldots, t_{n,r-1,w}) \in \mathbb{F}_p^{rw}$$

et les séquences $\{t_{n,j,i}\}_{n \geq 0}$ sont caractérisées par les fonctions génératrices

$$H_{jw+i-1}(z) = h_{jw+i-1}(z)/Q(z) = t_{0,j,i} z^{-1} + t_{1,j,i} z^{-2} + \cdots .$$

On peut également définir la récurrence

$$\mathbf{t}_n = T(\mathbb{F}_p)\mathbf{t}_{n-1} \in \mathbb{F}_p^{rw} \tag{3.6}$$

où $T(\mathbb{F}_p)$ est une matrice de dimension $rw \times rw$ obtenue en remplaçant chaque élément $T_{i,j}$ de T par $A_{T_{i,j}}$, une matrice $w \times w$, dont ses éléments sont dans $\mathbb{F}_p$, qui effectue la multiplication par $T_{i,j}$, mais dans la représentation de la base $(\beta_1, \ldots, \beta_w)$. On remarque que la récurrence (3.6) entre dans le cadre défini par la récurrence (3.2) et que (3.6) peut engendrer des réseaux polynômiaux $\mathcal{L}_\ell(T(\mathbb{F}_p))$ pour $\ell = 1, \ldots, rw$. La base du réseau $\mathcal{L}_\ell(T(\mathbb{F}_p))$ est $\{\bar{\mathbf{v}}_0(z), \ldots, \bar{\mathbf{v}}_\ell(z)\}$ où $\bar{\mathbf{v}}_0(z) = \bar{\mathbf{G}}_\ell(T(\mathbb{F}_p))$ et $\bar{\mathbf{v}}_j(z)$ est le $(j+1)$-ième vecteur de la base canonique pour $j = 1, \ldots, \ell - 1$. Dans ce qui suit, on explique comment obtenir $\bar{\mathbf{G}}_\ell(T(\mathbb{F}_p))$ à partir de $\bar{\mathbf{G}}_r(T)$

Soit $(\beta_1, \ldots, \beta_w)$, une base ordonnée, et $\{x_j\}_{j \geq 0}$, une séquence d'éléments dans $\mathbb{F}_q$ avec sa fonction génératrice

$$G(z) = g(z)/P(z) = x_0 z^{-1} + x_1 z^{-2} + \cdots \in \mathbb{L}_q$$

où $x_i = (x_{i1}\beta_1 + \cdots + x_{iw}\beta_w) \in \mathbb{F}_q$ et $x_{ij} \in \mathbb{F}_p$ pour $j = 1, \ldots, w$, $i \geq 0$. Alors $G(z)$ vérifie les équations suivantes

$$\begin{aligned} G(z) &= (x_{01}\beta_1 + \cdots + x_{0w}\beta_w)z^{-1} + (x_{11}\beta_1 + \cdots + x_{1w}\beta_w)z^{-2} + \cdots \\ &= \sum_{i\geq 0}(x_{i1}\beta_1 + \cdots + x_{iw}\beta_w)z^{-i-1} \\ &= \sum_{i\geq 0} x_{i1}\beta_1 z^{-i-1} + \cdots + \sum_{i\geq 0} x_{iw}\beta_w z^{-i-1} \\ &= \beta_1 h_0(z)/Q(z) + \cdots + \beta_w h_{w-1}(z)/Q(z) \end{aligned}$$

où $h_j(z) \in \mathbb{F}_p[z]$ pour $i \geq 1$ et $0 \leq j < w$, et $Q(z) \in \mathbb{F}_p[z]$ est le polynôme minimal de la séquence $\{x_i\}_{i\geq 0}$ sur $\mathbb{F}_p$. Alors, on obtient

$$g(z)/P(z) = \beta_1 h_0(z)/Q(z) + \cdots + \beta_w h_{w-1}(z)/Q(z) \tag{3.7}$$

où $P(z)$ divise $Q(z)$ dans $\mathbb{F}_q[z]$ parce que $Q(z)$ et $P(z)$ sont des polynômes caractéristiques de la même récurrence dans $\mathbb{F}_q$. De cette dernière équation, il est facile de déterminer les polynômes $h_j(z) \in \mathbb{F}_p[z]$ pour $j = 0, \ldots, w-1$ à partir de $g(z)$.

Exemple 3.7. Soit $g(z) = 1$ et $P(z) = z^3 + (1+\zeta)z^2 + \zeta z + \zeta$. On a

$$G(z) = g(z)/P(z) = 0z^{-1} + 0z^{-2} + z^{-3} + (1+\zeta)z^{-4} + 0z^{-5} + (1+\zeta)z^{-6} + \cdots .$$

Soit $(\beta_1, \beta_2) = (1, \zeta)$, une base ordonnée de $\mathbb{F}_4$ sur $\mathbb{F}_2$. Dans ce cas, $q = 4$, $p = 2$ et $w = 2$. Avec cette base, on obtient

$$\begin{aligned} G(z) = g(z)/P(z) &= \beta_1.(0z^{-1} + 0z^{-2} + z^{-3} + z^{-4} + 0z^{-5} + z^{-6} + \cdots) \\ &+ \beta_2(0z^{-1} + 0z^{-2} + 0z^{-3} + z^{-4} + 0z^{-5} + z^{-6} + \cdots) \\ &= \beta_1 h_0(z)/Q(z) + \beta_2 h_1(z)/Q(z) \end{aligned}$$

Le polynôme minimal de la séquence $0, 0, 1, (1+\zeta), 0, (1+\zeta), \ldots$ sur $\mathbb{F}_2$ est $Q(z) = z^6 + z^5 + 1$. Ceci est facilement vérifiable en observant que $Q(z)$ est irréductible sur $\mathbb{F}_2$ et que la séquence obéit à la récurrence

$$x_{n+6} = x_{n+5} + x_n. \tag{3.8}$$

On remarque que $P(z)|Q(z)$ puisque $Q(z) = P(z)(z^3 + \zeta z^2 + (1+\zeta)z + (1+\zeta))$. Les polynômes $h_j(z)$, $j = 0, 1$, sont $h_0(z) = z^3 + z + 1$ et $h_1(z) = z^2 + z + 1$.

Définition 3.3. On appelle le réseau $\mathcal{L}_\ell(T(\mathbb{F}_p))$ le *réseau polynômial en* $(\mathbb{F}_p, \ell)$-*résolution* décrit par T.

Soit $\bar{\mathbf{G}}_\nu(T) = (1, g_1(z), \ldots, g_{\nu-1}(z))/P(z)$. De l'équation (3.7), on voit que

$$\mathbf{G}_\ell(T(\mathbb{F}_p), \mathbf{t}_0) = (h_0(z), \ldots, h_{\ell-1}(z))/Q(z) \tag{3.9}$$

pour une certaine initialisation $\mathbf{t}_0 \in \mathbb{F}_p^{rw}$ où $\ell \leq \nu w$ et

$$g_j(z)/P(z) = (\beta_1 h_{jw}(z) + \cdots + \beta_w h_{jw+w-1}(z))/Q(z) \tag{3.10}$$

pour $j = 0, \ldots, \nu - 1$. À partir de ces équations, il est possible de construire une base pour le réseau en $(\mathbb{F}_p, \ell)$-résolution $\mathcal{L}_\ell(T(\mathbb{F}_p))$. Ceci est illustré dans l'exemple 3.8.

Mais avant de passer à cet exemple qui clôt ce chapitre, nous précisons que pour la plupart des générateurs dont il est question au chapitre 2, on a $q = 2$ (donc $w = 1$), $r = k$ et que la matrice T est la matrice X de transition. Ainsi, le raffinement dont il est question dans cette section n'est pas possible et les réseaux polynômiaux en $(\mathbb{F}_q, \nu)$-résolution et en $(\mathbb{F}_p, \ell)$-résolution sont le même réseau quand $\ell = \nu$. Par contre, ce raffinement est très utile pour les nouveaux générateurs introduits au chapitre 5.

Exemple 3.8. Soit T, la matrice définie à l'exemple 3.1. Pour représenter les éléments de $\mathbb{F}_4$, on utilise la base ordonnée polynômiale $(\beta_1, \beta_2) = (1, \zeta)$. Correspondant à la matrice T, la matrice $T(\mathbb{F}_2)$ est

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}.$$

Soit la récurrence $\mathbf{t}_n = T(\mathbb{F}_2)\mathbf{t}_{n-1}$. On pourrait calculer $\bar{\mathbf{G}}_6(T(\mathbb{F}_2))$ de la même manière qu'on a calculé $\bar{\mathbf{G}}_3(T)$ dans les exemples 3.1–3.6, mais nous utiliserons plutôt (3.10). On sait que $\bar{\mathbf{G}}_3(T) = (1, (1+\zeta)z^2+\zeta z, (1+\zeta)z)/P(z)$. Pour une initialisation $\mathbf{t}_0 \in \mathbb{F}_2^6$ non encore déterminée, calculons $\mathbf{G}_6(T(\mathbb{F}_2), \mathbf{t}_0 = (h_0(z), \ldots, h_5(z))/Q(z)$. À l'exemple 3.7, avec $g_0(z) = 1/P(z)$, on a obtenu $h_0(z) = z^3+z+1$ et $h_1(z) = z^2+z+1$. En procédant de la même manière qu'à l'exemple 3.7 avec $g_1(z)$ et $g_2(z)$, on obtient

$$\begin{aligned} g_1(z)/P(z) &= (1+\zeta)z^{-1} + 0z^{-2} + z^{-3} + \zeta z^{-4} + (1+\zeta)z^{-5} + (1+\zeta)z^{-6} + \cdots \\ g_2(z)/P(z) &= 0z^{-1} + (1+\zeta)z^{-2} + \zeta z^{-3} + 0z^{-4} + \zeta z^{-5} + \zeta z^{-6} + \cdots \end{aligned}$$

et

$$\begin{aligned} h_2(z)/Q(z) &= z^{-1} + 0z^{-2} + z^{-3} + 0z^{-4} + z^{-5} + z^{-6} + \cdots \\ h_3(z)/Q(z) &= z^{-1} + 0z^{-2} + 0z^{-3} + z^{-4} + z^{-5} + z^{-6} + \cdots \\ h_4(z)/Q(z) &= 0z^{-1} + z^{-2} + 0z^{-3} + 0z^{-4} + 0z^{-5} + 0z^{-6} + \cdots \\ h_5(z)/Q(z) &= 0z^{-1} + 1z^{-2} + 1z^{-3} + 0z^{-4} + z^{-5} + z^{-6} + \cdots \end{aligned}$$

où

$$\begin{aligned} h_2(z) &= z^5 + z^4 + z^3 + z^2 + z \\ h_3(z) &= z^5 + z^4 + z^2 \\ h_4(z) &= z^4 + z^3 \\ h_5(z) &= z^4 + z^2 + z \end{aligned}$$

Soit la séquence $R_{jw+i-1} = \{t_{n,j,i}\}_{n\geq 0}$ pour $j = 0, 1, 2$ et $i = 1, 2$. Il est facile de calculer z^{21} mod $P(z)$ et de démontrer que $h_1(z) \equiv z^{21}h_0(z)$ mod $P(z)$. Ceci signifie que la séquence R_1 est en avance de 21 itérations sur la séquence R_0 dans la récurrence (3.8). Aussi, on observe que $h_2(z) \equiv z^{57}h_0(z)$ mod $P(z)$, ce qui indique que la séquence R_2 est en avance de 57 itérations sur la R_0, ce qu'on aurait pu prédire puisqu'on avait remarqué le même phénomène entre les séquences S_1 et S_0 à l'exemple 3.3.

Puisque la même base de $\mathbb{F}_4$ sur $\mathbb{F}_2$ est utilisée pour représenter les éléments des séquences S_0, S_1 et S_2, on peut « prédire » que $h_3(z) \equiv z^{21}h_2(z)$ et $h_5(z) \equiv z^{21}h_4(z)$. On peut également prédire, à partir des résultats obtenus à l'exemple 3.3, que $h_4(z) \equiv z^{43}h_0(z)$. Grâce à tous ces résultats et le fait que $P(z)$ soit primitif, on peut prédire

l'avance de chacune des séquences R_j, $1 \leq j \leq 5$, par rapport à la séquence R_0 dans la récurrence (3.8). Le tableau 3.4 illustre les séquences S_1, S_2, S_3 ainsi que leurs séquences dérivées $R_0, \ldots, R_5$. Pour ce faire, on utilise comme référence la séquence S_0 dont la fonction génératrice est

$$G_0(z) = 1/P(z) = x_0 z^{-1} + x_1 z^{-2} + x_2 z^{-3} + x_3 z^{-4} + x_4 z^{-5} + \cdots$$

où $x_n \in \mathbb{F}_4$ pour $n \geq 0$. Étant donné que $G_1(z) = (z^{57} g_0(z) \bmod P(z))/P(z)$, on a

$$G_1(z) = 1/P(z) = x_{57} z^{-1} + x_{58} z^{-2} + x_{59} z^{-3} + x_{60} z^{-4} + x_{61} z^{-5} + \cdots .$$

On peut obtenir $G_2(z)$ de la même manière. Dans les premières colonnes du tableau 3.4, on donne les fonctions génératrices $G_0(z)$, $G_1(z)$ et $G_2(z)$. On peut aussi considérer la $(i+1)$-ième colonne comme la séquence S_i, pour $i = 1, 2, 3$.

Soit

$$H_0(z) = h_0(z)/Q(z) = \bar{x}_0 z^{-1} + \bar{x}_1 z^{-2} + \bar{x}_2 z^{-3} + \bar{x}_3 z^{-4} + \bar{x}_4 z^{-5} + \cdots .$$

On peut se servir de cette fonction génératrice comme référence pour décrire $H_1(z)$, $\ldots, H_5(z)$. Par exemple, puisque $H_1(z) = (z^{21} h_0(z) \bmod Q(z))/Q(z)$, on peut écrire

$$H_1(z) = \bar{x}_{21} z^{-1} + \bar{x}_{22} z^{-2} + \bar{x}_{23} z^{-3} + \bar{x}_{24} z^{-4} + \bar{x}_{25} z^{-5} + \cdots .$$

On peut exprimer $H_2(z), \ldots, H_5(z)$ de la même manière. Dans le tableau 3.4, on donne les fonctions génératrices $H_0(z), \ldots, H_5(z)$ dans les six dernières colonnes. On peut aussi considérer la $(i+6)$-ième colonne comme la séquence R_i pour $i = 0, \ldots, 5$.

On peut maintenant déterminer $\bar{\mathbf{G}}_6(T(\mathbb{F}_2))$ en calculant l'inverse multiplicatif de $h_0(z)$ dans $\mathbb{F}_2[z]/Q(z)$, soit $h_0(z)^{-1}$, et en multipliant $h_j(z)$ par $h_0(z)^{-1}$ dans

$\mathbb{F}_2[z]/Q(z)$, pour $j = 0, \ldots, 5$. On a $h_0(z)^{-1} \equiv z^5 + z^2 + 1 \bmod Q(z)$ et

$$\begin{aligned}
h_0(z)h_0(z)^{-1} &\equiv 1 && \bmod Q(z)\\
h_1(z)h_0(z)^{-1} &\equiv z^5 + z^4 + z^3 + 1 && \bmod Q(z)\\
h_2(z)h_0(z)^{-1} &\equiv z^5 + z^4 + 1 && \bmod Q(z)\\
h_3(z)h_0(z)^{-1} &\equiv z^4 + z^2 + 1 && \bmod Q(z)\\
h_4(z)h_0(z)^{-1} &\equiv z^4 + 1 && \bmod Q(z)\\
h_5(z)h_0(z)^{-1} &\equiv z && \bmod Q(z)
\end{aligned}$$

Ainsi, le réseau polynômial $\mathcal{L}_6(T(\mathbb{F}_2))$ a pour base

$$\begin{aligned}
\mathbf{v}_0(z) &= (1, \ z^5 + z^4 + z^3 + 1 \ , \ z^5 + z^4 + 1 \ , \ z^4 + z^2 + 1 \ , \ z^4 + 1 \ , z)/Q(z)\\
\mathbf{v}_1(z) &= (0, \ 1 \ , \ 0 \ , \ 0 \ , \ 0 \ , 0)\\
\mathbf{v}_2(z) &= (0, \ 0 \ , \ 1 \ , \ 0 \ , \ 0 \ , 0)\\
\mathbf{v}_3(z) &= (0, \ 0 \ , \ 0 \ , \ 1 \ , \ 0 \ , 0)\\
\mathbf{v}_4(z) &= (0, \ 0 \ , \ 0 \ , \ 0 \ , \ 1 \ , 0)\\
\mathbf{v}_5(z) &= (0, \ 0 \ , \ 0 \ , \ 0 \ , \ 0 \ , 1)
\end{aligned}$$

et une base du réseau dual $\mathcal{L}_6^*(T(\mathbb{F}_2))$ est

$$\begin{aligned}
\mathbf{w}_0(z) &= (z^6 + z^5 + 1 \ , 0, 0, 0, 0, 0)\\
\mathbf{w}_1(z) &= (z^5 + z^4 + z^3 + 1 \ , 1, 0, 0, 0, 0)\\
\mathbf{w}_2(z) &= (z^5 + z^4 + 1 \ , 0, 1, 0, 0, 0)\\
\mathbf{w}_3(z) &= (z^4 + z^2 + 1 \ , 0, 0, 1, 0, 0)\\
\mathbf{w}_4(z) &= (z^4 + 1 \ , 0, 0, 0, 1, 0)\\
\mathbf{w}_5(z) &= (z \ , 0, 0, 0, 0, 1)
\end{aligned}$$

Considérons maintenant le réseau polynômial $\mathcal{L}_2(T(\mathbb{F}_2))$. En prenant chacun de ses éléments $\mathbf{G}(z) \in \mathcal{L}_2(T(\mathbb{F}_2))$, et en plaçant dans $[0,1)^2$ les points en deux dimensions obtenus en remplaçant z par 2 dans l'expression de $\mathbf{G}(z)$, on obtient l'ensemble de points illustré à la figure 3.2. En calculant le plus court vecteur dans le réseau $\mathcal{L}_2^*(T(\mathbb{F}_2))$ (qui ne correspond pas à la plus courte distance entre deux points illustré

Tableau 3.1 – Les fonctions génératrices $G_0(z)$, $G_1(z)$ et $G_2(z)$ exprimées en fonction de $G_0(z)$, ainsi que les fonctions génératrices $H_0(z), \ldots, H_5(z)$ en fonction de $H_0(z)$.

	$G_0(z)$	$G_1(z)$	$G_2(z)$		$H_0(z)$	$H_1(z)$	$H_2(z)$	$H_3(z)$	$H_4(z)$	$H_5(z)$
z^{-1}	x_0	x_{57}	x_{43}		$\bar{x}_0$	$\bar{x}_{21}$	$\bar{x}_{57}$	$\bar{x}_{15}$	$\bar{x}_{43}$	$\bar{x}_1$
z^{-2}	x_1	x_{58}	x_{44}		$\bar{x}_1$	$\bar{x}_{22}$	$\bar{x}_{58}$	$\bar{x}_{16}$	$\bar{x}_{44}$	$\bar{x}_2$
z^{-3}	x_2	x_{59}	x_{45}		$\bar{x}_2$	$\bar{x}_{23}$	$\bar{x}_{59}$	$\bar{x}_{17}$	$\bar{x}_{45}$	$\bar{x}_3$
z^{-4}	x_3	x_{60}	x_{46}		$\bar{x}_3$	$\bar{x}_{24}$	$\bar{x}_{60}$	$\bar{x}_{18}$	$\bar{x}_{46}$	$\bar{x}_4$
z^{-5}	x_4	x_{61}	x_{47}		$\bar{x}_4$	$\bar{x}_{25}$	$\bar{x}_{61}$	$\bar{x}_{19}$	$\bar{x}_{47}$	$\bar{x}_5$
z^{-6}	x_5	x_{62}	x_{48}		$\bar{x}_5$	$\bar{x}_{26}$	$\bar{x}_{62}$	$\bar{x}_{20}$	$\bar{x}_{48}$	$\bar{x}_6$
z^{-7}	x_6	x_0	x_{49}		$\bar{x}_6$	$\bar{x}_{27}$	$\bar{x}_0$	$\bar{x}_{21}$	$\bar{x}_{49}$	$\bar{x}_7$
z^{-8}	x_7	x_1	x_{50}		$\bar{x}_7$	$\bar{x}_{28}$	$\bar{x}_1$	$\bar{x}_{22}$	$\bar{x}_{50}$	$\bar{x}_8$
z^{-9}	x_8	x_2	x_{51}		$\bar{x}_8$	$\bar{x}_{29}$	$\bar{x}_2$	$\bar{x}_{23}$	$\bar{x}_{51}$	$\bar{x}_9$
z^{-10}	x_9	x_3	x_{52}		$\bar{x}_9$	$\bar{x}_{30}$	$\bar{x}_3$	$\bar{x}_{24}$	$\bar{x}_{52}$	$\bar{x}_{10}$
z^{-11}	x_{10}	x_4	x_{53}		$\bar{x}_{10}$	$\bar{x}_{31}$	$\bar{x}_4$	$\bar{x}_{25}$	$\bar{x}_{53}$	$\bar{x}_{11}$
z^{-12}	x_{11}	x_5	x_{54}		$\bar{x}_{11}$	$\bar{x}_{32}$	$\bar{x}_5$	$\bar{x}_{26}$	$\bar{x}_{54}$	$\bar{x}_{12}$
z^{-13}	x_{12}	x_6	x_{55}		$\bar{x}_{12}$	$\bar{x}_{33}$	$\bar{x}_6$	$\bar{x}_{27}$	$\bar{x}_{55}$	$\bar{x}_{13}$
z^{-14}	x_{13}	x_7	x_{56}		$\bar{x}_{13}$	$\bar{x}_{34}$	$\bar{x}_7$	$\bar{x}_{28}$	$\bar{x}_{56}$	$\bar{x}_{14}$
z^{-15}	x_{14}	x_8	x_{57}		$\bar{x}_{14}$	$\bar{x}_{35}$	$\bar{x}_8$	$\bar{x}_{29}$	$\bar{x}_{57}$	$\bar{x}_{15}$
z^{-16}	x_{15}	x_9	x_{58}		$\bar{x}_{15}$	$\bar{x}_{36}$	$\bar{x}_9$	$\bar{x}_{30}$	$\bar{x}_{58}$	$\bar{x}_{16}$
z^{-17}	x_{16}	x_{10}	x_{59}		$\bar{x}_{16}$	$\bar{x}_{37}$	$\bar{x}_{10}$	$\bar{x}_{31}$	$\bar{x}_{59}$	$\bar{x}_{17}$
z^{-18}	x_{17}	x_{11}	x_{60}		$\bar{x}_{17}$	$\bar{x}_{38}$	$\bar{x}_{11}$	$\bar{x}_{32}$	$\bar{x}_{60}$	$\bar{x}_{18}$
z^{-19}	x_{18}	x_{12}	x_{61}		$\bar{x}_{18}$	$\bar{x}_{39}$	$\bar{x}_{12}$	$\bar{x}_{33}$	$\bar{x}_{61}$	$\bar{x}_{19}$
z^{-20}	x_{19}	x_{13}	x_{62}		$\bar{x}_{19}$	$\bar{x}_{40}$	$\bar{x}_{13}$	$\bar{x}_{34}$	$\bar{x}_{62}$	$\bar{x}_{20}$
z^{-21}	x_{20}	x_{14}	x_0		$\bar{x}_{20}$	$\bar{x}_{41}$	$\bar{x}_{14}$	$\bar{x}_{35}$	$\bar{x}_0$	$\bar{x}_{21}$
z^{-22}	x_{21}	x_{15}	x_1		$\bar{x}_{21}$	$\bar{x}_{42}$	$\bar{x}_{15}$	$\bar{x}_{36}$	$\bar{x}_1$	$\bar{x}_{22}$
z^{-23}	x_{22}	x_{16}	x_2		$\bar{x}_{22}$	$\bar{x}_{43}$	$\bar{x}_{16}$	$\bar{x}_{37}$	$\bar{x}_2$	$\bar{x}_{23}$
z^{-24}	x_{23}	x_{17}	x_3		$\bar{x}_{23}$	$\bar{x}_{44}$	$\bar{x}_{17}$	$\bar{x}_{38}$	$\bar{x}_3$	$\bar{x}_{24}$
z^{-25}	x_{24}	x_{18}	x_4		$\bar{x}_{24}$	$\bar{x}_{45}$	$\bar{x}_{18}$	$\bar{x}_{39}$	$\bar{x}_4$	$\bar{x}_{25}$
z^{-26}	x_{25}	x_{19}	x_5		$\bar{x}_{25}$	$\bar{x}_{46}$	$\bar{x}_{19}$	$\bar{x}_{40}$	$\bar{x}_5$	$\bar{x}_{26}$
z^{-27}	x_{26}	x_{20}	x_6		$\bar{x}_{26}$	$\bar{x}_{47}$	$\bar{x}_{20}$	$\bar{x}_{41}$	$\bar{x}_6$	$\bar{x}_{27}$
z^{-28}	x_{27}	x_{21}	x_7		$\bar{x}_{27}$	$\bar{x}_{48}$	$\bar{x}_{21}$	$\bar{x}_{42}$	$\bar{x}_7$	$\bar{x}_{28}$
z^{-29}	x_{28}	x_{22}	x_8		$\bar{x}_{28}$	$\bar{x}_{49}$	$\bar{x}_{22}$	$\bar{x}_{43}$	$\bar{x}_8$	$\bar{x}_{29}$
z^{-30}	x_{29}	x_{23}	x_9		$\bar{x}_{29}$	$\bar{x}_{50}$	$\bar{x}_{23}$	$\bar{x}_{44}$	$\bar{x}_9$	$\bar{x}_{30}$
z^{-31}	x_{30}	x_{24}	x_{10}		$\bar{x}_{30}$	$\bar{x}_{51}$	$\bar{x}_{24}$	$\bar{x}_{45}$	$\bar{x}_{10}$	$\bar{x}_{31}$
z^{-32}	x_{31}	x_{25}	x_{11}		$\bar{x}_{31}$	$\bar{x}_{52}$	$\bar{x}_{25}$	$\bar{x}_{46}$	$\bar{x}_{11}$	$\bar{x}_{32}$
z^{-33}	x_{32}	x_{26}	x_{12}		$\bar{x}_{32}$	$\bar{x}_{53}$	$\bar{x}_{26}$	$\bar{x}_{47}$	$\bar{x}_{12}$	$\bar{x}_{33}$
z^{-34}	x_{33}	x_{27}	x_{13}		$\bar{x}_{33}$	$\bar{x}_{54}$	$\bar{x}_{27}$	$\bar{x}_{48}$	$\bar{x}_{13}$	$\bar{x}_{34}$
z^{-35}	x_{34}	x_{28}	x_{14}		$\bar{x}_{34}$	$\bar{x}_{55}$	$\bar{x}_{28}$	$\bar{x}_{49}$	$\bar{x}_{14}$	$\bar{x}_{35}$
z^{-36}	x_{35}	x_{29}	x_{15}		$\bar{x}_{35}$	$\bar{x}_{56}$	$\bar{x}_{29}$	$\bar{x}_{50}$	$\bar{x}_{15}$	$\bar{x}_{36}$
z^{-37}	x_{36}	x_{30}	x_{16}		$\bar{x}_{36}$	$\bar{x}_{57}$	$\bar{x}_{30}$	$\bar{x}_{51}$	$\bar{x}_{16}$	$\bar{x}_{37}$
z^{-38}	x_{37}	x_{31}	x_{17}		$\bar{x}_{37}$	$\bar{x}_{58}$	$\bar{x}_{31}$	$\bar{x}_{52}$	$\bar{x}_{17}$	$\bar{x}_{38}$
z^{-39}	x_{38}	x_{32}	x_{18}		$\bar{x}_{38}$	$\bar{x}_{59}$	$\bar{x}_{32}$	$\bar{x}_{53}$	$\bar{x}_{18}$	$\bar{x}_{39}$
z^{-40}	x_{39}	x_{33}	x_{19}		$\bar{x}_{39}$	$\bar{x}_{60}$	$\bar{x}_{33}$	$\bar{x}_{54}$	$\bar{x}_{19}$	$\bar{x}_{40}$
z^{-41}	x_{40}	x_{34}	x_{20}		$\bar{x}_{40}$	$\bar{x}_{61}$	$\bar{x}_{34}$	$\bar{x}_{55}$	$\bar{x}_{20}$	$\bar{x}_{41}$
z^{-42}	x_{41}	x_{35}	x_{21}		$\bar{x}_{41}$	$\bar{x}_{62}$	$\bar{x}_{35}$	$\bar{x}_{56}$	$\bar{x}_{21}$	$\bar{x}_{42}$
z^{-43}	x_{42}	x_{36}	x_{22}		$\bar{x}_{42}$	$\bar{x}_0$	$\bar{x}_{36}$	$\bar{x}_{57}$	$\bar{x}_{22}$	$\bar{x}_{43}$
z^{-44}	x_{43}	x_{37}	x_{23}		$\bar{x}_{43}$	$\bar{x}_1$	$\bar{x}_{37}$	$\bar{x}_{58}$	$\bar{x}_{23}$	$\bar{x}_{44}$
z^{-45}	x_{44}	x_{38}	x_{24}		$\bar{x}_{44}$	$\bar{x}_2$	$\bar{x}_{38}$	$\bar{x}_{59}$	$\bar{x}_{24}$	$\bar{x}_{45}$
z^{-46}	x_{45}	x_{39}	x_{25}		$\bar{x}_{45}$	$\bar{x}_3$	$\bar{x}_{39}$	$\bar{x}_{60}$	$\bar{x}_{25}$	$\bar{x}_{46}$
z^{-47}	x_{46}	x_{40}	x_{26}		$\bar{x}_{46}$	$\bar{x}_4$	$\bar{x}_{40}$	$\bar{x}_{61}$	$\bar{x}_{26}$	$\bar{x}_{47}$
z^{-48}	x_{47}	x_{41}	x_{27}		$\bar{x}_{47}$	$\bar{x}_5$	$\bar{x}_{41}$	$\bar{x}_{62}$	$\bar{x}_{27}$	$\bar{x}_{48}$
z^{-49}	x_{48}	x_{42}	x_{28}		$\bar{x}_{48}$	$\bar{x}_6$	$\bar{x}_{42}$	$\bar{x}_0$	$\bar{x}_{28}$	$\bar{x}_{49}$
z^{-50}	x_{49}	x_{43}	x_{29}		$\bar{x}_{49}$	$\bar{x}_7$	$\bar{x}_{43}$	$\bar{x}_1$	$\bar{x}_{29}$	$\bar{x}_{50}$
z^{-51}	x_{50}	x_{44}	x_{30}		$\bar{x}_{50}$	$\bar{x}_8$	$\bar{x}_{44}$	$\bar{x}_2$	$\bar{x}_{30}$	$\bar{x}_{51}$
z^{-52}	x_{51}	x_{45}	x_{31}		$\bar{x}_{51}$	$\bar{x}_9$	$\bar{x}_{45}$	$\bar{x}_3$	$\bar{x}_{31}$	$\bar{x}_{52}$
z^{-53}	x_{52}	x_{46}	x_{32}		$\bar{x}_{52}$	$\bar{x}_{10}$	$\bar{x}_{46}$	$\bar{x}_4$	$\bar{x}_{32}$	$\bar{x}_{53}$
z^{-54}	x_{53}	x_{47}	x_{33}		$\bar{x}_{53}$	$\bar{x}_{11}$	$\bar{x}_{47}$	$\bar{x}_5$	$\bar{x}_{33}$	$\bar{x}_{54}$
z^{-55}	x_{54}	x_{48}	x_{34}		$\bar{x}_{54}$	$\bar{x}_{12}$	$\bar{x}_{48}$	$\bar{x}_6$	$\bar{x}_{34}$	$\bar{x}_{55}$
z^{-56}	x_{55}	x_{49}	x_{35}		$\bar{x}_{55}$	$\bar{x}_{13}$	$\bar{x}_{49}$	$\bar{x}_7$	$\bar{x}_{35}$	$\bar{x}_{56}$
z^{-57}	x_{56}	x_{50}	x_{36}		$\bar{x}_{56}$	$\bar{x}_{14}$	$\bar{x}_{50}$	$\bar{x}_8$	$\bar{x}_{36}$	$\bar{x}_{57}$
z^{-58}	x_{57}	x_{51}	x_{37}		$\bar{x}_{57}$	$\bar{x}_{15}$	$\bar{x}_{51}$	$\bar{x}_9$	$\bar{x}_{37}$	$\bar{x}_{58}$
z^{-59}	x_{58}	x_{52}	x_{38}		$\bar{x}_{58}$	$\bar{x}_{16}$	$\bar{x}_{52}$	$\bar{x}_{10}$	$\bar{x}_{38}$	$\bar{x}_{59}$
z^{-60}	x_{59}	x_{53}	x_{39}		$\bar{x}_{59}$	$\bar{x}_{17}$	$\bar{x}_{53}$	$\bar{x}_{11}$	$\bar{x}_{39}$	$\bar{x}_{60}$
z^{-61}	x_{60}	x_{54}	x_{40}		x_{60}	$\bar{x}_{18}$	$\bar{x}_{54}$	$\bar{x}_{12}$	$\bar{x}_{40}$	$\bar{x}_{61}$
z^{-62}	x_{61}	x_{55}	x_{41}		$\bar{x}_{61}$	$\bar{x}_{19}$	$\bar{x}_{55}$	$\bar{x}_{13}$	$\bar{x}_{41}$	$\bar{x}_{62}$
z^{-63}	x_{62}	x_{56}	x_{42}		$\bar{x}_{62}$	$\bar{x}_{20}$	x_{56}	$\bar{x}_{14}$	$\bar{x}_{42}$	$\bar{x}_0$
z^{-64}	x_0	x_{57}	x_{43}		$\bar{x}_0$	$\bar{x}_{21}$	$\bar{x}_{57}$	$\bar{x}_{15}$	$\bar{x}_{43}$	$\bar{x}_1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

à la figure 3.2), on peut déterminer la valeur t_2 par un théorème qui est présenté au prochain chapitre.

Chapitre 4

Critères d'uniformité

Pour un générateur de nombres aléatoires, la séquence des nombres produits par celui-ci doit se comporter le plus fidèlement possible comme une séquence de variables aléatoires $\{U_n\}_{n\geq 0}$ indépendantes uniformes sur l'intervalle $[0,1)$. Bien sûr, comme il a été dit précédemment, il lui est impossible de remplir son rôle à la perfection étant donné son caractère déterministe. Malgré cela, on peut parvenir assez près du but visé pour que les générateurs de nombres aléatoires soient utiles.

Les variables aléatoires uniformes sur $[0,1)$ de la séquence $\{U_n\}_{n\geq 0}$ sont indépendantes et identiquement distribuées si et seulement si, pour tout entier $i > 0$ et $t > 0$, le vecteur $\mathbf{u}_{i,t} = (U_i, \ldots, U_{i+t-1})$ est uniformément distribué sur $[0,1)^t$. Pour un vecteur $\mathbf{u}$ uniformément distribué sur $[0,1)^t$, on a la propriété suivante.

Proposition 4.1. *Un vecteur* $\mathbf{u} \in [0,1)^t$ *est uniformément distribué sur* $[0,1)^t$ *si et seulement si la probabilité que celui-ci soit contenu dans un sous-ensemble mesurable* M *de* $[0,1)^t$ *est égale au volume de ce sous-ensemble.*

Par cette propriété, on peut tenter d'évaluer dans quelle mesure la séquence $\{u_n\}_{n\geq 0}$ produite par un générateur de nombres aléatoires imite bien une séquence de

variables aléatoires uniformes et indépendantes sur $[0,1)$. Une manière de faire cela est par l'équidistribution.

Pour déterminer l'équidistribution, on divise l'hypercube $[0,1)^t$ en $2^{t\ell}$ hypercubes de même taille et de même volume $2^{-t\ell}$. Pour un ensemble de points P_n de $n = 2^k$ points qui sont tous des variables aléatoires uniformes indépendantes sur $[0,1)^t$, par la proposition 4.1, on devrait retrouver, en moyenne, $2^{k-t\ell}$ points dans chaque hypercube. Pour un ensemble de point P_n donné, la valeur de Λ_t (définie dans la section 1.5) est une mesure de l'écart entre le comportement moyen d'un ensemble de points uniformément distribués sur $[0,1)^t$ et le comportement de P_n. La q-valeur, une autre mesure d'uniformité définie à la section 4.4, exploite les mêmes idées sauf que celle-ci ne considère pas seulement les partitions qui divisent $[0,1)^t$ en hypercubes, mais aussi celles qui le divisent en hyper-rectangles de plusieurs formes et volumes.

Une autre mesure d'uniformité considérée dans cette thèse est $d^*(P_n)$, la plus courte distance entre deux points de P_n. Cette mesure est appelée *distance minimale.* Dans ce cas-ci, on mesure l'uniformité par le niveau « d'étalement » des points dans $[0,1)^t$. Ainsi, si les points de P_n sont près les uns des autres, ou si certains points sont concentrés dans une région, intuitivement, on affirmerait que P_n n'est pas uniforme. Par contre, si les points sont loin les uns des autres et qu'il ne semble pas y avoir de concentration de points dans aucune région, alors on considérerait P_n comme un ensemble de points uniforme. Plus $d^*(P_n)$ est grand, plus on considère P_n uniforme.

Dans ce chapitre, on passe en revue ces différentes mesures d'uniformité qui nous permettront de trouver de bons générateurs de nombres aléatoires, ainsi que de bons ensembles de points pour l'intégration quasi-Monte Carlo. Les mêmes outils sont utilisés pour mesurer l'uniformité dans les deux applications. Par contre, certains critères, dont la distance minimale et la q-valeur, sont très difficiles à calculer pour la plupart des générateurs de nombres aléatoires à cause du nombre élevé de points à considérer. Également, pour certains générateurs, comme les générateurs à congruence

linéaire ordinaires, leur structure se prête bien au calcul de la distance minimale. Les générateurs dont il est question dans cette thèse n'ont pas une structure qui permet de déterminer efficacement la distance minimale.

La contribution de ce chapitre est constituée de deux nouveaux algorithmes qui permettent de calculer la distance minimale. Ces nouveaux algorithmes sont particulièrement adaptés pour les générateurs qui entrent dans le cadre des équations (1.1)–(1.4). Un avantage de ces algorithmes est qu'ils peuvent détecter rapidement si la distance minimale est plus petite qu'un certain seuil. Ceci est utile lorsqu'on fait une recherche pour obtenir un ensemble de points avec une distance minimale élevée. Si la distance est jugée trop faible, alors on peut rapidement éliminer l'ensemble de points de notre recherche.

4.1 Définitions de base

Les critères d'uniformité introduits dans ce chapitre utilisent, pour la plupart, la notion de $\mathbf{p}$-équidissection en base b. Soit $\mathbf{p} = (p_1, \ldots, p_t)$, un vecteur d'entiers non nuls qui définit une partition de $[0,1)^t$ telle que l'axe j est divisé en b^{p_j} sous-intervalles, pour $j = 1, \ldots, t$. Pour un vecteur $\mathbf{p}$ et une base b donnés, ceux-ci définissent $b^{p_1+\cdots,p_t}$ hyper-rectangles, qu'on appelle *boîtes*, de même forme et volume. La partition définie par le vecteur $\mathbf{p}$ est appelée une $\mathbf{p}$-*équidissection* en base b.

Exemple 4.1. En base $b = 2$, si $t = 2$, le vecteur $\mathbf{p} = (p_1, p_2) = (1, 3)$ représente la partition qui sépare en 2^1 parties la première coordonnée et en 2^3 parties la deuxième coordonnée. Cette partition divise le carré $[0,1)^2$ en rectangles de dimension $2^{-1} \times 2^{-3}$. L'aire (ou le volume) de chacun des rectangles est 2^{-4}. La figure 4.1 illustre cette $(1,3)$-équidissection.

Définition 4.1. [40] L'ensemble de $n = b^k$ points $P_n = \{\mathbf{u}_0, \ldots, \mathbf{u}_{n-1}\} \subset [0,1)^t$ est dit $\mathbf{p}$-équidistribué si chaque boîte contient b^s points de P_n, où $s = k - p_1 - \ldots - p_t$.

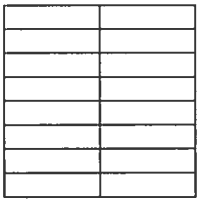


Figure 4.1 – La $(1,3)$-équidissection en base 2 de $[0,1)^2$.

Les ensembles de points $\Omega_{t,k}$ produits par un générateur qui entre dans le cadre des équations (1.1)–(1.4) peuvent être construits à l'aide de réseaux digitaux. Un réseau digital constitue une manière de construire un ensemble de points. Ainsi, pour les générateurs considérés, si on considère l'ensemble de tous les points $(u_n, \ldots, u_{n+t-1}) \in [0,1)^t$ à partir de toutes les initialisations possibles, alors on peut construire ce même ensemble de points à l'aide d'un réseau digital. Par contre, un réseau digital ne définit pas un générateur de nombres aléatoires puisque, avec ce premier, l'ordre dans lequel les points sont générés est sans importance et que son utilisation n'a de sens que si on utilise *tous* les points du réseau digital. Dans le cas du générateur de nombres aléatoires, on ne désire pas utiliser tous les points $(u_n, \ldots, u_{n+t-1})$ possibles dans une application de simulation. Malgré ces différences marquées, le réseau digital est utile pour mesurer l'uniformité d'ensembles de points produits par les générateurs de cette thèse.

Voici la définition de ce type de construction d'ensemble de points.

Définition 4.2. [74] Soit $C^{(0)}, \ldots, C^{(t-1)}$, des matrices $k \times k$, dont les éléments sont dans $\mathbb{F}_b$ et $\mathbf{C} = (C^{(0)} : \ldots : C^{(t-1)})$, une matrice $k \times kt$. De plus, soit l'ensemble de points

$$\Psi_t^b(\mathbf{C}, H, L) = \{(w_0, \ldots, w_{t-1}) : \mathbf{x} \in \mathbb{F}_b^k\}$$

où

$$\mathbf{y}_n = HC^{(n)}\mathbf{x} \in \mathbb{F}_b^L, \;\; 0 \leq n < t,$$

où H est une matrice de tempering $L \times k$ et

$$w_n = \sum_{i=1}^{L} y_n^{(i-1)} b^{-i}, \quad 0 \leq n < t.$$

L'ensemble de b^k points $\Psi_t^b(\mathbf{C}, H, L)$ est appelé *réseau digital* en base b.

Il est important de remarquer que dans la définition d'un réseau digital de [74], on considère différentes manières de produire les valeurs w_n à partir de $\mathbf{x}$, dont certaines ne sont pas des transformations linéaires. Dans cette thèse, nous allons nous restreindre à ce qui est inclus dans la définition 4.2

À partir de cette définition, pour $J = \{j_1, \ldots, j_s\}$, un ensemble d'indices tel que $0 \leq j_1 < \ldots < j_s < t$, on définit l'ensemble

$$\Psi_s^b(\mathbf{C}, H, L, J) = \{(w_{j_1}, \ldots, w_{j_s}) : \mathbf{x} \in \mathbb{F}_b^k\}.$$

L'ensemble $\Psi_s^b(\mathbf{C}, H, L, J)$ est aussi un réseau digital. Il s'agit de l'ensemble de points qui correspond aux projections des points de $\Psi_t^b(\mathbf{C}, H, L)$ sur les coordonnées définies par J.

En considérant les matrices $C^{(i)} = X^i$ pour $i = 0, \ldots, t$, on peut constater que l'ensemble $\Omega_{k,t}$ défini dans l'introduction est un réseau digital en base 2. Dans ce cas, $\Omega_{k,t} = \Psi_t^2(\mathbf{X}, H, L)$ où $\mathbf{X} = (I_k : X : X^2 : \ldots)$ est une matrice $k \times \infty$, $t = \infty$ et $H = YB$. Tous les réseaux digitaux considérés dans cette thèse sont également en base 2. C'est pourquoi on définit $\Psi_t(\mathbf{C}, H, L) = \Psi_t^2(\mathbf{C}, H, L)$ et $\Psi_s(\mathbf{C}, H, L, J) = \Psi_s^2(\mathbf{C}, H, L, J)$.

C'est dans l'optique que les ensembles de points $\Omega_{k,t}$ produits par les générateurs qui entrent dans le cadre des équations (1.1)–(1.4) sont des réseaux digitaux en base 2 que nous discutons des notions d'uniformité dans les prochaines sections. Notons que, pour le reste de cette thèse, nous abandonnons la notation $\Omega_{k,t}$ et la remplaçons par $\Psi_t(\mathbf{C}, H, L)$. Nous allons aussi considérer l'uniformité de réseaux digitaux $\Psi_s(\mathbf{C}, H, L, J)$ où J est un ensemble de s indices.

Remarque. Si on désire obtenir un ensemble de points dont la cardinalité n'est pas fixée à l'avance (ou n'est pas une puissance d'un nombre premier), comme pour les réseaux digitaux en base b, on peut utiliser une *suite digitale* en base b. Ce type de construction s'apparente aux réseaux digitaux sauf que, dans ce cas, les matrices $C^{(j)}$ sont de dimensions $L \times \infty$ et les points ont un ordre particulier qui est déterminé par l'expansion en base b des entiers. Soient $i = \sum_{j=0}^{\infty} a_{i,j} b^j$, $\mathbf{a}_i = (a_{i,0}, a_{i,1}, \ldots)$,

$$\mathbf{z}_i = (\mathbf{z}_{0,i}, \ldots, \mathbf{z}_{0,t-1}) = (C^{(0)}\mathbf{a}_i, \ldots, C^{(t-1)}\mathbf{a}_i),$$

et

$$\mathbf{w}_i = (w_{0,i}, \ldots, w_{t-1,i}) \in [0,1)^t$$

où

$$w_{p,i} = \sum_{j=0}^{L} z_{p,i}^{(j)} 2^{-j-1}.$$

L'ensemble de points contenant les n premiers points de la suite digitale en base b est

$$T_n = \{\mathbf{w}_i : 1 \leq i \leq n\}$$

Il est intéressant de remarquer que si $n = b^k$ pour une certaine valeur de k, alors l'ensemble de points T_n peut être construit par un réseau digital en base b. Les suites de Sobol [98], Faure [21], Niederreiter [73] et Niederreiter-Xing[78] sont des implantations concrètes de ce type de construction.

4.2 Équidistribution

Dans l'introduction, on explique brièvement ce qu'est le critère d'équidistribution. Comme mentionné, l'équidistribution est une mesure de l'uniformité de points dans $[0,1)^t$. Dans cette section, nous traitons plus en détails de cette mesure d'uniformité.

Soit $\Psi_s(\mathbf{C}, H, L, J)$ où $J = \{j_1, \ldots, j_s\}$, un réseau digital en base 2. Considérons, en base 2, l'équidissection $p_1 = \cdots = p_s = \ell$. La plus grande valeur de ℓ telle

que $\Psi_s(\mathbf{C}, H, L, J)$ est **p**-équidistribué est appelée la *résolution* en dimension s de $\Psi_s(\mathbf{C}, H, L, J)$ et est notée $\ell(J)$. Dans ce cas, on dit que $\Psi_s(\mathbf{C}, H, L, J)$ est $(s, \ell(J))$-*équidistribué*. Ceci n'est possible que si $k \geq s\ell(J)$, parce que la cardinalité de $\Psi_s(\mathbf{C}, H, L, J)$ est au plus 2^k. On a la borne supérieure

$$\ell(J) \leq \ell_s^* \stackrel{\text{def}}{=} \min(L, \lfloor k/s \rfloor). \tag{4.1}$$

À partir de cette borne, on définit *l'écart en dimension s* de la projection J par $\Lambda(J) \stackrel{\text{def}}{=} \ell_{|J|}^* - \ell(J)$.

Considérons maintenant le réseau digital $\Psi_t(\mathbf{C}, H, L)$. Dans ce cas, on note $\ell_t = \ell(S_t)$ et $\Lambda_t = \Lambda(S_t)$ où $S_t = \{0, \ldots, t-1\}$. L'ensemble $\Psi_t(\mathbf{C}, H, L)$ (et le générateur ou le réseau digital qui le produit) est dit *équidistribué au maximum* (*ME*), si ℓ_t atteint sa borne supérieure pour toutes les valeurs de t possibles, c'est-à-dire $\ell_t^* = \min(L, \lfloor k/t \rfloor)$ pour $t = 1, \ldots, k$. On peut définir des valeurs t_ℓ qui s'apparentent aux valeurs ℓ_t. On définit t_ℓ par la plus grande valeur de t pour laquelle $\Psi_t(\mathbf{C}, H, L)$ est (t, ℓ)-équidistribué pour une valeur de ℓ fixée. On a la borne supérieure

$$t_\ell \leq t_\ell^* \stackrel{\text{def}}{=} \min(d, \lfloor k/\ell \rfloor) \quad \text{pour } \ell = 1, \ldots, \min(k, L). \tag{4.2}$$

On définit *l'écart en résolution* ℓ par

$$\Delta_\ell \stackrel{\text{def}}{=} t_\ell^* - t_\ell.$$

On dit que le réseau digital (ou le générateur) a une *résolution maximale en dimension* t si $\Lambda_t = 0$. De plus, celui-ci est dit de *dimension maximale en résolution* ℓ, si $\Delta_\ell = 0$. Selon ces définitions, on peut dire que le réseau digital est équidistribué au maximum (ME), si $\Lambda_t = 0$ pour $t = 1, \ldots, \min(k, d)$ ou, de façon équivalente, si $\Delta_\ell = 0$ pour $\ell = 1, \ldots, \min(k, L)$.

Pour évaluer la qualité générale des ensembles de points $\Psi_t(\mathbf{X}, H, L)$, $1 \leq t \leq k$, produits par un générateur, on peut utiliser le critère

$$V = \sum_{\ell=1}^{L} \Delta_\ell. \tag{4.3}$$

Plus la valeur de V est petite, plus on considérera que le générateur donne de bons ensembles de points $\Psi_t(\mathbf{X}, H, L)$. Ce critère est utilisé aux chapitres 5, 7 et 8.

Dans les prochaines sections, il est question de différentes façons de calculer l'équidistribution.

4.2.1 Calcul de l'équidistribution par la méthode matricielle

Le calcul de l'équidistribution de $\Psi_s(\mathbf{C}, H, L, J)$ revient au calcul du rang de la matrice $T_{s,\ell}$ qui satisfait l'équation

$$\mathbf{s}_{s,\ell} = T_{s,\ell}\mathbf{x}_0 \tag{4.4}$$

où $\mathbf{s}_{s,\ell} = (\text{trunc}_\ell(\mathbf{y}_{j_1}), \ldots, \text{trunc}_\ell(\mathbf{y}_{j_s}))$ et la fonction $\text{trunc}_\ell(\mathbf{x})$ retourne le vecteur de ℓ bits $(x^{(0)}, \ldots, x^{(\ell-1)})$.

Proposition 4.2. *[33] L'ensemble de points $\Psi_s(\mathbf{C}, H, L, J)$ est (s, ℓ)-équidistribué si et seulement si la matrice $T_{s,\ell}$ est de plein rang $s\ell$.*

Dans [33], l'auteur explique comment obtenir la matrice $T_{s,\ell}$. La méthode utilisée pour calculer le rang de la matrice $T_{s,\ell}$ est l'élimination gaussienne.

4.2.2 Calcul de l'équidistribution avec le réseau en résolution

Les générateurs à récurrence linéaire modulo 2 qui entrent dans le cadre défini par les équations (1.1)–(1.4) définissent des réseaux polynômiaux en résolution

$$\mathcal{L}_\ell(BXB^{-1}) \subset \mathbb{L}_2^\ell$$

pour $\ell = 1, \ldots, L$ si les matrices X et B sont de plein rang et $Y = I_{L\times k}$. Grâce à ces réseaux, il est possible de calculer l'équidistribution d'un générateur seulement

pour $\Psi_s(\mathbf{X}, H, L)$. Pour calculer l'équidistribution du réseau digital $\Psi_s(\mathbf{X}, H, L, J)$ décrit par un générateur où $H = YB$, si le générateur n'est pas un générateur de Tausworthe, il faut se rabattre sur la méthode matricielle puisqu'aucune méthode de détermination de l'équidistribution des ensembles de points produits par des vecteurs de valeurs non successives utilisant le réseau en résolution n'est connue.

Les deux théorèmes suivants indiquent comment déterminer l'équidistribution de $\Psi_t(\mathbf{X}, H, L)$ à l'aide des réseaux $\mathcal{L}_\ell(BXB^{-1})$ ou des réseaux duaux $\mathcal{L}_\ell^*(BXB^{-1})$

Théorème 4.1. *[101]*
L'ensemble de points $\Psi_t(\mathbf{X}, B, L)$ est tel que $t_\ell = v_\ell$, où $v_\ell = \log_2 \sigma_\ell$ et σ_ℓ est le ℓ-ième minimum successif du réseau polynomial en $(\mathbb{F}_2, \ell)$-résolution, $\mathcal{L}_\ell(BXB^{-1})$.

Théorème 4.2. *[11]*
L'ensemble de points $\Psi_t(\mathbf{X}, H, L)$ est tel que $t_\ell = \log_2 \sigma_1$ où σ_1 est le premier minimum successif du dual du réseau polynomial en $(\mathbb{F}_2, \ell)$-résolution, $\mathcal{L}_\ell^*(BXB^{-1})$.

4.3 Ensemble de points sans collision

Soit $\mathbf{p}$, une équidissection de $[0,1)^t$ en base b. On dit qu'un ensemble de points $P_n \subset [0,1)^t$ est $\mathbf{p}$-*sans collision* si chacune des boîtes de l'équidissection contient au plus un point de P_n. Si on considère l'équidissection $p_1 = \cdots = p_t = \ell$ et que l'ensemble P_n est $\mathbf{p}$-sans collision, alors on dit que P_n est *sans collision en résolution ℓ en base b.*

Le théorème suivant permet de déterminer si un ensemble $\Psi_s(\mathbf{C}, H, L, J)$ est sans collision en résolution ℓ.

Théorème 4.3. *[33]*
Soit $\Psi_s^b(\mathbf{C}, H, L, J)$, un réseau digital en base b et $\mathbf{p}$, une équidissection en base b de $[0,1)^s$. Soit $T_\mathbf{p}$, une matrice $(p_1 + \cdots + p_t) \times k$ composée des p_1 premières lignes

de $C^{(j_1)}$, des p_2 premières lignes de $C^{(j_2)}$, ..., et des p_t premières lignes de $C^{(j_s)}$. Le réseau digital $\Psi_s(\mathbf{C}, H, L, J)$ est sans collision si et seulement si $T_{\mathbf{p}}$ est de rang k.

Pour que $\Psi_s(\mathbf{C}, H, L, J)$ soit sans collision, évidemment, il est nécessaire que $p_1 + \cdots + p_t \geq k$. Pour cette thèse, on n'utilise pas la notion d'ensemble de points sans collision directement afin de mesurer l'uniformité. Elle est plutôt utilisée dans les nouveaux algorithmes qui sont introduits plus loin dans ce chapitre. Ceux-ci utilisent l'information qu'un ensemble de points est sans collision en résolution ℓ afin de déterminer la plus courte distance entre toutes les paires de points de P_n.

4.4 Réseau digital et q-valeur

Dans cette section, on définit la notion de (q, k, t)-réseau et de la q-valeur qui est une généralisation de l'équidistribution. Voici une définition d'un (q, k, t)-réseau en base b.

Définition 4.3. [74, 94]
Soit $b \geq 2, t \geq 1$ et $0 \leq q \leq k$ des entiers. Un ensemble de points composé de b^k points de $[0,1)^t$ forme un (q, k, t)-réseau en base b si chaque sous-intervalle $J = \prod_{i=1}^{t} = [a_i b^{-p_i}, (a_i + 1)b^{-p_i})$ de $[0,1)^t$ avec les entiers $p_i \geq 0$ et $0 \leq a_i < b^{p_i}$ pour $1 \leq i \leq t$ et de volume b^{q-k} contient exactement b^q points de l'ensemble de points.

Avant de continuer, il est important pour le lecteur de faire la distinction entre un (q, k, t)-réseau et un réseau polynômial. Les notions sont totalement différentes. La confusion entre les noms n'est pas présente en anglais où un (q, k, t)-réseau est appelé « (q, k, t)-net » et un réseau polynômial est appelé « polynomial lattice » . Un réseau digital est appelé « digital net » et plus loin dans cette section, on verra qu'il existe des (q, k, t)-réseaux digitaux (« (q, k, t)-digital nets »).

Soit l'ensemble

$$\Phi_t(v) = \{\mathbf{p} = (p_1, \ldots, p_t) : p_1 + \cdots + p_t = v, p_i \geq 0, 1 \leq i \leq t\}.$$

Un élément de $\Phi(v)$ représente une équidissection (dans une certaine base b) de l'hypercube $[0,1)^t$, en (hyper-) rectangles de volume b^{-v}.

Un ensemble de points est un (q,k,t)-réseau si et seulement si pour chaque partition $(p_1, \ldots, p_t) \in \Phi_t(k-q)$ on retrouve exactement b^q points dans chaque rectangle. On observe que, pour $q < k$, un (q,k,t)-réseau est également un $(q+1,k,t)$-réseau. Puisque chaque partition $(p'_1, \ldots, p'_t)$ de $\Phi_t(k-q-1)$ est obtenue en prenant une partition de $(p_1, \ldots, p_t)$ de $\Phi_t(k-q)$ et en divisant b fois moins le long d'un axe, ceci fait en sorte que chaque rectangle de la partition $(p'_1, \ldots, p'_t)$ est composé de b rectangles de la partition $(p_1, \ldots, p_t)$. Étant donné que chaque rectangle de $(p_1, \ldots, p_t)$ contient b^q points, il s'ensuit que chaque rectangle de $(p'_1, \ldots, p'_t)$ contient b^{q+1} points. Ce qui démontre l'observation.

Voici maintenant la définition de la notion de q-valeur qui peut être utilisée pour quantifier l'uniformité d'un ensemble de points.

Définition 4.4. La plus petite valeur de q pour laquelle un ensemble de points est un (q,k,t)-réseau en base b est la *q-valeur en base b* de l'ensemble de points. Plus la q-valeur est petite, plus le (q,k,t)-réseau en base b est considéré uniforme.

À mesure que la valeur de $k-q$ augmente, le nombre d'équidissections contenues dans $\Phi_t(k-q)$ augmente, ce qui fait en sorte que déterminer si un ensemble de points est un (q,k,t)-réseau (ainsi que la q-valeur) devient de plus en plus complexe. Ainsi, déterminer la q-valeur d'un ensemble de points est plus difficile si celle-ci est petite.

Exemple 4.2. La figure 4.2 montre un ensemble de 4 points en 2 dimensions qui a une q-valeur en base 2 de 0. Pour déterminer celle-ci, il faut vérifier qu'il y ait toujours 2^q points dans chaque rectangle pour chaque équidissection de $\Phi_2(2) =$

$\{(0,2),(1,1),(2,0)\}$. Pour montrer que l'ensemble de points est aussi un $(1,2,2)$-réseau et un $(2,2,2)$-réseau, on illustre les équidissections contenues dans $\Phi_2(0)$ et $\Phi_2(1)$.

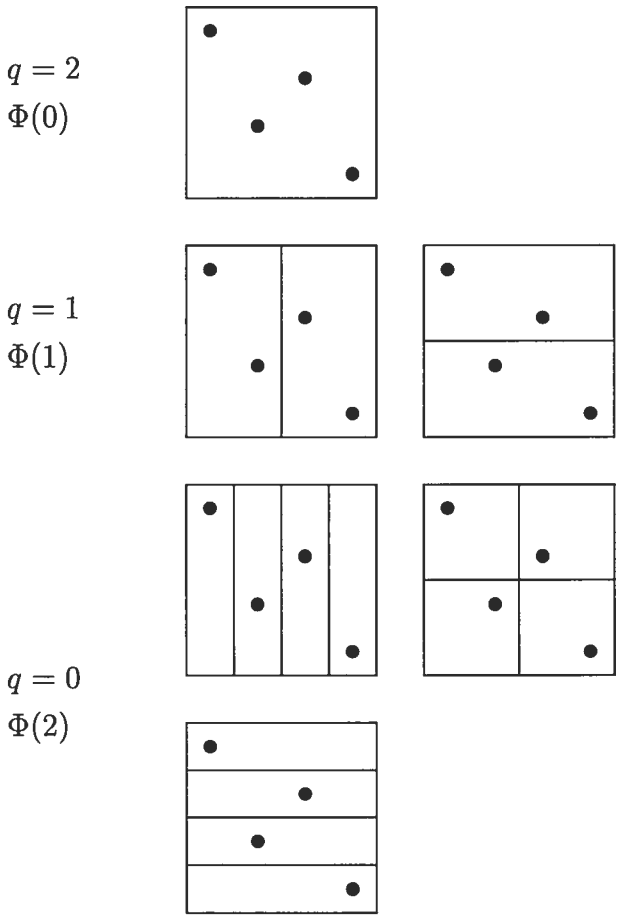


Figure 4.2 – Un $(0,2,2)$-réseau en base 2.

Pour faire un rapprochement entre la q-valeur en base 2 et l'équidistribution, on pourrait comparer la q-valeur à la valeur de ℓ_t. En fait, pour déterminer la valeur de ℓ_t, on considère seulement les partitions telles que $p_1 = \cdots = p_t = \ell$. On donne à ℓ_t la plus grande valeur de ℓ telle que chacun des petits cubes partitionnant $[0,1)^t$ contient exactement $2^{k-t\ell}$ points de l'ensemble de points. Il est facile de voir que le nombre de partitions considéré est beaucoup moins grand pour l'équidistribution que pour la

q-valeur. C'est pourquoi, on pourrait dire que la q-valeur est un critère d'uniformité beaucoup plus exigeant que l'équidistribution.

Les ensembles de points $\Psi_t^b(\mathbf{C}, H, L, J)$ sont des (q, k, t)-réseaux en base b. Au pire, leur valeur de q est k. En fait, un ensemble de points construit de la manière décrite par la définition 4.2 est appelé *(q, k, t)-réseau digital en base b*

Remarque. Pour les suites digitales en base b (voir la remarque de la section 4.1), pour évaluer leur qualité, on utilise le couple (q, t). Ainsi, une *(q, t)-suite digitale en base b* est une suite digitale (en base b) en t dimensions tel que, pour tout $r \geq 0$ et $k \geq t$, les points $\mathbf{w}_{rb^k+1}, \ldots, \mathbf{w}_{(r+1)b^k}$ forment un (q, k, t)-réseau digital en base b. Faure [21] donne une construction pour une $(0, b)$-suite en base b où b est premier et Niederreiter [73] montre qu'on peut aussi obtenir des $(0, p)$-suites en base $p = b^r$ où b est premier.

4.4.1 Calcul de la q-valeur

Avant d'être capable de calculer la q-valeur, il est utile d'introduire la définition suivante.

Définition 4.5. (Définition 1.3, [88]) Soit d, un entier tel que $0 \leq d \leq k$. Le système $\{\mathbf{c}_j^{(i)} \in \mathbb{F}_b^m : 1 \leq j \leq k, 1 \leq i \leq t\}$ de vecteurs est appelé un (d, k, t)-système sur $\mathbb{F}_b$ si, pour n'importe quels entiers non négatifs $p_1, \ldots, p_t$ avec $\sum_{i=1}^t p_i = d$, les vecteurs $\mathbf{c}_j^{(i)}$, $1 \leq j \leq p_i$, $1 \leq i \leq t$, sont linéairement indépendants sur $\mathbb{F}_b$. (L'ensemble vide est considéré comme linéairement indépendant.)

Soit

$$C^{(i)} = \begin{pmatrix} \mathbf{c}_1^{(i)} \\ \vdots \\ \mathbf{c}_k^{(i)} \end{pmatrix}$$

pour $i = 1, \ldots, k$ où les $\mathbf{c}_j^{(i)}$, $1 \leq j \leq k$, sont des vecteurs de dimension $1 \times k$. Le théorème suivant nous indique comment déterminer si un réseau digital est un (q, k, t)-réseau.

Proposition 4.3. *(Lemme 1.1, [88]) Les matrices $C^{(1)}, \ldots, C^{(k)}$ fournissent un $(k - d, k, t)$-réseau digital en base b si et seulement si le système $\{\mathbf{c}_j^{(i)} \in \mathbb{F}_b^m : 1 \leq j \leq k, 1 \leq i \leq t\}$ de leurs vecteurs lignes est un (d, k, t)-système sur $\mathbb{F}_b$.*

Dans [88], les auteurs donnent deux techniques qui permettent de déterminer la q-valeur d'un réseau digital en déterminant la plus grande valeur de d pour laquelle $\{\mathbf{c}_j^{(i)} \in \mathbb{F}_b^m : 1 \leq j \leq k, 1 \leq i \leq t\}$ est un (d, k, t)-système sur $\mathbb{F}_b$. Une est basée sur un code Gray et l'autre sur l'élimination gaussienne. Ils montrent que la méthode par le code Gray est plus rapide dans le cas où $b = 2$.

Dans [40], on utilise la notion de réseau gonflé et dégonflé pour déterminer la q-valeur. Une norme différente est utilisée pour mesurer les éléments du réseau. Mais, jusqu'à présent, cette théorie n'a donné aucune méthode efficace à cause de la difficulté de calculer le plus court vecteur avec la norme utilisée.

4.5 Distance minimale

Dans les sections qui suivent, on s'intéresse à la distance minimale de $P_n \subset [0, 1)^t$, un ensemble de n points. On définit la *distance minimale de P_n* par

$$d_p^*(P_n) = \min\{d_p(\mathbf{x}, \mathbf{y}) : \mathbf{x}, \mathbf{y} \in P_n, \mathbf{x} \neq \mathbf{y}\}$$

où $d_p(\mathbf{x}, \mathbf{y})$ est la distance entre les points $\mathbf{x}$ et $\mathbf{y}$ selon la norme L_p. On s'intéresse à cette distance minimale afin de définir des critères d'uniformité sur un ensemble de points. L'idée vient du fait que l'équidistribution ou la q-valeur ne dit pas tout sur l'uniformité des points de P_n. Pour un $(0, k, 2)$-réseau, deux points peuvent être

arbitrairement près l'un de l'autre. Par exemple, à la figure 4.3, on affirmerait intuitivement que l'ensemble de points de droite est plus uniforme que celui de gauche. Une plus grande distance entre les points nous pousse à cette affirmation. Pourtant, les deux ensembles de points on une q-valeur de 0, la meilleure possible pour ce critère.

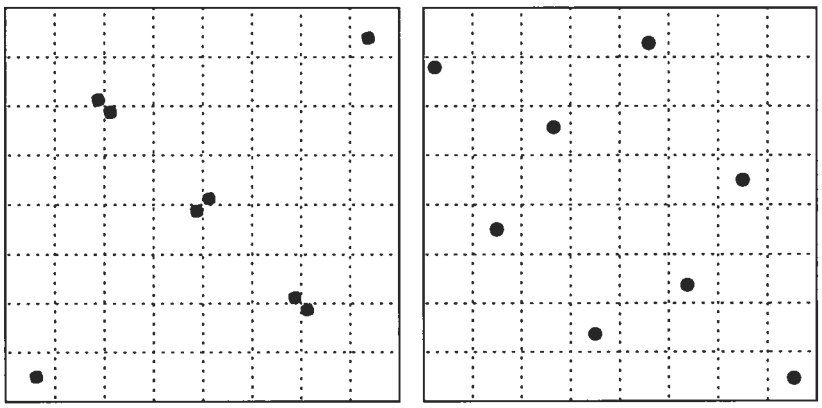


Figure 4.3 – Deux $(0, 3, 2)$-réseaux.

Dans les prochaines sections, nous décrivons l'algorithme de Khuller-Matias qui calcule la distance minimale d'un ensemble de points $P_n \subset [0, 1)^t$. Il constitue une base pour développer deux nouveaux algorithmes. Ces nouveaux algorithmes déterminent également la distance minimale. Tous ces algorithmes s'exécutent en temps moyen $O(n)$. Pour un ensemble donné, on considère t comme une constante, mais en réalité, le temps d'exécution de ces deux algorithmes est exponentiel en t. Pour nos applications (au chapitre 6), nous n'allons considérer que des ensembles de points en 2 ou 3 dimensions, ce qui fait que le temps d'exécution n'« explose » pas.

Les deux nouveaux algorithmes tirent à profit l'information sur la distribution des points donnée par le fait que l'ensemble de points soit sans collision pour certaines **p**-équidissections. À partir de ces nouveaux algorithmes, nous définissons des critères d'uniformité basés sur la distance minimale.

4.5.1 Algorithme de Khuller-Matias

Dans tout bon livre de géométrie computationnelle, par exemple [90] et [97], on retrouve des algorithmes déterministes qui permettent de calculer $d_p^*(P_n)$ en temps $O(n \log n)$ et des algorithmes probabilistes qui permettent de le faire en temps $O(n)$ en moyenne. Parmi les algorithmes probabilistes, il y a celui rapporté dans [23]. On y décrit un algorithme qui permet de trouver $d_2^*(P_n)$ d'un ensemble de points $P_n \subset [0,1)^2$ et qui s'exécute en temps $O(n)$ en moyenne. On peut généraliser certains résultats présentés dans [23] pour le cas où $P_n \subset [0,1)^t$. C'est cette généralisation que l'on présente dans cette section.

Les normes utilisées dans cette thèses sont la norme euclidienne (L_2), qui est définie par

$$d_2(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{s=1}^{t}(x^{(s)} - y^{(s)})^2}$$

et la norme L_∞, qui est définie par

$$d_\infty(\mathbf{x},\mathbf{y}) = \max_{1\le s\le t} |x^{(s)} - y^{(s)}|.$$

On peut aussi définir, pour un point $\mathbf{x} \in P_n$ et une norme donnés, la distance de $\mathbf{x}$ au point le plus proche par

$$d_p(\mathbf{x}) = \min\{d_p(\mathbf{x},\mathbf{y}) : \mathbf{y} \in P_n, \mathbf{y} \neq \mathbf{x}\}.$$

L'algorithme que nous allons maintenant décrire utilise des *grillages* de l'hypercube unitaire $[0,1)^t$. Un grillage $\tilde{G}_b$ est une partition de $[0,1)^t$ où chacune des parties est cubique et de côté b. Les côtés des cubes sont alignés avec les axes et l'origine se trouve dans le coin d'un cube. Bien sûr, si b ne divise pas 1, alors certains cubes se trouveront à chevaucher la frontière de $[0,1)^t$. Dans ce cas, le grillage $\tilde{G}_b$ ne correspond pas à une **p**-équidissection. Dans l'autre cas, le grillage correspond bien à une **p**-équidissection en base $1/b$ telle que $p_1 = \cdots = p_t = 1$. Dans ce qui suit, on appellera

chacun des cubes *cellule*. On représente chacune des cellules par un intervalle

$$I(a_1, \ldots, a_t) = \prod_{s=1}^{t} b[a_s, a_s + 1), \quad 0 \leq a_1, \ldots, a_t < \lceil 1/b \rceil$$

où les a_i, $1 \leq i \leq t$, sont des entiers. Remarquons que l'intervalle est fermé à gauche et ouvert à droite pour chacune des dimensions.

On dit que deux cellules sont *adjacentes* si elles ont au moins un coin en commun. On définit le *voisinage* d'une cellule C par l'ensemble de toutes les cellules adjacentes à C. On le note par $\tilde{V}_b(C)$.

Soit $P_n \subset [0,1)^t$, un ensemble de n points et $\tilde{G}_b$, un grillage de $[0,1)^t$. On note par $\tilde{C}_b(\mathbf{x})$ la cellule $I(a_1, \ldots, a_t)$ telle que $\mathbf{x} \in I(a_1, \ldots, a_t)$. On définit le voisinage de $\mathbf{x}$ par l'ensemble des points $\mathbf{x}$ qui sont contenus dans les cellules voisines de $\tilde{C}_b(\mathbf{x})$. Celui-ci est

$$\tilde{N}_b(\mathbf{x}) = P_n \cap \tilde{V}_b(\tilde{C}_b(\mathbf{x}))$$

Exemple 4.3. À la figure 4.4, on divise $[0,1)^2$ grâce au grillage $\tilde{G}_{\frac{1}{6}}$. Il y a 36 cellules, soit $I(a_1, a_2)$, $0 \leq a_i < 6$, $i = 1, 2$. Le voisinage de la cellule $I(1,4)$ est illustré par la surface hachurée et le voisinage d'un point $\mathbf{x}$ est illustré par la zone grise. Le grillage $\tilde{G}_{\frac{1}{6}}$ est une $(1,1)$-équidissection en base 6.

Soit $\tilde{N}_b(\mathbf{x})$, le voisinage de $\mathbf{x}$ pour le grillage $\tilde{G}_b$. Les propriétés suivantes, énoncées pour $t = 2$ dans [23], s'appliquent également dans le cas où $t > 2$:

Propriété 4.1. *Tous les points* $\mathbf{y}$ *dont* $d_2(\mathbf{x}, \mathbf{y}) \geq 2b\sqrt{t}$ *ne sont pas dans* $\tilde{N}_b(\mathbf{x})$.

Propriété 4.2. *Tous les points* $\mathbf{y}$ *dont* $d_2(\mathbf{x}, \mathbf{y}) \leq b$ *sont dans* $\tilde{N}_b(\mathbf{x})$.

En émettant l'hypothèse que $b/(2\sqrt{t}) \leq d_2^*(P_n) \leq b$, alors on a les propriétés additionnelles suivantes :

Propriété 4.3. *Pour chaque point* $\mathbf{x} \in P_n$, $\tilde{N}_b(\mathbf{x})$ *contient un nombre de points plus petit qu'une constante qui dépend de* t.

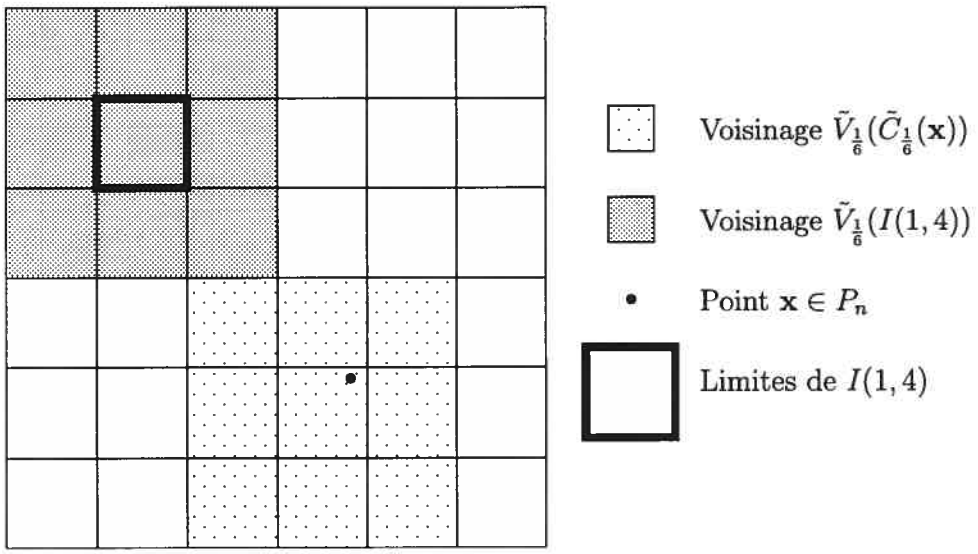


Figure 4.4 – Grillage $\tilde{G}_{\frac{1}{6}}$.

Propriété 4.4. *Pour chaque paire* $\mathbf{x}, \mathbf{y} \in P_n$, *si* $d_2(\mathbf{x}, \mathbf{y}) = d_2^*(P_n)$, *alors* $\mathbf{y} \in \tilde{N}_b(\mathbf{x})$.

L'algorithme décrit dans [23], utilise ces propriétés et des grillages $\tilde{G}_b$ successifs de plus en plus fins afin de « filtrer » l'ensemble des points dans P_n. Ce processus de filtrage permet d'éliminer les points qui sont loin de leur plus proche voisin afin d'obtenir une approximation de $d_2^*(P_n)$ telle que $b/(2\sqrt{t}) \leq d_2^*(P_n) \leq b$. Voici en détail l'algorithme.

Algorithme 4.1. (Algorithme de Khuller-Matias [23])

1. $S_1 \leftarrow P_n$ et $i \leftarrow 0$.
2. Tant que $S_{i+1} \neq \emptyset$ faire

   $i \leftarrow i + 1$

   Choisir un point $\mathbf{x}_i$ de S_i au hasard.

   Construire un grillage avec $\tilde{G}_b$, $b = d_2(\mathbf{x}_i)/(2\sqrt{t})$.

   $X_i \leftarrow \{\mathbf{x} \in S_i | \tilde{N}_b(\mathbf{x}) = \{\mathbf{x}\}\}$.

   $S_{i+1} \leftarrow S_i - X_i$.

3. $i^* \leftarrow i$. (Invariant : $S_{i^*} = \emptyset$ et $d_2(\mathbf{x}_{i^*})/(2\sqrt{t}) \leq d_2^*(P_n) \leq d_2(\mathbf{x}_{i^*})$).
4. Construire un grillage de grosseur $d_2(\mathbf{x}_{i^*})$. Pour chaque point $\mathbf{x} \in P_n$, calculer la distance au point le plus près de $\mathbf{x}$ dans $\tilde{N}_b(\mathbf{x})$ (si $\tilde{N}_b(\mathbf{x}) \neq \emptyset$). La distance $d_2^*(P_n)$ est la plus petite des distances calculées.

Dans un premier temps, on initialise S_1 à P_n. On choisit au hasard un point $\mathbf{x}_1$ de S_1 et calcule $d_2(\mathbf{x}_1)$, ce qui se fait en temps $O(n)$. A la dernière étape de la boucle, on élimine de S_i, grâce à la propriété 4.1, tous les points $\mathbf{x}$ dont $d_2(\mathbf{x}) > 2b\sqrt{t} = d_2(\mathbf{x}_i)$ pour obtenir S_{i+1}. Si $S_{i+1} = \emptyset$, on sait que $d_2^*(P_n) \geq b = d_2(\mathbf{x}_i)/(2\sqrt{t})$, puisque $S_i \neq \emptyset$ et qu'à la dernière étape de la boucle on aurait éliminé les paires de points $\mathbf{x}, \mathbf{y}$ telles que $d_2^*(P_n) = d_2(\mathbf{x}, \mathbf{y})$. On sait également que $d_2^*(P_n) \leq d_2(\mathbf{x}_{i^*})$ par définition de $d_2^*(P_n)$. Ceci implique qu'au début de l'étape 4, on sait que $d_2(\mathbf{x}_{i^*})/2\sqrt{t} \leq d_2^*(P_n) \leq d_2(\mathbf{x}_{i^*})$. Par la propriété 4.4, on sait que l'étape 4 va effectivement calculer la distance $d_2^*(P_n)$.

Afin de vérifier le voisinage d'un point, on peut utiliser une table de hachage. On numérote les cellules de façon à ce que chaque cellule soit identifiée par un numéro unique. On emmagasine tous les points de l'ensemble de points à considérer, soit S_i, dans une table de hachage. La fonction de hachage utilise le numéro de cellule dans laquelle le point se trouve. Pour vérifier le voisinage d'un point, on n'a qu'à examiner dans la table de hachage s'il y a des points dans les cases voisines. En utilisant la technique du hachage parfait, il est possible de faire cela en temps $O(|S_i|)$ [23]. Noter que, pour chaque point, le nombre de cellules voisines à vérifier est 3^t.

Dans [23], on explique que le processus de filtrage, soit les opérations contenues dans la boucle, s'effectue en temps moyen linéaire. En choisissant $\mathbf{x}_i$ au hasard, la moitié des points $\mathbf{x} \in S_i$ sont tels que $d_2(\mathbf{x}) > d_2(\mathbf{x}_i)$ en moyenne, quelque soit l'ensemble P_n et la dimension t. Ceci implique qu'à chaque itération on élimine la moitié des points en moyenne (car, à chaque itération, on élimine les points $\mathbf{x}$ tels que $d_2(\mathbf{x}) > d_2(\mathbf{x}_i)$). Soit $s_i = |S_i|$ à l'avant dernière étape de la boucle. Le nombre

espéré de fois qu'il faut vérifier le voisinage de points pendant tout l'algorithme est donné par

$$\mathrm{E}\left[\sum_{i=1}^{i^*} s_i\right] \leq \mathrm{E}\left[\sum_{i=1}^{n} s_1/2^{i-1}\right] = \sum_{i=1}^{n} n/2^{i-1} \leq 2n \sum_{i=1}^{\infty} 1/2^i = 2n.$$

En tout, on vérifie, en moyenne, au plus $2n$ voisinages de points. Puisque chaque cellule a, au plus, 3^t voisins, les étapes 1 à 4, toutes itérations confondues, se font dans un temps dans $O(n)$ en moyenne. À noter que la moyenne est faite sur l'ensemble de toutes les décisions possibles pendant l'exécution de l'algorithme et non sur l'ensemble de tous les ensembles de points P_n possibles. L'étape 5, à cause de la propriété 4.3, se fait aussi dans un temps $O(n)$. On vient de montrer que l'algorithme de Khuller-Matias s'exécute, en moyenne, en $O(n)$ opérations.

4.5.2 Propriétés de $d_p^*(P_n)$ quand $b = 2^v$

Dans cette section, nous étudions l'information supplémentaire qu'on peut déduire sur $d_2^*(P_n)$ ou $d_\infty^*(P_n)$ quand on sait que P_n est sans collision pour certaines résolutions v. Cette information permet de modifier l'algorithme de Khuller-Matias pour obtenir deux nouveaux algorithmes qui sont particulièrement bien adaptés au cas où P_n est un réseau digital en base 2. Pour les réseaux digitaux, déterminer si un ensemble de points est sans collision est très rapide par le théorème 4.3. Nous reviendrons sur cet aspect à la section 4.5.6. Ces deux algorithmes sont présentés aux sections 4.5.3 et 4.5.5

Pour ces algorithmes, on ne considère que des grillages où b est de la forme 2^{-v} et v est un entier positif. À noter que ce grillage constitue une **p**-équidissection en base 2 où $p_1 = \ldots = p_t = v$.

Une première observation sur ces grillages est que $1/b = 2^v$ est toujours un entier, ce qui veut dire qu'aucune cellule ne se trouve à chevaucher la frontière de l'hypercube.

Ceci est un avantage si on veut utiliser un algorithme semblable à l'algorithme de Khuller-Matias pour calculer la distance entre des points sur le tore défini sur $[0,1)^t$. Dans le tore, en considérant $[0,1)^t$ comme un hypercube, les paires de points qui sont près de faces opposés se retrouvent près l'un de l'autre. Dans ce cas particulier, le voisinage est bien défini puisque qu'aucune cellule ne chevauche la « frontière » du tore. Dans ce qui suit, on ne considère que les distances sur le tore $[0,1)^t$. Aussi, lorsqu'il est question de l'hypercube unitaire $[0,1)^t$, on fait implicitement référence au tore $[0,1)^t$. L'algorithme de Khuller-Matias, tel que présenté, ne permet pas de calculer les distances sur le tore puisque $1/b$ n'est pas nécessairement un entier.

Supposons que l'on divise l'hypercube unitaire en t dimensions $[0,1)^t$ en 2^v parties égales le long de chacun des axes pour obtenir le grillage $G_v = \tilde{G}_{2^{-v}}$. Ceci implique que l'on divise l'hypercube unitaire en 2^{vt} petits hypercubes identiques de volume 2^{-vt}. Appelons le grillage G_v *grillage de résolution v en t dimensions.* Les cellules dans ce cas sont

$$I(a_1,\ldots,a_t) = \prod_{s=1}^{t} 2^{-v}[a_s, a_s+1) \;\; 0 \le a_1,\ldots,a_t < 2^v$$

Pour un grillage G_v, la cellule qui contient le point $\mathbf{x}$ est notée $C_v(\mathbf{x}) = \tilde{C}_{2^{-v}}(\mathbf{x})$. Soit $P_n \subset [0,1)^t$, un ensemble fini de n points dans l'hypercube unitaire $[0,1)^t$. On dit qu'un ensemble de points P_n est *sans collision en résolution v* si chacun des points $\mathbf{x} \in P_n$ se retrouve seul dans sa cellule pour un grillage de résolution v. On définit *le voisinage d'un point $\mathbf{x} \in P_n$ en résolution v* par $N_v(\mathbf{x}) = \tilde{N}_{2^{-v}}(\mathbf{x})$ et le *voisinage en résolution v d'une cellule C* par $V_v(C) = \tilde{V}_{2^{-v}}(C)$. Pour simplifier la notation, on définit $V_v(\mathbf{x}) = V_v(C_v(\mathbf{x}))$. Un ensemble de points P_n est *sans voisin en résolution v* si $N_v(\mathbf{x}) = \{\mathbf{x}\}$ pour tout $\mathbf{x} \in P_n$.

Voici une série de propriétés, avec leurs preuves, concernant ces notions sur le voisinage. Ces propriétés seront utiles dans les prochaines sections pour justifier l'exactitude des deux nouveaux algorithmes.

