Université de Montréal

Des générateurs récursifs multiples combinés rapides avec des coefficients de la forme $\pm 2^{p_1} \pm 2^{p_2}$

par

Renée Touzin

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures

en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)

en informatique option recherche opérationnelle

mai 2001

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé:

Des générateurs récursifs multiples combinés rapides avec des coefficients de la forme $\pm 2^{p_1} \pm 2^{p_2}$

présenté par:

Renée Touzin

a été évalué par un jury composé des personnes suivantes:

Felisa J. Vázquez-Abad

(président-rapporteur)

Pierre L'Écuyer

(directeur de recherche)

Patrice Marcotte

(membre du jury)

Mémoire accepté le:

Sommaire

Les générateurs à congruence linéaire simples ou combinés sont le type de générateurs de nombres aléatoires le plus couramment utilisé. Leur fonctionnement étant simple mais non simpliste, ils sont faciles à analyser. Les propriétés recherchées sont une longue période, une bonne uniformité et une vitesse rapide de production des nombres.

Dans ce mémoire, nous nous penchons sur la vitesse de production des nombres par des *générateurs récursifs multiples* (GRM). Notre but est de dépasser la vitesse des générateurs existants tout en ne négligeant pas les autres qualités du générateur.

À la base des générateurs à congruence linéaire se situe l'opération $y := ax \bmod m$. Dans un premier temps, nous faisons un tour d'horizon des méthodes existantes qui calculent rapidement cette opération. Une des méthodes est celle de Wu [44], que nous appelons décomposition en puissances de 2. Elle utilise des coefficients de la forme $a = \pm 2^{p_1} \pm 2^{p_2}$ et des modules de la forme $m = 2^e - h$. Jusqu'à présent, aucun GRM utilisant la décomposition en puissances de 2 et ayant de bonnes propriétés théoriques, du point de vue de l'uniformité des points produits, n'avait été proposé. Nous avons également montré théoriquement que la méthode proposée récemment par Deng et Lin [7] est mauvaise.

Puis, nous avons programmé quelques fonctions permettant de faire des recherches de générateurs avec des coefficients de la forme $a = \pm 2^{p_1} \pm 2^{p_2}$ et nous les avons intégrées au logiciel `Latmrg` [27]. Le test spectral, test couramment utilisé pour vérifier les qualités théoriques d'un générateur linéaire, a permis de départager, au départ, les bons et les mauvais générateurs. Nous avons cherché et trouvé une liste de générateurs simples et combinés.

Des tests statistiques furent appliqués à nos générateurs pour tester empiriquement l'hypothèse $\mathcal{H}_0$: «les valeurs produites par le générateur peuvent être vues comme des variables aléatoires indépendantes suivant la loi uniforme sur l'intervalle $(0, 1)$.» Les générateurs

passèrent facilement les tests.

Nous avons aussi testé la vitesse de nos générateurs et nous l'avons comparée avec celle de générateurs existants [19, 23]. Nos générateurs, programmés dans le langage C, possèdent une plus grande vitesse d'exécution que tout autre générateur à congruence linéaire ayant de bonnes propriétés théoriques et une période équivalente.

Table des matières

Liste des figures

Liste des tableaux

Remerciements

J'aimerais tout d'abord remercier mon directeur de maîtrise, Pierre L'Ecuyer, pour ses précieux conseils, son aide et le support financier qu'il m'a accordés tout au long de ces deux années.

Je voudrais également offrir mes remerciements au Centre de Recherche sur les Transports (CRT) et aux entreprises GIRO inc. pour les bourses qu'ils m'ont octroyées afin de me permettre d'effectuer ma maîtrise.

Je remercie le CRSNG qui m'a accordé une bourse pour les deux dernières sessions de ma maîtrise.

Je remercie Jacinthe Granger-Piché qui a lu mon mémoire et m'a offert ses commentaires constructifs ainsi que Richard Simard qui m'a aidé pour les tests statistiques et qui a bien voulu répondre à plusieurs de mes questions informatiques.

Je remercie Annie Pronovost qui a accepté de jeter un coup d'oeil à la qualité du français de ce mémoire.

Je remercie Danielle pour son encouragement et Dominique pour ses conseils rationnels.

Enfin, je remercie mes parents, pour l'encouragement et le soutien financier qu'ils m'ont accordés depuis le début de mes études universitaires.

Chapitre 1

Introduction

Plusieurs branches de l'informatique, et plus particulièrement la simulation, exigent une grande quantité de nombres aléatoires. En effet, lors de simulations, nous avons besoin de variables aléatoires qui suivent différentes distributions statistiques comme la gaussienne, la poisson, la gamma, l'uniforme,... L'emploi de diverses techniques comme la méthode d'inversion ou de composition permet de transformer la distribution uniforme sur l'intervalle $(0, 1)$ vers les autres distributions mentionnées [16]. Ces techniques de transformation ne sont pas abordées dans le présent mémoire. Cependant, il faut remarquer qu'à partir du moment où nous sommes capables de générer adéquatement des variables *indépendantes et identiquement distribuées* (i.i.d.) *uniformes sur l'intervalle* $(0, 1)$ $(U(0, 1))$, nous pouvons reproduire les autres distributions statistiques. La génération de valeurs aléatoires i.i.d. $U(0, 1)$ est donc primordiale.

Pour approximer sur un ordinateur une distribution uniforme $(0, 1)$, nous avons recours à une structure mathématique appelée générateur de nombres pseudo-aléatoires. La structure mathématique est une récurrence déterministe qui, vue de l'extérieur, donne l'illusion de se comporter de manière stochastique. Le mot «pseudo» rappelle que le générateur est déterministe. Dans ce mémoire, nous nous intéressons à un type de générateur particulier, nommé *générateur récursif multiple* ou GRM.

Un GRM [14, 16, 21] est défini par une récurrence de la forme :

$$x_n = (a_1 x_{n-1} + a_2 x_{n-2} + \ldots + a_k x_{n-k}) \bmod m, \tag{1.1}$$

$$u_n = x_n / m. \tag{1.2}$$

Le vecteur $(x_{n-k}, ..., x_{n-1})$ représente l'état du générateur et les constantes $a_1, ..., a_k$ sont les coefficients ou multiplicateurs. La variable m est appelée le module et k l'ordre du générateur.

Une définition complète des caractéristiques du GRM est donnée au chapitre 2.

Avant de proposer un générateur particulier, nous lui faisons passer certains tests. Il est d'usage courant de débuter par appliquer le test spectral [8, 10, 14, 19, 23, 44] aux générateurs de type linéaire dont fait partie le GRM. Ce test, dit théorique, sert de premier critère de sélection afin de déterminer de «bons» générateurs. L'origine du test, son fonctionnement ainsi que les motivations qui nous incitent à l'utiliser comme critère de base sont expliqués au chapitre 2.

Par la suite, nous appliquons des tests statistiques [28, 32, 31]. Ceux-ci mesurent le comportement de nos générateurs sous des aspects différents du test spectral. Ces tests font l'objet du chapitre 7. Toutefois, nous abordons la méthodologie générale des tests statistiques dès le chapitre 2.

Les simulations contemporaines sont de plus en plus voraces en ce qui a trait à la quantité de nombres exigés ; une grande vitesse d'exécution de la part d'un générateur constitue donc une qualité non négligeable. La difficulté lors de la construction de générateurs se situe dans le regroupement de tous les attributs requis pour produire un bon générateur, c'est-à-dire obtenir un résultat acceptable au test spectral, passer les tests statistiques et une production des nombres suffisamment rapide.

Actuellement, il existe des générateurs [19, 23] qui satisfont assez bien toutes les propriétés énumérées précédemment. Cependant l'expression «toujours plus vite» est devenue une maxime en informatique. Ainsi, le but ultime de cette recherche est la confection de générateurs plus rapides sans négliger la qualité de l'uniformité des points produits.

Au chapitre 2, nous présentons quelques résultats connus sur les GRM. Nous y introduisons les concepts de base de la génération de nombres aléatoires. Nous abordons quelques types de générateurs existants, ainsi que leurs propriétés. Nous expliquons l'origine du test spectral et nous décrivons son fonctionnement et la manière dont nous l'exploitons. Dans ce même chapitre, nous présentons une introduction aux tests statistiques. Nous expliquons la méthodologie générale des tests statistiques, de même que la façon exacte dont nous nous en servons dans le cas de tests sur des générateurs.