Propriété 4.5. $V_{v+1}(\mathbf{x}) \subset V_v(\mathbf{x})$

Démonstration. Soit

$$\mathbf{x} = (x^{(0)}, \ldots, x^{(t-1)})$$

où

$$x^{(j)} = \sum_{i=0}^{\infty} a_{j,i} 2^{-i-1}$$

où $a_{j,i} \in \{0,1\}$, pour $j = 0, \ldots, t-1$. Soit $s(\mathbf{x}, j, v) = \sum_{i=0}^{v-1} a_{j,i} 2^{-i-1}$. On a

$$V_v(\mathbf{x}) = \{\mathbf{y} \in [0,1)^t : s(\mathbf{x}, j, v) - 2^{-v} \leq y^{(j)} < s(\mathbf{x}, j, v) + 2^{-v+1} \text{ pour } j = 0, \ldots, t-1\}.$$

Si

$$s(\mathbf{x}, j, v) - 2^{-v} < s(\mathbf{x}, j, v+1) - 2^{-v-1} \text{ pour } j = 0, \ldots, t-1$$

et

$$s(\mathbf{x}, j, v) + 2^{-v+1} > s(\mathbf{x}, j, v+1) + 2^{-v} \text{ pour } j = 0, \ldots, t-1,$$

alors nécessairement $V_{v+1}(\mathbf{x}) \subset V_v(\mathbf{x})$. La première inégalité est équivalente à $-1 < a_{j,v}$ pour $j = 0, \ldots, t-1$ et la deuxième à $2 > a_{j,v}$. Les deux inégalités sont toujours vraies, donc $V_{v+1}(\mathbf{x}) \subset V_v(\mathbf{x})$. □

Propriété 4.6. *Si un ensemble de points P_n est sans collision en résolution v, alors il le sera également en résolution $v+1$.*

Démonstration. Soit K_v, l'ensemble de toutes les cellules en résolution v. L'ensemble P_n est sans collision en résolution v, ce qui indique qu'il y a au plus un point par cellule en résolution v. Puisque chaque cellule de K_v est partionnée en 2^t cellules pour obtenir K_{v+1}, ceci implique qu'en résolution $v+1$, chaque cellule a au plus un point et que P_n est sans collision. □

Propriété 4.7. *Un ensemble de points P_n qui est sans voisin en résolution v est nécessairement sans collision en résolution v.*

Démonstration. Si $N_v(\mathbf{x}) = \{\mathbf{x}\}$ pour tout $\mathbf{x} \in P_n$, ceci implique que tous les points $\mathbf{x}$ sont seuls dans leurs cellules. □

Propriété 4.8. *Si P_n est sans voisin en résolution v, alors il en sera de même en résolution $v+1$.*

Démonstration. Supposons que $P_n \subset [0,1)^t$ soit sans voisin en résolution v. Soit $\mathbf{x}$, un point arbitraire de P_n. Soit $V_v(\mathbf{x})$, le voisinage de $\mathbf{x}$ en résolution v. On sait que $N_v(\mathbf{x}) = \{\mathbf{x}\}$. Puisque $V_{v+1}(\mathbf{x}) \subset V_v(\mathbf{x})$, ceci implique que le nombre de points dans $N_{v+1}(\mathbf{x})$ ne peut pas augmenter et qu'en fait $N_{v+1}(\mathbf{x}) = N_v(\mathbf{x}) = \{\mathbf{x}\}$. Puisque le choix de $\mathbf{x}$ a été arbitraire, ceci implique que $N_{v+1}(\mathbf{x}) = N_v(\mathbf{x}) = \{\mathbf{x}\}$ pour tout $\mathbf{x} \in P_n$. □

Propriété 4.9. *Si P_n n'est pas sans collision en résolution v, alors*

$$d_2^*(P_n) < 2^{-v}\sqrt{t} \text{ et } d_\infty^*(P_n) < 2^{-v}.$$

Démonstration. Si P_n n'est pas sans collision en résolution v, ceci indique qu'il existe une paire de points $\mathbf{x}, \mathbf{y} \in P_n$ qui sont contenus à l'intérieur d'une même cellule. La plus grande distance euclidienne possible entre deux points à l'intérieur d'une même cellule est la longueur de la diagonale de la cellule, soit $2^{-v}\sqrt{t}$. Nécessairement $d_2(\mathbf{x},\mathbf{y}) < 2^{-v}\sqrt{t}$, ce qui implique que $d_2^*(P_n) < 2^{-v}\sqrt{t}$ par la définition de $d_2^*(P_n)$. De plus, selon la norme L_∞, la plus grande distance possible entre deux points $\mathbf{x}, \mathbf{y} \in P_n$ à l'intérieur d'une même cellule est la longueur d'un côté d'une cellule, soit 2^{-v}. Ainsi, $d_\infty(\mathbf{x},\mathbf{y}) < 2^{-v}$ et $d_\infty^*(P_n) < 2^{-v}$. □

Propriété 4.10. *Si P_n est sans voisin en résolution v, alors*

$$d_2^*(P_n) > 2^{-v} \text{ et } d_\infty^*(P_n) > 2^{-v}.$$

Démonstration. Supposons que P_n soit sans voisin en résolution v. Pour $p = 2$ et $p = \infty$, soit $\mathbf{x}_p^*$ et $\mathbf{y}_p^*$, une paire de points de P_n telle que $d_p(\mathbf{x}^*, \mathbf{y}^*) = d_p^*(P_n)$. Pour

$p = 2$ et $p = \infty$, le fait que P_n soit sans voisin en résolution v indique que le point le plus proche de $\mathbf{x}^*$, soit $\mathbf{y}^*$, se trouve à l'extérieur de $V_v(\mathbf{x}^*)$. Également, pour $p = 2$ et $p = \infty$, la plus courte distance possible entre $\mathbf{x}^*$ et $\mathbf{y}^*$ est donnée par la largeur d'une cellule, soit 2^{-v}. Ceci veut dire que $d_2(\mathbf{x}^*, \mathbf{y}^*) > 2^{-v}$ et $d_\infty(\mathbf{x}^*, \mathbf{y}^*) > 2^{-v}$. □

Propriété 4.11. *Si P_n n'est pas sans voisin en résolution v, alors*

$$d_2^*(P_n) < 2^{-v+1}\sqrt{t} \text{ et } d_\infty^*(P_n) < 2^{-v+1}.$$

Démonstration. Si P_n n'est pas sans voisin en résolution v, alors il existe deux points $\mathbf{x}, \mathbf{y} \in P_n$ tels que $\mathbf{y} \in N_v(\mathbf{x})$. La plus grande valeur possible de $d_2(\mathbf{x}, \mathbf{y})$ est deux fois la longueur de la diagonale d'une cellule, soit $2^{-v}\sqrt{t} \times 2$. Donc $d_2(\mathbf{x}, \mathbf{y}) < 2^{-v+1}\sqrt{t}$ et $d_2^*(P_n) < 2^{-v+1}\sqrt{t}$ par la définition de $d_2^*(P_n)$. Pour la norme L_∞, la plus grande valeur possible de $d_\infty(\mathbf{x}, \mathbf{y})$ est deux fois la longueur d'un côté d'une cellule, soit $2^{-v} \times 2$. Donc $d_\infty(\mathbf{x}, \mathbf{y}) < 2^{-v+1}$ et $d_\infty^*(P_n) < 2^{-v+1}$. □

Propriété 4.12. *Si P_n n'est pas sans collision en résolution $v-1$ et est sans collision en résolution v, alors celui-ci n'est pas sans voisin en résolution v.*

Démonstration. En résolution $v-1$, puisque P_n n'est pas sans collision, par définition, ceci indique qu'il existe une cellule C qui contient plus d'un point. Soit M, l'ensemble des points contenus dans la cellule C. En résolution v, la cellule C est divisé en 2^t cellules. Les points de M sont distribués de manière que chacune des 2^t cellules reçoive au plus un point puisque P_n est sans collision en résolution v. Mais chacune des 2^t cellules sont adjacentes les unes aux autres, ce qui implique que P_n n'est pas sans voisin en résolution v. □

Propriété 4.13. *Si P_n est sans voisin en résolution v, mais ne l'est pas en résolution $v-1$, alors*

$$2^{-v} < d_2^*(P_n) < 2^{-v+2}\sqrt{t} \text{ et } 2^{-v} < d_\infty^*(P_n) < 2^{-v+2}.$$

Démonstration. Ceci est déduit directement des propositions 4.10 et 4.11. □

Propriété 4.14. *Soit* $c_{t,2} = \min\{c \in \mathbb{Z} : 2^c \geq 2\sqrt{t}\}$ *et* $c_{t,\infty} = 1$. *Pour* $p = 2$ *et* $p = \infty$, *soient* $\mathbf{x}_p^*, \mathbf{y}_p^* \in P_n$ *tels* $d_p(\mathbf{x}_p^*, \mathbf{y}_p^*) = d_p^*(P_n)$. *Soit* v_0, *la plus petite résolution à laquelle* P_n *est sans collision. Pour* $p = 2$ *et* $p = \infty$, *on a* $\mathbf{x}_p^* \in N_{v_0 - c_{t,p}}(\mathbf{y}_p^*)$.

Démonstration. Puisque P_n n'est pas sans collision en résolution $v_0 - 1$, alors $d_2^*(P_n) < 2^{-v_0+1}\sqrt{t}$ et $d_\infty^*(P_n) < 2^{-v_0+1}$ par la propriété 4.9. En choisissant un grillage de résolution $v_0 - c_{t,p}$, chaque cellule est telle que la longueur de ses côtés est $2^{-v_0+c_{t,p}}$. Pour que $\mathbf{y}_p^*$ ne soit pas dans $N_{v_0-c_{t,p}}(\mathbf{x}_p^*)$, il est nécessaire que $d_p(\mathbf{x}_p^*, \mathbf{y}_p^*) > 2^{-v_0+c_{t,p}}$. Pour $p = 2$, ceci est impossible car $d_2^*(P_n) < 2^{-v_0+1}\sqrt{t}$ et $2^{-v_0+1}\sqrt{t} \leq 2^{-v_0+c_{t,2}}$, ce qui montre que $\mathbf{x}^* \in N_{v_0-c_{t,2}}(\mathbf{y}^*)$. Pour $p = \infty$, ceci est aussi impossible car $d_\infty^*(P_n) < 2^{-v_0+1}$ et $2^{-v_0+1} \leq 2^{-v_0+c_{t,\infty}} = 2^{-v_0+1}$, ce qui montre que $\mathbf{x}^* \in N_{v_0-c_{t,\infty}}(\mathbf{y}^*)$. □

Propriété 4.15. *Soit* $c_{t,2} = \min\{c \in \mathbb{Z} : 2^c \geq 2\sqrt{t}\}$ *et* $c_{t,\infty} = 1$. *Pour* $p = 2$ *et* $p = \infty$, *soient* $\mathbf{x}_p^*, \mathbf{y}_p^* \in P_n$ *tels* $d_p(\mathbf{x}_p^*, \mathbf{y}_p^*) = d_p^*(P_n)$. *Pour* $p = 2$ *et* $p = \infty$, *si* P_n *est tel qu'il est sans voisin en résolution* v, *mais n'est pas sans voisin en résolution* $v - 1$, *alors* $\mathbf{x}_p^* \in N_{v-c_{t,p}-1}(\mathbf{y}_p^*)$.

Démonstration. Pour une résolution $\bar{v}$ et une norme L_p, pour être absolument certain que $\mathbf{x}_p^* \in N_{\bar{v}}(\mathbf{y}_p^*)$, il faut que $d_p^*(P_n) < 2^{-\bar{v}}$. On sait, par la proposition 4.13 que $2^{-v} < d_2^*(P_n) < 2^{-v+2}\sqrt{t}$ et $2^{-v} < d_\infty^*(P_n) < 2^{-v+2}$. Pour $p = 2$, si $2^{-v+2}\sqrt{t} \leq 2^{-\bar{v}}$, alors on a $\mathbf{x}_2^* \in N_{\bar{v}}(\mathbf{y}_2^*)$. Si on pose $\bar{v} = v - c_{t,2} - 1$, alors la condition est respectée, ce qui montre que $\mathbf{x}_2^* \in N_{v-c_{t,2}-1}(\mathbf{y}_2^*)$. Pour $p = \infty$, si $2^{-v+2} \leq 2^{-\bar{v}}$, alors on a que $\mathbf{x}_\infty^* \in N_{\bar{v}}(\mathbf{y}_\infty^*)$. Si on pose $\bar{v} = v - c_{t,\infty} - 1$, alors la condition est respectée, ce qui montre que $\mathbf{x}_\infty^* \in N_{v-c_{t,2}-1}(\mathbf{y}_\infty^*)$. □

4.5.3 Premier nouvel algorithme

Le nouvel algorithme défini dans cette section permet de calculer $d_2^*(P_n)$ ou $d_\infty^*(P_n)$ en temps moyen $O(n)$. Celui-ci utilise v_0, la plus petite résolution pour laquelle P_n

est sans collision, comme information lui permettant de restreindre le nombre de distances, entre paires de points, à calculer. Pour cet algorithme, on définit $c_{t,2} = \min\{c \in \mathbb{Z} : 2^c \geq 2\sqrt{t}\}$ et $c_{t,\infty} = 1$. Voici ce nouvel algorithme qui permet de déterminer $d_2^*(P_n)$ ou $d_\infty^*(P_n)$.

Algorithme 4.2. Calcul de $d_p^*(P_n)$.

1. Calculer v_0, la plus petite résolution pour laquelle l'ensemble de points P_n est sans collision.
2. $v \leftarrow v_0 - c_{t,p}$.
3. Pour chaque point $\mathbf{x} \in P_n$, calculer $d_p(\mathbf{x}) = \min\{d_p(\mathbf{x}, \mathbf{y}) : \mathbf{y} \in N_v(\mathbf{x}), \mathbf{y} \neq \mathbf{x}\}$.
4. Calculer $d_p^*(P_n) = \min\{d_p(\mathbf{x}) : \mathbf{x} \in P_n\}$.

La justesse de l'algorithme est démontrable par la proposition 4.14. Soit $\mathbf{x}_p$ et $\mathbf{y}_p$, les points de P_n tels que $d_p^*(P_n) = d_p(\mathbf{x}_p, \mathbf{y}_p)$. La résolution $v_0 - c_{t,p}$ garantit que $\mathbf{x}_p^* \in N_{v_0 - c_t}(\mathbf{y}_p^*)$ par la proposition 4.14 et que l'algorithme calcule vraiment la quantité désirée, soit $d_p^*(P_n)$. On remarque que, pour les dimensions $t = 2, 3, 4$, $c_{t,2} = 2$.

En dimension t, une cellule a 3^t cellules adjacentes. Puisqu'on sait qu'en résolution v l'ensemble de points P_n est sans collision, en résolution $v_0 - c_{t,p}$ chaque cellule contiendra au plus $2^{tc_{t,p}}$ points. Dans le pire des cas, le nombre de points à considérer (et le nombre de distances à calculer) pour le voisinage d'un point est $3^t 2^{tc_{t,p}}$. Cette valeur peut être très grande, même avec $t = 2$. Dans ce cas, $3^2 2^{2\times 2} = 144$. En supposant que l'étape 1 se fait en $O(g(n))$ opérations, l'algorithme 4.2 s'exécute en $O(g(n) + n)$ opérations. Si $g(n) \in O(\log^4(n))$ (c'est le cas, tel qu'expliqué à la section 4.5.6, si P_n est un réseau digital en base 2), alors l'algorithme s'exécute en $O(n)$ opérations.

4.5.4 Voisinage

Le fait d'avoir $3^t 2^{tc_{t,p}}$ distances à calculer en pire cas, pour chaque point dans le précédent algorithme, peut le rendre peu intéressant. Par contre, celui-ci constitue une base pour un nouvel algorithme. En effet, il est possible de le modifier afin de définir un nouveau voisinage qui ne contient que $3^t - 2^t$ points. Supposons que P_n soit sans collision en résolution v. En résolution $v+1$, pour vérifier le voisinage $N_{v+1}(\mathbf{x})$ pour un point $\mathbf{x} \in P_n$, il n'est pas nécessaire de vérifier toutes les 3^t cellules adjacentes à $C_{v+1}(\mathbf{x})$. Grâce au fait que P_n soit sans collision en résolution v, on peut éliminer de l'ensemble des cellules à considérer les 2^t cellules en résolution $v+1$ qui forment $C_v(\mathbf{x})$. Il est certain que ces cellules ne contiennent pas d'autre point que $\mathbf{x}$. Ainsi, si P_n est sans collision en résolution v, alors $N_{v+1}(\mathbf{x})$ contient au plus $3^t - 2^t$ points. Pour tenir compte de ce fait, on définit le voisinage $\bar{N}_v(\mathbf{x})$ qui est le même que $N_v(\mathbf{x})$, mais en ne considérant pas les cellules en résolution v qui forment $C_{v+1}(\mathbf{x})$. Quand P_n est sans collision en résolution v, alors $N_{v+1}(\mathbf{x}) = \bar{N}_{v+1}(\mathbf{x}) + \{\mathbf{x}\}$.

Exemple 4.4. La figure 4.5 illustre les cellules considérées pour les voisinages $N_v(\mathbf{x})$ et $\bar{N}_v(\mathbf{x})$ en deux dimensions. Les lignes minces représentent les frontières entre les cellules en résolution $v+1$, tandis que les lignes épaisses représentent les frontières entre les cellules en résolution v. Soit $\mathbf{x}$, le point illustré. Dans la figure de gauche, les cellules hachurées sont celles qui sont considérées dans $N_v(\mathbf{x})$ et dans la figure de droite, les cellules hachurées sont celles qui sont considérées dans $\bar{N}_v(\mathbf{x})$. Si P_n est sans collision en résolution v, alors $\mathbf{x}$ se retrouve seul dans les quatre cellules en résolution $v+1$ qui forment $C_v(\mathbf{x})$ et $N_v(\mathbf{x}) = \bar{N}_v(\mathbf{x}) + \{\mathbf{x}\}$.

De plus, il est possible pour certains points $\mathbf{x}$ et résolutions v d'avoir la certitude que $N_v(x) = \{\mathbf{x}\}$. Pour ce faire, introduisons $v(\mathbf{x}, v_0)$, la plus petite résolution v pour laquelle $V_v(\mathbf{x}) \subset C_{v_0}(\mathbf{x})$:

$$v(\mathbf{x}, v_0) = \min\{v : V_v(\mathbf{x}) \subset C_{v_0}(\mathbf{x})\}.$$

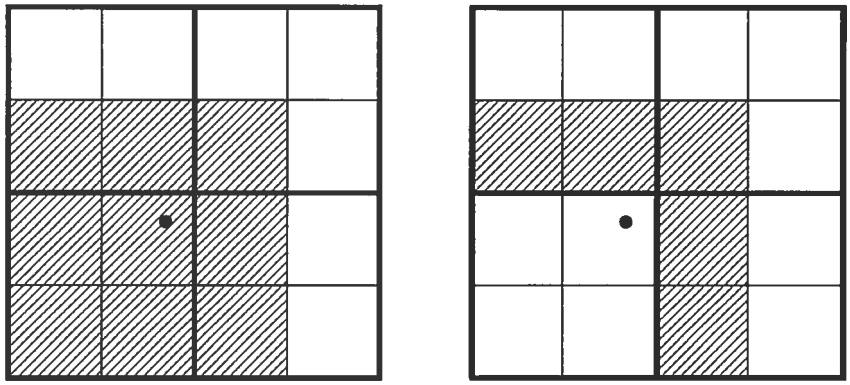


Figure 4.5 – Voisinages $N_v(\mathbf{x})$ et $\bar{N}_v(\mathbf{x})$.

S'il n'existe pas de résolution v pour laquelle $V_v(\mathbf{x}) \subset C_{v_0}(\mathbf{x})$ (c'est-à-dire, si $\mathbf{x}$ se trouve sur la frontière de la cellule), on dit que $v(\mathbf{x}, v_0)$ est non défini et, par convention, on met $v(\mathbf{x}, v_0) = \infty$. L'utilité de $v(\mathbf{x}, v_0)$ est illustrée par les prochaines propositions.

Proposition 4.4. *Si P_n est sans collision en résolution v_0 et $v(\mathbf{x}, v_0) < \infty$, alors* $N_{v(\mathbf{x},v_0)}(\mathbf{x}) = \{\mathbf{x}\}$.

Démonstration. Si P_n est sans collision en résolution v_0, alors ceci implique qu'il n'y a qu'un seul point dans $C_{v_0}(\mathbf{x})$ ainsi que dans $V_{v(\mathbf{x},v_0)}(\mathbf{x})$, donc que $N_{v(\mathbf{x},v_0)}(\mathbf{x}) = \{\mathbf{x}\}$. □

Proposition 4.5. $V_v(\mathbf{x}) \subset C_{v_0}(\mathbf{x})$ *pour* $v \geq v(\mathbf{x}, v_0)$.

Démonstration. Ceci est déduit directement du fait que $V_{v+1}(\mathbf{x}) \subset V_v(\mathbf{x})$ et de la définition de $v(\mathbf{x}, v_0)$. □

Corollaire 4.1. *Si P_n est sans collision en résolution v_0 et $v \geq v(\mathbf{x}, v_0)$, alors* $N_v(\mathbf{x}) = \{\mathbf{x}\}$.

Ce dernier corollaire indique pour quelles résolutions v le voisinage de $\mathbf{x}$ ne contient que $\mathbf{x}$. Cette information peut être fort utile pour réduire l'effort de calcul.

Voici un théorème qui indique comment obtenir $v(\mathbf{x}, v_0)$.

Théorème 4.4. *Soit $P_n \in [0,1)^t$, un ensemble de n points. Soit*

$$\mathbf{x} = (x^{(0)}, \ldots, x^{(t-1)}) \in P_n$$

où

$$x^{(j)} = \sum_{i=0}^{\infty} a_{j,i} 2^{-i-1}$$

où $a_{j,i} \in \{0,1\}$, pour $j = 0, \ldots, t-1$. On a que $v(\mathbf{x}, v_0)$ est indéfini quand une des coordonnées de $\mathbf{x}$ est exactement de la forme $m2^{-v_0}$ où $m \in \{0, 1, \ldots, 2^{v_0} - 1\}$. Dans les autres cas,

$$v(\mathbf{x}, v_0) = \min \left\{ v : 1 \leq \sum_{i=v_0}^{v-1} a_{j,i} 2^{v-i-1} \leq 2^{v-v_0} - 2 \text{ pour } j = 0, \ldots, t-1 \right\}.$$

Démonstration. Pour que $V_v(\mathbf{x}) \subset C_{v_0}(\mathbf{x})$, il est impératif que $v > v_0$. Si $x^{(j)} = m2^{-v_0}$ pour une seule coordonnée j, alors, quelque soit v, le voisinage $V_v(\mathbf{x})$ croise toujours la frontière de la cellule $C_{v_0}(\mathbf{x})$ et il n'y a aucune résolution v pour laquelle $V_v(\mathbf{x}) \subset C_{v_0}(\mathbf{x})$. Dans ce cas, $v(\mathbf{x}, v_0)$ est indéfini. Dans les autres cas, on obtient ce qui suit. Soit $s(\mathbf{x}, j, v) = \sum_{i=0}^{v-1} a_{j,i} 2^{-i-1}$. Soient

$$V_v(\mathbf{x}) = \{\mathbf{y} \in [0,1)^t : s(\mathbf{x}, j, v) - 2^{-v} \leq y^{(j)} < s(\mathbf{x}, j, v) + 2^{-v+1} \text{ pour } j = 0, \ldots, t-1\}$$

et

$$C_{v_0}(\mathbf{x}) = \{\mathbf{y} \in [0,1)^t : s(\mathbf{x}, j, v_0) \leq y^{(j)} < s(\mathbf{x}, j, v_0) + 2^{-v_0} \text{ pour } j = 0, \ldots, t-1\}$$

À partir de ces définitions, $V_v(\mathbf{x}) \subset C_{v_0}(\mathbf{x})$ si et seulement si

$$s(\mathbf{x}, j, v_0) \leq s(\mathbf{x}, j, v) - 2^{-v}$$

et

$$s(\mathbf{x}, j, v) + 2^{-v+1} \leq s(\mathbf{x}, j, v_0) + 2^{-v_0}$$

pour $j = 0, \ldots, t-1$. La première inégalité est équivalente à

$$1 \leq \sum_{i=v_0}^{v-1} a_{j,i} 2^{v-i-1}$$

et la deuxième, à

$$\sum_{i=v_0}^{v-1} a_{j,i} 2^{v-i-1} \leq 2^{v-v_0} - 2$$

En combinant ces deux inégalités, on obtient que $V_v(\mathbf{x}) \subset C_{v_0}(\mathbf{x})$ si et seulement si

$$1 \leq \sum_{i=v_0}^{v-1} a_{j,i} 2^{v-i-1} \leq 2^{v-v_0} - 2$$

pour $j = 0, \ldots, t-1$.

On a défini $v(\mathbf{x}, v_0)$ comme la plus petite valeur de v telle que $V_v(\mathbf{x}, v_0) \subset C_{v_0}(\mathbf{x})$, c'est pourquoi $v(\mathbf{x}, v_0)$ est indéfini quand une seule coordonnée est un multiple de 2^{-v_0} et dans les autres cas

$$v(\mathbf{x}, v_0) = \min \left\{ v : 1 \leq \sum_{i=v_0}^{v-1} a_{j,i} 2^{v-i-1} \leq 2^{v-v_0} - 2 \text{ pour } j = 0, \ldots, t-1 \right\}.$$

□

Exemple 4.5. Considérons un ensemble de points $P_n \subset [0,1)^2$ qui soit sans collision en résolution $v_0 = 2$. Soit $\mathbf{x}_1, \mathbf{x}_2 \in P_n$ où $\mathbf{x}_1 = (2^{-1} + 2^{-4}, 2^{-1} + 2^{-2} + 2^{-3})$ et $\mathbf{x}_2 = (2^{-1} + 2^{-3} + 2^{-4}, 2^{-1} + 2^{-2} + 2^{-3})$. Les deux points sont illustrés à la figure 4.6. Pour $\mathbf{x}_1$, on a

$$\mathbf{a} = (a_{0,0}, a_{0,1}, a_{0,2}, a_{0,3}, a_{1,0}, a_{1,1}, a_{1,2}, a_{1,3}) = (1,0,0,1,1,1,1,0)$$

et $v(\mathbf{x}_1, 2) = 4$. La figure 4.6 illustre le voisinage $V_4(\mathbf{x}_1)$. Elle montre aussi que $V_4(\mathbf{x}_1) \subset C_2(\mathbf{x}_1)$. On peut concevoir, en observant la figure, que $V_3(\mathbf{x}_1) \not\subset C_2(\mathbf{x}_1)$. On peut conclure que $v(\mathbf{x}_1, 2) = 4$. Pour $\mathbf{x}_2$, on a $\mathbf{a} = (1,0,1,1,1,1,1,0)$. La figure illustre les voisinage $V_3(\mathbf{x}_2)$, $V_4(\mathbf{x}_2)$ et $V_5(\mathbf{x}_2)$. On remarque que seul $V_5(\mathbf{x}_2)$ est contenu dans $C_2(\mathbf{x}_2)$. À partir de la figure, il est facile de concevoir que $V_v(\mathbf{x}_2) \subset C_2(\mathbf{x}_2)$ pour

$v > 5$. Donc, la plus petite résolution v telle que $V_v(\mathbf{x}_2)$ est contenu dans $C_2(\mathbf{x}_2)$ est $v = v(\mathbf{x}_2, 2) = 5$.

Puisque $v(\mathbf{x}_2, 2) = 4$, par le dernier corollaire, si P_n est sans collision en résolution 2, alors $N_v(\mathbf{x}) = \{\mathbf{x}\}$ pour tout $v \geq 4$. Dans les cas où $v < 4$, on ne peut déterminer si $N_v(\mathbf{x}) = \{\mathbf{x}\}$ ou non en ne connaissant que $v(\mathbf{x}_2, 2)$.

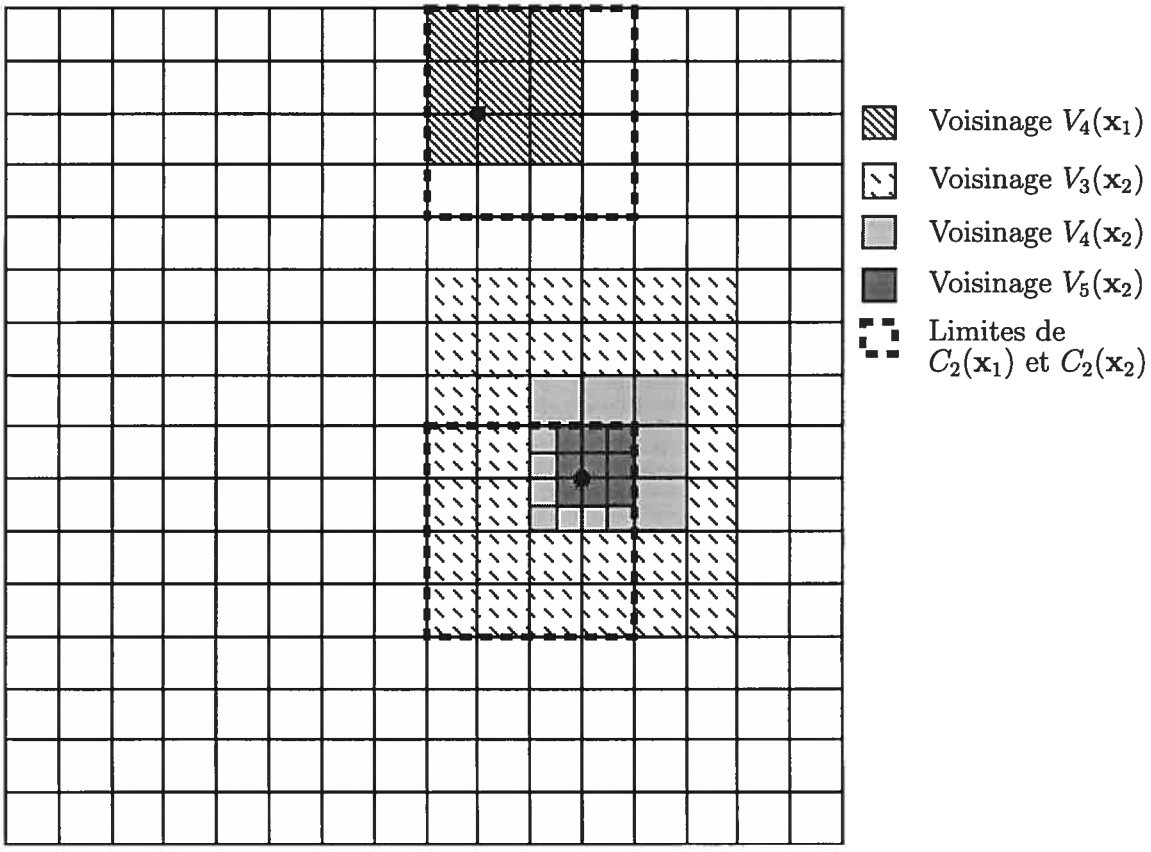


Figure 4.6 – Voisinages des points $\mathbf{x}_1$ et $\mathbf{x}_2$ définis à l'exemple 4.5.

4.5.5 Second nouvel algorithme

Voici un nouvel algorithme inspiré de celui de Khuller-Matias. Dans celui-ci, lorsqu'on utilise $N_v(\mathbf{x})$ ou $\bar{N}_v(\mathbf{x})$, on peut appliquer le corollaire 4.1. Pour l'algorithme, on définit $R_v(S) = \{\mathbf{x} \in S | \bar{N}_v(\mathbf{x}) = \{\mathbf{x}\}\}$ où $S \subseteq P_n$. Cet algorithme permet de calculer $d_2^*(P_n)$ ou $d_\infty^*(P_n)$.

Algorithme 4.3. Calcul de $d_p^*(P_n)$

1. Calculer v_0, la plus petite résolution pour laquelle l'ensemble de points P_n est sans collision.
2. $v \leftarrow v_0 + 1$.
3. $S_v \leftarrow P_n$.
4. Répéter

   $X \leftarrow S_v - R_v(S_v)$.

   Si $X = \emptyset$, alors $v^* \leftarrow v$ et sortir de la boucle.

   Parmi les points $\mathbf{x} \in X$, en choisir un au hasard, $\mathbf{x}'$.

   $\mathbf{y} \leftarrow \arg\min\{d_\infty(\mathbf{x}', \mathbf{y}) : \mathbf{y} \in \bar{N}_v(\mathbf{x}'), \mathbf{y} \neq \mathbf{x}'\}$.

   $v \leftarrow$ la plus petite valeur de v telle que $\mathbf{y} \notin \bar{N}_v(\mathbf{x}')$ (voir l'algorithme 4.4).

   $S_v \leftarrow X$.
5. (Invariant : $2^{-v^*} < d_2^*(P_n) < 2^{-v^*+2}\sqrt{t}$).
6. (Invariant : $2^{-v^*} < d_\infty^*(P_n) < 2^{-v^*+2}$).
7. $v \leftarrow v^* - c_{t,p} - 1$.
8. Si $v < v_0 + 1$, pour chaque point $\mathbf{x} \in P_n$, calculer $d_p(\mathbf{x}) = \min\{d_p(\mathbf{x}, \mathbf{y}) : \mathbf{y} \in N_v(\mathbf{x}), \mathbf{y} \neq \mathbf{x}\}$. La valeur de $d_p^*(P_n)$ est $\min\{d_p(\mathbf{x}) : \mathbf{x} \in P_n\}$. Terminer.
9. Sinon, pour chaque point $\mathbf{x} \in P_n$, calculer $d_p(\mathbf{x}) = \min\{d_p(\mathbf{x}, \mathbf{y}) : \mathbf{y} \neq \mathbf{x}, \mathbf{y} \in \bar{N}_v(\mathbf{x})\}$. La valeur de $d_p^*(P_n)$ est $\min\{d_p(\mathbf{x}) : \mathbf{x} \in P_n\}$. Terminer.

Ce dernier algorithme exploite l'information que l'ensemble de points soit sans collision ou non en résolution $v-1$ pour déterminer les points dans le voisinage $N_v(\mathbf{x})$. De plus, l'information donnée par la résolution $v(\mathbf{x}, v_0)$ peut être mise à profit pour accélérer le calcul de $N_v(\mathbf{x})$.

Analysons ce dernier algorithme et comparons-le à celui de Khuller-Matias. Une différence majeure est que l'on ne considère que des grillages de la forme $\tilde{G}_{2^{-v}}$. Le processus de filtrage est semblable, sauf que dans notre cas, on utilise toujours la fonction de distance $d_\infty(\mathbf{x}, \mathbf{y})$, indépendamment du fait qu'on veuille calculer $d_2(P_n)$ ou $d_\infty(P_n)$. Cette fonction est plus appropriée puisque nous utilisons des cellules cubiques et que le but du filtrage est de trouver la plus grande résolution v telle que $N_{v-1}(\mathbf{x}) \neq \{\mathbf{x}\}$ et $N_v(\mathbf{x}) = \{\mathbf{x}\}$ pour au moins un $\mathbf{x} \in P_n$. Pour ce faire, on choisit au hasard un point $\mathbf{x}'$ et on détermine son plus proche voisin $\mathbf{y}$ contenu dans son voisinage $\bar{N}_v(\mathbf{x}')$. La résolution v choisie pour le prochain grillage est telle que $\mathbf{y} \in N_{v-1}(\mathbf{x}')$ et $\mathbf{y} \notin N_v(\mathbf{x}')$. La proposition suivante nous aide à trouver cette résolution.

Proposition 4.6. *Soient* $\mathbf{x}, \mathbf{y}$, *deux éléments arbitraires de* P_n, *et* $d = d_\infty(\mathbf{x}, \mathbf{y})$.

1. *Si* $d \geq 2^{-v}$, *alors* $\mathbf{y} \notin C_v(\mathbf{x})$.
2. *Si* $d \geq 2^{-v+1}$, *alors* $\mathbf{y} \notin N_v(\mathbf{x})$.
3. *Si* $d < 2^{-v}$, *alors* $\mathbf{y} \in N_v(\mathbf{x})$.
4. *Si* $2^{-v} \leq d < 2^{-v+1}$, *alors* $\mathbf{y} \in N_{v-1}(\mathbf{x})$ *et* $\mathbf{y} \notin N_{v+1}(\mathbf{x})$

Démonstration.

1. Si d est plus grand que la largeur d'une cellule, alors $\mathbf{x}$ et $\mathbf{y}$ ne peuvent être dans la même cellule.
2. Si d est plus grand que deux fois la largeur d'une cellule, alors $\mathbf{y}$ ne peut être dans une cellule adjacente à $C_v(\mathbf{x})$.
3. Si d est plus petite que la largeur d'une cellule, alors $\mathbf{y}$ est nécessairement dans une cellule adjacente à $C_v(\mathbf{x})$.

4. Découle directement des énoncés 2 et 3 de cette proposition. □

Soient deux points $\mathbf{x}$ et $\mathbf{y}$ qui sont tels que $d = d_\infty(\mathbf{x}, \mathbf{y})$. Soit v, un entier tel que $2^{-v} \leq d < 2^{-v+1}$. Cette valeur de v est

$$\min\{v \in \mathbb{Z} : 2^{-v} \leq d\} = \min\{v \in \mathbb{Z} : v \geq -\log_2(d)\} = \lceil -\log_2(d) \rceil.$$

Par le quatrième énoncé de la précédente proposition, on sait que $\mathbf{y} \in N_{v-1}(\mathbf{x})$ et $\mathbf{y} \notin N_{v+1}(\mathbf{x})$, mais on ne sait pas si $\mathbf{y} \in N_v(\mathbf{x})$. Pour le savoir, il faut vérifier si $C_v(\mathbf{y}) \in V_v(\mathbf{x})$ en déterminant la cellule en résolution v qui contient $\mathbf{y}$ et celles qui constituent $V_v(\mathbf{x})$. Si $C_v(\mathbf{y}) \in V_v(\mathbf{x})$, alors $\mathbf{y} \in N_v(\mathbf{x})$, sinon $\mathbf{y} \notin N_v(\mathbf{x})$. Ainsi, la plus petite valeur de v telle que $\mathbf{y} \notin N_v(\mathbf{x}')$ est calculée par l'algorithme suivant.

Algorithme 4.4. Calcul de la plus petite valeur de v telle que $\mathbf{y} \notin N_v(\mathbf{x})$.

1. $d \leftarrow d_\infty(\mathbf{x}, \mathbf{y})$
2. $v \leftarrow \lceil -\log_2(d) \rceil$.
3. Si $C_v(\mathbf{y}) \in V_v(\mathbf{x})$, alors retourner $v + 1$, sinon retourner v.

Dans la boucle de l'algorithme 4.3, on élimine tous les points $\mathbf{x} \in S_v$ tels que $d_\infty(\mathbf{x}) > d_\infty(\mathbf{x}')$ sauf la première fois où on élimine tous les points $\mathbf{x} \in P_n$ tels que $d_\infty(\mathbf{x}) > 2^{-v_0-1}$. Puisque le point $\mathbf{x}'$ est choisi au hasard, en moyenne, la moitié des points $\mathbf{y} \in S_v$ sont tels que $d_\infty(\mathbf{x}) > d_\infty(\mathbf{x}')$. Soit $s_v = |S_v|$, au début de la boucle, et

$$\eta = |R_{v_0+1}(S_{v_0+1})|/|S_{v_0+1}|,$$

la fraction des points de S_{v_0+1} qui ont été éliminés au début de la boucle. Si S_v n'est

pas défini dans l'algorithme, alors $s_v = 0$. On obtient

$$
\begin{aligned}
E\left[\sum_{v=v_0+1}^{v^*} s_v\right] &= n + E\left[\sum_{v=v_0+2}^{v^*} s_v\right] && (4.5)\\
&\leq n + \sum_{v=v_0+2}^{n} n\eta/2^{v-v_0-2} && (4.6)\\
&= n + \sum_{i=0}^{n-v_0-2} n\eta/2^{i} && (4.7)\\
&\leq n + n\eta \sum_{i=0}^{\infty} 1/2^{i} && (4.8)\\
&= n(1+2\eta) && (4.9)
\end{aligned}
$$

Ceci permet de dire que $E[\sum_{v=v_0+1}^{v^*} s_v] < n(1+2\eta)$ et puisque $\eta \leq 1$, on a

$$E\left[\sum_{v=v_0+1}^{v^*} s_v\right] \in O(n).$$

En résolution v, pour vérifier si l'ensemble de point S_v est sans voisin, on emmagasine les coordonnées des points dans une table de hachage. La fonction de hachage n'utilise que les v bits les plus significatifs pour calculer la clé de chacun des points. Chaque clé correspond à une cellule en résolution v. Une fois tous les points emmagasinés dans la table, on vérifie si les points ont un voisin en examinant les entrées de la table où leurs voisins devraient se trouver.

Un avantage majeur de ce nouvel algorithme par rapport à l'algorithme de Khuller-Matias est que les cellules de différentes résolutions sont imbriquées les unes dans les autres. Ceci simplifie grandement la vérification du voisinage d'un point. Voici pourquoi.

Pour construire la table de hachage, on choisit une résolution $\bar{v}$ assez grossière pour calculer la clé de chacun des points et chaque clé unique correspond à une cellule en résolution $\bar{v}$. On emmagasine les points dans la table de hachage selon ces clés. Pour vérifier le voisinage d'un point $\mathbf{x}$ en résolution $v \geq \bar{v}$, on calcule les clés des points

qui se trouveraient dans les 3^t cellules voisines et on regarde, avec ces clés, dans la table de hachage s'ils s'y trouvent. Puisque les cellules de différentes résolution sont imbriquées les unes dans les autres, pour vérifier s'il y a un point dans une cellule voisine C de résolution v, on n'a qu'à vérifier dans un seul emplacement dans la table de hachage, soit l'emplacement qui correspond à la cellule $\bar{C}$ de résolution $\bar{v}$ telle que $C \subset \bar{C}$. Si les cellules de différentes résolutions n'étaient pas imbriquées, comme c'est le cas dans l'algorithme de Khuller-Matias, alors il faudrait vérifier, dans la table de hachage, tous les emplacements qui correspondent aux cellules $\bar{C}_i$, $i = 1, ..d$, telles que $C \subset \cup_{i=1}^{d} \bar{C}_i$. Pour une cellule C, déterminer les cellules $\bar{C}_i$, $i = 1, ..d$, correspond à trouver où tombent les coins de la cellule C. De plus, pour certaines cellules C, la valeur de d peut être, dans le pire des cas, 2^t, ce qui correspond au cas où le coin d'une cellule $\bar{C}_i$ se trouve complètement à l'intérieur de C. Par contre, dans l'article [23], l'auteur construit une nouvelle table de hachage à chaque fois que la grosseur du grillage change.

Pour les réseaux digitaux, on peut facilement construire la table de hachage. Pour ceux-ci, il est facile de calculer la valeur de ℓ_t et de se servir de cette résolution pour construire la table de hachage. De cette manière, on sait qu'il y a exactement $2^{t\ell_t}$ clés possibles et $2^{k-t\ell_t}$ points qui ont exactement la même clé. Ainsi, on peut adopter un modèle de table de hachage assez simple qui réserve $2^{k-t\ell_t}$ espaces pour chaque clé. Pour trouver un point dans la table, il suffit de calculer la clé et de faire une recherche linéaire dans l'espace réservé à cette clé. L'avantage de cette stratégie est que toute la mémoire allouée pour la table est utilisée. Par contre, si on juge que $2^{k-t\ell_t}$ est trop grand pour la recherche linéaire, alors on peut choisir une résolution $\bar{v} > \ell_t$. et allouer $2^{k-t\ell_t}$ espaces pour chaque clé. Ainsi, on il y a $2^{t\bar{v}}$ clés possibles et on alloue $2^{k-t\ell_t}$ espaces pour chaque clé. En choisissant une valeur de $\bar{v}$ telle que $2^{k-t\bar{v}}$ est plus petit que $2^{k-t\ell_t}$, on s'assure d'avoir une recherche linéaire plus efficace, par contre la table de hachage est plus éparse et gaspille, en quelque sorte, plus de mémoire.

Lorsque l'algorithme arrive à l'étape 7, puisque P_n est sans voisin en résolution v^* et n'est pas sans voisin en résolution v^*-1, alors on sait que $2^{-v^*} < d_2^*(P_n) < 2^{-v^*+2}\sqrt{t}$ et que $2^{-v^*} < d_\infty^*(P_n) < 2^{-v^*+2}$ par la proposition 4.13. Par la propriété 4.15, on sait que l'étape 8 ou 9 calcule vraiment $d_p^*(P_n)$. La différence entre les étapes 8 et 9 est que si $v < v_0 + 1$, alors on n'a pas la propriété que P_n est sans collision en résolution $v-1$ et la propriété $\bar{N}_v(\mathbf{x}) = N_v(\mathbf{x})$ pour tout $\mathbf{x} \in P_n$ ne tient pas non plus. Ainsi, le nombre de points dans le voisinage $N_v(\mathbf{x})$ de $\mathbf{x}$ est inférieur ou égal à $3^t 2^t$. Par contre, si $v \geq v_0 + 1$, alors on a la propriété $\bar{N}_v(\mathbf{x}) = N_v(\mathbf{x})$ pour tout $\mathbf{x} \in P_n$. Le nombre de points dans le voisinage $N_v(\mathbf{x})$ de $\mathbf{x}$ est au plus $3^t - 2^t$. Dans les deux cas, le nombre d'opérations est $O(n)$.

On peut maintenant affirmer, en supposant que l'étape 1 de l'algorithme s'effectue en $O(g(n))$ opérations, que l'algorithme 4.3 s'exécute en $O(g(n) + n)$ opérations.

On remarque que pour calculer la distance minimale avec la norme L_2, on doit utiliser, à l'étape 8 ou 9, la résolution $v^* - c_{t,2} - 1 < v^* - 2$. Pour la norme L_∞, cette résolution est toujours $v^* - 2$, quelle que soit la dimension. C'est pourquoi, pour un point donné on peut s'attendre à ce que le voisinage à considérer dans le calcul de $d_2^*(P_n)$ contienne beaucoup plus de points que dans le calcul de $d_\infty^*(P_n)$. C'est principalement ce qui explique la différence marquée du temps d'exécution de l'algorithme pour le calcul de $d_2^*(P_n)$ et de $d_\infty^*(P_n)$.

4.5.6 Déterminer v_0 quand P_n est un réseau digital

Dans les deux nouveaux algorithmes présentés dans les sections précédentes, la première étape consiste à trouver la plus petite résolution v_0 telle que P_n soit sans collision. Pour les ensembles de points qui nous intéressent, soit les réseaux digitaux en base 2, il est possible de le faire dans un temps $g(n) \in O(\log^4 n)$, comme il est montré dans la démonstration de la prochaine proposition.

Proposition 4.7. *Si P_n est un réseau digital en base 2, les algorithmes 4.2 et 4.3 s'exécutent, en moyenne, en $O(n)$ opérations.*

Démonstration. Par le théorème 4.3, en utilisant l'élimination gaussienne sur la matrice $T_{\mathbf{p}}$ (où $p_1 = \cdots = p_t = v$), on peut déterminer si un réseau digital en base 2 est sans collision ou non. L'élimination se fait en $O(k^3) = O(\log^3 n)$ opérations. Pour déterminer la plus petite résolution v_0 telle que P_n est sans collision, on vérifie pour $v = \lceil k/t \rceil, \ldots, k$. La résolution $\lceil k/t \rceil$ est la plus petite telle que $n/2^{tv} = 2^{k-tv} \leq 1$, condition nécessaire pour que P_n soit sans collision en résolution v. Le nombre de résolutions, en pire cas, à vérifier est $O(k) = O(\log n)$. Le nombre d'opérations pour déterminer v_0 est donc $O(\log^4 n) \subset O(n)$. La fonction $g(n)$ qu'on n'a pas encore définie est $g(n) = \log^4 n$ si P_n est un réseau digital en base 2. Puisque $g(n) \in O(n)$, ceci implique directement que les deux algorithmes s'exécutent, en moyenne, en $O(n)$ opérations si P_n est un réseau digital en base 2. □

4.5.7 Modifications possibles à l'algorithme 4.3

Si l'ensemble de points P_n n'est pas un réseau digital en base 2, mais un ensemble de points quelconque, alors il faut tenter d'obtenir la valeur de v_0 par un algorithme approprié. Si un tel algorithme n'est pas disponible, on peut procéder aux modifications suivantes. Les étapes 1 et 2 sont remplacées par $v \leftarrow \lceil \log_2 n/t \rceil$. Pour qu'un ensemble de n points soit sans collision, il faut nécessairement que $n/2^{tv} \leq 1$, d'où $v \leftarrow \lceil \log_2(n)/t \rceil$. Dans la boucle, il faut utiliser le voisinage $N_v(\mathbf{x})$ au lieu de $\bar{N}_v(\mathbf{x})$ jusqu'à ce qu'on ait la certitude que P_n est sans collision en résolution $v-1$. Ceci est facile à déterminer à cette étape. Une fois que l'on rencontre une résolution v telle que P_n est sans collision, mettre $v_0 \leftarrow v$. À partir de ce moment, quand $v > v_0$, on peut utiliser le voisinage $\bar{N}_v(\mathbf{x})$, car on sait que P_n est sans collision en résolution $v-1$. Le reste de l'algorithme se déroule tel que décrit à la section 4.5.3. Un point à souligner

est que si on n'a jamais vérifié le voisinage avec $v = v_0 - 1$ à l'intérieur de la boucle, il n'y a aucun moyen de déterminer si v_0 est la plus petite résolution telle que P_n est sans collision. Ceci veut dire qu'il n'est pas impossible qu'on exécute l'étape 8, alors qu'on aurait pu exécuter l'étape 9 à la place, qui est plus efficace. L'algorithme 4.3 s'exécute quand même en $O(n)$ opérations si on effectue ces modifications.

4.5.8 Performance de l'algorithme 4.3

Pour comparer la vitesse d'exécution de notre deuxième nouvel algorithme, nous l'avons comparé avec un algorithme déterministe qui s'exécute en temps $O(n \log n)$. Il s'agit d'une modification d'un algorithme décrit dans [90]. L'implantation utilisée, qui est celle du module `sn_pair` de TestU01 [48], est très générale, mais a été très optimisée selon l'auteur de [48], Pierre L'Ecuyer. Voir [41] pour plus de détails sur l'algorithme. Dans cette section, nous nommons l'algorithme utilisé dans TestU01 « algorithme A ».

Nous avons mesuré la vitesse d'exécution des deux algorithmes dans les cas où on utilise la norme L_2 et la norme L_∞, pour toutes les projections en deux et en trois dimensions d'ensembles de points en 19 dimensions. La cardinalité des ensembles de points varie de 2^{10} à 2^{18}. On donne aussi le temps pris pour trouver la résolution v^* à laquelle une projection est sans-voisin. Les résultats sont exposés dans les tableaux 4.1 et 4.2.

Au tableau 4.1, on remarque que l'algorithme 4.3 devient éventuellement meilleur que l'algorithme A, en deux dimensions, à mesure que le nombre de points croît pour les deux normes considérées. Ceci semble confirmer, d'une manière empirique, les temps d'exécution dans $O(n)$ et $O(n \log n)$ des algorithmes 4.3 et A, respectivement.

Le tableau 4.1 montre qu'il faut que la cardinalité des ensembles de points soit grande avant que l'algorithme 4.3 soit plus rapide que l'algorithme A. Pour la norme

Tableau 4.1 – Temps, en secondes, pour calculer la distance minimale de 18 projections bi-dimensionelles d'un ensemble de points en 19 dimensions.

	Algorithme 4.3		Algorithme A		v^*
	L_2	L_∞	L_2	L_∞	
2^{10}	0.02	0.01	**0.02**	**0.01**	0.01
2^{12}	0.09	0.06	**0.04**	**0.04**	0.03
2^{14}	0.40	**0.25**	**0.34**	0.34	0.16
2^{15}	**0.85**	**0.55**	0.94	0.94	0.35
2^{16}	**2.0**	**1.4**	2.5	2.5	1.1
2^{18}	**9.7**	**6.0**	13	13	5.5

Tableau 4.2 – Temps, en secondes, pour calculer la distance minimale de 153 projections tri-dimensionelles d'un ensemble de points en 19 dimensions.

	Algorithme 4.3		Algorithme A		v^*
	L_2	L_∞	L_2	L_∞	
2^{10}	0.86	0.21	**0.12**	**0.12**	0.16
2^{12}	4.9	0.95	**0.50**	**0.48**	0.68
2^{14}	12	4.1	**3.2**	**3.1**	3.6
2^{15}	30	9.4	**7.8**	**7.5**	8.5
2^{16}	48	**18**	**19**	20	17
2^{18}	270	**120**	**120**	**120**	110

L_∞, l'algorithme 4.3 commence à obtenir des temps semblables à l'algorithme A quand la cardinalité des ensembles de points est autour de 2^{14} et pour la norme L_2, l'algorithme 4.3 devient meilleur quand le nombre de points dépasse 2^{15}.

Le tableau 4.2 montre que l'algorithme 4.3 commence à être compétitif quand la cardinalité d'ensembles de points en trois dimensions atteint 2^{16}.

Si on est intéressé à connaître que la résolution v^* à laquelle un ensemble de points en deux dimensions est sans voisin, alors on remarque que celle-ci est généralement obtenue plus rapidement que la distance minimale avec l'algorithme A. Pour les ensembles de points en trois dimensions, il faut que la cardinalité de l'ensemble de points considéré soit au moins de 2^{16}.

Il est évident que les résultats obtenus sont très sensibles à l'implantation de chacun des algorithmes, mais les tendances seraient les mêmes pour d'autres implantations.

Nous avons contacté les auteurs de [23] afin d'obtenir une implantation de l'algorithme de Khuller-Matias original. Malheureusement, aucune n'est disponible. Un collègue de Khuller, William Gasarch, a engagé un étudiant pour l'implanter, mais celui-ci n'a pas réussi à obtenir une implantation très efficace et n'a pas conservé le code. Cette démarche auprès des auteurs de [23] nous a convaincu du fait que l'algorithme 4.3 permet des implantations beaucoup plus efficaces que l'algorithme de Khuller-Matias original. En voici les principales raisons :

- l'algorithme de Khuller-Matias n'est pas utile pour calculer la plus courte distance sur le tore $[0,1)^t$, l'algorithme 4.3 l'est.
- le fait de choisir des cellules dont la valeur de b est une puissance négative de 2 permet de déterminer rapidement le numéro de la cellule dans laquelle un point tombe et son voisinage à l'aide de masques de bits ; quand b est une valeur réelle, il faut diviser les coordonnées d'un point par b pour connaître sa cellule

et son voisinage. Les opérations sur les bits sont généralement plus rapides que les opérations sur les réels sur un ordinateur.

- le fait que les cellules considérées tout au long de l'algorithme sont toutes imbriquées les unes dans les autres dans l'algorithme 4.3 permet de simplifier grandement la gestion de la table de hachage. On n'a besoin d'insérer les points qu'une seule fois dans la table de hachage. Dans l'algorithme de Khuller-Matias, les cellules considérées ne sont pas imbriquées les unes dans les autres et il faut remplir la table de hachage chaque fois que la grosseur des cellules changent. Une autre alternative est de remplir une seule fois la table de hachage, mais la recherche d'un point dans la table de hachage devient plus compliquée, comme il est expliqué à la section 4.5.5.

Ces raisons permettent de croire qu'il serait très difficile, pour une implantation de l'algorithme de Khuller-Matias, de faire mieux que notre implantation de l'algorithme 4.3.

4.5.9 Critères d'uniformité basés sur la distance minimale

Dans cette section, nous définissons des critères d'uniformité basés sur l'algorithme 4.3 de la section 4.5.5. En général, pour deux ensembles de points donnés, P_n et P'_n, on dira que P_n est plus uniforme que P'_n si $d_p^*(P_n) > d_p^*(P'_n)$ (par exemple, voir la figure 4.3).

Le premier critère que nous définissons n'est pas la distance minimale $d_p^*(P_n)$ en tant que telle. C'est plutôt la plus petite résolution v^* pour laquelle l'ensemble de points est sans voisin. On sait que cet entier v^* est tel que

$$2^{-v^*} \leq d_2^*(P_n) \leq 2^{-v^*+2}\sqrt{t} \quad \text{et} \quad 2^{-v^*} \leq d_p^*(P_n) \leq 2^{-v^*+2}. \tag{4.10}$$

Ce v^* est obtenu à l'étape 7 de l'algorithme 4.3. Ainsi, la valeur de v^* nous donne une bonne idée de l'ordre de grandeur de $d_p^*(P_n)$. Pour utiliser ce v^* comme critère

d'uniformité, on dira que plus la valeur de v^* est petite, plus l'ensemble de points est uniforme. Nous savons que la plus petite résolution pour laquelle un ensemble de points qui a $n = 2^k$ points est sans collision est la plus petite valeur $\tilde{v}_t$ telle que $2^{k-t\tilde{v}_t} \leq 1$, donc $\tilde{v}_t = \lceil k/t \rceil$. Par la propriété 4.12, on sait que pour une résolution $\tilde{v}_t$, l'ensemble de points ne peut pas être sans voisin. Ainsi, la plus petite résolution pour laquelle il soit possible que P_n soit sans voisin est

$$v_t \stackrel{\text{def}}{=} \lceil k/t \rceil + 1$$

À partir de cette valeur, on peut définir

$$\Gamma_t = v^* - v_t.$$

La valeur Γ_t, qui est toujours positive, représente l'écart entre la meilleure valeur de v^* possible et celle obtenue. En t dimensions, pour un ensemble de points P_n donné, on dira que plus Γ_t est petit, plus l'ensemble de points est uniforme.

Exemple 4.6. Considérons les deux ensembles de points illustrés à la figure 4.3. Il s'agit de deux ensembles de $n = 2^3$ points en $t = 2$ dimensions. La valeur de v_2 est donc $\lceil 3/2 \rceil + 1 = 3$. Pour l'ensemble de points de droite, on voit qu'il est sans collision, mais pas sans voisin, en résolution $v = 2$. Par contre, il est sans voisin en résolution $v = 3$. On obtient donc, pour cet ensemble de points, $\Gamma_2 = v_2 - 3 = 0$, ce qui est le mieux qu'on puisse obtenir pour ce critère.

Pour l'ensemble de points de gauche, on observe qu'il est sans collision, mais pas sans voisin, en résolution $v = 2$. On peut facilement concevoir que la résolution v pour laquelle chacun des points ne se trouvera pas dans le voisinage d'un autre point sera très élevée. Ceci est dû au fait que certaines paires de points sont extrêmement rapprochées.

4.6 Propriétés des projections

Dans cette section, on donne des définitions qui permettent de caractériser un ensemble de points $P_n \subset [0,1)^t$ par rapport à ses projections sur diverses coordonnées. Soit

$$P_n = \{\mathbf{u}_i = (u_{0,i}, \ldots, u_{t-1,i}) : 1 \leq i \leq n\},$$

et $J = \{j_1, \ldots, j_s\}$ où $0 \leq j_1 < \cdots < j_s < t$, un ensemble d'indices. Également, soit $P_n(J) = \{(u_{j_1,i}, \ldots, u_{j_s,i}) : 1 \leq i \leq n\}$, la projection de P_n sur les coordonnées définies par J.

Définition 4.6. [53] Un ensemble de points $P_n \subset [0,1)^t$ est *stationnaire dans la dimension* si, pour tout entier s tel que $0 \leq s < t$ et toute projection $J = \{j_1, \ldots, j_s\}$ où $0 \leq j_1 < \cdots < j_s < t$, on a que

$$P_n(J) = P_n(\{j_1 + j, \ldots, j_s + j\})$$

pour tout $0 \leq j \leq t - j_s$.

Pour un ensemble de points, la propriété qu'il soit stationnaire dans la dimension est très pratique lors de l'analyse de l'uniformité de projections. Quand la dimension t est grande, il existe une quantité très grande de projections en s dimensions. C'est pourquoi, il est souvent impraticable d'analyser chacune des projections individuellement. Quand l'ensemble de points à analyser est stationnaire dans la dimension, alors en analysant une projection $J = \{j_1, \ldots, j_s\}$, on se trouve à analyser du même coup les projections $J + j$ pour $j = 0, \ldots, t - j_s$. La prochaine proposition nous indique que les ensembles de points $\Psi_t(\mathbf{X}, H, L)$ ont cette propriété, que ne possèdent pas, en général, les réseaux digitaux.

Proposition 4.8. *Soit $\Psi_t(\mathbf{X}, H, L)$, un réseau digital défini par les équations (1.1)–(1.4). Si la matrice $\mathbf{X}$ est non singulière, alors $\Psi_t(\mathbf{X}, H, L)$ est stationnaire dans la dimension.*

Démonstration. Soient $J = \{j_1, \ldots, j_s\}$, $J + j = \{j_1 + j, \ldots, j_s + j\}$ et l'ensemble

$$\bar{\Psi}_t(\mathbf{X}, J) = \{(X^{j_1}\mathbf{x}, \ldots, X^{j_s}\mathbf{x} : \mathbf{x} \in \mathbb{F}_2^k\}.$$

Soit $f : \mathbb{F}_2^k \to \mathbb{F}_2^L$, une fonction définie par $f(\mathbf{x}) = H\mathbf{x}$, et $\phi : \mathbb{F}_2^L \to [0, 1)$, définie par $\phi(\mathbf{y}) = \sum_{i=1}^{L} y^{(i-1)}2^{-i}$. Le réseau digital $P_n(J) = \Psi_t(\mathbf{X}, H, L, J)$ est obtenu en appliquant $f \circ \phi$ composante par composante à chacun des points de $\bar{\Psi}_t(\mathbf{X}, J)$. Soit

$$\bar{\Psi}_t(\mathbf{X}, J + j) = \{(X^{j_1+j}\mathbf{x}, , \ldots, X^{j_s+j}\mathbf{x}) : \mathbf{x} \in \mathbb{F}_2^k\}.$$

Puisque la matrice X est non singulière, il en est de même pour X^j. C'est pourquoi pour tout $\mathbf{x} \in \mathbb{F}_2^k$, il existe un unique $\mathbf{x}' \in \mathbb{F}_2^k$ tel que $\mathbf{x} = X^j\mathbf{x}'$. On peut donc remplacer $\mathbf{x}$ par $X^j\mathbf{x}'$ dans la définition de $\bar{\Psi}_t(\mathbf{X}, J)$ pour obtenir

$$\begin{aligned}
\bar{\Psi}_t(\mathbf{X}, J) &= \{(X^{j_1}\mathbf{x}, \ldots, X^{j_s}\mathbf{x}) : \mathbf{x} \in \mathbb{F}_2^k\} \\
&= \{(X^{j_1+j}\mathbf{x}', \ldots, X^{j_s+j}\mathbf{x}') : \mathbf{x} = X^j\mathbf{x}' \in \mathbb{F}_2^k\} \\
&= \{(X^{j_1+j}\mathbf{x}', \ldots, X^{j_s+j}\mathbf{x}') : \mathbf{x}' \in \mathbb{F}_2^k\} \\
&= \bar{\Psi}_t(\mathbf{X}, J + j)
\end{aligned}$$

Puisque $\bar{\Psi}_t(\mathbf{X}, J) = \bar{\Psi}_t(\mathbf{X}, J + j)$ pour tout $j > 0$, alors

$$\Psi_t(\mathbf{X}, H, L, J) = \Psi_t(\mathbf{X}, H, L, J + j)$$

pour tout $j > 0$. Ce qui complète la démonstration. □

4.7 Critères d'uniformité tenant compte de diverses projections

Les sections précédentes de ce chapitre décrivent des critères qui évaluent un ensemble de points P_n en t dimensions. Dans cette section, en se basant sur ces critères,

on définit d'autres critères qui examinent l'uniformité de P_n sur ses projections en faibles dimensions $P_n(J)$.

Le premier critère est

$$\Delta(S, C) = \max_{J \in S} C(P_n(J))$$

où S est est un ensemble de projections et $C(\cdot)$ est un critère tel que défini aux sections précédentes de ce chapitre. On appelle $C(\cdot)$ le *critère sous-jacent* à $\Delta(S, C)$. Plus la valeur de $\Delta(S, C)$ est petite, plus l'ensemble de points sera considéré uniforme.

Exemple 4.7. Dans [53], on utilise comme critère $C(P_n(J))$ la valeur de $\Lambda(J)$ de l'ensemble de points $P_n(J)$ et l'ensemble S est

$$S = \left(\bigcup_{s=1}^{r} \{\{j_1, \ldots, j_s\}, 0 = j_1 \leq \ldots, j_s < t_s\} \right) \bigcup \{\{0, \ldots, s\}, 1 \leq s < t_1\} \qquad (4.11)$$

où $s, t_1, \ldots, t_s$ sont des paramètres déterminés par les besoins de l'utilisateur. Ce critère examine l'équidistribution de toutes les projections de dimensions successives $J = \{0, \ldots, d-1\}$ pour $d = 0, \ldots, t_1 - 1$ et aussi l'équidistribution de toutes les projections J de dimension qui ne dépasse pas s et pour lesquelles $j_1 = 0$ et $j_s \leq t_s$. La valeur de $\Delta(S, \Lambda)$ retourne l'écart en résolution de la pire projection considérée.

Lorsque l'ensemble de points P_n est stationnaire en dimension, on se trouve à vérifier implicitement beaucoup plus de projections que $|S|$ avec le critère $\Delta(S, C)$. En fait, on vérifie toutes les projections contenues dans

$$S' = \{\{j_1 + m, \ldots, j_t + m\} : m \geq 0, J = \{j_1, \ldots, j_t\} \in S\}.$$

Pour un ensemble P_n qui a cette propriété, on a $C(P_n(J)) \leq \Delta(S, C)$ pour tout $J \in S'$.

Le deuxième type de critère est défini par

$$\Theta(S, C) = \sum_{J \in S} C(P_n(J)).$$

Celui-ci fait une sommation de la valeur du critère $C(P_n(J))$ sur toutes les projections considérées. Plus la valeur de $\Theta(S, C)$ est petite, plus l'ensemble de points sera considéré uniforme. Une propriété de ce critère est qu'il n'accorde pas une importance démesurée à une mauvaise projection, ce qui est le cas pour le critère $\Delta(S, C)$. Avec le critère $\Delta(S, C)$, toutes les projections de S, sauf une, pourraient être excellentes selon $C(P_n(J))$, mais cette seule projection pourrait faire en sorte qu'on considère P_n comme moins uniforme. Ce type de situation est moins susceptible d'arriver avec le critère $\Theta(S, C)$.

Exemple 4.8. Dans [53], on définit ce critère avec l'ensemble S de l'équation (4.11) et le critère $C(P_n(J))$ est l'écart en résolution $\Lambda(J)$ de la projection $P_n(J)$.

Dans cette thèse, on utilise les critères $\Delta(S, C)$ et $\Theta(S, C)$ avec les critères sous-jacents suivants : l'équidistribution ($\Lambda(J)$), la q-valeur et la distance minimale (Γ_t). Tous ces critères nous permettront de trouver de bons ensembles de points pour les applications Monte Carlo et quasi-Monte Carlo aux prochains chapitres.

Les critères d'uniformité que nous avons définis jusqu'à présent dans ce chapitre ont tous été implantés dans une bibliothèque informatique qu'on appelle REGPOLY. Cette bibliothèque, qui constitue une des plus grandes contributions de cette thèse, est présentée dans la prochaine section.

4.8 Description de REGPOLY

La bibliothèque de fonctions REGPOLY [82, 84] permet de faire des recherches pour de bons réseaux digitaux en base 2. Le développement de celui-ci a commencé pendant ma maîtrise et s'est poursuivi tout au long de mon doctorat. Il permet d'analyser l'uniformité de tous les générateurs dont il est question dans cette thèse. Il contient aussi des outils qui permettent de déterminer la période de générateurs.

Les recherches peuvent se faire sur des générateurs combinés ou des générateurs simples. On peut également chercher de bons (q, k, t)-réseaux avec des matrices $C^{(i)}$ quelconques. Différents critères d'uniformité peuvent être utilisés pour la recherche. Tous les critères utilisés dans cette thèse ont été implantés dans REGPOLY. La bibliothèque contient également des outils qui permettent de déterminer le polynôme caractéristique de la récurrence du générateur et déterminer si celui-ci est irréductible ou primitif. Les fonctions qui analysent des polynômes font appel à la bibliothèque ZEN [8] qui permet de faire des calculs dans des anneaux finis.

Depuis la version déposée à la fin de ma maîtrise, REGPOLY a subi de nombreux changements de conception. Dans la version originale, il fallait recompiler la bibliothèque chaque fois qu'on voulait analyser de gros générateurs. Les vecteurs de bits avaient une longueur fixe qui était déterminée par une constante. Si l'état du générateur à analyser dépassait cette constante, il fallait changer la constante et recompiler la bibliothèque. Dans la version actuelle, les vecteurs de bits peuvent avoir n'importe quelle longueur et cet inconvénient n'existe plus.

Dans la version originale, il n'était pas possible d'analyser l'uniformité de réseaux digitaux, d'automates cellulaires, ni des Mersenne twister. On ne pouvait pas analyser le polynôme caractéristique de n'importe quelle récurrence linéaire sur $\mathbb{F}_2$. Une attention particulière a été donnée aux implantations des algorithmes qui permettent de déterminer la primitivité d'un polynôme. En combinant plusieurs idées qui sont présentées à la fin du chapitre 2, on est arrivé à un implantation très efficace. Pour donner un exemple de l'efficacité de REGPOLY, dans [69], les auteurs affirment que, selon leurs expériences, il faudrait plusieurs années pour les algorithmes actuels pour trouver un générateur qui possède un polynôme caractéristique primitif et de degré supérieur à 10000. REGPOLY a réussi a trouver plusieurs générateurs de période $2^{19937} - 1$, $2^{21701} - 1$, $2^{23209} - 1$ et $2^{44497} - 1$; ces générateurs sont présentés au chapitre 7.

nant il y a, en plus de l'équidistribution, la q-valeur et la distance minimale. Pour ce qui est de l'équidistribution, la méthode basée sur le théorème 4.2 qui consiste à calculer le plus court vecteur dans le réseau dual du réseau en résolution engendré par un générateur a été implantée. Celle-ci a nécessité plus de six implantations expérimentales avant d'arriver à l'implantation finale. Pour donner un ordre de grandeur de l'amélioration de l'efficacité entre la première version et la dernière, pour calculer l'équidistribution du MT19937 [69], le temps de calcul est passé de plusieurs heures à moins de deux minutes.

Cette librairie est l'outil principal de recherche pour cette thèse. Elle a été utilisée avec succès pour obtenir plusieurs résultats dans le récents [40, 42, 45, 46, 53, 82, 85]. Dans cette thèse, REGPOLY est utilisé pour trouver de bons paramètres pour les générateurs dont il est question dans les prochains chapitres. Pour en savoir plus sur REGPOLY, on peut consulter le guide d'utilisation de la librairie [84].

Chapitre 5

Générateurs basés sur une récurrence dans $\mathbb{F}_{2^w}$

Dans ce chapitre, nous définissons deux récurrences dans le corps fini à 2^w éléments. Ces récurrences sont inspirées de la récurrence de base (2.1) et du GCL polynômial défini à la section 2.3.3. En fait, il s'agit des mêmes récurrences, sauf qu'on utilise le corps fini $\mathbb{F}_{2^w}$ au lieu de $\mathbb{F}_2$. Ces nouvelles récurrences entrent dans la famille des récurrences définies par l'équation (3.2). Les éléments de la matrice T, qui sont déterminés à partir du polynôme caractéristique de la récurrence, sont choisis de manière à ce que la multiplication de cette matrice par un vecteur $\mathbf{t} \in \mathbb{F}_{2^w}^r$ soit rapide. Le choix du corps fini $\mathbb{F}_{2^w}$ est justifié par le fait que l'analyse du réseau engendré par un générateur construit à partir de ces récurrence est relativement facile étant donné l'état des connaissances des réseaux polynômiaux. De plus, il existe des techniques d'exponentiation qui sont rapides pour le corps fini $\mathbb{F}_{2^w}$ et ses extensions (par exemple, $\mathbb{F}_{2^w}/P(z)$ où $P(z)$ est un polynôme irréductible sur $\mathbb{F}_{2^w}$). L'exponentiation permet de sauter en avant dans la récurrence d'un grand nombre d'itérations rapidement. Ceci est utile pour implanter un générateur qui permet l'utilisation de plusieurs flots de variables aléatoires où chaque flot est construit à partir de la même

séquence d'éléments dans $\mathbb{F}_{2^w}$, mais tous espacés d'un grand nombre d'itérations. C'est ce type d'implantation que nous avons dans notre bibliothèque informatique F2wStreams dont le guide est présenté en annexe C. Cette bibliothèque sera vue plus en détail à la fin de ce chapitre.

À partir des deux récurrences qu'on définit, on montre comment on peut produire des nombres aléatoires à l'aide des équations (1.1)-(1.4). De plus, on discute de l'équidistribution des ensembles de points produits par ces générateurs. Ces générateurs ont une propriété vraiment intéressante pour ce qui est des projections en 2 dimensions : la très grande majorité de celles-ci sont parfaitement équidistribuées. De plus, pour certains choix de paramètres, on déduit une borne sur l'équidistribution de $\Psi_t(\mathbf{X}, H, L)$ qui est une généralisation d'une borne sur l'équidistribution des TGFSR sans tempering qui est rapportée dans [102].

Également, plusieurs paramètres sont trouvés avec le logiciel REGPOLY et on applique des tests statistiques à ces générateurs. On présente des techniques d'implantation de ces générateurs et les programmes en langage C qui les implantent. Des temps d'exécution sont donnés pour ces implantations et on les compare à ceux de générateurs courants.

5.1 Registre à décalage à rétroaction linéaire dans $\mathbb{F}_{2^w}$

Soit la récurrence

$$m_n = \sum_{i=1}^{r} b_i m_{n-i}, \tag{5.1}$$

dans $\mathbb{F}_{2^w}$, où $m_n \in \mathbb{F}_{2^w}, n \geq 0$, r est un entier positif et $b_i \in \mathbb{F}_{2^w}$. Cette récurrence est la récurrence (30) de [75] et peut s'écrire sous la forme décrite par l'équation (3.2) où

$$T = T_{GCL} = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ b_r & b_{r-1} & \dots & b_2 & b_1 \end{pmatrix}. \tag{5.2}$$

et $\mathbf{t}_n = (m_{n-r+1}, \dots, m_{n-1}, m_n) \in \mathbb{F}_{2^w}^r$. Le polynôme caractéristique sur $\mathbb{F}_{2^w}$ de cette récurrence est le polynôme

$$P(z) = z^r - \sum_{i=1}^{r} b_i z^{r-i}. \tag{5.3}$$

La récurrence atteint la période maximale $2^{rw} - 1$ si et seulement si $P(z)$ est primitif sur $\mathbb{F}_{2^w}$ [75]. Comme dans le cas de la récurrence de base (2.1), on peut associer une fonction génératrice

$$G(z) = g(z)/P(z) = m_0 z^{-1} + m_1 z^{-2} + \cdots \tag{5.4}$$

à la séquence $\{m_n\}_{n\geq 0}$ où $g(z) \in \mathbb{F}_{2^w}[z]/P(z)$, étant donnée une initialisation $m_0, m_1, \dots, m_{r-1}$. Il est possible de sauter de j itérations en avant dans la récurrence (5.1) en effectuant $h(z) = z^j g(z) \bmod P(z)$. La fonction génératrice $H(z) = h(z)/P(z)$ nous donne explicitement l'état du générateur j itérations en avant. Une autre méthode est décrite à la section 5.10.

Pour implanter la récurrence (5.1), une base de représentation de $\mathbb{F}_{2^w}$ doit être sélectionnée (voir Annexe B). Dans ce cas, la récurrence (5.1) devient

$$\mathbf{v}_n = \sum_{i=0}^{r} A_{b_i} \mathbf{v}_{n-i} \tag{5.5}$$

où le vecteur $\mathbf{v}_n$ est le vecteur de w bits représentant m_n (dans la base choisie) et la matrice A_{b_i} de dimension $w \times w$ permet de multiplier un élément de $\mathbb{F}_{2^w}$ par b_i

(dans la base choisie). Habituellement, on utilise $\mathbf{y}_n = \mathbf{v}_n$ pour produire la sortie. On appelle ce type de générateur *registre à décalage à rétroaction linéaire* ou LFSR (de l'anglais « linear feedback shift register ») dans $\mathbb{F}_{2^w}$.

Le générateur basé sur cette récurrence utilise la méthode matricielle à récurrences multiples telle que décrite à la section 2.3.6. Pour définir un générateur de nombres aléatoires avec (5.5), on utilise les équations (1.1)–(1.4) avec la matrice de transition

$$X = T(\mathbb{F}_2) = \begin{pmatrix} & I_w & & & \\ & & I_w & & \\ & & & \ddots & \\ & & & & I_w \\ A_{b_r} & A_{b_{r-1}} & \dots & A_{b_2} & A_{b_1} \end{pmatrix} \tag{5.6}$$

où $T(\mathbb{F}_2)$ est la matrice de l'équation (3.6). L'état de la récurrence est

$$\mathbf{x}_n = (\mathbf{v}_{n-r+1}^\mathsf{T}, \dots, \mathbf{v}_{n-1}^\mathsf{T}, \mathbf{v}_n^\mathsf{T})^\mathsf{T}.$$

Exemple 5.1. La figure 5.1 illustre de quelle manière l'état du générateur est modifié pour un LFSR avec polynôme caractéristique $P(z) = z^7 + b_4 z^3 + b_7 \in \mathbb{F}_{2^w}[z]$. L'état indiqué est celui à l'itération n. En exécutant les opérations indiquées par les flèches, on obtient l'état à l'itération $n+1$. On voit que toutes les valeurs, sauf m_{n-6}, sont déplacées d'une position vers la gauche. La valeur de m_{n+1} est mise à la valeur $b_7 m_{n-6} + b_4 m_{n-3}$.

Exemple 5.2. Soit A, la matrice donnée par (2.29) et $Q(z) = \det(A - zI)$ le polynôme caractéristique de la matrice A. Supposons que $Q(z)$ soit irréductible et que $\zeta \in \mathbb{F}_{2^w}$ soit une de ses racines. On observe qu'un TGFSR (voir équation (2.14)) avec cette matrice A est un cas spécial d'un LFSR dans $\mathbb{F}_{2^w}$. Effectivement, si on utilise la base polynômiale et si le polynôme caractéristique $P(z)$ est tel que

$$b_i = \begin{cases} 1 & \text{si } i = r - m \\ \zeta & \text{si } i = r \\ 0 & \text{sinon,} \end{cases}$$

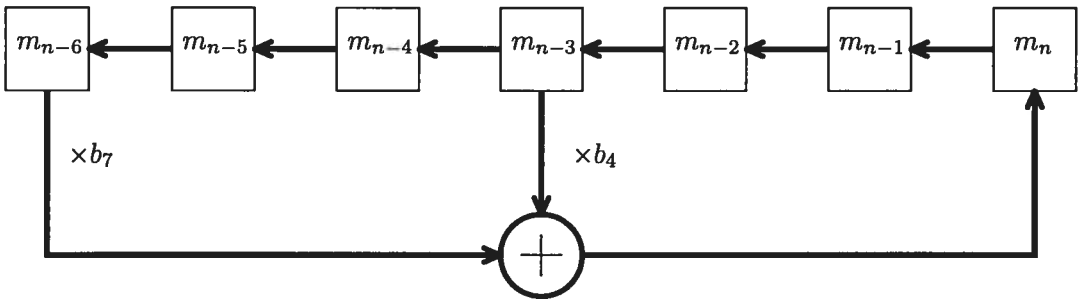


Figure 5.1 – Diagramme d'un LFSR.

alors le générateur résultant est un TGFSR. Le GCL polynômial de la section 2.3.3 est aussi un cas spécial car il utilise la récurrence $m_n = \zeta^s m_{n-1}$ avec la base polynômiale pour représenter $\mathbb{F}_{2^w}$.

Il est à remarquer que lorsqu'on applique un tempering sur ce type de générateur de la manière suivante : $\mathbf{y}_n = ((R\mathbf{v}_{n-r+1})^\mathsf{T}, \ldots, (R\mathbf{v}_n)^\mathsf{T})^\mathsf{T}$ où R est une matrice $w \times w$ de plein rang (comme c'est fait pour le TGFSR dans [67]), tout ce que l'on fait, c'est changer la base de représentation des éléments de $\mathbb{F}_{2^w}$. Étant donné que le tempering change l'uniformité de la séquence des nombres produits par le générateur, ceci suggère que le choix de la base est important pour l'uniformité des ensembles de points produits par le générateur.

5.2 GCL polynômial dans $\mathbb{F}_{2^w}[z]/P(z)$

Soit $P(z) \in \mathbb{F}_{2^w}[z]$, le polynôme de degré r tel que défini à l'équation (5.3) et $\mathbb{F}_{2^w}[z]/P(z)$, l'anneau des polynômes dans $\mathbb{F}_{2^w}[z]$ modulo $P(z)$ (si $P(z)$ est irréductible sur $\mathbb{F}_{2^w}$, alors $\mathbb{F}_{2^w}[z]/P(z)$ est aussi un corps fini). Soit la récurrence

$$q_n(z) = zq_{n-1}(z) \bmod P(z) \tag{5.7}$$

où pour chaque $n \geq 0$, $q_n(z) = q_{n,1}z^{r-1} + \cdots + q_{n,r-1}z + q_{n,r} \in \mathbb{F}_{2^w}[z]/P(z)$. En divisant cette récurrence par $P(z)$, on obtient

$$q_n(z)/P(z) = zq_{n-1}(z)/P(z) \bmod 1 \tag{5.8}$$

de laquelle on peut écrire

$$q_n(z)/P(z) = \sum_{j=1}^{\infty} x_{n+j} z^{-j}$$

où $\{x_j\}_{j\geq 1}$ est une séquence qui suit la récurrence (5.1). Dans un sens, ceci signifie que (5.1), (5.7) et (5.8) sont toutes des représentations différentes de la même récurrence. La récurrence (5.7) peut s'écrire également sous la forme (3.2) avec

$$T = \begin{pmatrix} & & & b_r \\ 1 & & & b_{r-1} \\ & 1 & & \vdots \\ & & \ddots & b_2 \\ & & 1 & b_1 \end{pmatrix} \tag{5.9}$$

et $\mathbf{t}_n = (q_{n,r}, \ldots, q_{n,1}) \in \mathbb{F}_{2^w}^r$. Pour déduire cette matrice de transition, examinons plus en détail comment la récurrence transforme les vecteurs de la base canonique. Si $q_n(z) = z^i$ où $0 \leq i < r-1$, alors $q_{n+1}(z) = z^{i+1}$. De plus, si $q_n(z) = z^{r-1}$, alors $q_{n+1}(z) = \sum_{i=1}^{r} b_i z^{i-1}$. Ces observations nous permettent de déduire complètement la matrice T ci-haut. En comparant cette matrice de transition avec (5.2) (celle du LSFR dans $\mathbb{F}_{2^w}$), on remarque que l'une est la transposée de l'autre.

Le polynôme caractéristique sur $\mathbb{F}_{2^w}$ de chacune de ces récurrences est évidemment $P(z)$. Pour « sauter » en avant de e itérations dans la récurrence, on effectue

$$q_{n+e}(z) = z^e q_n(z) \bmod P(z).$$

Un algorithme donné à la section 5.10 permet de faire ce calcul efficacement.

Pour implanter un générateur basé sur la récurrence (5.7), une base de représentation de $\mathbb{F}_{2^w}$ doit être choisie, comme pour le LFSR. Soit $\mathbf{q}_{n,i}$, la représentation dans

la base choisie du coefficient $q_{n,i}$ de $q_n(z)$. Soit $\mathbf{x}_n = (\mathbf{q}_{n,r}, \ldots, \mathbf{q}_{n,1})$, le vecteur d'état du générateur. En remplaçant b_i par A_{b_i} et 1 par I_w dans la matrice T, on obtient

$$X = T(\mathbb{F}_2) = \begin{pmatrix} & & & A_{b_r} \\ I_w & & & A_{b_{r-1}} \\ & I_w & & \vdots \\ & & \ddots & A_{b_2} \\ & & I_w & A_{b_1} \end{pmatrix}.$$

On définit un générateur basé sur (5.7) en utilisant les équations (1.1)-(1.4) avec la matrice X définie ci-haut. Ce type de générateur est appelé *générateur à congruence linéaire (GCL) polynômial dans* $\mathbb{F}_{2^w}[z]/P(z)$

On peut implanter la multiplication par la matrice X par

$$\mathbf{x}_{n+1} = X\mathbf{x}_n = (\mathbf{0}, \mathbf{q}_{n,r}, \ldots, \mathbf{q}_{n,2}) \quad + \quad (A_{b_r}, \ldots, A_{b_1})\mathbf{q}_{n,1}. \tag{5.10}$$

Exemple 5.3. La figure 5.2 montre comment l'état d'un GCL polynômial est modifié. Elle illustre un GCL polynômial avec polynôme caractéristique $P(z) = z^7 + b_4 z^3 + b_7$ (comme à la figure 5.1). On observe que les valeurs sont décalées vers la droite d'une position. La valeur de $q_{n+1,7}$ est mise à $b_7 q_{n,1}$ et la valeur de $q_{n+1,4}$, à $q_{n,5} + b_4 q_{n,1}$.

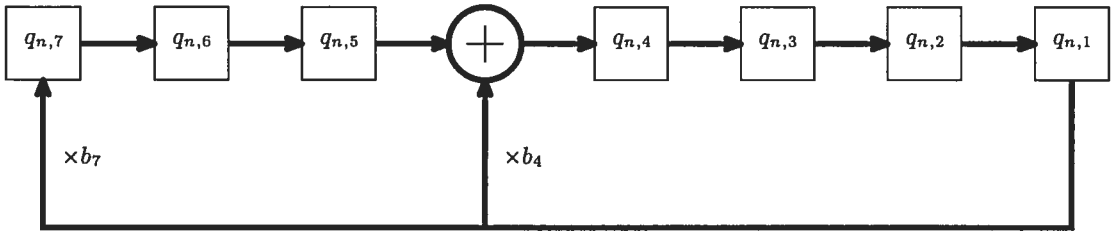


Figure 5.2 – Diagramme d'un GCL polynômial.

Lors d'un atelier sur les générateurs de nombres pseudo-aléatoires, le professeur Makoto Matsumoto a fait remarquer à l'auteur que dans le cas du LFSR, on observe

plusieurs valeurs dans $\mathbb{F}_{2^w}$ de l'état pour n'en modifier qu'une seule. Dans le cas d'un GCL polynômial, on n'observe qu'une seule valeur dans $\mathbb{F}_{2^w}$ de l'état pour en modifier plusieurs. Il a de plus suggéré que l'on pourrait construire des récurrences qui examinent plusieurs valeurs pour en modifier plusieurs. C'est en partant de cette idée qu'on a développé les générateurs qui seront proposés au chapitre 7.

5.3 Réseau en résolution décrit par les récurrences dans $\mathbb{F}_{2^w}$

Dans cette section, on examine les réseaux en $(\mathbb{F}_{2^w}, \nu)$-résolution des générateurs décrits dans les sections 5.1 et 5.2. Ceci prépare la prochaine section où l'on explique comment produire plus d'une valeur aléatoire par itération pour le GCL polynômial. Elle permet de mettre en évidence les différences principales entre les GCL polynômiaux et les LFSR dans $\mathbb{F}_{2^w}$.

Les récurrences

$$m_n = \sum_{i=1}^{r} b_i m_{n-i}, \tag{5.11}$$

et

$$q_n(z) = z\, q_{n-1}(z) \bmod P(z), \tag{5.12}$$

sont des cas spéciaux de (3.2) quand $q = 2^w$. Pour la récurrence (5.11), la matrice de transition est

$$T_{LFSR} = \begin{pmatrix} & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ b_r & b_{r-1} & \dots & b_2 & b_1 \end{pmatrix} \tag{5.13}$$

et $\mathbf{t}_n = (m_{n-r+1}, \dots, m_n)$. Pour (5.12), la matrice de transition, donnée en (5.2), est

$T_{GCL} = T_{LFSR}^{\mathsf{T}}$ et $\mathbf{t}_n = (q_{n,r}, \ldots, q_{n,1})$.

Pour chacune de ces récurrences, on peut définir des réseaux en $(\mathbb{F}_{2^w}, \nu)$-résolution $\mathcal{L}_\nu(T_{LFSR})$ ou $\mathcal{L}_\nu(T_{GCL})$. Par la définition 3.2, une base possible de $\mathcal{L}_\nu(T_{LFSR})$ ou de $\mathcal{L}_\nu(T_{GCL})$ est telle que $\mathbf{v}_j(z) = \mathbf{e}_j$ pour $j = 1, \ldots, \nu-1$ où $\mathbf{e}_j$ est le $(j+1)$-ième vecteur de la base canonique. Le vecteur $\mathbf{v}_0(z)$ pour le réseau $\mathcal{L}_\nu(T_{LFSR})$ est $\bar{\mathbf{G}}_\nu(T_{LFSR})$ et il est $\bar{\mathbf{G}}_\nu(T_{GCL})$ pour le réseau $\mathcal{L}_\nu(T_{GCL})$. Les deux théorèmes suivants nous indiquent leur forme.

Théorème 5.1. *Soit $T = T_{LFSR}$ la matrice définie à l'équation (5.6) dont le polynôme caractéristique $P(z)$ est irréductible sur $\mathbb{F}_{2^w}$. On a*

$$\bar{\mathbf{G}}_\nu(T) = (1, z \bmod P(z), \ldots, z^{\nu-1} \bmod P(z))/P(z).$$

Démonstration. Soit

$$\mathbf{t}_n = (t_n^{(0)}, \ldots, t_n^{(r-1)}) = (m_{n-r+1}, \ldots, m_n),$$

et

$$\mathbf{G}_\nu(T, \mathbf{t}_0) = (g_0(z), \ldots, g_{\nu-1}(z))/P(z)$$

pour une initialisation $\mathbf{t}_0$ fixée.

En observant la récurrence $\mathbf{t}_n = T\mathbf{t}_{n-1}$, on remarque que

$$t_{n+1}^{(j)} = t_n^{(j+1)} \tag{5.14}$$

pour $j = 0, \ldots, r-2$. On sait que

$$G_j(z) = g_j(z)/P(z) = \sum_{n \geq 0} t_n^{(j)} z^{-n-1} \tag{5.15}$$

pour $j = 1, \ldots, r-1$. En combinant les équations (5.14) et (5.15), on obtient

$$\begin{aligned}
\sum_{n\geq 0} t_n^{(j+1)} z^{-n-1} &= \sum_{n\geq 0} t_{n+1}^{(j)} z^{-n-1} \\
&= \sum_{n\geq 1} t_n^{(j)} z^{-n} \\
G_{j+1}(z) &= zG_j(z) - t_0^{(j)} \\
g_{j+1}(z)/P(z) + t_0^{(j)} &= zg_j(z)/P(z) \\
g_{j+1}(z) &\equiv zg_j(z) \bmod P(z)
\end{aligned}$$

pour $j = 0, \ldots, r-2$. La dernière équation est valide car $\deg(g_{j+1}(z)) < \deg(P(z))$. On obtient de cette dernière équation $g_j(z) = g_0(z)z^j$ pour $j = 1, \ldots, r-1$. En divisant $g_j(z)$ par $g_0(z)$ dans $\mathbb{F}_{2^w}[z]/P(z)$ pour $j = 0, \ldots, r-1$, on obtient

$$\bar{\mathbf{G}}_\nu(T) = (1, z \bmod P(z), \ldots, z^{\nu-1} \bmod P(z))/P(z).$$

□

Théorème 5.2. *Soit $T = T_{GCL}$, la matrice définie par l'équation (5.9) dont le polynôme caractéristique $P(z)$ est irréductible. Soit*

$$\mathbf{G}_r(T, (q_{0,r}, \ldots, q_{0,1})) = (g_0(z), \ldots, g_{r-1}(z))/P(z).$$

Les $g_j(z)$ sont inter-connectés par les équations

$$zg_0(z) \equiv b_r g_{r-1}(z) \bmod P(z) \tag{5.16}$$

$$zg_{r-j}(z) \equiv b_j g_{r-1}(z) + g_{r-j-1}(z) \bmod P(z) \quad j = 1, \ldots, r-1. \tag{5.17}$$

Démonstration. Soit

$$G_{r-j}(z) = g_{r-j}(z)/P(z) = \sum_{n\geq 0} q_{n,j} z^{-n-1}$$

pour $j = 1, \ldots, r$. De la matrice T_{GCL}, on observe que $q_{n+1,r} = b_r q_{n,1}$ pour $n \geq 0$. On obtient de cela

$$\begin{aligned}
\sum_{n\geq 0} q_{n+1,r} z^{-n-1} &= b_r \sum_{n\geq 0} q_{n,1} z^{-n-1} \\
\sum_{n\geq 1} q_{n,r} z^{-n} &= b_r G_{r-1}(z) \\
-q_{0,r} + \sum_{n\geq 0} q_{n,r} z^{-n} &= b_r G_{r-1}(z) \\
-q_{0,r} + z G_0(z) &= b_r G_{r-1}(z) \\
-q_{0,r} P(z) + z g_0(z) &= b_r g_{r-1}(z) \\
z g_0(z) &\equiv b_r g_{r-1}(z) \bmod P(z),
\end{aligned}$$

ce qui démontre l'équation (5.16). De plus, de la matrice T_{GCL}, on remarque que $q_{n+1,j} = q_{n,j+1} + b_j q_{n,1}$ pour $n \geq 0$ et $j = 1, \ldots, r-1$. On obtient

$$\begin{aligned}
\sum_{n\geq 0} q_{n+1,j} z^{-n-1} &= \sum_{n\geq 0} q_{n,j+1} z^{-n-1} + b_j \sum_{n\geq 0} q_{n,1} z^{-n-1} \\
\sum_{n\geq 1} q_{n,j} z^{-n} &= G_{r-j-1}(z) + b_j G_{r-1}(z) \\
-q_{n,0} + z G_{r-j}(z) &= G_{r-j-1}(z) + b_j G_{r-1}(z) \\
z g_{r-j}(z) &= g_{r-j-1}(z) + b_j g_{r-1}(z) + q_{n,0} P(z) \\
z g_{r-j}(z) &\equiv g_{r-j-1}(z) + b_j g_{r-1}(z) \bmod P(z),
\end{aligned}$$

ce qui démontre (5.17). □

De ce théorème, il est facile de trouver une base pour le réseau polynômial $\mathcal{L}_\nu(T_{GCL})$. En fixant $g_0(z) = 1$, on obtient le vecteur $\bar{\mathbf{G}}_\nu(T_{GCL})$ à partir des équations (5.16) et (5.17).

Un *réseau polynômial de Korobov* est un réseau polynomial dont une base possible est définie par $\{\mathbf{v}_0(z), \ldots, \mathbf{v}_{\nu-1}(z)\}$ où

$$\mathbf{v}_0(z) = (1, a(z), \ldots, a(z)^{\nu-1})/P(z)$$

et $\mathbf{v}_i(z)$ est le i-ième vecteur de la base canonique pour $i = 2, \ldots, \nu$. Il est intéressant de noter que le réseau $\mathcal{L}_\nu(T_{GCL})$ est rarement un réseau polynômial de Korobov et que le réseau $\mathcal{L}_\nu(T_{LFSR})$ en est toujours un avec $a(z) = z$.

Exemple 5.4. Revenons à la matrice T définie à l'exemple 3.1. On remarque que cette matrice correspond à la matrice T d'un GCL polynômial (voir l'équation (5.9)). Pour cette matrice, on avait calculé que $\bar{\mathbf{G}}_3(T) = (1, (1+\zeta)z^2 + \zeta z, (1+\zeta)z)/P(z)$ où

$$P(z) = z^3 + b_1 z^2 + b_2 z + b_3 = z^3 + (1+\zeta)z^2 + \zeta z + \zeta.$$

Soit $\mathbf{t}_0$, l'initialisation telle que $\mathbf{G}_3(T, \mathbf{t}_0) = (g_0(z), g_1(z), g_2(z))/P(z) = \bar{\mathbf{G}}_3(T)$. Dans ce cas, il est facile de vérifier l'équation (5.16) :

$$\begin{aligned} z g_0(z) &\equiv b_3 g_2(z) && \bmod P(z) \\ z &\equiv \zeta(1+\zeta)z && \bmod P(z) \\ z &\equiv z && \bmod P(z). \end{aligned}$$

On peut aussi le faire pour l'équation (5.17) :

$$\begin{aligned} z g_2(z) &\equiv b_1 g_2(z) + g_1(z) && \bmod P(z) \\ z(1+\zeta)z &\equiv (1+\zeta)(1+\zeta)z + (1+\zeta)z^2 + \zeta z && \bmod P(z) \\ z^2 + \zeta z^2 &\equiv z + \zeta z + \zeta z + (1+\zeta)z + (1+\zeta)z^2 + \zeta z && \bmod P(z) \\ z^2 + \zeta z^2 &\equiv (1+\zeta)z^2 && \bmod P(z) \end{aligned}$$

et

$$\begin{aligned} z g_1(z) &\equiv b_2 g_2(z) + g_0(z) && \bmod P(z) \\ z((1+\zeta)z^2 + \zeta z) &\equiv \zeta(1+\zeta)z + 1 && \bmod P(z) \\ z + 1 &\equiv z + 1 && \bmod P(z). \end{aligned}$$

5.4 Générateur à sorties multiples

Il ressort de l'étude du réseau polynômial en $(\mathbb{F}_{2^w}, \nu)$-résolution $\mathcal{L}_\nu(T^\mathsf{T})$ que, si $P(z)$ est primitif sur $\mathbb{F}_{2^w}$, les séquences $S_i = \{q_{n,i}\}_{n \geq 0}$ et $S_j = \{q_{n,j}\}_{n \geq 0}$ pour $i \neq j$

sont les mêmes, mais pas leurs points de départs. Dans cette section, on remarque qu'en fait leurs points de départs sont souvent extrêmement éloignés l'un de l'autre pour certaines valeurs de i et de j. Nous montrons comment tirer profit de cela pour produire plusieurs nombres aléatoires à chaque itération d'un GCL polynômial.

Soit $G_i(z) = g_i(z)/P(z)$, la fonction génératrice de S_i. On définit le *délai relatif* $\hat{\sigma}(i,j)$ de la séquence S_i par rapport à la séquence S_j comme le plus petit entier d tel que $q_{n+d,i} = q_{n,j}$ pour tout $n \geq 0$, c'est-à-dire

$$\hat{\sigma}(i,j) = \min\{d : d > 0, z^d g_{r-i}(z) \equiv g_{r-j}(z) \bmod P(z)\}.$$

On définit le *délai absolu* $\sigma(i,j)$ par

$$\sigma(i,j) = \min\{\hat{\sigma}(i,j), \hat{\sigma}(j,i)\}.$$

À noter que cette dernière équation implique un logarithme discret dans $\mathbb{F}_{2^{rw}}$. Trouver $\hat{\sigma}(i,j)$ est un problème très difficile à résoudre quand rw est grand. Par exemple, le record, quant à la taille du corps fini, pour la résolution de logarithmes discrets (qui soient non triviaux) dans un corps de caractéristique 2 s'est fait dans $\mathbb{F}_{2^{607}}$[105]. Ce record a été effectué après avoir utilisé plus de 19000 MIPS-années, une quantité énorme d'opérations, pour calculer des tables de pré-calculs. Avec ces tables, il est maintenant possible d'effectuer n'importe quel logarithme discret dans $\mathbb{F}_{2^{607}}$ en moins de trois heures. Pour les générateurs pour lesquels on recherche de bons paramètres à la section 5.8, trouver le délai $\sigma(i,j)$ implique un logarithme discret dans $\mathbb{F}_{2^{96}}$, $\mathbb{F}_{2^{256}}$, $\mathbb{F}_{2^{400}}$ et $\mathbb{F}_{2^{800}}$, ce qui est évidemment très difficile à calculer.

De l'équation (5.17), on observe que pour un indice j tel que $b_j = 0$, on a que $zg_{r-j}(z) \equiv g_{r-j-1}(z) \bmod P(z)$. Dans ce cas, les séquences S_j et S_{j+1} ont un délai absolu $\sigma(j, j+1) = 1$. Pour un indice j tel que $b_j \neq 0$, on a $zg_{r-j}(z) \equiv b_j g_{r-1}(z) + g_{r-j-1}(z) \bmod P(z)$. Dans ce cas, le délai $\sigma(j+1, j)$ est difficile à calculer quand 2^{rw} est grand.

La difficulté vient du fait qu'il est impossible de prédire la valeur de d telle que $z^d g_{r-j}(z) \equiv g_{r-j-1}(z) \bmod P(z)$ à partir des expressions de $g_{r-j}(z)$ et $g_{r-j-1}(z)$. La sécurité de certains algorithmes d'encryption repose sur la propriété que toutes les valeurs de d sont équiprobables et qu'il n'existe pas d'algorithme efficace pour trouver la valeur de d [72]. Dans ce qui suit supposons que d soit choisi au hasard dans $\{0, \ldots, 2^{rw} - 1\}$. Dans ce cas, si $b_j \neq 0$, on a que $\sigma(j+1, j)$ est uniformément distribué dans $\{0, \ldots, \lfloor (2^{rw} - 1)/2 \rfloor\}$ et

$$\Pr[\sigma(j+1, j) \leq 2^h] \approx 2^h / 2^{rw-1}.$$

Cette dernière équation indique que la probabilité que le délai entre S_j est S_{j+1} soit plus petit que 2^h est 2^{h-rw+1}. Cette valeur est très petite, même pour des valeurs de h moyennement près de rw. Par exemple, si $h = 200$ et $rw = 256$, on a

$$\Pr[\sigma(i, j) \leq 2^h] \approx 2^{200-255} = 2^{-55},$$

ce qui est extrêmement petit. Ainsi, on peut considérer, avec une relative certitude, que le délai $\sigma(j, j+1)$ est presque toujours gigantesque quand $b_j \neq 0$.

Cette analyse serait juste si d était aléatoire, mais ce n'est pas le cas. Par contre, si d est prouvé être très grand, alors on peut poursuivre notre analyse.

Soit $J = \{i_1, \ldots, i_m\}$, un ensemble d'entiers tels que $1 \leq i_j \leq r$ pour $j = 1, \ldots, m$. Soit la fonction

$$f(J, h) = \begin{cases} 1 & \text{si } \sigma(i, j) > 2^h \ \ \forall i, j \in J, i \neq j \\ 0 & \text{sinon.} \end{cases}$$

Soit $J^*(h) = \arg\max_J \{|J| : f(J, h) = 1\}$, le plus grand ensemble d'indices $J = \{i_1, \ldots, i_m\}$ tel que le délai $\sigma(i_j, i_k) > 2^h$ quand $j \neq k$ et $1 \leq j, k \leq m$.

Souvent, ce qui risque de se produire c'est que $J^*(h) = \{j : b_j \neq 0\}$. Quand h est suffisamment grand, on pourrait considérer chaque séquence de nombres aléatoires produite par $\{\mathbf{y}_n = \mathbf{q}_{n,j}\}_{n \geq 0}$, où $j \in \{j : b_j \neq 0\}$, comme une source de nombres

aléatoires uniformes et indépendants. L'indépendance n'est pas nécessairement garantie par Le fait que les points de départ de chacune des séquences sont très éloignés les uns des autres ne garantit pas l'indépendance, mais nous porte à croire que c'est bien le cas. Par exemple, si $P(z)$ est un trinôme, le GCL polynômial correspondant peut produire 2 nombres aléatoires par itération.

Le fait de considérer deux ou plusieurs séquences de nombres pseudo-aléatoires provenant d'une même séquence n'est pas nouveau. En effet, dans [49], on segmente la séquence produite par un certain générateur en sous-séquences. Les débuts de chaque sous-séquence sont tellement éloignés les uns des autres que celles-ci sont considérées indépendantes statistiquement. De cette manière, un seul générateur peut être utilisé comme étant plusieurs générateurs à la fois. Un générateur qui supporte ce type d'implantation est appelé *générateur à plusieurs flots de nombres aléatoires* [49] et un *flot* est une sous-séquence de la séquence globale produite par le générateur.

Ce qu'il y a de nouveau avec cette approche, c'est qu'on peut produire plusieurs nombres aléatoires « statistiquement indépendants » à chacune des itérations. Un GCL polynômial dont le polynôme caractéristique a quatre coefficients non nuls produit, à chaque itération, un vecteur aléatoire uniformément distribué en trois dimension (u, v, w).

Puisque les délais $\sigma(i, j)$ sont inconnus et très difficiles à calculer, il se pourrait que certains soient relativement petits sans qu'on ne le sache. Pour pallier à cet éventuel problème, on pourrait appliquer un tempering différent à chaque sortie du générateur. Ceci ferait en sorte que les séquences produites seraient différentes et que le risque d'avoir des corrélations entre les différentes sorties serait minimisé.

Nous jugeons l'idée de générateur à sorties multiples valable, mais le développement de cette idée, dans cette thèse, s'arrêtera à cette section. Il faudrait réfléchir plus longuement afin de garantir l'indépendance statistique entre les différentes sorties

(par exemple, en trouvant une méthode pour garantir que les délais $\sigma(i,j)$ soient plus grands qu'une certaine valeur ou en trouvant des générateurs dont le vecteur des différentes sorties soit bien équidistribué). Nous nous réservons ce travail pour plus tard, une fois cette thèse déposée. Par contre, dans les implantations en langage C des GCL polynômiaux, nous les donnons pour le cas où on considère des générateurs à sorties multiples afin de pouvoir mesurer la vitesse d'exécution de ces éventuels générateurs.

5.5 Limitations sur l'équidistribution

L'équidistribution de l'ensemble de points $\Psi_t(\mathbf{X}, H, L)$, où $H = I_{Lk}$, produit par les LFSR et les GCL polynômiaux dans $\mathbb{F}_{2^w}$ est déterminée par les coefficients du polynôme caractéristique $P(z) \in \mathbb{F}_{2^w}[z]$ et la base choisie pour représenter $\mathbb{F}_{2^w}$. Pour la base polynômiale, en expérimentant avec des générateurs dont le polynôme caractéristique $P(z)$ a des coefficients d'une certaine forme, on a observé certaines limitations sur l'équidistribution de $\Psi_t(\mathbf{X}, H, L)$. Dans cette section, on déduit une borne sur la valeur de t_ℓ qui dépend des coefficients de $P(z)$ (et non de leur nombre). Celle-ci est obtenue en examinant les réseaux en $(\mathbb{F}_2, \ell)$-résolution produits par le générateur. Ceux-ci sont $\mathcal{L}_\ell(X)$ pour $\ell = 1, \ldots, rw$.

Soit $P(z) \in \mathbb{F}_{2^w}[z]$, le polynôme caractéristique de la matrice T de l'équation (3.2). Pour le prochain théorème, on décompose le polynôme $P(z) \in \mathbb{F}_{2^w}[z]$ de l'équation (5.3) sous la forme

$$P(z) = p_0(z) + \zeta p_1(z) + \cdots + \zeta^{\gamma^*} p_{\gamma^*}(z) \tag{5.18}$$

où $p_i(z) \in \mathbb{F}_2[z]$ pour $0 \leq i \leq \gamma^*$ et $\zeta \in \mathbb{F}_{2^w}$ Par exemple, si $P(z) = (1+\zeta) + \zeta z + (1+\zeta+\zeta^2)z^5$, alors $p_0(z) = 1 + z^5$, $p_1(z) = 1 + z + z^5$, $p_2(z) = z^5$ et $\gamma^* = 2$. Ceci veut dire que chacun des coefficients b_i de $P(z)$ est de la forme $\sum_{i=0}^{\gamma^*} c_i \zeta^i$ où $c_i \in \mathbb{F}_2$.

Théorème 5.3. *Soit la récurrence* $\mathbf{t}_n = T\mathbf{t}_{n-1}$ *où* T *est une matrice* $r \times r$ *d'éléments dans* $\mathbb{F}_{2^w}$ *et* $P(z) \in \mathbb{F}_{2^w}[z]$, *son polynôme caractéristique qui est irréductible sur* $\mathbb{F}_{2^w}$. *Soit* $\bar{\mathbf{t}}_n^{(0)} \in \mathbb{F}_2^w$, *un vecteur qui représente* $t_n^{(0)}$ *dans la base polynômiale* $(1, \zeta, \ldots, \zeta^{w-1})$ *où* ζ *est la racine d'un polynôme irréductible* $Q(z) \in \mathbb{F}_2[z]$. *Soit le générateur défini par cette récurrence et la sortie* $u_n = \sum_{i=1}^{w} y_n^{(i-1)} 2^{-i}$ *où* $\mathbf{y}_n = \bar{\mathbf{t}}_n^{(0)}$. *Soit* $P(z) = p_0(z) + \zeta p_1(z) + \cdots + \zeta^{\gamma^*} p_{\gamma^*}(z)$, *une décomposition du polynôme* $P(z)$ *où* $p_i(z) \in \mathbb{F}_2[z]$ *pour* $i = 0, \ldots, \gamma^*$. *L'équidistribution de ce générateur est telle que*

$$t_{\gamma^*+1} \leq r.$$

Ceci implique que $t_\ell \leq r$ *pour* $\ell > \gamma^*$.

Démonstration. L'idée de la preuve consiste à trouver un vecteur dans le réseau dual $\mathcal{L}^*_{\gamma^*+1}(T(\mathbb{F}_2))$ dont la norme est inférieure à 2^r. Ceci compléterait la démonstration par le théorème 4.2.

Pour ce générateur c'est la séquence $\{t_n^{(0)}\}_{n\geq 0}$ qui nous donne la sortie du générateur. Par le lemme 3.2 et la définition 3.1, on sait que $\bar{\mathbf{G}}_1(T) = 1/P(z)$. Soit $\mathbf{v}_0(z) = \bar{\mathbf{G}}_\ell(T(\mathbb{F}_2)) = (h_0(z), \ldots, h_{\ell-1}(z))/M(z)$ où $M(z) \in \mathbb{F}_2[z]$ est le polynôme minimal de la séquence $\{m_n\}_{n\geq 0}$ sur $\mathbb{F}_2$. À noter que $P(z)$ divise $M(z)$ dans $\mathbb{F}_{2^w}[z]$.

Soit ζ, un générateur de $\mathbb{F}_{2^w}$, dont $Q(z) \in \mathbb{F}_2[z]$ est le polynôme minimal (dans ce cas, $Q(z)$ est nécessairement irréductible). On utilise la base polynômiale

$$(\beta_1, \ldots, \beta_w) = (1, \zeta, \ldots, \zeta^{w-1})$$

pour représenter les éléments de $\mathbb{F}_{2^w}$. Des équations (3.9) et (3.10), on obtient

$$\begin{aligned} 1/P(z) &= (\beta_1 h_0(z) + \cdots + \beta_w h_{w-1}(z))/M(z) \\ M(z) &= (h_0(z) + \zeta h_1(z) \cdots + \zeta^{w-1} h_{w-1}(z))P(z). \end{aligned} \tag{5.19}$$

De cette dernière équation, on a

$$(\tilde{h}_0(z) + \zeta \tilde{h}_1(z) + \cdots + \zeta^{w-1}\tilde{h}_{w-1}(z))(p_0(z) + \cdots + \zeta^{\gamma^*} p_{\gamma^*}(z)) \equiv 0 \bmod M(z)$$

où $\tilde{h}_i(z) \equiv h_i(z)h_0(z)^{-1} \bmod M(z)$ pour $i = 0, \ldots, w-1$. On obtient alors

$$\sum_{j=0}^{w-1} \zeta^j \sum_{i=\max(0,j-\gamma^*)}^{j} p_{j-i}(z)\tilde{h}_i(z) + \sum_{j=w}^{w+\gamma^*-1} \zeta^j \sum_{i=j-\gamma^*}^{w-1} p_{j-i}(z)\tilde{h}_i(z) \equiv 0 \bmod M(z). \quad (5.20)$$

Soit

$$m_j(z) = \sum_{i=j-\gamma^*}^{w-1} p_{j-i}(z)\tilde{h}_i(z) \ \in \mathbb{F}_2[z]$$

pour $j = w, \ldots, w+\gamma^*-1$. Également, soit $\zeta^j = \delta_0^j + \delta_1^j\zeta + \ldots + \delta_{w-1}^j\zeta^{w-1}$ pour $j = w, \ldots, w+\gamma^*-1$. Dans cette dernière équation, les $\delta_i^j \in \mathbb{F}_2$ dépendent du polynôme $Q(z) \in \mathbb{F}_2[z]$.

On peut réécrire (5.20) comme

$$\sum_{j=0}^{w-1} \zeta^j \sum_{i=\max(0,j-\gamma^*)}^{j} p_{j-i}(z)\tilde{h}_i(z) + \sum_{j=w}^{w+\gamma^*-1} (\delta_0^j+\delta_1^j\zeta+\ldots+\delta_{w-1}^j\zeta^{w-1})m_j(z) \equiv 0 \bmod M(z).$$

Cette dernière équation peut être séparée en w équations, une pour chaque ζ^j, $0 \leq j \leq w-1$. On peut faire cela puisque $M(z)$ n'a aucune composante en ζ^j pour $j > 0$ et que les ζ^j sont linéairement indépendants sur $\mathbb{F}_2[z]$. Chaque ligne du tableau suivant correspond à une de ces équations et chaque colonne correspond au coefficient de $\tilde{h}_j(z), 0 \leq j \leq w-1$ ou $m_j(z), w \leq j \leq w+\gamma^*-1$. Remarquons qu'il s'agit d'un système de w équations linéaires homogène dans $\mathbb{F}_2[z]/M(z)$.

$\tilde{h}_0(z)$	$\tilde{h}_1(z)$	$\ldots$	$\tilde{h}_{\gamma^*}(z)$	$\tilde{h}_{\gamma^*+1}(z)$	$\ldots$	$\tilde{h}_{w-1}(z)$	$m_w(z)$	$\ldots$	$m_{w+\gamma^*-1}(z)$	mod $M(z)$
$p_0(z)$	0	$\ldots$	0	0	$\ldots$	0	δ_0^w	$\ldots$	$\delta_0^{w+\gamma^*-1}$	0
$p_1(z)$	$p_0(z)$	$\ldots$	0	0	$\ldots$	0	δ_1^w	$\ldots$	$\delta_1^{w+\gamma^*-1}$	0
$\vdots$	$\vdots$	$\ddots$	$\vdots$	0	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	0
p_{γ^*}	p_{γ^*-1}	$\ldots$	$p_0(z)$	0	$\ldots$	0	$\delta_{\gamma^*}^w$	$\ldots$	$\delta_{\gamma^*}^{w+\gamma^*-1}$	0
0	p_{γ^*}	$\ldots$	$p_1(z)$	$p_0(z)$	$\ldots$	0	$\delta_{\gamma^*+1}^w$	$\ldots$	$\delta_{\gamma^*+1}^{w+\gamma^*-1}$	0
$\vdots$	$\vdots$	$\ddots$	$\vdots$	0	$\ddots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	0
0	0	$\ldots$	0	0	$\ldots$	$p_{\gamma^*}(z)$	δ_{w-1}^w	$\ldots$	$\delta_{w-1}^{w+\gamma^*-1}$	0

En sélectionnant quelques des (γ^*+1) premières lignes, il est possible d'obtenir une combinaison linéaire sur $\mathbb{F}_2$ des lignes sélectionnées de telle manière que les coefficients des polynômes $m_w(z)$ à $m_{w+\gamma^*-1}$ soient tous nuls. Ceci est vrai puisque la sous-matrice

composée des $(\gamma^* + 1)$ premières lignes et les γ^* dernières colonnes ne peut contenir $(\gamma^* + 1)$ lignes linéairement indépendantes. Soit

$$r_0(z)\tilde{h}_0(z) + r_1(z)\tilde{h}_1(z) + \cdots + r_{\gamma^*}(z)\tilde{h}_{\gamma^*}(z) \equiv 0 \bmod M(z),$$

le résultat de cette combinaison linéaire. Notons que chaque $r_i(z)$ est une combinaison linéaire sur $\mathbb{F}_2$ des polynômes $p_0(z), \ldots, p_{\gamma^*}$ ce qui veut dire que $\deg(r_i(z)) \leq \deg(P(z))$ pour $i = 0, \ldots, \gamma^*$.

Par la proposition 3.2, une base possible pour le réseau dual $\mathcal{L}^*_{\gamma^*+1}(T(\mathbb{F}_2))$ de $\mathcal{L}_{\gamma^*+1}(T(\mathbb{F}_2))$ est $\{\mathbf{v}_0(z), \ldots, \mathbf{v}_{\gamma^*}(z)\}$ où

$$\begin{aligned}
\mathbf{v}_0(z) &= (M(z), 0, \ldots, 0) \\
\mathbf{v}_1(z) &= (\tilde{h}_1(z), 1, \ldots, 0) \\
&\vdots \\
\mathbf{v}_{\gamma^*}(z) &= (\tilde{h}_{\gamma^*}, 0, \ldots, 1).
\end{aligned}$$

Soit

$$\mathbf{w}(z) = r_1(z)\mathbf{v}_1(z) + \cdots + r_{\gamma^*}(z)\mathbf{v}_{\gamma^*}(z).$$

La première coordonnée du vecteur $\mathbf{w}(z)$ est $r_0(z)\tilde{h}_0(z)M(z)t(z)$ pour une valeur de $t(z) \in \mathbb{F}_2[z]$. Soit $\mathbf{w}'(z) = \mathbf{w}(z) - r_0(z)\tilde{h}_0(z)t(z)\mathbf{v}_0(z)$. Évidemment, $\mathbf{w}'(z)$ est dans le dual. Il est facile de voir que la première coordonnée de $\mathbf{w}'(z)$ est zéro et que les polynômes constituant les autres coordonnées ont des degrés qui n'excèdent pas $r = \deg(P(z))$. C'est pourquoi, le premier minimum successif $s_{1,\ell}$ du réseau $\mathcal{L}^*_{\gamma^*+1}(T(\mathbb{F}_2))$ doit être plus petit ou égal à 2^r. Par le théorème 4.2, on a que, pour les générateurs LFSR dans $\mathbb{F}_{2^w}$ ou les GCL polynômiaux dans $\mathbb{F}_{2^w}[z]/P(z)$,

$$t_\ell \leq \log_2 s_{1,\ell} \leq r$$

où $s_{1,\ell}$ est le premier minimum successif de $\mathcal{L}^*_\ell(T(\mathbb{F}_2))$. □

Ce théorème, qui est très général, peut s'appliquer au GCL polynômial et au LFSR. Les deux corollaires suivants démontrent ceci.

Corollaire 5.1. *En supposant une représentation dans la base* $(1, \zeta, \ldots, \zeta^w)$ *des éléments de* $\mathbb{F}_{2^w}$ *où* ζ *est la racine d'un polynôme irréductible* $Q(z) \in \mathbb{F}_2[z]$, *l'équidistribution d'un GCL polynômial dans* $\mathbb{F}_{2^w}[z]/P(z)$, *où* $P(z)$ *est irréductible sur* $\mathbb{F}_{2^w}$, $P(z) = p_0(z) + \zeta p_1(z) + \cdots + \zeta^{\gamma^*} p_{\gamma^*}(z)$ *et* $p_i(z) \in \mathbb{F}_2[z]$ *pour* $i = 0, \ldots, \gamma^*$ *et la sortie est donnée par* $\mathbf{y}_n = \mathbf{q}_{n,r}$, *est telle que*

$$t_{\gamma^*+1} \leq r.$$

Ceci implique que $t_\ell \leq r$ *pour* $\ell > \gamma^*$.

Corollaire 5.2. *En supposant une représentation dans la base* $(1, \zeta, \ldots, \zeta^w)$ *des éléments de* $\mathbb{F}_{2^w}$ *où* ζ *est la racine d'un polynôme irréductible* $Q(z) \in \mathbb{F}_2[z]$, *l'équidistribution d'un LFSR dans* $\mathbb{F}_{2^w}$, *dont le polynôme caractéristique est irréductible sur* $\mathbb{F}_{2^w}$ *et est* $P(z) = p_0(z) + \zeta p_1(z) + \cdots + \zeta^{\gamma^*} p_{\gamma^*}(z)$ *où* $p_i(z) \in \mathbb{F}_2[z]$ *pour* $i = 0, \ldots, \gamma^*$ *et la sortie est donnée par* $\mathbf{y}_n = \mathbf{v}_n$, *est telle que*

$$t_{\gamma^*+1} \leq r.$$

Ceci implique que $t_\ell \leq r$ *pour* $\ell > \gamma^*$.

Démonstration. Ces deux corollaires peuvent se démontrer en observant que pour le LFSR, la matrice T est définie par (5.6) et $\bar{\mathbf{t}}_n^{(0)} = \mathbf{v}_n$ et que pour le GCL polynômial dans $\mathbb{F}_{2^w}[z]/P(z)$, la matrice T est définie par (5.9) et $\bar{\mathbf{t}}_n^{(0)} = \mathbf{q}_{n,r}$ □

Ce théorème démontre la limitation observée sur l'équidistribution des TGFSR sans tempering. Pour le TGFSR, $P(z) = p_0(z) + \zeta p_1(z)$ où $p_0(z) = z^r + z^m$ et $p_1(z) = 1$, ce qui implique que $t_2 \leq r$. Dans [102], on démontre ce fait, mais par une autre technique qui est difficilement adaptable à notre cas.

5.6 Équidistribution pour certaines projections

Dans cette section, on discute des propriétés des LFSR dans $\mathbb{F}_{2^w}$ et des GCL polynomiaux dans $\mathbb{F}_{2^w}[z]/P(z)$ par rapport à l'équidistribution de l'ensemble de points $\Psi_t(\mathbf{X}, H, L, J)$ où $J = \{j_1, j_2, \ldots, j_t\}$ et $j_1 = 0$. La propriété la plus intéressante démontrée dans cette section est que la très grande majorité des projections en deux dimensions, $\Psi_t(\mathbf{X}, H, L, \{j_1, j_2\})$, quand $L = w$, affichent une équidistribution parfaite. On donne aussi des résultats pour les projections de dimension plus grande que deux. Les résultats sont montrés avec la matrice de tempering $H = [R\ 0]$ où R est une matrice $w \times w$ non singulière et H est une matrice $w \times k$. En d'autres mots, la sortie est produite à partir de $\mathbf{y}_n = R\mathbf{v}_n$ ou $\mathbf{y}_n = R\mathbf{q}_{n,0}$ dépendemment du type de générateur.

Pour l'analyse de cette section, on émet l'hypothèse suivante.

Hypothèse 5.1. *Pour tous les résultats de cette section, supposons que $P(z)$ soit irréductible sur $\mathbb{F}_{2^w}$.*

Pour un ensemble d'indices $J = \{0, j_2, \ldots, j_t\}$, par la proposition 4.2, on dit que $\Psi_t(\mathbf{X}, H, L, J)$ est de résolution ℓ si et seulement si l'application linéaire

$$\mathbf{x}_0 \longrightarrow (\mathrm{trunc}_\ell(\mathbf{y}_0), \mathrm{trunc}_\ell(\mathbf{y}_{j_2}) \ldots, \mathrm{trunc}_\ell(\mathbf{y}_{j_t})) \tag{5.21}$$

est de plein rang [33].

Pour les LFSR dans $\mathbb{F}_{2^w}$, avec $\ell = L = w$, $\mathbf{y}_n = R\mathbf{v}_n$, pour $t \leq r$, l'application (5.21) est équivalente à

$$(m_0, m_1, \ldots, m_{r-1}) \longrightarrow (m_0, m_{j_2}, \ldots, m_{j_t})$$

quand R est une matrice $w \times w$ de plein rang. Pour les GCL polynômiaux dans $\mathbb{F}_{2^w}[z]/P(z)$, avec $\ell = L = w$, $\mathbf{y}_n = R\mathbf{q}_{n,r}$, pour $t \leq r$, (5.21) est équivalent à

$$(q_{0,r}, \ldots, q_{0,1}) \longrightarrow (q_{0,r}, q_{j_2,r}, \ldots, q_{j_t,r}). \tag{5.22}$$

Puisque dans les deux cas (LFSR et GCL), $\{R\mathbf{q}_{n,r}\}_{n\geq 0}$ et $\{R\mathbf{v}_n\}_{n\geq 0}$ sont la même séquence, mais avec des points de départ différents, on se concentre sur le GCL polynômial dans $\mathbb{F}_{2^w}[z]/P(z)$ pour analyser l'équidistribution dans les deux cas.

Lemme 5.1. *L'application linéaire*

$$(q_{0,r},\ldots,q_{0,1}) \rightarrow (q_{0,i},\ldots,q_{r-1,i}).$$

est de plein rang pour $i = 1,\ldots,r$.

Démonstration. Nous allons démontrer qu'étant donné un vecteur $(q_{0,r},\ldots,q_{0,1}) \in \mathbb{F}_{2^w}^r$, le vecteur $(q_{0,i},\ldots,q_{r-1,i}) \in \mathbb{F}_{2^w}^r$ correspondant est unique et vice-versa.

Étant donné l'initialisation $\mathbf{t}_0 = (q_{0,r},\ldots,q_{0,1})$, la fonction génératrice $G_i(z) = q_{0,i}z^{-1}+\cdots+q_{r-1,i}z^{-r}+\cdots$ correspondante est unique et est déterminée entièrement par l'initialisation $\mathbf{t}_0$. D'autre part, si on choisit un vecteur $(q_{0,i},\ldots,q_{r-1,i}) \in \mathbb{F}_{2^w}^r$ arbitrairement, alors celui-ci est produit par au moins une initialisation $\mathbf{t}_0$. Supposons qu'il y ait deux initialisations $\mathbf{t}_0$ et $\tilde{\mathbf{t}}_0$ qui génèrent la fonction génératrice $G_i(z) = q_{0,i}z^{-1} + \cdots + q_{r-1,i}z^{-r} + \cdots$. Dans ce cas, la fonction génératrice de l'initialisation $\mathbf{t}_0 + \tilde{\mathbf{t}}_0$ est $G_i(z) = 0$. Ceci implique que, soit la i-ième ligne de la matrice de transition est nulle, soit $\mathbf{t}_0 = \tilde{\mathbf{t}}_0$. On sait que la matrice de transition est de plein rang (car on suppose que $P(z)$ est irréductible), ce qui veut dire qu'il n'y a qu'une seule initialisation qui produit la fonction génératrice $G_i(z)$. □

On définit les matrices W_i, $1 \leq i \leq r$, par celles qui correspondent aux applications linéaires suivantes (pas nécessairement de plein rang)

$$W_i(q_{0,i},\ldots,q_{r-1,i})^{\mathsf{T}} = (q_{0,i}, q_{j_2,i},\ldots,q_{j_t,i})^{\mathsf{T}}. \tag{5.23}$$

Lemme 5.2. *Les matrices* W_i, *pour* $i = 1,\ldots,r$, *sont une seule et même matrice* W.

Démonstration. Soient $G_i(z) = q_{0,i}z^{-1} + \cdots + q_{r-1,i}z^{-r} + \cdots$. On sait que les coefficients de $G_i(z)$ suivent la récurrence linéaire

$$q_{n,i} = \sum_{j=1}^{r} b_j q_{n-j,i}$$

quelle que soit la valeur de i. Pour une valeur de j_h fixée, en manipulant cette dernière équation, on peut obtenir l'équation

$$q_{j_h,i} = w_{h,0}q_{0,i} + \cdots + w_{h,r-1}q_{r-1,i},$$

où $w_{h,l} \in \mathbb{F}_{2^w}$, $0 \leq l < r$, qui relie q_{j_h} à $q_{0,i}, \ldots, q_{r-1,i}$. Remarquons que les coefficients $w_{h,l}$, $0 \leq l < r$, ne dépendent que de j_h et non de i. Notons également que la matrice W_i contient à la première ligne le vecteur $(1, 0, \ldots, 0)$ et à la h-ième ligne, $2 \leq h \leq r$, le vecteur $(w_{h,0}, \ldots, w_{h,r-1})$. Tous ces vecteurs ne dépendent pas de i, donc, peu importe sa valeur, W_i est toujours la même matrice. □

Puisque les matrices W_i sont toutes la même matrice, dans ce qui suit, on note les W_i par une seule et même matrice W. Additionnons les applications linéaires (5.23) de la manière suivante

$$\sum_{i=1}^{r} W(q_{0,i}, \ldots, q_{r-1,i})^{\mathsf{T}} z^{r-i} = \sum_{i=1}^{r} (q_{0,i}, q_{j_2,i}, \ldots, q_{j_t,i})^{\mathsf{T}} z^{r-i}$$

pour obtenir

$$W(q_0(z), \ldots, q_{r-1}(z))^{\mathsf{T}} = (q_0(z), q_{j_2}(z), \ldots, q_{j_t}(z))^{\mathsf{T}}$$

où

$$W = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ w_{2,0} & w_{2,2} & \ldots & w_{2,r-1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{t,0} & w_{t,2} & \ldots & w_{t,r-1} \end{pmatrix} = \begin{pmatrix} \mathbf{w}_{1\cdot} \\ \mathbf{w}_{2\cdot} \\ \vdots \\ \mathbf{w}_{t\cdot} \end{pmatrix}$$

et les $w_{i,j} \in \mathbb{F}_{2^w}$, $\mathbf{w}_{i\cdot} \in \mathbb{F}_{2^w}^r$, $1 \leq i \leq t$, $0 \leq j \leq r-1$.

Théorème 5.4. *L'application (5.22) est de plein rang si et seulement si la matrice W est de plein rang.*

Démonstration. Puisqu'il existe une application de plein rang de $(q_{0,r}, \ldots, q_{0,1})$ vers $(q_{0,r}, \ldots, q_{r-1,r})$, pour démontrer que (5.22) est de plein rang, il suffit que

$$(q_{0,r}, \ldots, q_{r-1,r}) \rightarrow (q_{0,r}, q_{j_2,r}, \ldots, q_{j_t,r})$$

soit de plein rang, donc que W soit de plein rang. De même, si (5.22) est de plein rang, ceci implique directement que W est de plein rang. □

Ainsi, pour déterminer si $\Psi_t(\mathbf{X}, H, L, J)$ est (t, w)-équidistribué, il suffit de construire la matrice W correspondante et de déterminer son rang. Il est (t, w)-équidistribué si W est de plein rang.

Soit $J_{t,j} = \{(j_1, j_2, \ldots, j_t) : 0 = j_1 < j_2 < \ldots < j_t \leq j\}$. On définit par j_t^* la valeur maximale de j pour laquelle l'application (5.22) est de plein rang pour tous les tuplets d'indices contenus dans $J_{t,j}$. Par exemple, en deux dimensions, la valeur de j_2^* est la plus grande valeur de j_2 telle que l'ensemble de points $\Psi_t(\mathbf{X}, H, L, \{0, j_2\})$ est $(2, w)$-équidistribué pour tout $j_2 \leq j_2^*$. Soit le critère $\Delta(S(a), \Lambda) = \max_{J \in S(a)} \Lambda(J)$ où $S(a) = \{\{0, j\} : 1 < j \leq a\}$. Pour ce critère, on a que $\Delta(S(j_2^*), \Lambda) = 0$ et $\Delta(S(j_2^* + 1), \Lambda) > 0$. Si on cherche des générateurs avec le critère $\Delta(S(a), \Lambda)$, il est préférable que la valeur de j_2^* soit la plus grande possible. En général, pour un générateur, on voudrait que j_t^* soit le plus grand possible pour chacune des dimensions t qui nous intéressent. Dans ce qui suit, on donne différents résultats sur la manière dont se comporte j_t^*.

Lemme 5.3. *On a $j_2^* \geq j_3^* \geq \ldots \geq j_r^*$.*

Démonstration. Soit $J = \{0, j_2, \ldots, j_{s-1}, j_s^* + 1\}$, une projection telle que l'ensemble de points $\Psi_s(\mathbf{X}, H, L, J)$ n'est pas (s, w)-équidistribué et $s < r$. Soit W_J, la matrice W

de l'application (5.22) pour cette projection. On sait que W_J n'est pas de plein rang. Soient $J' = \{0, j_2, \ldots, j_{s-1}, j_s^* + 1, j'\}$ où $j' > j_s^* + 1$ et $J'' = \{0, j_2, \ldots, j'', \ldots, j_s^* + 1\}$ où $0 < j'' < j_s^*$ et $j'' \neq j_i$ pour $i = 2, \ldots, s-1$, deux projections en dimension $s+1$. Montrons que $\Psi_{s+1}(\mathbf{X}, H, L, J')$ et $\Psi_{s+1}(\mathbf{X}, H, L, J'')$ ne sont pas $(s+1, w)$-équidistribués. Soient $W_{J'}$ et $W_{J''}$, les matrices W de l'application (5.22) pour les projections J' et J'', respectivement. Les matrices W_J et $W_{J'}$ ont les mêmes lignes, sauf que $W_{J'}$ a une ligne de plus (la dernière de $W_{J'}$). De plus, les matrice W_J et $W_{J''}$ ont aussi les mêmes lignes, excepté que $W_{J''}$ a une ligne de plus. Puisque W_J n'est pas de plein rang et $s < r$, ceci implique que $W_{J'}$ et $W_{J''}$ ne peuvent pas non plus être de plein rang. On en conclut que $j_{s+1}^* \leq j_s^*$ □

Lemme 5.4. *$j_t^* \geq r - 1$ pour $t \leq r$.*

Démonstration. Soit $t \leq r$, un entier. Pour $J = \{0, j_2, \ldots, j_t\}$ où $j_t < r$, la matrice W est la matrice qui contient à la h-ième ligne le vecteur composé de zéros sauf un « 1 » à la j_h-ième coordonnée. Évidemment, cette matrice est de plein rang. Ce qui montre que $j_t^* \geq r - 1$ pour $t \leq r$. □

Corollaire 5.3. *Pour une valeur de s inférieure à r, si $j_s^* = r - 1$, alors $j_t^* = r - 1$ pour $s < t \leq r$.*

Démonstration. Le résultat est obtenu en combinant les lemmes 5.4 et 5.3. □

Proposition 5.1. *Soit p, le nombre de coefficients non nuls de $P(z)$. Si $p \leq r$, alors $j_p^* = r - 1$.*

Démonstration. Supposons que $p \leq r$. Soit

$$P(z) = z^r - \sum_{j=1}^{t-1} b_{r-n_j} z^{n_j}$$

où $0 = n_1 < n_2 < \ldots < n_{t-1} < r$. Soient $t = p$ et $J = \{0, j_1, \ldots, j_t\}$ où $j_t = r$. Pour cette projection, la matrice W est telle que

$$w_{v,k} = \begin{cases} 1 & \text{si } k = j_v \\ 0 & \text{sinon} \end{cases}$$

pour $v = 2, \ldots, t-1$, $k = 0, \ldots, r-1$, et $w_{t,k} = b_{r-k}$ pour $k = 0, \ldots, r-1$, puisque $z^r \equiv \sum_{j=1}^{t-1} b_{r-n_j} z^{n_j} \bmod P(z)$. Pour la projection $J = \{0, n_2, \ldots, n_{t-1}, r\}$, on construit la matrice W de la manière prescrite ci-haut. On observe que

$$\mathbf{w}_{t\cdot} = b_{n_{r-n_1}} \mathbf{w}_{1\cdot} + \cdots + b_{r-n_{t-1}} \mathbf{w}_{t-1\cdot},$$

puisque $q_r(z) = b_{n_{r-n_1}} q_{n_1}(z) + \cdots + b_{r-n_{t-1}} q_{n_{t-1}}(z)$. Pour cette projection, la relation (5.22) n'est pas de plein rang par le théorème 5.4. Pour les valeurs de $j_t \leq r-1$, il est clair que la relation (5.22) est de plein rang, ce qui implique que $j_t^* = j_p^* \leq r-1$. Mais puisque $p \leq r$, par le lemme 5.4, on obtient $j_p^* = r-1$. □

Corollaire 5.4. *La valeur de j_r^* est $r-1$.*

Démonstration. Ce corollaire est une conséquence de la proposition 5.1 et du corollaire 5.3. □

Le prochain théorème est probablement celui qui donne le plus de valeur aux LFSR dans $\mathbb{F}_{2^w}$ et aux GCL polynômiaux dans $\mathbb{F}_{2^w}[z]/P(z)$. En effet, celui-ci garantit que l'équidistribution sera parfaite, jusqu'en résolution w, pour la très grande majorité des projections en deux dimensions que l'on pourrait considérer. Le théorème est important car cette propriété réduit de façon considérable le risque d'obtenir des dépendances statistiques entre les paires de nombres (u_n, u_{n+c}) pour n'importe quelle valeur de c. Le théorème est valide pour n'importe quel choix de polynôme caractéristique $P(z)$ irréductible sur $\mathbb{F}_{2^w}$.

Théorème 5.5. *Supposons que $P(z) \in \mathbb{F}_{2^w}[z]$, un polynôme irréductible sur $\mathbb{F}_{2^w}$, soit le polynôme caractéristique sur $\mathbb{F}_{2^w}$ d'un GCL polynômial et qu'il soit tel que son*

nombre de coefficients non nuls, p, soit supérieur à 2. Soit $g(z)$, un élément primitif de $\mathbb{F}_{2^w}[z]/P(z)$ et q, un entier positif tel que $g(z)^q \equiv z \bmod P(z)$. Pour ce GCL polynômial, on a que

$$j_2^* = (2^{wr}-1)/(\text{pgcd}(2^{wr}-1,q)(2^w-1)) - 1.$$

Démonstration. Soit m, l'ordre de z dans $\mathbb{F}_{2^w}[z]/P(z)$. Par le théorème 1.15 (ii) de [56], on a $m = (2^{wr}-1)/\text{pgcd}(2^{wr}-1,q)$. La deuxième ligne de W n'est jamais nulle et, quand $t = 2$, la seule manière que la matrice W ne soit pas de plein rang est que $w_{2,0} \neq 0$ et $w_{2,j} = 0$ pour tout $j = 1,\ldots,r-1$. Dans ce cas,

$$\begin{aligned}
q_{j_2}(z) &= w_{2,0}q_0(z) \\
z^{j_2}q_0(z) &\equiv w_{2,0}q_0(z) \bmod P(z) \\
z^{j_2} &\equiv w_{2,0} \bmod P(z) \\
(z^{j_2})^{2^w-1} &\equiv w_{2,0}^{2^w-1} \bmod P(z) \\
z^{(j_2)(2^w-1)} &\equiv 1 \bmod P(z) \\
j_2(2^w-1) &\equiv 0 \bmod m \\
j_2(2^w-1) &= vm \\
j_2 &= vm/(2^w-1),
\end{aligned}$$

où v est un entier positif. La cinquième égalité est valide puisque $w_{2,0}^{2^w-1} = 1$ par le théorème 1.15 (ii) de [56]. La sixième égalité découle du fait que z est d'ordre m dans $\mathbb{F}_{2^w}[z]/P(z)$. On veut la plus petite valeur non nulle de j_2 possible, qui est $m/(2^w-1)$ (quand $v = 1$). On peut conclure que $j_2^* = m/(2^w-1) - 1$. Par contre, si $p \leq 2$, alors par la proposition 5.1, on obtient $j_2^* = r - 1$. □

Corollaire 5.5. *Supposons que $P(z) \in \mathbb{F}_{2^w}[z]$, un polynôme primitif sur $\mathbb{F}_{2^w}$ avec p coefficients non nuls, soit le polynôme caractéristique d'un GCL polynômial. Pour ce GCL polynômial, si $p > 2$, alors $j_2^* = (2^{wr}-1)/(2^w-1) - 1$.*

Démonstration. En faisant référence au théorème 5.5, si $P(z)$ est primitif, alors $g(z) = z$. Dans ce cas, $q = 1$ et $\text{pgcd}(2^{wr} - 1, q) = 1$. □

Pour les deux prochains corollaires, on suppose que $P(z)$ est primitif.

Corollaire 5.6. *Soit $J = \{0, j\}$, $j > 0$. L'ensemble de points $\Psi_2(\mathbf{X}, H, L, J)$ est $(2, w)$-équidistribué si et seulement si j n'est pas un multiple de $(2^{wr} - 1)/(2^w - 1)$.*

Corollaire 5.7. *Soit $J = \{i, i + j\}$, $i \geq 0$, $j > 0$. La proportion des ensembles de points $\Psi_2(\mathbf{X}, H, L, J)$ qui ne sont pas $(2, w)$-équidistribués est $(2^w - 1)/(2^{wr} - 1)$.*

Démonstration. Soit $J = \{0, j\}$. Par le corollaire 5.6, on doit compter le nombre de valeurs de j, où $0 \leq j \leq 2^{wr} - 1$, telles que $j = v(2^{wr} - 1)/(2^w - 1)$ où v est un entier. On sait que $j \leq 2^{wr} - 1$, ce qui implique $v \leq 2^w - 1$. Le nombre de valeurs de j telles que $j = v(2^{wr} - 1)/(2^w - 1)$ est donc $2^w - 1$. Puisque le nombre de valeurs de j possibles est $2^{wr} - 1$, la proportion des projections J telles que $\Psi_2(\mathbf{X}, H, L, J)$ est $(2, w)$-équidistribué est $(2^w - 1)/(2^{wr} - 1)$. Puisque que $\Psi_t(\mathbf{X}, H, L)$ est stationnaire dans la dimension, on obtient le résultat du corollaire. □

Ce dernier corollaire nous dit que presque toutes les projections en 2 dimensions sont parfaites du points de vue de l'équidistribution quand $L = w$ et $P(z)$ est primitif sur $\mathbb{F}_{2^w}$. Ceci réduit de façon considérable le risque de corrélation entre des paires de valeurs produites par le générateur.

Exemple 5.5. Quand $L = w = 32$ et $r = 25$, on a que seulement 1 projection sur 2^{768} n'est pas parfaite ! Même quand $r = 2$, on obtient un ratio de 1 projection sur 2^{32} qui n'est pas parfaite. Ceci s'avère être une qualité importante de ce type de générateur.

Cette propriété peut être intéressante pour l'utilisation de ce type de générateurs pour construire des générateurs de nombres aléatoires parallèles. Ces générateurs sont utilisés pour effectuer des simulations en parallèle sur plusieurs processeurs. Pour ce

type d'applications, on utilise un générateur différent sur chacun des processeurs. Souvent, on émet l'hypothèse que chacun des générateurs de nombres aléatoires produit une séquence de nombres aléatoires qui est indépendante des autres séquences produites par les autres générateurs. Une technique courante pour implanter plusieurs générateurs est d'affecter un segment d'une même séquence aléatoire à chacun des K processeurs. Soit $\{u_n\}_{n\geq 0}$, la séquence de base. Les processeurs sont numérotés de 1 à K. Le processeur k utilise la séquence $\{u_{n+(k-1)Z}\}_{n\geq 0}$ où Z est grand. Puisque Z est grand, on suppose que les séquences ne se chevauchent pas et qu'elles sont indépendantes les unes des autres.

Pour certains types de générateurs, tels les GCL ordinaires, il faut être prudent quand on choisit la valeur de Z. Dans [14, 15, 16, 13, 17, 18], les auteurs étudient les corrélations de long délai, c'est-à-dire les corrélations des ensembles de points formés par les par (u_n, u_{n+Z}) pour les GCL ordinaires. Ils affirment que les corrélations sont importantes quand les résultats au test spectral de l'ensemble de points sont insatisfaisants. Le test spectral est un test théorique qui mesure l'uniformité de générateurs basés sur une récurrence linéaire dans $\mathbb{Z}_p$ ou $p > 2$ est un nombre premier. L'équidistribution et le test spectral sont semblables à plusieurs égards [53]. Similairement, puisque l'équidistribution des ensemble de points $\{(u_n, u_{n+c}) : n \geq 0\}$ est parfaite pour presque toutes les valeurs de c, on pourrait conclure que les corrélations de long délai d'ordre 2 sont pratiquement inexistantes pour les GCL polynômiaux dans $\mathbb{F}_{2^w}/P(z)$ et pour les LFSR dans $\mathbb{F}_{2^w}$. De plus, on peut facilement éviter les mauvaises valeurs de c, puisqu'on les connaît.

Dans [20], on analyse l'ensemble de points $P_{t,Z} = \{(u_n, u_{n+Z}, \ldots, u_{n+(t-1)Z}) : n \geq 1\}$, produit par un GCL ordinaire, avec le test spectral. On utilise l'ensemble de points pour intégrer des fonctions tests. L'auteur démontre de manière empirique, qu'un biais important est induit à l'estimateur Monte Carlo quand $P_{t,Z}$ n'obtient pas de bons résultats au test spectral.

Ce résultat de [20], nous incite à croire que la même chose pourrait se produire si l'ensemble de points $P_{t,Z} = \{(u_n, u_{n+Z}, \ldots, u_{n+(t-1)Z}) : n \geq 1\}$ produit par un GCL polynômial ne performe pas bien par rapport à l'équidistribution. Pour utiliser ce type de générateurs dans des applications de simulation parallèle, il faudrait analyser l'équidistribution de cet ensemble de points. Si l'équidistribution de ceux-ci est satisfaisante pour $t = 1, \ldots, wr$, alors le générateur est un bon candidat pour être utilisé comme générateur parallèle. Trouver les paires $(P(z), Z)$ tels que $P_{t,Z}$ est bien équidistribué pour $t = 1, \ldots, rw$ pourrait être une avenue de recherche intéressante pour la construction de générateurs parallèles.

5.7 Techniques d'implantation

Dans cette section, on discute de deux techniques pour implanter les générateurs qui utilisent le corps fini $\mathbb{F}_{2^w}$. La difficulté dans l'implantation de ces générateurs est d'effectuer les multiplications dans le corps fini $\mathbb{F}_{2^w}$ rapidement et efficacement du point de vue de la mémoire. Afin de surmonter cette difficulté, on utilise des tableaux de valeurs pré-calculées. Les prochaines sous-sections proposent deux techniques.

5.7.1 Récurrences avec coefficients b_i quelconques

La première technique considère le cas où les coefficients du polynôme $P(z)$ sont quelconques, c'est à dire qu'ils peuvent prendre n'importe quelle valeur de $\mathbb{F}_{2^w}$. Soit b, un de ces coefficients. Pour multiplier un élément $v \in \mathbb{F}_{2^w}$ par b, il faut multiplier sa représentation $\mathbf{v}$ dans la base choisie par la matrice A_b. Cette multiplication se fait en $O(w^2)$ opérations, ce qui n'est pas très efficace.

Pour une plus grande efficacité, en termes de rapidité, on pourrait construire une table de 2^w valeurs pré-calculées qui contiendrait toutes les multiplications possibles.

Ceci devient vite impraticable, sur le plan de la mémoire, quand w est moindrement grand. Par exemple, si $w = 32$, alors cette table aurait 2^{32} entrées et occuperait 16 Go de mémoire.

Une façon de pallier au problème causé par la grande quantité de mémoire nécessaire est d'effectuer la multiplication en un petit nombre c d'accès à la mémoire. Pour ce faire, on divise la matrice A_b en sous-matrices $A_b^{(1)}, A_b^{(2)}, \ldots, A_b^{(c)}$, chaque matrice $A_b^{(i)}$ étant de dimension $h \times w_i$ de telle manière que

$$A_b = [A_b^{(1)} : \ldots, A_b^{(c)}] \tag{5.24}$$

et

$$\sum_{i=1}^{c} w_i = w. \tag{5.25}$$

Ainsi, la multiplication $A_b\mathbf{v}$ peut être effectuée par $A_b^{(1)}\mathbf{v}^{(1)} + \cdots + A_b^{(c)}\mathbf{v}^{(c)}$ où $\mathbf{v} = ({\mathbf{v}^{(1)}}^\mathsf{T}, \ldots, {\mathbf{v}^{(c)}}^\mathsf{T})^\mathsf{T}$ et $\mathbf{v}^{(i)}$ est un vecteur de w_i bits. Les multiplications par les matrices $A_b^{(i)}$ sont pré-calculées. Dans ce cas, la mémoire nécessaire pour emmagasiner les multiplications par les matrices $A_b^{(i)}$ est

$$w \sum_{i=1}^{c} 2^{w_i}$$

bits.

Exemple 5.6. Prenons le cas où $w = 32$, $c = 2$ et $w_1 = w_2 = 16$. La mémoire nécessaire est de 512 Ko et le nombre d'accès à la mémoire pour effectuer la multiplication est 2. On peut voir une nette amélioration de l'utilisation de la mémoire par rapport au cas où l'on ne fait qu'un seul accès à la mémoire. Dans le cas où $w = 32, c = 3, w_1 = w_2 = 11$ et $w_3 = 10$, la mémoire nécessaire n'est que 20 Ko. Ces exemples nous montrent qu'il est possible d'utiliser des tables pré-calculées qui utilisent peu de mémoire et peu d'accès à la mémoire et ce, pour n'importe quelle matrice A_b.

Un autre problème des tables pré-calculées est le temps nécessaire pour les calculer. Une estimation optimiste voudrait que l'on puisse effectuer 2×10^7 multiplications en 1 seconde sur un ordinateur de table courant. Dans le cas où $w = 32$ et $c = 1$, les 2^{32} multiplications se font en 215 secondes. Mais dans le cas où $w = 32, c = 3, w_1 = w_2 = 11$ et $w_3 = 10$, une évaluation pessimiste (où l'on ne fait que 10^5 multiplications par seconde) donne que les pré-calculs se font en 5×10^{-2} secondes. Ce temps est acceptable puisque cette opération n'est effectuée qu'une seule fois, soit à l'initialisation du générateur.

Grâce à ce type d'analyse, on peut déterminer s'il est mieux de pré-calculer à l'exécution ou d'effectuer les pré-calculs une seule fois, d'emmagasiner les résultats dans un fichier et de les charger à l'exécution.

Un avantage de ce type d'implantation est d'avoir une très grande flexibilité dans le choix de la matrice de tempering. En effet, si le tempering n'est effectué que sur les w premiers bits du vecteur d'état, alors il est possible d'intégrer ce tempering dans les tables de calcul. Voici comment.

Soit

$$B = \begin{pmatrix} R & 0 & \cdots & 0 \\ 0 & R & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R \end{pmatrix} \tag{5.26}$$

la matrice de tempering où R est une matrice $w \times w$ non singulière. Pour un LFSR dans $\mathbb{F}_{2^w}$, on produit la sortie avec $\mathbf{y}_n = B\mathbf{x}_n = ((R\mathbf{v}_n)^\mathsf{T}, \ldots, (R\mathbf{v}_{n-r+1})^\mathsf{T})^\mathsf{T}$. Soit

$\mathbf{w}_n = R\mathbf{v}_n$. De la récurrence (5.5), on obtient

$$\begin{aligned} R\mathbf{v}_n &= \sum_{i=0}^{r} RA_{b_i}\mathbf{v}_{n-i} \\ \mathbf{w}_n &= \sum_{i=0}^{r} RA_{b_i}R^{-1}R\mathbf{v}_{n-i} \\ \mathbf{w}_n &= \sum_{i=0}^{r} RA_{b_i}R^{-1}\mathbf{w}_{n-i}. \end{aligned}$$

Ainsi, au lieu de pré-calculer les multiplications par les matrices A_{b_i}, on pré-calcule les multiplications par les matrices $RA_{b_i}R^{-1}$. La sortie devient

$$\mathbf{y}_n = (\mathbf{w}_n^{\mathsf{T}}, \ldots, \mathbf{w}_{m-r+1}^{\mathsf{T}})^{\mathsf{T}}.$$

Ceci permet de choisir n'importe quel tempering B de la forme (5.26), en autant que R soit de rang w, et de l'implanter de manière efficace, sans grand coût supplémentaire.

Un développement semblable peut être fait pour le GCL polynômial. Le type de tempering défini par (5.26) correspond à transformer la base de $\mathbb{F}_{2^w}$. On pourrait implanter ce générateur en remplaçant A_{b_i} par $RA_{b_i}R^{-1}$ dans la matrice X donnée par (5.6) et utiliser $B = I$, puisque le tempering est incorporé dans la récurrence.

Le problème avec une matrice de tempering R compliquée est le temps nécessaire qui est souvent long pour effectuer la multiplication. Puisque le tempering est inclus dans les tables de multiplications, alors peu importe la complexité de la matrice R, le temps d'exécution du générateur ne s'en trouve nullement affecté.

5.7.2 Récurrences avec coefficients b_i particuliers

Malgré le fait que l'on peut effectuer la multiplication par A_{b_i} de façon relativement efficace sur le plan de la mémoire, la technique décrite dans la section précédente peut

se révéler peu efficace lorsque les générateurs sont utilisés dans des applications qui utilisent intensivement la cache de l'ordinateur. Pour pallier à ce problème, on utilise un polynôme $P(z)$ qui a des coefficients b_i d'une forme particulière. Les coefficients considérés sont de la forme

$$b = \sum_{i=1}^{s} c_i \zeta^{i-1}. \tag{5.27}$$

Expliquons comment cette forme permet d'effectuer la multiplication plus efficacement. Soient $Q(z) = z^w + \sum_{i=1}^{r} a_i z^{r-i}$, le polynôme minimal de ζ sur $\mathbb{F}_2$ et

$$A_\zeta = \begin{pmatrix} & & & a_r \\ 1 & & & a_{r-1} \\ & 1 & & \vdots \\ & & \ddots & a_2 \\ & & 1 & a_1 \end{pmatrix}$$

(la matrice (B.1) de l'annexe B), qui permet de multiplier par ζ dans la base polynômiale. Il est facile de voir que, si $q < w$, alors on a

$$A_\zeta^q = \begin{pmatrix} 0 & 0 & \dots & 0 & p_{11} & \dots & p_{1q} \\ 0 & 0 & \dots & 0 & p_{21} & \dots & p_{2q} \\ \vdots & & & \vdots & \vdots & & \vdots \\ 1 & 0 & \dots & 0 & p_{q1} & \dots & p_{qq} \\ 0 & 1 & & 0 & p_{q+1,1} & \dots & p_{q+1,q} \\ & & \ddots & & & & \\ 0 & 0 & \dots & 1 & p_{w1} & \dots & p_{wq} \end{pmatrix}.$$

On peut exprimer A_ζ^q par $R_q + T_q$ où

$$R_q = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & & \vdots & \vdots & & \vdots \\ 1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 1 & & 0 & 0 & \dots & 0 \\ & & \ddots & & & & \\ 0 & 0 & \dots & 1 & 0 & \dots & 0 \end{pmatrix}$$

et

$$T_q = \begin{pmatrix} 0 & 0 & \dots & 0 & p_{11} & \dots & p_{1q} \\ 0 & 0 & \dots & 0 & p_{21} & \dots & p_{2q} \\ \vdots & & & \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 & p_{q1} & \dots & p_{qq} \\ 0 & 0 & & 0 & p_{q+1,1} & \dots & p_{q+1,q} \\ & & \ddots & & & & \\ 0 & 0 & \dots & 0 & p_{w1} & \dots & p_{wq} \end{pmatrix}.$$

La multiplication $A^q\mathbf{v} = (R_q + T_q)\mathbf{v}$ se fait en deux étapes. La première consiste à décaler vers la droite de q positions le vecteur $\mathbf{v}$ et la deuxième consiste à regarder dans une table le résultat de la multiplication $T_q\mathbf{v}$. Le résultat de cette multiplication est uniquement déterminé par les q bits les moins significatifs de $\mathbf{v}$. Ceci veut dire que la table ne contient que 2^q valeurs. Dans le cas où $q = 8$ et $w = 32$, alors la table n'occupe que 1 Ko.

Maintenant, supposons que l'on veuille multiplier par $b = \sum_{i=0}^{s} c_i\zeta^{i-1}$. Dans ce

cas, on a

$$\begin{aligned} A_b = \sum_{i=0}^{s} c_i A^{i-1} &= \sum_{i=0}^{s} c_i (R_{i-1} + T_{i-1}) \\ &= \sum_{i=0}^{s} c_i R_{i-1} + \sum_{i=0}^{s} c_i T_{i-1} \\ &= \sum_{i=0}^{s} c_i R_{i-1} + T. \end{aligned}$$

L'implantation de la multiplication $A_b\mathbf{v}$ se fait aussi en deux étapes. La première étape, la multiplication $(\sum_{i=0}^{s} c_i R_{i-1})\mathbf{v}$, consiste à faire autant de décalages vers la droite qu'il y a de coefficients c_i non nuls, $0 \leq i \leq s$. La multiplication $(\sum_{i=0}^{s} c_i T_{i-1})\mathbf{v}$ se fait en ne regardant que dans une seule table de valeurs pré-calculées.

Cette technique a l'avantage de permettre l'implantation de générateurs utilisant le corps fini $\mathbb{F}_{2^w}$ avec de petites tables de valeurs pré-calculées. Par contre, comme on a pu le constater à la section 5.5, il y a une limitation sur l'équidistribution qui peut être obtenue avec un LFSR ou un GCL polynômial dont le polynôme caractéristique $P(z)$ a des coefficients de la forme (5.27). Pour pallier à cette restriction sur l'équidistribution, nous utilisons un tempering pour obtenir de bons générateurs à la section 5.8. Également, dans la première étape de l'implantation, le nombre de c_i différents de zéros influence la rapidité de la multiplication. Ceci suggère qu'il faudrait choisir des coefficients qui ont peu de valeurs de c_i non nulles, ce qui restreint davantage la sélection des b_i.

5.8 Recherche de bons générateurs

Dans cette section, on expose les résultats de recherches de bons générateurs. Dans un premier temps, on donne les paramètres de générateurs qui utilisent la technique d'implantation exposée dans la section 5.7.1 et, dans un deuxième temps, on donne les paramètres de générateurs qui peuvent être implantés par les deux techniques

expliquées aux sections 5.7.1 et 5.7.2. Les générateurs trouvés ont des périodes de $2^{96}-1$, $2^{256}-1$, $2^{416}-1$ et $2^{800}-1$.

5.8.1 Paramètres de générateurs avec coefficients quelconques

Dans les Tableaux 5.1, 5.2, 5.3 et 5.4, on donne les paramètres de générateurs qui peuvent être implantés par la technique décrite à la section 5.7.1. Le polynôme caractéristique est de la forme

$$P(z) = z^r + b_{r-t}z^t + b_{r-q}z^q + b_r \in \mathbb{F}_{2^w}[z]. \tag{5.28}$$

Le coefficient b_{r-q} est quelquefois nul. La recherche de paramètres s'est fait en cherchant les générateurs qui minimisent le critère (4.3) qui fait la sommation des valeurs de Δ_ℓ pour $\ell = 1, \ldots, 32$, soit

$$V = \sum_{\ell=1}^{32} \Delta_\ell$$

où Δ_ℓ est l'écart en dimension pour la résolution ℓ. La recherche a été effectuée à l'aide de la bibliothèque REGPOLY et s'est restreint à des générateurs de pleine période de longueur $2^{96}-1$, $2^{256}-1$, $2^{416}-1$ et $2^{800}-1$.

Dans les tableaux présentés dans cette section et la prochaine, les coefficients du polynôme $P(z)$ sont donnés selon la base $(1, \zeta, \ldots, \zeta^{31})$, sous une forme hexadécimale. Le polynôme $Q(z) = z^{32} - \sum_{i=1}^{32} a_i z^{32-i}$ est le polynôme minimal de ζ. On utilise la notation hexadécimale pour représenter le vecteur $(a_{32}, \ldots, a_1)$. Les vecteurs de 32 bits $\mathbf{m}_1$ et $\mathbf{m}_2$ sont ceux utilisés pour le tempering de Matsumoto-Kurita. Dans cette section, les générateurs présentés ne comportent pas de tempering de Matsumoto-Kurita, au contraire de ceux présentés à la prochaine section. Le tempering utilisé est

celui défini par les opérations suivantes :

$$\begin{aligned}
\mathbf{z}_n &= \begin{cases} \text{trunc}_w(\mathbf{v}_n) & \text{si LFSR} \\ \text{trunc}_w(\mathbf{q}_{n,r}) & \text{si GCL polynômial} \end{cases} \\
\mathbf{z}_n &= \mathbf{z}_n \oplus ((\mathbf{z}_n \ll 7) \,\&\, \mathbf{m}_1) \\
\mathbf{y}_n &= \mathbf{z}_n \oplus ((\mathbf{z}_n \ll 15) \,\&\, \mathbf{m}_2).
\end{aligned}$$

Pour les générateurs dont $P(z)$ peut avoir des coefficients quelconques, on réussit à trouver de très bons générateurs. Par exemple, on a trouvé plusieurs générateurs de période $2^{96} - 1$ qui sont ME. Également, les générateurs de période $2^{800} - 1$ du tableau 5.4 sont tous tels que $\Delta_\ell = 0$ pour $\ell = 1, \ldots, 15$. Au tableau 5.5, nous donnons les valeurs de Δ_ℓ, pour $\ell = 16, \ldots, 32$ pour tous les générateurs du tableau 5.4. Un fait intéressant à remarquer est que pour ces générateurs, même si nous avions appliqué du tempering, les gains n'auraient pas été énormes. Par exemple, pour le premier générateur du tableau 5.4, si on transforme sa sortie avec un tempering de Matsumoto-Kurita, alors la meilleure valeur de V qu'on a pu obtenir est 69, comparativement à 74. Étant donné que les bits les plus significatifs se comportent déjà très bien, on juge non nécessaire d'appliquer un tempering sur ces générateurs.

Pour les besoins de la comparaison, notons que le TT800 [67], qui est un excellent générateur de période $2^{800} - 1$, est tel que $V = 261$ et $\Delta_\ell = 0$ pour seulement $\ell = 1, 2, 4, 8, 16, 32$. Son équidistribution est donnée au tableau 5.6.

5.8.2 Paramètres de générateurs avec coefficients b_i particuliers

Dans cette section, on présente des générateurs qui ont un polynôme caractéristique de la forme (5.28) où les coefficients b_i, $1 \leq i \leq r$, sont de la forme $\sum_{i=0}^{s} a_i \zeta^i$ où s est petit. Grâce à la technique décrite à la section 5.7.2, il est possible d'implanter un LFSR ou un GCL polynômial efficacement sous ces contraintes. On pourrait

Tableau 5.1 – Générateurs de période $2^{96} - 1$, $r = 3$, $w = 32$.

r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$Q(z)$	$\mathbf{m}_1$	$\mathbf{m}_2$	V
3	1	–	30a72fa7	–	537a531f	ccb06f34	–	–	1
3	1	–	04a87b98	–	4dd5e06e	ccb06f34	–	–	1
3	1	–	25bd9c69	–	5fa987e6	ccb06f34	–	–	1
3	1	–	45d1259a	–	0907d8a2	ccb06f34	–	–	1
3	2	1	bbf58bb6	bd0c7735	b7c5019c	d53c36b9	–	–	0
3	2	1	db3bd1c3	ffbaad94	2f55958b	d53c36b9	–	–	0
3	2	1	556191da	8c4c7161	8f4aaa6f	d53c36b9	–	–	0
3	2	1	04c10d1c	6766b08e	1ab21bb1	d53c36b9	–	–	0
3	2	1	7e518ac9	4580b6c3	fc64ab9b	d53c36b9	–	–	0

Tableau 5.2 – Générateurs de période $2^{256} - 1$, $r = 8$, $w = 32$.

r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$Q(z)$	$\mathbf{m}_1$	$\mathbf{m}_2$	V
8	6	3	fba454a9	045861d5	c5fb7653	ce023b3b	–	–	4
8	5	2	623a6e23	de6f829f	17600ef0	ce023b3b	–	–	4
8	5	2	ff109816	046abcb0	1a31ec46	ce023b3b	–	–	4
8	6	3	52d7df7d	da3d9833	505c3b55	ce023b3b	–	–	4
8	5	2	7502f382	b97e4366	1a171f8c	ce023b3b	–	–	4

également utiliser la technique décrite à la section 5.7.1. Avec la technique de la section 5.7.2, la taille des tables de pré-calculs pour chacun des coefficients est de 2^s entrées de 32 bits. Dans les tableaux 5.7, 5.8, 5.9 et 5.10, la valeur de s est indiquée par la mention « taille des tables : 2^s ». Le format des tableaux est le même que ceux de la section précédente.

On recherche des générateurs qui minimisent le critère V (voir (4.3)) et dont le polynôme caractéristique a des coefficients $b_1, \ldots, b_r$ de la forme $\sum_{i=0}^{s} c_i \zeta^i$ avec $s = 3$ et $s = 7$. Pour les générateurs avec $s = 3$, la taille des tables est 2^3 entrées de 32 bits et pour ceux avec $s = 7$, la taille des tables est 2^7 entrées de 32 bits. Pour ces générateurs, il existe une borne sur la valeur de t_ℓ par le théorème 5.3. Pour contrer cette borne, on utilise le tempering de Matsumoto-Kurita décrit à la section précédente. On se

Tableau 5.3 – Générateurs de période $2^{416} - 1$, $r = 13$, $w = 32$.

r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$Q(z)$	$\mathbf{m}_1$	$\mathbf{m}_2$	V
13	8	–	2be45a08	–	b4816b12	f9820db6	–	–	20
13	5	–	7a64a92e	–	c0643058	f9820db6	–	–	20
13	5	–	239e0deb	–	73c92f6c	f9820db6	–	–	21
13	8	–	bda6212c	–	22684d7d	f9820db6	–	–	21
13	5	–	a2f56680	–	5f1bd0a9	f9820db6	–	–	21
13	8	–	c1d47af0	–	cfa4a4e8	f9820db6	–	–	21
13	10	5	99e34535	f09bf592	9803caf7	9f26eaa3	–	–	9
13	10	5	62a42238	e765704a	2f95dc0e	9f26eaa3	–	–	10
13	9	2	723e9f93	cdb0bf04	49c6606d	9f26eaa3	–	–	11
13	9	6	329e2294	d423ce0e	98e4d109	9f26eaa3	–	–	12
13	11	7	8bed0cd7	7cc178d0	66a3e242	9f26eaa3	–	–	13
13	11	7	516c29cb	836b3c64	bcba11d5	9f26eaa3	–	–	13
13	12	7	7315e8c0	ccff0488	cf4d0331	9f26eaa3	–	–	13
13	12	5	cd7ba19e	bd10df04	0bd46a31	9f26eaa3	–	–	13

Tableau 5.4 – Générateurs de période $2^{800} - 1$, $r = 25$, $w = 32$.

r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$Q(z)$	$\mathbf{m}_1$	$\mathbf{m}_2$	V
25	7	–	e6a68d20	–	287ab842	fa4f9b3f	–	–	74
25	18	–	26dc0579	–	88fc8c8a	fa4f9b3f	–	–	77
25	20	14	0001e6f1	1d5e07e3	3e433359	f70211b8	–	–	42
25	24	16	be1ed999	e21e9910	e09361e8	f70211b8	–	–	54

Tableau 5.5 – Valeurs de Δ_ℓ pour les générateurs du tableau 5.4 pour $\ell = 16, \ldots, 32$.

ℓ	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
$V = 74$	1	1	5	3	8	6	11	9	8	7	5	4	3	2	1		
$V = 77$	1	1	1	3	8	13	11	9	8	7	5	4	3	2	1		
$V = 42$		1			1	2	4	4	8	7	5	4	3	2	1		
$V = 54$					3	4	9	8	8	7	5	4	3	2	1		

restreint à des générateurs de pleine période de longueurs $2^{96} - 1$, $2^{256} - 1$, $2^{416} - 1$ et $2^{800} - 1$. Les résultats de la recherche sont exposés dans les tableaux 5.7, 5.8, 5.9 et 5.10. Pour certaines valeurs de s, aucun polynôme primitif n'a été trouvé. C'est pour cette raison que, pour certaines combinaisons de paramètres, on ne donne aucun

Tableau 5.6 – Valeurs de Δ_ℓ pour le TT800 pour $\ell = 1, \ldots, 32$.

ℓ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ℓ	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
			16		10	8	14		13	5	22	16	11	7	3	
	22	19	17	15	13	11	9	8	7	5	4	3	2	1		

Tableau 5.7 – Générateurs de période $2^{96} - 1$, $r = 3$, $w = 32$.

r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$Q(z)$	$\mathbf{m}_1$	$\mathbf{m}_2$	V
taille des tables : 2^3									
3	2	–	30000000	–	a0000000	f6b5876b	5ccce080	71d7800c	3
3	2	–	30000000	–	a0000000	f6b5876b	792b3701	9fe700b6	3
taille des tables : 2^7									
3	2	–	0c000000	–	41000000	958357a6	8c5f6000	f00e8066	3
3	2	–	a0000000	–	12000000	958357a6	1d768200	d1e701c2	3
3	1	–	05000000	–	41000000	958357a6	5f226e01	f75181e6	3
taille des tables : 2^3									
3	2	1	90000000	a0000000	50000000	8a81f5f4	24b97381	f9d98000	0
3	2	1	90000000	30000000	50000000	8a81f5f4	b9b76401	b24b0001	0
3	2	1	60000000	a0000000	50000000	8a81f5f4	cee64401	6aae8000	0
3	2	1	90000000	30000000	50000000	8a81f5f4	a75d4481	a6598000	0
taille des tables : 2^7									
3	2	1	03000000	48000000	18000000	fcb5f714	a4d07c01	be2f8001	0
3	2	1	21000000	12000000	0a000000	fcb5f714	77f22481	57eb8001	0
3	2	1	41000000	30000000	c0000000	fcb5f714	b3af7681	bb680001	0
3	2	1	0a000000	a0000000	11000000	fcb5f714	21549901	cf8f8001	0

générateur. Par exemple, on n'a trouvé aucun générateur de période $2^{256}-1$ avec $s = 3$ et $b_{r-q} = 0$. Probablement qu'il n'existe pas de tels polynômes étant donné le peu de paramètres possibles et que la recherche aléatoire de polynômes n'a donné aucun résultat.

Tous les générateurs contenus dans les tableaux 5.7, 5.8, 5.9 et 5.10 sont excellents du point de vue de l'équidistribution. On a pu trouver des générateurs de période $2^{96} - 1$ qui sont ME. Pour les générateurs de période $2^{800} - 1$, l'équidistribution est telle que $\Delta_\ell = 0$ pour $\ell = 1, \ldots, 16$. Les autres écarts en dimension Δ_ℓ, pour les

résolutions $\ell = 17, \ldots, 32$, sont donnés au tableau 5.11.

Tableau 5.8 – Générateurs de période $2^{256} - 1$, $r = 8$, $w = 32$.

r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$Q(z)$	$\mathbf{m}_1$	$\mathbf{m}_2$	V
taille des tables : 2^3									
8	5	3	a0000000	c0000000	30000000	d3e9de82	a13a9c81	5e6d801b	4
8	7	3	c0000000	50000000	60000000	d3e9de82	4c0ad481	ebd30053	5
8	7	4	60000000	90000000	c0000000	d3e9de82	b39e2581	36f30072	5
8	7	4	c0000000	90000000	30000000	d3e9de82	98fd4c01	eea3003c	5
taille des tables : 2^7									
8	5	2	03000000	44000000	28000000	ae397b58	05bf4081	eb67000c	4
8	6	3	41000000	05000000	60000000	ae397b58	1360c281	f3eb8004	4
8	5	2	24000000	81000000	30000000	ae397b58	9b6ae201	73ce0002	4
8	6	3	60000000	11000000	41000000	ae397b58	c76a8801	cf2f0001	4

Tableau 5.9 – Générateurs de période $2^{416} - 1$, $r = 13$, $w = 32$.

r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$Q(z)$	$\mathbf{m}_1$	$\mathbf{m}_2$	V
taille des tables : 2^3									
13	5	–	50000000	–	30000000	ae8b80e1	c55b6000	fcbd0015	23
13	5	–	50000000	–	30000000	ae8b80e1	360d4401	eb31803f	23
13	5	–	50000000	–	30000000	ae8b80e1	9b5c6101	b2cb0058	25
taille des tables : 2^7									
13	8	–	0c000000	–	28000000	c65a6fe2	977e1101	fac78000	20
13	5	–	21000000	–	44000000	c65a6fe2	df850601	e3758001	20
taille des tables : 2^7									
13	9	6	06000000	41000000	05000000	92bb39c1	5f9bca01	fd9d8006	9
13	8	4	24000000	06000000	48000000	92bb39c1	79f85701	3fcb000e	9
13	8	5	11000000	0c000000	30000000	92bb39c1	b8404581	22e30003	9
13	9	4	24000000	42000000	0c000000	92bb39c1	41830700	83b50008	9

5.9 Implantations en C

Dans cette section, on présente deux implantations de générateurs. Chacune utilise une technique différente pour effectuer les multiplications dans $\mathbb{F}_{2^w}$. Les deux

Tableau 5.10 – Générateurs de période $2^{800} - 1$, $r = 25$, $w = 32$.

r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$Q(z)$	$\mathbf{m}_1$	$\mathbf{m}_2$	V
taille des tables : 2^3									
25	11	–	30000000	–	50000000	e307bc0e	f7b31a80	af530001	72
25	11	–	30000000	–	50000000	e307bc0e	f0ba1601	ab4b0000	75
25	14	–	60000000	–	30000000	e307bc0e	cd1a1000	8dc70001	78
taille des tables : 2^7									
25	11	–	05000000	–	12000000	f282ea95	a6ea0881	4de58000	67
25	14	–	11000000	–	18000000	f282ea95	e18e4881	2d5f8001	67
25	9	–	09000000	–	28000000	f282ea95	fa3cc981	6cf88000	68
25	16	–	81000000	–	18000000	f282ea95	f7660f01	3fec0000	69
25	9	–	22000000	–	11000000	f282ea95	bff09280	59ca0000	69
25	9	–	60000000	–	28000000	f282ea95	2e89a401	de278000	69
taille des tables : 2^3									
25	21	6	30000000	c0000000	a0000000	e397e5c4	994aa401	5a9d8001	45
25	13	6	c0000000	30000000	a0000000	e397e5c4	9475ce01	e7ed0001	47
25	19	7	c0000000	60000000	90000000	e397e5c4	b3965001	2b6c8001	49
25	19	4	30000000	50000000	90000000	e397e5c4	366ac280	e7750001	49
25	18	8	60000000	90000000	50000000	e397e5c4	e2171580	9bd98001	52
25	13	11	a0000000	60000000	50000000	e397e5c4	d9769501	6ae58001	54
taille des tables : 2^7									
25	18	13	42000000	21000000	50000000	9f1f0184	c19ee400	7e778000	36
25	13	5	12000000	28000000	06000000	9f1f0184	9e60e080	736b0000	37
25	21	12	60000000	48000000	09000000	9f1f0184	ce20b000	785a8000	37
25	21	12	60000000	48000000	09000000	9f1f0184	68090201	c3cf8001	38
25	16	10	82000000	24000000	48000000	9f1f0184	26fc4000	69570000	38
25	20	9	05000000	06000000	41000000	9f1f0184	8d868001	505c8001	38

Tableau 5.11 – Valeurs de Δ_ℓ pour les générateurs du tableau 5.10 pour $\ell = 16, \ldots, 32$.

ℓ	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
$V = 36$							2	4	8	7	5	4	3	2	1		
$V = 37$							3	4	8	7	5	4	3	2	1		
$V = 37$		1		1	1	1	2	5	4	7	5	4	3	2	1		
$V = 38$					1		2	5	8	7	5	4	3	2	1		
$V = 38$					1	2	2	3	8	7	5	4	3	2	1		
$V = 38$					1	1	2	4	8	7	5	4	3	2	1		

techniques sont données dans les sections 5.7.1 et 5.7.2 et pour celles-ci, on utilise la base polynômiale $(1, \zeta, \ldots . \zeta^{w-1})$ pour représenter les éléments de $\mathbb{F}_{2^w}$, où $Q(z)$ est le polynôme minimal de ζ. Celui-ci est irréductible et de degré 32 (donc, $w = 32$). Les éléments de $\mathbb{F}_{2^w}$ sont représentés par des variables de type `unsigned int` qui sont considérées comme des vecteurs de 32 bits. Le bit le plus significatif est $\zeta^0 = 1$

et le moins significatif est ζ^{w-1}. Par exemple, `0x8000000fUL` représente l'élément $1+\zeta^{28}+\zeta^{29}+\zeta^{30}+\zeta^{31}$. Pour représenter le polynôme $Q(z) = z^{32} + \sum_{i=1}^{32} a_i z^{32-i}$, on utilise le vecteur $(a_{32}, \ldots, a_1)$ qui est emmagasiné dans une variable de type `unsigned int`.

Pour chacune de deux techniques, il est nécessaire de remplir des tableaux de valeurs qui sont utilisés par les générateurs. À cette fin, la fonction `multiply()` de la figure 5.3 a été conçue pour multiplier deux éléments de $\mathbb{F}_{2^w}$. L'argument `modPoly` est le polynôme $Q(z)$ et `a` et `b` sont les deux éléments de $\mathbb{F}_{2^w}$ à multiplier. La fonction `multiplyz()`, qui est utilisée par `multiply()`, retourne la valeur $a\zeta^k$ où a est l'élément de $\mathbb{F}_{2^w}$ représenté par `a`. Ces deux fonctions sont montrées à la figure 5.3.

```
#define W 32

/* Fonction qui retourne a zeta^k dans GF(2^32) */
unsigned int multiplyz (unsigned int a, int k, unsigned int modPoly)
{
   int i;
   if (k == 0)
      return a;
   else {
      for (i = 0; i < k; i++) {
         if (1UL & a) {
            a = (a >> 1) ^ modPoly;
         } else
            a = a >> 1;
      }
      return a;
   }
}
/* Fonction qui retourne a fois b, dans GF(2^32) */
unsigned int multiply (unsigned int a, unsigned int b,
                       unsigned int modPoly)
{
   int i;
   unsigned int res = 0, verif = 1UL;
   for (i = 0; i < W; i++) {
      if (b & verif)
         res ^= multiplyz (a, W - 1 - i, modPoly);
      verif <<= 1;
   }
   return res;
}
```

Figure 5.3 – Fichier `GenF2w.c`.

5.9.1 Implantation en C de générateurs avec coefficients quelconques

Dans cette sous-section, on présente une implantation en langage C de générateurs qui utilisent une récurrence linéaire dans $\mathbb{F}_{2^w}$. Cette récurrence a un polynôme caractéristique $P(z)$ dont les coefficients sont quelconques. Pour celle-ci, le polynôme caractéristique $P(z) \in \mathbb{F}_{2^w}[z]$ est le trinôme $z^r + b_{r-t}z^t + b_r$ qui est décrit à la première ligne de le tableau 5.4 et $L = w = 32$. Ainsi, on a $r = 25$, $t = 7$, $b_{18} = 1 + \zeta + \zeta^2 + \zeta^5 + \zeta^6 + \zeta^8 + \zeta^{10} + \zeta^{13} + \zeta^{14} + \zeta^{16} + \zeta^{20} + \zeta^{21} + \zeta^{23} + \zeta^{26}$, $b_{25} = \zeta^2 + \zeta^4 + \zeta^9 + \zeta^{10} + \zeta^{11} + \zeta^{12} + \zeta^{14} + \zeta^{16} + \zeta^{18} + \zeta^{19} + \zeta^{20} + \zeta^{25} + \zeta^{30}$ et $Q(z) = 1+z+z^2+z^3+z^4+z^6+z^9+z^{12}+z^{13}+z^{14}+z^{15}+z^{16}+z^{19}+z^{20}+z^{22}+z^{23}+z^{26}+z^{27}+z^{28}+z^{29}+z^{30}+z^{31}+z^{32}$. Pour ce générateur, l'équidistribution est telle que $V = 74$. Nous présentons l'implantation de ce générateur puisque son polynôme caractéristique n'a que deux coefficients non nuls (ce qui économise de l'espace mémoire et améliore la vitesse d'exécution) et obtient une bonne valeur de V.

L'implantation est donnée à la figure 5.4. Celle-ci utilise la méthode présentée à la section 5.7.1 et, aux équations (5.24) et (5.25), on utilise $c = 3$, $w_1 = 10$, $w_2 = 11$, $w_3 = 11$. Ceci veut dire que la multiplication par $\mathbf{b}_{r-t}$ ou $\mathbf{b}_r$ nécessite 3 tables qui prennent chacune 32×2^{10}, 32×2^{11} et 32×2^{11} bits, pour un total de 20 Ko. Puisque $Q(z)$ est un trinôme, il faut en tout 6 tables et elles occupent au total 40 Ko. Elles sont implantés par les tableaux `BRmT1`, `BRmT2`, `BRmT3` (pour $\mathbf{b}_{r-t}$) et `Br1`, `Br2`, `Br3` (pour $\mathbf{b}_r$) qui contiennent des valeurs de type `unsigned int`.

La fonction `InitTables2_32_800()` remplit les tableaux en utilisant la fonction `multiply()`. Celle-ci fait les multiplications dans $\mathbb{F}_{2^w}$ nécessaires afin de remplir chacun des tableaux en tenant compte des valeurs de $\mathbf{b}_{r-t}$ et $\mathbf{b}_r$.

Les macros `MultBRminusT` et `MultBr` implantent les multiplications par $\mathbf{b}_{r-t}$ et $\mathbf{b}_r$. L'argument `X` de chacun des macros représente un élément de $\mathbb{F}_{2^w}$ dans la base

polynômiale que l'on veut multiplier.

La fonction `F2wLFSR2_32_800()` est l'implantation du LFSR dans $\mathbb{F}_{2^w}$. On peut remarquer que l'implantation ressemble à celle d'un TGFSR [66] sans tempering. La différence réside dans les macros `MultBRminusT` et `MultBr`. Pour obtenir le T800 [66], il faudrait redéfinir les macros

```
#define MultBRminusT(X)  X
#define MultBr(X)        (X>>1) ^ (0x8ebfd028UL * (X & 0x00000001UL))
```

La valeur de t est la même pour le T800.