Nous abordons, au chapitre 3, quelques techniques existantes qui améliorent la vitesse d'exécution des générateurs. Nous expliquons en particulier l'idée de Wu [44]. Cette idée permet d'éliminer certaines multiplications et divisions afin d'améliorer la vitesse. Nous exposons un résumé d'un article de L'Ecuyer et Simard [30] dans lequel les auteurs démontrent, à l'aide de tests statistiques, que les sorties produites par les générateurs de Wu [44] sont fortement corrélées.

Au chapitre 4, nous expliquons un type particulier de GRM proposé récemment par Deng et Lin [7]. Nous démontrons, par l'utilisation judicieuse d'inégalités provenant du test spectral, que leur méthode possède des défauts importants.

Nous débutons le chapitre 5 par la présentation de conditions pratiques que doivent rencontrer nos générateurs. La méthodologie employée pour l'ensemble de notre recherche ainsi qu'un schéma du processus de construction des générateurs y sont aussi présentés.

Le chapitre 6 présente la différence entre des recherches de générateurs aléatoires et exhaustives. La plupart du temps, nous devons nous limiter à des recherches aléatoires mais lorsque cela est possible, nous effectuons plutôt des recherches exhaustives. Nous présentons les paramètres trouvés lors des recherches. Nous terminons le chapitre 6 par l'analyse des résultats obtenus jusqu'à présent. La première étape pour la construction de générateurs rapides est franchie.

Au chapitre 7, nous décrivons et justifions les tests statistiques employés. Il s'agit d'une deuxième étape pour les générateurs, qui après avoir obtenu de bons résultats au test spectral, doivent se soumettre à des tests statistiques. Les résultats aux tests statistiques sont présentés et analysés.

Il ne reste qu'à déterminer si les générateurs que nous avons sont plus rapides que des générateurs existants [19, 23]. Nous présentons, au chapitre 8, les temps de production des nombres avec divers compilateurs, puis nous comparons ces temps avec ceux d'autres générateurs réputés rapides. Les programmes, en langage C, se retrouvent à l'annexe I.

Le chapitre 9 présente les modifications apportées aux programmes existants [27] de recherche de générateurs. Nous décrivons plus spécifiquement la programmation faite. Nous concluons,

au chapitre 10, par un résumé de la recherche effectuée ainsi qu'avec quelques idées sur ce qui pourrait se faire dans le futur.

La contribution

La principale contribution de ce mémoire est la construction de générateurs récursifs multiples combinés qui possèdent une plus grande vitesse de calcul que tout autre générateur récursif multiples combiné existant et ayant de bonnes propriétés d'uniformité. Nos générateurs vont jusqu'à deux fois plus vite sur des architectures de type Pentium, tout en possédant de bonnes propriétés théoriques et en passant avec succès les tests statistiques les plus courants [28, 32, 31].

Avant de passer à l'aspect recherche des générateurs, nous avons regardé d'autres types de GRM existants qui, grâce à l'utilisation de techniques particulières, possèdent une bonne vitesse de calcul. Les techniques analysées sont la factorisation approximative [1, 4, 41], la méthode en point flottant [22] et la méthode de Wu [44]. Nous nous sommes aussi penchés sur un type particulier de GRM proposé par Deng et Lin [7]. Nous avons démontré que la méthode de Deng et Lin n'est pas bonne si elle est appliquée telle quelle. La démonstration a été faite à l'aide de bornes provenant du test spectral.

Nous avons généralisé la méthode de Wu, que nous avons baptisée décomposition en puissances de 2. Les coefficients permis peuvent se composer de λ puissances de 2 et les modules sont de la forme $m = 2^e - h$, $e, h \in \mathbb{N}$, comme proposé dans [30].

Nous nous sommes attaqués par la suite à la construction de générateurs qui utilisent la méthode de décomposition en puissances de 2. Une fois en possession de bons générateurs vis-à-vis du test spectral, nous les avons soumis à des tests statistiques. Nous avons programmé les générateurs en langage C et nous les avons testés sous divers compilateurs et architectures d'ordinateurs pour déterminer les meilleures implantations.

Pour effectuer la recherche des nouveaux générateurs, il nous a fallu programmer quelques fonctions que nous avons intégrées au logiciel `LatMRG` [27]. Comme nous l'expliquons dans le guide d'utilisation modifié, au chapitre 9, le programme `seeks` [27], qui fait partie du logiciel LatMRG, pouvait déjà effectuer des recherches de générateurs avec des coefficients de la forme

$\pm 2^{p_1} \pm 2^{p_2}$. Toutefois, à peu près aucune latitude n'était présente pour effectuer les recherches de générateurs. Notons seulement qu'il n'était permis que de faire des recherches exhaustives. Comme nous le verrons, ceci est pratiquement impossible si nous désirons regarder du côté des GRM d'ordre 6, par exemple. Le nombre de combinaisons de coefficients à tester devient rapidement trop grand.

Pour le cas où le coefficient est de la forme $\pm 2^{p_1} \pm 2^{p_2}$, nous avons dérivé une fonction qui calcule exactement le nombre de coefficients admissibles lorsque nous avons une borne inférieure et supérieure sur la valeur des exposants p_1 et p_2. Cette formule nous a permis de comprendre comment étaient formés les coefficients et avec quelle fréquence nous les retrouvons dans un certain intervalle. Cette même formule nous a servi lors de l'analyse de la répétition de quelques coefficients. En effet, certains nombres peuvent s'écrire de quelques manières différentes avec des puissances de 2, par exemple $2^4 - 2^2 = 2^3 + 2^2$. Lorsque nous effectuons les recherches aléatoires, l'algorithme que nous utilisons ne se soucie pas de cet aspect de répétitions. Nous avons voulu déterminer s'il y avait une grande fraction du temps perdu à tester, possiblement, le même générateur.

Chapitre 2

La théorie

Le chapitre 2 nous introduit à quelques types de générateurs de nombres pseudo-aléatoires. Nous donnons les caractéristiques des générateurs présentés ainsi qu'une partie de la théorie s'y rattachant. Nous poursuivons par une description et une explication du test spectral. Il s'agit d'un test théorique qui sert à classer tous les générateurs par rapport à un critère particulier qui sera décrit en détails.

Nous retrouvons par la suite un résumé des principaux avantages et défauts des générateurs à congruence linéaire, le type de générateurs auquel nous nous intéressons. Cette section nous permet de relever clairement les défauts de nos générateurs mais aussi de mettre en lumière, dès le départ, leurs avantages.

Nous présentons une revue de la méthodologie statistique utilisée au cours de ce mémoire. Nous éclairons le lecteur sur les raisons qui nous poussent à faire usage de tests statistiques dans le cadre de l'analyse de générateurs de nombres aléatoires.

2.1 Les généralités

Nous présentons tout d'abord une définition d'un générateur de nombres pseudo-aléatoires [21].

Définition 2.1.1 Un *générateur pseudo-aléatoire* est une structure mathématique de la forme $\mathcal{G} = (S, s_0, T, U, G)$ où,

S est un ensemble fini d'états,

s_0 est l'état initial dans S,

$T : S \to S$ est la fonction de transition,

U est un ensemble fini de symboles de sortie,

$G : S \to U$ est la fonction de sortie.

À partir d'un état initial, s_0, le générateur évolue de manière déterministe vers d'autres états selon la récurrence $s_n = T(s_{n-1})$ où $s_n \in S$, $n \geq 1$. La fonction de sortie G associe à l'état s_n, le symbole de sortie $u_n = G(s_n) \in U$.

Puisque S est un ensemble fini d'états, le générateur doit immanquablement cycler, c'est-à-dire revenir à un état déjà visité. Le nombre d'états qu'il doit visiter entre deux états identiques est appelé la période du générateur.

Définition 2.1.2 [21] La *période* d'un générateur est le plus petit $\rho \in \mathbb{N}$ tel que $\exists\ \tau \in \mathbb{N}$, $\forall n > \tau$, $s_{\rho+n} = s_n$.

Une première qualité requise pour un générateur est une longue période. En effet, plus la période est courte, plus il est facile de se rendre compte que le générateur cycle et qu'il n'est en vérité pas du tout aléatoire.

2.2 Les générateurs à congruence linéaire

Le type de générateur le plus célèbre, ceci étant dû certainement à sa très grande simplicité, se nomme le générateur à congruence linéaire (GCL), [14, 16, 18]. Avant d'introduire la formulation exacte du générateur à congruence linéaire, rappelons quelques définitions.

Définition 2.2.1 [12, 42] Soit $a \in \mathbb{Z}$ et $m \in \mathbb{Z}^+$. L'opérateur mod retourne le reste de la division de a par m,

$$a \bmod m = a - m\left\lfloor \frac{a}{m} \right\rfloor.$$

Définition 2.2.2 [12, 42] Soit $m \in \mathbb{Z}^+$. Pour $a, b \in \mathbb{Z}$, on dit que *a est congruent à b modulo m*, noté $a \equiv b \bmod m$, si m divise $(a - b)$ ou, de façon équivalente, si $a = b + km$ pour un certain $k \in \mathbb{Z}$.

Le *générateur à congruence linéaire* (GCL) possède la structure suivante :

$$x_n = (ax_{n-1} + c) \bmod m, \tag{2.1}$$

$$u_n = x_n/m. \tag{2.2}$$

où $x_n \in \mathbb{N}$ est l'état à l'étape n,

$u_n \in [0, 1)$ est la sortie du générateur,

$m \in \mathbb{N}$, le module,

$a \in \mathbb{Z}$, le multiplicateur,

$c \in \mathbb{Z}$, la constante additive.

Parfois, nous ne voulons jamais avoir exactement $u_n = 0$ (2.2) pour des raisons techniques. On évite ce problème en remplaçant (2.2) par $u_n = (x_n + 1)/(m + 1)$.

La période maximale d'un GCL est m, le module, et elle est atteinte si les conditions suivantes sont remplies.

Théorème 2.2.1 [14] Le GCL possède une *période maximale* de longueur m si et seulement si

1. Le plus grand commum diviseur de c et m est 1,
2. $b = a - 1$ est un multiple de p, $\forall p$ divisant m,
3. b est un multiple de 4 si m est un multiple de 4.

Lorsque $c = 0$, nous obtenons un type particulier du GCL : il s'agit du *générateur à congruence linéaire multiples* (GCLM). Il possède une période maximale de $m - 1$. En effet, l'état 0 est absorbant, c'est-à-dire que, si l'état 0 est visité, le générateur ne peut plus aller vers d'autres états.

Les conditions de période maximale pour un GCLM sont données par le théorème suivant.

Théorème 2.2.2 [14, 16] Le GCLM possède une *période maximale* de longueur $m - 1$ si et seulement si,

1. m est premier,
2. $a \not\equiv 0 \bmod m$,
3. $a^{(m-1)/q} \not\equiv 1 \bmod m$ pour tout q facteur premier de $m - 1$.

2.3 Les générateurs récursifs multiples

Les *générateurs récursifs multiples* (GRM) constituent une généralisation des GCLM. Au lieu de considérer uniquement le dernier état visité, on utilise dans la récurrence linéaire les k états précédents.

La récurrence devient :

$$x_n = (a_1 x_{n-1} + a_2 x_{n-2} + \ldots + a_k x_{n-k}) \bmod m, \tag{2.3}$$

$$u_n = x_n / m. \tag{2.4}$$

où $(x_{n-k}, ..., x_{n-1}) \in \mathbb{Z}_m^k$ est l'état à l'étape n,

$u_n \in [0, 1)$ est la sortie du générateur,

a_i, $i = 1, ..., k$, sont des entiers dans $\{-(m-1), ..., m-1\}$ appelés multiplicateurs,

$m \in \mathbb{N}$ est le module,

k est l'ordre du générateur.

Il existe m^k vecteurs $(x_{n-k}, ..., x_{n-1})$ différents dans S car les x_i, $i = n-k, ..., n-1$, peuvent prendre chacun m valeurs différentes pour un total de m^k combinaisons. Comme pour les GCLM, l'état $(0, \ldots, 0)$ est un état absorbant que l'on doit éviter.

Pour trouver les conditions de période maximale du générateur, nous avons recours au polynôme caractéristique de la récurrence. Pour un GRM de la forme (2.3) et (2.4), le polynôme caractéristique est :

$$P(z) = z^k - a_1 z^{k-1} - \ldots - a_k. \tag{2.5}$$

Théorème 2.3.1 Soit m un nombre premier et $r = (m^k - 1)/(m - 1)$.

1. [38] Le générateur (2.3) et (2.4) possède une *période* de $m^k - 1$ si et seulement si le polynôme $P(z)$ (2.5) est primitif sur $\mathbb{F}_m$.
2. [14] Le polynôme $P(z)$ est *primitif* sur $\mathbb{F}_m$ si et seulement si les conditions suivantes sont remplies.
   (a) $((-1)^{k+1} a_k)^{(m-1)/q} \not\equiv 1 \bmod m$ pour tout facteur premier q de $m - 1$,

(b) $(z^r \bmod P(z)) \equiv (-1)^{k+1} a_k \bmod m$,

(c) $z^{r/q} \bmod P(z)$ est de dégré > 0 pour tout facteur premier q de r, $1 < q < r$.

2.4 Les générateurs combinés

De la même manière que les GRM sont une généralisation des GCLM, les générateurs combinés constituent une généralisation des GRM. L'idée est de faire évoluer quelques générateurs en parallèle, en général deux ou trois, et de combiner leurs sorties pour produire une sortie unique.

On peut combiner plusieurs types de générateurs, par exemple, des linéaires avec des non linéaires [21]. Nous nous limitons cependant à la combinaison de générateurs qui sont tous de type linéaire car cela rend l'analyse plus simple. Le type de combinaison que nous utilisons provient de [19].

Soit J GRM évoluant en parallèle :

$$\begin{aligned} x_{1,n} &= (a_{1,1}x_{1,n-1} + \cdots + a_{1,k_1}x_{1,n-k_1}) \bmod m_1, \\ x_{2,n} &= (a_{2,1}x_{2,n-1} + \cdots + a_{2,k_2}x_{2,n-k_2}) \bmod m_2, \\ &\vdots \\ x_{J,n} &= (a_{J,1}x_{J,n-1} + \cdots + a_{J,k_J}x_{J,n-k_J}) \bmod m_J. \end{aligned} \tag{2.6}$$

Nous supposons que les m_j, $j = 1, ..., J$, sont tous relativement premiers. Définissons :

$$z_n = \Big(\sum_{j=1}^{J} \delta_j x_{j,n} \Big) \bmod m_1; \qquad u_n = z_n/m_1, \tag{2.7}$$

où les $\delta_1, \ldots, \delta_J$ sont des entiers quelconques tels que δ_j est relativement premier à m_j, $j = 1, ..., J$.

Nous considérons aussi la combinaison suivante :

$$v_n = \Big(\sum_{j=1}^{J} \frac{\delta_j x_{j,n}}{m_j} \Big) \bmod 1. \tag{2.8}$$

Définissons le GRM suivant [19] :

$$x_n = (a_1 x_{n-1} + \cdots + a_k x_{n-k}) \bmod m, \tag{2.9}$$

$$w_n = x_n/m, \tag{2.10}$$

où $k = \max(k_1, ..., k_J)$, $m = m_1 \times m_2 \times ... \times m_J$ et
$a_i = \left(\sum_{j=1}^{J} a_{j,i} [(m/m_j)^{-1} \bmod m_j](m/m_j) \right) \bmod m$, $i = 1, ..., k$.

Le prochain théorème nous dit que la combinaison (2.8) est équivalente au GRM (2.9) et (2.10).

Théorème 2.4.1 [19] Si $(w_0, ..., w_{k-1}) = (v_0, ..., v_{k-1})$, alors $w_n = v_n$, $\forall n \geq 0$.