Dans cette implantation, on calcule, à chaque 25 appels de la fonction, la sortie des 24 prochains appels et la sortie du présent appel de fonction. Ceci est plus efficace du point de vue de la gestion de la mémoire cache, car l'état du générateur (qui contient une quantité relativement élevée d'information) est chargé dans la cache seulement une fois à toutes les 25 itérations.

Pour ce qui est du GCL polynômial, son implantation est donnée par la fonction `F2wPolyLCG2_32_800()`. Dans cette fonction, on utilise un « tableau rotatif » pour éviter de décaler les valeurs dans le vecteur d'état à chaque itération. Ainsi, à la n-ième itération, le coefficient $q_{n,r-i}$ est représenté par `x[(L+`i`)%R]`, pour $i = 1, \ldots, r$. On remarque que la fonction `F2wPolyLCG2_32_800()` prend en argument un pointeur vers une variable de type `double`. Ce paramètre représente le deuxième flot de nombres aléatoires qui est produit par le GCL polynômial (comme il est discuté à la section 5.4). Le deuxième bloc de 32 bits qui est mis à jour, soit `x[L+T]` ou `x[L+T-R]`, est utilisé pour produire ce deuxième flot de valeurs aléatoires.

```
#include "GenF2w.h"
#include "F2w2_32.h"
#define R 25
#define T 7
#define BrmT 0xe6a68d20UL
#define Br   0x287ab842UL
#define modP 0xfa4f9b3fUL

#define MultBrmT(X)(BrmT1[(X&0xffe00000UL)>>21] ^ BrmT2[(X&0x001ffc00UL)>>10] ^ BrmT3[X& 0x000003ffUL])
#define MultBr(X) (Br1[(X&0xffe00000UL)>>21]   ^ Br2[  (X&0x001ffc00UL)>>10] ^ Br3[X& 0x000003ffUL])

static unsigned int BrmT1[2048], BrmT2[2048], BrmT3[1024], Br1[2048], Br2[2048], Br3[1024];

void InitTables2_32_800 (void){
   unsigned int j;
   for (j = 0; j < 2048; j++) {
      BrmT1[j] = multiply (BrmT, j << 21, modP);
      BrmT2[j] = multiply (BrmT, j << 10, modP);
      Br1[j] = multiply (Br, j << 21, modP);
      Br2[j] = multiply (Br, j << 10, modP);
   }
   for (j = 0; j < 1024; j++) {
      BrmT3[j] = multiply (BrmT, j, modP);
      Br3[j] = multiply (Br, j, modP);
   }
}
double F2wLFSR2_32_800 (void){
   static int k = 0;
   static unsigned int x[R] = {     /* initial seeds : R=25 words */
      0x95f24dab, 0x0b685215, 0xe76ccae7, 0xaf3ec239, 0x715fad23,
      0x24a590ad, 0x69e4b5ef, 0xbf456141, 0x96bc1b7b, 0xa7bdf825,
      0xc1de75b7, 0x8858a9c9, 0x2da87693, 0xb657f9dd, 0xffdc8a9f,
      0x8121da71, 0x8b823ecb, 0x885d05f5, 0x4e20cd47, 0x5a9ad5d9,
      0x512c0c03, 0xea857ccd, 0x4cc1d30f, 0x8891a8a1, 0xa6b7aadb   };
   if (k == R) {                    /* generate R words at one time */
      int kk;
      for (kk = 0; kk < R - T; kk++) {
         x[kk] = MultBrmT (x[kk + T]) ^ MultBr (x[kk]);
      }
      for (; kk < R; kk++) {
         x[kk] = MultBrmT (x[kk + (T - R)]) ^ MultBr (x[kk]);
      }
      k = 0;
   }
   return ((double) x[k++] * 2.3283064370807974e-10);
}
double F2wPolyLCG2_32_800 (double *stream2){
   static int L = R - 1;
   static unsigned int x[R] = {     /* initial seeds : R=25 words */
      0x95f24dab, 0x0b685215, 0xe76ccae7, 0xaf3ec239, 0x715fad23,
      0x24a590ad, 0x69e4b5ef, 0xbf456141, 0x96bc1b7b, 0xa7bdf825,
      0xc1de75b7, 0x8858a9c9, 0x2da87693, 0xb657f9dd, 0xffdc8a9f,
      0x8121da71, 0x8b823ecb, 0x885d05f5, 0x4e20cd47, 0x5a9ad5d9,
      0x512c0c03, 0xea857ccd, 0x4cc1d30f, 0x8891a8a1, 0xa6b7aadb
   };
   if (L < 0)  L = R - 1;
   if (x[L]) {
      if (L + T < R) {
         x[L + T] ^= MultBrmT (x[L]);
         *stream2 = (double) x[L + T] * 2.3283064370807974e-10;
      } else {
         x[L + T - R] ^= MultBrmT (x[L]);
         *stream2 = (double) x[L + T - R] * 2.3283064370807974e-10;
      }
      x[L] = MultBr (x[L]);
   }
   return ((double) x[L--] * 2.3283064370807974e-10);
}
```

Figure 5.4 – Fichier `GenF2w2_32.c`.

5.9.2 Implantation en C de générateurs avec coefficients b_i particuliers

```
#include "GenF2w3_7.h"
#include "GenF2w.h"
#define R 25
#define W 32
#define BrmT 0x42000000UL
#define BrmQ 0x21000000UL
#define Br   0x50000000UL
#define modP 0x9f1f0184UL
#define T 18
#define Q 13
#define BrmT1 1
#define BrmT2 6
#define BrmQ1 2
#define BrmQ2 7
#define Br1  1
#define Br2  3
#define MultBrmT( X )  (((X&0xffffff80UL)>>BrmT1)^((X&0xffffff80UL)>>BrmT2)^TabBrmT[X & 0x0000007fUL])
#define MultBrmQ( X )  (((X&0xffffff80UL)>>BrmQ1)^((X&0xffffff80UL)>>BrmQ2)^TabBrmQ[X & 0x0000007fUL])
#define MultBr( X )    (((X&0xffffff80UL)>>Br1)  ^((X&0xffffff80UL)>>Br2)  ^TabBr  [X & 0x0000007fUL])
#define TemperM1 0xc19ee400UL
#define TemperM2 0x7e778000UL
#define SIZETABLE 7
static unsigned int TabBrmT[1UL<<SIZETABLE], TabBrmQ[1UL<<SIZETABLE], TabBr  [1UL<<SIZETABLE] ;

void InitTables3_7_800(void){
  unsigned int j ;
  for(j=0 ;j<(1UL<<SIZETABLE) ;j++)
    TabBrmT[j] = multiply (BrmT, j, modP) ;
  for(j=0 ;j<(1UL<<SIZETABLE) ;j++)
    TabBrmQ[j] = multiply (BrmQ, j, modP) ;
  for(j=0 ;j<(1UL<<SIZETABLE) ;j++)
    TabBr[j] = multiply (Br, j, modP) ;
}

double F2wPolyLCG3_7_800 ( double *stream2, double *stream3 ){
  static int L = R-1 ;
  static unsigned int  y, y2, y3, x[R] = {
    0x95f24dab, 0x0b685215, 0xe76ccae7, 0xaf3ec239, 0x715fad23, 0x24a590ad, 0x69e4b5ef, 0xbf456141,
    0x96bc1b7b, 0xa7bdf825, 0xc1de75b7, 0x8858a9c9, 0x2da87693, 0xb657f9dd, 0xffdc8a9f, 0x8121da71,
    0x8b823ecb, 0x885d05f5, 0x4e20cd47, 0x5a9ad5d9, 0x512c0c03, 0xea857ccd, 0x4cc1d30f, 0x8891a8a1,
    0xa6b7aadb} ;

  if(L<0)
    L=R-1 ;
  if(x[L]){
    if(L+T<R)   y2 = x[L+T]   ^= MultBrmT(x[L]) ;    else      y2 = x[L+T-R] ^= MultBrmT(x[L]) ;
    if(L+Q<R)   y3 = x[L+Q]   ^= MultBrmQ(x[L]) ;    else      y3 = x[L+Q-R] ^= MultBrmQ(x[L]) ;
    x[L]        = MultBr(x[L]) ;
  }
  y = x[L] ^ ((x[L]<<7) & TemperM1) ;    y = y    ^ ((y <<15)  & TemperM2) ;
  y2 = y2  ^ ((y2<<7 )  & TemperM1) ;    y2 = y2  ^ ((y2<<15)  & TemperM2) ;
  y3 = y3  ^ ((y3<<7 )  & TemperM1) ;    y3 = y3  ^ ((y3<<15)  & TemperM2) ;
  L-- ;
  *stream2 = (double)y2 * 2.3283064370807974e-10 ;
  *stream3 = (double)y3 * 2.3283064370807974e-10 ;
  return ( (double)y * 2.3283064370807974e-10 ) ;
}
```

Figure 5.5 – Fichier `GenF2w3_7.c`.

Cette section discute de l'implantation de générateurs dont le polynôme caractéristique $P(z) = z^r + b_{r-t}z^t + b_{r-q}z^q + b_r \in \mathbb{F}_{2^w}[z]$ a quatre coefficients non nuls de la forme

```
double F2wLFSR3_7_800 ( void ) {
  static int k = 0;
  static unsigned int y,x[R]={
    0x95f24dab, 0x0b685215, 0xe76ccae7, 0xaf3ec239, 0x715fad23, 0x24a590ad, 0x69e4b5ef, 0xbf456141,
    0x96bc1b7b, 0xa7bdf825, 0xc1de75b7, 0x8858a9c9, 0x2da87693, 0xb657f9dd, 0xffdc8a9f, 0x8121da71,
    0x8b823ecb, 0x885d05f5, 0x4e20cd47, 0x5a9ad5d9, 0x512c0c03, 0xea857ccd, 0x4cc1d30f, 0x8891a8a1,
    0xa6b7aadb};

  if (k==R) {
    int kk;
    for (kk=0;kk<R-T;kk++)
      x[kk] = MultBrmT(x[kk+T])   ^ MultBrmQ(x[kk+Q])   ^ MultBr(x[kk]);
    for (; kk<R-Q;kk++)
      x[kk] = MultBrmT(x[kk+T-R]) ^ MultBrmQ(x[kk+Q])   ^ MultBr(x[kk]);
    for (; kk<R;kk++)
      x[kk] = MultBrmT(x[kk+T-R]) ^ MultBrmQ(x[kk+Q-R]) ^ MultBr(x[kk]);
    k=0;
  }
  y = x[k] ^ ((x[k]<<7) & TemperM1);
  y = y    ^ ((y  <<15) & TemperM2);
  k++;
  return ((double) y * 2.3283064370807974e-10  );
}
```

Figure 5.6 – Fichier `GenF2w3_7.c` (suite).

particulière $b = \sum_{i=0}^{7} c_i \zeta^i$. Pour chacun de ces coefficients, il n'y a que 2 valeurs de c_i qui sont non nulles. Ces contraintes permettent une implantation rapide et efficace au niveau de la mémoire, comme il est discuté dans la section 5.7.2.

Le polynôme caractéristique utilisé est celui du seizième générateur du tableau 5.10 (celui dont $V = 36$). Il est tel que $b_{r-t} = \zeta + \zeta^6$, $b_{r-q} = \zeta^2 + \zeta^7$ et $b_r = \zeta + \zeta^3$. Le polynôme qui définit $\mathbb{F}_{2^w}$ est $Q(z) = 1 + z^3 + z^4 + z^5 + z^6 + z^7 + z^{11} + z^{12} + z^{13} + z^{14} + z^{15} + z^{23} + z^{24} + z^{29} + z^{32}$. À remarquer que ce générateur nécessite un tempering de Matsumoto-Kurita afin d'avoir l'équidistribution donnée.

Voici une explication du code de la figure 5.5. Les constantes `R`, `W`, `T` et `Q` sont les valeurs des entiers r, w, t et q . Les constantes `BrmT`, `BrmQ`, `Br` et `modP` sont les valeurs b_{r-t}, b_{r-q},b_r et $Q(z)$ représentées en notation hexadécimale. Les constantes `BrmT1`, `BrmT2`, `BrmQ1`, `BrmQ2`, `Br1`, `Br2` sont les exposants des coefficients non négatifs de b_{r-t}, b_{r-q} et b_r. Par exemple, puisque $b_{r-t} = \zeta + \zeta^6$, on a que `BrmT1`=1 et `BrmT2`=6.

Les macros `MultBrmT( X )`, `MultBrmQ( X )`, `MultBr( X )` permettent de multiplier un élément `X` de $\mathbb{F}_{2^w}$ par b_{r-t}, b_{r-q} et b_r. Ces macros implantent la technique

discutée à la section 5.7.2. Chacune de ces multiplications se fait en 2 décalages vers la droite et d'une consultation de tableau. Les tableaux utilisés pour chacun de ces macros sont les tableaux `TabBrmT[]`, `TabBrmQ[]` et `TabBr[]`. Ils sont remplis par la fonction `InitTables3_7_800()` qui doit être appelée avant d'utiliser le générateur. Les constantes `TemperM1` et `TemperM2` sont les vecteurs de bits $\mathbf{m}_1$ et $\mathbf{m}_2$ nécessaires pour le tempering de Matsumoto-Kurita.

L'implantation des fonctions `F2wPolyLCG3_7_800` et `F2wLFSR3_7_800` est semblable à celle des fonctions `F2wPolyLCG2_32_800` et `F2wLFSR2_32_800`. Par contre, ces générateurs appliquent un tempering de Matsumoto-Kurita à la sortie du générateur. Également, la fonction `F2wPolyLCG3_7_800` retourne trois flots de nombres aléatoires car $P(z)$ a quatre coefficients non nuls.

5.10 Saut d'un grand nombre de pas dans les récurrences

Souvent, pour l'implantation d'un générateur qui permet plusieurs flots de nombres aléatoires, comme c'est le cas pour les générateurs inclus dans la librairie SSJ [39] et la librairie F2wStreams dont il est question à la section 5.11, il est nécessaire de sauter dans la séquence d'un grand nombre ($> 2^{50}$) d'itérations efficacement. Dans cette section, on montre comment sauter en avant de 2^d itérations pour le GCL polynômial et pour le LFSR dans $\mathbb{F}_{2^w}$.

Dans le cas du GCL polynômial, étant donné un état $q_n(z)$, le problème est de trouver $q_{n+e}(z) = z^e q_n(z) \bmod P(z)$ où e est un grand nombre. La solution par la force brute n'est pas considérée étant donné le temps nécessaire trop grand pour arriver à la solution. Par exemple, supposons que l'on puisse avancer de 10^8 itérations en 1 seconde, alors pour avancer de 2^{50} itérations, cela prendrait 130 jours, ce qui est

impraticable pour les besoins de la simulation.

Une autre solution est de calculer directement $q_{n+e}(z)$. Ce problème est l'exponentiation dans le corps fini $\mathbb{F}_{2^w}[z]/P(z)$. Il existe plusieurs algorithmes d'exponentiation dans la littérature (par exemple, voir [72]). Dans ce qui suit, on discute d'un seul algorithme qui s'applique quand e est une puissance de 2.

Soit ζ, un élément de $\mathbb{F}_{2^w}$ dont le polynôme minimal $Q(z) \in \mathbb{F}_2[z]$ est irréductible. L'élément ζ est donc un générateur de $\mathbb{F}_{2^w}$. Soit $\kappa = \sum_{j=0}^{w-1} \kappa_i \zeta^i$ un élément de $\mathbb{F}_{2^w}$, où $\kappa_i \in \mathbb{F}_2$. Puisque $c^2 = c$ quand $c \in \mathbb{F}_2$ et que $\mathbb{F}_{2^w}$ est un corps fini de caractéristique 2, on a la propriété

$$\kappa^2 = \left(\sum_{j=0}^{w-1} \kappa_i \zeta^i \right)^2 = \sum_{j=0}^{w-1} \kappa_i \zeta^{2i}.$$

Pour calculer κ^{2^d}, on n'a qu'à utiliser cette propriété d fois pour arriver à notre fin. Ceci se résume à l'algorithme suivant.

Algorithme 5.1. Exponentiation de $\kappa \in \mathbb{F}_{2^w}$ par $e = 2^d$

1. Calculer $\zeta_i = \zeta^{2i} \bmod Q(\zeta)$ pour $i = 0, \ldots, w-1$.
2. $\tau \leftarrow \kappa$
3. Répéter l'action suivante d fois

   Soit $\tau = \sum_{j=0}^{w-1} \tau_j \zeta^j$ où $\tau_j \in \mathbb{F}_2$ pour $j = 0, \ldots, w-1$.

   $\tau \leftarrow \sum_{j=0}^{w-1} \tau_j \zeta_j$.

À la fin de cet algorithme, on a que τ est κ^e.

Cet algorithme s'exécute en $O(w + dw)$ opérations en supposant que la multiplication par ζ dans $\mathbb{F}_{2^w}$ et l'addition dans $\mathbb{F}_{2^w}$ sont des opérations unitaires. Si on met en mémoire les résultats ζ_i, $0 \leq i < w$, lors du premier appel à l'algorithme, chacun des appels suivants se fait en $O(dw)$ opérations.

Cet algorithme d'exponentiation dans $\mathbb{F}_{2^w}$ nous est utile pour le prochain algorithme qui effectue l'exponentiation dans $\mathbb{F}_{2^w}[z]/P(z)$. Puisque $\mathbb{F}_{2^w}[z]/P(z)$ est aussi

un corps fini de caractéristique 2, on peut bénéficier de la propriété suivante :

$$(\xi_0 + \xi_1 z + \ldots + \xi_{r-1} z^{r-1})^2 \equiv (\xi_0^2 + \xi_1^2 z^2 + \ldots + \xi_{r-1}^2 z^{2(r-1)}) \bmod P(z).$$

Pour calculer $z^{2^d} \bmod P(z)$, on utilise l'algorithme suivant qui utilise cette propriété d fois.

Algorithme 5.2. Exponentiation de $q(z) \in \mathbb{F}_{2^w}[z]/P(z)$ par $e = 2^d$

1. Calculer $p_i(z) = z^{2i} \bmod P(z)$ pour $i = 0, \ldots, r-1$.
2. $\xi(z) \leftarrow q(z)$
3. Répéter l'action suivante d fois

   Soit $\xi(z) = \sum_{j=0}^{r-1} \xi_j z^j$ où $\xi_j \in \mathbb{F}_{2^w}$ pour $j = 0, \ldots, r-1$.

   $\xi(z) \leftarrow \sum_{j=0}^{r-1} \xi_j^2 p_j(z)$.

À la fin de cet algorithme, on obtient $\xi(z) = q(z)^{2^d} \bmod P(z)$.

À l'étape 3 de l'algorithme, on pourrait utiliser l'algorithme 5.1 pour calculer ξ_j^2. En supposant que la multiplication dans $\mathbb{F}_{2^w}$ se fait en $O(w^2)$ opérations, on remarque que l'algorithme s'exécute en $O(dr^2w^2)$ opérations.

L'algorithme 5.2 nous donne une méthode efficace pour calculer l'état d'un GCL polynômial après 2^d itérations. Du même coup, cet algorithme nous donne une méthode pour calculer l'état d'un LFSR 2^d itérations en avant. Voici maintenant comment utiliser l'algorithme 5.2 afin de calculer m_{n+2^d}.

De (5.1), on a

$$\begin{aligned} m_{n+1} &= \sum_{i=1}^{r} b_i m_{n-i+1} \qquad (5.29) \\ m_{n+2} &= \sum_{i=1}^{r} b_i m_{n-i+2} \\ &= b_1 m_{n+1} + \sum_{i=2}^{r} b_i m_{n-i+2} \end{aligned}$$

On remplace m_{n+1} par (5.29) et on obtient

$$m_{n+2} = b_1 \sum_{i=1}^{r} b_i m_{n-i+1} + \sum_{i=2}^{r} b_i m_{n-i+2}$$

On peut répéter ce stratagème pour calculer $m_{n+3}, \ldots, m_{n+e}$ en obtenant toujours ces valeurs en termes de $m_n, \ldots, m_{n-r+1}$. À un certain moment, on obtient

$$m_{n+j} = \sum_{i=1}^{r} v_{j,i} m_{n-i+1}$$

où $v_{j,i} \in \mathbb{F}_{2^w}$ et à l'étape suivante,

$$\begin{aligned}
m_{n+j+1} &= \sum_{i=1}^{r} v_{j,i} m_{n-i+2} \\
&= v_{j,1} m_{n+1} + \sum_{i=2}^{r} v_{j,i} m_{n-i+2} \\
&= v_{j,1} \sum_{i=1}^{r} b_i m_{n-i+1} + \sum_{i=2}^{r} v_{j,i} m_{n-i+2} \\
&= v_{j,1} \sum_{i=1}^{r} b_i m_{n-i+1} + \sum_{i=1}^{r-1} v_{j,i+1} m_{n-i+1} \\
&= \sum_{i=1}^{r} v_{j+1,i} m_{n-i+1}
\end{aligned}$$

où

$$v_{j+1,i} = \begin{cases} v_{j,1} b_i + v_{j,i+1} & \text{si } 1 \leq i < r \\ v_{j,1} b_r & \text{si } i = r \end{cases}.$$

Soit $v_j = (v_{j,r}, \ldots, v_{j,1})$ et $v_0 = (b_r, \ldots, b_1)$. On observe que

$$v_{j+1} = (0, v_{j,r}, \ldots, v_{j,2}) + v_{j,1} v_0. \tag{5.30}$$

L'équation (5.10) est la même que (5.30), sauf dans une représentation vectorielle. On pourrait donc faire avancer un LFSR dans $\mathbb{F}_{2^w}$ de j itération en avant à l'aide d'un GCL polynômial dans $\mathbb{F}_{2^w}[z]/P(z)$. Pour ce faire, il suffit d'initialiser le GCL polynômial avec $q_0(z) = b_1 z^{r-1} + \cdots + b_{r-1} z + b_r$ et calculer $q_e(z) = z^e q_0(z) \bmod P(z)$ où $e = 2^d$. On obtient

$$m_{n+e} = q_{e,1} m_n + \ldots + q_{e,r} m_{n-r+1}. \tag{5.31}$$

De cette dernière équation, on déduit l'algorithme suivant.

Algorithme 5.3. ([4]) Calcul de $(m_{n+e}, \ldots, m_{n+e-r+1})$, l'état d'un LFSR $e = 2^d$ itérations en avant à partir du vecteur $(m_n, \ldots, m_{n-2r+1})$.

1. $q_0(z) \leftarrow b_r + b_{r-1}z + \cdots + b_1 z^{r-1}$
2. Calculer $q_e(z) = z^e q_0(z) \bmod P(z)$ à l'aide de l'algorithme 5.2.
3. Pour $j = 0, \ldots, r-1$,

$$m_{n+e-j} \leftarrow q_{e,1} m_{n-j} + \ldots + q_{e,r} m_{n-r+1-j}$$

Dans cet algorithme, afin de calculer l'état complet d'un LFSR, il est nécessaire de connaître les $2r$ valeurs m_{n-j} pour $j = 0, \ldots, 2r-1$. Dans une implantation efficace d'un LFSR, on ne dispose que du vecteur d'état $(m_n, \ldots, m_{n-r+1})$. Par contre, il est possible de calculer $(m_{n-r}, \ldots, m_{n-2r+1})$ en utilisant un LFSR dont le polynôme caractéristique est

$$P^*(z) = \frac{z^r P(1/z)}{b_r} = \sum_{i=0}^{r} \frac{b_{r-i}}{b_r} z^i = z^r + \sum_{i=1}^{r} \tilde{b}_i z^{r-i}$$

où $b_0 = 1$. Un LFSR avec ce polynôme caractéristique produit la même séquence cyclique, mais dans l'ordre inverse. Pour construire ce LFSR, on assigne $\tilde{m}_{n-j} = m_{n-r+j+1}$ pour $j = 0, \ldots, r-1$. La récurrence est

$$\tilde{m}_n = \sum_{i=1}^{r} \tilde{b}_i \tilde{m}_{n-i}.$$

Grâce à cette récurrence, on calcule $\tilde{m}_{n+1}, \ldots, \tilde{m}_{n+r}$. On obtient alors

$$m_{n-j} = \tilde{m}_{n-r+j+1}$$

pour $j = r, \ldots, 2r-1$.

En pratique, pour implanter des générateurs à plusieurs flots de variables aléatoires, il n'est pas nécessaire d'avoir l'état du générateur *exactement* 2^d itérations en

avant. On pourrait se contenter d'obtenir l'état $2^d + r$ itérations en avant. On pourrait utiliser l'algorithme 5.3 pour calculer $(m_{n+r+e}, \ldots, m_{n+e+1})$ à partir du vecteur $(m_{n+r}, \ldots, m_{n-r+1})$ qui peut être facilement connu en itérant r fois la récurrence.

À la prochaine section, on présente la bibliothèque informatique F2wStreams qui utilise les algorithmes de cette section. Cette librairie implante un générateur de nombres aléatoires qui peut être divisé en plusieurs flots de nombres aléatoires.

5.11 F2wStreams

À l'annexe C, on retrouve le guide pour la bibliothèque de fonctions F2wStreams. Celle-ci permet d'implanter plusieurs flots de nombres aléatoires à partir d'un même générateur basé sur une récurrence linéaire dans un corps de caractéristique 2. Tous les générateurs qui sont donnés à la section 5.8 peuvent être implantés par cette bibliothèque de fonctions. Le générateur duquel proviennent les flots de nombres aléatoires est appelé *générateur de base*. Le développement de la bibliothèque a été fortement inspiré de celle exposée dans [49].

Soit I_1, l'état initial du générateur de base et

$$G = \{u_n\}_{n \geq 0},$$

la séquence de valeurs produites par le générateur initialisé à I_1. Les flots sont produits en découpant la séquence G en sections. Le g-ième flot est défini par

$$S_g = \{u_{Z(g-1)+n}\}_{n \geq 0}$$

où Z est une puissance de 2. L'état I_g est l'état du générateur qui produit le flot S_g. À noter que $S_1 = G$.

Chaque flot est divisé en sous-flots. Le h-ième sous-flot du g-ième flot est défini

par

$$T_{g,h} = \{u_{Z(g-1)+W(h-1)+n}\}_{n\geq 0}$$

où W est une puissance de 2. Les valeurs W et Z, qui sont définies par l'utilisateur, doivent être telles que $W < Z$ et jugées assez grandes pour qu'une application n'utilise qu'une petite fraction de W valeurs aléatoires. Pour le g-ième flot, il n'y a qu'un seul sous-flot actif. Soit h^*, le numéro du sous-flot actif. Quand l'utilisateur demande un nombre aléatoire du g-ième flot, celui-ci retourne le prochain nombre aléatoire du flot T_{g,h^*}.

Pour un flot S_g donné, il y a quatre racines d'importance, soit I_g, C_g, B_g et N_g. La racine I_g est la racine du début du flot. La racine C_g est l'état actuel du générateur pour le sous-flot actif. La racine B_g indique le début du prochain sous-flot et N_g indique le début du prochain flot. Ces racines sont manipulées par l'implantation et ceci se fait à l'insu de l'utilisateur.

Pour la bibliothèque F2wStreams, on utilise par défaut $L = w = 32$. Également, puisque les séquences de nombres aléatoires produites par les LFSR et les GCL polynômiaux sont les mêmes (pour un polynôme caractéristique $P(z)$ donné), alors seulement le GCL polynômial est implanté.

Pour une application de simulation, un utilisateur crée un seul générateur de base (celui de son choix parmi ceux trouvés à la section 5.8). Il peut créer un flot pour chaque processus stochastique de la simulation. Pour chaque répétition de la simulation, il peut changer de sous-flot. Un exemple d'un application est donné dans le guide de F2wStreams.

Exemple 5.7. Un autre exemple d'utilisation d'un générateur à plusieurs flots est pour un problème de simulation où il faut créer des objets avec plusieurs attributs qui sont choisis selon certaines lois de probabilité. Concrètement, supposons qu'on veuille simuler une file d'attente avec un seul serveur. Pour chaque client qui arrive, on doit générer deux variables aléatoires, son heure d'arrivée et son temps de service,

selon les lois de probabilités prescrites par le modèle. Si on utilise l'inversion pour produire ces variables aléatoires, alors une variable aléatoire uniforme est nécessaire pour chacune d'elle. Dans l'implantation du modèle, on n'a qu'à créer deux flots, un pour chaque variable aléatoire propre à chaque client.

L'avantage de cette implantation est que la synchronisation des variables aléatoires est plus facile. La synchronisation est une technique de simulation qui permet de comparer des modèles stochastiques semblables selon une mesure de performance définie. En simulant les divers modèles dans les « mêmes conditions de hasard », on arrive à comparer les modèles de manière efficace (voir [28] pour plus de détails sur ces techniques de simulation). Pour les modèles de file d'attente, en utilisant un flot pour les temps d'attente et un flot pour les temps d'arrivées, on peut garantir que les temps d'arrivées et les temps de services sont les mêmes pour différents modèles, ce qui n'est pas le cas si on utilise un seul flot pour tous les événements aléatoires des systèmes.

5.12 Tests Statistiques

L'équidistribution est une mesure du niveau d'uniformité d'ensembles de points sur toute la période d'un générateur. Si un générateur performe bien par rapport à l'équidistribution, celui-ci devient un excellent candidat pour devenir un bon générateur de nombres pseudo-aléatoires. Afin de décider si un candidat est bon, il faut que celui-ci performe bien par rapport à plusieurs tests statistiques empiriques. Ces tests vérifient l'hypothèse nulle

$\mathcal{H}_0 : u_0, u_1, \ldots$ sont des variables aléatoires uniformes indépendantes sur $[0, 1)$.

Il existe des batteries de tests statistiques qui contiennent plusieurs tests qui vérifient $\mathcal{H}_0$. Dans TestU01 [48], on implante plusieurs batteries de tests tels que

Crush et Rabbit. Il existe aussi la populaire batterie de tests DIEHARD [58] et la suite de tests de NIST [93]. Seulement de très bons générateurs réussissent à passer tous les tests inclus dans toutes ces batteries.

Les deux générateurs présentés à la section 5.9 ont été testés en utilisant les batteries DIEHARD, Crush et Rabbit. À part les tests qui dépendent des relations linéaires entre les bits, aucun test n'a réussi à mettre en doute sérieusement l'hypothèse nulle. Les tests qui dépendent des relations linéaires sur des bits sont échoués à coup sûr puisque les nombres aléatoires sont effectivement produits par des relations linéaires sur des bits (comme il est expliqué à la section 2.7). Ces relations étant déterministes, on ne peut donc pas s'attendre à ce qu'elles se comportent de manière aléatoire. Par contre, en ayant des relations linéaires de grand ordre, on s'assure que ce défaut ne soit pas un handicap pour les générateurs, ce qui est le cas pour nos générateurs. On peut donc conclure que ces générateurs sont de bonne qualité puisqu'ils performent bien par rapport à l'équidistribution et, aussi, parce qu'aucun test statistique n'a réussi à mettre en doute sérieusement l'hypothèse $\mathcal{H}_0$.

5.13 Performances

Dans cette section, on expose les performances, du point de vue de la rapidité d'exécution, des quatre générateurs donnés dans la section 5.9. Ces générateurs sont comparés à divers générateurs courants. Ceux-ci sont le Mersenne Twister [69], le MRG32k3a [36] et TT800 [67]. Les temps d'exécutions sont donnés à la table 5.12. Le test consistait à chronométrer le temps nécessaire à un générateur pour exécuter 10^9 itérations et additionner tous les nombres produits. Dans ces tables, on donne aussi la valeur de V, qui indique l'équidistribution, pour chacun des générateurs. Pour le MT19937, sa valeur est mise en parenthèse puisqu'il ne s'agit pas d'un bon point de comparaison étant donné que ce dernier a une période beaucoup plus grande.

Pour les générateurs qui peuvent produire des sorties multiples à chaque itération (comme il est expliqué à la section 5.4), nous avons testé les deux types d'implantations : celle qui retourne une seule valeur par itération et celle qui retourne plusieurs valeurs par itération. Pour celle qui retourne plusieurs valeurs par itération, nous avons chronométré le temps nécessaire pour exécuter 10^9 itérations et additionner toutes les sorties générées. Ce temps a été divisé par le nombre de sorties par itérations, afin de comparer équitablement la vitesse de ces générateurs aux autres. Ces générateurs sont indiqués par un astérisque (*).

Le test a été exécuté sur un ordinateur avec 1 Go de mémoire vive équipé d'un processeur Intel Pentium 4 2.8Ghz. Le système d'exploitation était Linux, distribution Redhat 8.0, et le programme a été compilé avec gcc 3.2 avec l'option d'optimisation "-O2".

Tableau 5.12 – Temps nécessaire pour itérer 10^9 fois et additionner tous les nombres aléatoires produits.

Générateur	Temps (s)	V
F2wLFSR2_32_800	39.5	74
F2wPolyLCG2_32_800	36.4	74
F2wPolyLCG2_32_800(*)	31.3	74
F2wLFSR3_7_800	32.8	36
F2wPolyLCG3_7_800	40.7	36
F2wPolyLCG3_7_800(*)	33.0	36
TT800	44.0	261
MT19937	31.6	(6750)
MRG32k3a	101	–

Les résultats indiquent que le MT19937 est toujours le plus rapide des générateurs

testés. Par contre, la différence avec nos nouveaux générateurs n'est pas très grande. Alors, pourquoi devrions-nous favoriser ces nouveaux générateurs par rapport au MT19937 ? La réponse est tout ce qu'il y a de plus pragmatique : le MT19937 est difficile à utiliser à cause de son vecteur d'état qui est énorme. Un problème que pourrait rencontrer un utilisateur du MT19937 est la difficulté de choisir l'état initial. On pourrait utiliser un autre générateur pour l'initialiser. Par contre, si on choisit un autre générateur qui a un plus petit nombre d'états que le MT19937, alors ce ne sont pas toutes les racines initiales du MT19937 qui sont disponibles à travers cet autre générateur. Ce problème existe également pour nos nouveaux générateurs et le TT800, mais ces derniers sont beaucoup moins sensibles à une mauvaise initialisation. Par exemple, si un utilisateur initialise l'état avec un vecteur ne contenant que très peu de bits mis à un, alors le vecteur d'état risque d'avoir peu de bits non nuls pendant un très grand nombre d'itérations. Plus l'état est grand, plus le problème persiste longtemps. Pour plus de renseignement sur la manière d'initialiser ces générateurs, voir le site internet `http://random.mat.sbg.ac.at/news/seedingTT800.html`. Nous reviendrons sur ce sujet au chapitre 7

Un autre problème du MT19937 est la manière de l'implanter pour une utilisation avec plusieurs flots de nombres aléatoires. On pourrait utiliser une racine aléatoire pour chaque flot, mais, dans ce cas, on doit garder en mémoire toutes les racines utilisées pour répéter les simulations. Un autre approche est d'utiliser différents Mersenne twisters de petite période (par exemple, $2^{521}-1$) pour chaque flot et de les créer dynamiquement. C'est ce qui est fait dans [70]. Un défaut de cette approche est qu'il faut faire une recherche de paramètres pour trouver un générateur de pleine période et aussi une recherche pour trouver le bon tempering à appliquer au générateur pour chaque flot qui est créé. Ces deux recherches doivent être faites à chaque nouveau flot et le temps pour les effectuer peut être non négligeable. Aussi, la qualité du générateur risque de n'être pas aussi bonne que si on fait une recherche approfondie des paramètres.

Dans le cas de nos nouveaux générateurs, le problème de l'initialisation est présent mais beaucoup moins accentué que pour le MT19937 (une expérience présentée au chapitre 7 tend à le démontrer). De plus, l'implantation d'un générateurs à plusieurs flots de nombres aléatoires est relativement facile, comme il est expliqué à la section 5.11. L'état prend beaucoup moins de mémoire que pour le MT19937. Le seul défaut potentiel de ces nouveaux générateurs est l'utilisation des tables qui prennent de la mémoire. Par contre, dans une implantation avec plusieurs flots, un seul ensemble de tables est nécessaire, donc le « coût » est amorti sur tous les flots utilisés, ce qui fait, qu'en bout de ligne, l'utilisation de la mémoire pourrait être beaucoup moins grande pour nos nouveaux générateurs.

Chapitre 6

Ensembles de points pour l'intégration quasi-Monte Carlo

Dans ce chapitre, on introduit de nouveaux ensembles de points pour l'intégration quasi-Monte Carlo construits à partir des récurrences dont il est question au chapitre 5. Pour ces ensembles de points, nous ne sommes pas intéressés à ce que chaque point soit considéré comme une variable aléatoire, mais plutôt à ce que l'ensemble de points couvre de manière uniforme l'hypercube unitaire $[0,1)^t$. Une particularité des ensembles de points construits à partir d'une récurrence est que la dimension est infinie. On dit qu'ils ont une dimension infinie puisqu'il est toujours possible de rajouter une coordonnée à un point en effectuant une itération de plus à la récurrence. Ceci est fort utile lorsque la dimension du problème n'est pas connue d'avance. Les ensembles de points que nous allons définir sont aussi stationnaires dans la dimension par le lemme 4.8. Ces deux propriétés constituent deux avantages importants des ensembles de points introduits dans ce chapitre. En général, les réseaux digitaux n'ont aucune de ces deux propriétés.

On utilise des critères basés sur l'équidistribution, la q-valeur et la distance mini-

male afin de trouver de bons ensembles de points. Cette recherche se fait a l'aide de la bibliothèque informatique REGPOLY. Une fois que nous avons trouvé les meilleurs ensembles de points possibles par rapport à chacun des critère, nous évaluons leur performance sur différents problèmes d'intégration. Nous allons comparer les résultats obtenus avec ceux obtenus par des ensembles de points de Sobol randomisés [98, 99]. Pour conduire ces expériences, nous utilisons la bibliothèque informatique « Stochastic Simulation in Java » (SSJ), un outil efficace pour la simulation stochastique, dans laquelle nous avons programmé une implantation pour les ensembles de points présentés. Parmi tant d'autres fonctionnalités, il permet l'utilisation de différents ensembles de points pour l'intégration quasi-Monte Carlo.

6.1 Intégration quasi-Monte Carlo randomisée

Soit $P_n \in [0,1)^t$, un ensemble de n points jugé uniforme (par un critère quelconque). Nous voulons évaluer l'intégrale d'une fonction $f : [0,1)^t \to \mathbb{R}$ sur le domaine $[0,1)^t$, c'est-à-dire

$$I(f) = \int_{[0,1)^t} f(\mathbf{x}) d\mathbf{x}$$

à l'aide de l'estimateur *quasi-Monte Carlo*

$$\hat{\mu}_m = \frac{1}{m} \sum_{j=1}^{m} Q_{n,j}$$

où

$$Q_{n,j} = \frac{1}{n} \sum_{\mathbf{u} \in P_{n,j}} f(\mathbf{u})$$

et $P_{n,j}$ est l'ensemble de points P_n auquel on a appliqué une randomisation. La randomisation consiste en une transformation que l'on applique à un ensemble de points qui fait en sorte que chacun des points de $P_{n,j}$ peut être considéré comme une variable aléatoire uniforme sur $[0,1)^t$, mais qui conserve l'uniformité (ou l'enviable structure) de l'ensemble de points original. Dans ce cas, l'estimateur $\hat{\mu}$ est un estimateur sans

biais de l'intégrale $I(f)$. Quand on utilise l'intégration quasi-Monte Carlo, on veut profiter du fait que P_n couvre bien le domaine d'intégration pour obtenir une petite erreur d'intégration. Par contre, si on estime $I(f)$ avec seulement $m = 1$ ensemble de points, on n'a aucune idée de l'ordre de grandeur de l'erreur d'intégration. En répétant plusieurs fois, avec plusieurs ensembles de points randomisés indépendamment, il est possible d'estimer l'erreur d'intégration.

Ainsi, les $Q_{n,j}$ sont des variables aléatoires indépendantes et identiquement distribuées de moyenne $\mathrm{E}[Q_{n,j}] = I(f)$. La variance de $Q_{n,j}$ peut être estimée par

$$\hat{\sigma}_m^2 = \frac{1}{m-1}\sum_{j=1}^{m}(Q_{n,j} - \hat{\mu}_m)^2.$$

On a que $\sigma_{QMC,n}^2 = \mathrm{E}[\hat{\sigma}_m^2] = \mathrm{Var}[Q_{n,j}] = m\mathrm{Var}[\hat{\mu}_m]$. Par le théorème de la limite centrale, on a que

$$\sqrt{m}(\hat{\mu}_m - I(f))/\hat{\sigma}_m \Rightarrow \mathrm{N}(0,1)$$

quand $m \to \infty$ et on peut construire un intervalle de confiance sur la valeur de $I(f)$. Plus la valeur de $\hat{\sigma}_m$ est petite, plus l'intervalle est étroit. Pour une bonne estimation de $I(f)$, il est préférable d'avoir une petite valeur de $\sigma_{QMC,n}^2$.

L'estimateur dit *Monte Carlo* est

$$\bar{\mu}_n = \frac{1}{n}\sum_{i=1}^{n} f(\mathbf{u}_i)$$

où les $\mathbf{u}_i$ sont des variables aléatoires indépendantes et uniformes sur $[0,1)^t$. La variance échantillonnale des $f(\mathbf{u}_i)$ est

$$\bar{\sigma}_n^2 = \frac{1}{n-1}\sum_{i=1}^{n}(f(\mathbf{u}_i) - \bar{\mu}_n)^2.$$

On a que $\sigma_{MC}^2 = \mathrm{E}[\bar{\sigma}_n^2] = \mathrm{Var}[f(\mathbf{u}_i)] = n\mathrm{Var}[\bar{\mu}_n]$.

Pour évaluer la performance d'un ensemble de points, on utilise comme référence le facteur de réduction de variance qui est défini par

$$\rho = \frac{1}{n}\frac{\sigma_{MC}^2}{\sigma_{QMC,n}^2}$$

où n est le nombre de points dans l'ensemble de points quasi-Monte Carlo. Plus le rapport est grand, meilleur est jugé l'ensemble de points P_n. Pour estimer ρ, on utilise

$$\hat{\rho} = \frac{1}{n}\frac{\bar{\sigma}^2_{n'}}{\hat{\sigma}^2_m}.$$

où n est le nombre de points dans l'ensemble de points quasi-Monte Carlo, m est le nombre de randomisations utilisées dans la méthode quasi-Monte Carlo et n' est le nombre de variables aléatoires uniformes sur $[0,1]^t$ utilisées dans la méthode Monte Carlo.

Pour juger de la qualité de nos résultats, on aimerait obtenir un intervalle de confiance de niveau α sur la valeur de ρ. On sait que le rapport de deux variances empiriques suit la distribution F [106] en supposant que les données suivent une loi normale. Nous pourrions prétendre que le rapport $\bar{\sigma}^2_n/\hat{\sigma}^2_m$ suit une loi de distribution F. Mais quelques vérifications, pour quelques fonctions difficiles à intégrer, nous ont convaincus que l'hypothèse n'est pas valide. Quand la fonction est difficile à intégrer, l'estimateur $\hat{\sigma}^2_m$ a une grande variance. Par exemple, pour la fonction f_6 (dont la description est donnée plus loin dans ce chapitre) en 75 dimensions, avec un ensemble de 2^{10} points, $\hat{\sigma}^2_{100}$ peut prendre des valeurs variant de 7×10^{-3} à 3. Étant donné ces difficultés, nous n'allons pas intégrer des fonctions en très haute dimension pour éviter que l'estimateur de la variance contienne trop de bruit. Certaines fonctions à intégrer sont telles qu'on sait que $\sigma^2_{MC} = 1$, ce qui simplifie nos calculs. Dans les autres cas, on utilise un grand nombre de valeurs aléatoires uniformes sur $[0,1]^t$ pour estimer σ^2_{MC} et on prend cet estimateur pour la vraie valeur. Cette simplification n'a pas de conséquences véritables car nous sommes intéressés à connaître seulement l'ordre de grandeur de ρ.

Pour les besoins de cette thèse, nous ne considérons que deux types de randomisations. Celles-ci sont particulièrement bien adaptées dans le cas où l'ensemble de points est un réseau digital. Soit $P_n = \Psi_t(\mathbf{C}, H, L, J)$, le réseau digital considéré. Dans la définition 4.2, on observe qu'à chaque point $(w_0, \ldots, w_{t-1}) \in P_n$ correspond

un vecteur de bits $(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1})$. Ce dernier vecteur de bits est composé de t vecteurs de L bits. On se sert de ces vecteurs pour effectuer les randomisations.

La première randomisation est obtenue par l'algorithme suivant :

Algorithme 6.1. Translation digitale sur $P_n = \Psi_t(\mathbf{C}, H, L, J)$.

1. Choisir aléatoirement et uniformément un vecteur de bits $\mathbf{U}_t = (\mathbf{u}_0, \ldots, \mathbf{u}_{t-1})$ où $\mathbf{u}_j \in \mathbb{F}_2^L$.
2. À chaque point $(w_0, \ldots, w_{t-1}) \in P_n$ correspond un vecteur $(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1})$. Pour chacun d'eux, calculer $(w'_0, \ldots, w'_{t-1})$ en prenant le vecteur

$$(\mathbf{y}'_0, \ldots, \mathbf{y}'_{t-1}) = (\mathbf{y}_0, \ldots, \mathbf{y}_{t-1}) \oplus \mathbf{U}_t.$$

3. L'ensemble de points randomisé est constitué de tous les points $(w'_0, \ldots, w'_{t-1})$ ainsi obtenus. On le note $P_n(\mathbf{U}_t)$.

La deuxième randomisation considérée est appelé le *mixage linéaire*. Elle est obtenue par l'algorithme suivant :

Algorithme 6.2. Mixage linéaire sur $P_n = \Psi_t(\mathbf{C}, H, L, J)$.

1. Choisir aléatoirement et uniformément une matrice M de bits, de dimension $L \times L$, parmi une classe bien précise de matrices.
2. À chaque point $(w_0, \ldots, w_{t-1}) \in P_n$ correspond un vecteur $(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1})$. Pour chacun d'eux, calculer $(w'_0, \ldots, w'_{t-1})$ en prenant le vecteur

$$(\mathbf{y}'_0, \ldots, \mathbf{y}'_{t-1}) = M(\mathbf{y}_0, \ldots, \mathbf{y}_{t-1})$$

3. L'ensemble de points randomisé est constitué de tous les points $(w'_0, \ldots, w'_{t-1})$ ainsi obtenus. On le note $P_n(M)$.

À la première étape de l'algorithme 6.2, on fait référence à une « classe bien précise de matrices ». La classe de matrices que l'on considère dans cette thèse est celle des

matrices où les entrées au-dessus de la diagonale sont nulles, ceux dans la diagonale sont tous 1. Pour les entrées sous la diagonale, pour les k premières lignes, elles sont 1 ou 0 et pour les $L-k$ dernières, elles sont toutes nulles. Cette classe de matrices correspond à un type de mixage linéaire introduit par Matoušek [64]. Bien que le mixage linéaire ne se limite pas seulement à cette classe de matrices, nous allons nous restreindre à celle-ci. Ainsi, pour le reste de cette thèse, le terme mixage linéaire correspondra à cette classe de matrices. Ce mixage linéaire est intéressant puisque l'ensemble de points $P_n(M)$ a la même q-valeur que l'ensemble de points original [64] (tout comme la translation digitale). Nous ferons aussi référence à ce type de mixage linéaire au chapitre 8 pour expliquer certaines propriétés de générateurs de nombres aléatoires.

On peut combiner les deux méthodes de randomisation en appliquant successivement le mixage linéaire et la translation digitale. L'ensemble de points ainsi obtenu est noté $P_n(M, \mathbf{U}_t)$. Avec seulement le mixage linéaire, les points ne suivent pas la loi uniforme. Par contre, en ajoutant la translation digitale, chacun des points suit une loi uniforme et l'estimateur quasi-Monte Carlo résultant est sans biais.

6.2 Sortie $\mathbf{y}_n$ à utiliser pour quasi-Monte Carlo

Pour intégrer une fonction à l'aide des récurrences définies au chapitre 5, on utilise l'ensemble de points $\Psi_t(\mathbf{X}, B, L)$. Habituellement, les ensembles de points utilisés pour l'intégration quasi-Monte Carlo (QMC) ont une cardinalité inférieure à 2^{20}. Puisque la cardinalité de $\Psi_t(\mathbf{X}, B, L)$ est 2^{rw}, il faut choisir r et w de manière à ce que le produit rw ne soit pas trop grand. Au chapitre précédent, on avait toujours $L = w = 32$ puisque c'était pratique du point de vue de l'implantation et qu'on recherchait des générateurs avec de longues périodes. Dans le cas qui nous intéresse maintenant, il n'est pas possible d'avoir $L = w = 32$. Quand $r \geq 2$, il faut nécessairement que

$w \leq 10$ pour obtenir un nombre de points inférieur à 2^{20}, mais on aimerait tout de même avoir la résolution à la sortie L la plus près de 32 possible. Pour ces raisons, on doit modifier la manière de construire la sortie.

On doit également modifier les récurrences puisque les récurrences définies au chapitre 5 ne permettent pas d'obtenir une très bonne équidistribution avec la nouvelle sortie que nous allons définir. Dans cette section, nous définissons ces nouvelles récurrences et ces nouvelles manières de construire $\mathbf{y}_n$ afin que les récurrences dans $\mathbb{F}_{2^w}$ puissent être utilisées pour construire un ensemble de points utile pour l'intégration QMC.

Pour le GCL polynômial, on modifie la récurrence pour obtenir

$$q_n(z) = z^s q_{n-1}(z) \bmod P(z)$$

où s est un entier positif. Le polynôme caractéristique de cette récurrence n'est pas nécessairement $P(z)$. En fait, il s'agit de $\tilde{P}(z)$, le polynôme minimal de z^s dans $\mathbb{F}_{2^w}[z]/P(z)$. La sortie est donnée par

$$\begin{aligned} \mathbf{p}_n &= (\mathbf{q}_{n,r}, \ldots, \mathbf{q}_{n,1}, \mathbf{q}_{n+1,r}, \ldots, \mathbf{q}_{n+1,1}, \ldots) \\ \mathbf{y}_n &= \mathrm{trunc}_L(\mathbf{p}_n). \end{aligned}$$

Dans le cas des ensembles de points construits à partir d'une récurrence (comme c'est le cas ici), le fait que la récurrence soit de période maximale ou non n'a pas d'importance si ceux-ci sont utilisés pour l'intégration quasi-Monte Carlo. On est seulement intéressé à ce que l'ensemble de points résultant (dans notre cas $\Psi_t(\mathbf{X}, B, L)$) soit le plus uniforme possible.

Pour le LFSR, on utilise la récurrence (5.1), mais la n-ième sortie est donnée par

$$\begin{aligned} \mathbf{p}_n &= (\mathbf{v}_{ns}, \mathbf{v}_{ns+1}, \ldots) \\ \mathbf{y}_n &= \mathrm{trunc}_L(\mathbf{p}_n) \end{aligned}$$

où s est un entier positif. Soit $P(z)$, le polynôme caractéristique de la récurrence (5.1). Le polynôme minimal de la séquence $\{m_{ns}\}_{n\geq 0}$ sur $\mathbb{F}_{2^w}$ est $\tilde{P}(z)$, le polynôme minimal de z^s dans $\mathbb{F}_{2^w}[z]/P(z)$.

Les deux théorèmes et le corollaire qui suivent justifient ces deux nouvelles récurrences pour l'intégration quasi-Monte Carlo. Une conséquence de ces théorèmes est que si $s < r$, alors, pour certaines valeurs de r et w, il est impossible pour le générateur résultant d'être ME. Par exemple, si $r = 4$, $w = 4$, $L = 16$ et $s = 1$, les résultats suivants indiquent que le générateur résultant est tel que $t_w = t_w^* = r$, mais aussi que $t_{2w} = 1 \leq t_{2w}^*$. Dans ce cas, il est impossible de construire un GCL polynômial ou un LFSR, avec les sorties définies ci-haut, qui soit ME puisque $\Delta_8 = t_8^* - t_8 = 2 - 1 = 1 \neq 0$. Il faut donc que s soit plus grand que r pour obtenir de bons générateurs.

Théorème 6.1. *Prenons un LFSR qui produit à la i-ième sortie le vecteur* $\mathbf{y}_i = (\mathbf{v}_{is}, \mathbf{v}_{is+1}, \ldots, \mathbf{v}_{is+r-1})$ *où* $s < r$. *Ce générateur est tel que* $t_{sw} = t_{sw}^* = \lfloor r/s \rfloor$ *et* $t_{(s+1)w} = 1$.

Démonstration. Soit $\mathbf{e}_j$, le j-ième vecteur unitaire de $\mathbb{F}_2^{rw}$. Soit

$$\mathbf{m}_{n,p} = (\mathbf{v}_{ns}^\mathsf{T}, \mathbf{v}_{ns+1}^\mathsf{T}, \ldots, \mathbf{v}_{ns+p-1}^\mathsf{T}) \in \mathbb{F}_2^{pw}$$

et $\mathbf{m}_{n,p,j}$, le vecteur $\mathbf{m}_{n,p}$ résultant lorsque la récurrence est initialisée avec $(\mathbf{v}_0, \ldots, \mathbf{v}_{r-1}) = \mathbf{e}_j \in \mathbb{F}_2^{rw}$ pour $j = 1, \ldots, rw$. Quand $n < t$, une propriété des vecteurs $\mathbf{m}_{n,p,j}$ est que

$$\mathbf{m}_{n,p,j} = \begin{cases} \mathbf{e}_{j-nsw} & \text{si } 1 \leq j - nsw \leq pw \\ 0 & \text{sinon.} \end{cases} \tag{6.1}$$

Ceci est dû au fait que, en aucun cas, les valeurs de $\mathbf{v}_i$, pour $i \geq r$, ne sont impliquées dans l'expression de $\mathbf{m}_{n,p,j}$.

Quand $t = \lfloor r/s \rfloor$ et $\ell = sw$, pour déterminer l'équidistribution, il faut déterminer le rang de la matrice H définie par la relation

$$H(\mathbf{v}_0, \ldots, \mathbf{v}_{r-1})^\mathsf{T} = (\mathbf{m}_{0,s}, \mathbf{m}_{1,s}, \ldots, \mathbf{m}_{t-1,s})^\mathsf{T}. \tag{6.2}$$

On a que

$$H = \begin{pmatrix} \mathbf{m}_{0,s,1} & \mathbf{m}_{1,s,1} & \dots & \mathbf{m}_{t-1,s,1} \\ \mathbf{m}_{0,s,2} & \mathbf{m}_{1,s,2} & \dots & \mathbf{m}_{t-1,s,2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{m}_{0,s,rw} & \mathbf{m}_{1,s,rw} & \dots & \mathbf{m}_{t-1,s,rw} \end{pmatrix}.$$

Par (6.1), on observe que H est une matrice $rw \times tsw$ semblable à la matrice identité : sur la j-ième ligne, on retrouve le j-ième vecteur unitaire pour $j = 1, \dots, tsw$ et les autres lignes sont nulles. Évidemment, cette matrice est de plein rang tsw, donc le générateur est tel que $t_{sw} = t^*_{sw} = \lfloor r/s \rfloor$.

Considérons maintenant la résolution $\ell = (s+1)w$. Si $\lfloor r/(s+1) \rfloor = 1$, alors, trivialement, on a que $t_{(s+1)w} = t^*_{(s+1)w} = 1$. Sinon, vérifions l'équidistribution en $t = 2$ dimensions. Dans ce cas, la matrice H est

$$\begin{pmatrix} \mathbf{m}_{0,s+1,1} & \mathbf{m}_{1,s+1,1} \\ \mathbf{m}_{0,s+1,2} & \mathbf{m}_{1,s+1,2} \\ \vdots & \\ \mathbf{m}_{0,s+1,rw} & \mathbf{m}_{1,s+1,rw} \end{pmatrix}$$

Une condition suffisante pour que H ne soit pas de plein rang est qu'elle ait au moins $m = rw - 2(s+1)w + 1$ lignes nulles. Pour que la ligne $[\mathbf{m}_{0,s+1,j} \;\; \mathbf{m}_{1,s+1,j}]$ soit nulle, par l'équation (6.1), il faut que j ne remplisse aucune de ces deux conditions :

1. $1 \leq j \leq (s+1)w$
2. $1 \leq j - sw \leq (s+1)w$.

Si on combine ces conditions, on obtient que pour $j > (2s+1)w$, les lignes correspondantes sont nulles. La matrice H a rw lignes, donc elle comprend $rw - 2sw - w$ lignes nulles ce qui est supérieur à m. On peut donc dire que H n'est pas de plein rang et que $t_{(s+1)w} \neq 2$. Par contre, en une dimension, le générateur est trivialement équidistribué pour la résolution $(s+1)w$. On a donc $t_{(s+1)w} = 1$. □

Théorème 6.2. *Un GCL polynômial dont le multiplicateur est z^s et la i-ième sortie est donnée par $\mathbf{y}_i = (\mathbf{q}_{i,r}, \ldots, \mathbf{q}_{i,1})$ a la même q-valeur qu'un LFSR qui produit à la i-ième sortie le vecteur $\mathbf{y}_i = (\mathbf{v}_{is}, \mathbf{v}_{is+1}, \ldots, \mathbf{v}_{is+r-1})$ pour $t = 1, \ldots, rw$.*

Démonstration. Soit la matrice inversible de dimension $r \times r$

$$D_r = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ b_1 & 1 & 0 & \ldots & 0 & 0 \\ b_2 & b_1 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ b_{r-2} & b_{r-2} & \ldots & b_1 & 1 & 0 \\ b_{r-1} & b_{r-2} & b_{r-3} & \ldots & b_1 & 1 \end{pmatrix}$$

avec ses éléments dans $\mathbb{F}_{2^w}$. Il est bien connu [32], et facilement vérifiable, que

$$(q_{i,r}, \ldots, q_{i,1}) = D_r(m_{is}, \ldots, m_{is+r-1}).$$

Soit $D_r(\mathbb{F}_2)$, la matrice obtenue en remplaçant chacun des éléments $d \in \mathbb{F}_{2^w}$ de D_r par la matrice A_d qui effectue la multiplication par d dans la base choisie. Cette dernière a ses éléments dans $\mathbb{F}_2$ et est de dimension $rw \times rw$. On a alors

$$(\mathbf{q}_{i,r}, \ldots, \mathbf{q}_{i,1}) = D_r(\mathbb{F}_2)(\mathbf{v}_{is}, \ldots, \mathbf{v}_{is+r-1}). \tag{6.3}$$

Ceci implique que la sortie du GCL polynômial $\mathbf{y}_i = (\mathbf{q}_{i,r}, \ldots, \mathbf{q}_{i,1})$ peut être obtenue en appliquant le tempering

$$\mathbf{y}_i = D_r(\mathbb{F}_2)(\mathbf{v}_{is}, \ldots, \mathbf{v}_{is+r-1})$$

au LFSR défini au théorème 6.1. La matrice $D_r(\mathbb{F}_2)$ est une matrice triangulaire avec des uns dans la diagonale principale et des éléments non nuls sous la diagonale principale. Il est bien connu [63] que ce type de tempering ne change pas la q-valeur (et l'équidistribution) par rapport au même générateur sans tempering. □

Corollaire 6.1. *Si le multiplicateur du GCL polynômial est z^s, $s < r$ et la sortie est donnée par $\mathbf{y}_n = (\mathbf{q}_{n,r}, \ldots, \mathbf{q}_{n,1})$, alors on a $t_{sw} = t^*_{sw} = \lfloor r/s \rfloor$ et $t_{(s+1)w} = 1$.*

Supposons que $P(z)$ soit primitif. À la section 5.6, on montre que pour les LFSR dans $\mathbb{F}_{2^w}$ et les GCL polynômiaux, presque toutes les projections en deux dimensions sont $(2,w)$-équidistribuées. Pour les ensembles de points qu'on a définis dans ce chapitre, on a également une propriété similaire si $P(z)$ est primitif. Les deux prochains corollaires donnent ce résultat.

Corollaire 6.2. *Soit*

$$\begin{aligned}
\mathbf{m}_j &= \sum_{i=1}^{r} A_{b_i}\mathbf{m}_{j-i} \\
\mathbf{y}_n &= \mathbf{m}_{ns} \\
u_n &= \sum_{i=1}^{w} y_n^{(i-1)}2^{-i} \\
\Psi_t &= \{(u_0,u_1,\ldots,u_{t-1}) : (\mathbf{m}_{r-1},\ldots,\mathbf{m}_0) \in \mathbb{F}_2^{rw}\}
\end{aligned}$$

Supposons que $P(z)$, le polynôme minimal sur $\mathbb{F}_{2^w}$ de la séquence $\{m_n\}_{n\geq 0}$, soit primitif. Soit $b = (2^{wr}-1)/(2^w-1)$ et $h = ppcm(b,s)/s$. La projection $\Psi_2(\{i,i+j\})$ est $(2,w)$-équidistribuée si et seulement si j n'est pas un multiple de h.

Corollaire 6.3. *Soit $q_n(z) = q_{n,1}z^{r-1} + \cdots + q_{n,r} = z^s q_{n-1}(z) \bmod P(z)$ où $P(z) \in \mathbb{F}_{2^w}[z]$ est primitif sur $\mathbb{F}_{2^w}$. Soit $\mathbf{y}_n = \mathbf{q}_{n,r}$ où $\mathbf{q}_{n,r}$ représente $q_{n,r}$ dans une base choisie et $u_n = \sum_{i=1}^{w} y_n^{(i-1)}2^{-i}$. Soit l'ensemble de points*

$$\Psi_t = \{(u_0,u_1,\ldots,u_{t-1}) : q_0(z) \in \mathbb{F}_{2^w}[z]/P(z)\}.$$

Soit $b = (2^{wr}-1)/(2^w-1)$ et $h = ppcm(b,s)/s$. La projection $\Psi_2(\{i,i+j\})$ est $(2,w)$-équidistribuée si et seulement si j n'est pas un multiple de h.

Démonstration. Du théorème 5.5, pour les GCL polynômiaux et les LFSR, on sait que l'ensemble de points $\{(u_0,u_j) : \mathbf{x}_0 \in \mathbb{F}_2^{rw}\}$ n'est pas $(2,w)$-équidistribué si et seulement si j est un multiple de b. On peut voir, pour les deux corollaires, que l'ensemble de points $\Psi_2(\{i,i+j\})$ n'est pas $(2,w)$-équidistribué si et seulement si js est un multiple de b. Pour démontrer les deux corollaires, on doit trouver la plus petite

valeur de c telle que $c = js = ab$ pour un entier a. La solution est $c = \text{ppcm}(b, s)$, ce qui implique que $j = \text{ppcm}(b, s)/s$. On a donc que la projection $\Psi_2(\{i, i + j\})$ n'est pas $(2, w)$-équidistribuée si et seulement si j est un multiple de $h = \text{ppcm}(b, s)/s$. □

6.3 Résultats de recherche de paramètres

Nous avons cherché de bons petits générateurs du type LFSR qui utilisent la sortie définie à la section 6.2 à l'aide de différents critères. Étant donné qu'on n'utilisera pas ces générateurs comme générateurs de nombres aléatoires mais plutôt pour des applications quasi-Monte Carlo, il n'est pas utile de parler de la période des générateurs, mais plutôt de la cardinalité des ensembles de points qui peuvent être produits par les générateurs. Ainsi, la cardinalité des ensembles de points considérés varie de 2^{10} (quand $rw = 10$) à 2^{18} (quand $rw = 18$).

Soit

$$S(s, t_1, \ldots, t_s) = \left(\bigcup_{i=1}^{s} \{\{j_1, \ldots, j_i\}, 1 = j_1 \leq \ldots, j_i \leq t_i\} \right) \bigcup \{\{1, \ldots, j\}, 1 \leq j \leq t_1\},$$

$S_1 = S(5, k, 24, 16, 8, 8)$ où $k = rw$ et $S_2 = S(3, 3, 24, 16)$. En tout, six critères de recherche (voir section 4.7) ont été considérés : $\Delta(S_1, \Lambda)$, $\Theta(S_1, \Lambda)$, $\Delta(S_1, q - \text{valeur})$, $\Theta(S_1, q - \text{valeur})$, $\Delta(S_2, \Gamma_t)$ et $\Theta(S_2, \Gamma_t)$. Les deux premiers critères sont basés sur l'équidistribution, les deux suivants sur la q-valeur et les deux derniers sur la distance minimale. Pour les quatre premiers critères, on considère les projections jusqu'en k dimensions successives, les projections en 2 dimensions dont l'espacement entre les indices des coordonnées n'excèdent pas 24, les projections en 3 dimensions dont l'espacement entre les indices des coordonnées n'excèdent pas 16, et les projections en 4 et 5 dimensions dont l'espacement entre les indices des coordonnées n'excèdent pas 8. Pour les critères basés sur la distance minimale, on se restreint aux projections successives en 2 et 3 dimensions, les projections en 2 dimensions dont l'espacement

entre les indices des coordonnées n'excèdent pas 24 et les projections en 3 dimensions dont l'espacement entre les indices des coordonnées n'excèdent pas 16.

Ces paramètres de recherche sont quelque peu arbitraires. En effet, rien ne dit que ces projections sont celles qui sont importantes pour toutes les applications. Néanmoins, nous les choisissons en espérant qu'elles sont importantes pour plusieurs applications. Pour les critères basés sur la distance minimale, nous nous concentrons sur les projections en 2 et 3 dimensions pour deux raisons d'ordre pratique : le temps d'exécution de l'algorithme utilisé pour calculer Γ_t est exponentiel en la dimension et le temps nécessaire pour mesurer l'uniformité avec ce critère est déjà assez long avec ces projections et qu'en rajouter d'autres ferait en sorte qu'on ne puisse pas échantillonner suffisamment d'ensembles de points dans la recherche.

Aux tableaux 6.1–6.6, on donne les résultats de ces recherches. La manière de représenter $Q(z)$ et les éléments de $\mathbb{F}_{2^w}$ est la même qu'au chapitre 5. Pour les critères du type $\Delta(\cdot,\cdot)$, pour un couple (r,w) donné, il arrive fréquemment que deux ensembles de points performent avec la même valeur au critère. Pour briser l'égalité, nous avons utilisé le critère $\Theta(\cdot,\cdot)$ avec le même critère sous-jacent. Malgré ces efforts pour discriminer les ensembles de points, il arrive fréquemment que plusieurs ensembles de points ne soient pas discriminés avec les deux critères. Dans ces cas, on donne jusqu'à trois ensembles de paramètres pour un couple (r,w) (malgré que dans certains cas, on en a trouvé beaucoup plus). Dans les tableaux 6.1–6.6, on affiche les résultats obtenus aux deux critères pour les ensembles de points pour lesquels on désire une petite valeur de $\Delta(\cdot,\cdot)$. On remarque que, en comparant deux à deux les tableaux 6.1–6.2, 6.3–6.4 et 6.5–6.6, si on désire une petite valeur de $\Theta(\cdot,\cdot)$, il est parfois payant de sacrifier la qualité de certaines projections pour arriver à notre but.

Tableau 6.1 – Résultats de la recherche d'ensembles de $n = 2^{rw}$ points qui minimisent le critère $\Delta(S_1, \Lambda)$.

r	w	$Q(z)$	s	$\mathbf{b}_1$	$\mathbf{b}_2$	$\mathbf{b}_3$	$\mathbf{b}_4$	$\mathbf{b}_5$	$\mathbf{b}_6$	$\mathbf{b}_7$	$\mathbf{b}_8$	$\mathbf{b}_9$	$\Delta(S_1, \Lambda)$	$\Theta(S_1, \Lambda)$
2	5	1d	217	1	9	–	–	–	–	–	–	–	1	41
2	5	14	248	3	1	–	–	–	–	–	–	–	1	41
2	5	17	899	1d	9	–	–	–	–	–	–	–	1	41
5	2	3	306	2	0	1	0	3	–	–	–	–	1	45
5	2	3	306	3	0	3	0	3	–	–	–	–	1	45
5	2	3	480	2	0	1	3	1	–	–	–	–	1	45
2	6	30	121	1f	24	–	–	–	–	–	–	–	2	83
2	6	21	219	22	a	–	–	–	–	–	–	–	2	83
2	6	33	497	2d	1	–	–	–	–	–	–	–	2	83
3	4	9	94	e	4	4	–	–	–	–	–	–	2	40
3	4	f	163	3	0	9	–	–	–	–	–	–	2	40
3	4	f	491	0	e	d	–	–	–	–	–	–	2	40
4	3	5	358	2	5	6	7	–	–	–	–	–	1	67
4	3	6	436	6	0	0	5	–	–	–	–	–	1	67
4	3	6	436	7	0	0	6	–	–	–	–	–	1	67
6	2	3	99	0	1	1	2	0	1	–	–	–	2	45
6	2	3	99	0	2	1	1	0	1	–	–	–	2	45
2	7	77	152	73	52	–	–	–	–	–	–	–	1	12
2	7	6a	689	79	62	–	–	–	–	–	–	–	1	12
2	7	7e	726	4d	2a	–	–	–	–	–	–	–	1	12
7	2	3	548	2	0	0	2	1	0	1	–	–	1	12
7	2	3	548	2	0	0	2	3	0	3	–	–	1	12
3	5	1d	360	b	0	3	–	–	–	–	–	–	1	20
3	5	1d	697	10	0	1b	–	–	–	–	–	–	1	20
3	5	12	734	1e	19	d	–	–	–	–	–	–	1	20
5	3	5	19	5	3	4	1	2	–	–	–	–	1	78
2	8	b8	242	ab	2a	–	–	–	–	–	–	–	1	53
2	8	8e	539	fb	61	–	–	–	–	–	–	–	1	53
2	8	d8	736	e7	1d	–	–	–	–	–	–	–	1	53
4	4	9	842	3	e	0	e	–	–	–	–	–	1	32
8	2	3	42	0	0	1	0	1	0	0	1	–	1	44
8	2	3	42	0	0	3	0	3	0	0	3	–	1	44
8	2	3	836	0	1	0	1	0	2	2	1	–	1	44
2	9	189	13	150	2b	–	–	–	–	–	–	–	1	65
2	9	110	272	140	1ed	–	–	–	–	–	–	–	1	65
2	9	119	454	123	3e	–	–	–	–	–	–	–	1	65
3	6	24	497	27	3a	3b	–	–	–	–	–	–	1	8
6	3	5	365	3	0	5	0	3	7	–	–	–	2	10
9	2	3	897	0	3	1	3	3	0	3	1	3	2	33

Tableau 6.2 – Résultats de la recherche d'ensembles de $n = 2^{rw}$ points qui minimisent le critère $\Theta(S_1, \Lambda)$.

r	w	$Q(z)$	s	$\mathbf{b}_1$	$\mathbf{b}_2$	$\mathbf{b}_3$	$\mathbf{b}_4$	$\mathbf{b}_5$	$\mathbf{b}_6$	$\mathbf{b}_7$	$\mathbf{b}_8$	$\mathbf{b}_9$	$\Theta(S_1, \Lambda)$
2	5	17	124	13	12	–	–	–	–	–	–	–	41
2	5	14	620	a	8	–	–	–	–	–	–	–	41
2	5	12	713	9	2	–	–	–	–	–	–	–	41
5	2	3	306	3	0	3	0	3	–	–	–	–	45
5	2	3	631	3	3	0	0	1	–	–	–	–	45
5	2	3	883	3	0	0	0	3	–	–	–	–	45
2	6	21	97	25	25	–	–	–	–	–	–	–	79
2	6	33	298	12	18	–	–	–	–	–	–	–	79
2	6	2b	353	28	3b	–	–	–	–	–	–	–	79
3	4	9	218	7	0	b	–	–	–	–	–	–	39
3	4	f	328	c	0	d	–	–	–	–	–	–	39
3	4	9	437	0	7	3	–	–	–	–	–	–	39
4	3	5	205	6	0	2	7	–	–	–	–	–	61
4	3	6	380	3	0	7	3	–	–	–	–	–	61
4	3	6	485	6	5	0	3	–	–	–	–	–	61
6	2	3	702	0	1	2	0	1	1	–	–	–	38
6	2	3	702	0	2	2	0	2	3	–	–	–	38
2	7	41	562	24	3f	–	–	–	–	–	–	–	12
2	7	48	653	70	41	–	–	–	–	–	–	–	12
2	7	60	822	67	74	–	–	–	–	–	–	–	12
7	2	3	548	2	0	0	2	1	0	1	–	–	12
7	2	3	548	2	0	0	2	3	0	3	–	–	12
3	5	1e	360	8	0	3	–	–	–	–	–	–	20
3	5	1d	378	0	d	14	–	–	–	–	–	–	20
3	5	1e	877	0	1b	d	–	–	–	–	–	–	20
5	3	5	291	0	1	6	6	1	–	–	–	–	64
5	3	6	291	0	7	1	1	7	–	–	–	–	64
5	3	5	291	0	7	5	5	7	–	–	–	–	64
2	8	8d	552	ef	19	–	–	–	–	–	–	–	51
2	8	fa	641	2b	8	–	–	–	–	–	–	–	51
2	8	8e	846	95	a1	–	–	–	–	–	–	–	51
4	4	c	286	4	9	e	4	–	–	–	–	–	31
4	4	f	533	a	b	8	a	–	–	–	–	–	31
8	2	3	56	2	0	2	1	2	2	2	3	–	43
8	2	3	309	3	0	0	3	1	2	0	3	–	43
2	9	19d	190	1a4	153	–	–	–	–	–	–	–	57
2	9	116	247	aa	f8	–	–	–	–	–	–	–	57
2	9	13b	589	12a	60	–	–	–	–	–	–	–	57
3	6	24	79	a	0	2d	–	–	–	–	–	–	9
3	6	30	519	15	20	11	–	–	–	–	–	–	9
3	6	30	629	0	30	29	–	–	–	–	–	–	9
6	3	5	365	2	0	2	0	3	6	–	–	–	9
9	2	3	206	2	0	1	3	1	3	0	3	1	32
9	2	3	206	2	0	3	1	3	1	0	1	3	32
9	2	3	583	1	1	2	2	0	2	1	3	3	32

Tableau 6.3 – Résultats de la recherche d'ensembles de $n = 2^{rw}$ points qui minimisent le critère $\Delta(S_1, q - \text{valeur})$.

r	w	$Q(z)$	s	$\mathbf{b}_1$	$\mathbf{b}_2$	$\mathbf{b}_3$	$\mathbf{b}_4$	$\mathbf{b}_5$	$\mathbf{b}_6$	$\mathbf{b}_7$	$\mathbf{b}_8$	$\mathbf{b}_9$	$\Delta(S_1, q)$	$\Theta(S_1, q)$
2	5	14	62	1b	3	–	–	–	–	–	–	–	5	716
2	5	1e	217	1a	b	–	–	–	–	–	–	–	5	716
2	5	1e	899	1a	2	–	–	–	–	–	–	–	5	716
5	2	3	476	1	3	0	2	1	–	–	–	–	5	677
5	2	3	476	3	1	0	2	3	–	–	–	–	5	677
5	2	3	482	1	1	1	1	1	–	–	–	–	5	677
2	6	2b	37	19	35	–	–	–	–	–	–	–	6	849
3	4	9	837	5	4	4	–	–	–	–	–	–	6	818
4	3	6	139	0	6	3	7	–	–	–	–	–	6	778
4	3	6	706	0	6	3	7	–	–	–	–	–	6	778
6	2	3	116	2	0	1	2	3	3	–	–	–	6	811
2	7	7b	585	24	32	–	–	–	–	–	–	–	8	935
7	2	3	468	2	0	1	1	0	1	3	–	–	7	934
3	5	17	208	f	0	9	–	–	–	–	–	–	8	992
5	3	6	677	0	6	0	2	2	–	–	–	–	8	969
2	8	fa	809	e1	9c	–	–	–	–	–	–	–	9	1056
4	4	9	883	0	4	e	b	–	–	–	–	–	9	989
8	2	3	416	3	0	2	2	3	3	3	1	–	9	1006
2	9	17a	260	f4	f	–	–	–	–	–	–	–	10	1238
3	6	39	97	1f	34	10	–	–	–	–	–	–	10	1176
6	3	5	733	1	7	6	3	0	6	–	–	–	10	1160
9	2	3	116	0	2	2	2	3	0	0	3	3	10	1159

Tableau 6.4 – Résultats de la recherche d'ensembles de $n = 2^{rw}$ points qui minimisent le critère $\Theta(S_1, q - \text{valeur})$.

r	w	$Q(z)$	s	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	$\Theta(S_1, q)$
2	5	14	52	17	2	–	–	–	–	–	–	–	688
2	5	1b	427	1d	1f	–	–	–	–	–	–	–	688
2	5	1e	485	8	8	–	–	–	–	–	–	–	688
5	2	3	431	3	0	1	1	3	–	–	–	–	653
5	2	3	592	1	0	3	3	1	–	–	–	–	653
5	2	3	592	3	0	1	1	3	–	–	–	–	653
2	6	33	34	26	13	–	–	–	–	–	–	–	794
3	4	c	292	b	f	7	–	–	–	–	–	–	754
4	3	6	396	0	6	5	1	–	–	–	–	–	751
4	3	5	774	0	1	7	3	–	–	–	–	–	751
6	2	3	350	0	0	0	3	1	3	–	–	–	771
6	2	3	805	0	3	0	0	3	3	–	–	–	771
2	7	6a	499	45	6d	–	–	–	–	–	–	–	915
7	2	3	236	3	2	0	0	0	3	1	–	–	889
3	5	1e	448	a	14	1f	–	–	–	–	–	–	934
5	3	6	350	2	6	0	7	6	–	–	–	–	922
5	3	5	350	5	1	0	2	1	–	–	–	–	922
2	8	a6	300	81	2c	–	–	–	–	–	–	–	1045
2	8	8e	781	99	57	–	–	–	–	–	–	–	1045
4	4	9	816	0	3	d	3	–	–	–	–	–	959
8	2	3	859	0	0	2	3	0	1	1	1	–	997
2	9	198	652	144	2f	–	–	–	–	–	–	–	1155
3	6	21	731	0	29	37	–	–	–	–	–	–	1118
6	3	6	551	7	4	4	7	0	1	–	–	–	1089
9	2	3	484	2	0	0	1	0	1	0	2	1	1104

Tableau 6.5 – Résultats de la recherche d'ensembles de $n = 2^{rw}$ points qui minimisent le critère $\Delta(S_2, \Gamma_t)$.

r	w	$Q(z)$	s	$\mathbf{b}_1$	$\mathbf{b}_2$	$\mathbf{b}_3$	$\mathbf{b}_4$	$\mathbf{b}_5$	$\mathbf{b}_6$	$\mathbf{b}_7$	$\mathbf{b}_8$	$\mathbf{b}_9$	$\Delta(S_2, \Gamma_t)$	$\Theta(S_2, \Gamma_t)$
2	5	17	692	7	d	–	–	–	–	–	–	–	3	212
5	2	3	50	3	3	0	0	1	–	–	–	–	3	216
2	6	39	347	3c	24	–	–	–	–	–	–	–	4	311
3	4	c	202	2	1	4	–	–	–	–	–	–	4	316
3	4	9	665	0	2	b	–	–	–	–	–	–	4	316
4	3	5	656	0	2	3	7	–	–	–	–	–	4	307
6	2	3	434	2	3	1	0	3	3	–	–	–	4	314
6	2	3	748	3	0	0	3	0	3	–	–	–	4	314
6	2	3	856	0	0	3	1	0	1	–	–	–	4	314
2	7	44	292	76	50	–	–	–	–	–	–	–	4	308
2	7	77	393	29	45	–	–	–	–	–	–	–	4	308
7	2	3	199	1	0	3	0	1	1	1	–	–	4	303
3	5	1e	68	1e	14	4	–	–	–	–	–	–	4	348
3	5	17	399	0	1b	19	–	–	–	–	–	–	4	348
5	3	6	47	3	7	0	0	5	–	–	–	–	4	340
2	8	c5	261	52	42	–	–	–	–	–	–	–	4	303
2	8	b1	489	6	78	–	–	–	–	–	–	–	4	303
4	4	c	675	b	f	0	9	–	–	–	–	–	4	295
8	2	3	184	2	3	1	0	3	3	2	1	–	4	299
2	9	1ea	597	b2	91	–	–	–	–	–	–	–	4	388
3	6	36	20	0	21	f	–	–	–	–	–	–	5	381
6	3	5	259	3	4	1	6	7	2	–	–	–	5	389
9	2	3	279	0	3	0	1	0	0	0	2	1	5	390

Tableau 6.6 – Résultats de la recherche d'ensembles de $n = 2^{rw}$ points qui minimisent le critère $\Theta(S_2, \Gamma_t)$.

r	w	$Q(z)$	s	$\mathbf{b}_1$	$\mathbf{b}_2$	$\mathbf{b}_3$	$\mathbf{b}_4$	$\mathbf{b}_5$	$\mathbf{b}_6$	$\mathbf{b}_7$	$\mathbf{b}_8$	$\mathbf{b}_9$	$\Theta(S_2, \Gamma_t)$
2	5	1d	727	6	19	–	–	–	–	–	–	–	205
5	2	3	385	0	0	0	2	1	–	–	–	–	211
5	2	3	871	3	3	0	0	1	–	–	–	–	211
2	6	36	618	12	29	–	–	–	–	–	–	–	309
3	4	f	375	0	3	a	–	–	–	–	–	–	308
4	3	5	122	1	7	0	5	–	–	–	–	–	305
6	2	3	795	3	0	2	1	1	1	–	–	–	309
2	7	5f	101	30	1f	–	–	–	–	–	–	–	302
7	2	3	361	0	0	1	1	0	0	1	–	–	304
7	2	3	729	2	3	0	1	0	0	1	–	–	304
3	5	17	480	c	0	f	–	–	–	–	–	–	330
5	3	6	362	5	1	0	3	2	–	–	–	–	342
5	3	6	711	4	5	1	0	2	–	–	–	–	342
2	8	d8	702	88	da	–	–	–	–	–	–	–	294
4	4	9	165	d	c	3	b	–	–	–	–	–	295
4	4	c	741	c	d	a	9	–	–	–	–	–	295
8	2	3	356	0	0	0	0	3	3	2	1	–	289
2	9	1c7	474	1f3	6b	–	–	–	–	–	–	–	387
3	6	36	115	32	0	17	–	–	–	–	–	–	392
6	3	5	831	5	5	0	6	6	1	–	–	–	373
9	2	3	166	3	0	0	0	2	1	0	2	3	389
9	2	3	560	0	0	1	1	3	3	3	2	1	389

6.4 Fonctions test

Dans cette section, on décrit des fonctions f que l'on tentera d'intégrer à l'aide de différents ensemble de points. La liste des fonctions est donnée au tableau 6.7. On assigne un numéro à chacune des fonctions. Les fonctions f_2, f_6 et f_9 sont tirées de [25], tandis que les fonctions f_{11} à f_{14} sont proposées dans [91]. La fonction f_{15} est proposée pour la première fois dans cette thèse. Les fonctions f_2, f_6, f_9 et f_{11} à f_{15} sont telles que la valeur de l'intégrale sur l'hypercube unitaire $[0,1)^t$ est nulle. Les fonctions f_2, f_6, f_9 et f_{15} sont telles que $\sigma^2 = 1$ où $\sigma^2 = I(f^2) - I(f)^2$. Les fonctions f_{16}, f_{18} et f_{19} sont reliées au problème de l'évaluation d'une option asiatique. Ce problème est expliqué dans les prochaines sections.

Tableau 6.7 – Fonctions analytiques à intégrer

$f_2(\mathbf{x})$	=	$\sqrt{45/4t}(\sum_{j=0}^{t-1} x_j^2 - t/3)$
$f_6(\mathbf{x})$	=	$\prod_{j=0}^{t-1}(-2.4\sqrt{7}(x_j - 1/2) + 8\sqrt{7}(x_j - 1/2)^3)$
$f_9(\mathbf{x})$	=	$\sqrt{\frac{2}{t(t-1)}} \sum_{j=0}^{t-1} \sum_{i=0}^{j-1} g(x_i)g(x_j)$ où
		$g(x) = 27.20917094x^3 - 36.19250850x^2 + 8.983337562x + 0.7702079855.$
$f_{11}(\mathbf{x})$	=	$\prod_{j=0}^{t-1} \frac{\lvert 4x_j - 2\rvert + a_j}{1+a_j}$ où $a_j = 0.01$
$f_{12}(\mathbf{x})$	=	$\prod_{j=0}^{t-1} \frac{\lvert 4x_j - 2\rvert + a_j}{1+a_j}$ où $a_j = 1$
$f_{13}(\mathbf{x})$	=	$\prod_{j=0}^{t-1} \frac{\lvert 4x_j - 2\rvert + a_j}{1+a_j}$ où $a_j = j$
$f_{14}(\mathbf{x})$	=	$\prod_{j=0}^{t-1} \frac{\lvert 4x_j - 2\rvert + a_j}{1+a_j}$ où $a_j = j^2$
$f_{15}(\mathbf{x})$	=	$\frac{3^m}{n(4^m - 3^m)} \sum_{i=0}^{n-1} \left(1 - \prod_{j=0}^{m-1} 2x_{im+j}\right)$

6.4.1 Évaluation d'une option asiatique

Dans cette section, nous décrivons le problème qui consiste à évaluer la valeur d'une option. Une *option* est un contrat financier dont la valeur dépend du prix d'une *denrée sous-jacente*. Le type de contrat qu'on considère est une *option asiatique* sur une denrée dont le prix varie selon un mouvement Brownien géométrique. Soit $S(t)$, le prix de la denrée au temps t. La valeur de l'option au temps t est $C(t)$ et sa valeur à l'échéance du contrat est donnée par

$$C(T) = \max\left(0, \bar{S} - K\right)$$

où $\bar{S} = \frac{1}{s}\sum_{j=1}^{s} S(t_j)$ et les temps t_j, $1 \leq j \leq s$ et le prix K sont définis dans le contrat.

Pour modéliser le prix de la denrée sous-jacente, nous utilisons le modèle de Black et Scholes [1]. Sous ce modèle, on a que

$$dS(t) = \mu S(t)dt + \sigma S(t)dB(t)$$

où μ est le rendement moyen de la denrée, σ est la volatilité de la denrée et $B(\cdot)$ est un mouvement Brownien standard. Le modèle suppose aussi l'existence d'une mesure de risque nul sous laquelle

$$dS(t) = rS(t)dt + \sigma S(t)d\tilde{B}(t)$$

où r est le taux d'intérêt sans risque et $\tilde{B}(\cdot)$ est un mouvement Brownien standard sous la mesure de risque nul. La valeur au temps $t = 0$ de l'option asiatique est donnée par

$$C(0) = \tilde{\mathrm{E}}[e^{-rT}C(T)],$$

où $\tilde{\mathrm{E}}$ est l'espérance sous la mesure de risque nulle. Le calcul cette espérance ne peut se faire analytiquement et on utilise souvent les méthodes Monte Carlo/quasi-Monte Carlo pour y arriver. Dans ce cas, il faut simuler l'évolution du prix de la denrée

sous-jacente et être capable de le déterminer aux temps t_j, $1 \leq j \leq s$. Pour ce faire on utilise la formule

$$S(t_j) = S(0)\exp\left((r - 0.5\sigma^2)t_j + \sigma\tilde{B}(t_j)\right)$$

pour $j = 1, \ldots, s$ où $\tilde{B}(t)$ est une variable aléatoire normale de moyenne 0 et de variance t. Pour évaluer l'espérance $C(0)$, on simule n évolutions du prix de la denrée sous-jacente, pour chacune d'elle, on calcule la valeur $C(T)$. On génère n variables aléatoires $\mathbf{x}_1, \ldots, \mathbf{x}_n$, uniformes sur $[0,1)^s$ où la j-ième coordonnée sert à générer la variable aléatoire $B(t_j)$ par inversion. Ainsi, on définit

$$f_{16}(\mathbf{x}) = e^{-rT}C(T)$$

pour l'évolution du prix de la denrée sous-jacente décrite par le vecteur $\mathbf{x}$. Estimer $C(0)$ revient à approximer

$$\int_{[0,1)^s} f_{16}(\mathbf{x})d\mathbf{x}.$$

Pour le besoin de cette thèse, nous ne considérons que les options qui sont telles que $T = 120/365$, $t_j = (T - s + j)/365$, $S(0) = 100$, $\sigma = 0.2$, $r = \ln(1.09)$, comme c'est le cas pour [42, 43, 51]. On estimera la valeur de l'option pour $K = 90, 100, 110$ et $s = 10, 30$. Pour simuler le mouvement Brownien, on utilise la technique standard et le pont Brownien.