Théorème 2.4.2 [19] Soit $\rho_j, j = 1, \ldots, J$, la période de chacun des générateurs individuels de (2.6), alors *les périodes des générateurs combinés* (2.7) et (2.8) sont égales au plus petit commun multiple de $(\rho_1, \rho_2, \ldots, \rho_J)$, que nous dénotons par $PPCM(\rho_1, \rho_2, \ldots, \rho_J)$.

Théorème 2.4.3 [19] Soit,

$$\begin{aligned}
\Psi^+ &= \{j | 2 \leq j \leq J \text{ et } (m_j - m_1)\delta_j > 0)\}, \\
\Psi^- &= \{j | 2 \leq j \leq J \text{ et } (m_j - m_1)\delta_j < 0)\}, \\
\Delta^+ &= \sum_{j \in \Psi^+} \frac{(m_j - m_1)(m_j - 1)\delta_j}{m_1 m_j} + \sum_{j \in \Psi^-} \frac{(m_j - m_1)\delta_j}{m_1 m_j}, \\
\Delta^- &= \sum_{j \in \Psi^+} \frac{(m_j - m_1)\delta_j}{m_1 m_j} + \sum_{j \in \Psi^-} \frac{(m_j - m_1)(m_j - 1)\delta_j}{m_1 m_j}.
\end{aligned}$$

Si $(w_0, ..., w_{k-1}) = (u_0, ..., u_{k-1})$, alors $u_n = (w_n + \epsilon_n) \bmod 1$, où pour tout $n \geq 0$, nous avons $\Delta^- \leq \epsilon_n \leq \Delta^+$.

Le théorème précédent donne une borne sur la différence entre u_n et w_n. Si les m_j, $j = 1, ..., J$, sont de tailles relativement semblables alors les valeurs de Δ^+ et Δ^- sont très faibles et nous pouvons remplacer l'analyse de la récurrence (2.7) par l'analyse de (2.10). Ceci est utile lors de l'analyse théorique par le test spectral, car le type de combinaison que nous allons utiliser est (2.7) mais nous allons effectuer le test spectral sur le GRM (2.9) et (2.10), qui équivaut à la combinaison (2.8).

2.5 Les mesures de la qualité d'un générateur : le test spectral

Nous avons à ce point défini ce qu'est un générateur et quelques unes des propriétés qu'il doit posséder. Une longue période est un des critères importants. Depuis l'émergence des générateurs, des outils ont dû être développés afin d'évaluer les générateurs sous différents critères.

Le test théorique le plus utilisé en simulation pour tester des GCL et GRM est le test spectral [8, 10, 14, 19, 23, 44]. Il a été introduit en 1965 par Coveyou et MacPherson [6] et il sert de premier critère pour évaluer nos générateur.

Soit la définition suivante d'un réseau :

Définition 2.5.1 [5, 21] Un *réseau* de dimension t dans l'espace $\mathbb{R}^t$ est un espace vectoriel de la forme

$$L_t = \{V = \sum_{j=1}^{t} z_j V_j | z_1, z_2, ..., z_t \in \mathbb{Z}\}, \tag{2.11}$$

où les $V_1, ..., V_t$ sont des vecteurs linéairement indépendants dans $\mathbb{R}^t$ et constituent une base du réseau L_t.

Notation 2.5.1 [21, 25] Soit

$$T_t = \{(u_0, ..., u_{t-1}) | s_0 = (x_0, ..., x_{k-1}) \in \mathbb{Z}_m^k\}, \tag{2.12}$$

l'ensemble de tous les t-tuplets de valeurs successives produites par (2.3) et (2.4) à partir de tous les germes, s_0, possibles.

Théorème 2.5.1 [21, 25] L'ensemble T_t est l'intersection d'un réseau L_t avec l'hypercube $[0, 1)^t$ en dimension t.

Une technique pour construire une base du réseau est décrite dans [26].

Avant de poursuivre la définition théorique du test spectral, regardons un exemple simple.

Exemple : Considérons le générateur défini par :

$$x_n = (2^{15} - 2^{10}) x_{n-1} \bmod (2^{31} - 1); \quad u_n = x_n / (2^{31} - 1). \tag{2.13}$$

Examinons l'ensemble

$$T_2 = \{(u_0, u_1) | x_0 \in \mathbb{Z}_m\} \tag{2.14}$$

des couples produits par le générateur. Nous représentons une partie des points de l'ensemble T_2 sur le graphique suivant :

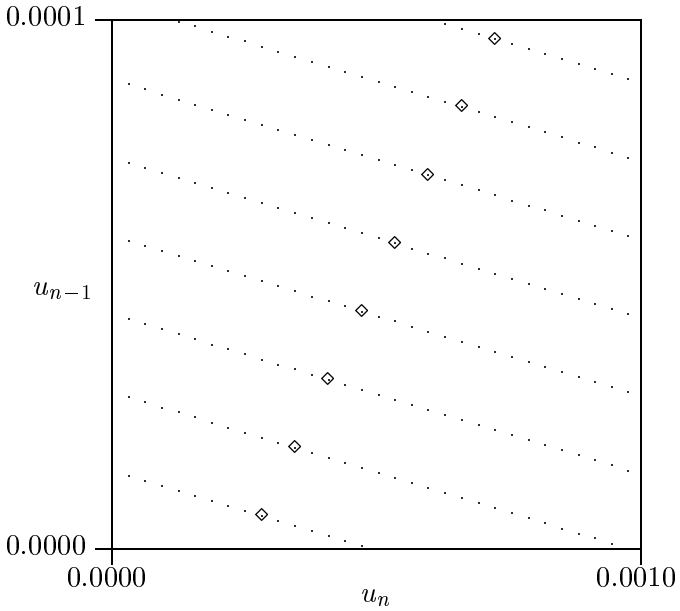

Figure 2.1: Structure de réseau pour $x_n = (2^{15} - 2^{10})x_{n-1} \bmod (2^{31} - 1)$ avec $t = 2$

Les points dans le graphique ne sont pas dispersés de manière aléatoire, ils forment une structure régulière. Les points produisent des ensembles de droites parallèles et équidistantes. Les plus évidentes sur le graphique précédent sont au nombre de huit et possèdent une pente négative. Il existe d'autres ensembles de droites parallèles et équidistantes, par exemple, celles allant dans le sens des $\diamond$ sur le graphique.

Tout dépendant de la récurrence définissant le générateur, les droites ne sont pas situées au même endroit, sont espacées différemment et comportent plus ou moins de points. Puisque les nombres produits par le générateur doivent se comporter comme des variables aléatoires i.i.d. $U(0, 1)$, nous voulons couvrir le plan le plus uniformément avec les points. En dimension t, les points (ou t-tuplets) ne forment plus des droites mais plutôt des hyperplans dans l'espace $[0, 1)^t$.

Le test spectral détermine la distance la plus grande entre deux hyperplans parallèles consécutifs parmi toutes les familles d'hyperplans qui couvrent complètement et uniquement l'ensemble des points de T_t. Nous désirons que cette distance, notée d_t, soit la plus petite possible pour que les points de T_t soient distribués le plus uniformément dans la boîte $[0,1)^t$. Le test spectral n'est utile que dans les dimensions plus grandes que l'ordre k d'un générateur linéaire car pour $t \leq k$, on a

$$T_t = \{(x_0/m, x_1/m, ..., x_{t-1}/m) | s_0 = (x_0, ..., x_{t-1}) \in \mathbb{Z}_m^k\} \tag{2.15}$$

peu importe la récurrence définissant le GRM d'ordre k, de sorte que $d_t = 1/m$.

Le calcul de d_t ne se fait pas en générant explicitement tous les points de T_t pour calculer d_t. Voyons comment cela est fait en pratique.

Définition 2.5.2 [5] Le *réseau dual* $L_t^\perp$ du réseau L_t est défini par :

$$L_t^\perp = \Big\{ W \in \mathbb{R}^t : W \cdot V \in \mathbb{Z} \quad \forall V \in L_t \Big\}, \tag{2.16}$$

où W et V sont des vecteurs et $W \cdot V$ représente le produit scalaire entre W et V.

Théorème 2.5.2 [14, 21] La valeur de d_t pour la récurrence (2.3) et (2.4) est égale à $1/\ell_t$, où ℓ_t est la *longueur euclidienne du plus court vecteur non nul du réseau dual* associé à (2.3) et (2.4).

Définition 2.5.3 [27] La *base duale* correspondant aux vecteurs de base $V_1, ..., V_t$ du réseau L_t est l'ensemble des $W_1, ..., W_t \in \mathbb{R}^t$ tels que :

$$V_i \cdot W_j = \delta_{ij}, \text{ où } \delta_{ij} = \begin{cases} 1 & \text{si } i = j, \\ 0 & \text{sinon.} \end{cases}$$

Ces vecteurs forment une base du réseau dual $L_t^\perp$. Étant donné une base quelconque $\{W_1, ..., W_t\}$, tout vecteur de $L_t^\perp$ peut s'écrire comme $W = \sum_{i=1}^t z_i W_i$, $z_i \in \mathbb{Z}$. Ainsi, déterminer un plus court vecteur non nul dans le réseau dual revient à résoudre le problème d'optimisation quadratique en nombres entiers suivant [20, 26], où les variables de décision sont $z_1, ..., z_t$.

$$\begin{aligned} \min \quad & ||W||^2 = \Big(\sum_{i=1}^{t} z_i W_i\Big) \cdot \Big(\sum_{j=1}^{t} z_j W_j\Big) = \sum_{i=1}^{t}\sum_{j=1}^{t} z_i W_i W_j z_j \\ \text{s.l.c} \quad & (z_1, ..., z_t) \neq 0, \\ & z_1, ..., z_t \in \mathbb{Z}. \end{aligned} \tag{2.17}$$

Le logiciel `LatMRG` [27], créé au laboratoire d'optimisation et de simulation du département d'informatique et de recherche opérationnelle de l'Université de Montréal, permet de trouver la longueur euclidienne d'un plus court vecteur par la résolution du problème d'optimisation.

La valeur de d_t [9] est normalisée par une borne inférieure d_t^* sur toutes les valeurs possibles de d_t pour un réseau ayant m^k points par unité de volume. Le rapport d_t^*/d_t se trouve toujours dans l'intervalle (0,1). Il est ainsi plus facile de déterminer si un générateur particulier possède une bonne uniformité en dimension t. La meilleure borne inférieure est :

$$d_t^* = \frac{1}{\gamma_t m^{k/t}}, \tag{2.18}$$

où γ_t est la constante de Hermite, dont la valeur exacte est connue pour $t \leq 8$ [14]. La borne d_t^* donne la distance entre les hyperplans lorsque les points sont distributés le plus uniformément dans l'hypercube $[0,1)^t$.

Pour $t > 8$, puisque γ_t n'est pas connue exactement, on se sert de bornes inférieures et supérieures sur γ_t. L'une des bornes supérieures est la borne de Rogers [5, 23], définie par :

$$\rho_t = 2e^{R(t)/t}. \tag{2.19}$$

Les valeurs de $R(t)$ peuvent être trouvées dans [5] pour $t \leq 24$. Au-delà de cette dimension, la valeur de $R(t)$ peut être approximée avec une assez bonne précision. On connaît également une borne inférieure sur γ_t donnée par

$$2\lambda_t^{-1/(2t)}, \tag{2.20}$$

où les valeurs de λ_t sont données dans [5]. On peut remplacer γ_t par l'une de ces bornes dans (2.18). Pour les recherches de générateurs que nous faisons, nous utilisons la borne de Rogers (2.19).

Une fois que nous avons d_t et d_t^*, nous posons

$$S_t = \frac{d_t^*}{d_t}. \tag{2.21}$$

Plus la valeur de S_t approche 1, plus le générateur testé affiche de bonnes caractéristiques d'uniformité dans cette dimension, selon ce critère.

Un bon générateur devra bien se comporter dans plus d'une dimension. Typiquement, nous observons les performances jusqu'en dimension T, où T se situe habituellement quelque part entre 5 et 50 [14, 22]. Nous prenons comme critère de sélection la pire valeur de S_t :

$$M_T = \min_{k \leq t \leq T} S_t. \tag{2.22}$$

2.5.1 Les indices lacunaires

Le test spectral, de la manière dont nous l'avons décrit jusqu'à maintenant, utilise seulement des vecteurs formés de sorties successives. Il peut aussi être intéressant de regarder ce qui se produit si nous prenons des sorties u_n qui ne sont pas consécutives. On peut regarder, par exemple, les vecteurs formés de deux coordonnées distancées d'un pas. C'est-à-dire l'ensemble des points formés de (u_0, u_2), lorsque $s_0 = \{x_0, x_1, ..., x_{k-1}\} \in \mathbb{Z}_m^k$. Il s'agit d'une façon plus générale et de plus en plus courante [20, 29] d'appliquer le test spectral.

De façon générale, posons $I = \{i_1, i_2, ..., i_t\}$, un ensemble de t valeurs entières. L'ensemble T_t, dénoté maintenant $T_t(I)$, est formé ainsi :

Notation 2.5.2 [20, 26] On note

$$T_t(I) = \big\{(u_{i_1}, u_{i_2}, ..., u_{i_t}) | s_0 = (x_0, ..., x_{k-1}) \in \mathbb{Z}_m^k\big\}, \tag{2.23}$$

l'ensemble de tous les *t-tuplets* espacés selon I et produits par (2.3) et (2.4), à partir de tous les germes, s_0, possibles.

Théorème 2.5.3 [26] L'ensemble $T_t(I)$ est l'intersection d'un réseau $L_t(I)$ avec l'hypercube $[0, 1)^t$ en dimension t.

Il est possible de trouver une base pour le réseau $L_t(I)$ [20]. Par la suite, déterminer $d_t(I)$, se fait de manière similaire au calcul de d_t.

2.5.2 Une autre borne sur d_t

Le vecteur de dimension t, $(a_1, a_2, ..., a_k, -1, 0, 0, ..., 0)$, fait toujours partie de $L_t^{\perp}$, puisque le produit scalaire entre ce vecteur et n'importe quel vecteur de L_t donne 0. Ainsi, le plus court vecteur dans le réseau dual doit nécessairement être de longueur inférieure ou égale à ce vecteur [21]. Donc,

$$\ell_t \leq \left(1 + \sum_{i=1}^{k} a_i^2\right)^{1/2} \tag{2.24}$$

et nous trouvons

$$d_t \geq \left(1 + \sum_{i=1}^{k} a_i^2\right)^{-1/2} \tag{2.25}$$

en remplaçant ℓ_t par $1/d_t$.

Nous remarquons que plus les a_i sont petits, plus la borne inférieure (2.25) sur d_t est élevée. Puisque nous désirons que d_t soit petit, il est nécessaire que les coefficients a_i de notre générateur soient de grands nombres. Cette borne n'est pas très serrée et dépend fortement des coefficients.

2.6 Les avantages et défauts des générateurs à congruence linéaire

Les générateurs de type GRM possèdent de nombreux avantages et également quelques inconvénients. Une énumération et une courte analyse des qualités et défauts de ce type de générateurs nous permettront de voir clairement ce qu'il faut éviter ou favoriser.

Débutons par leurs avantages. Simplement en regardant les quelques récurrences exposées dans ce chapitre, nous pouvons voir qu'elles sont extrêmement faciles à programmer. Elles peuvent tenir en quelques lignes de code dans un langage de haut niveau, par exemple, dans un langage qui contient l'opération mod et les opérations arithmétiques élémentaires. Il faut cependant faire attention : ces quelques lignes de code évidentes au premier abord ne sont peut-être pas toujours des plus efficaces du point de vue de la vitesse.

Leur très grande simplicité permet de les étudier plus facilement que d'autres générateurs. Il est possible d'analyser la structure des t-tuplets $(x_{i_1}, x_{i_2}, ..., x_{i_t})$. Ceux-ci forment comme nous l'avons dit des hyperplans, et cette structure se retrouve dans l'ensemble des générateurs à congruence linéaire. Par opposition, les générateurs non linéaires ne possèdent pas cette caractéristique et sont, en général, plus difficiles à analyser [21].

La linéarité des points peut cependant se retourner contre nous. Cette caractéristique ne permet pas d'utiliser sans danger l'ensemble de la période, à des fins de simulation par exemple. Au moins un test statistique échoue lorsque plus de la racine cubique de la période est utilisée [31]. Il serait peut-être même possible de construire des tests qui échoueraient avant la racine cubique de la période. Il est donc recommandé de se servir de beaucoup moins que de la racine cubique de la période.