6.4.2 Évaluation de la dérivée du prix d'une option asiatique

La prochaine fonction à évaluer est directement liée au problème du prix d'une option asiatique. On est intéressé à déterminer la sensibilité du prix par rapport à deux paramètres du modèle, soit σ et $S(0)$. Autrement dit, on veut évaluer

$$\mu_d = \frac{\partial C(0)}{\partial S(0)}$$

et

$$\mu_v = \frac{\partial C(0)}{\partial \sigma}.$$

Ces deux valeurs sont communément nommés *delta* et *vega* dans le monde de la finance. Dans [6], on montre que

$$\hat{\mu}_d = e^{-rT}\frac{\partial C(T)}{\partial S(0)} = e^{-rT}\mathbf{1}(\bar{S} \geq K)\frac{\bar{S}}{S(0)}$$

et

$$\hat{\mu}_v = e^{-rT}\frac{\partial C(T)}{\partial \sigma} = e^{-rT}\mathbf{1}(\bar{S} \geq K)\frac{1}{s\sigma}\sum_{i=1}^{s} S(t_i)\left[\ln\left(\frac{S(t_i)}{S(0)}\right) - (r+\sigma^2/2)t_i\right]$$

sont des estimateurs sans biais de μ_d et μ_v respectivement. Les fonctions $f_{18}(\mathbf{x})$ et $f_{19}(\mathbf{x})$ sont définies par la valeur des estimateurs $\hat{\mu}_d$ et $\hat{\mu}_v$ respectivement où $\mathbf{x}$ contient l'information nécessaire pour produire une évolution du prix de la denrée sous-jacente. Ainsi, on a que

$$\mu_d = E[\hat{\mu}_d] = \int_{[0,1)^s} f_{18}(\mathbf{x})d\mathbf{x}$$

et

$$\mu_v = E[\hat{\mu}_v] = \int_{[0,1)^s} f_{19}(\mathbf{x})d\mathbf{x}.$$

On tentera d'évaluer ces intégrales par les méthodes quasi-Monte Carlo. Dans cette thèse, on choisit les paramètres du modèle comme dans [6, 52]. Ceux-ci sont $r = 0.07$, $K = 100$, $T = 0.2$ années. On estimera les intégrales pour $s \in \{10, 30\}$ et $S(0) \in \{90, 100, 110\}$.

6.5 Méthodologie expérimentale

Afin de démontrer de manière efficace les forces et les faiblesses des ensembles de points basés sur des récurrences linéaires dans $\mathbb{F}_{2^w}$ que nous avons introduites, pour l'intégration quasi-Monte Carlo, nous avons estimé le facteur de réduction de variance ρ pour plusieurs ensembles de points et pour les fonctions définies dans les sections

précédentes. Les résultats de ces expériences sont illustrés à l'aide de graphiques aux figures 6.8 à 6.36. L'interprétation de ces graphiques est donnée un peu plus loin dans cette section.

Voici l'« algorithme » ou la méthodologie qui a permis de construire ces graphiques.

Algorithme 6.3. Méthodologie

- Les fonctions à intégrer sont :
  - f_2, f_6, f_9, f_{11}, f_{12}, f_{13} et f_{14} en 10 et 30 dimensions,
  - f_{15} en 100 dimensions avec $m = 2, 5, 10, 20, 50$,
  - l'évaluation de l'option asiatique en 10 et 30 dimensions avec $K = 90$, $K = 100$ et $K = 110$,
  - l'évaluation de vega pour l'option asiatique en 10 et 30 dimensions avec $S(0) = 90$, $S(0) = 100$ et $S(0) = 110$,
  - l'évaluation de delta pour l'option asiatique en 10 et 30 dimensions avec $S(0) = 90$, $S(0) = 100$ et $S(0) = 110$.
- Les couples (r, w) à considérer sont :
  - $(2,5), (5,2), (2,6), (3,4), (4,3), (6,2), (2,7), (7,2), (3,5), (5,3)$, $(2,8), (4,4), (8,2), (2,9), (3,6), (6,3), (9,2)$.
- Les critères d'uniformité sont :
  - $\Delta(S_1, \Lambda)$, $\Theta(S_1, \Lambda)$,
  - $\Delta(S_1, q - \text{valeur})$, $\Theta(S_1, q - \text{valeur})$,
  - $\Delta(S_2, \Gamma_t)$ et $\Theta(S_2, \Gamma_t)$.
- Pour chaque fonction f à intégrer, faire

  Estimer la valeur de σ^2_{MC} avec $\bar{\sigma}_{10^7}$.

  Pour chaque couple (r, w) à considérer, faire

  Pour chaque critère d'uniformité C à considérer, faire

  - Avec $m = 1000$, calculer $\hat{\sigma}^2_m$

    avec la translation digitale comme randomisation,

pour un ensemble de points, de paramètres (r, w),
qui démontre la meilleure valeur observée du critère C

- $\hat{\rho}[f, C, r, w] \leftarrow \bar{\sigma}_{10^7}/(2^{rw}\hat{\sigma}_m^2)$

Dans cet algorithme, on fait référence à « un ensemble de points, de paramètres (r, w), qui démontre la meilleure valeur observée du critère C ». Pour illustrer ce que nous voulons dire, prenons, par exemple, $C = \Delta(S_1, \Lambda)$ et $(r, w) = (2, 5)$. Dans ce cas, nous avons trouvé 2358 ensembles de points qui démontrent la meilleure valeur observée, soit $\Delta(S_1, \Lambda) = 41$. Parmi ces ensembles de points, nous en prenons un seul pour calculer $\hat{\rho}[f, C, r, w]$.

Dans la plupart de résultats exposés, les estimateurs de la réduction de variance sont précis à au moins 10%. Les estimateurs les plus mauvais sont ceux pour les fonctions f_6, f_{11}, et f_{19}. Pour f_6 et f_{19}, la plupart des estimateurs de réduction de variance ont une erreur relative de moins de 25%. Pour f_{11}, la plupart ont une erreur relative de moins de 75%.

Les figures 6.8 à 6.36 contiennent beaucoup d'information. Commençons par les figures 6.8 à 6.14 qui présentent les réductions de variance pour les fonctions f_2, f_6, f_9, f_{11}, f_{12}, f_{13} et f_{14} en 10 et 30. Toutes ces figures utilisent le même format. Elles sont composées de six graphiques qui présentent les réductions de variance pour $k = 10, 12, 14, 15, 16, 18$ où $k = rw$. L'échelle de l'axe vertical est logarithmique en base 10. Chacun de ces graphiques donne la réduction de variances d'ensemble de points sélectionnés par rapport à 6 critères d'uniformité. Chaque groupe de colonnes d'une même couleur représente un critère d'uniformité selon la table suivante :

Couleur	Critère
rouge	$\Delta(S_1, \Lambda)$
vert	$\Theta(S_1, \Lambda)$
bleu	$\Delta(S_1, q - \text{valeur})$
mauve	$\Theta(S_1, q - \text{valeur})$
turquoise	$\Delta(S_2, \Gamma_t)$
gris	$\Theta(S_2, \Gamma_t)$

Les deux dernières colonnes indiquent la réduction de variance obtenue pour les ensembles de points de Sobol avec le même nombre de points, mais avec deux randomisations différentes. L'avant-dernière colonne donne le résultat obtenu avec une simple translation digitale et la dernière illustre la réduction de variance dans le cas où on applique un mixage linéaire avec une translation digitale.

La hauteur de la colonne indique la valeur de $\log_{10} \hat{\rho}[f, C, r, w]$ obtenue quand la dimension de la fonction à intégrer est 10 pour les ensembles de points qui ont été sélectionnés par rapport au critère associé à la couleur de la colonne. Le « ◆ » situé dans le même axe vertical de la colonne indique la valeur de $\log_{10} \hat{\rho}[f, C, r, w]$ obtenue quand la dimension de la fonction à intégrer est 30 pour les ensembles de points qui ont été sélectionnés par rapport au critère associé à la couleur de la colonne.

Pour chaque couleur, il y a plusieurs colonnes qui correspondent chacune à un couple (r, w). Par exemple, quand $k = 10$, on a $(r, w) = (2, 5)$ et $(r, w) = (5, 2)$. La première colonne d'une couleur correspond au couple (r, w) avec la plus petite valeur de r, la deuxième colonne, à la deuxième plus petite valeur de r, etc..

Exemple 6.1. La figure 6.1 illustre un exemple d'un graphique dans le cas où $k = 10$ et où la fonction à intégrer est soit f_2, f_6, f_9, f_{11}, f_{12}, f_{13} ou f_{14}. La légende, en haut du graphique, indique quelles colonnes correspondent à quels critères. Les douze premières colonnes donnent les résultats obtenus par des ensembles de points avec

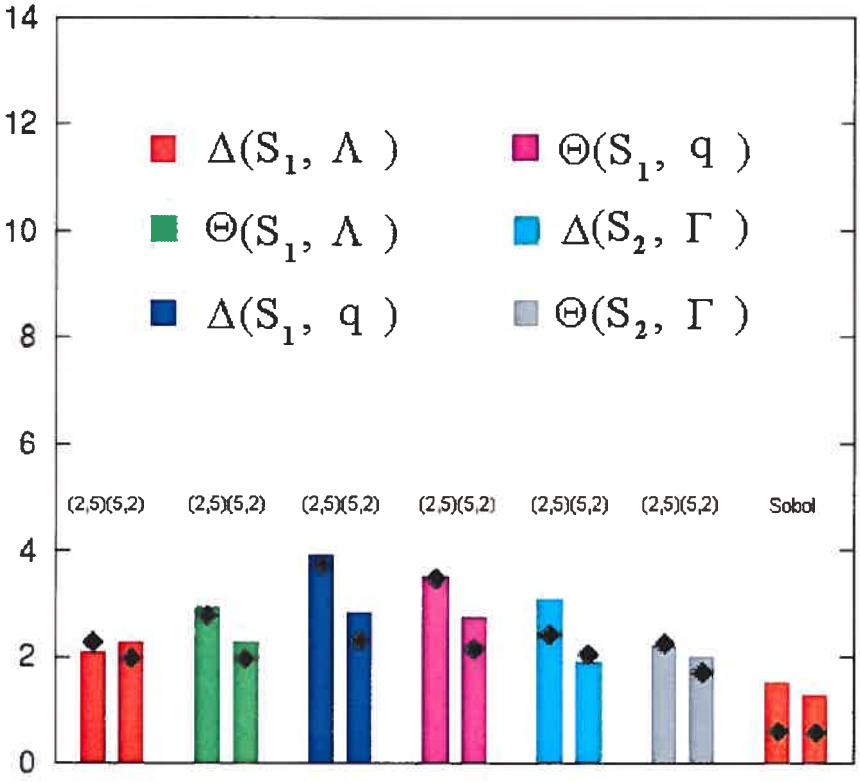


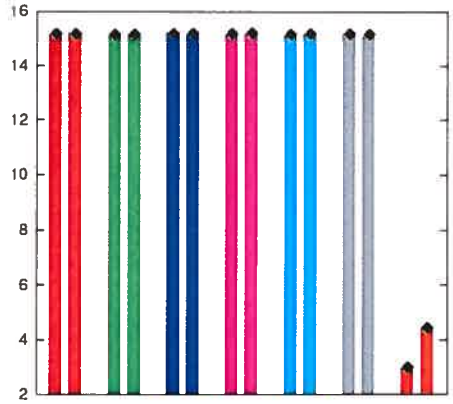
Figure 6.1 – Graphique expliquant la présentation des résultats.

$(r, w) = (2, 5)$ et $(r, w) = (5, 2)$, en alternance. Les deux dernières colonnes donnent les résultats obtenus pour les ensembles de points de Sobol randomisés avec la translation digitale et le mixage linéaire avec la translation digitale.

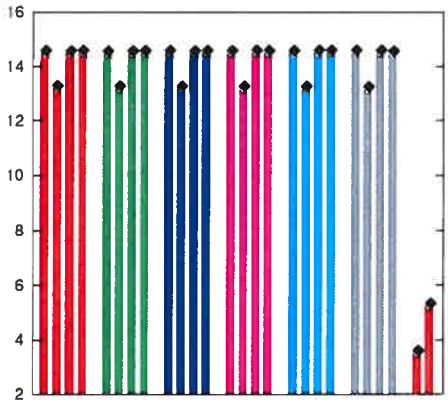
Pour les tableaux 6.15 à 6.18, le format est similaire sauf que la hauteur des colonnes correspond à la réduction de variance dans le cas où la dimension de la fonction est $nm = 100$ et m est indiqué dans l'en-tête du tableau.

Pour les tableaux 6.19 à 6.30 (où on présente les facteurs de réduction de variance pour l'évaluation du prix d'option asiatique, pour vega et delta), le format est similaire à celui décrit par l'exemple 6.1.

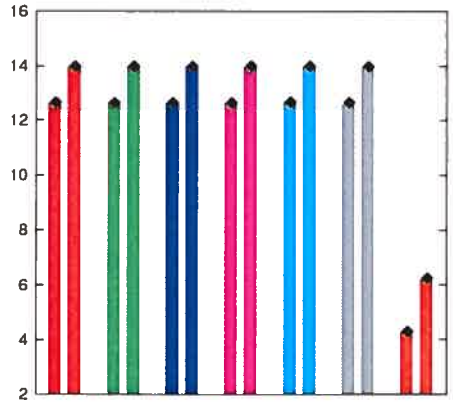
Tableau 6.8 – Réduction de variance pour la fonction f_2.



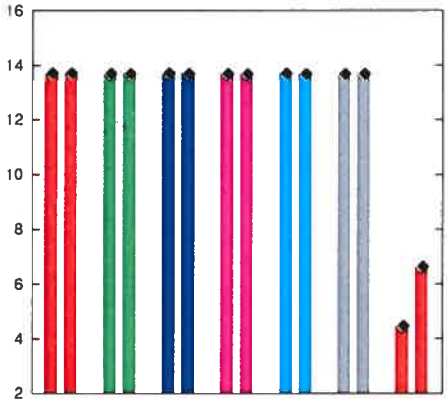
$k = 10$



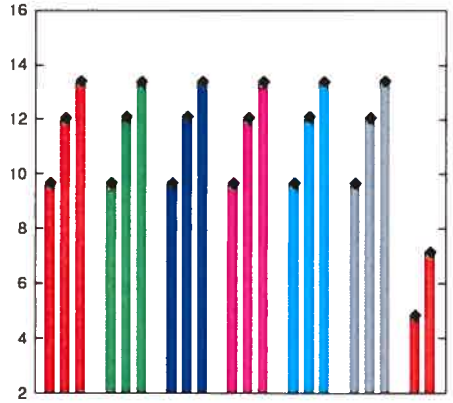
$k = 12$



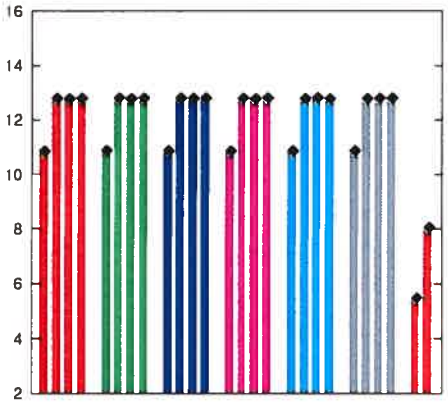
$k = 14$



$k = 15$

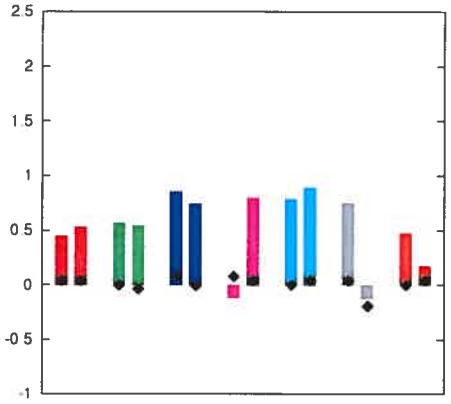


$k = 16$

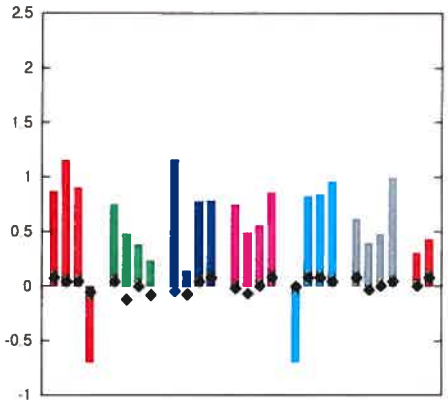


$k = 18$

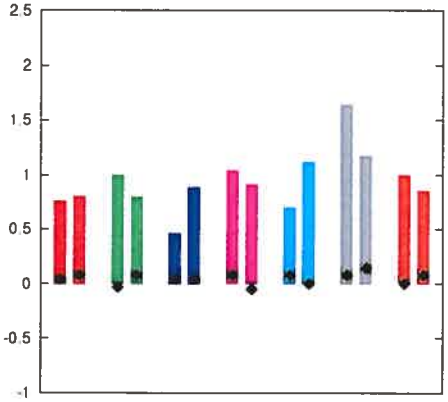
Tableau 6.9 – Réduction de variance pour la fonction f_6.



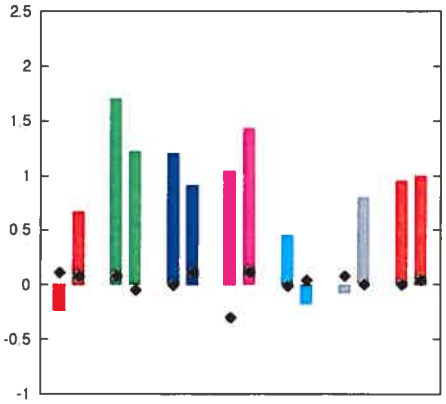
$k = 10$



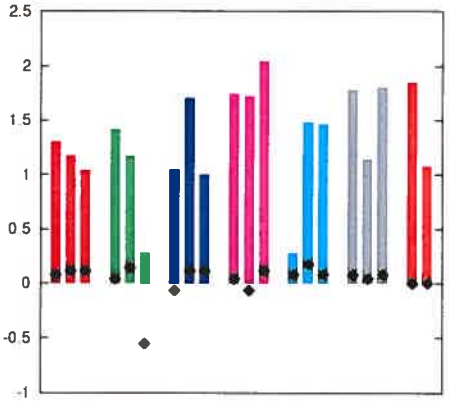
$k = 12$



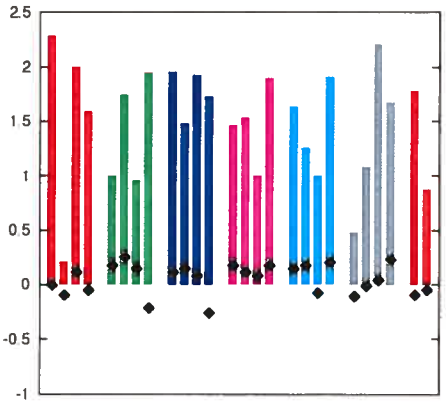
$k = 14$



$k = 15$

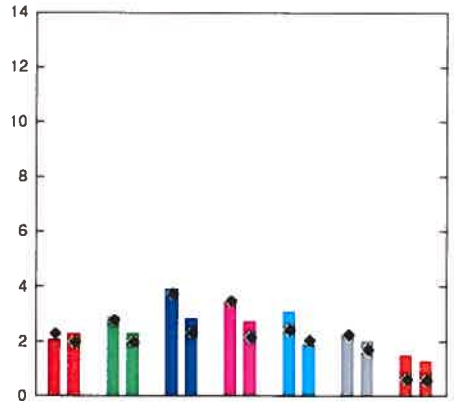


$k = 16$

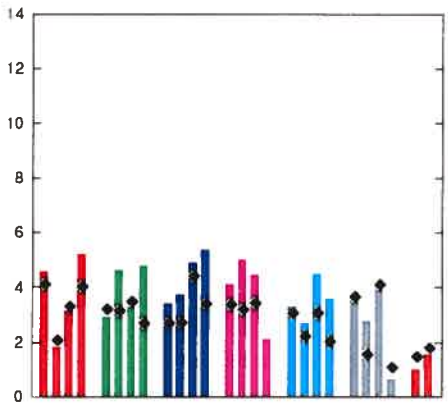


$k = 18$

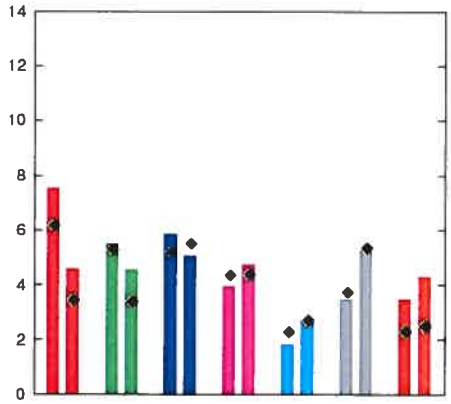
Tableau 6.10 – Réduction de variance pour la fonction f_9.



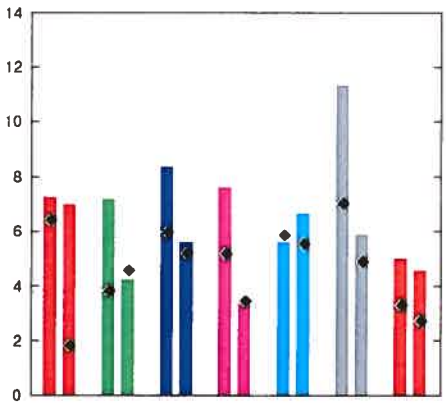
$k = 10$



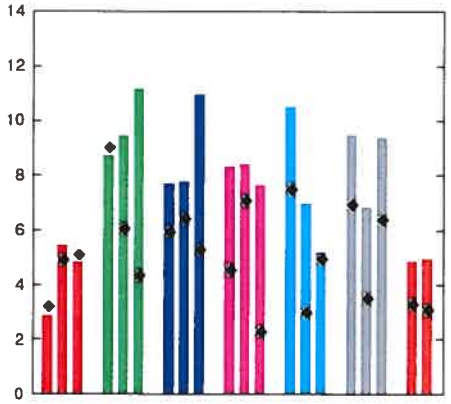
$k = 12$



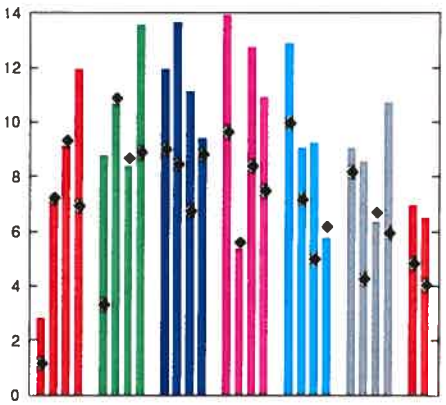
$k = 14$



$k = 15$

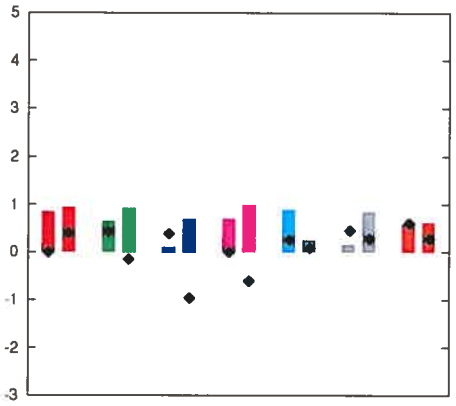


$k = 16$

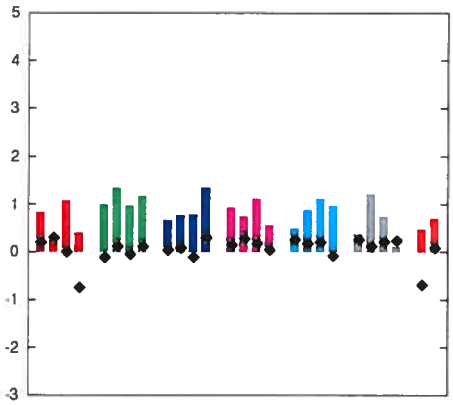


$k = 18$

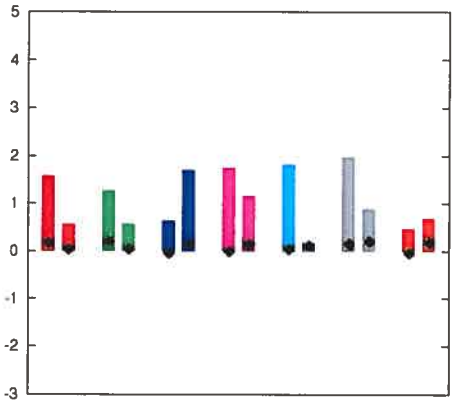
Tableau 6.11 – Réduction de variance pour la fonction f_{11}.



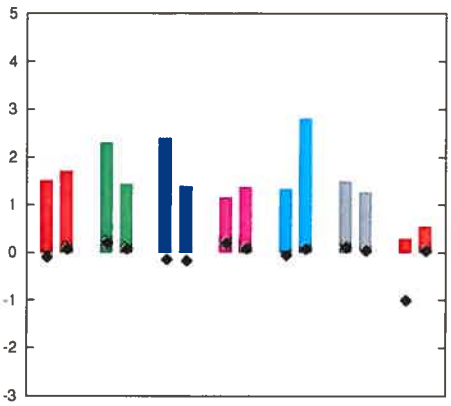
$k = 10$



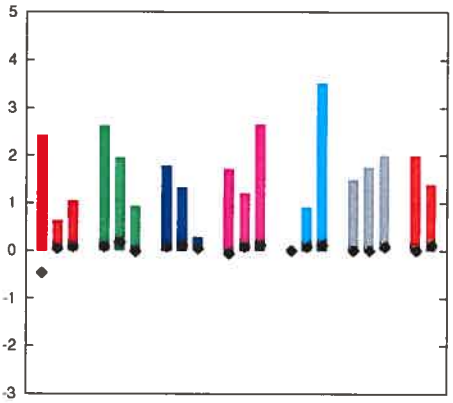
$k = 12$



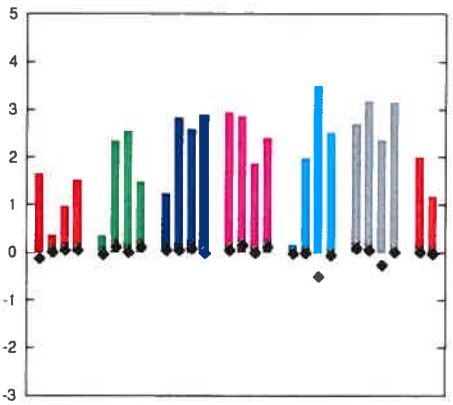
$k = 14$



$k = 15$

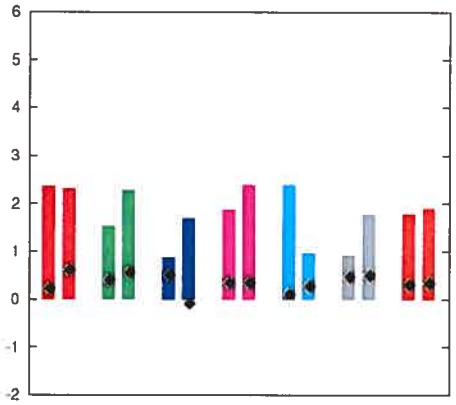


$k = 16$

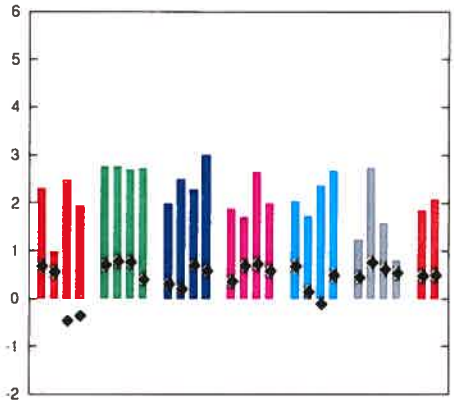


$k = 18$

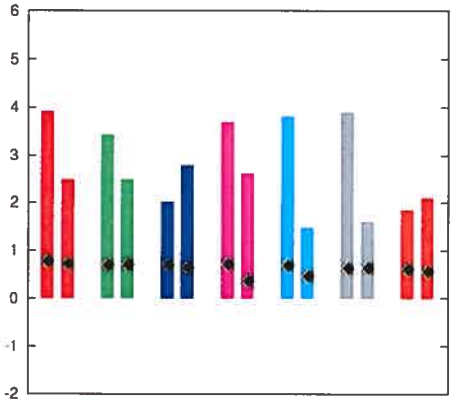
Tableau 6.12 – Réduction de variance pour la fonction f_{12}.



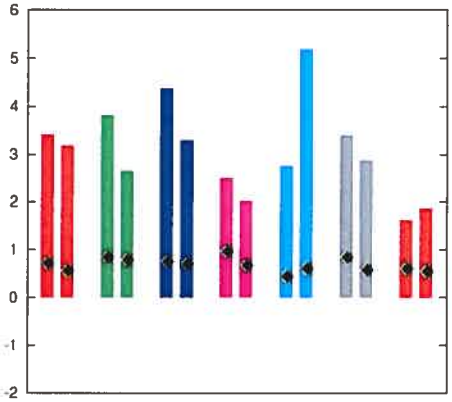
$k = 10$



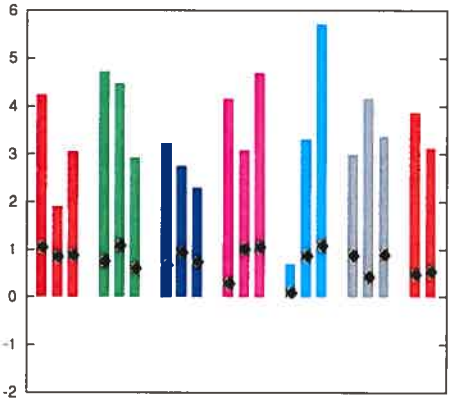
$k = 12$



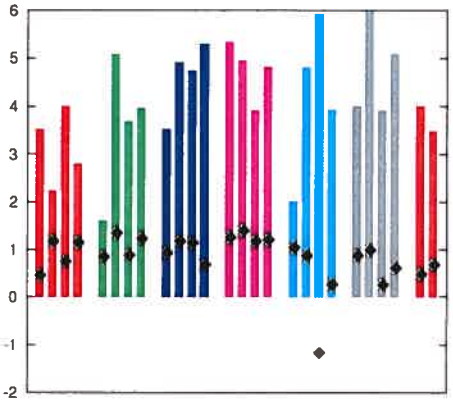
$k = 14$



$k = 15$

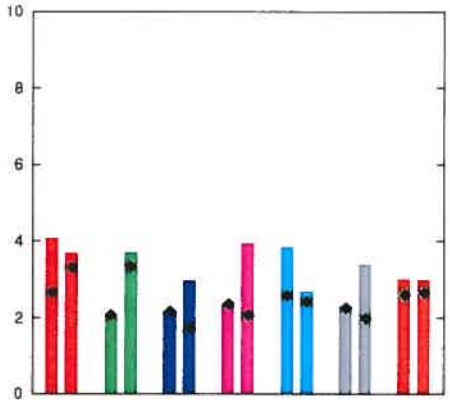


$k = 16$

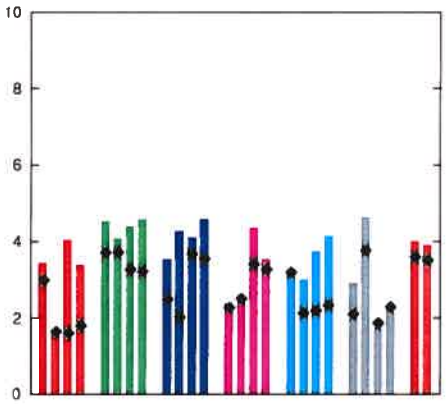


$k = 18$

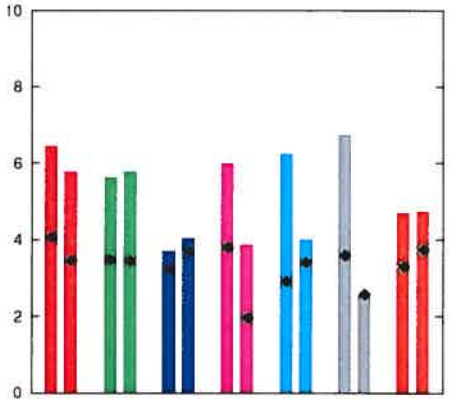
Tableau 6.13 – Réduction de variance pour la fonction f_{13}.



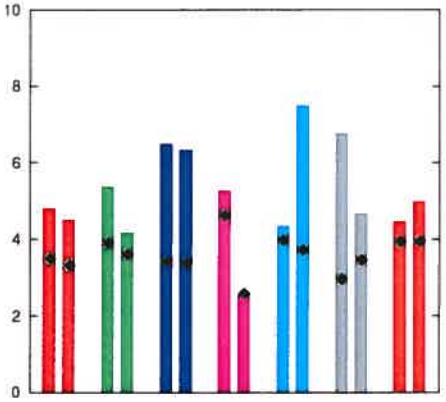
$k = 10$



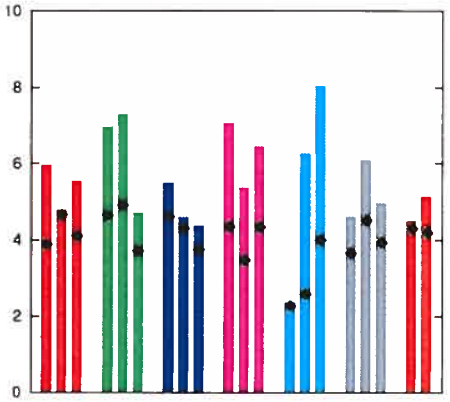
$k = 12$



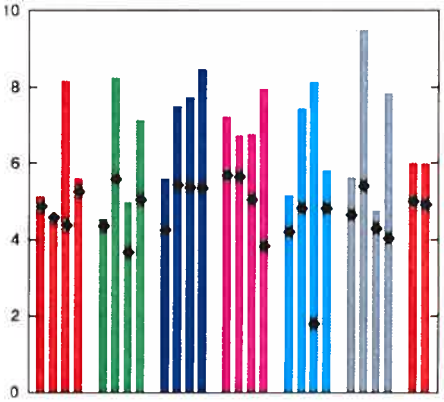
$k = 14$



$k = 15$

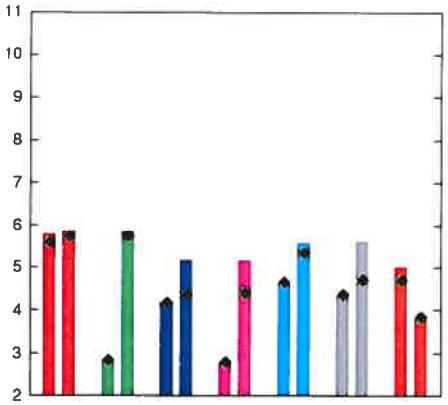


$k = 16$

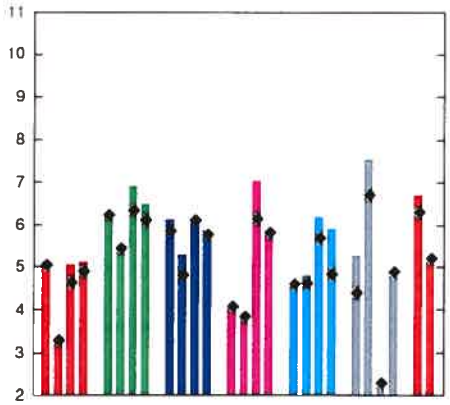


$k = 18$

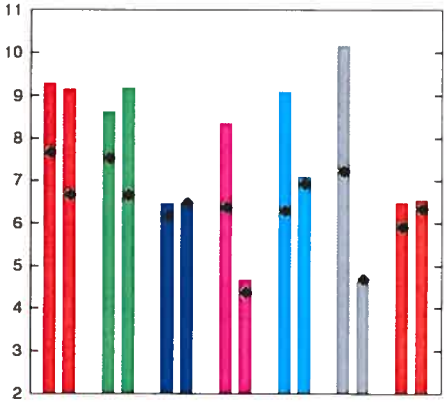
Tableau 6.14 – Réduction de variance pour la fonction f_{14}.



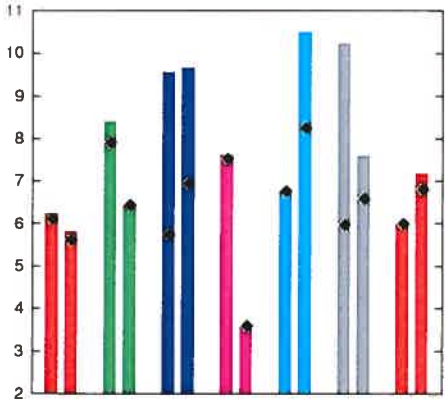
$k = 10$



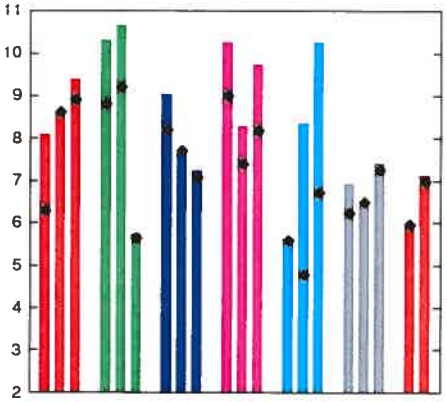
$k = 12$



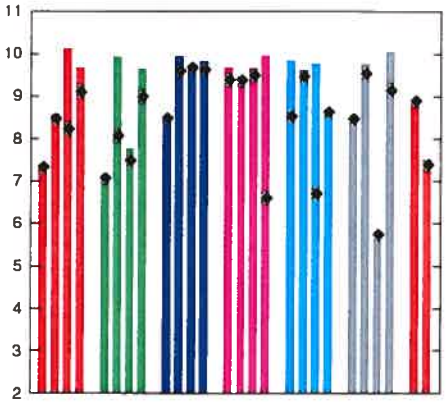
$k = 14$



$k = 15$

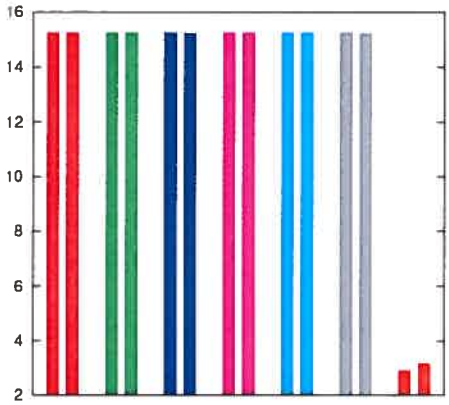


$k = 16$

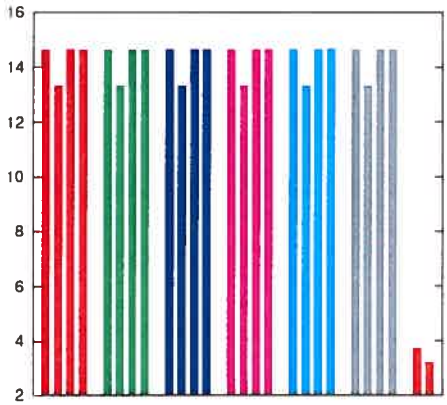


$k = 18$

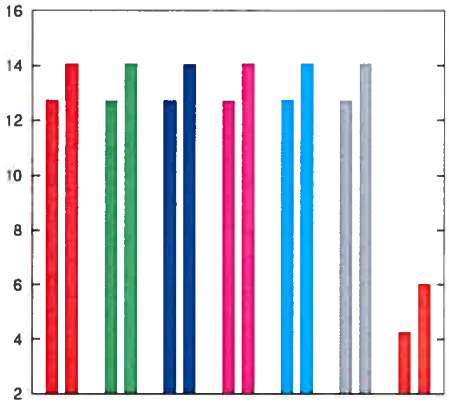
Tableau 6.15 – Réduction de variance pour la fonction f_{15}, $m = 2$.



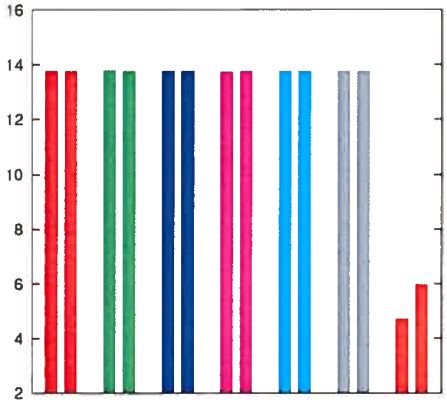
$k = 10$



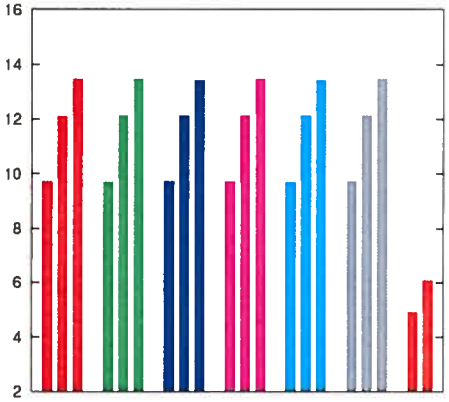
$k = 12$



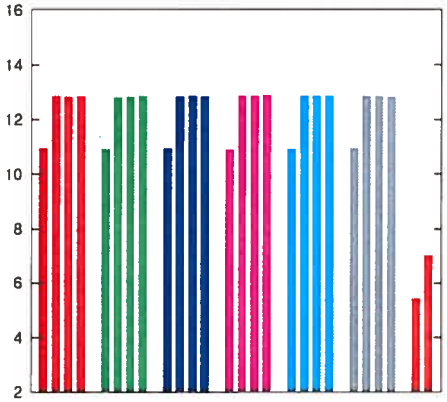
$k = 14$



$k = 15$

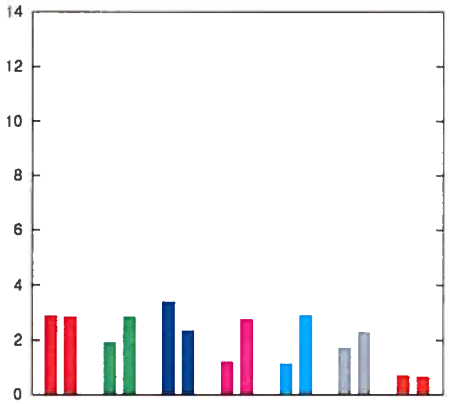


$k = 16$

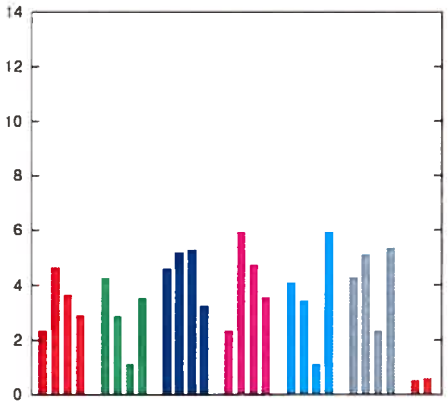


$k = 18$

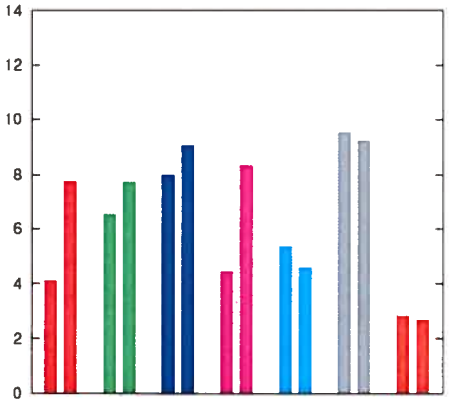
Tableau 6.16 – Réduction de variance pour la fonction f_{15}, $m = 5$.



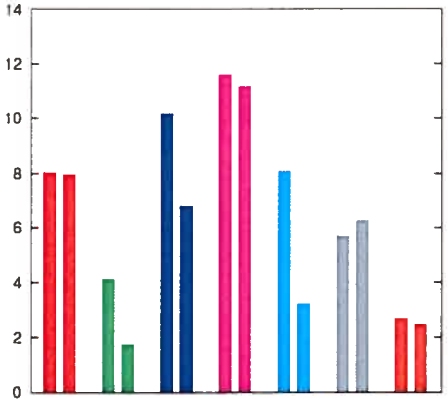
$k = 10$



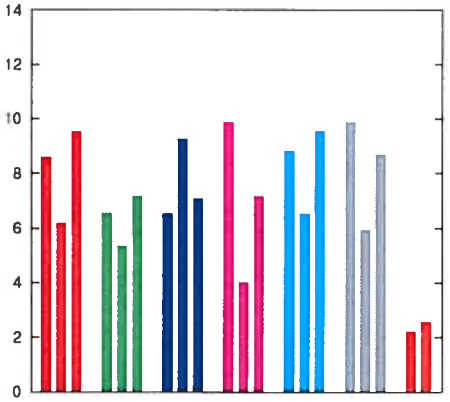
$k = 12$



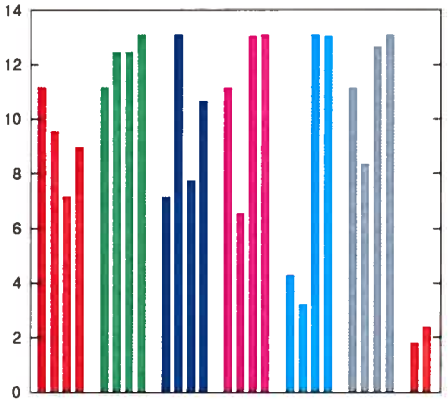
$k = 14$



$k = 15$

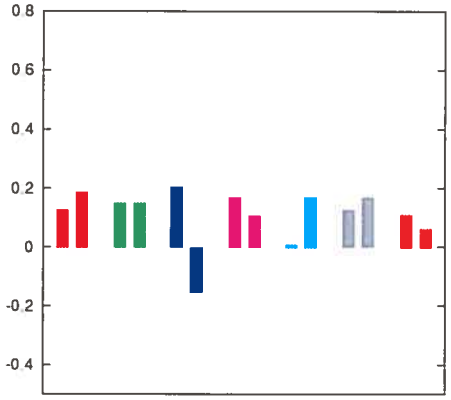


$k = 16$

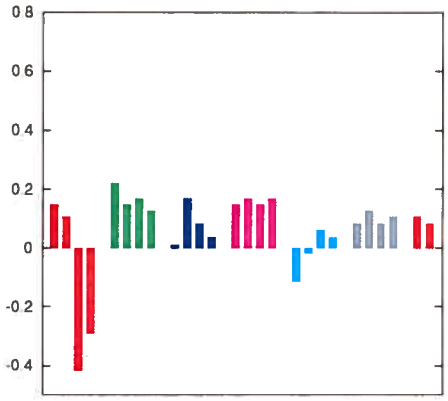


$k = 18$

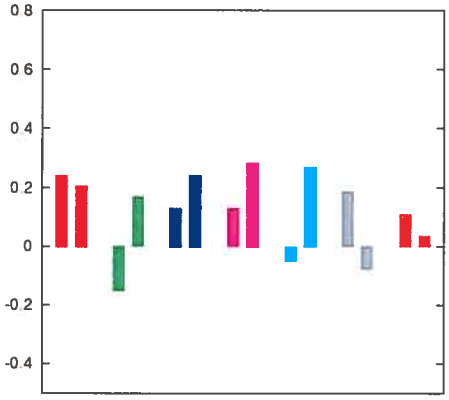
Tableau 6.17 – Réduction de variance pour la fonction f_{15}, $m = 20$.



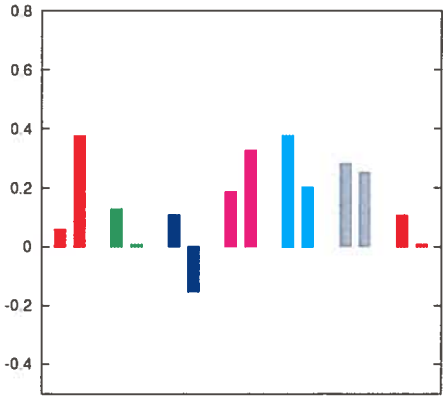
$k = 10$



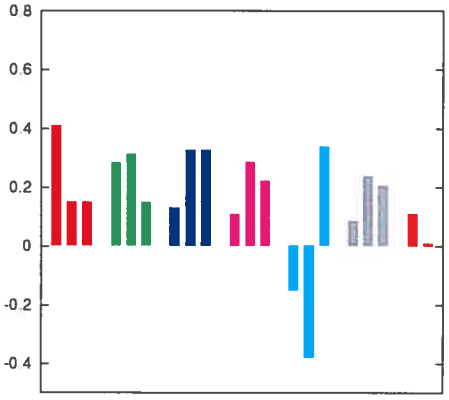
$k = 12$



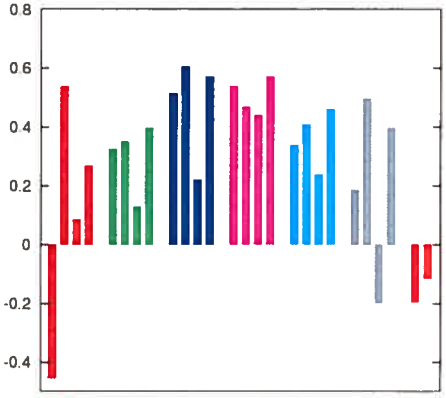
$k = 14$



$k = 15$

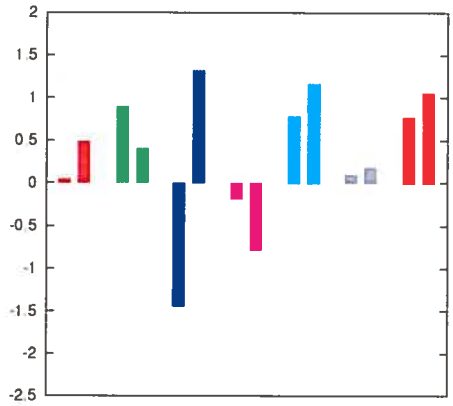


$k = 16$

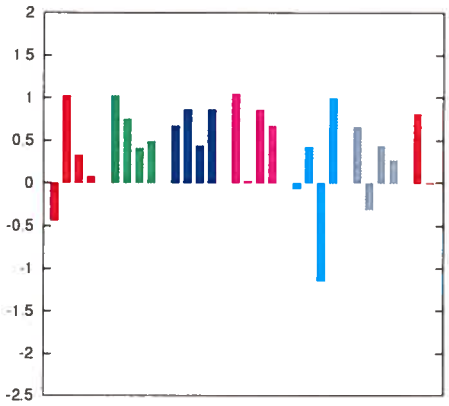


$k = 18$

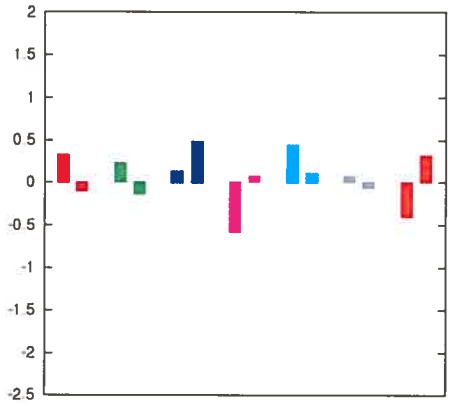
Tableau 6.18 – Réduction de variance pour la fonction f_{15}, $m = 50$.



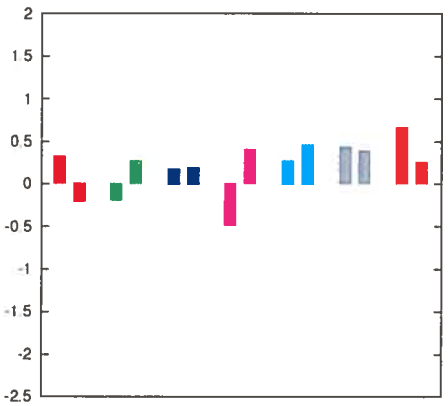
$k = 10$



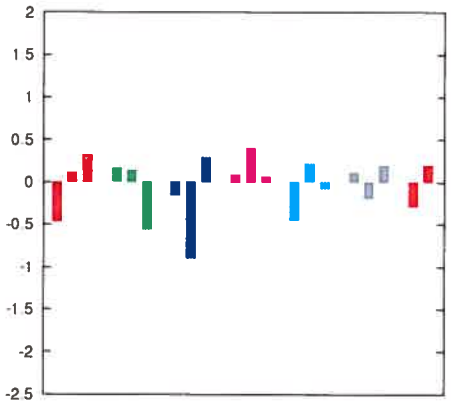
$k = 12$



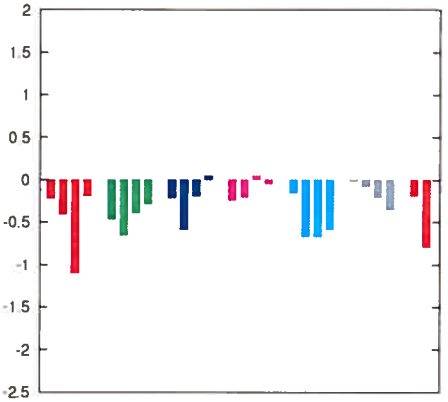
$k = 14$



$k = 15$

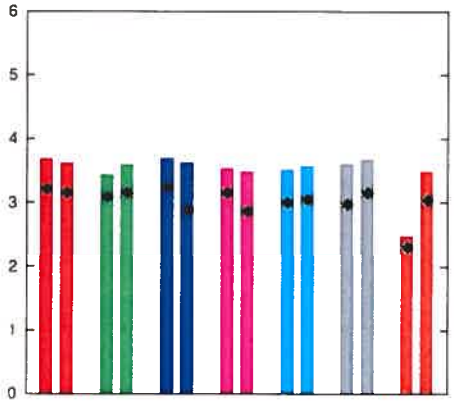


$k = 16$

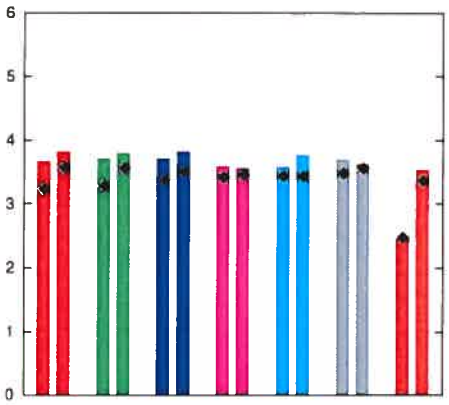


$k = 18$

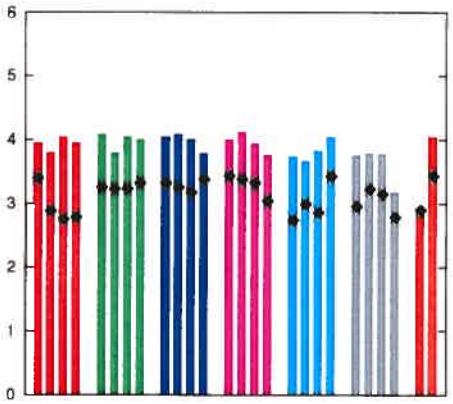
Tableau 6.19 – Réduction de variance pour le prix de l'option asiatique, $K = 90$.



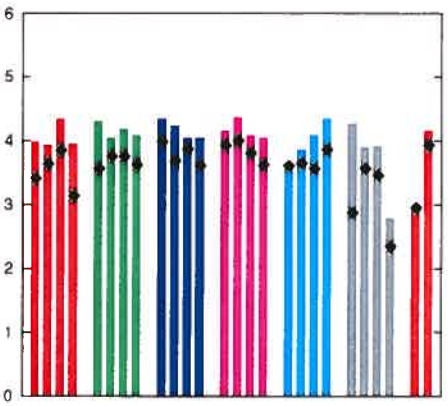
$k = 10$



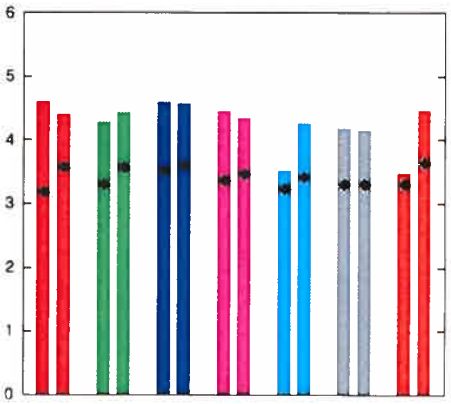
$k = 10$ (Pont Brownien)



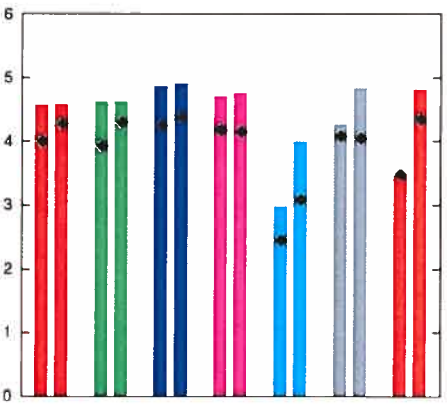
$k = 12$



$k = 12$ (Pont Brownien)

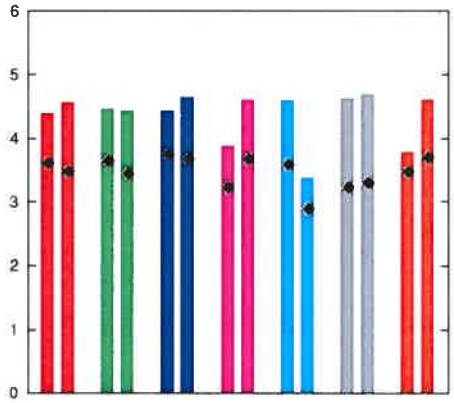


$k = 14$

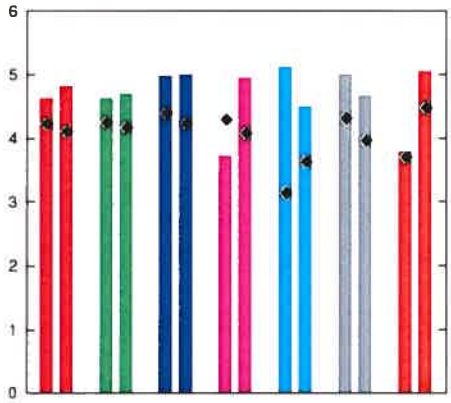


$k = 14$ (Pont Brownien)

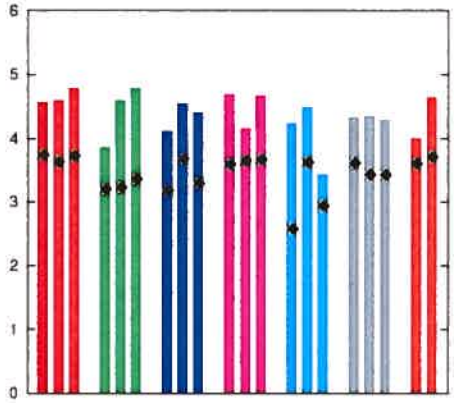
Tableau 6.20 – Réduction de variance pour le prix de l'option asiatique, $K = 90$.



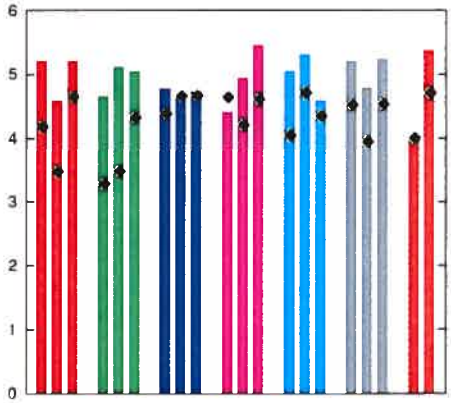
$k = 15$



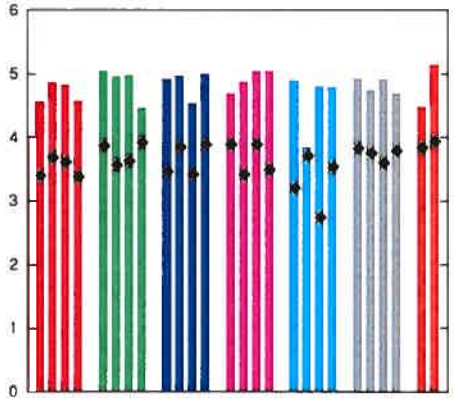
$k = 15$ (Pont Brownien)



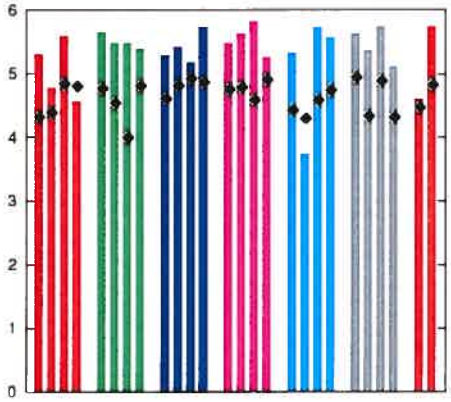
$k = 16$



$k = 16$ (Pont Brownien)

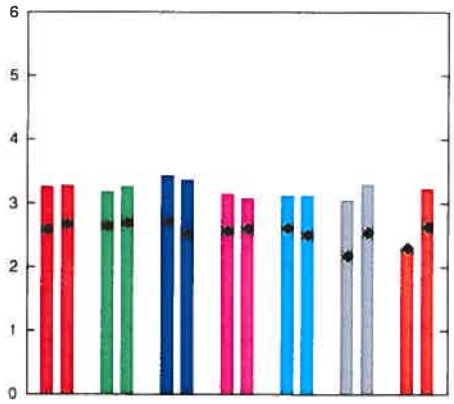


$k = 18$

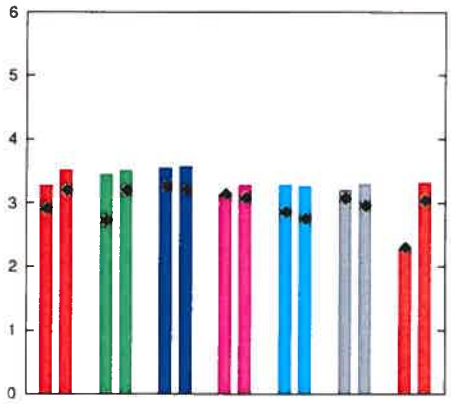


$k = 18$ (Pont Brownien)

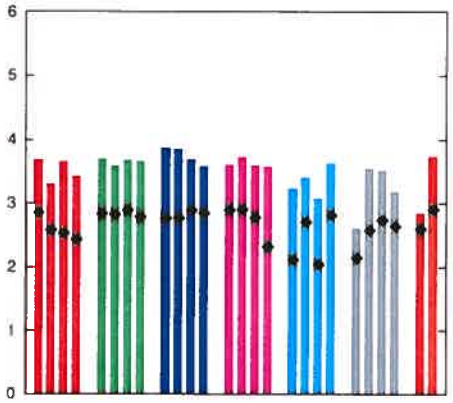
Tableau 6.21 – Réduction de variance pour le prix de l'option asiatique, $K = 100$.



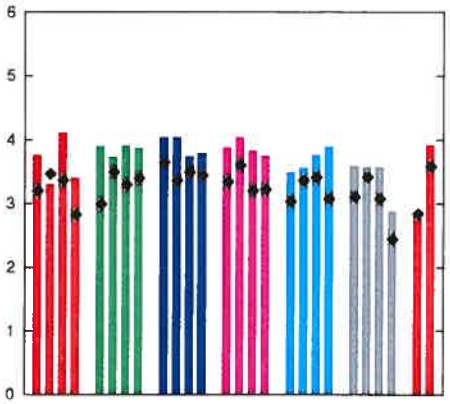
$k = 10$



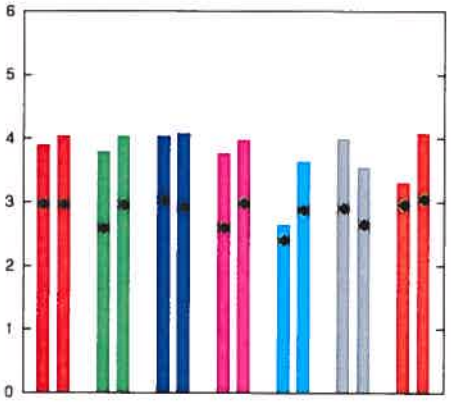
$k = 10$ (Pont Brownien)



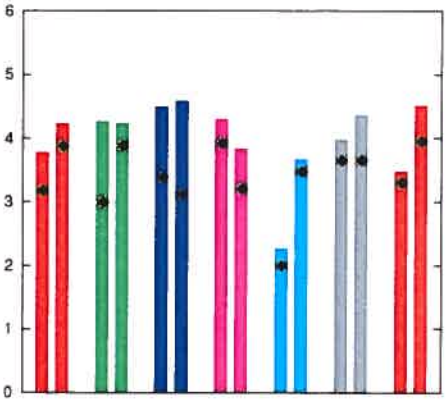
$k = 12$



$k = 12$ (Pont Brownien)

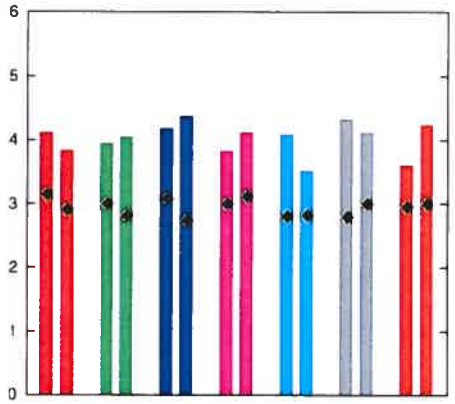


$k = 14$

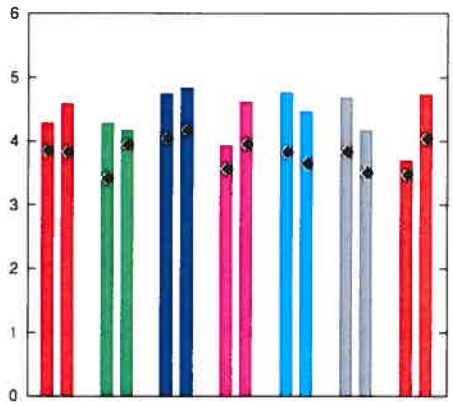


$k = 14$ (Pont Brownien)

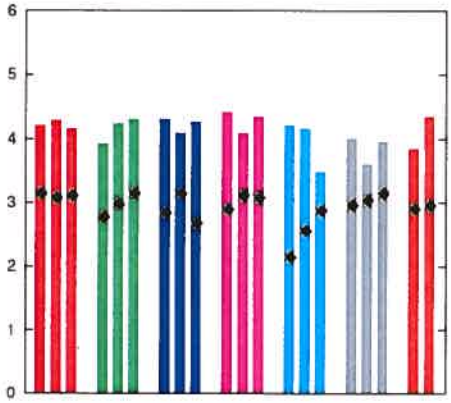
Tableau 6.22 – Réduction de variance pour le prix de l'option asiatique, $K = 100$.



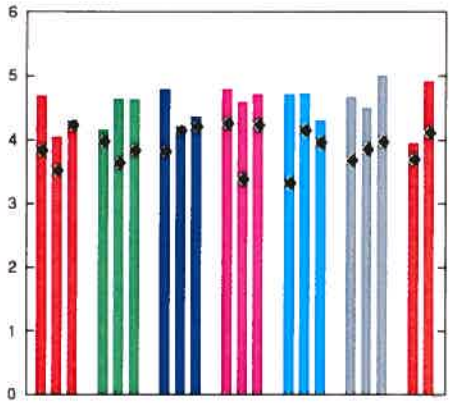
$k = 15$



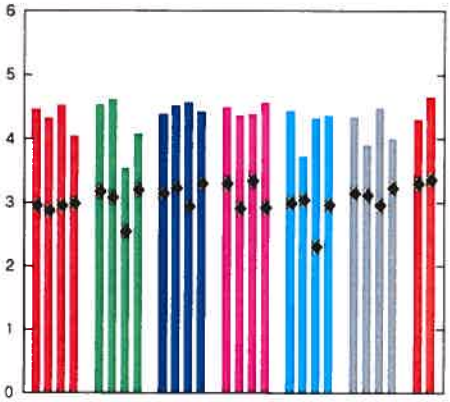
$k = 15$ (Pont Brownien)



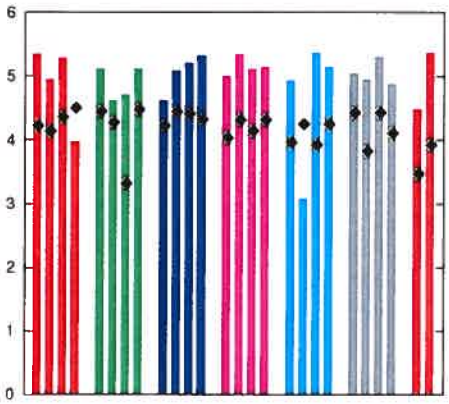
$k = 16$



$k = 16$ (Pont Brownien)

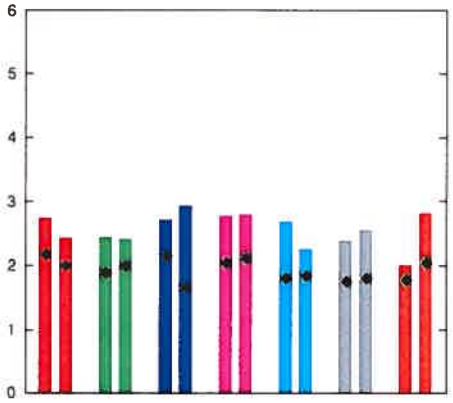


$k = 18$

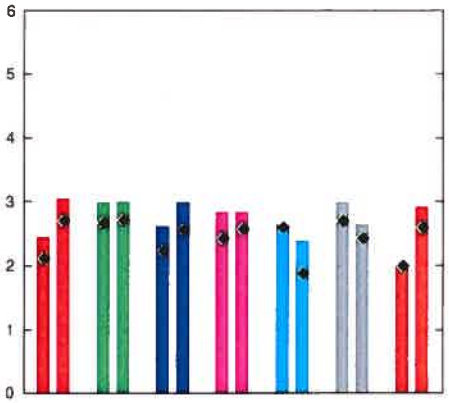


$k = 18$ (Pont Brownien)

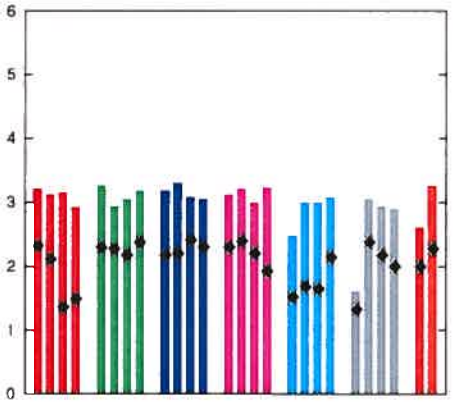
Tableau 6.23 – Réduction de variance pour le prix de l'option asiatique, $K = 110$.



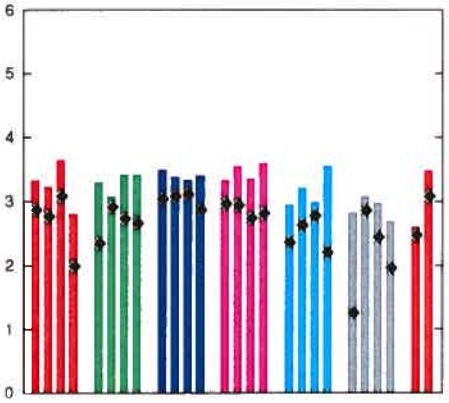
$k = 10$



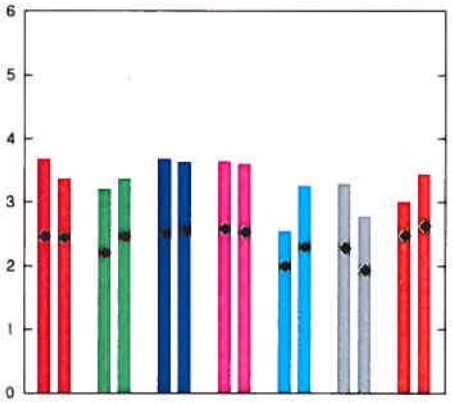
$k = 10$ (Pont Brownien)



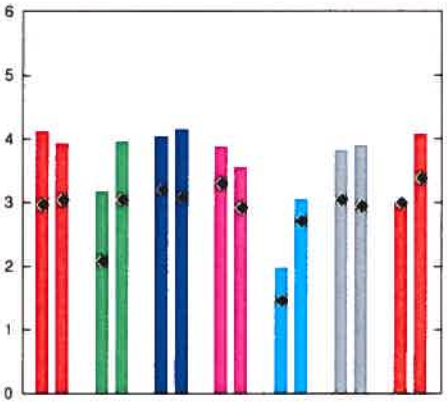
$k = 12$



$k = 12$ (Pont Brownien)

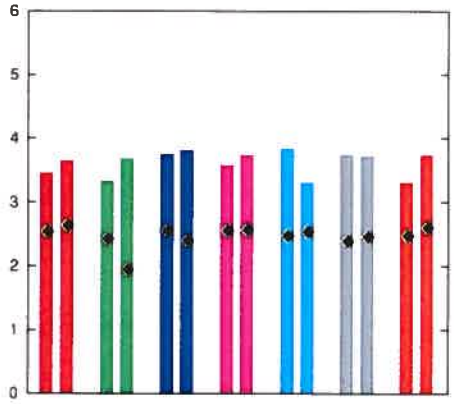


$k = 14$

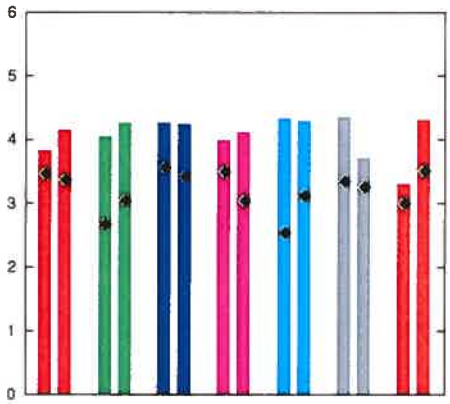


$k = 14$ (Pont Brownien)

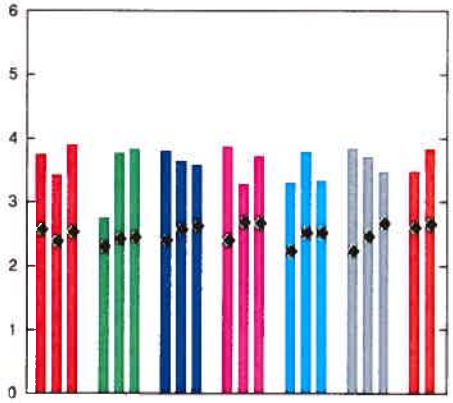
Tableau 6.24 – Réduction de variance pour le prix de l'option asiatique, $K = 110$.



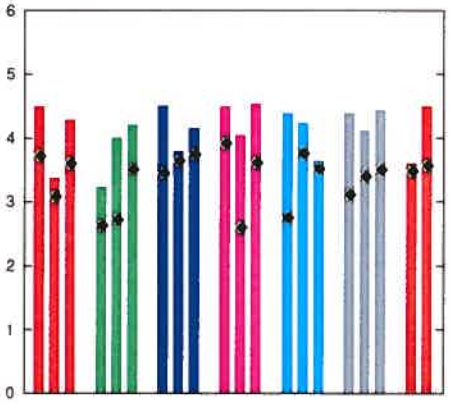
$k = 15$



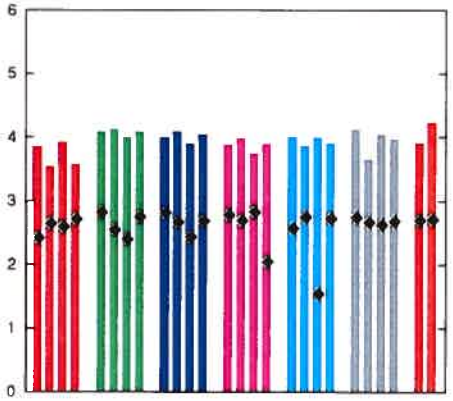
$k = 15$ (Pont Brownien)



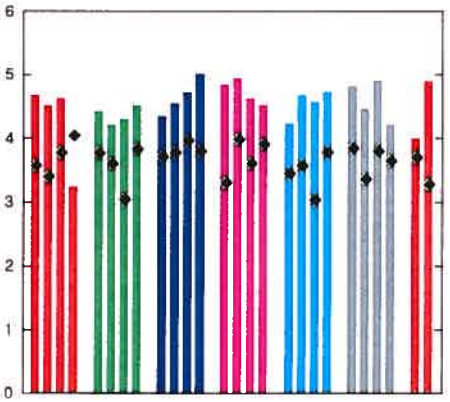
$k = 16$



$k = 16$ (Pont Brownien)

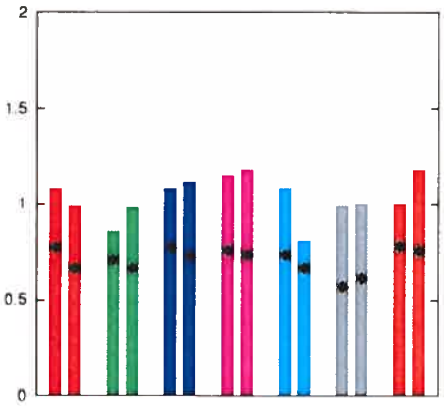


$k = 18$

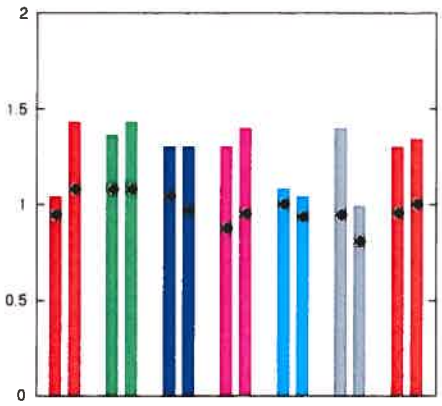


$k = 18$ (Pont Brownien)

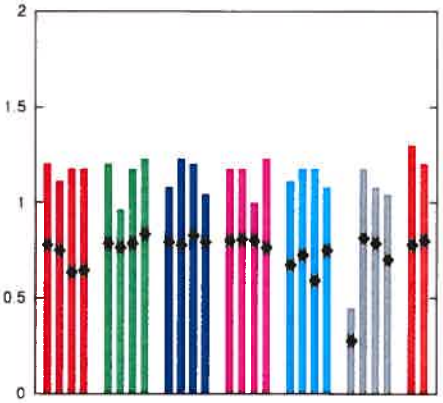
Tableau 6.25 – Réduction de variance pour la fonction vega, $S(0) = 90$.



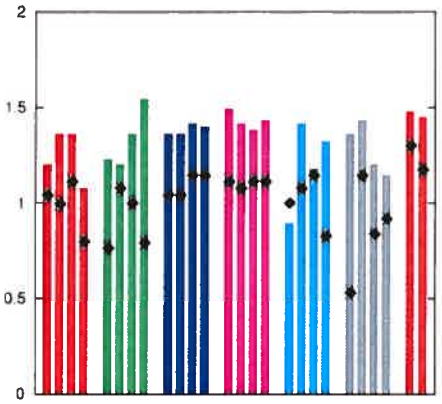
$k = 10$



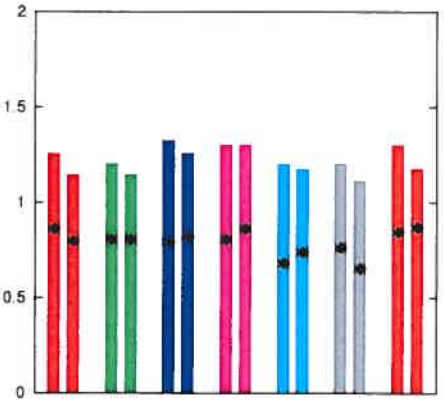
$k = 10$ (Pont Brownien)



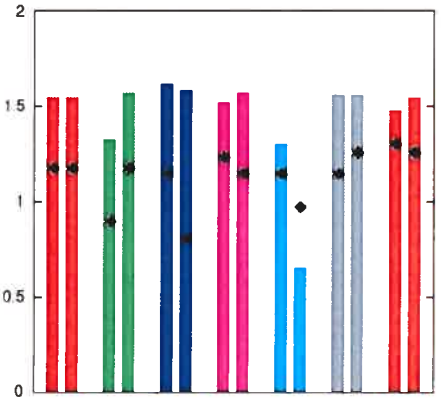
$k = 12$



$k = 12$ (Pont Brownien)

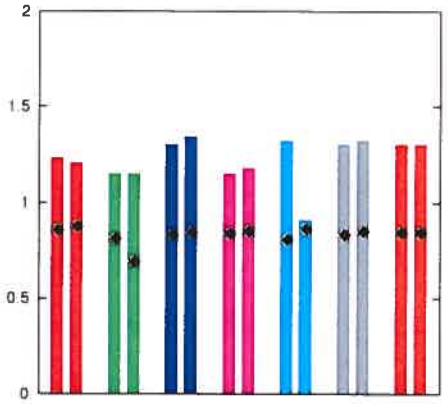


$k = 14$

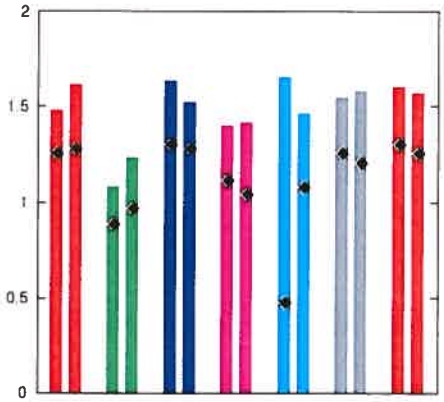


$k = 14$ (Pont Brownien)

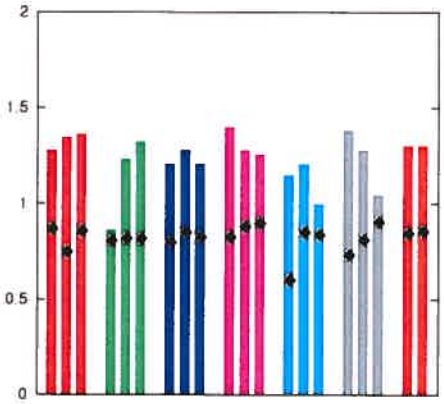
Tableau 6.26 – Réduction de variance pour la fonction vega, $S(0) = 90$.



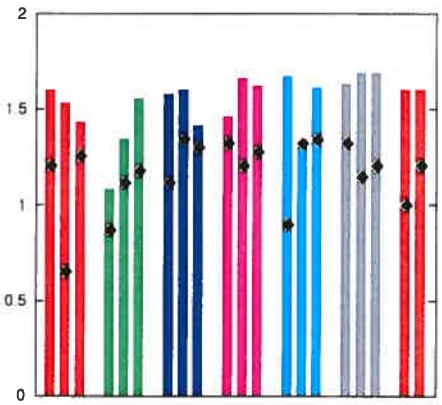
$k = 15$



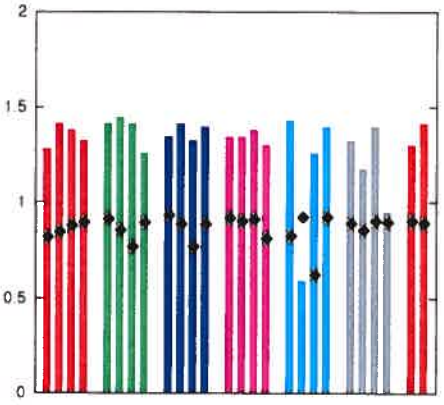
$k = 15$ (Pont Brownien)



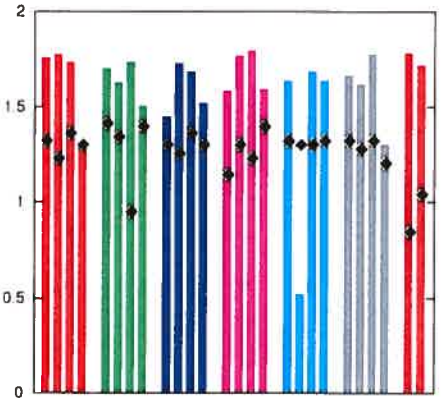
$k = 16$



$k = 16$ (Pont Brownien)

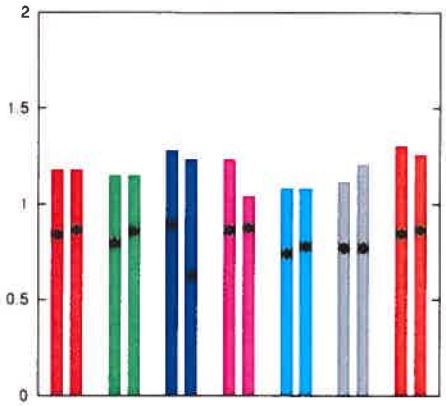


$k = 18$

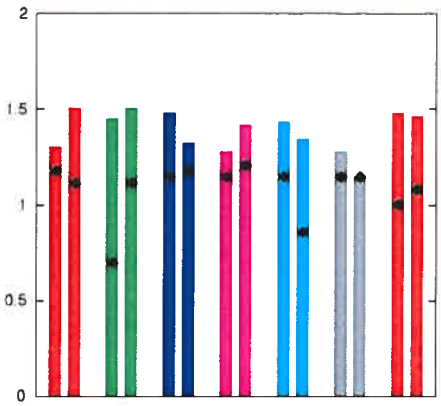


$k = 18$ (Pont Brownien)

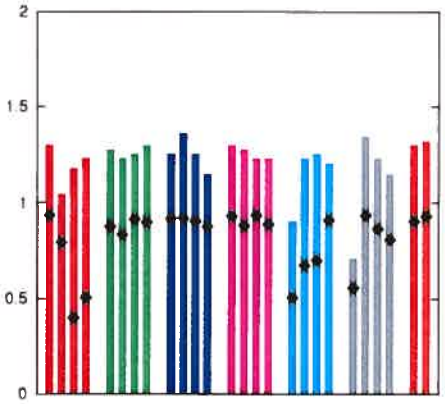
Tableau 6.27 – Réduction de variance pour la fonction vega, $S(0) = 100$.



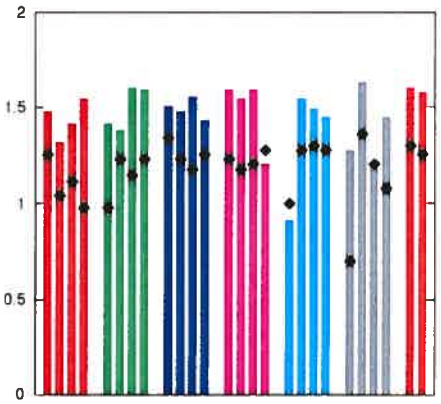
$k = 10$



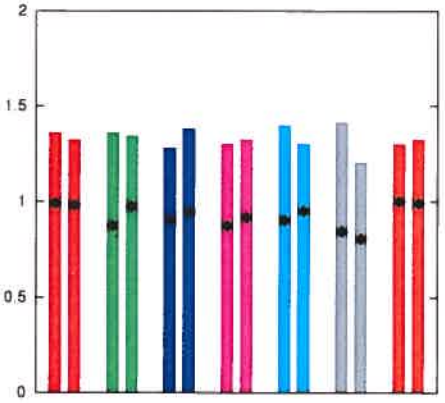
$k = 10$ (Pont Brownien)



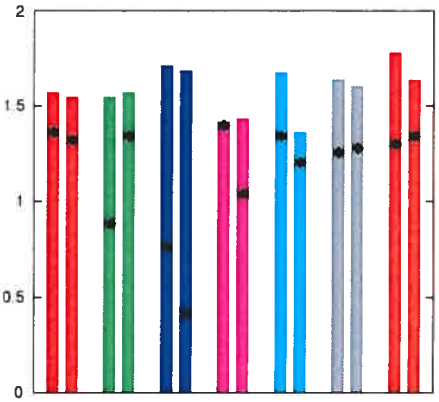
$k = 12$



$k = 12$ (Pont Brownien)

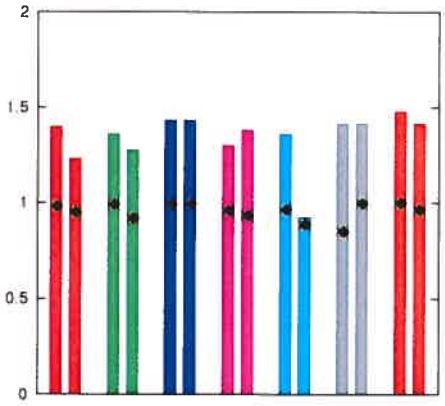


$k = 14$

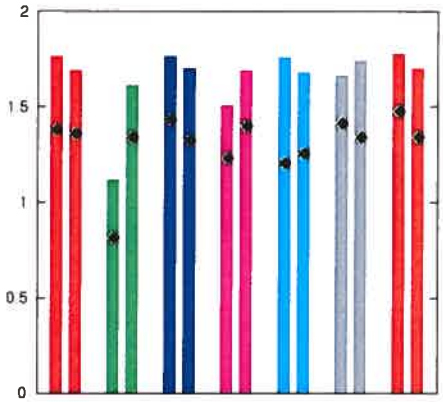


$k = 14$ (Pont Brownien)

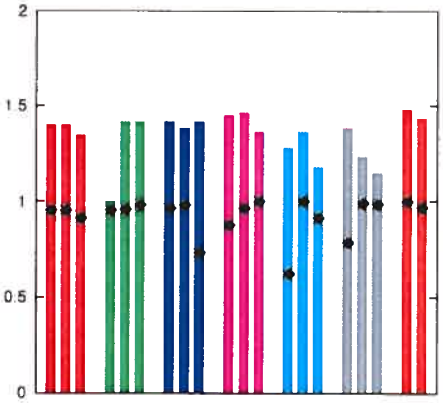
Tableau 6.28 – Réduction de variance pour la fonction vega, $S(0) = 100$.



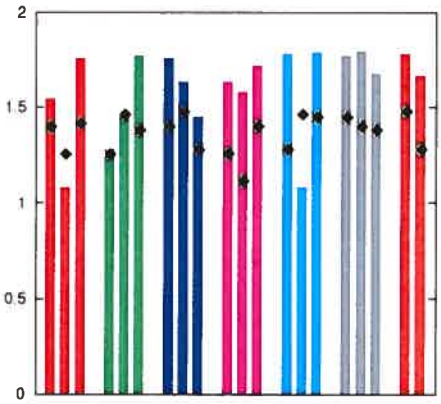
$k = 15$



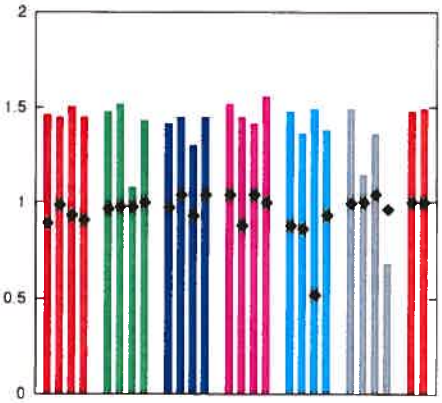
$k = 15$ (Pont Brownien)



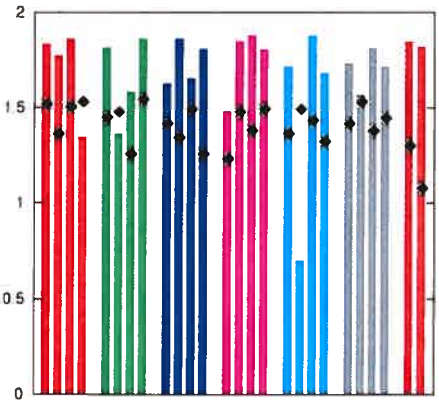
$k = 16$



$k = 16$ (Pont Brownien)

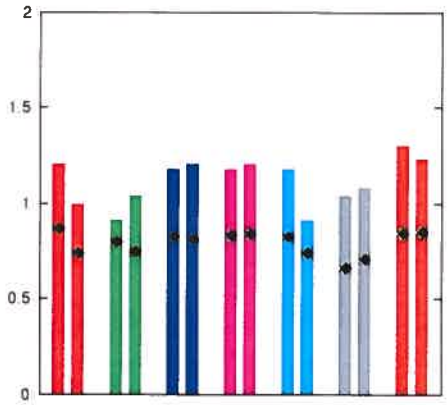


$k = 18$

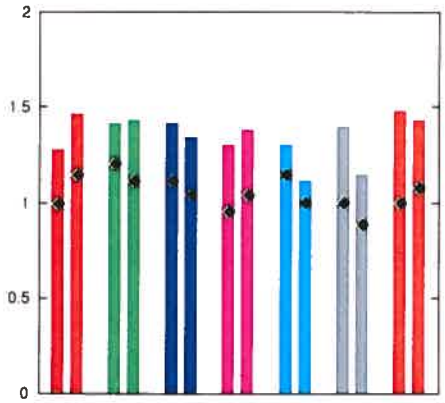


$k = 18$ (Pont Brownien)

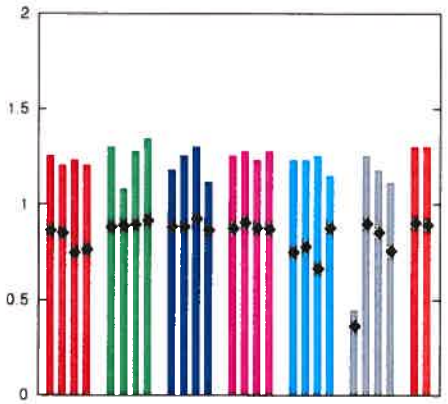
Tableau 6.29 – Réduction de variance pour la fonction vega, $S(0) = 110$.



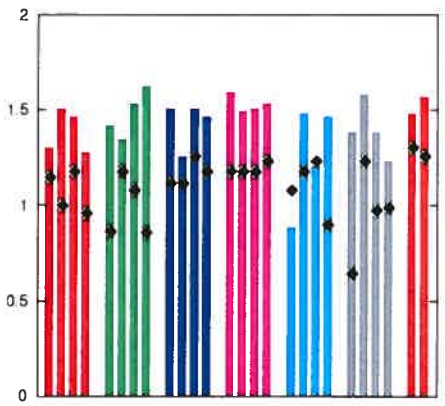
$k = 10$



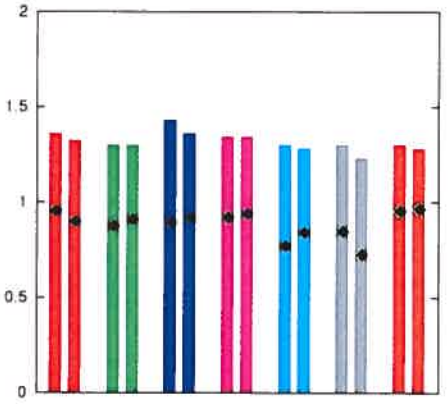
$k = 10$ (Pont Brownien)



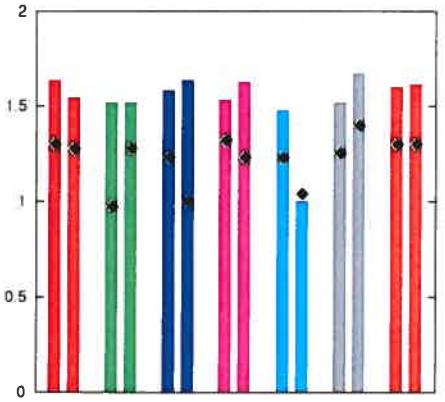
$k = 12$



$k = 12$ (Pont Brownien)

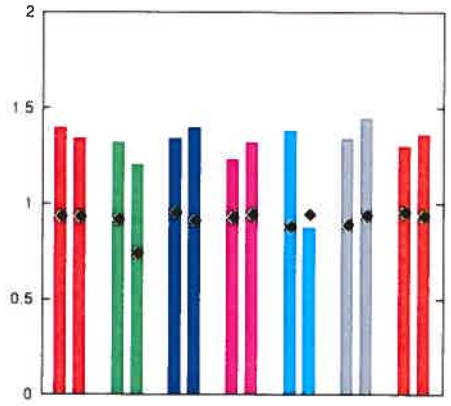


$k = 14$

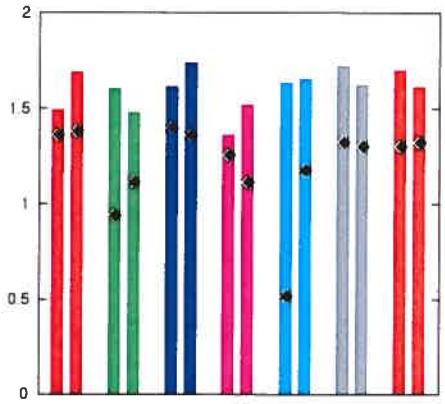


$k = 14$ (Pont Brownien)

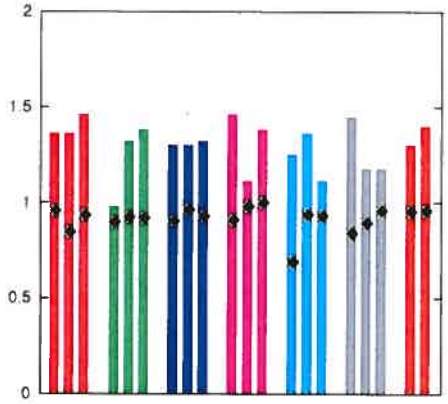
Tableau 6.30 – Réduction de variance pour la fonction vega, $S(0) = 110$.



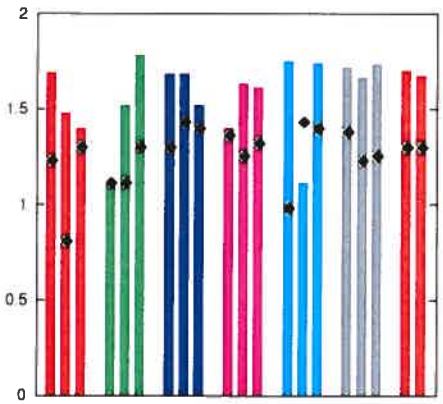
$k = 15$



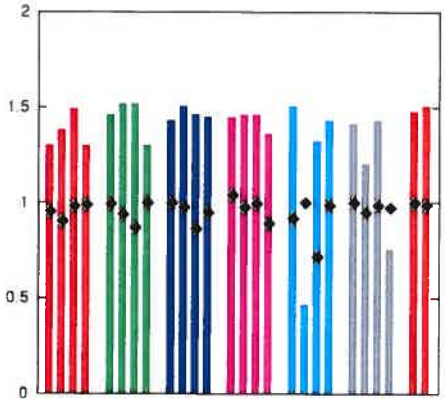
$k = 15$ (Pont Brownien)



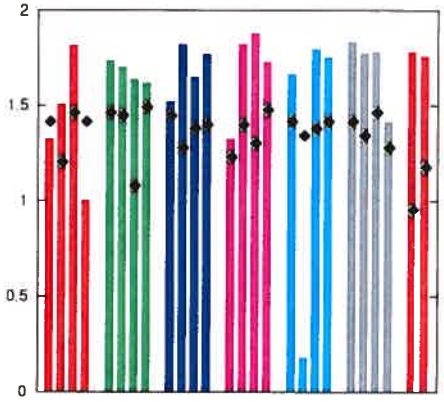
$k = 16$



$k = 16$ (Pont Brownien)

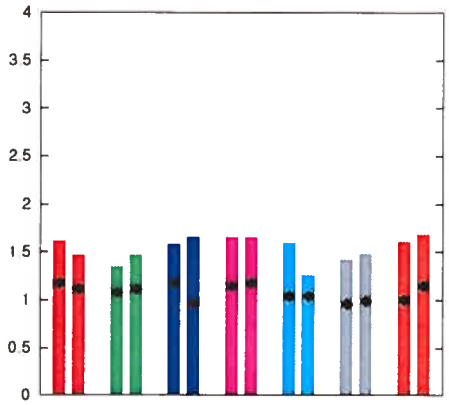


$k = 18$

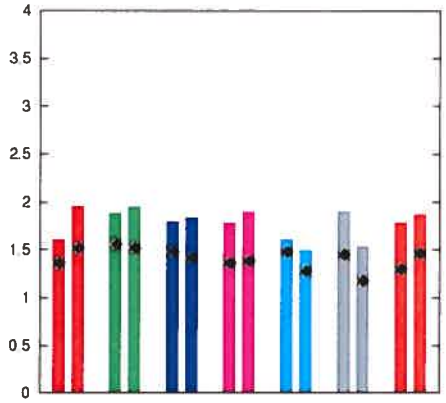


$k = 18$ (Pont Brownien)

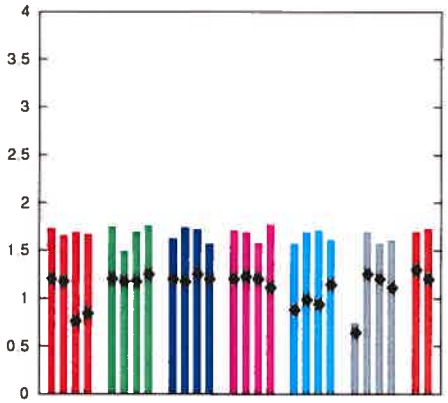
Tableau 6.31 – Réduction de variance pour la fonction delta, $S(0) = 90$.



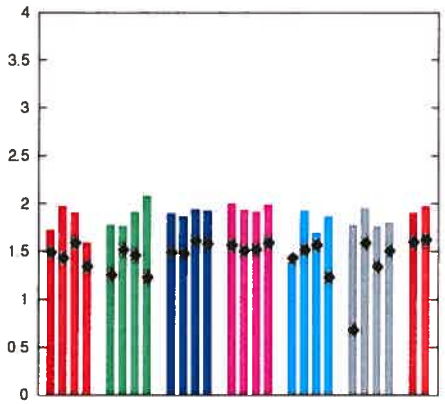
$k = 10$



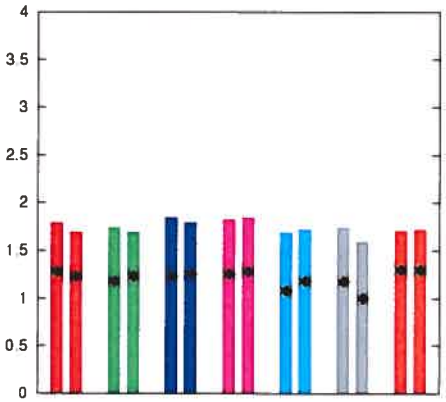
$k = 10$ (Pont Brownien)



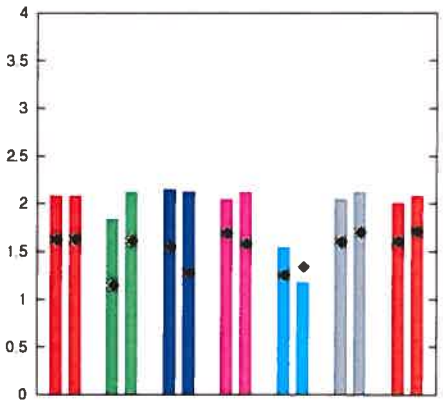
$k = 12$



$k = 12$ (Pont Brownien)

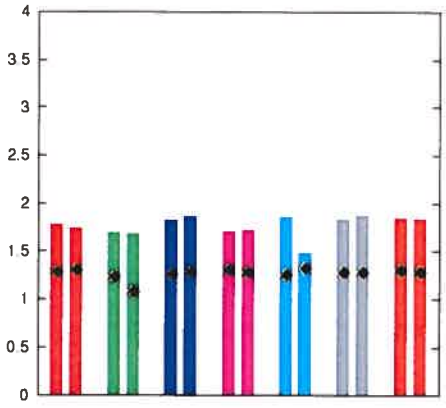


$k = 14$

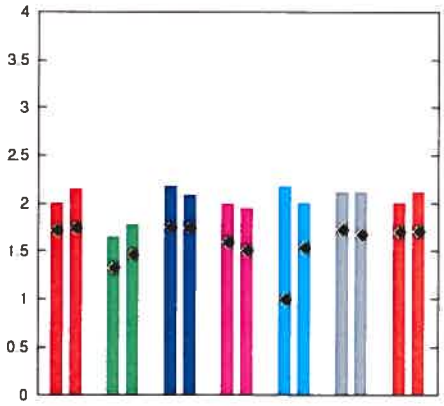


$k = 14$ (Pont Brownien)

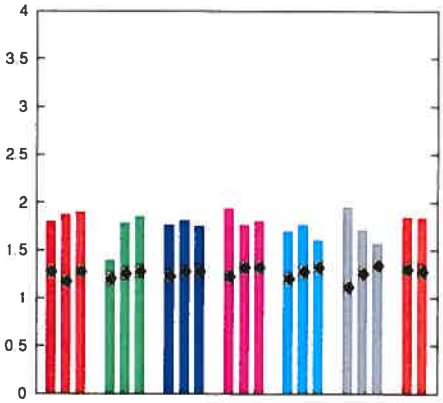
Tableau 6.32 – Réduction de variance pour la fonction delta, $S(0) = 90$.



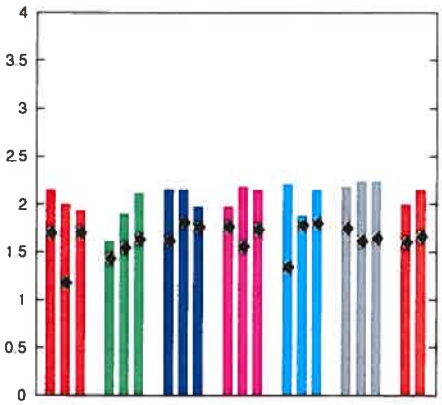
$k = 15$



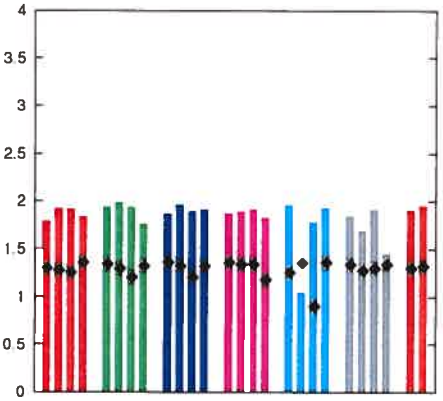
$k = 15$ (Pont Brownien)



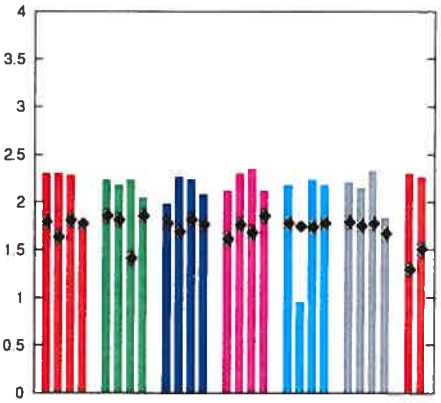
$k = 16$



$k = 16$ (Pont Brownien)

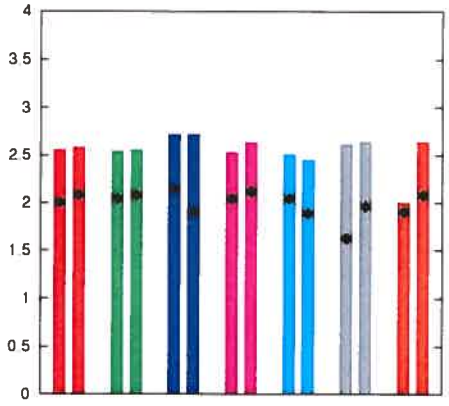


$k = 18$

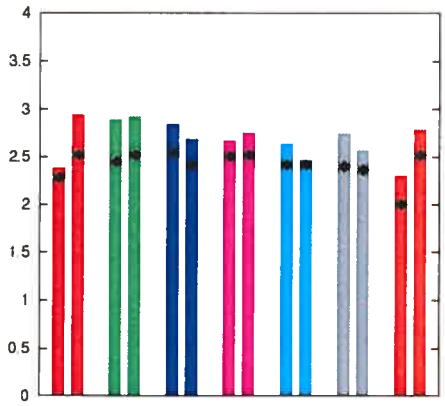


$k = 18$ (Pont Brownien)

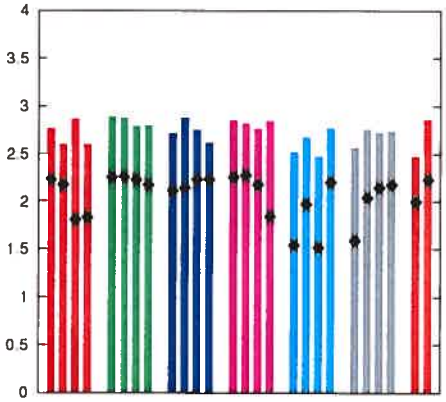
Tableau 6.33 – Réduction de variance pour la fonction delta, $S(0) = 100$.



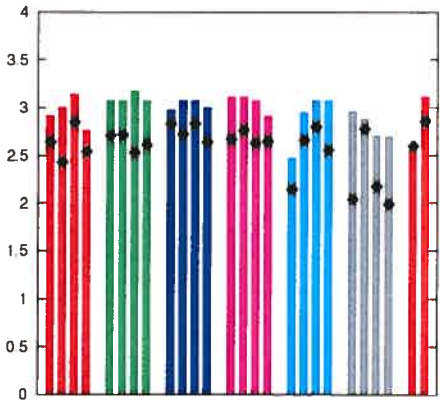
$k = 10$



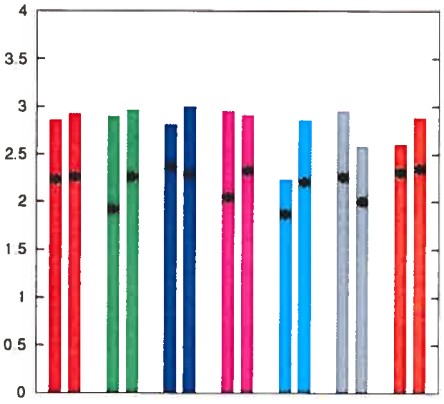
$k = 10$ (Pont Brownien)



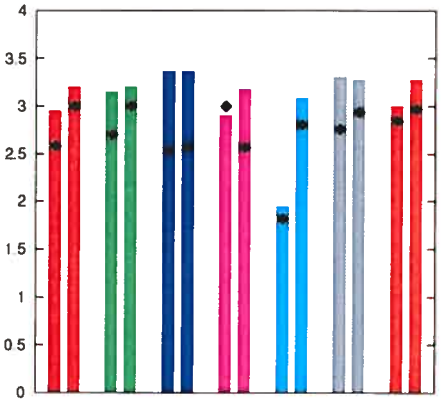
$k = 12$



$k = 12$ (Pont Brownien)

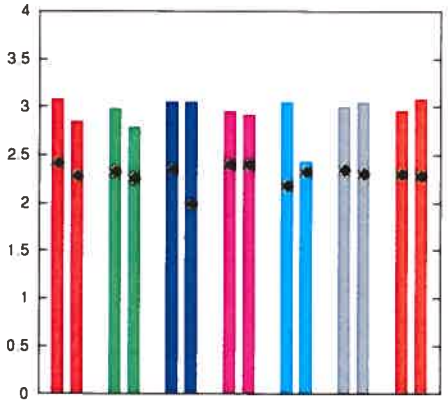


$k = 14$

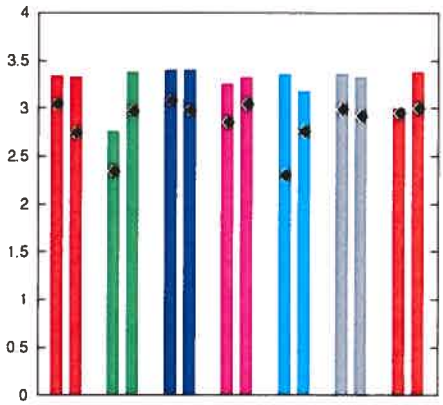


$k = 14$ (Pont Brownien)

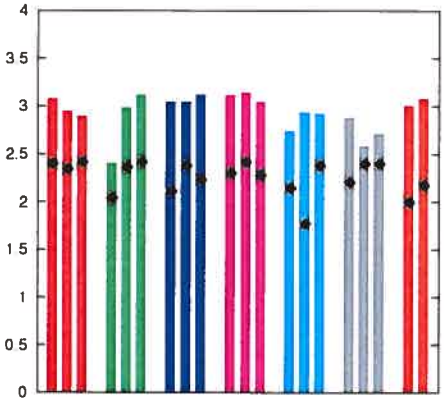
Tableau 6.34 – Réduction de variance pour la fonction delta, $S(0) = 100$.



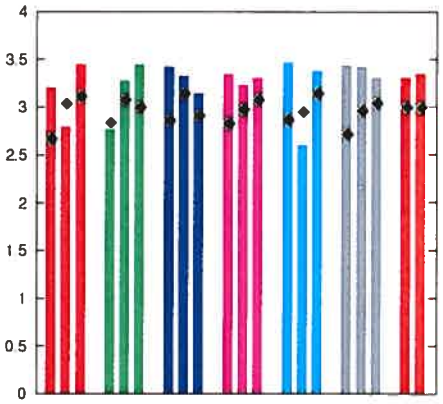
$k = 15$



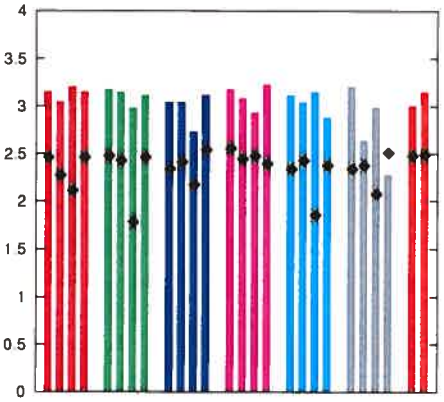
$k = 15$ (Pont Brownien)



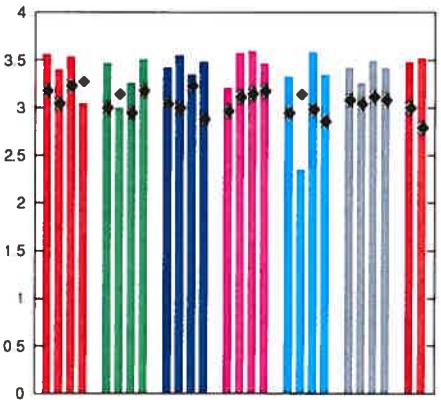
$k = 16$



$k = 16$ (Pont Brownien)

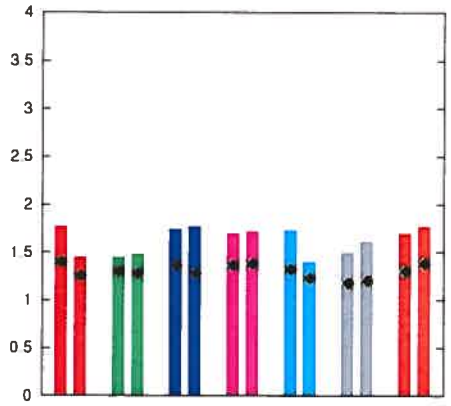


$k = 18$

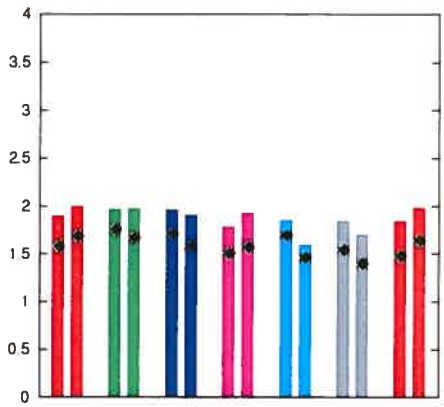


$k = 18$ (Pont Brownien)

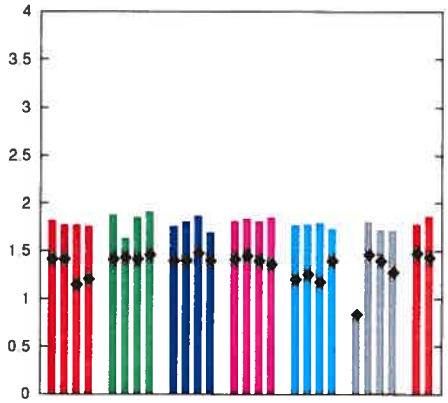
Tableau 6.35 – Réduction de variance pour la fonction delta, $S(0) = 110$.



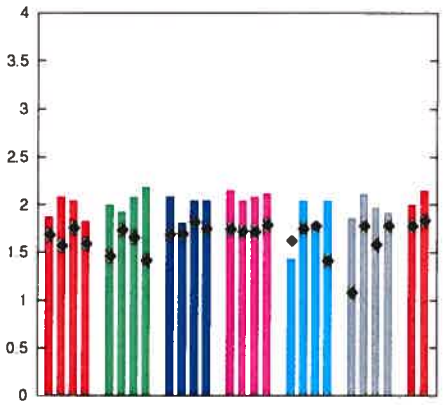
$k = 10$



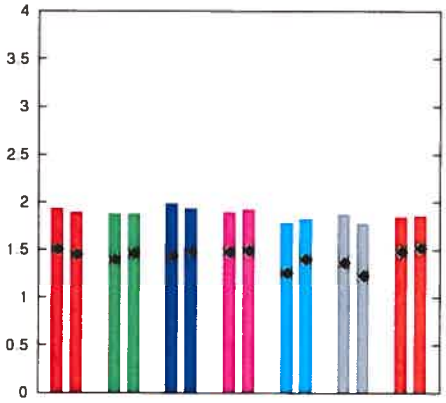
$k = 10$ (Pont Brownien)



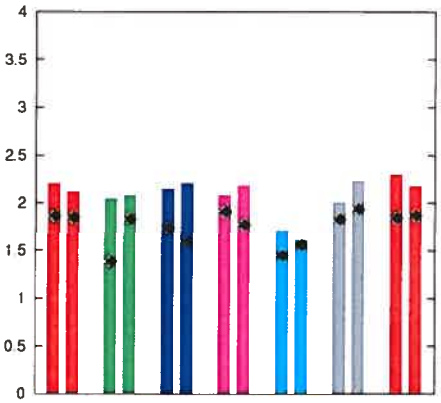
$k = 12$



$k = 12$ (Pont Brownien)

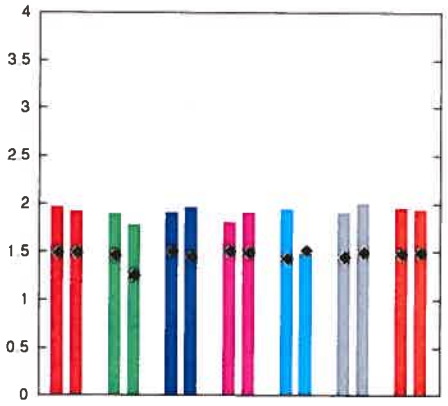


$k = 14$

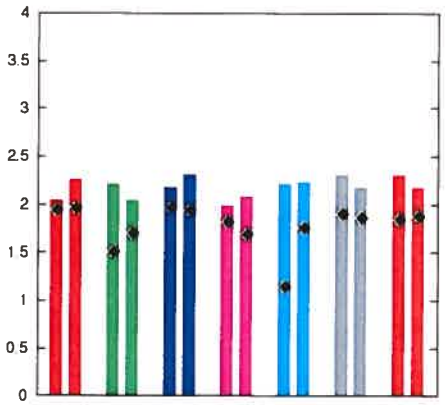


$k = 14$ (Pont Brownien)

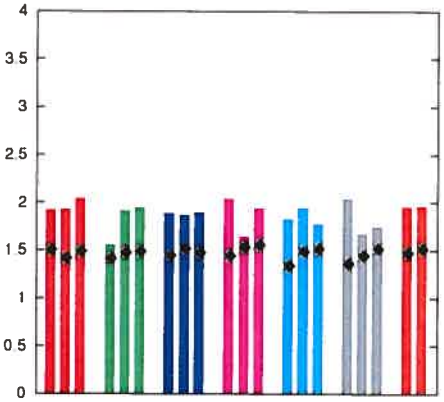
Tableau 6.36 – Réduction de variance pour la fonction delta, $S(0) = 110$.



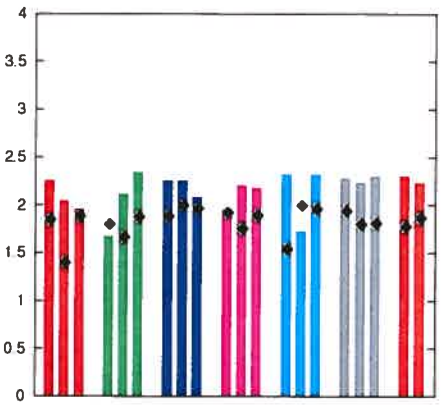
$k = 15$



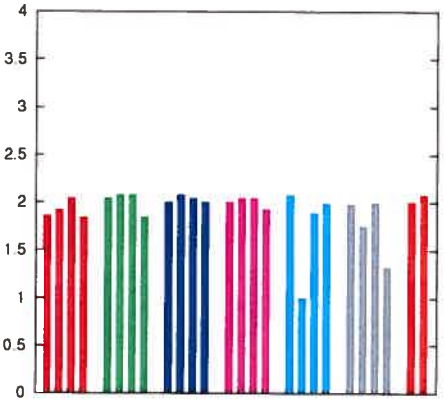
$k = 15$ (Pont Brownien)



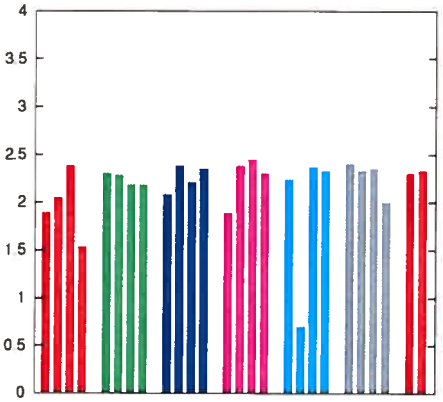
$k = 16$



$k = 16$ (Pont Brownien)



$k = 18$



$k = 18$ (Pont Brownien)

6.6 Interprétation des résultats

La première remarque à faire sur les résultats est que les problèmes sont, pour la plupart, en dimension 10 et 30 et que les ensembles de points ont été choisis avec des critères qui considèrent 24 dimensions. Les résultats sont tout de même assez convaincants sur la performance des ensembles de points basés sur une récurrence dans $\mathbb{F}_{2^w}$. Probablement que, si on avait choisi nos ensembles de points avec des critères taillés sur mesure pour les fonctions à intégrer, on aurait obtenu des résultats encore meilleurs.

En observant tous les tableaux de résultats, un point qui semble ressortir est qu'il n'y a pas un critère d'uniformité qui semble favoriser outrageusement une des fonctions à intégrer qu'on a utilisée. Un autre point qui ressort est que les ensembles de points qui sont choisis avec un critère qui dépend de la distance minimale performent très bien pour tous les problèmes qu'on a considérés. Parfois, c'est un ensemble de points qui est sélectionné avec un critère basé sur la distance minimale qui performe le mieux. On savait que la distance minimale est un critère important pour les ensembles de points utilisés en infographie [26]. Les résultats qu'on obtient confirment que la distance minimale peut être aussi un critère important pour d'autres types d'applications et ce, indépendemment de la q-valeur de l'ensemble de points considéré. Au lieu d'utiliser Γ_t ou la q-valeur, on peut construire des ensembles de points intéressants en ne regardant que la distance minimale pour plusieurs applications.

Dans ce qui suit, on fera références à l'ensemble de points de Sobol avec la translation digital comme « Sobol-1 » et l'ensemble de points de Sobol avec le mixage linéaire et la translation digital comme « Sobol-2 ».

Le tableau 6.8 montre que nos nouveaux ensembles de points performent mieux que les ensembles de points de Sobol, même randomisés, pour l'intégration de f_2. Les facteurs de réductions de variance vont de 10^9 à 10^{15} pour nos ensembles de

points et les ensembles de points de Sobol obtiennent, au mieux, 10^8. Au premier regard, on pourrait penser qu'il s'agit d'une erreur étant donnée la magnitude des facteurs de réductions de variance (c'est du moins ce que croyait l'auteur) et que f_2 n'est qu'une sommation de fonctions en une seule dimension. Mais après une étude préliminaire, il semblerait que les ensembles de points en une dimension (ou les projections en une dimension) construits par un LFSR dans $\mathbb{F}_{2^w}$ ont la particularité que des groupes de points sont localement antithétiques, c'est-à-dire que, pour un intervalle $[a2^{-v}, (a+1)2^{-v}]$, $0 \leq a < 2^v$, la moyenne des points qui tombent dans cet intervalle est égale à $(a + 1/2)2^{-v}$ pour $v = 0, 1 \ldots, v_0$ pour une valeur de v_0. Dans un article récent, Owen [80] montre comment obtenir des ensembles de points localement antithétiques *paire-à-paire* à l'aide d'un mixage linéaire approprié. Pour ses ensembles de points, dans un intervalle $[a2^{-v}, (a+1)2^{-v}]$, $0 \leq a < 2^v$, tous les points ont un « partenaire » tel que la moyenne des deux points est $(a + 1/2)2^{-v}$ pour $v = 0, \ldots, k-1$ où 2^k est la cardinalité de l'ensemble de points. Owen [80] montre que l'estimateur quasi-Monte Carlo qui utilise un ensemble de points localement antithétique a une variance dans $O(2^{-4k})$ pour des fonctions en une dimension. Nous croyons que ce résultat s'applique à nos ensembles de points et que ceci pourrait expliquer la très grande réduction de variance.

Étant donné que nous avons découvert cette propriété de nos ensemble de points dans la semaine précédant la soumission de cette thèse, nous n'avons pas pu analyser celle-ci plus en détail. Évidemment, nous allons analyser cet aspect intéressant une fois la thèse déposée.

Pour la fonction f_6, la dimensionalité de l'intégrale est vraiment de t (par dimensionalité, on signifie que l'intégrale de f_6 ne peut pas se décomposer en une somme d'intégrales de plus faible dimension). Dans ce cas, plus la dimension augmente, plus l'intégrale est difficile à intégrer par quasi-Monte Carlo. On observe ce phénomène car quand $t = 30$, il n'y a plus de gain à utiliser quasi-Monte Carlo : il n'y a prati-

quement pas de réduction de variance et même, quelquefois, on fait pire. Par contre, en dimension $t = 10$, on obtient une réduction de variance et plusieurs ensembles de points font mieux que Sobol. Quand $k = 18$, les facteurs de réduction de variance sont environ 100 pour plusieurs ensembles de points. Dans ce cas, on fait mieux que Sobol-1 et Sobol-2.

La fonction f_9 est une sommation de fonctions en deux dimensions. Chaque élément de la sommation utilise une projection en deux dimensions différentes. Par le théorème 5.6, on sait que la majorité des projections en deux dimensions sont parfaites jusqu'en résolution w. On doit donc s'attendre à ce que nos ensembles de points performent bien pour cette fonction. C'est bien le cas : quand $k = 18$, les facteurs de réduction de variance vont jusqu'à 10^{14} pour nos ensembles de points, tandis que les ensembles de points de Sobol, même randomisés avec du mixage linéaire, obtiennent des facteurs autour de 10^8. Pour toutes les valeurs de k, on remarque que la grande majorité de nos ensembles de points font mieux que les ensembles de points de Sobol.

Les fonctions f_{11} à f_{14} sont des fonctions dont la dimensionalité est t, mais où l'importance de chaque coordonnée, dans sa contribution pour la variance, décroît quand $a_0 < \ldots < a_{t-1}$ et reste la même quand $a_0 = \ldots = a_{t-1}$. Pour f_{11} et f_{12}, l'importance ne décroît pas avec l'indice de la coordonnée et pour f_{13} et f_{14}, l'importance décroît. La fonction f_{11} semble être très difficile à intégrer par quasi-Monte Carlo. En $t = 10$ dimensions, pour plusieurs de nos ensembles de points, on obtient des facteurs de réduction de variance qui sont supérieurs à ceux obtenus par les ensembles de points de Sobol. Pour $k = 18$, les facteurs vont jusqu'à 1000 pour nos ensembles de points, tandis que pour Sobol, les facteurs de réduction de variance ne dépassent pas 100. Par contre, pour $t = 30$, on n'arrive pas à faire mieux que la méthode Monte Carlo.

Pour la fonction f_{12}, quand $t = 10$, on réussit toujours à faire mieux avec nos nouveaux ensembles de points que ceux de Sobol. En particulier quand $k = 18$, les facteurs sont de 10^6 pour certains ensembles choisis par rapport à la distance

minimale. Pour Sobol, quand $k = 18$, le mieux qu'on ait pu faire est 10^4. Quand $t = 30$, nos nouveaux ensembles de points font, en moyenne, un peu mieux que les ensembles de points de Sobol. Les facteurs de réduction de variance vont de 1 à environ 10.

Pour les fonctions f_{13} et f_{14}, de très bons facteurs de réductions de variance sont obtenus pour $t = 10$ et $t = 30$. Pour f_{13}, quand $k = 18$, on obtient un facteur de réduction de variance de l'ordre de 10^9 pour un ensemble de points choisi avec un critère basé sur la distance minimale. Pour f_{13} et f_{14}, même avec peu de points, on a de bons facteurs. Par exemple, pour la fonction f_{14}, quand $t = 30$ et $k = 10$, on peut obtenir des facteurs de l'ordre de 10^6 avec des ensembles de points choisis avec un critère basé sur l'équidistribution. Dans cet exemple, Sobol-1 et Sobol-2 obtiennent des facteurs de 10^5 et 10^4, respectivement.

La fonction f_{15} est une sommation de fonctions dont la dimensionalité est m. Pour celle-ci, la dimension de l'intégrale est $t = nm = 100$. Nous avons conçu cette fonction pour observer l'efficacité de nos nouveaux ensembles de points quand la fonction à intégrer est sensible à l'uniformité de projections de faibles dimensions successives. Quand $m = 2$, les résultats obtenus par nos ensembles de points sont assez impressionnants. Quand $k = 10$, les facteurs de réduction de variance sont de l'ordre de 10^{15} avec nos nouveaux ensembles de points. Pour les ensembles de points de Sobol, quand $k = 10$, les facteurs obtenus sont de l'ordre de 10^3. Ce résultat est explicable par le fait que les ensembles de points de Sobol sont reconnus pour avoir certaines projections en deux dimensions qui ne sont pas très uniformes et aussi à cause du fait que nos ensembles de points sont stationnaires dans la dimension et que la majorité des projections en deux dimensions sont parfaitement équidistribués jusqu'au w-ième bit par le corollaire 5.6. Pour $m = 5$, nos nouveaux ensembles de points obtiennent encore de très bons résultats. Ils font, pour la plupart, mieux que les ensembles de points de Sobol. Malgré que les facteurs de réduction de variance

sont moins bons que dans le cas où $m = 2$, on obtient tout de même, quand $k = 18$, des facteurs de l'ordre de 10^{12}.

Pour $m = 20$ et $m = 50$, les fonctions sont très difficiles à intégrer. Les résultats démontrent, en général, que nos nouveaux ensembles de points ne réussissent pas à faire vraiment mieux que par la méthode Monte Carlo, tout comme les ensembles de points Sobol-1 et Sobol-2.

Dans les tableaux 6.19 à 6.30, on expose les résultats obtenus pour les fonctions qui sont reliées au problème de l'évaluation d'une option asiatique. Pour simuler le mouvement Brownien, nous avons utilisé deux méthodes : la méthode standard et le pont Brownien. Le pont Brownien fait en sorte que les coordonnées les plus importantes dans l'évolution du mouvement Brownien sont les premières. Habituellement, cette technique permet d'améliorer le facteur de réduction de variance [7]. Dans les tableaux, on remarque cette tendance : les facteurs de réduction de variances sont plus élevés quand le mouvement Brownien est produit par un pont Brownien.

Pour le problème de l'évaluation du prix d'une option asiatique, nos nouveaux ensembles de points obtiennent des facteurs de réduction de variance, quand $K = 90$, pouvant aller jusqu'à 10^5 quand $t = 10$ et $10^{4.5}$ quand $t = 30$. On remarque que nos nouveaux ensembles de points obtiennent des résultats semblables aux ensembles de points de Sobol pour ce problème. On remarque également que la réduction de variance est moindre quand $t = 30$ comparativement au cas où $t = 10$.

Pour les problème de l'évaluation de delta et de vega, nos nouveaux ensembles de points performent aussi bien que les ensembles de points de Sobol. Pour vega, les facteurs de réduction de variance vont de $10^{1/2}$ à $10^{7/8}$. Pour delta, on obtient de meilleurs facteurs : ils vont de 10 à $10^{3.5}$.

6.7 Conclusions

Dans la section précédente, nous avons présenté les résultats de certaines expériences qui montrent empiriquement la bonne qualité des ensembles de points qui sont donnés dans les tableaux 6.1 à 6.6. Les qualités principales de ces ensembles de points sont qu'ils ont une dimension infinie, qu'ils sont stationnaires dans la dimension, que la plupart des projections en deux dimensions sont parfaitement équidistribuées jusqu'au w-ième bit et qu'il semble que les projections en une dimension donnent des ensembles de points qui sont localement antithétiques.

Les expériences nous ont montré que, dans la plupart des fonctions que nous avons utilisées, nos nouveaux ensembles de points font mieux que les ensembles de points de Sobol, même ceux qui sont randomisés avec du mixage linéaire.

Les résultats démontrent aussi l'utilité de critères basés sur la distance minimale, puisque les ensembles de points choisis selon ces critères performent aussi bien que ceux choisis selon des critères basés sur la **p**-équidistribution. À ce moment-ci, les raisons qui expliquent cela restent à étudier. On pourrait définir des critères, basés sur la distance minimale, qui permettent de garantir certaines bornes sur l'erreur d'intégration de certaines classes de fonctions. Pour des applications en optimisation, il serait également intéressant d'étudier les liens qui pourraient exister entre la distance minimale et la dispersion (voir chapitre 6 de [74]) quand l'ensemble de points est un réseau digital.

Chapitre 7

Générateurs à opérations binaires

Pour la plupart des générateurs courants, on utilise une récurrence simple sur un corps fini à partir de laquelle est construit un générateur de nombres aléatoires. Souvent, la récurrence choisie n'est pas bien adaptée à une implantation sur ordinateur. Pour certains types de générateurs, il faut utiliser des astuces afin de réussir à obtenir une implantation efficace du point de vue de la rapidité. Par exemple, au chapitre 5, deux techniques d'implantations ont été présentées pour les générateurs basés sur une récurrence linéaire dans $\mathbb{F}_{2^w}$. Pour d'autres types de générateurs, on doit se restreindre dans le choix des paramètres pour que l'implantation soit rapide. Par exemple, pour les générateurs à récurrences multiples (ou « multiple recursive generator »), on doit faire des choix sur certains coefficients pour que l'implantation soit rapide [50]. Ceci fait en sorte qu'on s'impose des restrictions quant au choix des paramètres qui sont multiples. On doit choisir les paramètres de manière à ce que

- le générateur ait la période voulue,
- l'implantation soit efficace du point de vue de la mémoire et de la vitesse, et
- l'uniformité des points produits soit excellente.

Il semble que répondre à ces trois critères soit un but difficile à atteindre si on se limite à une récurrence donnée dans un corps fini. Par exemple, pour les générateurs intro-

duits au chapitre 5, on construit des générateurs basés sur des récurrences linéaires sur $\mathbb{F}_{2^w}$. Pour ceux-ci, l'implantation n'est pas aussi efficace du point de vue de la mémoire qu'un TGFSR ou du Mersenne Twister. Par contre, l'équidistribution est meilleure, à longueur de période similaire.

Dans ce chapitre, l'approche est d'utiliser une récurrence assez générale, qui tient compte de la manière dont l'ordinateur manipule l'information, afin de définir une nouvelle famille de générateurs. De cette manière, on s'assure de bien faire au niveau de l'efficacité. Cette récurrence générale combine, d'une certaine manière, l'implantation d'un GCL polynomial et d'un LFSR. Au lieu de n'observer qu'un seul bloc de 32 bits et d'en modifier plusieurs ou d'observer plusieurs blocs et d'en modifier qu'un, on propose une récurrence dans laquelle on observe six blocs pour en modifier deux.

Ce nouveau type de générateur utilise une récurrence linéaire sur $\mathbb{F}_2^k$, l'espace des vecteurs en k dimensions d'éléments dans $\mathbb{F}_2$. La récurrence est basée sur des opérations linéaires sur $\mathbb{F}_2^k$ qui peuvent s'effectuer rapidement sur un ordinateur. Un exemple d'une opération permise est l'addition de vecteurs dans $\mathbb{F}_{2^k}$. Celle-ci est implantée efficacement par un ou-exclusif bit à bit et est habituellement effectuée par blocs de 32 bits. Nous nommons ce nouveau type de générateur WELLRNG pour "**W**ell **E**quidistributed **L**ong-Period **L**inear **R**andom **N**umber **G**enerator". Le nom décrit bien ces générateurs puisque les ensembles de points produits par ces générateurs sont bien équidistribués, ils ont une longue période (même extrêmement longue dans certains cas) et ils utilisent une récurrence linéaire. Cet acronyme nous permettra d'identifier ce type de générateur sans ambiguïté.

Les qualités principales de ces nouveaux générateurs, en plus de leur rapidité et de leur longue période, sont le nombre élevé de coefficients non nuls dans le polynôme caractéristique de la matrice de transition et la très bonne équidistribution qui peut être obtenue. Le grand nombre de coefficients est une indication du niveau élevé de « brassage » de la matrice de transition. Le polynôme caractéristique indiquant la

relation linéaire liant chacun des $\mathbf{x}_n$, plus il y a de coefficients non nuls, plus la relation linéaire est compliquée et plus le nouvel état est « différent » par rapport aux états précédents.

La récurrence sur laquelle se base ce nouveau type de générateur entre dans le cadre des équations (1.1)-(1.4). La matrice X est composée de sous-matrices de dimension 32×32. Ce choix de dimension pour ces matrices est fondé sur le fait que la plupart des ordinateurs courants traitent efficacement les opérations sur des vecteurs de 32 bits.

Nous donnons des paramètres de générateurs qui ont des périodes qui vont jusqu'à $2^{44497} - 1$ et qui ont une excellente uniformité. Le développement de ces générateurs n'aurait jamais été possible sans l'implantation, dans REGPOLY, des algorithmes décrits aux sections 2.5 et 2.6 qui permettent de déterminer la primitivité du polynôme caractéristique de la matrice de transition X, même quand la dimension de X est énorme. De plus, sans une implantation efficace, dans REGPOLY, de l'algorithme de réduction d'une base d'un réseau polynômial, il aurait été très difficile d'analyser l'équidistribution de ces nouveaux générateurs. Beaucoup d'heures de programmation ont été investies afin d'optimiser les implantations et d'obtenir les générateurs que l'on introduit dans ce chapitre.

Dans ce qui suit, on définit la matrice de transition X utilisée par les nouveaux générateurs. Également, on introduit un algorithme de recherche qui permet de trouver des générateurs de pleine période. Grâce à cet algorithme, on trouve des générateurs qui ont une période maximale, qui possèdent une implantation efficace et une excellente équidistribution. De plus, on donne une implantation en langage C d'un générateur qui a une période de $2^{1024} - 1$ et une autre d'un autre générateur de période $2^{44497} - 1$. Finalement, les résultats à des tests statistiques et des tests de vitesse d'exécution sont présentés pour quelques générateurs.

7.1 Description des générateurs

L'état d'un WELLRNG est un vecteur de rw bits

$$\mathbf{x}_n = (\mathbf{v}_{n,0}^{\mathsf{T}}, \ldots, \mathbf{v}_{n,r-1}^{\mathsf{T}})^{\mathsf{T}}$$

où $n \geq 0$, $\mathbf{v}_{n,i} \in \mathbb{F}_2^w$, $0 \leq i < r$ et r est un entier positif. Les vecteurs $\mathbf{v}_{n,i}$ sont vus comme des vecteurs colonnes. La récurrence de ce générateur est

$$\mathbf{x}_n = X\mathbf{x}_{n-1}$$

où la multiplication par la matrice X est effectuée par l'algorithme 7.1. L'idée de l'algorithme est d'obtenir une matrice de transition X relativement compliquée avec un petit nombres d'opérations. Pour ce faire, l'algorithme utilise 6 blocs de w bits de l'état $\mathbf{x}_n$ et les combine pour obtenir les vecteurs de bits $\mathbf{z}_0$, $\mathbf{z}_1$ et $\mathbf{z}_2$. Ensuite, on recombine ces vecteurs pour obtenir les vecteurs $\mathbf{z}_3$ et $\mathbf{z}_4$. Ces deux derniers vecteurs sont utilisés pour mettre à jour l'état du générateur. Grâce à cette stratégie, on espère que la matrice X « mélange » assez bien les bits d'une itération à l'autre pour obtenir de bons générateurs.

Dans l'algorithme 7.1, les matrices T_i sont des matrices $w \times w$ et la matrice S, de dimension $w \times 2w$, est définie par

$$S = \begin{pmatrix} U_p & L_{w-p} \end{pmatrix}, \quad U_p = \begin{pmatrix} 0 & 0 \\ 0 & I_p \end{pmatrix}, \quad L_{w-p} = \begin{pmatrix} I_{w-p} & 0 \\ 0 & 0 \end{pmatrix},$$

où p est un entier positif, U_p et L_{w-p} sont des matrices de dimension $w \times w$ et I_p est une matrice identité de dimension $p \times p$. Les valeurs m_i sont sont des entiers positifs tels que $1 \leq m_i \leq r-2$, $1 \leq i \leq 3$.

Algorithme 7.1. Une itération d'un générateur WELLRNG.

$$
\begin{aligned}
\mathbf{z}_0 &\leftarrow S(\mathbf{v}_{n,r-2}^{\mathsf{T}}, \mathbf{v}_{n,r-1}^{\mathsf{T}})^{\mathsf{T}} \\
\mathbf{z}_1 &\leftarrow T_0\mathbf{v}_{n,0} \oplus T_1\mathbf{v}_{n,m_1} \\
\mathbf{z}_2 &\leftarrow T_2\mathbf{v}_{n,m_2} \oplus T_3\mathbf{v}_{n,m_3} \\
\mathbf{z}_3 &\leftarrow \mathbf{z}_1 \oplus \mathbf{z}_2 \\
\mathbf{z}_4 &\leftarrow T_4\mathbf{z}_0 \oplus T_5\mathbf{z}_1 \oplus T_6\mathbf{z}_2 \oplus T_7\mathbf{z}_3 \\
\mathbf{v}_{n+1,r-1} &\leftarrow \mathbf{v}_{n,r-2} \,\&\, \mathbf{m}_p \\
&\text{pour } j = r-2, \ldots, 2 \\
\quad \mathbf{v}_{n+1,j} &\leftarrow \mathbf{v}_{n,j-1} \\
\mathbf{v}_{n+1,1} &\leftarrow \mathbf{z}_3 \\
\mathbf{v}_{n+1,0} &\leftarrow \mathbf{z}_4
\end{aligned}
$$

On peut déduire la matrice de transition X en examinant l'algorithme 7.1. Le vecteur $\mathbf{v}_{n+1,0}$ est obtenu par

$$
\begin{aligned}
\mathbf{v}_{n+1,0} &= T_4\mathbf{z}_0 \oplus T_5\mathbf{z}_1 \oplus T_6\mathbf{z}_2 \oplus T_7\mathbf{z}_3 \\
&= T_4S(\mathbf{v}_{n,r-2}^{\mathsf{T}}, \mathbf{v}_{n,r-1}^{\mathsf{T}})^{\mathsf{T}} \oplus T_5(T_0\mathbf{v}_{n,0} \oplus T_1\mathbf{v}_{n,m_1}) \oplus T_6(T_2\mathbf{v}_{n,m_2} \oplus T_3\mathbf{v}_{n,m_3}) \\
&\quad \oplus T_7(T_0\mathbf{v}_{n,0} \oplus T_1\mathbf{v}_{n,m_1} \oplus T_2\mathbf{v}_{n,m_2} \oplus T_3\mathbf{v}_{n,m_3}) \\
&= T_4U_p\mathbf{v}_{n,r-2} \oplus T_4L_{w-p}\mathbf{v}_{n,r-1} \oplus (T_5T_0 \oplus T_7T_0)\mathbf{v}_{n,0} \oplus (T_5T_1 \oplus T_7T_1)\mathbf{v}_{n,m_1} \\
&\quad \oplus (T_6T_2 \oplus T_7T_2)\mathbf{v}_{n,m_2} \oplus (T_6T_3 \oplus T_7T_3)\mathbf{v}_{n,m_3}
\end{aligned}
$$

et le vecteur $\mathbf{v}_{n+1,1}$ est obtenu par

$$
\mathbf{v}_{n+1,1} = T_0\mathbf{v}_{n,0} \oplus T_1\mathbf{v}_{n,m_1} \oplus T_2\mathbf{v}_{n,m_2} \oplus T_3\mathbf{v}_{n,m_3}.
$$

Les vecteurs $\mathbf{v}_{n+1,j}$ pour $j = 2, \ldots, r-1$ sont obtenus par $\mathbf{v}_{n+1,j} = \mathbf{v}_{n,j-1}$. Grâce à

ces équations, on en déduit la matrice de transition X, de dimension $rw \times rw$, qui est

$$\begin{pmatrix}
T_{5,7,0} & 0 & 0 & \dots & 0 & T_{5,7,1} & 0 & \dots & 0 & T_{6,7,2} & 0 & \dots & 0 & T_{6,7,3} & 0 & \dots & 0 & T_4U_p & T_4L_{w-p} \\
T_0 & 0 & 0 & \dots & 0 & T_1 & 0 & \dots & 0 & T_2 & 0 & \dots & 0 & T_3 & 0 & \dots & 0 & 0 & 0 \\
0 & I & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
0 & 0 & I & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
 & & & & & & & \ddots & & & & & & & & & & & \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & I & 0 & 0 \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & I & 0
\end{pmatrix}$$

où $T_{i,j,k} = T_iT_k \oplus T_jT_k$. Les matrices T_1, T_2 et T_3 se trouvent dans les m_1-ième, m_2-ième et m_3-ième colonnes de blocs de matrices. On remarque que cette matrice n'est pas de plein rang rw si $p > 0$ car les p dernières colonnes de X sont nulles. Par contre, nous allons donner une condition nécessaire et suffisante pour que la matrice X soit de rang $rw - p$. Soit $X^{(i)}$, la i-ième ligne de la matrice X. En additionnant à $X^{(1)}$ le vecteur

$$T_{5,7}X^{(2)} \oplus (T_{6,7,2} + T_{5,7,2})X^{(m_2+2)} \oplus (T_{6,7,3} + T_{5,7,3})X^{(m_3+2)} \oplus T_4U_pX^{(r-2)}$$

et en plaçant le vecteur $X^{(1)}$ à la dernière ligne (en décalant toutes les autres lignes d'une position vers le haut) on obtient la matrice

$$\begin{pmatrix}
T_0 & 0 & 0 & \dots & 0 & T_1 & 0 & \dots & 0 & T_2 & 0 & \dots & 0 & T_3 & 0 & \dots & 0 & 0 & 0 \\
0 & I & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
0 & 0 & I & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\
 & & & & & & & \ddots & & & & & & & & & & & \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & I & 0 & 0 \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & I & 0 \\
0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & T_4L_{w-p}
\end{pmatrix}.$$

Cette transformation conserve le rang de la matrice X et nous permet de déduire une condition nécessaire et suffisante pour que celle-ci soit de rang $wr - p$.

Proposition 7.1. *X est de plein rang si et seulement si T_0 est de plein rang w et les $w - p$ premières lignes de T_4 sont linéairement indépendantes.*

Pour choisir nos matrices T_0 et T_4, au lieu d'utiliser cette proposition, nous allons plutôt utiliser le corollaire suivant qui donne une condition suffisante plus pratique.

Corollaire 7.1. *Si les matrices T_0 et T_4 sont de plein rang, alors la matrice de transition X d'un WELLRNG est de rang $wr - p$.*

Le fait que le rang de la matrice soit $wr - p$ peut sembler un désavantage, puisqu'on ne pourra pas jamais atteindre la période de $2^{wr} - 1$. Bien que cela soit vrai, l'avantage qu'on y gagne est dans la détermination de la période du générateur. Pour montrer qu'un générateur est de période $2^k - 1$, il faut montrer que le polynôme caractéristique de X est de degré k et qu'il est primitif. Dans le test de primitivité présenté à l'algorithme 2.2, il faut connaître la factorisation de $2^k - 1$. Quand k est grand, factoriser $2^k - 1$ est un problème très difficile. Bien qu'il existe des tables de ces factorisations (comme celles du projet Cunningham dont il est question à la section 2.6), celles-ci ne sont pas complètes et pour $k > 500$, il y a beaucoup de factorisations manquantes. Par contre, si $2^k - 1$ est un nombre premier de Mersenne, alors la factorisation est connue et cela simplifie énormément l'algorithme 2.2 puisqu'il suffit de vérifier si $z^{2^k} \equiv z \bmod P(z)$. De plus, si $2^k - 1$ est un nombre premier de Mersenne, alors un polynôme de degré k est primitif si et seulement s'il est irréductible. Tester l'irréductibilité d'un polynôme est moins long à vérifier que d'effectuer $z^{2^k} \bmod P(z)$ quand k est grand. Si le polynôme caractéristique de la matrice X est de degré rw, alors il est impossible que $2^{rw} - 1$ soit un nombre premier de Mersenne. En effet, il est facile de vérifier que

$$2^{rw} - 1 = (2^r - 1)(2^{r(w-1)} + 2^{r(w-2)} + \cdots + 2^r + 1)$$

Le Mersenne twister [69] a été conçu pour obtenir des générateurs de très longue période. En ayant une matrice de transition dont le degré du polynôme caractéristique est un exposant de Mersenne, les auteurs ont été capables de trouver des générateurs possédant de très longues périodes. Par contre, dans [69], pour déterminer la primitivité du polynôme caractéristique du Mersenne twister, les auteurs n'ont pas fait

appel à un test d'irréductibilité, ni au calcul direct de $z^{2^k} \bmod P(z)$, mais à une technique appelée *décimation inverse*. Pour un entier j, $0 \leq j < k$, s'il est possible de déterminer $\mathbf{x}_0$ à partir de $(x_0^{(j)}, x_1^{(j)}, \ldots, x_{k-1}^{(j)})$ en $O(k)$ opérations, alors la décimation inverse devient avantageuse par rapport aux méthodes plus classiques. Dans le cas des WELLRNG, cette technique ne semble pas être applicable, étant donné la complexité de la matrice X. La récurrence simple du Mersenne twister permet de déterminer $\mathbf{x}_0$ à partir de $(x_0^{(j)}, \ldots, x_{k-1}^{(j)})$ en $O(k)$ opérations.

Les auteurs de [69] semblaient pessimistes sur les chances de trouver des générateurs de périodes très longues sans la méthode de la décimation inverse (section 4.1) :

> In our case, k is very large (> 10000), and according to our experiment, the direct computation may need several years to catch a primitive polynomial.

Les auteurs, lorsqu'ils font référence à « direct computation », ils signifient la vérification directe de $z^{2^k} \equiv z \bmod P(z)$. Par contre, dans cette affirmation, on ne sait pas s'ils font référence au polynôme caractéristique d'un Mersenne twister ou de n'importe quel autre générateur qui entre dans le cadre des équations (1.1)-(1.4). Malgré tout, nous avons pu trouver des générateurs, sans la décimation inverse, mais plutôt grâce aux algorithmes, qu'on pourrait qualifier de « plus classiques », mais améliorés, qui sont présentés aux sections 2.5 et 2.6.

Il ne reste plus qu'à déterminer la sortie pour les WELLRNG. On considère la possibilité de prendre $\mathbf{y}_n = \mathbf{v}_{n,j}$ pour une valeur de j donnée telle que $j < r$. On peut tout de suite éliminer la possibilité $j = r - 1$, puisqu'on désire avoir une sortie qui a w bits de résolution. On remarque que, à chaque itération, seuls les vecteurs $\mathbf{v}_{n,0}$ et $\mathbf{v}_{n,1}$ sont mis à jour avec de « nouvelles valeurs ». Les autres vecteurs, $\mathbf{v}_{n,j}$, pour $j = 2, \ldots, r-2$, sont des copies de $\mathbf{v}_{n-1,j-1}$. Ainsi, étant donné une initialisation $\mathbf{x}_0$, la séquence $\{\mathbf{v}_{n,1}\}_{n\geq 0}$ est la même que $\{\mathbf{v}_{n+j-1,j}\}_{n\geq 0}$, pour $j = 2, \ldots, r-2$. Ceci implique que choisir $\mathbf{y}_n = \mathbf{v}_{n,j}$ où $j > 1$ est équivalent à choisir $\mathbf{y}_n = \mathbf{v}_{n,1}$. On peut

montrer que choisir $\mathbf{y}_n = \mathbf{v}_{n,1}$ n'est pas équivalent à choisir $\mathbf{y}_n = \mathbf{v}_{n,0}$ de manière expérimentale. En effet, les deux sorties, pour une même matrice X, ne donnent pas les mêmes propriétés d'équidistribution. Par conséquent, les seules valeurs de j intéressantes pour construire la sortie sont $j = 0$ et $j = 1$. Cette valeur de j est un des paramètres qui définit un WELLRNG.

7.2 Sous-matrices de X

Les matrices $T_0, \ldots, T_7$ n'ont pas été définies jusqu'à présent. On a seulement affirmé qu'une condition suffisante pour que X soit de rang $wr - p$ est que les matrices T_0 et T_4 doivent être de plein rang w. Dans cette section, on définit des transformations linéaires simples qui peuvent être représentées par une matrice. Pour définir un WELLRNG, on assigne à chacune des matrices $T_0, \ldots, T_7$ une de ces transformations linéaires. Les matrices admises sont décrites au tableau 7.1.

Les matrices $M_0(t)$, M_1 et $M_4(t, \mathbf{b})$ sont de rang w, à condition que le paramètre t ne soit pas nul. Les matrices M_7 et $M_3(t)$ où $t \neq 0$ ne sont jamais de plein rang. Pour $M_3(t)$, si $t = 0$, alors $M_3(t)$ est M_1, la matrice identité. Dans ce cas, $M_3(t)$ est de plein rang. Également, pour que $M_2(\mathbf{a})$ soit de plein rang, il faut que $a^{(0)} = 1$. La matrice $M_5(r, s, t, \mathbf{a})$ provient d'un type de tempering que l'on peut appliquer efficacement à un GCL polynômial. Voir [45] pour plus de détails sur cette matrice.

Le but de ces matrices est d'effectuer une transformation linéaire à un vecteur de w bits $\mathbf{x}$ rapidement sur un ordinateur. On remarque que, pour chacune des matrices, la transformation linéaire ne comporte pas le même nombre d'opérations. Par exemple, on pourrait s'attendre à ce que multiplier $\mathbf{x}$ par M_1 est plus rapide que le multiplier par $M_5(5, 5, 5, \mathbf{a})$. La multiplication par M_1 ne comporte qu'une seule affectation tandis que la multiplication par $M_5(5, 5, 5, \mathbf{a})$ nécessite une affectation, une comparaison, 2 ou-exclusifs bit à bit, un masque de bits et 2 décalages de bits. En comptant le nombre

Tableau 7.1 – Sous-matrices admises pour X.

Matrice M_i	Transformation linéaire $\mathbf{y} = M_i\mathbf{x}$
$M_0(t)$ $-w \leq t \leq w$	$\mathbf{y} = \begin{cases} \mathbf{x} \oplus (\mathbf{x} \gg t) & \text{si } t \geq 0 \\ \mathbf{x} \oplus (\mathbf{x} \ll -t) & \text{sinon} \end{cases}$
M_1	$\mathbf{y} = \mathbf{x}$
$M_2(\mathbf{a})$ $\mathbf{a} \in \mathbb{F}_2^w$	$\mathbf{y} = \begin{cases} (\mathbf{x} \gg 1) \oplus \mathbf{a} & \text{si } x^{(w-1)} = 1 \\ (\mathbf{x} \gg 1) & \text{sinon} \end{cases}$
$M_3(t)$ $-w \leq t \leq w$	$\mathbf{y} = \begin{cases} (\mathbf{x} \gg t) & \text{si } t \geq 0 \\ (\mathbf{x} \ll -t) & \text{sinon} \end{cases}$
$M_4(t, \mathbf{b})$ $-w \leq t \leq w$ $\mathbf{b} \in \mathbb{F}_2^w$	$\mathbf{y} = \begin{cases} \mathbf{x} \oplus ((\mathbf{x} \ll t)\&\mathbf{b}) & \text{si } t \geq 0 \\ \mathbf{x} \oplus ((\mathbf{x} \gg -t)\&\mathbf{b}) & \text{sinon} \end{cases}$
$M_5(r, s, t, \mathbf{a})$ $0 \leq r, s, t < w$	$\mathbf{y} = \begin{cases} (((\mathbf{x} \ll r) \oplus (\mathbf{x} \gg (w-r)))\&\mathbf{d}_s) \oplus \mathbf{a} & \text{si } x^{(t)} = 1 \\ (((\mathbf{x} \ll r) \oplus (\mathbf{x} \gg (w-r)))\&\mathbf{d}_s) & \text{sinon} \end{cases}$ Remarque : $\mathbf{d}_s \in \mathbb{F}_2^w$ est composé de uns sauf au bit $d^{(s)} = 0$.
M_7	$\mathbf{y} = 0$

d'opérations nécessaires pour effectuer la multiplication par chacune des matrices, on arrive à un système de pointage où est attribué un certain nombre de points à chacune des matrices. Le système utilisé est très simple : on donne 1 point pour chaque assignation, 1 point pour chaque ou-exclusif, 1 point pour un masque et 1 point pour chaque décalage. On donne 2 points pour une comparaison puisqu'elle est habituellement coûteuse en temps d'exécution. Le tableau 7.2 donne les points obtenus pour chacune des matrices. On note par $c(T)$, le pointage de la matrice T. On n'attribue aucun point à M_7 puisque le résultat de la multiplication est toujours zéro et que celui-ci est toujours additionné à un autre vecteur $\mathbf{x}$. Additionner zéro à $\mathbf{x}$ ne change pas $\mathbf{x}$, donc omettre la multiplication par M_7 revient au même, d'où le pointage de zéro à M_7. Soit X, la matrice définie par les sous-matrices $T_0, \ldots, T_7$. On définit une fonction de coût associée à la matrice X. Celle-ci est

$$C(X) = \sum_{i=0}^{7} c(T_i).$$

En choisissant des matrices $T_0, \ldots, T_7$ qui ont de faibles pointages, on espère que l'implantation résultante sera rapide. Considérons deux matrices de transition X_1 et X_2. On émet l'hypothèse que l'implantation du générateur basé sur la matrice de transition X_1 est plus rapide que celle de celui basé sur X_2 si et seulement si $C(X_1) < C(X_2)$. Évidemment, cette approche est un peu naïve et il s'agit d'une heuristique. Le système de pointage n'est pas parfait. Le compilateur peut optimiser dans certaines situations et dans d'autres pas, pour plusieurs raisons. La vitesse de l'implantation dépend aussi d'autres facteurs tels la machine utilisée, le langage de programmation, l'habilité du programmeur et aussi le compilateur utilisé.

Tableau 7.2 – Pointage attribué à chaque matrice.

Matrice	$M_0(t)$	M_1	$M_2(\mathbf{a})$	$M_3(t)$	$M_4(t, \mathbf{b})$	$M_5(r, s, t, \mathbf{a})$	M_7
$c(M_i)$	3	1	5	2	4	8	0

Dans la prochaine section, on présente l'algorithme qui permet de choisir les sous-matrices T_i parmi les matrices présentées dans cette section afin d'obtenir un générateur qui est de période $2^{wr-p}-1$, la période maximale pour ce type de générateur.

7.3 Algorithme de recherche de bons générateurs

Nous présentons maintenant l'algorithme de recherche de bons générateurs. En premier lieu, l'algorithme tente d'assigner aux matrices $T_0, \ldots, T_7$ une de celles présentées à la section 7.2. En choisissant judicieusement les matrices $T_0, \ldots, T_7$, on s'assure que la matrice X résultante soit de rang $wr-p$. L'algorithme 7.2 effectue la recherche de générateurs qui ont la période maximale et qui sont tels que $C(X) < C_{max}$.

Algorithme 7.2. Recherche d'un générateur de période $2^{wr-p}-1$ tel que $C(X) < C_{max}$

1. Répéter

   Pour $i \in \{0, 4\}$

   Assigner à T_i une des matrices suivantes :

   $M_0(t)$, M_1, $M_2(\mathbf{a})$, $M_4(t, \mathbf{b})$.

   Choisir les paramètres de T_i pour qu'elle soit de plein rang.

   Pour $i \in \{1, 2, 3, 5, 6, 7\}$

   Assigner à T_i une des matrices du tableau 7.1

   Choisir les paramètres de T_i

   Si $C(X) = \sum_{i=0}^{7} c(T_i) < C_{max}$, alors

   Déterminer si le le polynôme caractéristique $P(z)$ de X

   est primitif (voir section 2.5).

   Jusqu'à ce que $P(z)$ soit primitif.

2. La matrice X définit un générateur de période $2^{wr-p} - 1$.

Une fois que l'algorithme 7.2 a trouvé un générateur de période $2^{wr-p} - 1$, il ne reste qu'à vérifier son équidistribution.

7.4 Recherche de bons paramètres

Nous avons cherché des générateurs WELLRNG dont les périodes sont $2^{512} - 1$, $2^{521} - 1$, $2^{607} - 1$, $2^{1024} - 1$, $2^{19937} - 1$, $2^{21701} - 1$, $2^{23209} - 1$ et $2^{44497} - 1$. Pour chacun de ces générateurs, nous avons cherché des générateurs avec des petites valeurs de $C(T)$, afin que l'implantation soit la plus rapide possible. Aux tableaux 7.3, 7.6, 7.4 et 7.7, on donne les paramètres des générateurs qui ont montré la meilleure valeur de $V = \sum_{\ell=1}^{32} \Delta_\ell$. Dans les tableaux 7.3 et 7.6, en plus des paramètres, on donne le coût $C(X)$ associé à chacun des générateurs. Pour tous ces générateurs, on utilise la sortie $\mathbf{y}_n = \mathbf{v}_{n,0}$. Il s'est avéré que les générateurs qui utilisent la sortie $\mathbf{y}_n = \mathbf{v}_{n,1}$ n'obtiennent pas une aussi bonne équidistribution.

Pour la présentation des résultats, nous avons divisé celle-ci en deux sous-sections. La première concerne des générateurs qui ont des périodes inférieures ou égales à $2^{1024} - 1$. Pour une utilisation dans une application de simulation, ces générateurs sont ceux qui sont les plus pratiques étant donné que l'état est très grand, mais pas au point de devenir un handicap pour la facilité d'utilisation. La deuxième sous-section présente des générateurs qui ont une période extrêmement longue et un vecteur d'état énorme. Nous les présentons puisqu'ils s'agit de générateurs qui ont une période plus longue et une meilleure équidistribution que le Mersenne twister et démontrent à quel point la bibliothèque REGPOLY est efficace.

En effet, non seulement REGPOLY peut déterminer efficacement si le polynôme caractéristique d'un énorme générateur est primitif rapidement (en quelques minutes),

mais peut également calculer le plus court vecteur dans un réseau polynômial rapidement (même si le degré des polynômes de la base est énorme) pour ensuite déterminer l'équidistribution. Comme il a été dit précédemment, l'efficacité de l'implantation des algorithmes qui permettent de déterminer la primitivité d'un polynôme de très grand degré ainsi que ceux qui calculent le plus court vecteur d'un réseau polynômial a été une des grandes préoccupation de notre recherche. La présentation de générateurs de très grande période et qui ont une très bonne équidistribution confirme l'efficacité réelle de REGPOLY.

7.4.1 Générateurs de longue période

Nous présentons, dans cette sous-section, des ensembles de paramètres pour des WELLRNG de période $2^{512}-1$, $2^{521}-1$, $2^{607}-1$, $2^{800}-1$ et $2^{1024}-1$. Le choix des périodes $2^{521}-1$ et $2^{607}-1$ est justifié par le fait que ce sont deux nombres premiers de Mersenne, ce qui permet de simplifier le calcul de la primitivité des générateurs. Le choix de la période $2^{800}-1$ est intéressant puisqu'il nous permettra de comparer les WELLRNG avec le TT800 et certains générateurs obtenus au chapitre 5 du point de vue de l'équidistribution et de la vitesse d'exécution. Les périodes $2^{512}-1$ et $2^{1024}-1$ sont pratiques du point de vue de l'implantation, comme nous le verrons à la section 7.5.

Aux tableaux 7.3 et 7.4, on donne les paramètres de générateurs WELLRNG avec des périodes de $2^{512}-1$, $2^{521}-1$, $2^{607}-1$, $2^{800}-1$ et $2^{1024}-1$. Il est à remarquer que les générateurs avec $k = 512$, $k = 521$, $k = 607$ et $k = 1024$, sont tous ME, c'est-à-dire que l'équidistribution est parfaite et que $\Delta_\ell = 0$ pour $\ell = 1, \ldots, 32$. Pour les générateurs de période $2^{800}-1$, l'équidistribution est telle que $V = \sum_{\ell=1}^{32} = 3$. Le tableau 7.5 donne l'équidistribution plus en détails des générateurs trouvés et la compare avec le TT800 [67] et le LFSR dans $\mathbb{F}_{2^w}$ trouvé au chapitre 5 qui a démontré la meilleure équidistribution ($V = 36$), soit le seizième générateur du tableau 5.10. Dans

Tableau 7.3 – Générateurs trouvés.

Identification	T_0	T_1	T_2	T_3	$C(X)$
m_1 m_2 m_3	T_4	T_5	T_6	T_7	hw(P(z))
$k = 512$, $w = 32$, $r = 16$, $p = 0$					
WELLRNG512a	$M_0(-16)$	$M_0(-15)$	$M_0(11)$	M_7	21
13 9 5	$M_0(-2)$	$M_0(-18)$	$M_0(-28)$	$M_4(-5, \mathbf{a}_3)$	225
$k = 521$, $w = 32$, $r = 17$, $p = 23$					
WELLRNG521a	$M_0(-13)$	$M_0(-15)$	M_1	$M_3(-21)$	17
13 11 10	$M_0(-13)$	$M_3(1)$	M_7	$M_0(11)$	265
WELLRNG521b	$M_0(-21)$	$M_0(6)$	M_7	$M_0(-13)$	19
11 10 7	$M_0(13)$	$M_3(-10)$	$M_3(-5)$	$M_0(13)$	245
$k = 607$, $w = 32$, $r = 19$, $p = 1$					
WELLRNG607a	$M_0(19)$	$M_0(11)$	$M_0(-14)$	M_1	17
16 15 14	$M_0(18)$	M_1	M_7	$M_0(-5)$	295
WELLRNG607b	$M_0(-18)$	$M_0(-14)$	M_7	$M_0(18)$	18
16 8 13	$M_0(-24)$	$M_0(5)$	$M_0(-1)$	M_7	313
$k = 800$, $w = 32$, $r = 25$, $p = 0$					
WELLRNG800a	M_1	$M_0(-15)$	$M_0(10)$	$M_0(-11)$	19
14 18 17	$M_0(16)$	$M_3(20)$	M_1	$M_0(-28)$	303
WELLRNG800b	$M_0(-29)$	$M_3(-14)$	M_1	$M_3(19)$	20
9 4 22	M_1	$M_0(10)$	$M_2(\mathbf{a}_2)$	$M_0(-25)$	409
$k = 1024$, $w = 32$, $r = 32$, $p = 0$					
WELLRNG1024a	M_1	$M_0(8)$	$M_0(-19)$	$M_0(-14)$	19
3 24 10	$M_0(-11)$	$M_0(-7)$	$M_0(-13)$	M_7	407
WELLRNG1024b	$M_0(-21)$	$M_0(17)$	$M_2(\mathbf{a}_4)$	$M_0(15)$	21
22 25 26	$M_0(-14)$	$M_0(-21)$	M_1	M_7	475

Tableau 7.4 – Valeurs des vecteurs $\mathbf{a}_j$ de la table 7.3.

$\mathbf{a}_1$	6bdeef3a
$\mathbf{a}_2$	d3e43ffd
$\mathbf{a}_3$	da442d24
$\mathbf{a}_4$	8bdcb91e

ce tableau, on donne aussi hw($P(z)$), le nombre de coefficients non nuls du polynôme caractéristique $P(z)$ de la matrice de transition X, pour chacun des générateurs.

Le nombre de coefficients non nuls est important pour éviter d'avoir des corrélations entre certaines valeurs produites par le générateur. Par exemple, si le polynôme caractéristique de la matrice de transition est un trinôme, ceci signifie que chaque sortie $\mathbf{y}_n$ peut être déduite en additionnant, dans $\mathbb{F}_2^L$, deux sorties $\mathbf{y}_{n-i_1}$ et $\mathbf{y}_{n-i_2}$ telles que $i_1, i_2 \leq k$. Ainsi, si on considère les triplets $(u_n, u_{n-i_1}, u_{n-i_2})$, il y aura une forte dépendance entre les éléments de ce triplet [68, 67, 10]. Pour éviter ces dépendances, on préfère des générateurs dont la matrice de transition est telle que le polynôme caractéristique contient un nombre de coefficients non nuls près de $k/2$, ce qui est le cas pour les générateurs dont on donne les paramètres au tableau 7.3.

Pour donner une idée de la difficulté d'obtenir des WELLRNG qui sont ME, nous avons effectué une recherche de paramètres avec l'algorithme 7.2 avec $C_{max} \leq 25$. Nous avons utilisé l'algorithme à répétition pour obtenir 10000 générateurs de pleine période. Pour les générateurs avec $k = 521$, parmi les 10000 générateurs trouvés, il y en avait 20 qui étaient ME, soit 0.2% des générateurs. Pour ceux avec $k = 607$, parmi les 10000 générateurs, 195 étaient ME, soit près de 2% des générateurs. Pour les générateurs de période $2^{800} - 1$ nous n'en avons trouvé aucun qui était ME parmi les 10000 générateurs vérifiés. Ces pourcentages sont très révélateurs et montrent à quel point la famille des générateurs WELLRNG peut produire de très bons générateurs

Tableau 7.5 – Équidistribution des générateurs trouvés ($k = 800$).

Générateur	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7	Δ_8
	Δ_9	Δ_{10}	Δ_{11}	Δ_{12}	Δ_{13}	Δ_{14}	Δ_{15}	Δ_{16}
$V = \sum_{\ell=1}^{32} \Delta_\ell$	Δ_{17}	Δ_{18}	Δ_{19}	Δ_{20}	Δ_{21}	Δ_{22}	Δ_{23}	Δ_{24}
	Δ_{25}	Δ_{26}	Δ_{27}	Δ_{28}	Δ_{29}	Δ_{30}	Δ_{31}	Δ_{32}
WELLRNG800a								
$V = 3$				1				
hw$(P(z))$ =303	1							1
WELLRNG800b					1			
$V = 3$	1							
hw$(P(z))$ =409	1							
TT800			16		10	8	14	
	13	5	22	16	11	7	3	
$V = 261$	22	19	17	15	13	11	9	8
hw$(P(z)) = 93$	7	5	4	3	2	1		
LFSR								
$V = 36$						2	4	8
hw$(P(z)) = 375$	7	5	4	3	2	1		

du point de vue de l'équidistribution. Si on compare avec les générateurs introduits au chapitre 5, aucun des meilleurs générateurs trouvés n'était ME pour des périodes supérieures à $2^{96} - 1$.

7.4.2 Générateurs de période extrêmement longue

Aux tableaux 7.8–7.11, on donne, pour chacun des générateurs du tableau 7.6, la valeur du critère $V = \sum_{\ell=1}^{32} \Delta_\ell$, ainsi que la valeur de chacun des écarts en dimension obtenus Δ_ℓ pour $\ell = 1, \ldots, 32$. Également, on donne $\mathrm{hw}(P(z))$, le nombre de coefficients non nuls du polynôme caractéristique de la matrice X, pour chacun des générateurs. Pour fins de comparaison, on donne aussi les résultats du MT19937.

En comparant les résultats obtenus par les WELLRNG19937a, WELLRNG19937b et WELLRNG19937c avec ceux obtenus par le MT19937, on remarque que l'équidistribution est bien meilleure pour les WELLRNG que pour le Mersenne twister. Pour les générateurs WELLRNG19937a et WELLRNG19937b, on a $\Delta_\ell \leq 1$ pour toutes les résolutions. Pour le Mersenne twister, on n'a qu'à considérer $\ell = 3$ bits de résolution pour observer un grand écart Δ_ℓ. À part les résolutions $\ell = 1, 2, 4, 8, 16, 32$, le MT19937 est tel que $\Delta_\ell \geq 20$. Aussi, le nombre de coefficients non nuls du polynôme caractéristique de la matrice de transition est à l'avantage des WELLRNG. Le polynôme caractéristique de la matrice de transition du MT19937 n'a que 135 coefficients non nuls, alors que ceux des générateurs WELLRNG ont tous plus de 8000 coefficients non nuls.

On remarque que pour tous les générateurs proposés, on a toujours $\Delta_\ell \leq 1$ pour $\ell = 1, \ldots, 23$, à part pour deux : WELLRNG21701b et WELLRNG21701c. Pour le WELLRNG21701b, on a seulement $\Delta_2 = 2$ qui l'empêche de faire partie de ce groupe sélect de générateurs et pour le WELLRNG21701c, seulement $\Delta_{19} = 5$ lui fait défaut.

La signification de ces tableaux est que tous ces générateurs produisent des en-

sembles de points $\Psi(\mathbf{X}, I, L)$ extrêmement uniformes. Par exemple, si on considère l'ensemble de points en 1390 dimensions produit par le WELLRNG44497 et on divise chaque axe en 2^{32} divisions, on trouve exactement 2^{17} points dans chacune des divisions, car $t_{32} = t_{32}^* = 1390$ et $2^k/2^{t\ell} = 2^{44497}/2^{1390\times 32} = 2^{17}$.

Il se pourrait que l'on désire obtenir des générateurs qui ont une période extrêmement longue et qui sont ME, c'est-à-dire qu'on aimerait qu'ils soient tels que $\Delta_\ell = 0$. C'est un but réalisable pour les WELLRNG. À l'aide de REGPOLY, en faisant une recherche sur un bon tempering de Matsumoto-Kurita à la sortie du WELLRNG44497a, nous en avons obtenu un qui donne un générateur ME. Le tempering utilisé est

$$\begin{aligned} \mathbf{z}_n &= \mathbf{v}_{n,0} \\ \mathbf{z}_n &= \mathbf{z}_n \oplus ((\mathbf{z}_n \ll 7) \,\&\, \mathbf{b}) \\ \mathbf{y}_n &= \mathbf{z}_n \oplus ((\mathbf{z}_n \ll 15) \,\&\, \mathbf{c}) \end{aligned}$$

où $\mathbf{b} =$ `93dd1400` et $\mathbf{c} =$ `fa118000`. Ce générateur, avec ce tempering, constitue un record : il est celui dont l'équidistribution est parfaite pour tous les bits de la sortie et qui a la période la plus longue. Au cours de ma maîtrise, nous avions trouvé un TGFSR combiné dont la période était de l'ordre de 2^{1250} et qui était parfaitement équidistribué pour tous les bits de la sortie (29 bits) ; ceci était le record précédent.

Étant donné les opérations supplémentaires nécessaires pour implanter ce WELLRNG auquel on applique un tempering, nous ne poursuivons pas cette recherche de générateurs ME « tempérés », les gains en uniformité étant négligeables.

Tableau 7.6 – Générateurs trouvés.

Identification			T_0	T_1	T_2	T_3	$C(X)$
m_1	m_2	m_3	T_4	T_5	T_6	T_7	hw$(P(z))$
$k = 19937$, $w = 32$, $r = 624$, $p = 31$							
WELLRNG19937a			$M_0(-25)$	$M_0(27)$	$M_3(9)$	$M_0(1)$	21
70	179	449	M_1	$M_0(-9)$	$M_0(-21)$	$M_0(21)$	8585
WELLRNG19937b			$M_0(7)$	M_1	$M_0(12)$	$M_0(-10)$	21
203	613	123	$M_0(-19)$	$M_3(-11)$	$M_0(4)$	$M_0(-10)$	9679
WELLRNG19937c			M_1	$M_0(10)$	$M_2(\mathbf{a}_1)$	$M_0(-6)$	23
520	127	300	$M_0(-20)$	$M_4(-3, \mathbf{a}_2)$	M_1	$M_0(-16)$	8161
$k = 21701$, $w = 32$, $r = 679$, $p = 27$							
WELLRNG21701a			M_1	$M_0(-26)$	$M_0(19)$	M_7	24
151	327	84	$M_0(27)$	$M_0(-11)$	$M_5(15, 10, 27, \mathbf{a}_3)$	$M_0(-16)$	7609
WELLRNG21701b			$M_0(8)$	$M_0(-15)$	M_7	$M_0(17)$	21
498	316	651	$M_0(25)$	$M_0(-23)$	$M_0(-15)$	$M_0(23)$	9051
WELLRNG21701c			$M_0(13)$	$M_3(6)$	$M_0(13)$	$M_0(-2)$	22
272	473	131	M_1	$M_0(16)$	$M_4(-22, \mathbf{a}_4)$	$M_0(-12)$	8711
$k = 23209$, $w = 32$, $r = 726$, $p = 23$							
WELLRNG23209a			$M_0(28)$	M_1	$M_0(18)$	$M_0(3)$	22
667	43	462	$M_0(21)$	$M_0(-17)$	$M_0(-28)$	$M_0(-1)$	10871
WELLRNG23209b			$M_2(\mathbf{a}_5)$	M_1	$M_5(15, 30, 15, \mathbf{a}_6)$	$M_0(-24)$	24
610	175	662	$M_0(-26)$	M_1	M_7	$M_0(16)$	10651
WELLRNG23209c			M_1	$M_0(2)$	M_7	$M_0(9)$	19
207	482	424	$M_0(-18)$	$M_0(-13)$	$M_0(15)$	$M_0(-30)$	8643
$k = 44497$, $w = 32$, $r = 1391$, $p = 15$							
WELLRNG44497a			$M_0(-24)$	$M_0(30)$	$M_0(-10)$	$M_3(-26)$	24
23	481	229	M_1	$M_0(20)$	$M_5(9, 14, 5, \mathbf{a}_7)$	M_1	16883
WELLRNG44497b			$M_0(-20)$	$M_0(5)$	$M_0(2)$	$M_0(-25)$	20
806	1191	1287	$M_0(-13)$	M_7	$M_0(-4)$	$M_3(25)$	17917
WELLRNG44497c			M_1	$M_3(-26)$	$M_4(3, \mathbf{a}_8)$	M_7	22
916	768	671	$M_0(-15)$	$M_0(27)$	$M_5(7, 11, 4, \mathbf{a}_9)$	M_1	8261

Tableau 7.7 – Valeur des vecteurs $\mathbf{a}_j$ de la table 7.6.