Ce type de générateur n'offre pas une assez grande sécurité pour l'encryption de données en cryptographie. Par la construction d'algorithmes capables de prédire, à partir de quelques nombres, la suite de la séquence, il a été prouvé [2, 15] que les GRM ne sont pas «PT-perfect». La notion de «PT-perfect» est très importante en cryptographie même si pour aucun générateur, il n'a été possible, jusqu'à maintenant, de prouver qu'un générateur était «PT-perfect». Pour une définition de «PT-perfect», se referer à [18]. Nous devons être conscients de ce problème chez les GRM sans pour autant rejeter la méthode, car ce n'est pas le but à atteindre dans le cas présent.

2.7 La méthodologie des tests statistiques

Le test spectral décrit à la section précédente est un test théorique qui classe, selon un critère précis, les générateurs de nombres pseudo-aléatoires. Une des qualités du test spectral est qu'il regarde l'ensemble de toutes les sorties possibles du générateur. Ainsi, si le test réussit ou échoue, la totalité du générateur est fautive et non pas seulement le sous-ensemble de sorties dont nous nous serions servi. Aucun élément aléatoire n'entre en jeu lorsque nous appliquons le test spectral.

Cependant, lorsque le test spectral calcule M_T, il regarde les doublets, les triplets,... des valeurs de sorties prises en un tout. Aucune indication n'est présente sur l'ordre de l'apparition d'un doublet, par exemple. Ainsi, il peut exister des corrélations entre les doublets ou les triplets successifs et le test spectral ne les détecte pas. Ceci ne fait pas du test spectral un mauvais test mais nous incite à pousser l'analyse des générateurs par des tests statistiques.

Cette section introduit le lecteur aux tests statistiques de manière générale et aussi en rapport avec les générateurs de nombres aléatoires. Nous discutons brièvement de l'origine des tests statistiques dans l'univers des générateurs de nombres pseudo-aléatoires. Nous expliquons par la suite ce qu'est un test d'hypothèse et comment nous les appliquons aux séquences de nombres produites par les générateurs.

2.7.1 Les tests statistiques et les générateurs

L'évaluation des générateurs de nombres pseudo-aléatoires par des tests statistiques trouve son origine au début de la construction de générateurs de nombres pseudo-aléatoires. Un des premiers ensemble de tests proposé le fut par Knuth [14]. Cet ensemble de tests, énormément utilisé par le passé, l'est encore de nos jours. Toutefois, tout comme les générateurs évoluent avec les années, les tests statistiques le font également. Les chercheurs, avec le temps, ont construit des tests qu'ils considèrent plus discriminants que les tests précédents. Marsaglia, avec son module DIEHARD [35], propose un ensemble de tests qu'il considère très robuste. La librairie ≪testu01≫ [24] développée par L'Ecuyer regroupe les tests statistiques les plus connus et utilisés.

2.7.2 Qu'est-ce qu'un test statistique ?

Le but d'un test d'hypothèse est de mettre à l'épreuve une hypothèse formulée. La première étape d'un test d'hypothèse consiste à fixer l'hypothèse nulle : $\mathcal{H}_0$. En général, dans un test d'hypothèse nous retrouvons aussi une hypothèse alternative. Toutefois, l'hypothèse alternative n'est pas précisée explicitement lors de tests d'hypothèse appliqués aux générateurs de nombres aléatoires. Le test d'hypothèse évalue la force de l'évidence contre l'hypothèse nulle.

Un test statistique est défini par une variable aléatoire T. Pour calculer T, nous utilisons les données pour lesquelles nous voulons déterminer si elles sont en accord avec l'hypothèse nulle. Nous devons avoir une idée juste, basée sur de la théorie, de la distribution F de la variable aléatoire T, sous l'hypothèse $\mathcal{H}_0$, afin de construire le test d'hypothèse correctement.

2.7.3 La p-valeur

La p-valeur est la probabilité, calculée en supposant que $\mathcal{H}_0$ est vraie, que la variable aléatoire T prenne une valeur aussi extrême ou plus extrême que la valeur observée t_1. En termes mathématiques, la p-valeur, à droite, se définit comme :

$$p_1 = P[T \geq t_1 | \mathcal{H}_0]. \tag{2.26}$$

Si F est une distribution continue, alors $p_1 = 1 - F(t_1)$ et p_1 est une variable aléatoire de loi uniforme sur $(0, 1)$ sous l'hypothèse $\mathcal{H}_0$. Il est habituel de nos jours de déterminer non seulement si une hypothèse est rejetée ou non, mais avec quel niveau de signification. En effet, comme le souligne [34], «significance probabilities (or p-values), with the additionnal information they provide, are typically more appropriate than fixed levels in scientific problems ...». Lorsque nous utilisons la p-valeur, il n'est pas nécessaire de déterminer à l'avance un seuil de rejet α.

2.7.4 L'application des tests statistiques aux générateurs

Les tests statistiques tentent de détecter des irrégularités ou des régularités dans la séquence des nombres générés. Le but des tests statistiques est de mettre à l'épreuve les générateurs d'un point de vue pratique. L'hypothèse nulle $\mathcal{H}_0$ à tester est toujours : «les valeurs produites par le générateur peuvent être vues comme des variables aléatoires indépendantes suivant la loi $U(0, 1)$». L'hypothèse $\mathcal{H}_0$ découle directement de l'objectif d'imiter des variables aléatoires i.i.d $U(0, 1)$. Au chapitre 7, nous effectuons cinq tests statistiques différents pour $\mathcal{H}_0$. Pour chacun de ces tests, nous calculons la valeur d'une variable aléatoire pour laquelle nous connaissons la distribution sous $\mathcal{H}_0$.

Chapitre 3

Les méthodes existantes de calcul rapide pour un GCL

La vitesse de production des nombres a toujours été un critère important à considérer lors de la création d'un générateur. Il ne s'agit pas uniquement de la manière dont est optimisé le programme lorsque les paramètres du générateur sont connus. Les techniques utilisées jusqu'à ce jour, et celles proposées dans ce chapitre, ajoutent des contraintes sur les coefficients des générateurs. Nous devons en tenir compte avant de débuter toute recherche de générateurs.

Dans un GCLM, il y a une multiplication et une opération modulo. Dans un GRM d'ordre k, nous aurons k multiplications et à nouveau une opération modulo. La tâche consiste à accélérer la vitesse de l'ensemble de ces opérations.

Dans un ordinateur conventionnel, les opérations se font en binaire. La multiplication peut être vue de la manière suivante, qui n'est pas nécessairement la manière dont elle est faite dans l'ordinateur. La multiplication par 2^p, $p \in \mathbb{N}$, est équivalente à un décalage de p bits vers la gauche. Suite au décalage, des zéros sont automatiquement placés dans les p bits les moins significatifs. Si le coefficient est une somme de λ puissances de 2, alors l'opération de décalage est faite un nombre correspondant de fois et les résultats sont additionnés. De même, si le module est de la forme $2^e, e \in \mathbb{N}$, l'opération modulo se réduit à conserver les e derniers bits du nombre subissant l'opération.

Ces techniques ont déjà été utilisées pour implanter des générateurs ayant des modules et des coefficients formés de quelques puissances de 2.

Un exemple célèbre est le générateur RANDU de IBM [13], défini par :

$$x_n = 65539 x_{n-1} \bmod 2^{31}; \quad u_n = x_n / 2^{31}. \tag{3.1}$$

Son multiplicateur, 65539, s'écrit aussi $2^{16} + 2^1 + 2^0$. La multiplication peut être remplacée

par deux décalages et deux additions. L'opération mod 2^{32}, sur un ordinateur à 32 bits revient à mettre le bit le plus significatif à 0. Chacune des opérations de remplacement est très peu coûteuse lorsqu'elle est considérée individuellement, ceci par comparaison avec les opérations remplacées.