$\mathbf{a}_1$	=	9815b976
$\mathbf{a}_2$	=	6ba3cd00
$\mathbf{a}_3$	=	86a9d87e
$\mathbf{a}_4$	=	2ca4a19e
$\mathbf{a}_5$	=	a8c296d1
$\mathbf{a}_6$	=	5d6b45cc
$\mathbf{a}_7$	=	b729fcec
$\mathbf{a}_8$	=	7d2456af
$\mathbf{a}_9$	=	7fc7906a

Tableau 7.8 – Équidistribution des générateurs trouvés ($k = 19937$).

Générateur	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7	Δ_8
	Δ_9	Δ_{10}	Δ_{11}	Δ_{12}	Δ_{13}	Δ_{14}	Δ_{15}	Δ_{16}
$V = \sum_{\ell=1}^{32} \Delta_\ell$	Δ_{17}	Δ_{18}	Δ_{19}	Δ_{20}	Δ_{21}	Δ_{22}	Δ_{23}	Δ_{24}
	Δ_{25}	Δ_{26}	Δ_{27}	Δ_{28}	Δ_{29}	Δ_{30}	Δ_{31}	Δ_{32}
WELLRNG19937a		1					1	
							1	
$V = 4$								
hw$(P(z))$ =8585				1				
WELLRNG19937b			1					
	1					1		1
$V = 5$								
hw$(P(z))$ =9679								1
WELLRNG19937c					1			
$V = 126$								
hw$(P(z))$ =8161					64	41	20	
MT19937			405		249	207	355	
	346	124	564	415	287	178	83	
$V = 6750$	549	484	426	373	326	283	243	207
hw$(P(z))$ =135	174	143	115	89	64	41	20	

Tableau 7.9 – Équidistribution des générateurs trouvés($k = 21701$).

Générateur	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7	Δ_8
	Δ_9	Δ_{10}	Δ_{11}	Δ_{12}	Δ_{13}	Δ_{14}	Δ_{15}	Δ_{16}
$V = \sum_{\ell=1}^{32} \Delta_\ell$	Δ_{17}	Δ_{18}	Δ_{19}	Δ_{20}	Δ_{21}	Δ_{22}	Δ_{23}	Δ_{24}
	Δ_{25}	Δ_{26}	Δ_{27}	Δ_{28}	Δ_{29}	Δ_{30}	Δ_{31}	Δ_{32}
WELLRNG21701a								
$V = 1$				1				
hw$(P(z))$ =7609								
WELLRNG21701b		2						
	1					1		
$V = 28$				1				
hw$(P(z))$ =9051		2		1		19	1	
WELLRNG21701c				1	1		1	
	1					1		1
$V = 51$	1	1	5	1				
hw$(P(z))$ =8711	1	2	3	4	5		22	

Tableau 7.10 – Équidistribution des générateurs trouvés($k = 23209$).

Générateur	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7	Δ_8
	Δ_9	Δ_{10}	Δ_{11}	Δ_{12}	Δ_{13}	Δ_{14}	Δ_{15}	Δ_{16}
$V = \sum_{\ell=1}^{32} \Delta_\ell$	Δ_{17}	Δ_{18}	Δ_{19}	Δ_{20}	Δ_{21}	Δ_{22}	Δ_{23}	Δ_{24}
	Δ_{25}	Δ_{26}	Δ_{27}	Δ_{28}	Δ_{29}	Δ_{30}	Δ_{31}	Δ_{32}
WELLRNG23209a						1		
$V = 3$							1	1
hw$(P(z))$ =10871								
WELLRNG23209b			1	1				
				1				
$V = 3$								
hw$(P(z))$ =10651								
WELLRNG23209c					1			1
				1				
$V = 149$								
hw$(P(z))$ =8643					75	48	23	

Tableau 7.11 – Équidistribution des générateurs trouvés($k = 44497$).

Générateur	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5	Δ_6	Δ_7	Δ_8
	Δ_9	Δ_{10}	Δ_{11}	Δ_{12}	Δ_{13}	Δ_{14}	Δ_{15}	Δ_{16}
$V = \sum_{\ell=1}^{32} \Delta_\ell$	Δ_{17}	Δ_{18}	Δ_{19}	Δ_{20}	Δ_{21}	Δ_{22}	Δ_{23}	Δ_{24}
	Δ_{25}	Δ_{26}	Δ_{27}	Δ_{28}	Δ_{29}	Δ_{30}	Δ_{31}	Δ_{32}
WELLRNG44497a		1	1	1				1
								1
$V = 7$								1
hw$(P(z))$ =16883			1					
WELLRNG44497b		1		1		1		1
	1		1					1
$V = 548$								1
hw$(P(z))$ =17917			60	198	144	93	45	
WELLRNG44497c						1		1
	1		1					
$V = 1915$								463
hw$(P(z))$ = 8261	388	321	258	199	144	93	45	

7.5 Implantation des générateurs

Dans cette section, nous présentons deux implantations de générateurs WELLRNG. La première est celle du WELLRNG1024a tandis que la deuxième est celle du WELLRNG44497a. Les deux implantations sont données en langage C. Les figures 7.1 et 7.2 présentent l'implantation du WELLRNG1024a. Dans celle-ci, on définit un tableau `STATE` de $r = 32$ `unsigned ints`. À l'initialisation, il contient, dans `STATE[`i`]`, le vecteur $\mathbf{v}_{0,i}$ pour $i = 0, \ldots, 31$. On remarque, dans l'algorithme 7.1 qu'on assigne à $\mathbf{v}_{n+1,j}$ la valeur de $\mathbf{v}_{n,j-1}$ pour $j = r-1$ à $j = 2$. Ce qui pourrait se faire de manière très coûteuse avec une boucle `for( ; ; )`. Au lieu de cela, on introduit la variable entière `state_i`. Par cette variable, on utilise `STATE[state_i+`i mod r`]` pour désigner $\mathbf{v}_{n,i}$ où `state_i` est égal à n mod r. De cette manière, l'affectation $\mathbf{v}_{n,j-1} \leftarrow \mathbf{v}_{n+1,j}$ se fait de manière automatique quand on soustrait à `state_i` une unité.

Pour faciliter la lecture du code, on utilise plusieurs macros. Les macros `R`, `W`, `P`, `M1`, `M2` et `M3`, représentent les valeurs de r, w, p, m_1, m_2 et m_3. On représente aussi les vecteurs de bits $\mathbf{v}_{n,i}$ par des macros. Par exemple, `V0`, `VM1` et `VRm1` représentent les vecteurs $\mathbf{v}_{n,0}$, $\mathbf{v}_{n,m_1}$ et $\mathbf{v}_{n,r-1}$. D'autres représentent la valeur des vecteurs $\mathbf{v}_{n+1,i}$. Ceux-ci sont `newV0` et `newV1` qui représentent $\mathbf{v}_{n+1,0}$ et $\mathbf{v}_{n+1,1}$.

À chaque itération, on soustrait un à la variable `state_i` modulo r. Dans le cas du WELLRNG1024a, la valeur de r est une puissance de 2 et l'opération « modulo r » peut s'effectuer avec un masque de bits. C'est ce qui est fait dans l'implantation présentée. Ce truc permet de simplifier grandement l'implantation. On remarque que soustraire une unité modulo r à `state_i` correspond à ajouter $(r-1)$ modulo r à `state_i`.

Examinons la fonction `WELLRNG1024` afin de comprendre comment celle-ci implante l'algorithme 7.1. La première ligne correspond à la multiplication $S(\mathbf{v}_{n,r-2}^{\mathsf{T}}, \mathbf{v}_{n,r-1}^{\mathsf{T}})^{\mathsf{T}}$. Puisque $p = 0$, cette multiplication est triviale et le résultat est $\mathbf{v}_{n,r-1}$, ce à quoi

correspond la ligne `z0 = VRm1 ;`.

La deuxième ligne correspond à $\mathbf{v}_{n,0} \oplus M_0(8)\mathbf{v}_{n,m_1}$. Le macro `MATOPOS` effectue la multiplication par $M_0(t)$ quand t est positif. La troisième ligne exécute $\mathbf{z}_2 \leftarrow M_0(-19)\mathbf{v}_{n,m_2} \oplus M_0(-14)\mathbf{v}_{n,m_3}$. Le macro `MATONEG` effectue la multiplication par $M_0(t)$ quand t est négatif. La quatrième ligne correspond à $\mathbf{v}_{n+1,1} \leftarrow \mathbf{z}_3 \leftarrow \mathbf{z}_1 \oplus \mathbf{z}_2$. Ainsi, la variable $\mathbf{z}_3$ est sous-entendue et pour y faire référence à la prochaine ligne on utilise `newV1` qui représente $\mathbf{v}_{n+1,1}$. La prochaine ligne effectue $\mathbf{v}_{n+1,0} \leftarrow M_0(-11)\mathbf{z}_0 \oplus M_0(-7)\mathbf{z}_1 \oplus M_0(-13)\mathbf{z}_2$. La valeur retournée est `((double) STATE[state_i]) * FACT` qui représente $\sum_{i=1}^{32} v_{n+1,0}^{(i-1)} 2^{-i}$.

La figure 7.3 donne un exemple d'utilisation du générateur. Dans cet exemple, on produit 100 valeurs aléatoires, on les additionne et affiche la moyenne de ceux-ci. On remarque que l'état du générateur a été initialisé à l'aide de la fonction `InitWELLRNG1024()`. Une attention particulière doit être apportée à la manière de choisir l'état initial, c'est-à-dire les valeurs mises dans le tableau `seed`, à cause du grand état du générateur. Des effets non désirables peuvent se produire si le germe est choisi trop simplement. Par exemple, si l'état initial est constitué de zéros et très rarement de uns, alors les états subséquents peuvent être semblables pour un grand nombre d'itérations consécutives, donnant des nombres aléatoires de mauvaise qualité. Dans notre exemple, on a initialisé l'état du générateur avec un autre générateur (`favoriteOtherRNG`). Pour plus de détails sur l'initialisation de générateurs qui ont un état immense, nous vous invitons à aller voir le site internet

`http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html`

maintenu par Makoto Matsumoto. Nous présentons, à la prochaine section, une petite expérience qui démontre que le problème de l'initialisation observé sur le MT19937 et le TT800 est beaucoup moins prononcé pour les WELLRNG.

L'implantation du générateur WELLRNG44497a est présentée aux figures 7.4, 7.5

et 7.6. Pour ce générateur, la valeur de r n'est pas une puissance de 2 et l'opération modulo r ne peut pas être effectuée par un masque de bits. L'opération modulo étant une opération coûteuse et qui ralentit grandement l'implantation de générateurs, nous l'éviterons. Ceci a pour conséquence de compliquer l'implantation du générateur par rapport à celle du WELLRNG1024a, mais permet une implantation rapide.

En plus des macros définis dans l'implantation du WELLRNG1024a, nous définissons les macros `VM1Out`, `VM2Out`, `VM3Out`, `VRm1Out`, `VRm2`, `VRm2Out` et `newV0Out` qui sont les mêmes macros du même nom sans la mention `Out`. Ces macros sont utiles lorsque l'adresse dans le tableau `STATE` à laquelle ils réfèrent n'est pas comprise dans $\{0, \ldots, r-1\}$.

À chaque itération, on doit consulter le contenu de $\mathbf{v}_{n,0}$, $\mathbf{v}_{n,m_1}$, $\mathbf{v}_{n,m_2}$, $\mathbf{v}_{n,m_3}$, $\mathbf{v}_{n,r-2}$ et $\mathbf{v}_{n,r-1}$ et modifier le contenu de $\mathbf{v}_{n+1,0}$ et $\mathbf{v}_{n+1,1}$ (le contenu de $\mathbf{v}_{n+1,j}$ pour $j = 2, \ldots, r-1$ est mis à jour implicitement en décrémentant `state_i`). Pour une valeur de `state_i` donnée, le macro correspondant à chacune de ces valeurs doit porter la mention `Out` ou non dépendemment si celui-ci pointe vers une valeur à l'extérieur du tableau `STATE`. Ceci constitue un problème et peut être résolu en identifiant 6 cas spécifiques. Ces cas sont résumés dans le tableau 7.5. Par exemple, si `state_i` est 0, alors le macro qui désigne $\mathbf{v}_{n,r-1}$ doit être `VRm1Out` puisque `VRm1` est `STATE[-1]` qui n'est pas défini.

Pour tenir compte de ces 6 cas, on pourrait utiliser, à chaque itération, une expression conditionnelle qui vérifie la valeur de `state_i`, mais ceci se révèle inefficace. Par contre, on peut profiter du fait que l'on sait toujours quel cas s'applique après l'itération courante. Pour ce faire, on définit une fonction qui exécute une itération du générateur pour chacun des 6 cas. Celles-ci sont les fonctions `case_1`, .., `case_6`. On définit également le pointeur vers une fonction `WELLRNG44497a`. À l'initialisation, puisque `state_i` est zéro, on affecte à `WELLRNG44497a` la valeur `case_1`. À la fin de la fonction `case_1`, on change `WELLRNG44497a` pour `case_3` puisqu'on sait qu'au début

de la prochaine itération, la valeur de `state_i` sera de $r-1$. De même, à la fin de chaque fonction `case_x`, on change la valeur du pointeur `WELLRNG44497a` pour tenir compte de la nouvelle valeur de `state_i`.

Dans le cas du WELLRNG44497a, la multiplication par S n'est pas triviale. À cette fin, on a définit les macros `MASKU` et `MASKL` qui représentent des masques de bits. Le premier est composé de $w-p$ zéros suivis de p uns et le deuxième est composé de $w-p$ uns suivis de p zéros. Ces masques sont utiles pour la multiplication par la matrice S. Ainsi, la multiplication pas S s'effectue par l'opération `z0 = (Vrm1Out & MASKL) | (Vrm2Out & MASKU) ;`.

Tableau 7.12 – Macros à utiliser en fonction de la valeur de `state_i` (en supposant $m_1 < m_3 < m_2$, comme pour le WELLRNG44497a).

`state_i`	Macros à utiliser
$= 0$	`V0, VM1, VM2, VM3, VRm1Out, VRm2Out, newV0Out, newV1`
$= 1$	`V0, VM1, VM2, VM3, VRm1, VRm2Out, newV0, newV1`
$\in \{r-m_1, \ldots, r-1\}$	`V0, VM1Out, VM2Out, VM3Out, VRm1, VRm2, newV0, newV1`
$\in \{r-m_2, \ldots, r-m_1-1\}$	`V0, VM1, VM2Out, VM3, VRm1, VRm2, newV0, newV1`
$\in \{r-m_3, \ldots, r-m_2-1\}$	`V0, VM1, VM2Out, VM3Out, VRm1, VRm2, newV0, newV1`
$\in \{2, \ldots, r-m_3-1\}$	`V0, VM1, VM2, VM3, VRm1, VRm2, newV0, newV1`

```
#include "WELLRNG1024.h"
#define W 32
#define R 32
#define P 0
#define M1 3
#define M2 24
#define M3 10

#define MATOPOS(t,v) (v^(v>>t))
#define MATONEG(t,v) (v^(v<<(-t)))
#define Identity(v) (v)

#define V0            STATE[state_i                   ]
#define VM1           STATE[(state_i+M1)  & 0x0000001fUL]
#define VM2           STATE[(state_i+M2)  & 0x0000001fUL]
#define VM3           STATE[(state_i+M3)  & 0x0000001fUL]
#define VRm1          STATE[(state_i+31)  & 0x0000001fUL]
#define newV0         STATE[(state_i+31)  & 0x0000001fUL]
#define newV1         STATE[state_i                    ]
#define A1_TEMP   0xdb79fb31UL
#define B1_TEMP   0x0fdaa2c1UL
#define B2_TEMP   0x27c5a58dUL
#define C1_TEMP   0x71e993feUL

#define FACT 2.32830643653869628906e-10

static unsigned int state_i = 0;
static unsigned int STATE[R];
static unsigned int z0, z1, z2, s1,s2,s3,t1,t2,y1,y2;
static unsigned int j,counter;


void InitWELLRNG1024 (unsigned int *init){
   int j;
   state_i = 0;
   counter = R-3;
   for (j = 0; j < R; j++)
     STATE[j] = init[j];
}

double WELLRNG1024 (void){

  z0    = VRm1;
  z1    = Identity(V0)       ^ MATOPOS (8, VM1);
  z2    = MATONEG (-19, VM2) ^ MATONEG(-14,VM3);
  newV1 = z1                 ^ z2;
  newV0 = MATONEG (-11,z0)   ^ MATONEG(-7,z1)    ^ MATONEG(-13,z2);
  state_i = (state_i + 31) & 0x0000001fUL;
  return ((double) STATE[state_i]) * FACT;
}
```

Figure 7.1 – Fichier `WELLRNG1024.c`.

```
void InitWELLRNG1024 (unsigned int *init);
double WELLRNG1024 (void);
```

Figure 7.2 – Fichier `WELLRNG1024.h`.

```
#include <stdio.h>
#include <stdlib.h>
#include "WELLRNG1024.h"

int main (void)
{
   int j;
   unsigned int seed[32];
   double sum = 0.0;
   for (j = 0; j < 32; j++)
      seed[j] = favoriteOtherRNG ();

   InitWELLRNG1024 ( seed );
   for (j = 0; j < 100; j++)
      sum += WELLRNG1024 ();

   printf ("moyenne (sur 100 valeurs) = %8.7f\n", sum / 100.0);
   return 0;
}
```

Figure 7.3 – Exemple d'utilisation du WELLRNG1024a.

```
#include<stdio.h>
#include<stdlib.h>

#define W 32
#define R 1391
#define P 15
#define MASKU (0xffffffffUL>>(W-P))
#define MASKL (~ MASKU)
#define M1 23
#define M2 481
#define M3 229
#define MATOPOS(t,v) (v^(v>>t))
#define MATONEG(t,v) (v^(v<<(-t)))
#define MAT1(v) v
#define MAT3NEG(t,v) (v<<(-t))
#define MAT5(r,a,ds,dt,v) ((v & dt)?((((v<<r)^(v>>(W-r)))&ds)^a) :(((v<<r)^(v>>(W-r)))&ds))
#define V0            STATE[state_i]
#define VM1Out        STATE[state_i+M1-R]
#define VM1           STATE[state_i+M1]
#define VM2Out        STATE[state_i+M2-R]
#define VM2           STATE[state_i+M2]
#define VM3Out        STATE[state_i+M3-R]
#define VM3           STATE[state_i+M3]
#define VRm1          STATE[state_i-1]
#define VRm1Out       STATE[state_i+R-1]
#define VRm2          STATE[state_i-2]
#define VRm2Out       STATE[state_i+R-2]
#define newV0         STATE[state_i-1]
#define newV0Out      STATE[state_i-1+R]
#define newV1         STATE[state_i]
#define FACT 2.32830643653869628906e-10

static int state_i=0;
static unsigned int STATE[R];
static unsigned int z0,z1,z2;
static double case_1(void);
static double case_2(void);
static double case_3(void);
static double case_4(void);
static double case_5(void);
static double case_6(void);
       double (*WELLRNG44497a)(void);


void InitWELLRNG44497a(unsigned int *init){
   int j;
   state_i=0;
   WELLRNG44497a = case_1;
   for(j=0;j<R;j++)  STATE[j]=init[j];
}
```

Figure 7.4 – Fichier `WELLRNG44497a.c`.

```
void InitWELLRNG44497a(unsigned int *init);
extern double (*WELLRNG44497a)(void);
```

Figure 7.5 – Fichier `WELLRNG44497.h`.

```
double case_1(void){  // state_i == 0
  z0           = (VRm1Out & MASKL)  | (VRm2Out & MASKU);
  z1           = MATONEG(-24,V0)    ^ MATOPOS(30,VM1);
  z2           = MATONEG(-10,VM2)   ^ MAT3NEG(-26,VM3);
  newV1        = z1                 ^ z2;
  newV0Out     = MAT1(z0)      ^ MATOPOS(20,z1)
               ^  MAT5(9,0xb729fcecUL,0xfbffffffUL,0x00020000UL,z2) ^ MAT1(newV1);
  state_i      = R-1;
  WELLRNG44497a = case_3;
  return ((double)STATE[state_i]*  2.3283064370807974e-10 );
}
static double case_2(void){  // state_i == 1
  z0           = (VRm1 & MASKL)     | (VRm2Out & MASKU);
  z1           = MATONEG(-24,V0)    ^ MATOPOS(30,VM1);
  z2           = MATONEG(-10,VM2)   ^ MAT3NEG(-26,VM3);
  newV1        = z1                 ^ z2;
  newV0        =  MAT1(z0)     ^ MATOPOS(20,z1)
               ^ MAT5(9,0xb729fcecUL,0xfbffffffUL,0x00020000UL,z2) ^ MAT1(newV1);
  state_i      = 0;
  WELLRNG44497a = case_1;
  return ((double)STATE[state_i]*  2.3283064370807974e-10 );
}
static double case_3(void){  // state_i+M1 >= R
  z0           = (VRm1 & MASKL)      | (VRm2 & MASKU);
  z1           = MATONEG(-24,V0)     ^ MATOPOS(30,VM1Out);
  z2           = MATONEG(-10,VM2Out) ^ MAT3NEG(-26,VM3Out);
  newV1        = z1                  ^ z2;
  newV0        = MAT1(z0)       ^ MATOPOS(20,z1)
               ^ MAT5(9,0xb729fcecUL,0xfbffffffUL,0x00020000UL,z2) ^ MAT1(newV1);
  state_i--;
  if(state_i+M1<R)
    WELLRNG44497a = case_4;
  return ((double)STATE[state_i]*  2.3283064370807974e-10 );
}
static double case_4(void){  // state_i+M3 >= R
  z0           = (VRm1 & MASKL)      | (VRm2 & MASKU);
  z1           = MATONEG(-24,V0)     ^ MATOPOS(30,VM1);
  z2           = MATONEG(-10,VM2Out) ^ MAT3NEG(-26,VM3Out);
  newV1        = z1                  ^ z2;
  newV0        = MAT1(z0)       ^ MATOPOS(20,z1)
               ^ MAT5(9,0xb729fcecUL,0xfbffffffUL,0x00020000UL,z2) ^ MAT1(newV1);
  state_i--;
  if (state_i+M3 < R)
    WELLRNG44497a = case_5;
  return ((double)STATE[state_i]*  2.3283064370807974e-10 );
}
static double case_5(void){  //state_i+M2 >= R
  z0           = (VRm1 & MASKL)      | (VRm2 & MASKU);
  z1           = MATONEG(-24,V0)     ^ MATOPOS(30,VM1);
  z2           = MATONEG(-10,VM2Out) ^ MAT3NEG(-26,VM3);
  newV1        = z1                  ^ z2;
  newV0        = MAT1(z0)       ^ MATOPOS(20,z1)
               ^ MAT5(9,0xb729fcecUL,0xfbffffffUL,0x00020000UL,z2) ^ MAT1(newV1);
  state_i--;
  if(state_i+M2 < R)
    WELLRNG44497a = case_6;
  return ((double)STATE[state_i]*  2.3283064370807974e-10 );
}
static double case_6(void){  // 2 <= state_i <= R-M2-1
  z0           = (VRm1 & MASKL)    | (VRm2 & MASKU);
  z1           = MATONEG(-24,V0)   ^ MATOPOS(30,VM1);
  z2           = MATONEG(-10,VM2)  ^ MAT3NEG(-26,VM3);
  newV1        = z1                ^ z2;
  newV0        = MAT1(z0)     ^ MATOPOS(20,z1)
               ^ MAT5(9,0xb729fcecUL,0xfbffffffUL,0x00020000UL,z2) ^ MAT1(newV1);
  state_i--;
  if(state_i == 1 )
    WELLRNG44497a = case_2;
  return ((double)STATE[state_i]*  2.3283064370807974e-10 );
}
```

Figure 7.6 – Fichier `WELLRNG44497.c` (suite).

7.6 Tests statistiques et performances

Nous avons fait passer les batteries de tests smallCrush, Crush et BigCrush[48] aux générateurs WELLRNG512a, WELLRNG607a, WELLRNG800a, WELLRNG1024a, WELLRNG19937a et WELLRNG44497a. Tous les tests, sans exception, ont été réussis. La batterie de test Crush prend une heure et demie à s'exécuter sur un pentium 2.8Ghz et la batterie Crush s'exécute en un plus plus de neuf heures. Étant donné la rigueur des tests contenus dans ces batteries de tests, on peut conclure que ces générateurs sont très robustes et peuvent être utilisés avec confiance dans des applications de simulation stochastique.

Comme pour les générateurs introduits au chapitre 5, avec les mêmes machines et les même options de compilation, nous avons chronométré le temps nécessaire pour produire et additionner 10^9 nombres aléatoires pour les générateurs WELLRNG607a, WELLRNG800a, WELLRNG19937a et WELLRNG44497a dont l'implantation est semblable à celle du générateur WELLRNG44497a décrite à la section 7.5. Les générateurs qui ont un (b) ont été implantés avec une expression conditionnelle afin d'éviter de faire référence à une adresse à l'extérieur du tableau `STATE`. Dans ces implantations, on n'utilise pas de pointeur vers une fonction : tout se fait dans la même fonction. Nous avons aussi mesuré la vitesse des implantations des générateurs WELLRNG512a et WELLRNG1024a. L'implantation du WELLRNG512a est semblable à celle du WELLRNG1024a qui est décrite à la section 7.5.

Les temps sont exposés dans le tableau 7.13. Pour la comparaison, nous avons copié les résultats du tableau 5.12.

On remarque que les temps d'exécution sont semblables pour tous les WELLRNG. Par contre, les WELLRNG512a et WELLRNG1024a sont les plus rapides par une marge d'environ 5% sur les autres WELLRNG. Le plus lent est le WELLRNG44497a qui prend environ 14% plus de temps que le plus rapide des WELLRNG. On remarque

que les implantations qui utilisent une expression conditionnelle sont plus lentes pour les WELLRNG19937a et WELLRNG44497a, mais, de manière surprenante, ce type d'implantation est plus rapide pour le WELLRNG607a. On peut voir que la stratégie d'utiliser un pointeur vers une fonction dont la valeur dépend de `state_i` peut être payante du point de vue de la vitesse d'exécution. Mais cette approche n'est pas portable dans certains langages de programmation. Java, par exemple, ne permet pas l'utilisation de pointeurs et l'implantation avec une expression conditionnelle serait à privilégier.

Si on compare les résultats avec d'autres générateurs, on remarque qu'il sont semblables au niveau de la vitesse. Comparativement au MT19937, le WELLRNG1024a et le WELLRNG19937a sont environ 15% plus lents. Alors, pourquoi utiliser le WELLRNG19937a au lieu du MT19937 s'il n'est pas plus rapide? Après tout, le MT19937 est équidistribué en dimension 623, ce qui est excellent, et passe tous les tests statistiques raisonnables. La réponse est donnée à la prochaine section et elle est convaincante.

7.7 WELLRNG vs Mersenne twister

Dans cette dernière section, nous effectuons une expérience fort simple afin de démontrer que le problème de l'initialisation du WELLRNG n'est pas aussi grave que celui du Mersenne twister et du TT800. Quand l'état d'un Mersenne twister (ou un TT800) contient beaucoup de zéros et peu de uns, celui-ci a du mal à produire des valeurs qui sont éloignées de zéros et ce, pour un grand nombre d'itérations.

L'expérience que nous faisons est la suivante. Soit $u_{i,j}$, la $(i+1)$-ième valeur produite par le générateur considéré quand l'état initial $\mathbf{x}_0$ est mis à $\mathbf{e}_j$, le j-ième

Tableau 7.13 – Temps nécessaire pour itérer 10^9 fois et additionner tous les nombres aléatoires produits.

Générateur	Temps (s)
WELLRNG512a	36.0
WELLRNG607a	38.1
WELLRNG607a(b)	36.9
WELLRNG800a	40.7
WELLRNG1024a	36.0
WELLRNG19937a	36.9
WELLRNG19937a(b)	42.2
WELLRNG44497a	41.3
WELLRNG44497a(b)	50.7
F2wLFSR2_32_800	39.5
F2wPolyLCG2_32_800	36.4
F2wPolyLCG2_32_800(*)	31.3
F2wLFSR3_7_800	32.8
F2wPolyLCG3_7_800	40.7
F2wPolyLCG3_7_800(*)	33.0
TT800	44.0
MT19937	31.6
MRG32k3a	101

vecteur unitaire. Soit

$$\bar{u}_i = \frac{1}{k}\sum_{j=1}^{k} u_{i,j},$$

la moyenne des i-ièmes valeurs produites, sur toutes les initialisations considérées, pour $i \geq 0$.

Soit

$$\mu_n = \frac{1}{n}\sum_{i=0}^{n} \bar{u}_i,$$

pour $n \geq 0$, la moyenne des $(n+1)$ premières valeurs produites, à partir de toutes les initialisations considérées et

$$\mu_{n,p} = \frac{1}{p}\sum_{i=0}^{p-1} \bar{u}_{n+i},$$

pour $n \geq 0$, $p \geq 1$, la moyenne mobile qui considère les valeurs de $\bar{u}_n$ à $\bar{u}_{n+p-1}$ inclusivement.

Si les suites $\{u_{i,j}\}_{i\geq 0}$, pour $j = 1, \ldots, k$, étaient vraiment des suites de variables aléatoires uniformes et indépendantes sur $[0, 1)$, alors on aurait $\mathrm{E}[\mu_n] = \mathrm{E}[\mu_{n,p}] = 1/2$ pour tous $n \geq 0$ et $p \geq 0$.

À la figure 7.7, on illustre le comportement de μ_n pour $n = 0, \ldots, 10^5$ pour le WELLRNG800a et le TT800. On peut remarquer que la moyenne du TT800 prend énormément d'itérations ($> 10^5$) avant d'atteindre la demie. Par contre, pour le WELLRNG800a, en moins de $n = 1400$ itérations, μ_n atteint la valeur de 0.49.

À la figure 7.8, on illustre le comportement de $\mu_{n,p}$ pour $n = 0, \ldots, 10^5$ et $p = 100$ pour les générateurs WELLRNG800a et le TT800. On observe que la moyenne mobile du WELLRNG800a atteint presqu'immédiatement le cap de la demie. Pour ce qui est du TT800, on doit attendre autour de $n = 60000$ itérations avant d'obtenir un comportement acceptable.

Pour des fins de comparaisons, nous avons également illustré les résultats obtenus

par les générateurs F2wLFSR2_32_800 et F2wLFSR3_7_800 dont l'implantation est donnée au chapitre 5.

Dans les deux autres figures 7.9 et 7.10, on illustre la même chose, mais en comparant le WELLRNG19937a et le MT19937. À la figure 7.9, on illustre le comportement de μ_n pour $n = 0, \ldots, 10^6$. On peut remarquer que, comme le TT800, la moyenne du MT19937 prend énormément d'itérations avant d'atteindre la demie. Par contre, pour le WELLRNG19937a, en moins de $n = 18000$ itérations, μ_n atteint la valeur de 0.49.

À la figure 7.10, on illustre le comportement de $\mu_{n,p}$ pour $n = 0, \ldots, 10^6$ et $p = 1000$ pour les générateurs WELLRNG19937a et le MT19937. On observe que, comme pour le WELLRNG800a, la moyenne mobile du WELLRNG19937a atteint presqu'immédiatement le cap de la demie. Pour ce qui est du MT19937, on doit attendre autour de $n = 700000$ itérations avant d'obtenir un comportement acceptable.

Puisque les figures 7.8 et 7.10 n'indiquent pas clairement quand la moyenne mobile atteint la demie, nous présentons la figure 7.11 qui donne la moyenne mobile $\mu_{n,p}$ pour $n = 0, \ldots, 1000$ et $p = 5$. On peut observer que la vitesse à laquelle les deux générateurs atteignent la demie est très rapide. Dans ce graphique, la courbe décrite par le WELLRNG19937a est plus lisse que celle du WELLRNG800a. Ceci est dû à la plus grande valeur de k pour le WELLRNG19937a : $\mu_{n,p}$ considère plus de valeurs pour le WELLRNG19937a.

Pour les générateurs F2wLFSR2_32_800 et F2wLFSR3_7_800, on remarque que le comportement du F2wLFSR2_32_800 est très similaire au TT800 et celui du F2wLFSR-3_7_800 est très similaire au WELLRNG800a. Pour le F2wLFSR2_32_800, ceci n'est pas surprenant puisque les deux générateurs utilisent une récurrence de la forme $\mathbf{v}_n = B\mathbf{v}_{n-18} + A\mathbf{v}_{n-25}$ où seules les matrices A et B les différencient. Pour le générateur F2wLFSR3_7_800, à chaque itération, il observe trois blocs de 32 bits

pour en modifier un. Ceci semble procurer un assez bon « brassage » pour un état de 800 bits.

Ce qu'il est important de retenir de ces graphiques est que les WELLRNG semblent brasser davantage l'état d'une itération à l'autre. Quand l'état d'un Mersenne twister contient peu de bits mis à un, celui-ci tend à conserver cette propriété pour plusieurs itérations consécutives. Les WELLRNG, étant donné qu'ils ont une matrice de transition X beaucoup plus complexe que les Mersenne twister, n'ont pas ce problème et sont capables de produire rapidement des états qui contiennent beaucoup de bits en partant d'une initialisation contenant peu de bits. Le polynôme caractéristique des WELLRNG a beaucoup plus de coefficients non nuls que ceux des Mersenne Twister. Ceci est aussi une indication du meilleur brassage des WELLRNG.

Ce meilleur brassage constitue un avantage important des générateurs WELLRNG qui permet d'éviter des situations fâcheuses pour un utilisateur peu familier sur la manipulation des générateurs de nombres aléatoires. Les nouveaux utilisateurs ne comprennent pas nécessairement à fond l'importance de bien initialiser un générateur. S'il advient qu'un utilisateur initialise mal un WELLRNG, alors les effets sont beaucoup moins désastreux pour le WELLRNG que pour un Mersenne twister.

Un exemple de mauvaise utilisation d'un générateur est reporté à l'adresse internet `http://random.mat.sbg.ac.at/news/seedingTT800.html`. Le problème rencontré est que le TT800 produisait des sorties très semblables même avec des états initiaux différents. Ceci était dû au fait que les état initiaux avaient une distance de Hamming très faible, ce qui fait que le générateur, pour chaque initialisation, passait par des états très semblables et produisait, à peu près, les mêmes valeurs. Soit $\mathbf{x}'_0$ et $\mathbf{x}''_0$, deux états initiaux dont la distance de Hamming entre eux est de un, c'est-à-dire que $\mathbf{x}'_0 = \mathbf{e}_j \oplus \mathbf{x}''_0$ pour une valeur de j. Étant donné la linéarité de la récurrence et les résultats obtenus aux figures 7.7 et 7.8, on peut facilement concevoir que les résultats obtenus par les deux initialisation $\mathbf{x}'_0$ et $\mathbf{x}''_0$ seront très semblables pour un grand

nombre d'itérations. Si le générateur avait été un WELLRNG, le problème aurait été beaucoup moins prononcé.

La propriété d'un générateur de passer d'un état qui a poids de Hamming faible à un autre avec un poids de Hamming elevé est appelé la *capacité de diffusion* du générateur. Un générateur qui a une grande capacité de diffusion aura très peu d'endroits, dans son cycle complet, qui comportent plusieurs états consécutifs avec un faible poids de Hamming. Il s'agit d'une propriété importante pour les générateurs utilisés en cryptographie et pour ce qui est des générateurs linéaires, peu de gens ont senti la nécessité d'étudier cette capacité de diffusion. Pour ce qui est des WELLRNG, nous venons de démontrer, de manière empirique, que leur capacité de diffusion est de loin supérieure à celle des Mersenne twister et des TGFSR.

Figure 7.7 – Moyenne de n premières valeurs générées μ_n.

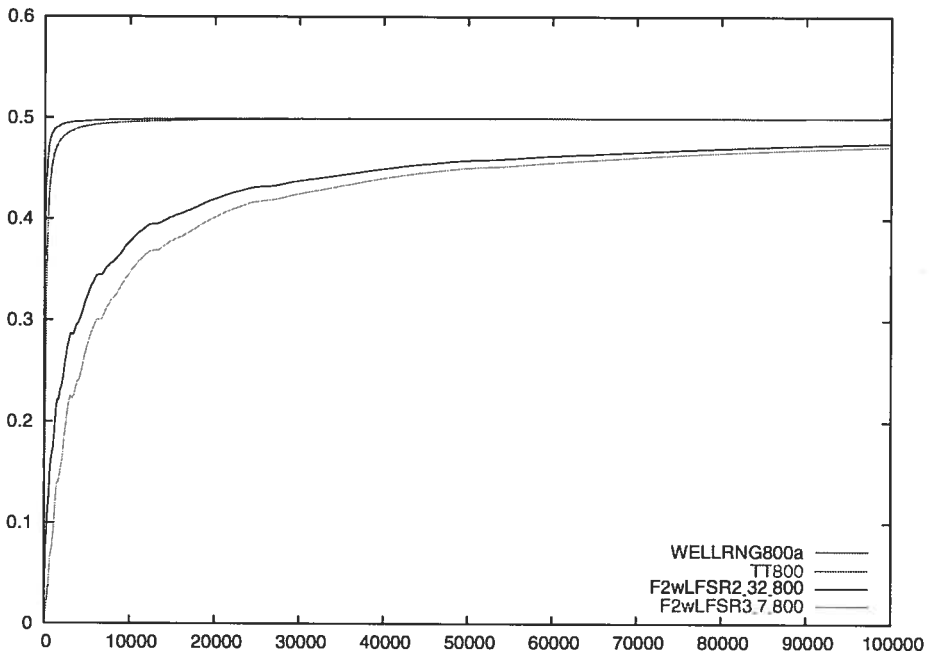


Figure 7.8 – Moyenne mobile $\mu_{n,100}$.

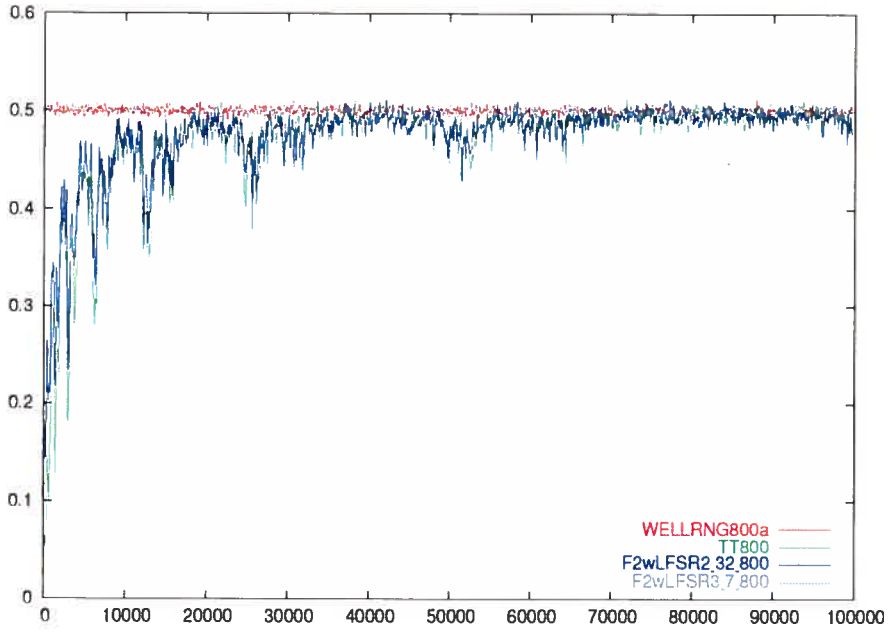


Figure 7.9 – Moyenne de n premières valeurs générées μ_n.

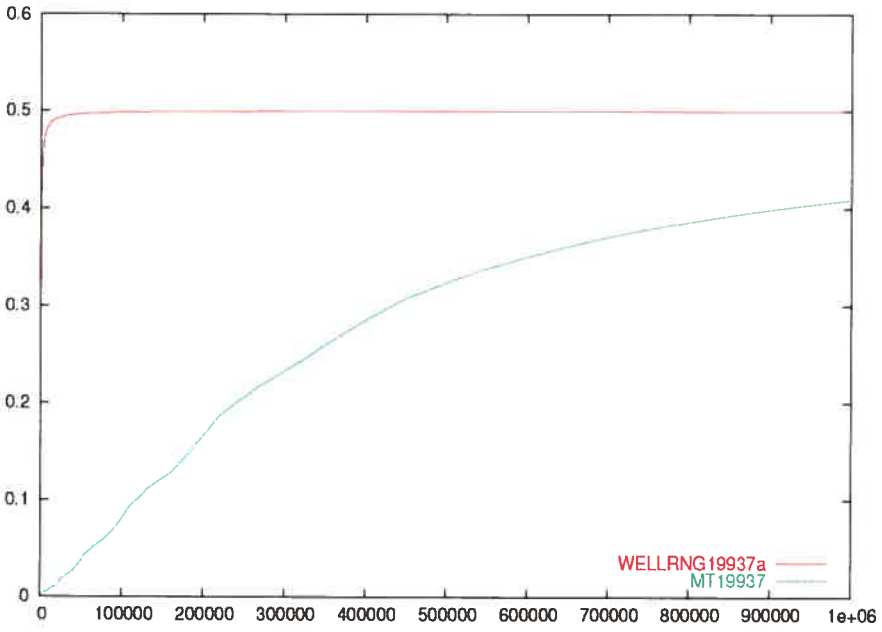


Figure 7.10 – Moyenne mobile $\mu_{n,1000}$.

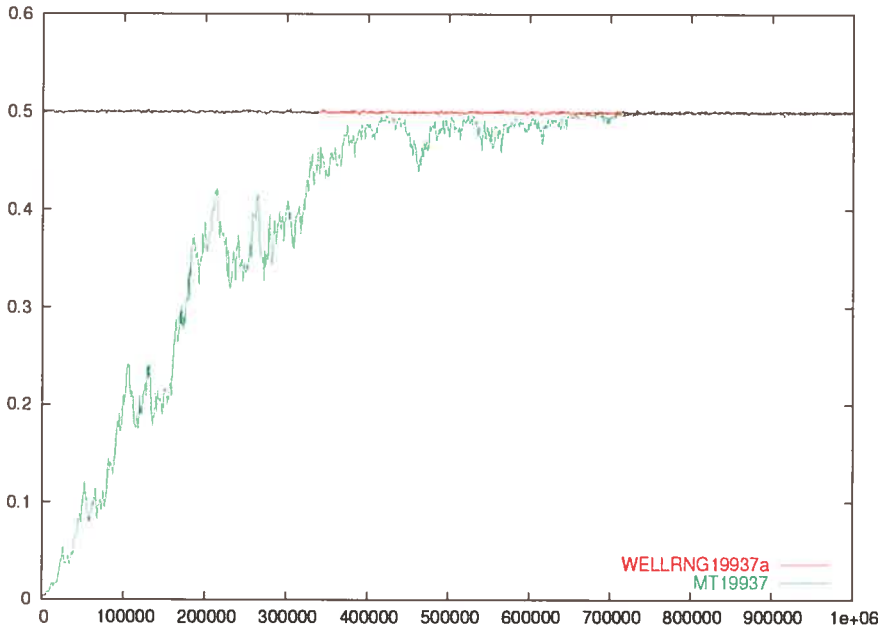
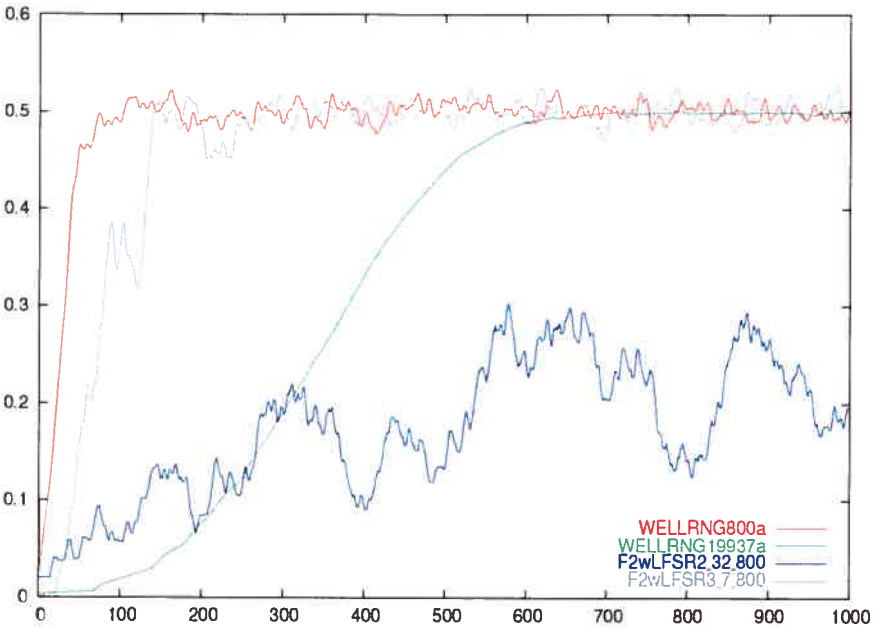


Figure 7.11 – Moyenne mobile $\mu_{n,5}$.



Chapitre 8

Propriétés des générateurs xorshift de Marsaglia

Dans un article récent, George Marsaglia [60] introduit trois nouvelles constructions de générateurs qu'il nomme « xorshift ». Dans cette thèse, nous nommerons ces trois constructions *générateurs xorshift de type I, II et III.* Ces générateurs sont conçus pour être rapides et simples à utiliser. Ils sont en effet très simples et rapides puisque, pour ceux de type I, une seule ligne de code en langage C est suffisante pour les implanter. Les opérations sont simples : très peu de ou-exclusifs bit-à-bit et seulement trois décalages de bits. L'auteur donne plusieurs ensembles de paramètres pour lesquels les générateurs ont la pleine période. De plus, il affirme que tous les 648 ensembles de paramètres donnés résultent en des générateurs de bonne qualité [60] :

> Although I have only tested a few of them, any one of the 648 choices above is likely to provide a very fast, simple, high quality RNG.

Il croit que le fait qu'un générateur basé sur un ensemble de paramètres passe avec succès une batterie de tests statistiques (dans ce cas-ci, sa fameuse batterie DIEHARD [59]) est garant du succès des autres générateurs. La réalité en est tout autre.

Nous avons vérifié l'équidistribution de tous les générateurs proposés et, sans difficulté, nous avons trouvé plusieurs générateurs qui sont mauvais du point de vue de l'équidistribution. Pour ces générateurs, nous avons effectué des tests statistiques et ils en ont échoué plusieurs. Même pour les générateurs qui possèdent une bonne équidistribution, plusieurs ne réussissent pas à passer certains tests statistiques contenus dans les batteries Smallcrush, Crush et Bigcrush de TestU01 [48].

Dans ce chapitre, on présente les trois constructions proposées par Marsaglia [60]. On montre que, dans la liste des générateurs proposés, il y a de la redondance puisque plusieurs paires de récurrences, présentées comme disctinctes, sont en fait la même récurrence. On donne l'équidistribution de tous les générateurs proposés dans l'article de Marsaglia et d'autres qu'il n'a pas identifiés. On donne quelques résultats théoriques sur l'équidistribution des générateurs proposés et la période de certaines variantes. Ensuite, on présente la performance de certains générateurs par rapport à divers tests statistiques.

Notre contribution consistera à démontrer qu'il n'est pas recommandé d'utiliser les générateurs proposés par Marsaglia et que ceux de la même famille de générateurs qui possèdent la meilleure équidistribution ne doivent être utilisés que si la période est grande (et seulement si on ne dispose pas d'autres alternatives). Dans notre recherche, notre but sera de tenter d'obtenir les meilleurs générateurs possibles en n'utilisant que trois décalage de bits. On se rend compte que ces opérations sont trop simples : malgré qu'on obtienne des générateurs qui ont une bonne équidistribution (même meilleure que certains générateurs connus et robustes comme le TT800 [67]), ceux-ci ne réussissent pas à obtenir de bons résultats aux tests statistiques. Ceci constitue une surprise, du moins pour l'auteur, car on aurait pu croire qu'un générateur bien équidistribué et de longue période devrait réussir tous les tests statistiques raisonnables.

8.1 Générateurs de type I

Pour les générateurs de type I, l'état $\mathbf{x}_n$ est un vecteur de w bits et la matrice de transition est

$$X = (I + H_3^c)(I + H_2^b)(I + H_1^a) \tag{8.1}$$

où a, b, c sont des entiers inférieurs à w, $H_1, H_2, H_3 \in \{G, D\}$, la matrice G est telle que $G\mathbf{x} = \mathbf{x} \ll 1$ et la matrice D est telle que $D\mathbf{x} = \mathbf{x} \gg 1$. Les matrices I, G et D sont de dimension $w \times w$ avec leurs éléments dans $\mathbb{F}_2$. La sortie est habituellement donnée par $\mathbf{y}_n = \mathbf{x}_n$.

Les matrices $(I+G^a)$ et $(I+D^a)$ ont quelques propriétés qui sont utiles à explorer afin d'étudier la récurrence (8.1). Tout d'abord, remarquons que $(I + D^a) = P(I + G^a)P^{-1}$ où

$$P = \begin{pmatrix} & & & & 1 \\ & & & 1 & \\ & & \iddots & & \\ & 1 & & & \\ 1 & & & & \end{pmatrix}$$

est une matrice $w \times w$ et que $P^{-1} = P$. De plus, si $H \in \{G, D\}$, alors $(I + H^a)$ est inversible si l'entier a est différent de zéro. Également, on a que $(I+G^a) = P(I+D^a)P$. On dit que deux matrices A et B sont *similaires* s'il existe une matrice C inversible telle que $A = CBC^{-1}$. On note la similarité par $A \sim B$. Il est bien connu que les polynômes caractéristiques de deux matrices similaires sont les mêmes. Soit X_1 et X_2, deux matrices de transitions de générateurs différents. Si $X_1 \sim X_2$ et que le polynôme caractéristique de X_1 est primitif, alors il est en de même pour X_2. Par conséquent, si un générateur basé sur X_1 est de pleine période, alors un autre basé sur X_2 sera de pleine période également.

Nous énonçons quatre propriétés des matrices de la forme $(I + H^a)$ où $H \in$

$\{G, D\}$ qui nous seront utiles pour démontrer la similarité entres certaines matrices de transition X.

Propriété 8.1. *Commutativité. Soit* $H \in \{G, D\}$. *On a* $(I + H^a)(I + H^b) = (I + H^a)(I + H^b)$.

Démonstration. On a $(I+H^a)(I+H^b) = I+H^a+H^b+H^{a+b} = I+H^b+H^a+H^{b+a} = (I + H^b)(I + H^a)$. □

Propriété 8.2. *Soit* $H_1, H_2, H_3 \in \{G, D\}$. *On a* $(I + H_3^c)(I + H_2^b)(I + H_1^a) \sim (I + H_1^a)(I + H_3^c)(I + H_2^b)$

Démonstration. On a $(I + H_3^c)(I + H_2^b)(I + H_1^a) \sim (I + H_1^a)(I + H_3^c)(I + H_2^b)(I + H_1^a)(I + H_1^a)^{-1} = (I + H_1^a)(I + H_3^c)(I + H_2^b)$. □

Propriété 8.3. *Soit* $X = (I + H_3^c)(I + H_2^b)(I + H_1^a)$ *où* $H_1, H_2, H_3 \in \{G, D\}$ *et* H_1, H_2 *et* H_3 *ne sont pas tous la même matrice. On a* $X \sim (I + H_3^c)(I + H_1^a)(I + H_2^b)$

Démonstration. Pour le démontrer, considérons les deux cas possibles : $H_1 = H_2$ et $H_2 = H_3$. Dans le premier cas, la démonstration est triviale car $(I + H_2^b)(I + H_1^a) = (I+H_1^a)(I+H_2^b)$ par la propriété 8.1. Dans le deuxième cas, $(I+H_3^c)(I+H_2^b)(I+H_1^a) = (I + H_2^b)(I + H_3^c)(I + H_1^a) \sim (I + H_3^c)(I + H_1^a)(I + H_2^b)$ où la similarité vient de la propriété 8.2. □

Propriété 8.4. *Soit* $H_i, K_i \in \{G, D\}$ *où* $H_i \neq K_i$ *pour* $i = 1, 2, 3$. *On a* $(I+H_3^c)(I+H_2^b)(I+H_1^a) \sim (I + K_3^c)(I + K_2^b)(I + K_1^a)$.

Démonstration. On a $(I+H_3^c)(I+H_2^b)(I+H_1^a) \sim P(I+H_3^c)PP(I+H_2^b)PP(I+H_1^a)P = (I + K_3^c)(I + K_2^b)(I + K_1^a)$ □

Soit

$$\begin{aligned}
X_1 &= (I+G^c)(I+D^b)(I+G^a) \\
X_2 &= (I+G^a)(I+D^b)(I+G^c) \\
X_3 &= (I+D^c)(I+G^b)(I+D^a) \\
X_4 &= (I+D^a)(I+G^b)(I+D^c) \\
X_5 &= (I+D^b)(I+G^c)(I+G^a) \\
X_6 &= (I+D^b)(I+G^a)(I+G^c) \\
X_7 &= (I+G^b)(I+D^c)(I+D^a) \\
X_8 &= (I+G^b)(I+D^a)(I+D^c).
\end{aligned}$$

Dans [60], on affirme, sans preuve ou argument, pour un triplet (a, b, c), que si la matrice X_1 procure un générateur de pleine période, alors il en sera de même pour les matrices $X_1, \ldots, X_8$. C'est effectivement vrai et nous allons le démontrer.

Par la propriété 8.1, il est facile de voir que $X_5 = X_6$ et $X_7 = X_8$. Donc les générateurs que Marsaglia donnait comme quatre générateurs distincts ne constituent en fait que deux générateurs distincts. En utilisant les propriétés 8.1–8.4, il est très simple de montrer que les matrices $X_1, \ldots, X_8$ sont toutes similaires les unes par rapports aux autres (à noter que la similarité est une relation d'équivalence, donc transitive). Au tableau 8.1, les matrices de similarité C telles que $X_i = CX_1C^{-1}$, pour $i = 1, \ldots, 8$ sont données. Pour un triplet (a, b, c), puisqu'elles sont toutes similaires entre elles, si X_1 a un polynôme caractéristique primitif, alors toutes les matrices $X_1, \ldots, X_8$ ont le même polynôme caractéristique primitif. Autrement dit, pour un triplet (a, b, c), si X_1 donne un générateur de pleine période $2^w - 1$, alors les générateurs basés sur $X_2, \ldots, X_8$ seront aussi de pleine période.

Dans [60], l'auteur n'a pas remarqué que les générateurs basés sur les matrices

$$\begin{aligned} X_9 &= (I+G^c)(I+G^a)(I+D^b) \\ X_{10} &= (I+G^a)(I+G^c)(I+D^b) \\ X_{11} &= (I+D^c)(I+D^a)(I+G^b) \\ X_{12} &= (I+D^a)(I+D^c)(I+G^b) \end{aligned}$$

ont aussi la même période que celui basé sur X_1 puisque $X_1 \sim X_i$ pour $i = 9, \ldots, 12$. Par contre, on a que $X_9 = X_{10}$ et $X_{11} = X_{12}$ par la propriété 8.1.

Proposition 8.1. *Les matrices $X_1, \ldots, X_{12}$ sont toutes similaires entre elles et ont toutes la même période pour un triplet (a, b, c) donné.*

On pourrait aussi considérer une récurrence basée sur $X_{13} = (I+G^c)(I+G^b)(I+G^a)$. Dans ce cas, si un générateur basé sur X_{13} est de pleine période, alors il en sera de même pour celui basé sur $X_{14} = (I+D^c)(I+D^b)(I+D^a)$ par la propriété 8.4 Par contre, le polynôme caractéristique de X_{13} sur $\mathbb{F}_2$ est toujours $(1+z)^w \in \mathbb{F}_2[z]$. Ce polynôme n'est pas irréductible. La matrice X_{13} est formée de 1 dans la diagonale principale et les éléments au-dessus de celle-ci sont tous nuls. Il est facile de calculer le polynôme caractéristique de cette matrice qui est $\det(X_{13} - I_w z) = (1+z)^w$. On peut voir, de la même manière, que $\det(X_{14} - I_w z) = (1+z)^w$. Il est donc inutile de chercher des générateurs de pleine période qui utilisent la matrice X_{13} ou X_{14} : il n'en existe aucun.

Seules les matrices X_i dont $i \in \{1, 2, 3, 4, 5, 7, 9, 11\}$ donnent véritablement des générateurs distincts. Par contre, nous allons montrer que pour un choix de triplet (a, b, c) les générateurs basés sur X_3, X_4, X_7 et X_{11} ont toujours la même équidistribution, même chose pour ceux basés sur X_5 et X_9. Pour montrer cela, mentionnons qu'appliquer le tempering $\mathbf{y}_n = B\mathbf{x}_n$ où B est une matrice triangulaire avec des 1 dans la diagonale, des zéros au-dessus de la diagonale et le reste des valeurs sont dans $\mathbb{F}_2$, ne change pas la q-valeur (ou l'équidistribution) de l'ensemble de points

$\Psi_t(\mathbf{X}, B, w)$ peu importe la dimension t. En fait, ce tempering ne fait qu'appliquer un mixage linéaire (voir chapitre 6) à l'ensemble de points $\Psi_t(\mathbf{X}, I, w)$ et il est connu que ce type de transformation ne change pas la q-valeur de l'ensemble de points résultant [63].

Soit la récurrence $\mathbf{x}_n = X_9\mathbf{x}_{n-1}$ et le tempering $\mathbf{y}_n = B\mathbf{x}_n = (I + D^b)\mathbf{x}_n$. On remarque que ce tempering est une forme de mixage linéaire. À partir de ces deux équations, on peut obtenir la récurrence

$$\mathbf{y}_n = BX_9B^{-1}\mathbf{y}_{n-1} = (I + D^b)(I + G^c)(I + G^a)(I + D^b)(I + D^b)^{-1}\mathbf{y}_{n-1} = X_5\mathbf{y}_{n-1}.$$

On voit imédiatement que l'ensemble de points $\Psi_t(\mathbf{X}_5, I, w)$ est le même que l'ensemble de points $\Psi_t(\mathbf{X}_9, I, w)$ auquel on a appliqué le mixage linéaire décrit par B. Ainsi, $\Psi_t(\mathbf{X}_9, I, w)$ et $\Psi_t(\mathbf{X}_5, I, w)$ ont tous les deux la même q-valeur et la même équidistribution, peu importe la dimension t.

On peut montrer, de la même manière qu'on l'a fait pour X_5 et X_9, que les ensembles de points $\Psi_t(\mathbf{X}_3, I, w)$, $\Psi_t(\mathbf{X}_4, I, w)$, $\Psi_t(\mathbf{X}_7, I, w)$ et $\Psi_t(\mathbf{X}_{11}, I, w)$ ont la même q-valeur pour une dimension t donnée.

Pour résumer cette section, on pourrait dire que seuls X_1, X_2, X_3, X_4, X_5, X_7, X_9 et X_{11} permettent de construire des générateurs distincts. Pour un triplet (a, b, c), si une de ces matrices procure un générateur de pleine période, alors elles procurent toutes des générateurs de pleine période. Également, pour un triplet (a, b, c), pour vérifier l'équidistribution de tous ces générateurs, il suffit de vérifier ceux basés sur X_1, X_2, X_3 et X_5. Le tableau 8.1 résume l'essentiel de cette section et donne les matrices de similarité C telles que $X_i = CX_1C^{-1}$ pour $i = 1, \ldots, 12$.

Tableau 8.1 – Liste des matrices X_j qui procurent des générateurs de type I de même période pour un triplet (a, b, c).

X	C	Notes
X_1	I	
X_2	$(I + G^a)(I + G^c)$	
X_3	P	
X_4	$(I + D^a)(I + D^c)P$	Même équidistribution que X_3
X_7	$(I + D^c)P$	Même équidistribution que X_3
X_{11}	$(I + D^a)P$	Même équidistribution que X_3
X_5	$(I + D^b)(I + G^a)$	
X_9	$(I + G^a)$	Même équidistribution que X_5
X_6	$(I + D^b)(I + G^a)$	Même récurrence que X_5
X_8	$(I + D^c)P$	Même récurrence que X_7
X_{10}	$(I + G^a)$	Même récurrence que X_9
X_{12}	$(I + D^a)P$	Même récurrence que X_{11}

8.2 Générateurs de type II

Les générateurs de type II ont pour état le vecteur $\mathbf{x}_n = (\mathbf{v}_n^\mathsf{T}, \mathbf{v}_{n-1}^\mathsf{T}, \ldots, \mathbf{v}_{n-r+1}^\mathsf{T})^\mathsf{T}$ et la récurrence est

$$\mathbf{v}_n = A\mathbf{v}_{n-m} \oplus B\mathbf{v}_{n-r}. \tag{8.2}$$

où A et B sont formés d'un produit de matrices de la forme $(I + H^d)$ où $H \in \{G, D\}$, $0 < d < w$ et $AB = (I + H_3^c)(I + H_2^b)(I + H_1^a)$. La matrice de transition est

$$X = \begin{pmatrix} 0 & 0 & \ldots & A & \ldots & 0 & B \\ I & & & & & & \\ & I & & & & & \\ & & \ddots & & & & \\ & & & & & I & \end{pmatrix}.$$

On remarque que les générateurs de type II utilisent la méthode matricielle à récurrence multiple (voir section 2.3.6) et que le polynôme caractéristique de la récurrence sur $\mathbb{F}_2$ est donné par $P(z) = \det(I_w z^r + Az^{r-m} + B) \in \mathbb{F}_2[z]$. La sortie est habituellement $\mathbf{y}_n = \mathbf{v}_n$. Pour identifier un générateur de type II, on le fera avec le triplet (A, B, m). Si on trouve un triplet (A, B, m) tel que la récurrence (8.2) est de pleine période, alors, pour une matrice C de similarité donnée, le triplet (CAC^{-1}, CBC^{-1}, m), qui décrit un générateur basé sur la récurrence

$$\mathbf{y}_n = CAC^{-1}\mathbf{y}_{n-m} \oplus CBC^{-1}\mathbf{y}_{n-r}, \tag{8.3}$$

donne aussi un générateur de pleine période. Cette dernière récurrence donne la séquence obtenue en appliquant le tempering $\mathbf{y}_n = C\mathbf{v}_n$ au générateur défini par la récurrence (8.2). Comme discuté à la section précédente, si la matrice de similarité correspond à appliquer un mixage linéaire, alors les deux générateurs ont la même équidistribution pour une dimension t donnée.

Dans [60], on mentionne seulement le cas $A = A_1 = (I + D^b)(I + G^a)$ et $B = B_1 =$

$(I + D^c)$ et $m = 1$. Dans ce cas, la récurrence est

$$\mathbf{v}_n = (I + D^b)(I + G^a)\mathbf{v}_{n-1} \oplus (I + D^c)\mathbf{v}_{n-r}. \tag{8.4}$$

Soit A_i et B_i pour $i = 2, 3, 4$, les matrices telles que définies au tableau 8.2. Nous pouvons montrer que, pour (A_1, B_1, m), si le triplet (a, b, c) procure un générateur de pleine période, il en sera de même pour les triplets (A_2, B_2, m), (A_3, B_3, m) et (A_4, B_4, m). Pour ce faire, il suffit de trouver la matrice de similarité C qui permet de « convertir » le générateur basé sur (A_1, B_1, m) en celui basé sur (A_i, B_i, m) pour $i = 2, 3, 4$. Au tableau 8.2, on liste les couples (A_i, B_i) et donne les matrices de similarité. Le lecteur peut facilement vérifier que les matrices C données sont les bonnes.

Tableau 8.2 – Matrices A_i et B_i qui donnent une récurrence (8.2) de pleine période si et seulement si A_1 et B_1 en donnent une également.

i	A_i	B_i	C
2	$(I + G^b)(I + D^a)$	$(I + G^c)$	P
3	$(I + G^a)(I + D^b)$	$(I + D^c)$	$(I + D^b)$
4	$(I + D^a)(I + G^b)$	$(I + G^c)$	$(I + G^b)P$

On remarque que la matrice de similarité pour (A_3, B_3, m) est $(I+D^b)$, qui est une matrice de mixage linéaire. Ceci implique que les générateurs basés sur (A_3, B_3, m) ont la même équidistribution que ceux basés sur (A_1, B_1, m), quelque soit la dimension t. Par contre, ceci ne veut pas dire qu'ils ne constituent pas deux générateurs distincts.

On pourrait aussi dire, par le même raisonnement que l'on vient de faire, qu'en trouvant un triplet (a, b, c) pour lequel un générateur basé sur la récurrence (8.2) avec $A = B_1$ et $B = A_1$ est de pleine période, on se trouve avoir trouvé trois autres générateurs de pleine période, à savoir ceux basés sur (B_2, A_2, m), (B_3, A_3, m) et (B_4, A_4, m).

Dans la même logique, si $A = (I+H^a)(I+H^b)$ et $B = (I+K^c)$, où $H, K \in \{G, D\}$ et $H \neq K$, et si (A, B, m) procure une récurrence (8.2) de pleine période, il en sera de même pour celle que procure $((I + K^a)(I + K^b), (I + H^c), m)$ par la matrice de similarité $C = P$.

Dans le cas où $AB = (I + H^a)(I + H^b)(I + H^c)$ et $H \in \{G, D\}$, le polynôme caractéristique de la matrice de transition X est donné par $\det(X - I_{rw}z) = \det(I_w z^r + Az^{r-m} + B)$. Puisque les matrices A et B sont des matrices triangulaires avec des 1 dans la diagonale principale et les éléments non nuls sont du même côté de la diagonale principale, on observe que $\det(X - I_{rw}z) = (z^r + z^{r-m} + 1)^w$. Ce polynôme n'est pas irréductible et les générateurs basés sur ce type de matrice ne sont jamais de pleine période $2^{rw} - 1$.

Si $A = I$ et $B = X_1$ (où X_1 est défini à la section 8.1), alors on remarque que la récurrence (8.2) est celle d'un TGFSR. Si (I, X_1, m) procure un générateur de pleine période pour un triplet (a, b, c), alors il en sera de même pour (X_i, I, m), pour $i = 2, \ldots, 12$, puisque toutes les matrices X_i sont similaires entre elles (voir le tableau 8.1 pour les matrices de similarité) et que $CIC^{-1} = I$.

Dans le cas où on considère (X_i, I, m), $i = 1, \ldots, 12$, il n'existe aucun générateur de pleine période. Le théorème suivant, dont la démonstration s'inspire de celle du théorème 2 de [66], le confirme.

Théorème 8.1. *Soit A, une matrice $w \times w$ avec ses éléments dans $\mathbb{F}_p$ où p est un nombre premier. La récurrence linéaire sur $\mathbb{F}_p^w$*

$$\mathbf{x}_n = A\mathbf{x}_{n-m} + \mathbf{x}_{n-r} \tag{8.5}$$

n'est pas de période $p^{rw} - 1$, quelque soit A.

Démonstration. Soit $Q(z) = \det(A - I_w z)$, le polynôme caractéristique de A sur $\mathbb{F}_p$ et supposons que $Q(z)$ soit irréductible sur $\mathbb{F}_p$. Soit $\eta \in \mathbb{F}_{p^w}$, une racine de $Q(z)$. Il

est bien connu que les éléments $1, \eta, \ldots, \eta^{w-1}$ forment une base de $\mathbb{F}_{p^w}$ sur $\mathbb{F}_p$.

On peut montrer que pour un vecteur non nul $\mathbf{t} \in \mathbb{F}_p^w$, l'homomorphisme

$$\begin{aligned} \phi : \mathbb{F}_{p^w} &\rightarrow \mathbb{F}_p^w \\ \eta^l &\mapsto A^l \mathbf{t}, \end{aligned}$$

où $0 \neq \mathbf{t} \in \mathbb{F}_p^w$ est fixé, est aussi un isomorphisme (voir la preuve du théorème 2 de [66]). Appliquons l'inverse de l'isomorphisme ϕ à la récurrence (8.5). On obtient la récurrence linéaire dans $\mathbb{F}_p^w$

$$\phi^{-1}(\mathbf{x}_n) = \phi^{-1}(A\mathbf{x}_{n-m}) + \phi^{-1}(\mathbf{x}_{n-r})$$

qui peut se réécrire comme la récurrence linéaire dans $\mathbb{F}_{p^w}$

$$x_n = \eta x_{n-m} + x_{n-r} \tag{8.6}$$

où $x_n = \phi^{-1}(\mathbf{x}_n) \in \mathbb{F}_{p^w}$, car $\phi^{-1}(A\mathbf{x}_{n-m}) = \eta x_{n-m}$ (voir le théorème 2 de [66]). Notons que, puisque les récurrences (8.5) et (8.6) sont reliées entre elles par un isomorphisme, elles ont la même période. Le polynôme caractéristique sur $\mathbb{F}_{p^w}$ de la récurrence (8.6) est

$$P(z) = z^r - \eta z^{r-m} - 1 \in \mathbb{F}_{p^w}[z]$$

Par le théorème 3.18 de [56], puisque $(-1)^r P(0) = (-1)^{r+1}$ n'est pas un élément primitif de $\mathbb{F}_{p^w}$, le polynôme $P(z)$ n'est pas primitif sur $\mathbb{F}_{p^w}$. Ceci implique que la récurrence (8.6) n'est pas de période $p^{rw} - 1$ et, par conséquent, ni la récurrence (8.5).

Maintenant, supposons que $Q(z)$ ne soit pas irréductible. Soit $q = r - m$. Par le théorème 2.3, le polynôme caractéristique de la récurrence (8.5) sur $\mathbb{F}_p$ est donné par

$$\begin{aligned} P(z) = \det(z^r I_w - z^q A - I_w) &= \det(z^q(z^m I_w - A - z^{-q} I_w)) \\ &= \det(z^q I_w)\det((z^m - z^{-q})I_w - A) \\ &= z^{wq} Q(z^m - z^{-q}) \in \mathbb{F}_p[z] \end{aligned}$$

Remarquons que $Q(z^m - z^{-q})$ n'est pas dans $\mathbb{F}_p[z]$ et que, par conséquent, on ne peut déduire, à ce stade, si $P(z)$ est irréductible sur $\mathbb{F}_p$ ou non. Par contre, $Q(z^m - z^{-q}) \in \mathbb{L}_p$. Puisque que $Q(z)$ n'est pas irréductible, il se décompose en $c > 1$ facteurs $Q_i(z) \in \mathbb{F}_p[z]$, chacun de degré $d_i > 0$.

Soit $h(z) = h_n z^n + h_{n-1} z^{n-1} + \cdots \in \mathbb{L}_p$. On définit la fonction $p(h(z)) = \min\{i : h_i \neq 0\}$. Pour que $h(z) \in \mathbb{L}_p$ soit dans $\mathbb{F}_p[z]$, il faut que $p(h(z)) \geq 0$. Constatons que, puisque $Q(z)$ est de degré w, on a $w = d_1 + \cdots + d_c$ et que $h(Q_i(z^m + z^{-q})) = -qd_i$. Soit $\bar{Q}_i(z) = z^{qd_i} Q_i(z^m + z^{-q})$. On a $h(\bar{Q}_i(z)) = 0$, ce qui implique que $\bar{Q}_i(z) \in \mathbb{F}_p[z]$. Avec ces éléments en mains, développons

$$\begin{aligned} z^{wq} Q(z^m - z^{-q}) &= z^{wq} \prod_{i=1}^{c} Q_i(z^m - z^{-q}) \\ &= \prod_{i=1}^{c} z^{qd_i} Q_i(z^m - z^{-q}) \\ &= \prod_{i=1}^{c} \bar{Q}_i(z). \end{aligned}$$

La dernière égalité implique que le polynôme caractéristique sur $\mathbb{F}_p$ de la récurrence (8.5) n'est pas irréductible sur $\mathbb{F}_p$ car il se décompose en facteurs $\bar{Q}_i(z) \in \mathbb{F}_p[z]$, $i = 1, \ldots, c$. Ainsi, si $Q(z)$ n'est pas irréductible sur $\mathbb{F}_p$, alors le polynôme $P(z)$ ne l'est pas non plus et la récurrence (8.5) n'a pas la période maximale $2^{rw} - 1$, ce qui complète la démonstration. □

Le tableau 8.2 donne toutes les combinaisons possibles (et intéressantes) de générateurs de type II. Elles sont regroupées de telle manière que celles qui procurent des générateurs de même période se retrouvent ensemble.

On a discuté du choix des matrices A et B, mais pas du choix de m. Pour certaines valeurs de r, ce ne sont pas toutes les valeurs de m qui peuvent produire des générateurs de pleine période. Quand $A = I$ et B est une matrice de plein rang, la récurrence (8.2) est celle d'un TGFSR. Dans ce cas, on a les résultats suivants qui pro-

viennent tous de [66]. Le polynôme caractéristique sur $\mathbb{F}_2$ de la récurrence est primitif seulement si $Q(z)$, le polynôme caractéristique de la matrice B, est irréductible sur $\mathbb{F}_2$. Si c'est le cas, la récurrence est de période $2^{rw}-1$ si et seulement si $z^r + z^{r-m} + \eta$ est primitif sur $\mathbb{F}_{2^w}$ où η est une racine de $Q(z)$. Pour déterminer si le polynôme n'est pas irréductible, la proposition 1 de [66], que l'on donne comme la proposition 8.2, s'applique.

Proposition 8.2. *Soit $\eta \in \mathbb{F}_{2^w}$. Le polynôme $z^r + z^q + \eta \in \mathbb{F}_{2^w}[z]$ n'est pas irréductible si une des conditions suivantes est vraie :*

1. $r \equiv \pm 3 \bmod 8$, *w est impair, q est pair et q ne divise pas $2r$.*
2. $r \equiv \pm 3 \bmod 8$, *w est impair, q est impair et $(r-q)$ ne divise pas $2r$.*
3. *r est pair et q est pair.*
4. *r est pair, $r \neq 2q$ et $rq \equiv 0, 2 \bmod 8$.*
5. *r est pair, $r \neq 2q$, $rq \equiv -2, 4 \bmod 8$ et w est pair.*

Dans le cas où $A \neq I$ et $B \neq I$, nous émettons une conjecture qui donne une condition suffisante pour que la récurrence (8.2) ne soit pas de pleine période.

Conjecture 8.1. *Soit la récurrence (8.2) où $A \neq I$ et $B \neq I$. Le polynôme caractéristique sur $\mathbb{F}_2$ de la récurrence n'est pas primitif si m et $(r-m)$ sont pairs.*

Pour appuyer cette conjecture, dans notre recherche de bons générateurs de type II avec $r = 4$, $r = 8$ et $r = 12$, nous n'avons trouvé aucun générateur dont le polynôme est primitif pour lequel $(r-m)$ est pair. Malgré nos efforts, nous n'avons pas pu démontrer cette conjecture.

Tableau 8.3 – Liste des combinaisons (A_i, B_i) regroupées en générateurs de type II de même polynôme caractéristique.

i	A_i	B_i	C	Notes
1	$(I+D^b)(I+G^a)$	$(I+D^c)$	I	Même équid. que $i=3$
2	$(I+G^b)(I+D^a)$	$(I+G^c)$	P	
3	$(I+G^a)(I+D^b)$	$(I+D^c)$	$(I+D^b)$	Même équid. que $i=1$
4	$(I+D^a)(I+G^b)$	$(I+G^c)$	$(I+G^b)P$	
5	$(I+D^c)$	$(I+D^b)(I+G^a)$	I	Même équid. que $i=7$
6	$(I+G^c)$	$(I+G^b)(I+D^a)$	P	
7	$(I+D^c)$	$(I+G^a)(I+D^b)$	$(I+D^b)$	Même équid. que $i=5$
8	$(I+G^c)$	$(I+D^a)(I+G^b)$	$(I+G^b)P$	
9	$(I+D^a)(I+D^b)$	$(I+G^c)$	I	
10	$(I+G^a)(I+G^b)$	$(I+D^c)$	P	
11	$(I+G^c)$	$(I+D^a)(I+D^b)$	I	
12	$(I+D^c)$	$(I+G^a)(I+G^b)$	P	
13–14	I	X_j, $j\in\{1,2\}$	Voir	
15–16	I	X_j, $j\in\{5,9\}$	tableau	Même équidistribution
17–20	I	X_j, $j\in\{3,4,7,11\}$	8.1	Même équidistribution

8.3 Générateurs de type III

Les générateurs de type III ont pour état le vecteur $\mathbf{x}_n = (\mathbf{v}_n^\mathsf{T}, \mathbf{v}_{n-1}^\mathsf{T}, \ldots, \mathbf{v}_{n-r+1}^\mathsf{T})^\mathsf{T}$ et la récurrence est

$$\mathbf{v}_n = \sum_{i=1}^{r} (I + H_i^{a_i}) \mathbf{v}_{n-i}$$

où $H_i \in \{G, D\}$, $1 \leq i \leq r$. Dans [60], l'auteur se limite au cas où $H_i = G$ si i est pair, sinon $H_i = D$.

La matrice de transition est donc

$$X = \begin{pmatrix} (I + H_1^{a_1}) & (I + H_2^{a_2}) & \ldots & (I + H_r^{a_r}) \\ I & & & \\ & I & & \\ & & \ddots & \\ & & & I \end{pmatrix}.$$

On remarque que pour les générateurs de type III, le nombre de décalage de bits n'est pas restreint à trois, mais pourrait aller jusqu'à r (en supposant que certaines valeurs de a_i pourraient être nulles). Ce type de générateur n'est pas très intéressant du point de vue de l'implantation d'un générateur de très longue période puisque le temps d'exécution pour générer une valeur dépend directement de la longueur de la période. Le nombre d'opérations en dans $O(r)$, donc plus r est grand, plus le générateur sera lent. La sortie, pour ce type de générateur est donnée par $\mathbf{y}_n = \mathbf{v}_n$.

Pour le rendre intéressant du point de vue de la vitesse (et aussi pour respecter l'objectif qu'on s'était fixé au début du chapitre, soit de faire le mieux possible avec seulement trois décalage de bits), nous allons considérer le cas où il n'y a que trois valeurs de a_i qui sont non nulles pour $i = 1, \ldots, r$. Dans ce cas, définissons la récurrence

$$\mathbf{v}_n = (I + H_1^a)\mathbf{v}_{n-m_1} \oplus (I + H_2^b)\mathbf{v}_{n-m_2} \oplus (I + H_3^c)\mathbf{v}_{n-r} \tag{8.7}$$

où $H_i \in \{G, D\}$, $1 \leq i \leq 3$ et $1 \leq m_1, m_2 < r$. Pour les générateurs de type III, dénotons par $(H_1, H_2, H_3, a, b, c, m_1, m_2)$ le générateur défini par la récurrence (8.7) et la sortie $\mathbf{y}_n = \mathbf{v}_n$.

Si $(H_1, H_2, H_3, a, b, c, m_1, m_2)$ est de pleine période $2^{rw} - 1$, alors $(K_1, K_2, K_3, a, b, c, m_1, m_2)$ l'est aussi, où $K_i \neq H_i$ et $K_i \in \{G, D\}$. Ceci se démontre avec la matrice de similarité P.

8.4 Recherche de bons générateurs

Nous avons effectué une recherche exhaustive de tous les générateurs de type I, de type II avec $r = 2, 3, 4, 5, 8, 12, 25$, ainsi ceux de type III qui utilisent la récurrence (8.7) avec $r = 3, 4, 5, 8, 12, 25$. Nous avons recensé, de manière exhaustive, tous ceux qui sont de pleine période. Pour tous ces générateurs, nous avons calculé la valeur de $V = \sum_{\ell=1}^{32} \Delta_\ell$, qui est un critère d'uniformité basé sur l'équidistribution. Plus la valeur de V est petite, plus le générateur est jugé uniforme.

Pour ceux qui ont donné la meilleure valeur de V, nous avons effectué des tests statistiques. La première batterie de tests effectuée est « Smallcrush » de la bibliothèque TestU01 [48]. Si les résultats sont satisfaisants, nous effectuons la batterie « Crush » de la même bibliothèque. Si nécessaire, nous effectuons également la batterie « Bigcrush », également de TestU01. Le tableau 8.4 donne les tests auxquels nous ferons référence dans ce chapitre. Ils sont nommés selon le nom de la fonction qui exécute le test. Les paramètres de chacune des fonctions sont également donnés. Nous ferons référence à ces tests en indiquant le numéro du test qui est donné dans la première colonne du tableau 8.4. Pour plus de détails sur ce que chaque test fait spécifiquement, nous référons le lecteur au guide de TestU01 [48].

Dans les prochaines sous-sections, nous exposons les résultats de la recherche de

bons générateurs obtenus pour chacun des types de générateur.

8.4.1 Générateurs de type I

Dans [60], l'auteur a trouvé tous les ensembles de triplets (a, b, c), où $a < c$, tels que la matrice X_1 produit un générateur de pleine période avec $w = 32$. Il en a trouvé 81. Nous avons vérifié qu'ils étaient tous de pleine période et qu'il n'y en avait pas d'autres. Les résultats de Marsaglia sont corrects, mais nous avons décelé une erreur qu'on pourrait interpréter comme une erreur de retranscription à la page 2 de [60]. Ainsi, Marsaglia affirme que le triplet $(9, 5, 1)$ donne un générateur de pleine période, alors que ce n'est pas le cas. Par contre, il ne donne pas le triplet $(9, 5, 14)$ qui, lui, donne un générateur de pleine période. Nous avons aussi vérifié tous les triplets donnés par Marsaglia pour $w = 64$. Il n'y a aucune omission ou erreur.

Nous avons vérifié l'équidistribution de tous ces générateurs. Pour chacun, nous avons calculé $V = \sum_{\ell=1}^{32} \Delta_\ell$. Les résultats de ces calculs sont donnés aux tableaux 8.5 et 8.6 pour le cas où $w = 32$ et aux tableaux 8.7–8.13 pour $w = 64$. En analysant les tableaux 8.5 et 8.6 qui donnent les résultats pour $w = 32$, on remarque que ce ne sont pas tous les triplets (a, b, c) qui donnent des valeurs de V petites. On remarque que, souvent, les pires générateurs sont ceux pour lesquelles les valeurs de a, b et c sont près de 1 ou 31. Par exemple, le triplet $(1, 27, 27)$ donne les pires générateurs. La moyenne de la valeur de V donnée par les générateurs X_1, X_2, X_3 et X_5 formés par ce triplet est 48.8. La pire valeur de V possible est 56 et ceci n'arrive que lorsque le générateur est équidistribué en 32 dimensions pour le premier bit et en une dimension pour les bits subséquents, c'est-à-dire quand $t_1 = 32$ et $t_\ell = 1$ pour $\ell = 2, \ldots, 32$. Le triplet $(15, 1, 29)$ donne aussi de très mauvaises valeurs de V. Les triplets qui donnent la meilleur moyenne de V, $(5, 21, 12)$ et $(11, 17, 13)$ ont des valeurs de V près de 4. Il y a d'autres triplets qui ont beaucoup de variance dans les valeurs de V produites. Par exemple, le triplet $(7, 1, 9)$ donne des générateurs dont $V = 56$ et d'autres qui

Tableau 8.4 – Liste des tests utilisés au chapitre 8 et de leur p-valeur associée.

No	Nom du test	Paramètres	p-valeur
1	`smarsa_BirthdaySpacings`	$N = 100$, $n = 5 \times 10^6$, $r = 0$, $d = 2^{30}$, $t = 2$, $p = 1$	p_1
2	`swalk_RandomWalk1`	$N = 100$, $n = 5 \times 10^7$, $r = 0$, $s = 2^{30}$, $L_0 = 90$, $L_1 = 90$ (statistique H)	p_2
3	`sknuth_MaxOft`	$N = 10$, $n = 10^7$, $r = 0$, $d = 5 \times 10^5$, $t = 5$	p_3
4	`sknuth_MaxOft`	$N = 1$, $n = 2 \times 10^6$, $r = 0$, $d = 10^5$, $t = 6$	p_4
5	`sstring_HammingIndep`	$N = 1$, $n = 10^6$, $r = 0$, $s = 30$, $L = 300$, $d = -1$	p_5
6	`sstring_HammingIndep`	$N = 1$, $n = 10^6$, $r = 20$, $s = 10$, $L = 30$, $d = -2$	p_6
7	`smarsa_BirthdaySpacings`	$N = 5$, $n = 10^7$, $r = 14$, $d = 2^8$, $t = 8$, $p = 1$	p_7
8	`smarsa_MatrixRank`	$N = 1$, $n = 10^6$, $r = 20$, $s = 10$, $L = 200$, $k = 200$	p_8
9	`smarsa_MatrixRank`	$N = 1$, $n = 50000$, $r = 20$, $s = 10$, $L = 300$, $k = 300$	p_9
10	`smarsa_MatrixRank`	$N = 1$, $n = 10^6$, $r = 20$, $s = 10$, $L = 90$, $k = 90$	p_{10}

ont $V = 1$. Ce triplet produit les deux extrêmes. Nous pouvons aussi remarquer les mêmes tendances dans le cas où $w = 64$.

Pour $w = 32$ ou $w = 64$, si on compare la colonne de X_3 avec les autres colonnes, il semble que l'uniformité de ces générateurs soit, de façon générale, moins bonne que celle des générateurs qui utilisent X_1, X_2 ou X_5. On remarque que X_3 est le produit de deux matrices de la forme $(I+D^d)$ et d'une matrice $(I+G^g)$, alors que les autres sont le produit de deux matrices de la forme $(I+G^g)$ et d'une matrice $(I+D^d)$. Remarquons également que pour le critère d'équidistribution, l'uniformité au niveau des premiers bits de la sortie est très importante. Si elle n'est pas atteinte, alors l'équidistribution des bits subséquents ne sera pas bonne non plus. La multiplication $\mathbf{y} = (I + D^d)\mathbf{x}$ fait en sorte que les d premiers bits de $\mathbf{y}$ sont exactement les mêmes que ceux de $\mathbf{x}$. Ainsi, cette transformation linéaire n'est pas très utile pour faire varier les bits les plus significatifs. Par contre, la multiplication $\mathbf{y} = (I + G^g)\mathbf{x}$ permet d'obtenir un vecteur $\mathbf{y}$ pour lequel les $w - g$ premiers bits sont possiblement différents de ceux de $\mathbf{x}$. Ainsi, la multiplication $\mathbf{y} = (I + G^d)\mathbf{x}$ est plus efficace que $\mathbf{y} = (I + D^d)\mathbf{x}$ pour mélanger les bits les plus significatifs. Par contre, comme il a été écrit précédemment, il n'est pas possible d'avoir un générateur pour lequel X est le produit de trois matrices de la forme $(I + G^g)$ puisque le polynôme caractéristique de ces matrices n'est jamais irréductible.

Nous avons testé tous les générateurs basés sur la matrice X_1 contenus dans les tableaux 8.5 et 8.6 avec la batterie de tests Smallcrush de la bibliothèque TestU01. Quand $w = 32$, la plupart échoue plusieurs tests malgré le fait que les tests contenus dans Smallcrush ne soient pas des plus exigeants. C'est une des deux raisons principales pour lesquelles on ne peut recommander aucun des générateurs de type I avec $w = 32$ (pas seulement ceux avec X_1, mais aussi ceux avec $X_2, \ldots, X_{12}$). L'autre raison est leur période trop courte : $2^{32} - 1$ est insuffisant pour n'importe quelle application de simulation le moindrement sérieuse.

Pour les générateurs de type I avec $w = 64$, le problème de la période est moins criant (malgré qu'une période de $2^{64} - 1$ est considérée petite par rapport à celle d'autres générateurs modernes). On a testé tous les générateurs basés sur X_1 qui sont donnés dans les tableaux 8.7–8.13. La plupart de ces générateurs ne passent pas la batterie de tests Smallcrush. En fait, seulement 130 générateurs sur les 275 générateurs passent avec succès tous les tests de la batterie Smallcrush. Ceux qui performaient le mieux par rapport à l'équidistribution font partie de ceux qui ont le mieux performé par rapport à cette batterie, comme on aurait pu s'y attendre. Par exemple, les 10 premiers triplets du tableau 8.7 donnent des générateurs basés sur X_1 qui passent la batterie Smallcrush. Afin de vérifier la robustesse de ces 10 générateurs, nous leur avons fait passer la batterie de tests Crush. Deux tests se sont avérés difficiles à passer pour ces générateurs : les tests numéro 1 et 2. Soit p_1 et p_2, les deux p-valeurs obtenues pour un générateur à chacun de ces tests. Ces p-valeurs sont données au tableau 8.14. Quand il n'y a rien d'inscrit pour une p-valeur et un générateur donnés, ceci indique que la p-valeur n'est pas suspecte. Le symbole ϵ indique que la p-valeur obtenue est inférieure à 10^{-300}. Les quatres derniers générateurs donnés à ce tableau sont tels que $V = 3$, la meilleure valeur observée. Ces quatres générateurs performent bien par rapport aux tests numéro 1 et 2, mais chacun a une faiblesse à un ou plusieurs tests de la batterie Crush. Par exemple, au test numéro 3, le triplet $(23, 17, 25)$ montre une p-valeur de 2×10^{-165}, ce qui est évidemment inacceptable.

Tableau 8.5 – Valeur de V pour tous les générateurs de type I basés sur $X_1, \ldots, X_{12}$ avec $w = 32$.

(a, b, c)	X_1	X_2	X_3	X_5	moyenne
(5, 21, 12)	4	4	6	2	4.0
(11, 17, 13)	5	3	8	2	4.5
(2, 9, 15)	1	4	13	2	5.0
(5, 17, 13)	5	2	9	5	5.2
(5, 15, 17)	5	4	10	3	5.5
(6, 17, 9)	5	3	12	4	6.0
(11, 7, 16)	1	1	17	6	6.2
(2, 5, 15)	2	2	20	3	6.8
(5, 7, 22)	4	4	17	3	7.0
(6, 21, 13)	5	4	10	9	7.0
(5, 9, 7)	7	6	11	6	7.5
(5, 13, 6)	8	8	8	6	7.5
(3, 5, 20)	1	6	22	3	8.0
(8, 9, 23)	9	5	13	5	8.0
(9, 5, 25)	3	5	23	2	8.2
(9, 5, 14)	3	4	24	3	8.5
(11, 21, 13)	4	5	19	6	8.5
(1, 11, 16)	8	8	9	10	8.8
(2, 21, 9)	8	7	10	10	8.8
(3, 13, 7)	9	8	7	11	8.8
(4, 5, 15)	4	7	24	1	9.0
(10, 9, 25)	8	3	15	10	9.0
(2, 7, 9)	8	9	13	7	9.2
(2, 5, 21)	6	6	21	6	9.8
(3, 5, 22)	5	7	22	5	9.8
(6, 21, 7)	9	9	11	10	9.8
(11, 7, 12)	7	7	18	8	10.0
(12, 9, 23)	5	6	20	9	10.0
(7, 13, 25)	7	7	18	9	10.2
(9, 11, 19)	7	9	16	9	10.2
(7, 25, 12)	4	8	16	14	10.5
(9, 21, 16)	8	12	15	7	10.5
(5, 27, 8)	7	7	23	6	10.8
(7, 17, 21)	6	12	12	14	11.0
(3, 27, 11)	6	7	24	8	11.2
(13, 17, 15)	11	8	17	9	11.2
(2, 7, 7)	11	9	16	11	11.8
(4, 3, 17)	3	6	32	6	11.8
(10, 9, 21)	9	9	19	10	11.8
(13, 5, 19)	6	6	30	5	11.8
(5, 3, 21)	4	9	35	1	12.2

Tableau 8.6 – Valeur de V pour tous les générateurs de type I basés sur $X_1, \ldots, X_{12}$ avec $w = 32$ (suite).

(a, b, c)	X_1	X_2	X_3	X_5	moyenne
(2, 7, 25)	10	14	16	10	12.5
(5, 9, 28)	10	13	13	14	12.5
(13, 3, 17)	3	5	37	5	12.5
(1, 5, 19)	8	9	22	12	12.8
(17, 15, 23)	5	7	29	11	13.0
(1, 11, 6)	17	15	8	13	13.2
(3, 5, 25)	10	10	22	11	13.2
(5, 27, 21)	8	8	29	9	13.5
(1, 5, 16)	11	11	22	11	13.8
(13, 3, 27)	5	5	37	9	14.0
(1, 3, 10)	10	9	32	8	14.8
(2, 15, 17)	14	16	17	16	15.8
(2, 15, 25)	16	17	16	14	15.8
(6, 3, 17)	8	10	37	8	15.8
(7, 1, 9)	3	4	56	1	16.0
(17, 15, 20)	11	10	29	14	16.0
(17, 15, 26)	11	11	29	13	16.0
(3, 1, 14)	4	2	56	4	16.5
(3, 7, 29)	17	18	16	17	17.0
(3, 25, 24)	16	15	22	15	17.0
(5, 9, 31)	19	20	13	18	17.5
(6, 1, 11)	7	4	56	5	18.0
(5, 27, 25)	11	13	29	20	18.2
(7, 25, 20)	15	18	21	19	18.2
(4, 3, 27)	12	14	37	13	19.0
(14, 13, 15)	16	16	27	18	19.2
(5, 27, 28)	16	15	29	18	19.5
(7, 1, 25)	6	8	56	8	19.5
(3, 23, 25)	15	21	29	16	20.2
(7, 1, 18)	9	11	56	9	21.2
(8, 7, 23)	13	11	23	41	22.0
(1, 19, 3)	25	28	17	26	24.0
(3, 3, 26)	21	19	37	19	24.0
(3, 3, 28)	16	31	37	22	26.5
(1, 9, 29)	29	30	20	29	27.0
(14, 1, 15)	20	19	56	22	29.2
(3, 3, 29)	23	37	37	24	30.2
(1, 21, 20)	33	33	32	33	32.8
(15, 1, 29)	24	24	56	28	33.0
(1, 27, 27)	48	48	52	47	48.8

Tableau 8.7 – Valeur de V pour tous les générateurs de type I basés sur $X_1, \ldots, X_{12}$ avec $w = 64$.

(a, b, c)	X_1	X_2	X_3	X_5	moyenne
(9, 21, 40)	7	4	19	6	9.0
(6, 23, 27)	10	6	19	3	9.5
(5, 23, 36)	6	7	18	8	9.8
(11, 27, 26)	5	9	18	7	9.8
(8, 37, 21)	7	8	16	9	10.0
(11, 23, 42)	7	6	20	7	10.0
(11, 31, 18)	9	6	16	10	10.2
(5, 15, 27)	7	2	29	4	10.5
(8, 29, 19)	11	12	11	8	10.5
(12, 43, 19)	12	7	16	7	10.5
(9, 21, 20)	9	8	16	10	10.8
(16, 25, 43)	5	8	21	9	10.8
(15, 17, 27)	6	3	30	6	11.2
(16, 21, 35)	6	4	26	9	11.2
(19, 43, 27)	7	3	25	10	11.2
(15, 21, 46)	4	5	29	9	11.8
(13, 19, 28)	6	7	29	6	12.0
(13, 29, 35)	5	8	25	10	12.0
(8, 13, 25)	4	5	36	4	12.2
(12, 25, 27)	4	6	32	7	12.2
(14, 23, 33)	8	7	26	8	12.2
(11, 29, 14)	12	12	15	11	12.5
(13, 35, 30)	13	13	15	9	12.5
(13, 21, 18)	8	9	28	6	12.8
(4, 35, 21)	14	14	12	12	13.0
(14, 15, 19)	8	8	30	6	13.0
(6, 43, 21)	12	10	20	12	13.5
(8, 15, 21)	8	6	30	10	13.5
(8, 51, 21)	7	6	33	8	13.5
(17, 27, 22)	6	5	32	11	13.5
(7, 19, 17)	12	12	19	12	13.8
(14, 13, 17)	7	8	36	6	14.2
(3, 21, 31)	11	13	20	14	14.5
(21, 17, 24)	7	6	36	10	14.8
(21, 17, 48)	11	5	35	8	14.8
(13, 19, 47)	6	7	37	10	15.0
(17, 23, 29)	10	8	33	9	15.0
(21, 15, 29)	8	4	44	4	15.0
(23, 13, 38)	3	4	49	4	15.0
(4, 37, 21)	18	14	17	12	15.2
(5, 47, 23)	8	10	33	10	15.2

Tableau 8.8 – Valeur de V pour tous les générateurs de type I basés sur $X_1, \ldots, X_{12}$ avec $w = 64$ (suite).

(a, b, c)	X_1	X_2	X_3	X_5	moyenne
(11, 13, 40)	8	6	40	7	15.2
(13, 47, 23)	9	7	32	13	15.2
(16, 11, 19)	4	7	44	6	15.2
(18, 25, 21)	10	9	28	14	15.2
(13, 21, 49)	8	11	33	10	15.5
(19, 15, 46)	4	4	45	9	15.5
(21, 9, 29)	4	3	51	4	15.5
(23, 17, 25)	3	4	47	10	16.0
(24, 31, 35)	7	5	40	12	16.0
(4, 41, 19)	14	17	20	14	16.2
(5, 33, 29)	12	11	29	13	16.2
(9, 29, 12)	16	19	15	15	16.2
(19, 25, 20)	13	12	22	18	16.2
(17, 45, 22)	11	6	26	23	16.5
(11, 25, 48)	8	20	24	15	16.8
(6, 49, 29)	11	7	34	16	17.0
(7, 9, 38)	4	5	53	6	17.0
(11, 9, 34)	6	5	52	5	17.0
(21, 17, 30)	7	5	47	9	17.0
(24, 11, 29)	9	5	50	4	17.0
(3, 25, 20)	17	17	18	17	17.2
(8, 31, 17)	19	21	9	20	17.2
(15, 45, 17)	15	16	29	9	17.2
(29, 27, 37)	9	12	36	13	17.5
(13, 51, 21)	6	6	46	13	17.8
(2, 13, 23)	14	11	34	13	18.0
(5, 45, 16)	17	16	20	19	18.0
(8, 9, 25)	5	4	52	11	18.0
(9, 21, 11)	18	18	19	17	18.0
(10, 7, 33)	3	4	63	2	18.0
(23, 41, 34)	9	9	30	24	18.0
(13, 9, 15)	9	8	49	7	18.2
(17, 47, 28)	9	9	44	11	18.2
(7, 51, 24)	10	12	38	14	18.5
(7, 11, 35)	4	8	51	12	18.8
(9, 29, 37)	13	17	21	24	18.8
(21, 13, 52)	11	13	44	8	19.0
(5, 41, 20)	19	17	25	16	19.2
(6, 17, 47)	18	16	26	17	19.2
(3, 37, 17)	20	19	21	18	19.5

Tableau 8.9 – Valeur de V pour tous les générateurs de type I basés sur $X_1, \ldots, X_{12}$ avec $w = 64$ (suite).

(a, b, c)	X_1	X_2	X_3	X_5	moyenne
(9, 27, 10)	21	20	17	20	19.5
(4, 35, 15)	22	27	11	20	20.0
(4, 43, 21)	19	16	28	18	20.2
(16, 11, 27)	13	8	49	11	20.2
(11, 9, 14)	9	13	51	9	20.5
(15, 49, 26)	8	6	42	27	20.8
(24, 9, 35)	8	6	59	10	20.8
(5, 9, 23)	11	12	50	11	21.0
(15, 13, 52)	12	12	44	16	21.0
(19, 13, 37)	6	7	60	11	21.0
(25, 11, 57)	8	10	54	12	21.0
(10, 53, 13)	14	12	43	16	21.2
(21, 15, 28)	8	5	48	24	21.2
(9, 19, 18)	13	17	33	23	21.5
(13, 7, 17)	5	7	68	6	21.5
(13, 35, 38)	15	18	33	20	21.5
(18, 41, 23)	13	17	38	19	21.8
(23, 17, 54)	7	5	61	14	21.8
(13, 9, 50)	10	13	52	13	22.0
(15, 13, 28)	14	9	45	20	22.0
(25, 15, 47)	6	5	59	18	22.0
(3, 29, 40)	19	19	31	20	22.2
(14, 47, 15)	12	17	44	17	22.5
(17, 47, 29)	12	11	46	22	22.8
(11, 15, 37)	10	23	38	22	23.2
(7, 43, 28)	14	23	34	23	23.5
(23, 13, 58)	15	12	49	18	23.5
(3, 35, 14)	28	24	15	28	23.8
(7, 41, 40)	15	16	44	21	24.0
(19, 7, 36)	8	10	71	7	24.0
(9, 5, 43)	9	4	78	6	24.2
(9, 7, 18)	9	13	66	9	24.2
(16, 13, 55)	19	22	40	16	24.2
(25, 9, 39)	9	8	65	15	24.2
(7, 19, 54)	22	31	20	25	24.5
(23, 17, 56)	9	12	61	16	24.5
(4, 7, 19)	13	13	59	14	24.8
(6, 55, 17)	15	14	52	18	24.8
(16, 5, 17)	6	4	80	9	24.8
(17, 23, 52)	15	22	43	19	24.8

Tableau 8.10 – Valeur de V pour tous les générateurs de type I basés sur $X_1, \ldots, X_{12}$ avec $w = 64$ (suite).

(a, b, c)	X_1	X_2	X_3	X_5	moyenne
(5, 17, 11)	26	25	23	26	25.0
(19, 7, 55)	8	11	71	11	25.2
(4, 37, 11)	28	26	20	28	25.5
(12, 7, 13)	11	15	61	15	25.5
(25, 33, 36)	15	20	46	21	25.5
(7, 33, 8)	27	26	20	31	26.0
(25, 13, 29)	6	10	81	8	26.2
(1, 19, 16)	27	32	19	28	26.5
(7, 5, 41)	8	10	79	9	26.5
(7, 23, 8)	30	31	17	28	26.5
(2, 43, 27)	24	24	36	23	26.8
(7, 11, 10)	23	23	41	21	27.0
(11, 5, 34)	9	6	88	5	27.0
(6, 27, 7)	33	33	13	31	27.5
(20, 5, 29)	8	7	87	8	27.5
(25, 7, 49)	11	14	68	18	27.8
(1, 23, 14)	31	30	22	29	28.0
(3, 5, 21)	11	13	77	11	28.0
(4, 15, 51)	27	26	30	29	28.0
(11, 5, 45)	8	9	88	7	28.0
(9, 5, 36)	11	11	78	14	28.5
(3, 25, 31)	22	34	34	25	28.8
(3, 43, 11)	31	29	23	32	28.8
(12, 11, 47)	19	19	54	23	28.8
(1, 13, 45)	27	30	36	24	29.2
(9, 41, 45)	14	21	53	29	29.2
(19, 21, 52)	9	11	57	41	29.5
(23, 9, 38)	4	4	105	5	29.5
(4, 9, 13)	25	22	50	22	29.8
(4, 15, 53)	29	27	35	28	29.8
(4, 29, 45)	24	33	30	32	29.8
(11, 5, 32)	3	3	106	7	29.8
(1, 23, 29)	25	26	42	27	30.0
(11, 53, 23)	19	20	55	26	30.0
(12, 39, 49)	17	23	54	27	30.2
(28, 11, 53)	15	12	62	32	30.2
(5, 11, 54)	27	29	40	27	30.8
(11, 5, 43)	7	5	106	5	30.8
(23, 9, 48)	13	11	89	12	31.2
(13, 13, 19)	24	32	59	12	31.8

Tableau 8.11 – Valeur de V pour tous les générateurs de type I basés sur $X_1, \ldots, X_{12}$ avec $w = 64$ (suite).

(a, b, c)	X_1	X_2	X_3	X_5	moyenne
(3, 51, 16)	26	33	37	32	32.0
(7, 7, 20)	16	17	81	14	32.0
(7, 57, 12)	21	18	70	20	32.2
(17, 15, 58)	25	22	61	21	32.2
(23, 9, 57)	5	12	105	7	32.2
(25, 7, 46)	20	21	68	21	32.5
(16, 7, 39)	14	11	75	31	32.8
(25, 21, 44)	26	24	57	24	32.8
(7, 5, 47)	18	18	78	19	33.2
(23, 13, 61)	23	42	49	20	33.5
(4, 43, 31)	27	31	47	32	34.2
(15, 5, 37)	11	17	93	16	34.2
(16, 27, 53)	15	52	39	31	34.2
(1, 11, 50)	31	36	42	31	35.0
(10, 27, 59)	31	35	35	39	35.0
(4, 29, 49)	32	45	32	37	36.5
(3, 29, 49)	39	35	36	37	36.8
(7, 25, 58)	39	43	28	38	37.0
(9, 23, 57)	30	56	33	29	37.0
(12, 3, 13)	9	12	113	15	37.2
(23, 17, 62)	27	30	61	31	37.2
(7, 13, 58)	34	39	42	35	37.5
(19, 41, 21)	34	33	48	36	37.8
(21, 41, 23)	36	34	45	38	38.2
(31, 27, 35)	31	38	48	36	38.2
(4, 31, 33)	37	36	47	34	38.5
(13, 3, 40)	16	16	107	18	39.2
(8, 13, 61)	40	39	37	42	39.5
(3, 29, 47)	36	55	32	38	40.2
(31, 25, 37)	34	33	54	41	40.5
(15, 19, 63)	41	44	35	43	40.8
(4, 53, 7)	43	40	41	40	41.0
(13, 3, 53)	18	18	107	21	41.0
(21, 21, 34)	34	40	56	34	41.0
(25, 39, 54)	31	33	61	39	41.0
(1, 11, 35)	37	40	50	38	41.2
(1, 35, 11)	40	37	47	43	41.8
(12, 3, 49)	14	23	113	17	41.8
(1, 9, 50)	40	41	50	38	42.2
(16, 47, 17)	39	38	50	43	42.5

Tableau 8.12 – Valeur de V pour tous les générateurs de type I basés sur $X_1, \ldots, X_{12}$ avec $w = 64$ (suite).

(a, b, c)	X_1	X_2	X_3	X_5	moyenne
(15, 47, 16)	37	39	59	39	43.5
(5, 53, 20)	22	48	50	55	43.8
(22, 3, 39)	28	21	107	19	43.8
(3, 53, 7)	44	43	44	45	44.0
(17, 47, 54)	29	32	76	40	44.2
(23, 41, 51)	30	36	65	46	44.2
(8, 5, 59)	34	34	77	33	44.5
(18, 3, 43)	47	7	113	12	44.8
(13, 17, 43)	24	86	38	32	45.0
(17, 23, 51)	27	31	88	37	45.8
(1, 7, 44)	37	47	64	37	46.2
(1, 29, 34)	47	49	44	45	46.2
(7, 5, 55)	32	33	80	41	46.5
(15, 23, 23)	41	36	73	36	46.5
(1, 51, 13)	45	45	53	45	47.0
(3, 61, 17)	27	27	107	27	47.0
(1, 7, 9)	44	47	59	43	48.2
(3, 61, 26)	31	25	104	33	48.2
(5, 59, 33)	34	34	87	39	48.5
(1, 7, 46)	42	47	64	43	49.0
(6, 1, 17)	16	13	153	14	49.0
(12, 1, 31)	15	11	153	19	49.5
(6, 3, 49)	34	24	113	29	50.0
(33, 31, 43)	32	30	94	44	50.0
(3, 43, 6)	52	50	48	53	50.8
(1, 19, 6)	61	64	23	61	52.2
(9, 1, 27)	20	19	153	18	52.5
(1, 59, 14)	45	42	81	43	52.8
(3, 25, 56)	54	63	44	57	54.5
(7, 27, 59)	58	62	47	59	56.5
(14, 31, 45)	51	60	60	56	56.8
(3, 13, 59)	60	58	49	63	57.5
(15, 1, 19)	27	24	153	29	58.2
(3, 43, 4)	64	62	49	59	58.5
(8, 25, 59)	61	64	47	63	58.8
(3, 1, 11)	28	31	153	31	60.8
(11, 23, 56)	61	68	53	64	61.5
(25, 13, 39)	58	56	81	52	61.8
(21, 21, 37)	56	67	76	53	63.0
(25, 13, 62)	55	56	81	60	63.0

Tableau 8.13 – Valeur de V pour tous les générateurs de type I basés sur $X_1, \ldots, X_{12}$ avec $w = 64$ (suite).

(a, b, c)	X_1	X_2	X_3	X_5	moyenne
(2, 31, 51)	67	61	67	59	63.5
(28, 9, 55)	47	50	105	52	63.5
(1, 35, 5)	71	68	49	70	64.5
(21, 21, 38)	60	68	76	57	65.2
(33, 31, 55)	53	53	94	66	66.5
(2, 31, 53)	73	61	67	66	66.8
(18, 1, 25)	40	33	153	41	66.8
(49, 15, 61)	60	59	76	73	67.0
(43, 21, 46)	50	50	113	60	68.2
(18, 19, 19)	67	68	74	71	70.0
(5, 59, 35)	59	59	106	61	71.2
(1, 15, 4)	82	81	30	97	72.5
(20, 1, 31)	44	44	153	53	73.5
(21, 21, 32)	71	71	85	71	74.5
(4, 41, 45)	66	75	90	68	74.8
(1, 3, 45)	63	72	103	62	75.0
(25, 27, 53)	79	79	67	84	77.2
(21, 21, 40)	75	79	85	76	78.8
(21, 1, 27)	49	57	153	59	79.5
(55, 9, 56)	70	69	105	83	81.8
(25, 27, 27)	82	84	76	88	82.5
(1, 53, 3)	90	89	85	89	88.2
(2, 47, 49)	88	93	108	88	94.2
(1, 35, 34)	94	93	97	94	94.5
(1, 45, 37)	100	100	91	100	97.8
(5, 59, 63)	92	92	106	104	98.5
(31, 1, 51)	78	80	153	93	101.0
(1, 15, 63)	122	123	53	122	105.0
(21, 21, 41)	105	109	112	109	108.8
(24, 25, 25)	112	113	95	115	108.8
(21, 21, 43)	110	111	113	109	110.8
(1, 1, 54)	106	116	153	107	120.5
(1, 1, 55)	110	116	153	109	122.0

Tableau 8.14 – Les p-valeurs p_1 et p_2 obtenues par les dix premiers générateurs basés sur X_1 de la table 8.7.

(a, b, c)	p_1	p_2
(9, 21, 40)		8.0×10^{-4}
(6, 23, 27)		4.1×10^{-7}
(5, 23, 36)		ϵ
(11, 27, 26)		
(8, 37, 21)	1.9×10^{-16}	
(11, 23, 42)		
(11, 31, 18)	2.5×10^{-46}	
(5, 15, 27)		
(8, 29, 19)	ϵ	3.9×10^{-6}
(12, 43, 19)		
(23, 17, 25)		
(23, 13, 38)		
(10, 7, 33)		
(11, 5, 32)		

8.4.2 Générateurs de type II

Pour les générateurs de type II, nous avons calculé V pour tous les générateurs qui ont été proposés par Marsaglia [60]. Les résultats de ces calculs sont donnés dans la première colonne du tableau 8.15. Dans ce même tableau, on donne la valeur de V pour les générateurs basés sur $(A_1, B_1, m = 1)$, $(A_2, B_2, m = 1)$ et $(A_4, B_4, m = 1)$ que procurent les triplets proposés par Marsaglia. Ils se trouvent dans la première colonne de ce tableau. Les autres colonnes donnent la valeur de V pour les autres générateurs, de même période, qui sont basés sur $(A_2, B_2, m = 1)$ et $(A_4, B_4, m = 1)$. À noter que les générateurs basés sur $(A_2, B_2, m = 1)$ et $(A_4, B_4, m = 1)$ n'ont pas été identifiés par Marsaglia et que nous mettons ces résultats pour satisfaire la curiosité du lecteur.

Nous avons testé avec la batterie Smallcrush tous les générateurs basés sur $(A_1, B_1, m = 1)$ proposés par Marsaglia, soit ceux de la première colonne du tableau 8.15. Tous ces générateurs, sauf un, échouent soit le test numéro 4, soit le test numéro 5. Au tableau 8.15, nous donnons les p-valeurs obtenues pour ces deux tests, soit p_4 et p_5. Le symbole ϵ indique que la p-valeur obtenue est inférieure à 10^{-300}. Pour le générateur basé sur le triplet $(23, 24, 3)$, les p-valeurs ne sont pas suspectes pour aucun de tests de Smallcrush. Pour le tester davantage, nous lui avons fait passer la batterie de tests Crush. Pour le test numéro 7, le générateur échoue avec une p-valeur inférieure à 10^{-300}.

Au tableau 8.16, nous donnons les meilleurs générateurs de type II par rapport au critère V pour $r = 2, 3, 4, 5, 8, 12, 25$. Rappelons que la recherche a été faite de manière exhaustive. On remarque une nette amélioration du critère V par rapport à ceux qui sont proposés par Marsaglia. Par exemple, quand $r = 5$, la meilleure valeur de V des générateurs proposés par Marsaglia était 111 et le meilleur possible est $V = 18$ donné par $((I + D^7)(I + G^{11}), (I + G^{20}), m = 1)$. En observant le tableau 8.16, on remarque qu'un seul générateur est tel qu'il utilise deux matrices de la forme $(I + D^d)$.

Ceci pourrait s'expliquer, comme il a été écrit précédemment, par le fait que cette transformation ne mélange pas très bien les bits les plus significatifs (sauf, peut-être, pour le cas qu'on vient d'identifier).

En ne considérant que le critère V, on peut obtenir de très bons générateurs de type II. Le TT800 [67], un TGFSR considéré comme un très bon générateur de période $2^{800} - 1$, donne une valeur de $V = 261$. On peut voir, dans le tableau 8.16, que les meilleurs qui ont la même période, soit ceux avec $r = 25$, ont tous une valeur de V plus petite. Par exemple, le générateur basé sur $((I + D^8)(I + G^{11}), (I + G^{18}), m = 9)$ obtient $V = 123$, ce qui est nettement inférieur à la valeur obtenue par le TT800. Également, on remarque qu'aucun tempering n'est appliqué à la sortie du générateur, au contraire de TT800. On pourrait s'attendre à une implantation plus rapide pour ce générateur que celle du TT800 pour cette raison (voir à la fin du chapitre pour les temps d'éxécution des générateurs).

On a appliqué les batteries de tests Smallcrush et Crush à tous les générateurs qui sont dans le tableau 8.16. La plupart des générateurs ont passé tous les tests de Smallcrush, sans aucune valeur suspecte. Nous leur avons fait passer la batterie de tests Crush.

La grande faiblesse des générateurs du tableau 8.16 est qu'il ne réussissent pas à passer les tests du rang de la matrice. Nous avons fait passer le test numéro 8 (test du rang de la matrice dont il est question dans la remarque de la section 2.7) à tous les générateurs dont $r \geq 8$ et seuls les générateurs 21, 22, 26, 27, 28 et 29 ont réussi le test. Ainsi, aussi pour les générateurs dont $r = 12$ ou $r = 25$, plusieurs d'entre eux ne réussissent pas à passer le test numéro 9, un autre test sur la distribution d'une matrice aléatoire, sauf les générateurs numéro 22, 28 et 29 du tableau 8.16. Pour les générateurs avec $r < 12$, on ne rapporte pas le résultat du test puisqu'il est normal qu'il ait échoué. C'est pourquoi, on a mis un « – » pour ceux-ci.

Pour les générateurs dont $r = 2, 3, 4$ ou 5, ils échouent tous soit le test numéro 6, soit le test numéro 7. Soient p_6 et p_7, les p-valeurs obtenues pour ces deux tests. Au tableau 8.16, on présente les résultats obtenus pour ces deux tests.

Pour résumer les résultats obtenus par les générateur du tableau 8.16 aux tests statistiques contenus dans les batteries Smallcrush et Crush, on pourrait dire que seuls quatre générateurs ont réussi à les passer tous. Ces sont les générateurs numéro 21, 22, 28 et 29. Nous avons fait passser également la batterie Bigcrush à ces quatres générateurs et ils ont réussi à passer tous les tests contenus dans cette batterie.

Tableau 8.15 – Valeur de V pour tous les générateurs de type II proposés par Marsaglia [60] basés sur (A_i, B_i) pour $i = 1, 2, 4$ avec $w = 32$ et $m = 1$.

(a, b, c)	(A_1, B_1)	(A_2, B_2)	(A_4, B_4)	p_4	p_5
$r = 2$					
(10, 13, 10)	27	58	28	ϵ	5.2×10^{-5}
(8, 9, 22)	11	35	7	ϵ	
(2, 7, 3)	38	92	39	ϵ	
(23, 3, 24)	37	62	37		ϵ
$r = 3$					
(10, 5, 26)	67	97	68	ϵ	
(13, 19, 3)	36	63	27	ϵ	
(1, 17, 2)	47	190	47	ϵ	
(10, 1, 26)	81	61	82	ϵ	
$r = 4$					
(5, 14, 1)	68	117	67	ϵ	
(15, 4, 21)	41	78	30	2×10^{-3}	
(23, 24, 3)	85	135	85		
(5, 12, 29)	48	167	41	ϵ	
(11, 8, 19)	32	69	26	ϵ	0.9963
$r = 5$					
(2, 1, 4)	164	247	165	ϵ	
(7, 13, 6)	111	152	100	ϵ	
(1, 1, 20)	164	322	164	ϵ	

Tableau 8.16 – Valeur de V pour les meilleurs générateurs de type II avec $w = 32$ et $r = 2, 3, 4, 5, 8, 12, 25$.

No	r	m	B	A	V	p_7	p_6	p_9	p_8
1	2	1	$(I+G^{11})$	$(I+D^{13})(I+G^{19})$	4	3.3×10^{-12}	2.4×10^{-6}	–	–
2	2	1	$(I+G^{11})$	$(I+G^{19})(I+D^{13})$	7	6.6×10^{-12}	4.3×10^{-9}	–	–
3	2	1	$(I+D^{8})(I+G^{9})$	$(I+G^{22})$	7		ϵ	–	–
4	2	1	$(I+G^{11})(I+D^{9})$	$(I+G^{17})$	7	5.4×10^{-7}	ϵ	–	–
5	3	1	$(I+G^{23})$	$(I+D^{4})(I+G^{13})$	11		ϵ	–	–
6	3	1	$(I+G^{23})$	$(I+G^{11})(I+D^{7})$	12	1.0×10^{-3}	ϵ	–	–
7	3	1	$(I+G^{23})$	$(I+D^{7})(I+G^{11})$	12		ϵ	–	–
8	3	1	$(I+G^{18})$	$(I+D^{5})(I+G^{13})$	13		ϵ	–	–
9	4	1	$(I+D^{7})(I+G^{11})$	$(I+G^{20})$	13		ϵ	–	–
10	4	1	$(I+D^{7})(I+G^{11})$	$(I+G^{19})$	17		ϵ	–	–
11	4	1	$(I+G^{17})$	$(I+D^{5})(I+G^{12})$	17		ϵ	–	–
12	4	1	$(I+G^{19})$	$(I+D^{15})(I+G^{7})$	19	6.5×10^{-9}		–	–
13	4	1	$(I+G^{11})(I+D^{7})$	$(I+G^{19})$	19	6.5×10^{-12}	ϵ	–	–
14	5	1	$(I+D^{7})(I+G^{11})$	$(I+G^{20})$	18		ϵ	–	–
15	5	1	$(I+D^{6})(I+G^{11})$	$(I+G^{20})$	19		ϵ	–	–
16	5	3	$(I+G^{9})(I+D^{6})$	$(I+G^{20})$	25		ϵ	–	–
17	5	3	$(I+D^{6})(I+G^{9})$	$(I+G^{20})$	25		ϵ	–	–
18	8	3	$(I+D^{13})(I+G^{19})$	$(I+G^{8})$	45			–	ϵ
19	8	3	$(I+D^{14})(I+G^{17})$	$(I+G^{8})$	48			–	ϵ
20	8	1	$(I+D^{7})(I+G^{15})$	$(I+G^{10})$	52			–	ϵ
21	8	1	$(I+D^{11})(I+G^{8})$	$(I+G^{21})$	54			–	
22	12	5	$(I+G^{21})(I+D^{11})$	$(I+G^{6})$	74				
23	12	5	$(I+D^{6})(I+G^{7})$	$(I+G^{22})$	79			ϵ	ϵ
24	12	1	$(I+D^{8})(I+G^{7})$	$(I+G^{18})$	84			ϵ	ϵ
25	12	5	$(I+G^{7})(I+D^{6})$	$(I+G^{22})$	90			ϵ	ϵ
26	25	9	$(I+D^{8})(I+G^{11})$	$(I+G^{18})$	123			ϵ	
27	25	9	$(I+G^{11})(I+D^{8})$	$(I+G^{18})$	137			ϵ	
28	25	7	$(I+G^{20})$	$(I+D^{13})(I+G^{5})$	155				
29	25	2	$(I+G^{10})$	$(I+D^{13})(I+G^{19})$	158				

8.4.3 Générateurs de type III

Pour les générateurs de type III, nous avons effectué les mêmes recherches que pour les générateurs de type II. Nous avons cherché, de manière exhaustive, tous les générateurs qui utilisent la récurrence (8.7) pour $r = 3, 4, 5, 8, 12, 25$. Les paramètres des meilleurs générateurs trouvés par rapport au critère V sont donnés au tableau 8.17. À période égale, on remarque que les valeurs de V sont un peu plus élevées pour les meilleurs générateurs de type III que pour les meilleurs générateurs de type II. On remarque également que tous les générateurs du tableau 8.17 utilisent deux matrices $(I + G^g)$ et une seule matrice $(I + D^d)$, comme pour la très grande majorité des meilleurs générateurs de type I et type II.

Nous avons fait passer les tests contenus dans les batteries Smallcrush, Crush et Bigcrush à tous les générateurs contenus dans le tableau 8.17. Les résultats obtenus par les générateurs aux tests contenus dans Smallcrush ne permettent pas de rejeter l'hypothèse nulle pour aucun des générateurs. La batterie de tests Crush a detecté une faiblesse qui semble se généraliser pour les générateurs avec $r \leq 5$. Au test numéro 6, la plupart des générateurs ont obtenu des p-valeurs lamentables. Ce test, qui mesure les dépendances des poids de Hamming entre les valeurs produites par le générateur, tend à démontrer que la récurrence ne mélange pas assez les bits d'une itération à l'autre. En fait, parmi ces douzes générateurs, seuls les générateurs numéro 7, 8, 11 et 12 ont passé ce test. Également, plusieurs générateurs avec $r \leq 8$ ont échoué le test numéro 10. Seuls les générateurs numéro 7, 8, 11 et 14 ont réussi ce test. Pour les générateurs dont $r \geq 12$, nous n'avons pas trouvé de tests qui mettent en doute l'hypothèse nulle.

Tableau 8.17 – Valeur de V pour les meilleurs générateurs de type III avec $w = 32$ et $r = 3, 4, 5, 8, 12, 25$.

No	r	m_2	m_1	H_2^b	H_1^a	H_3^c	V	p_6	p_{10}
1	3	1	2	G^7	G^{21}	D^9	29	ϵ	ϵ
2	3	1	2	D^3	G^5	G^{12}	36	ϵ	ϵ
3	3	1	2	G^3	D^1	G^{17}	37	ϵ	ϵ
4	3	1	2	G^7	D^9	G^{16}	37	ϵ	ϵ
5	4	1	3	G^9	D^7	G^{21}	22	ϵ	ϵ
6	4	1	2	G^9	D^7	G^{22}	25	ϵ	ϵ
7	4	1	3	G^7	G^{21}	D^{11}	28		
8	4	2	3	G^5	D^{11}	G^{18}	29		
9	5	1	2	G^5	D^3	G^{19}	30	ϵ	ϵ
10	5	2	3	D^7	G^{21}	G^9	33	ϵ	ϵ
11	5	1	4	G^5	D^{11}	G^{20}	44		
12	5	1	2	G^{21}	G^3	D^5	48		ϵ
13	8	1	2	D^7	G^{22}	G^9	55		ϵ
14	8	2	5	G^7	D^{11}	G^{21}	55		
15	8	1	3	G^9	G^{21}	D^7	65		ϵ
16	8	1	5	G^3	G^{19}	D^5	68		ϵ
17	12	2	3	G^7	D^{11}	G^{21}	96		
18	12	5	11	G^5	G^{18}	D^{11}	100		
19	12	7	9	G^{18}	D^{11}	G^5	102		
20	12	5	10	G^5	G^{18}	D^{11}	103		
21	25	4	10	G^{21}	D^{11}	G^7	186		
22	25	5	24	G^5	D^{11}	G^{18}	188		
23	25	7	24	G^5	D^{11}	G^{18}	190		
24	25	5	16	G^{19}	D^{11}	G^5	219		

8.5 Performances

Nous avons implanté, en language C, des générateurs de type I, II et III et les avons chronométrés. La période du générateur de type I est de $2^{32} - 1$ et celle des générateurs de types II et III est de $2^{800} - 1$. Les implantations ne sont pas des macros, comme il est expliqué dans l'article original de Marsaglia [60], mais comme des fonctions qui retournent une valeur de type `double`. Les macros ne sont utiles que pour des générateurs de petite période. Le test de vitesse est le même que celui expliqué à la section 5.13. Le tableau 8.18 donne la vitesse de chacun des générateurs. Nous avons reproduit tous les temps du tableau 7.13 pour fins de comparaisons. En examinant le tableau 8.18, on remarque que les générateurs de Marsaglia sont rapides comparativement aux autres générateurs. Mais le gain en vitesse obtenu en se restreignant à seulement trois décalages de bits et trois ou-exclusifs bit-à-bit n'est pas vraiment spectaculaire.

8.6 Conclusion

Le point à retenir de l'étude des générateurs qui n'utilisent que trois décalages de bits, tels que décrits dans ce chapitre, est que ceux-ci ne mélangent pas suffisamment l'état d'une itération à l'autre pour obtenir des générateurs dans lesquels nous pouvons avoir confiance.

Les générateurs de type I n'ont pas des périodes assez longues pour des applications sérieuses. De nombreuses faiblesses ont été détectées pour les meilleurs générateurs (par rapport à l'équidistribution) de type II, même pour des périodes immenses de l'ordre de $2^{800} - 1$. Pour ceux qui ont passé tous les tests statistiques, il n'est pas recommandé de les utiliser puisqu'ils possèdent la même structure et sont susceptibles d'échouer d'autres tests statistiques. Pour les générateurs de type III, ceux-ci

Tableau 8.18 – Temps nécessaire pour itérer 10^9 fois et additionner tous les nombres aléatoires produits.

Générateur	Temps (s)
Générateur xorshift de Type I	32.4
Générateur xorshift de Type II	36.5
Générateur xorshift de Type III	34.7
F2wLFSR2_32_800	39.5
F2wPolyLCG2_32_800	36.4
F2wPolyLCG2_32_800(*)	31.3
F2wLFSR3_7_800	32.8
F2wPolyLCG3_7_800	40.7
F2wPolyLCG3_7_800(*)	33.0
WELLRNG512a	36.0
WELLRNG607a	38.1
WELLRNG607a(b)	36.9
WELLRNG800a	40.7
WELLRNG1024a	36.0
WELLRNG19937a	36.9
WELLRNG19937a(b)	42.2
WELLRNG44497a	41.3
WELLRNG44497a(b)	50.7
TT800	44.0
MT19937	31.6
MRG32k3a	101

semblaient plus robustes quand la période dépassait $2^{256} - 1$, mais peut être n'avons nous pas été assez imaginatif pour trouver un test simple pour lequel ces générateurs échouent. Les nombreuses faiblesses des générateurs de type III de plus petite période n'ont rien de rassurant.

En définitive, nous ne recommandons aucun générateur dont nous faisons mention dans ce chapitre, même ceux qui démontrent la meilleure équidistribution et qui passent les tests statistiques contenus dans les batteries de tests de Testu01.

Annexe A

Définitions relatives aux corps finis

Beaucoup de générateurs sont basés sur une récurrence linéaire dans un corps fini. Le livre de Lidl et Niederreiter [56] couvre la majorité des aspects mathématiques des corps finis discutés dans cette thèse. Les trois prochaines définitions mènent à la définition de corps fini.

Définition A.1. [71] Groupe.
Un groupe est un ensemble G sur lequel est défini une opération « + » qui est associative. De plus, il y a un élément « 0 » , appelé « élément neutre » , dans G tel que $0 + g = g + 0$ pour tout $g \in G$ et chaque élément $g \in G$ a un inverse $h \in G$ tel que $h + g = g + h = 0$.

Définition A.2. [71] Groupe Abélien.
Un groupe G est dit *Abélien* si l'opération « + » est commutative.

Définition A.3. [71] Corps.
Un Corps est un ensemble Q comprenant 2 éléments, « 0 » et « 1 » , combiné avec 2 opérations, « + » et « × » , tel que

1. Q est un groupe abélien sous l'opération « + » , avec l'élément neutre « 0 » .

2. Les éléments autres que « 0 » de Q forment un groupe abélien (où « 1 » est l'élément neutre) sous l'opération « $\times$ » .
3. La propriété $a \times (b + c) = a \times b + a \times c$ est valide.

Un corps est appelé *corps fini*, si Q est un ensemble fini.

Les tableaux A.1 et A.2 montrent les tables d'opération pour « $+$ » et « $\times$ » dans $\mathbb{F}_2$. Le corps $\mathbb{F}_2$ est celui le plus utilisé dans cette thèse. Pour $\mathbb{F}_2$, on a $Q = \{0, 1\}$

Tableau A.1 – Table de l'addition dans $\mathbb{F}_2$

+	0	1
0	0	1
1	1	0

Tableau A.2 – Table de la multiplication dans $\mathbb{F}_2$

×	0	1
0	0	0
1	0	1

On peut construire le corps fini $\mathbb{F}_{2^w}$ de plusieurs façons. Mais peu importe la manière, il existe toujours un isomorphisme entre celles-ci. Soit $\mathbb{F}_2[z]$, l'ensemble de tous les polynômes à coefficients éléments de $\mathbb{F}_2$. Si $P(z) \in \mathbb{F}_2[z]$ est un polynôme de degré w irréductible sur $\mathbb{F}_2$ et $w \geq 1$, alors on peut construire $\mathbb{F}_{2^w}$ avec l'anneau de polynômes $\mathbb{F}_2[z]/P(z)$, qui est l'espace des polynômes dans $\mathbb{F}_2[z]$ modulo $P(z)$ [56].

Cette annexe contient plusieurs définitions relatives aux corps finis qui peuvent être trouvées dans [56].

Définition A.4. Polynôme irréductible.

Un polynôme $p(z) \in \mathbb{F}_q[z]$ est dit *irréductible* sur $\mathbb{F}_q$ si $p(z)$ est de degré positif et $p(z) = b(z)c(z)$ avec $b(z), c(z) \in \mathbb{F}_q[z]$ implique que soit $b(z)$ ou $c(z)$ est le polynôme 1.

Définition A.5. Idéal.

Un sous-ensemble J d'un anneau R est appelé *idéal* si J est un sous-anneau de R et pour tout $a \in J$ et $r \in R$, on a $ar \in J$ et $ra \in J$.

Définition A.6. Élément primitif.

Un générateur du groupe cyclique $G - \{0\}$ est appelé *élément primitif de* G.

Définition A.7. Élément algébrique.

Soit K, un sous-corps de F. Si $\theta \in F$ satisfait une équation non-triviale $a_n\theta^n + \cdots + a_1\theta + a_0 = 0$ telle que les $a_i \in K$ ne sont pas tous zéros, alors on dit que θ est *algébrique* sur K.

Définition A.8. Polynôme minimal.

Soit K, un sous-corps de F. Si $\theta \in F$ est algébrique sur K, alors le polynôme monique $g(z) \in K[z]$ unique qui génère l'idéal $J = \{f \in K[z] : f(\theta) = 0\}$ de $K[z]$ est appelé *polynôme minimal* de θ sur K.

Définition A.9. Polynôme primitif dans $\mathbb{F}_q[z]$.

Un polynôme $f \in \mathbb{F}_q[z]$ de degré $m \geq 1$ est dit *primitif* sur $\mathbb{F}_q$ s'il est le polynôme minimal d'un élément primitif.

Annexe B

Arithmétique dans $\mathbb{F}_{2^w}$

Cette section passe en revue l'arithmétique dans le corps fini $\mathbb{F}_{2^w}$. Pour représenter un élément dans $\mathbb{F}_{2^w}$, choisissons une base ordonnée $(\beta_1, \ldots, \beta_w)$ de $\mathbb{F}_{2^w}$ sur $\mathbb{F}_2$ où les $\beta_i \in \mathbb{F}_{2^w}, 1 \leq i \leq w$. Grâce à cette base, n'importe quel élément $\alpha \in \mathbb{F}_{2^w}$ peut être écrit comme une combinaison linéaire

$$\alpha = \alpha_1\beta_1 + \cdots + \alpha_w\beta_w$$

où $\alpha_i \in \mathbb{F}_2, 1 \leq i \leq w$. Cette combinaison linéaire est unique puisque $(\beta_1, \ldots, \beta_w)$ est une base. À partir de cette combinaison linéaire, on peut associer le vecteur de bits $\mathbf{v} = (\alpha_1, \ldots, \alpha_w)^\mathsf{T}$ à α. A noter que ce vecteur de bits dépend de la base ordonnée choisie. Un avantage de cette représentation est la possibilité de manipuler facilement les éléments de $\mathbb{F}_{2^w}$ sur ordinateur.

Il existe deux types de bases couramment utilisées dans la littérature : la base polynômiale et la base normale. Soit $M(z) = z^w + \sum_{i=1}^{w} a_i z^{w-i} \in \mathbb{F}_2[z]$, un polynôme irréductible sur $\mathbb{F}_2$. Il existe un élément algébrique $\zeta \in \mathbb{F}_{2^w}$ dont le polynôme minimal sur $\mathbb{F}_2$ est $M(z)$. La base polynômiale définie par ζ est $(1, \zeta, \zeta^2, \ldots, \zeta^{w-1})$. Si chacune des composantes de $(\zeta, \zeta^2, \zeta^{2^2}, \ldots, \zeta^{2^{w-1}})$ sont linéairement indépendantes, alors celui-ci constitue une base normale [56]. Si le polynôme $M(z)$ est primitif (ou, de manière

équivalente, si ζ est primitif), alors n'importe quel élément de $\alpha \in \mathbb{F}_{2^w}$, sauf le zéro, peut être représenté par ζ^s pour une valeur de s.

La multiplication dans $\mathbb{F}_{2^w}$ est linéaire, ce qui veut dire qu'on peut effectuer la multiplication par ζ par une transformation linéaire $T : \mathbb{F}_{2^w} \to \mathbb{F}_{2^w}$. En représentant un élément de $\mathbb{F}_{2^w}$ par un vecteur de bits dans $\mathbb{F}_2^w$, la multiplication par ζ se fait par la multiplication par une matrice A_ζ. Pour déterminer cette matrice A_ζ, il suffit de trouver comment elle transforme les éléments de la base polynômiale. Soit $\mathbf{e}_i$, le vecteur représentant l'élément $\zeta^{i-1}, i = 1, \ldots, w$. On observe que $A_\zeta \mathbf{e}_i = \mathbf{e}_{i+1}$ pour $i = 1, \ldots, w-1$. Puisque $M(\zeta) = 0$, on sait que $\zeta^w = \sum_{i=1}^{w} a_i \zeta^{w-i}$. Donc $A_\zeta \mathbf{e}_w = \mathbf{a} = (a_w, \ldots, a_1)$.

On peut maintenant déduire que

$$A_\zeta = \begin{pmatrix} \mathbf{e}_2^\mathsf{T} & : & \ldots & : & \mathbf{e}_w^\mathsf{T} & : & \mathbf{a}^\mathsf{T} \end{pmatrix} = \begin{pmatrix} & & & & a_w \\ 1 & & & & a_{w-1} \\ & 1 & & & a_{w-2} \\ & & \ddots & & \vdots \\ & & & 1 & a_1 \end{pmatrix}. \tag{B.1}$$

La multiplication de $\eta \in \mathbb{F}_{2^w}$ par ζ^s peut être effectuée par une composition de s transformations linéaires T appliquées à η. Soit $\xi = \sum_{i=1}^{w} \alpha_i \zeta^{i-1}$. La multiplication de η par ξ se fait en multipliant sa représentation vectorielle par la matrice $A_\xi = \sum_{i=1}^{w} \alpha_i A_\zeta^{i-1}$.

Annexe C

Guide d'utilisation de F2wStreams

F2wStreams

$\mathbb{F}_{2^w}$-Random Number Generator Package with Streams

User's Guide and Functions Definitions

December 2003
Version 2.0

François Panneton
[redacted]
Département d'Informatique et Recherche Opérationnelle
CP 6128 succ Centre Ville
Montréal, Québec, H3C 3J7
Canada

Abstract

F2wStreams is a software package that implements random number generators based on linear recurrences in the finite field $\mathbb{F}_{2^w}$. Many streams of independant random numbers can be used for each generator.

Contents

1 Introduction

The random number generators included in this package are based on a linear recurrence in the finite field with 2^w elements, $\mathbb{F}_{2^w}$. The recurrence is

$$q_n(z) = zq_{n-1}(z) \bmod P(z) \tag{1}$$

where, for $n \geq 0$, $q_n(z) = q_{n,1}z^{r-1} + \cdots + q_{n,r} \in \mathbb{F}_{2^w}/P(z)$ and

$$P(z) = z^r - \sum_{i=1}^{r} b_i z^{r-i} \in \mathbb{F}_{2^w}[z]$$

is the characteristic polynomial of the recurrence and the $b_1, \ldots, b_r$ are constants in $\mathbb{F}_{2^w}$ The period of the recurrence is $2^{rw} - 1$ (maximal period) if and only if $P(z)$ is primitive.

To produce random numbers from these recurrences, we need to represent the elements of $\mathbb{F}_{2^w}$ with a given basis. The basis used in this package is the polynomial basis. Let $Q(z) \in \mathbb{F}_2[z]$ be an irreducible over $\mathbb{F}_2$ and $\zeta \in \mathbb{F}_{2^w}$ be one of its roots. A polynomial ordered basis is $(1, \zeta, \ldots, \zeta^{w-1})$. Under this basis, the recurrence (1) becomes

$$(\mathbf{q}_{n,1}, \ldots, \mathbf{q}_{n,r}) = (\mathbf{q}_{n,2}, \ldots, \mathbf{q}_{n,r}, 0) + (A_{b_1}, \ldots, A_{b_r})\mathbf{q}_{n,1} \tag{2}$$

where $\mathbf{q}_{n,i} \in \mathbb{F}_2^w$ is a w-bit vector that represents $q_{n,i}$ and A_{b_i} is the matrix that performs the multiplication by b_i under the polynomial basis.

The state of this generator at iteration n is

$$\mathbf{x}_n = (\mathbf{q}_{n,1}, \ldots, \mathbf{q}_{n,r}).$$

The output of the generator is obtained in two steps. The first one is the tempering. The tempered state $\mathbf{y}_n$ is obtained by

$$\mathbf{y}_n = B\mathbf{x}_n$$

where B is a $L \times k$ matrix and $\mathbf{y}_n = (y_n^{(0)}, \ldots, y_n^{(L-1)})$ is a L-bit vector. We obtain the output by

$$u_n = \sum_{i=1}^{L} y^{(i-1)} 2^{-i}.$$

In this package, we always use the values $w = L = 32$.

2 Streams

This package is designed so that a user can access many streams of pseudo-random numbers from the same generator. Its design was strongly influenced by RngStreams [1], a similar package but based on a different type of random number generator.

Let $G = \{u_n\}_{n\geq 0}$ be the sequence of numbers that a generator outputs, given a certain seed I_1. It can be divided in many subsequences called "streams". Each stream S_i, $i \geq 1$, comes from the sequence G and is defined as

$$S_g = \{s_{g,n}\}_{n\geq 0} = \{u_{(g-1)s+n}\}_{n\geq 0}$$

where s is the "inter-stream step". Usually, this value of s is very large. It is large enough so that any application would not come close to use more than s values. If it was not the case, then a stream could use values of other streams and that is not desirable. The seed of the generator that produces the sequence S_g is I_g. Notice that $G = S_1$.

A further division is possible. We divide the streams into "substreams". They are defined as

$$T_{g,h} = \{t_n\}_{n\geq 0} = \{s_{g,(h-1)t+n}\}_{n\geq 0} = \{u_{(g-1)s+(h-1)t+n}\}_{n\geq 0}.$$

The value of t is also large enough such that any application does not come close to use t values. Notice that $T_{g,1} = S_g$ for all $g \geq 1$.

To use streams from this package, one must first create a generator. This generator will become the "backbone" from which streams are created. To determine the sequence G, the user can choose a seed for the generator with the function `F2wStreams_setPackageState()`. If it is not called, then a default seed is given to the generator. Once the seed is chosen (or not), he can create streams. See the description of the package `F2wStreams` for the different functions that manipulate the streams. The substreams are not to be created. Once a stream is created, the stream is set to the start of the first substream. The user can advance to the next substream with the function `F2wStreams_resetNextSubstream()`.

For a given stream S_g, there are four seeds that are of interest: I_g, B_g, N_g, C_g. The seed I_g is the seed of the start of the g-th stream. It can be user defined or set to a default value. The seed B_g is the seed of the start of the current substream. The seed N_g is the start of the next substream. Finally, the seed C_g is the current state (seed) of the stream. In the description of the package, we use the words "seed" and "state" to designate the state of the generator.

3 A simple example

The figure 1 shows a program that produces random numbers using two streams coming from the same generator.

```
#include<stdio.h>
#include<stdlib.h>
#include"F2wStreams.h"
#include"F2wGenerator.h"
#include"F2wGenST.h"

int main(void){
  double uniform;
  F2wGenerator *G;
  F2wStream *S1,*S2;
  int i;
  G = F2wGenST_CreateNO(1);
  S1 = F2wStreams_Create (G);
  F2wStreams_WriteState(S1);
  for(i=0;i<25;i++)
    printf("%f\n",  F2wStreams_U01(S1));
  F2wStreams_WriteState(S1);

  S2 = F2wStreams_Create (G);
  F2wStreams_WriteState(S2);

  for(i=0;i<25;i++){
    uniform = F2wStreams_U01(S2);
    printf("%f\n",uniform);
  }
  F2wStreams_WriteState(S2);
  F2wStreams_Delete(S1);
  F2wStreams_Delete(S2);
  F2wGenerator_Delete(G);

  return 0;
}
```

Figure 1: Example of a program that uses streams.

APPENDIX

A. Functional definitions of the modules in F2wStreams

F2wStreams

This module implements the functions that manipulate the streams.

```
#include "F2wGenerator.h"

typedef struct stream {
  F2wGenerator *BackBoneGen;
  F2wElement *stateI;
  F2wElement *stateC;
  F2wElement *stateB;
  F2wElement *stateN;
  char anti;
  char descriptor[100];
  int l;
  int r;
} F2wStream;
```

This type contains the information needed by a stream. The pointer `BackBoneGen` points to the backbone generator of the stream. The arrays `stateI`, `stateC`, `stateB` and `stateN` contain the seeds I_g, C_g, B_g and N_g The flag `anti` indicates if antithetic variables must be output. The string `descriptor` identifies the stream. The variable `r` is the degree r of the caracteristic polynomial over $\mathbb{F}_{2^w}$ of the recurrence on which the generator is based.

```
F2wStream* F2wStreams_CreateWithDesc ( F2wGenerator *F2wGen,
                                       char descriptor[100]
                                     );
```

Function that creates a stream with the generator pointed by `F2wGen`. The string `descriptor` is used to identify the stream.

```
F2wStream* F2wStreams_Create ( F2wGenerator *F2wGen
                             );
```

Function that creates a stream with the generator pointed by `F2wGen`. The string "# g" is given as the descriptor for the g-th stream created.

```
void F2wStreams_Delete ( F2wStream *S
                       );
```

Function that deletes the stream pointed by `S`.

```
void F2wStreams_resetStartStream ( F2wStream *S
                                 );
```

Function that reset the current state to the start of the current stream. In other words, it sets C_g to I_g.

```
void F2wStreams_resetStartSubstream ( F2wStream *S
                                    );
```

Function that reset the current state to the start of the current substream. In other words, it sets C_g to B_g.

```
void F2wStreams_resetNextSubstream ( F2wStream *S
                                   );
```

Function that sets the current state to the start of the next substream (it sets C_g and B_g to N_g).

```
void F2wStreams_setAntithetic ( F2wStream *S,
                                char a
                              );
```

Function that sets the flag `anti` of the `F2wStream` pointed by `S`. If `a` is 0, then uniform random variables are outputted, otherwise, antithetic uniform random variables are outputted.

```
void F2wStreams_setState ( F2wStream *S ,
                           F2wElement *state
                         );
```

Function that sets the seeds I_g, B_g, C_g to the seed given by the array `state`.

```
void F2wStreams_getState ( F2wStream *S ,
                           F2wElement *state
                         );
```

Function that puts in the array `state` the value of the current state C_g.

```
void F2wStreams_WriteState ( F2wStream *S
                           );
```

Function that writes to the standard output the value of the current state C_g.

```
void F2wStreams_WriteFullState ( F2wStream *S
                               );
```

Function that writes to the standard output the value of the current state C_g, the intitial state I_g and the seed of the current substream B_g.

```
double F2wStreams_U01 ( F2wStream *S
                      );
```

Function that returns a uniform random variable (or an antithetic uniform random variable if `S->anti` is set to 1) from the interval $[0, 1)$.

```
int F2wStreams_randInt ( F2wStream *Gen,
                         int i,
                         int j
                       );
```

Function that returns a uniform random variable (or an antithetic uniform random variable if `S->anti` is set to 1) from the the set $\{i, \ldots, j\}$. It makes one call to `F2wStreams_U01()`.

F2wGenST

This module implements generators. The name `F2wGenST` refers to generators that use Small Tables (ST) to implement the multiplications by the coefficients b_i. The technique used is explained in [2], in which it is referred to as the "second method" of implementation. The implementation in language C is also explained in [2].

This implementation deals with the case where

$$P(z) = z^r - b_{r-t}z^t - b_{r-q}z^q - b_r$$

where b_{r-q} can be zero or not.

At the output, a Matsumoto-Kurita tempering is applied. It is done in the following fashion:

$$\begin{aligned} \mathbf{y}_n &\leftarrow \text{trunc}_{32}(\mathbf{x}_n) \\ \mathbf{y}_n &\leftarrow \mathbf{y}_n \oplus ((\mathbf{y}_n \ll 7) \,\&\, \mathbf{b}) \\ \mathbf{y}_n &\leftarrow \mathbf{y}_n \oplus ((\mathbf{y}_n \ll 15) \,\&\, \mathbf{c}) \end{aligned}$$

where $\text{trunc}_{32}(\mathbf{x}_n)$ returns only the 32 most significant bits of $\mathbf{x}_n$, $\oplus$ is a bit-to-bit XOR, & is a bit-to-bit AND and $(\mathbf{y}_n \ll s)$ shifts the vector $\mathbf{y}_n$ s positions to the left.

The best way to use this module is to create a generator with the function `F2wGenST_CreateNO()`. It creates one of the generator of Table 1. This table is stored in the file `F2wGenST.dat`.

```
typedef struct generator {
  int r;
  int t;
  int q;
  unsigned int modQ;
  F2wElement BRminusT;
  F2wElement BRminusQ;
  F2wElement BR;
  F2wElement *tabBRmT;
  F2wElement *tabBRmQ;
  F2wElement *tabBR;
  int t0,t1;
  int q0,q1;
  int r0,r1;
  unsigned int temper_b;
  unsigned int temper_c;
  int sizetable;
} F2wGenST;
```

This type holds the data necessary for the implementation of the generators. The values `r`, `t` and `q` are the values r, t and q. The variable `modQ` holds the polynomial $Q(z)$. The values `BRminusT`, `BRminusQ` and `BR` contain the coefficients b_{r-t}, b_{r-q} and b_r. The 32-bit vectors `temper_b` and `temper_c` contain the vectors **b** and **c**. The other variables are initialized with the function `F2wGenST_Create()` and concern the implementation.

Table 1: Generators of periods $2^{800} - 1$, $r = 25$, $w = 32$

No	r	t	q	$\mathbf{b}_{r-t}$	$\mathbf{b}_{r-q}$	$\mathbf{b}_r$	$Q(z)$	b	c
size of tables 2^3									
1	25	11	–	30000000	–	50000000	e307bc0e	f7b31a80	af530001
2	25	11	–	30000000	–	50000000	e307bc0e	f0ba1601	ab4b0000
3	25	14	–	60000000	–	30000000	e307bc0e	cd1a1000	8dc70001
size of tables 2^7									
4	25	11	–	05000000	–	12000000	f282ea95	a6ea0881	4de58000
5	25	14	–	11000000	–	18000000	f282ea95	e18e4881	2d5f8001
6	25	9	–	09000000	–	28000000	f282ea95	fa3cc981	6cf88000
7	25	16	–	81000000	–	18000000	f282ea95	f7660f01	3fec0000
8	25	9	–	22000000	–	11000000	f282ea95	bff09280	59ca0000
9	25	9	–	60000000	–	28000000	f282ea95	2e89a401	de278000
size of tables 2^3									
10	25	21	6	30000000	c0000000	a0000000	e397e5c4	994aa401	5a9d8001
11	25	13	6	c0000000	30000000	a0000000	e397e5c4	9475ce01	e7ed0001
12	25	19	7	c0000000	60000000	90000000	e397e5c4	b3965001	2b6c8001
13	25	19	4	30000000	50000000	90000000	e397e5c4	366ac280	e7750001
14	25	18	8	60000000	90000000	50000000	e397e5c4	e2171580	9bd98001
15	25	13	11	a0000000	60000000	50000000	e397e5c4	d9769501	6ae58001
size of tables 2^7									
16	25	18	13	42000000	21000000	50000000	9f1f0184	c19ee400	7e778000
17	25	13	5	12000000	28000000	06000000	9f1f0184	9e60e080	736b0000
18	25	21	12	60000000	48000000	09000000	9f1f0184	ce20b000	785a8000
19	25	21	12	60000000	48000000	09000000	9f1f0184	68090201	c3cf8001
20	25	16	10	82000000	24000000	48000000	9f1f0184	26fc4000	69570000
21	25	20	9	05000000	06000000	41000000	9f1f0184	8d868001	505c8001

```
F2wGenerator* F2wGenST_Create ( int r,
                                int t,
                                int q,
                                F2wElement BR,
                                F2wElement BRminusT,
                                F2wElement BRminusQ,
                                unsigned int M ,
                                unsigned int temper_b,
                                unsigned int temper_c,
                                int sizetable
                              );
```

Function that creates a `F2wGenerator` with specific parameters. A user not familiar with this type of generator should use a function of the type `F2wGenST_CreateNO()`. The argument `sizetable` indicates that the implementation will use tables of size $2^{\texttt{sizetable}}$.

```
void F2wGenST_Delete ( F2wGenerator *Gen
                     );
```

Function that deletes a `F2wGenerator` created with the function `F2wGenST_Create`.

```
F2wGenerator* F2wGenST_CreateNO ( int number
                                );
```

Same function as `F2wGenST_Create()`. It creates the generator #`number` from Table 1.

F2wPolynomial

This module implements the arithmetic in $\mathbb{F}_{2^w}$ and $\mathbb{F}_{2^w}[z]/P(z)$ where $w = 32$. It is used by the other modules to compute the state of the generator 2^d steps ahead in the recurrence (1). The elements of $\mathbb{F}_{2^w}$ are represented with the polynomial basis $(1, \zeta, \ldots, \zeta^{w-1})$ where ζ is a generator of $\mathbb{F}_{2^w}$. Let $Q(z)$ be the minimal polynomial of ζ over $\mathbb{F}_2$.

The Type `F2wPolynomial` is used to store a polynomial $P(z) \in \mathbb{F}_{2^w}[z]$. The degree of the polynomial is r. The type `F2wElement` represents an element of $\mathbb{F}_{2^w}$ under the polynomial basis. It is a vector of $w = 32$ bits where the least significant bit represents $1 = \zeta^0$ and the most significant bit represents ζ^{31}. The type `F2wPzElement` represents an element ξ of $\mathbb{F}_{2^w}[z]/P(z)$. Let $\xi = \xi_1 + \xi_2 z + \cdots + \xi_{32} z^{31}$. The array that represents ξ is $[\xi_1, \ldots, \xi_{32}]$. The type `F2wPzElement` is an array of `F2wElement`'s.

A normal user should not have to use any of these functions.

```
#define F2w_W 32
```

`F2w_W` is the value of w for $\mathbb{F}_{2^w}$. It is set to 32.

```
typedef unsigned int F2wElement;
typedef unsigned int *F2wPzElement;

typedef struct polyf2wZ{
  int r;
  F2wElement *coeff;
  int *nocoeff;
  unsigned int modQ;
  int nbcoeff;
  F2wElement zeta_i[ F2w_W ];
  F2wPzElement *z_i;
} F2wPolynomial;
```

This type is used to represent a monic polynomial $P(z) = z^r - \sum_{i=1}^{r} b_i z^{r-i}$ in $\mathbb{F}_{2^w}[z]$. The degree of the polynomial is `r` and the number of non zero b_i's is `nbcoeff`. The polynomial $Q(z)$ is represented by the 32-bit vector `modQ`. Notice that the coefficient z^{32} of $Q(z)$ is not represented in `modQ`. The polynomial $P(z)$ is

$$z^r - \sum_{i=0}^{\texttt{nbcoeff}} \texttt{coeff[i]} z^{\texttt{nocoeff[i]}}.$$

The array `zeta_i` contains in `zeta_i[i]` the value ζ^{2i} and the array `z_i` contains in `z_i[i]` the value z^i mod $P(z)$. These arrays are used to make the computations faster.

```
F2wPolynomial*  F2wPolynomial_Create ( int r,
                                       int nbcoeff,
                                       F2wElement *coeff,
                                       int *nocoeff,
                                       unsigned int modQ
                                     );
```

Function that creates a polynomial $P(z) \in \mathbb{F}_{2^w}[z]$.

```
void F2wPolynomial_Delete ( F2wPolynomial *P
                          );
```

Function that deletes the polynomial $P(z) \in \mathbb{F}_{2^w}[z]$ pointed by P.

```
F2wPzElement F2wPolynomial_CreateF2wPzElement ( F2wPolynomial *P
                                              );
```

Function that creates a F2wPzElement (i.e. an element of $\mathbb{F}_{2^w}[z]/P(z)$).

```
void F2wPolynomial_DeleteF2wPzElement ( F2wPzElement elem
                                      );
```

Function that deletes a F2wPzElement.

```
F2wElement F2wPolynomial_multiply ( F2wElement alpha,
                                    F2wElement beta,
                                    F2wPolynomial *P
                                  );
```

Function that returns the result of the multiplication alpha * beta in $\mathbb{F}_{2^w}$.

```
void  F2wPolynomial_Multiplybyz ( F2wPzElement res ,
                                  F2wPzElement beta,
                                  F2wPolynomial *P
                                );
```

Function that puts into res the result of z times beta in $\mathbb{F}_{2^w}/P(z)$.

```
void  F2wPolynomial_Multiply ( F2wPzElement res,
                               F2wPzElement alpha,
                               F2wPzElement beta,
                               F2wPolynomial *P
                             );
```

Function that puts into res the result of the multiplication beta * alpha in $\mathbb{F}_{2^w}/P(z)$.

```
void  F2wPolynomial_ExponF2wP ( F2wPzElement res,
                                F2wPzElement beta ,
                                int d,
                                F2wPolynomial *P
                              );
```

Function that puts into res the result of beta^{2^d} in $\mathbb{F}_{2^w}/P(z)$.

F2wGenerator

This module is used to manage the backbone generators. In this package, there is no implementation of generators. The implementations can vary and they are found in other modules like F2wGenST. A variable of type F2wGenerator holds all the information needed by a generator to create streams.

A normal user should not have to use any of these functions.

```
#include"F2wPolynomial.h"
struct stream;
typedef struct f2wgen{
  F2wPolynomial *P;
  void          *ParamGenerator;
  void          (*DeleteParamsGenerator) (struct f2wgen * );
  double        (*functionU01)           (struct stream *);
  F2wPzElement  multNextSubstream;
  F2wPzElement  multNextStream;
  F2wElement    *NextStream;
  int           nbstream;
} F2wGenerator;
```

This type stores all the data of a backbone generator. The polynomial pointed by P contains the caracteristic polynomial of the generator. The pointer ParamGenerator points to a structure that hold the parameters of a specific implementation of the generator. For example, it can point to a variable of type F2wGenST. The pointers to functions DeleteParamsGenerator, functionU01 point to functions that are related to the specific implementation of the generator. The function pointed by DeleteParamsGenerator deallocates the memory taken by the structure pointed by ParamGenerator. The function pointed by functionU01 is the function that is used to produced random numbers and depend on the implementation. The variable multNextStream (multNextSubstream) contains the value in $\mathbb{F}_{2^w}[z]/P(z)$ that the state must be multiplied with in order to advance to the next stream (substream). The array NextStream is the initial state for the next stream to be created with this generator and nbstream is the number of streams created with this generator.

```
F2wGenerator* F2wGenerator_Create(
           void          *ParamGen,
           F2wPolynomial *P,
           void          (*DeleteParamsGenerator) (F2wGenerator * ),
           double        (*functionU01)         (struct stream* ),
           void          (*functionU01Vector)   (struct stream* , double *),
           int           VectorSize,
           int           NextStreamStep,
           int           NextSubStreamStep
          );
```

Function that creates a F2wGenerator. The parameter NextStreamStep indicates that the start of consecutive streams will be $2^{\texttt{NextStreamStep}}$ iterations apart and NextSubStreamStep indicates that the start of consecutive substreams will be $2^{\texttt{NextSubStreamStep}}$ iterations apart.

```
void F2wGenerator_Delete ( F2wGenerator *F2wGen
                         );
```

Function that deletes a F2wGenerator.

```
void F2wGenerator_setPackageState( F2wGenerator *F2wGen,
                                   F2wElement *state
                                 );
```

Function that sets the initial seed of the generator pointed by `F2wGen`. It will be the seed I_1 for the first stream created with the generator pointed by `F2wGen`.

References

[1] P. L'Ecuyer, R. Simard, E. J. Chen, and W. D. Kelton. An object-oriented random-number package with many long streams and substreams. *Operations Research*, 50(6):1073–1075, 2002.

[2] F. Panneton and P. L'Ecuyer. Random number generators based on linear recurrences in F_{2^w}. In H. Niederreiter, editor, *Monte Carlo and Quasi-Monte Carlo Methods 2002*, pages 367–378, Berlin, 2004. Springer-Verlag.

Bibliographie

[1] F. Black et M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81 :637–654, 1973.

[2] L. Blum, M. Blum, et M. Schub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2) :364–383, 1986.

[3] P. Bratley, B. L. Fox, et L. E. Schrage. *A Guide to Simulation.* Springer-Verlag, New York, deuxième édition, 1987.

[4] R. P. Brent. On the periods of generalized fibonacci recurrences. *Mathematics of Computation*, 63(207) :389–401, 1994.

[5] R. P. Brent, S. Larvala, et P. Zimmermann. A fast algorithm for testing reducibility of trinomials mod 2 and some new primitive trinomials of degree 3021377. *Math. Comp.*, 72(243) :1443–1452, 2003.

[6] M. Broadie et P. Glasserman. Estimating security price derivatives using simulation. *Management Science*, 42 :269–285, 1996.

[7] R. E. Caflisch et B. Moskowitz. Modified Monte Carlo methods using quasi-random sequences. Dans H. Niederreiter et P. J.-S. Shiue, éditeurs, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, numéro 106 dans Lecture Notes in Statistics, pages 1–16, New York, 1995. Springer-Verlag.

[8] F. Chabaud et R. Lercier. *A toolbox for fast computation in finite extension over finite rings*, 2000. Guide d'utilisation du logiciel. Voir `http://www.di.ens.fr/~zen/`.

[9] A. Compagner. Definitions of randomness. *American Journal of Physics*, 59 :700–705, 1991.

[10] A. Compagner. The hierarchy of correlations in random binary sequences. *Journal of Statistical Physics*, 63 :883–896, 1991.

[11] R. Couture et P. L'Ecuyer. Lattice computations for random numbers. *Mathematics of Computation*, 69(230) :757–765, 2000.

[12] R. Couture, P. L'Ecuyer, et S. Tezuka. On the distribution of k-dimensional vectors for simple and combined Tausworthe sequences. *Mathematics of Computation*, 60(202) :749–761, S11–S16, 1993.

[13] A. De Matteis, J. Eichenauer-Herrmann, et H. Grothe. Computation of critical distances within multiplicative congruential pseudorandom number sequences. *J. Comp. Appl. Math.*, **39** :49–55, 1992.

[14] A. De Matteis et S. Pagnutti. Parallelization of random number generators and long-range correlations. *Numer. Math.*, **53** :595–608, 1988.

[15] A. De Matteis et S. Pagnutti. A class of parallel random number generators. *Parallel Comput.*, **13** :193–198, 1990.

[16] A. De Matteis et S. Pagnutti. Long-range correlations in linear and non-linear random number generators. *Parallel Comput.*, **14** :207–210, 1990.

[17] A. De Matteis et S. Pagnutti. Long-range correlation analysis of the Wichmann-Hill random number generator. *Statistics and Computing*, **3** :67–70, 1993.

[18] A. De Matteis et S. Pagnutti. Controlling correlations in parallel Monte Carlo. *Parallel Computing*, 21 :73–84, 1995.

[19] E. J. Dudewicz et T. G. Ralley. *The Handbook of Random Number Generation and Testing with TESTRAND Computer Code.* American Sciences Press, Columbus, Ohio, 1981.

[20] K. Entacher, O. Lendl, A. Uhl, et S. Wegenkittl. Analyzing streams of pseudo-random numbers for parallel monte carlo integration. Dans R. Wyrzykowski,

H. Piech, B. Mochnacki, M. Vajtersic, et P. Zinterhof, éditeurs, *Proceedings of the International Workshop on Parallel Numerics (Parnum'97)*, pages 59–71. Zakopane, Poland, septembre 1997.

[21] H. Faure. Discrépance des suites associées à un système de numération en dimension s. *Acta Arithmetica*, 61 :337–351, 1982.

[22] M. Fushimi et S. Tezuka. The k-distribution of generalized feedback shift register pseudorandom numbers. *Communications of the ACM*, 26(7) :516–523, 1983.

[23] S. Khuller et Y. Matias. A simple randomized sieve algorithm for the closest-pair problem. *Information and Computation*, 118(1) :34–37, 1995.

[24] D. E. Knuth. *The Art of Computer Programming, Volume 2 : Seminumerical Algorithms*. Addison-Wesley, Reading, Mass., troisième édition, 1998.

[25] L. Kocis et W. J. Whiten. Computational investigations of low-discrepancy sequences. *ACM Transactions on Mathematical Software*, 23(2) :266–294, juin 1997.

[26] T. Kollig et A. Keller. Efficient multidimensional sampling. *Computer Graphics Forum*, 21(3) :557–563, 2002.

[27] H. Krawczyk. How to predict congruential generators. Dans G. Brassard, éditeur, *Advances in Cryptology : Proceedings of CRYPTO'89*, volume 435 de *Lecture Notes in Computer Science*, pages 138–153. Springer-Verlag, Berlin, 1990.

[28] A. M. Law et W. D. Kelton. *Simulation Modeling and Analysis*. McGraw-Hill, New York, troisième édition, 2000.

[29] P. L'Ecuyer. Efficient and portable combined random number generators. *Communications of the ACM*, 31(6) :742–749 et 774, 1988. Voir aussi la correspondance dans le même journal, 32, 8 (1989) 1019–1024.

[30] P. L'Ecuyer. Random numbers for simulation. *Communications of the ACM*, 33(10) :85–97, 1990.

[31] P. L'Ecuyer. Testing random number generators. Dans *Proceedings of the 1992 Winter Simulation Conference*, pages 305–313. IEEE Press, décembre 1992.

[32] P. L'Ecuyer. Uniform random number generation. *Annals of Operations Research*, 53 :77–120, 1994.

[33] P. L'Ecuyer. Maximally equidistributed combined Tausworthe generators. *Mathematics of Computation*, 65(213) :203–213, 1996.

[34] P. L'Ecuyer. Random number generation. Dans Jerry Banks, éditeur, *Handbook of Simulation*, pages 93–137. Wiley, 1998.

[35] P. L'Ecuyer. Random number generators and empirical tests. Dans P. Hellekalek, G. Larcher, H. Niederreiter, et P. Zinterhof, éditeurs, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, volume 127 de *Lecture Notes in Statistics*, pages 124–138. Springer, New York, 1998.

[36] P. L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research*, 47(1) :159–164, 1999.

[37] P. L'Ecuyer. Tables of maximally equidistributed combined LFSR generators. *Mathematics of Computation*, 68(225) :261–269, 1999.

[38] P. L'Ecuyer. Software for uniform random number generation : Distinguishing the good and the bad. Dans *Proceedings of the 2001 Winter Simulation Conference*, pages 95–105, Pistacaway, NJ, 2001. IEEE Press.

[39] P. L'Ecuyer. *SSJ : A Java Library for Stochastic Simulation*, 2001. Guide d'utilisation du logiciel.

[40] P. L'Ecuyer. Polynomial integration lattices. Dans H. Niederreiter, éditeur, *Monte Carlo and Quasi-Monte Carlo Methods 2002*, pages 73–98, Berlin, 2004. Springer-Verlag.

[41] P. L'Ecuyer, J.-F. Cordeau, et R. Simard. Close-point spatial tests and their application to random number generators. *Operations Research*, 48(2) :308–317, 2000.

[42] P. L'Ecuyer et C. Lemieux. Quasi-Monte Carlo via linear shift-register sequences. Dans *Proceedings of the 1999 Winter Simulation Conference*, pages 632–639. IEEE Press, 1999.

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

[44] P. L'Ecuyer, L. Meliani, et J. Vaucher. SSJ : A framework for stochastic simulation in Java. Dans E. Yücesan, C.-H. Chen, J. L. Snowdon, et J. M. Charnes, éditeurs, *Proceedings of the 2002 Winter Simulation Conference*, pages 234–242. IEEE Press, 2002.

[45] P. L'Ecuyer et F. Panneton. A new class of linear feedback shift register generators. Dans J. A. Joines, R. R. Barton, K. Kang, et P. A. Fishwick, éditeurs, *Proceedings of the 2000 Winter Simulation Conference*, pages 690–696, Pistacaway, NJ, 2000. IEEE Press.

[46] P. L'Ecuyer et F. Panneton. Construction of equidistributed generators based on linear recurrences modulo 2. Dans K.-T. Fang, F. J. Hickernell, et H. Niederreiter, éditeurs, *Monte Carlo and Quasi-Monte Carlo Methods 2000*, pages 318–330. Springer-Verlag, Berlin, 2002.

[47] P. L'Ecuyer et R. Proulx. About polynomial-time "unpredictable" generators. Dans *Proceedings of the 1989 Winter Simulation Conference*, pages 467–476. IEEE Press, décembre 1989.

[48] P. L'Ecuyer et R. Simard. *TestU01 : A Software Library in ANSI C for Empirical Testing of Random Number Generators*, 2002. Guide d'utilisation du logiciel.

[49] P. L'Ecuyer, R. Simard, E. J. Chen, et W. D. Kelton. An object-oriented random-number package with many long streams and substreams. *Operations Research*, 50(6) :1073–1075, 2002.

[50] P. L'Ecuyer et R. Touzin. Fast combined multiple recursive generators with multipliers of the form $a = \pm 2^q \pm 2^r$. Dans J. A. Joines, R. R. Barton, K. Kang, et P. A. Fishwick, éditeurs, *Proceedings of the 2000 Winter Simulation Conference*, pages 683–689, Pistacaway, NJ, 2000. IEEE Press.

[51] C. Lemieux et P. L'Ecuyer. Efficiency improvement by lattice rules for pricing asian options. Dans D. J. Medeiros, E. F. Watson, J. S. Carson, et M. S. Manivannan, éditeurs, *Proceedings of the 1998 Winter Simulation Conference*, pages 579–586, Piscataway, NJ, 1998. IEEE Press.

[52] C. Lemieux et P. L'Ecuyer. Using lattice rules for variance reduction in simulation. Dans J. A. Joines, R. R. Barton, K. Kang, et P. A. Fishwick, éditeurs, *Proceedings of the 2000 Winter Simulation Conference*, pages 509–516, Piscataway, NJ, 2000. IEEE Press.

[53] C. Lemieux et P. L'Ecuyer. Randomized polynomial lattice rules for multivariate integration and simulation. *SIAM Journal on Scientific Computing*, 24(5) :1768–1789, 2003.

[54] A. K. Lenstra. Factoring multivariate polynomials over finite fields. *Journal of Computer and System Sciences*, 30 :235–248, 1985.

[55] T. G. Lewis et W. H. Payne. Generalized feedback shift register pseudorandom number algorithm. *Journal of the ACM*, 20(3) :456–468, 1973.

[56] R. Lidl et H. Niederreiter. *Introduction to Finite Fields and Their Applications.* Cambridge University Press, Cambridge, édition revisée, 1994.

[57] K. Mahler. On a theorem in the geometry of numbers in a space of Laurent series. *Journal of Number Theory*, 17 :403–416, 1983.

[58] G. Marsaglia. DIEHARD : a battery of tests of randomness. Voir `http://stat.fsu.edu/~geo/diehard.html`, 1996.

[59] G. Marsaglia. The Marsaglia random number CDROM including the DIEHARD battery of tests of randomness. Voir `http://stat.fsu.edu/pub/diehard`, 1996.

[60] G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14) :1–6, 2003. Voir `http://www.jstatsoft.org/v08/i14/xorshift.pdf`.

[61] G. Marsaglia, B. Narasimhan, et A. Zaman. A random number generator for PC's. *Computer Physics Communications*, 60 :345–349, 1990.

[62] J. L. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inf. Theor*, IT-15 :122–127, 1969.

[63] J. Matoušěk. On the L2-Discrepancy of Anchored Boxes. *Journal of Complexity*, 14 :527–556, 1998.

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

[65] M. Matsumoto. Simple cellular automata as pseudo-random m-sequence generators for built-in self test. *ACM Transactions on Modeling and Computer Simulation*, 8(1) :31–42, 1998.

[66] M. Matsumoto et Y. Kurita. Twisted GFSR generators. *ACM Transactions on Modeling and Computer Simulation*, 2(3) :179–194, 1992.

[67] M. Matsumoto et Y. Kurita. Twisted GFSR generators II. *ACM Transactions on Modeling and Computer Simulation*, 4(3) :254–266, 1994.

[68] M. Matsumoto et Y. Kurita. Strong deviations from randomness in m-sequences based on trinomials. *ACM Transactions on Modeling and Computer Simulation*, 6(2) :99–106, 1996.

[69] M. Matsumoto et T. Nishimura. Mersenne twister : A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1) :3–30, 1998.

[70] M. Matsumoto et T. Nishimura. Dynamic Creation of Pseudorandom Number Generators. Dans H. Niederreiter et J. Spanier, éditeurs, *Monte Carlo and Quasi-Monte Carlo Methods 1998*, pages 56–69, Berlin, 2000. Springer.

[71] R. J. McEliece. *Finite Fields for Computer Scientists and Engineers*, volume 23 de *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, 1986.

[72] A. J. Menezes, S. A. Vanstone, et P. C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.

[73] H. Niederreiter. Point sets and sequences with small discrepancy. *Monatshefte für Mathematik*, 104 :273–337, 1987.

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

[75] H. Niederreiter. Factorization of polynomials and some linear-algebra problems over finite fields. *Linear Algebra Appl.*, 192 :301–328, 1993.

[76] H. Niederreiter. The multiple-recursive matrix method for pseudorandom number generation. *Finite Fields and their Applications*, 1 :3–30, 1995.

[77] H. Niederreiter. Pseudorandom vector generation by the multiple-recursive matrix method. *Mathematics of Computation*, 64(209) :279–294, 1995.

[78] H. Niederreiter et C. Xing. The algebraic-geometry approach to low-discrepancy sequences. Dans P. Hellekalek, G. Larcher, H. Niederreiter, et P. Zinterhof, éditeurs, *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing*, volume 127 de *Lecture Notes in Statistics*, pages 139–160, New York, 1997. Springer-Verlag.

[79] T. Nishimura. Tables of 64-bit Mersenne twisters. *ACM Transactions on Modeling and Computer Simulation*, 10(4) :348–357, 2000.

[80] A. B. Owen. Variance with alternative scramblings of digital nets. *ACM Transactions on Modeling and Computer Simulation*, 13(4) :363–378, 2003.

[81] N. Packard et S. Wolfram. Two-dimensional cellular automata. *J. Statistical Physics*, 38,5/6 :901–946, 1985.

[82] F. Panneton. Générateurs de nombres aléatoires utilisant des récurrences linéaires modulo 2. Mémoire de maîtrise, Département d'informatique et de recherche opérationnelle, Université de Montréal, 2000.

[83] F. Panneton. *F2WSTREAMS*, 2003. Guide d'utilisation du logiciel.

[84] F. Panneton. *REGPOLY*, 2003. Guide d'utilisation du logiciel.

[85] F. Panneton et P. L'Ecuyer. Random number generators based on linear recurrences in F_{2^w}. Dans H. Niederreiter, éditeur, *Monte Carlo and Quasi-Monte Carlo Methods 2002*, pages 367–378, Berlin, 2004. Springer-Verlag.

[86] K. Paul, S.P. Chaudhuri, R. Ghosal, B. Sikdar, et D.R. Chowdhury. GF(2^p) CA based vector quantization for fast encoding of still images. Dans *Proc. of VLSI Design*, Inde, 2000.

[87] K. Paul et D.R. Chowdhury. Applications of GF(2^p) CA in burst error correcting codes generator. Dans *Proc. of VLSI Design*, Inde, 2000.

[88] G. Pirsic et W. Ch. Schmid. Calculation of the quality parameter of digital nets and application to their construction. *Journal of Complexity*, 17(4) :827–839, 2001.

[89] J. B. Plumstead. Inferring a sequence generated by a linear congruence. Dans *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 153–159, 1982.

[90] F. P. Preparata et M. I. Shamos. *Computational Geometry : An Introduction.* Texts and Monographs in Computer Science. Springer-Verlag, New York, 1985.

[91] I. Radovic, I.M. Sobol, et R. F. Tichy. Quasi-monte carlo methods for numerical integration : comparison of different low-discrepancy sequences. *Monte Carlo Methods and Applications*, 2(1) :1–14, 1996.

[92] A. Rieke, A.-R. Sadeghi, et W. Poguntke. On primitivity tests for polynomials. Dans *Proceedings of the 1998 IEEE International Symposium on Information Theory*. Cambridge, MA.

[93] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, et S. Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Publication spéciale de NIST 800-22, National Institute of Standards and Technology (NIST), Gaithersburg, Maryland, USA, 2001. Voir `http://csrc.nist.gov/rng/`.

[94] W. Ch. Schmid. The exact quality parameter of nets derived from sobol' and niederreiter sequences. *Recent Advances in Numerical Methods and Applications*, pages 287–295, 1999.

[95] B.K. Sikdar, K. Paul, G.P. Biswas, C.Yang, V. Bopanna, S.Mukherjee, et P.P. Chaudhuri. Theory and applications of GF(2^p) cellular automata as on-chip test pattern generator. Dans *Proc. of VLSI Design*, pages 556–561, Inde, 2000.

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

[97] M. Smid. Closest-point problems in computational geometry. Dans Jörg-Rudiger Sack et Jorge Urrutia, éditeurs, *Handbook of Computational Geometry*, chapitre 20, pages 877–935. Elsevier, 2000.

[98] I. M. Sobol'. The distribution of points in a cube and the approximate evaluation of integrals. *U.S.S.R. Comput. Math. and Math. Phys.*, 7 :86–112, 1967.

[99] I. M. Sobol'. Uniformly distributed sequences with an additional uniform property. *USSR Comput. Math. Math. Phys. Academy of Sciences*, 16 :236–242, 1976.

[100] R. C. Tausworthe. Random numbers generated by linear recurrence modulo two. *Mathematics of Computation*, 19 :201–209, 1965.

[101] S. Tezuka. The k-dimensional distribution of combined GFSR sequences. *Mathematics of Computation*, 62(206) :809–817, 1994.

[102] S. Tezuka. A unified view of large-period random number generators. *Journal of the Operations Research Society of Japan*, 37 :211–227, 1994.

[103] S. Tezuka. *Uniform Random Numbers : Theory and Practice.* Kluwer Academic Publishers, Norwell, Mass., 1995.

[104] S. Tezuka et M. Fushimi. A method of designing cellular automata as pseudorandom number generators for built-in self-test for vlsi. *Contemp. Math.*, 168 :363–367, 1994.

[105] E. Thomé. Computation of discrete logarithms in GF(2^607). Dans *Advances in Cryptology, Asiacrypt'2001*, numéro 2369 dans Lecture Notes in Computer Science, pages 107–124. Springer-Verlag, 2001.

[106] R. E. Walpole, R. H. Myers, et S. L. Myers. *Probability and Statistics for Engineers and Scientists.* Prentice-Hall, Inc., New Jersey, sixième édition, 1998.

[107] S. Wolfram. Statistical mechanics of cellular automata. *Rev. Modern Physics*, 55 :601–644, 1983.