Cependant, comme plusieurs auteurs le soulignèrent par la suite [14, 41], ce générateur ne réussit pas à passer de simples tests statistiques, par exemple, le test sériel fait dans [14]. Sa période maximale est seulement de $m/4$, à la place du $m-1$ auquel nous nous attendons de la part d'un GCLM de module m. Knuth [14] le qualifie de ≪complètement horrible≫ . Law et Kelton [16] précisent que RANDU ne devrait jamais être utilisé.

L'idée de RANDU n'en est pas restée là. Nous avons assisté à l'émergence de générateurs frères de RANDU et non moins épouvantables. Citons seulement le GCLM de Gottfried [11] : il fonctionne sur un ordinateur 16 bits avec $a = 2^8 + 3$ et $m = 2^{15}$. Ce générateur est malheureusement proposé sans aucune mise en garde.

Les opérations, lorsque les coefficients ou le module sont des puissances de 2, deviennent pratiquement trop simples. Également, un générateur avec $m = 2^e, e \in \mathbb{N}$ n'aura jamais une période maximale à moins qu'il y ait une constante c ajoutée au GCLM pour retrouver un GCL, théorème 2.2.1. Au mieux, elle atteindra $m/4$, si et seulement si $a \bmod 8 = 3$ ou 5, et où le germe initial est un entier impair. Il existe d'autres problèmes reliés à ce type de module [17]. Par exemple, pour un GCLM avec $m = 2^e, e \in \mathbb{N}$ et une période maximale de $m/4$, le générateur produira une permutation périodique de la moitié des entiers impairs entre 1 et $2^e - 1$.

Depuis plusieurs années, l'idée d'utiliser $m = 2^e, e \in \mathbb{N}$ comme module a été pratiquement abandonnée, à tout le moins par la communauté scientifique consciente des problèmes reliés à ce type de module. Pour respecter les conditions de période maximale, on prend un nombre premier, la plupart du temps près d'une puissance de 2, de la forme $2^e - h$, où h est petit, afin d'avoir une période la plus grande possible pour le nombre de bits utilisés.

En ce qui concerne les coefficients, l'idée de prendre une somme ou une différence de deux puissances de 2 continue d'être intéressante. Nous croyons que nous pouvons ainsi chercher

de bons générateurs sans négliger l'aspect vitesse. Cette idée a été réintroduite par Wu [44]. Il utilise des multiplicateurs qui sont des sommes ou des différences de puissances de 2 et le module est un nombre premier de la forme $2^e - 1$, c'est-à-dire un nombre de Mersenne. Les générateurs à congruence linéaire proposés sont très rapides.

Nous faisons un survol des méthodes existantes qui permettent d'améliorer la vitesse de calcul des générateurs. Nous exposons trois méthodes connues et faciles d'implantation. Une de ces méthodes, developpée par Wu [44], intéressante à plusieurs niveaux, présente cependant certaines lacunes qui ont été décelées par L'Ecuyer et Simard [30]. Nous terminons par une brève explication de la démarche employée par L'Ecuyer et Simard.

3.1 La factorisation approximative

Cette méthode provient de [1, 4, 21, 41]. Il s'agit d'une manière rapide d'effectuer le calcul de $ax \bmod m$ avec $0 < a, x < m$ et m premier. Cette méthode travaille avec des nombres entiers. Elle consiste à factoriser approximativement m, le module, d'où son nom. Nous retraçons ici la méthode en ajoutant les étapes intermédiaires qui ne sont pas toujours présentes dans la documentation consultée.

Nous calculons $x \leftarrow ax \bmod m$. Ce calcul est équivalent à

$$x \leftarrow ax - km, \qquad \text{où } k = \left\lfloor \frac{ax}{m} \right\rfloor = \left\lfloor \frac{ax}{aq + r} \right\rfloor. \tag{3.2}$$

Pour éviter de calculer x via (3.2), nous introduisons la variable Z calculée de la manière suivante :

$$\begin{aligned} Z &= ax - ym \text{ où } y = \left\lfloor \frac{x}{q} \right\rfloor, && (3.3)\\ &= ax - y(aq + r), && (3.4)\\ &= a(x - yq) - yr. && (3.5) \end{aligned}$$

En introduisant Z dans (3.2), nous trouvons

$$x = ax - km = ax - ym + ym - km = Z + (y - k)m. \tag{3.6}$$

Nous allons prouver que la différence $y - k$ est toujours égale à 0 ou 1. Le calcul de x en est ainsi beaucoup facilité pour quelques raisons. Les valeurs de q et r sont précalculées et les résultats des produits intermédiaires demeurent inférieurs à m. Nous nous assurons de cette dernière condition en posant des contraintes sur a et m.

• Montrons que $0 \leq a(x - yq) < m$. Nous avons que

$$0 \leq a\Big(x - \left\lfloor \frac{x}{\lfloor m/a \rfloor} \right\rfloor \left\lfloor \frac{m}{a} \right\rfloor\Big) = a\Big(x \bmod \left\lfloor \frac{m}{a} \right\rfloor\Big) < a\left\lfloor \frac{m}{a} \right\rfloor \leq \frac{am}{a} = m. \tag{3.7}$$

La condition est toujours remplie.

• Preuve que $0 \leq yr < m$. Ceci est équivalent à

$$0 \leq \left\lfloor \frac{x}{\lfloor m/a \rfloor} \right\rfloor r \leq \frac{xr}{\lfloor m/a \rfloor} < \frac{x\lfloor m/a \rfloor}{\lfloor m/a \rfloor} = x < m. \tag{3.8}$$

Ce raisonnement est vrai si $r < \lfloor m/a \rfloor$. Ce qui revient à

$$m \bmod a < \left\lfloor \frac{m}{a} \right\rfloor \leq \frac{m}{a}. \tag{3.9}$$

Notre condition est donc

$$a(m \bmod a) < m. \tag{3.10}$$

Nous venons de montrer via (3.5), (3.8) et (3.9), que $-m < Z < m$ si (3.10) est satisfaite. Puisque x se situe nécessairement entre 0 et m, il suffit d'ajouter m à Z, si $Z < 0$.

ALGORITHME GÉNÉRAL DU CALCUL
AVEC LA FACTORISATION APPROXIMATIVE

$y := \lfloor x/q \rfloor$
$x := a(x - yq) - yp$
if $x < 0$ **then** $x := x + m$

Figure 3.1: Pseudo-code pour le calcul avec la factorisation approximative

3.2 Le calcul en point flottant

Une autre technique consiste à représenter les nombres en point flottant [22]. Ceci fonctionne à condition que chacun des produits $a_i x_{n-i}$ et les sommes intermédiaires $\sum_{i=1}^{\ell} a_i x_{n-i}$, $1 \leq \ell \leq k$, soient inférieurs au plus grand entier qui est représenté exactement en point flottant. Une condition suffisante est que la mantisse des nombres représentés contienne au moins

$$\lceil \log_2((m-1)\sum_{i=1}^{k} a_i) \rceil$$

bits de précision. Ainsi, les calculs peuvent se faire directement en point flottant sans perte de précision. Avec une architecture efficace pour le calcul en point flottant, telle celle des machines Sun, les calculs sont rapides.

La technique de calcul en point flottant débute par l'évaluation explicite du produit

$$\sum_{i=1}^{k} a_i x_{n-i}.$$

Ensuite, l'opération mod m est effectuée. L'opération $k := x/m$ permet de trouver la valeur entière de la division de x par m. Un langage comme C, par exemple, peut exécuter cette instruction sans problème. La ligne $x := x - km$ donne le reste de la division entière de x par m, c'est-à-dire x mod m. La dernière ligne de l'algorithme, qui contient un **if**, est là pour nous assurer que le résultat est positif.

ALGORITHME GÉNÉRAL DU
CALCUL EN POINT FLOTTANT

x : réel ; k : entier
$x := \sum_{i=1}^{k} a_i x_{n-i}$
$k := x/m$
$x := x - km$
if $x < 0$ **then** $x := x + m$

Figure 3.2: Pseudo-code pour le calcul en point flottant

3.3 La méthode de Wu

Wu [44] propose une technique pour calculer $2^p x \bmod (2^e - 1)$ où $p < e$ et $e, p \in \mathbb{N}$. Nous débutons en illustrant la méthode par un schéma, puis nous commentons chacune des étapes pour montrer que tout fonctionne correctement.

Un schéma explicatif en trois étapes de la méthode de Wu

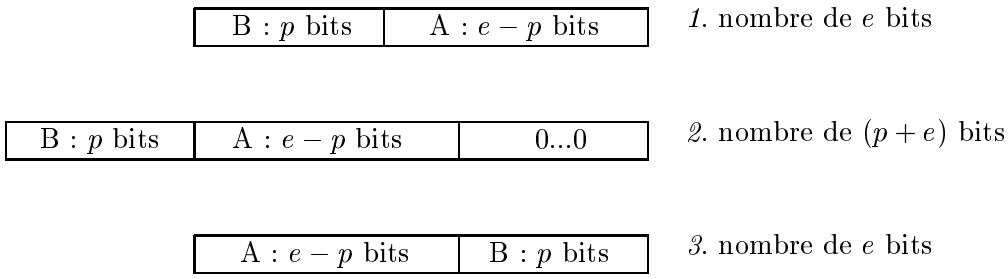

Figure 3.3: Schéma illustrant la méthode de Wu

Avant toute transformation, le nombre x est composé de e bits dont les p premiers (B) sont les plus significatifs et les $e - p$ autres (A) les moins significatifs.

La deuxième ligne correspond au nombre $2^p x$. Nous retrouvons ainsi p zéros ajoutés à la fin du nombre et les p bits les plus significatifs du départ (B) sortent maintenant de la représentation sur e bits.

L'opération à effectuer est :

$$2^p x \bmod (2^e - 1). \tag{3.11}$$

Celle-ci est équivalente à :

$$\left(2^p x - (2^e - 1)\left\lfloor \frac{2^p x}{2^e} \right\rfloor\right) \bmod (2^e - 1). \tag{3.12}$$

Remarquons que le nombre

$$2^p x - 2^e \left\lfloor \frac{2^p x}{2^e} \right\rfloor + \left\lfloor \frac{2^p x}{2^e} \right\rfloor \tag{3.13}$$

correspond à la ligne 1 où les blocs A et B sont interchangés. La représentation binaire du nombre de la ligne 1 ne contient pas seulement que des 1 car le nombre x est inférieur à $2^e - 1$. Ainsi, le nombre de la ligne 3 est également inférieur à $2^e - 1$ et nous pouvons laisser tomber la dernière opération mod dans (3.12). En d'autres mots, pour calculer $2^p x \bmod (2^e - 1)$, nous inversons simplement les blocs A et B de la ligne 1 pour retrouver la ligne 3.

3.4 Les premiers générateurs proposés utilisant la méthode de Wu

Suivant la méthode qu'il a proposée, Wu [44] suggère quelques GCLM rapides. Voici la liste de ceux-ci :

$m = 2^{31} - 1$		$m = 2^{61} - 1$	
$a = 2^{15} - 2^{10}$	$a = -2^{16} - 2^{11}$	$a = 2^{42} - 2^{31}$	$a = 2^{30} - 2^{19}$
$M_8 = 0{,}63942$	$M_8 = 0{,}57386$	$M_8 = 0{,}37807$	$M_8 = 0{,}36527$

Tableau 3.1: Test spectral pour les générateurs de Wu provenant de [44]

Il présente deux générateurs avec un module sur 32 bits, $m = 2^{31} - 1$, et deux autres avec un module pour une architecture 64 bits, $m = 2^{61} - 1$. Chacun des coefficients respecte les conditions imposées. La valeur M_8, définie en (2.22), est le résultat du test spectral jusqu'en dimension 8. Les deux générateurs ayant comme module $2^{31} - 1$ donnent un bon résultat au test spectral. En effet, une valeur autour de 0,6 constitue un bon résultat au test spectral. Toutefois, sans avoir besoin de procéder à d'autres tests, les générateurs avec $m = 2^{61} - 1$ sont plus faibles par rapport au critère M_8. Il est préférable de ne pas utiliser les générateurs du tableau 3.1 car ils possèdent tous une assez courte période.

3.5 Quelques lacunes des générateurs de Wu relevées par L'Ecuyer et Simard

Des tests statistiques effectués par L'Ecuyer et Simard dans [30] ont fait ressortir quelques problèmes reliés aux générateurs de Wu. Nous résumons ici la démarche qu'ils ont suivie. Ils testent l'indépendance entre les poids de Hamming des bits les plus significatifs dans la représentation de u_n et u_{n-1}. Ils définissent Y_n comme étant le nombre de «1» parmi les ℓ premiers bits de la représentation de u_n où ℓ est une constante.

Comme d'habitude, nous testons l'hypothèse nulle $\mathcal{H}_0$: «les u_n sont des variables aléatoires indépendantes et identiquement distribuées (i.i.d.) $U(0,1)$.» Sous $\mathcal{H}_0$, les Y_n sont des variables aléatoires i.i.d. qui suivent une loi binomiale de paramètre $(\ell, 1/2)$[37], c'est-à-dire

$$P(Y_n = k) = \binom{\ell}{k} \frac{1}{2^k} \frac{1}{2^{\ell-k}}. \tag{3.14}$$

Soit N un entier positif et $C_{i,j}$ le nombre de valeurs pour lesquelles $(Y_{2n-1}, Y_{2n}) = (i,j)$, $\quad 1 \leq n \leq N$ et $0 \leq i,j \leq \ell$. Sous l'hypothèse $\mathcal{H}_0$, $C_{i,j}$ suit une loi binomiale de paramètres $(N, p_{i,j})$ où $p_{i,j}$ est la probabilité d'avoir $(Y_{2n-1}, Y_{2n}) = (i,j)$. Sous l'hypothèse que les variables aléatoires Y_{2n-1} et Y_{2n} sont indépendantes, ceci est équivalent à

$$\begin{aligned} p_{i,j} = P[Y_{2n-1} = i] P[Y_{2n} = j] &= \left[\binom{\ell}{i} \frac{1}{2^i} \frac{1}{2^{\ell-i}}\right] \left[\binom{\ell}{j} \frac{1}{2^j} \frac{1}{2^{\ell-j}}\right], \\ &= \frac{1}{2^{2\ell}} \binom{\ell}{i} \binom{\ell}{j}. \end{aligned} \tag{3.15}$$

Lorsque N est très grand, nous pouvons approximer cette loi binomiale par une normale ayant les mêmes moyenne et variance, soit $Np_{i,j}$ et $Np_{i,j}(1-p_{i,j})$. Ceci est justifié par le théorème de la limite centrale [43] lorsque $N \to \infty$.

La variable standardisée

$$Z_{i,j} = \frac{C_{i,j} - Np_{i,j}}{\sqrt{Np_{i,j}(1-p_{i,j})}} \tag{3.16}$$

suit alors approximativement une loi normale de moyenne 0 et de variance 1. L'Ecuyer et Simard rejettent $\mathcal{H}_0$ si $|Z_{i,j}| \geq 5$ pour plusieurs $Z_{i,j}$.

Sous la forme d'un graphique en deux dimensions, nous présentons les valeurs de $Z_{i,j}$ pour $i = 0, ..., 30,\ j = 0, ..., 30$. Un carré représente une paire $Z_{i,j}$.

Légende

$Z_{i,j} \leq -5$ $Z_{i,j} = -2$ $Z_{i,j} = 0$ $Z_{i,j} = 2$ $Z_{i,j} \geq 5$

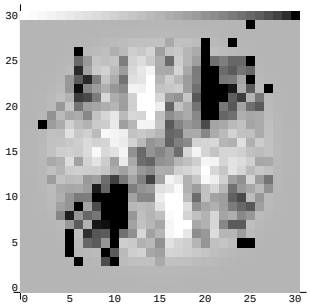

Figure 3.4: Image des $Z_{i,j}$ pour $a = 2^{15} - 2^{10}, \ell = 30, N = 2^{20}$

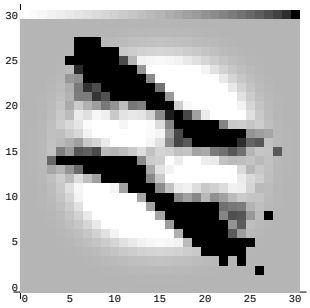

Figure 3.5: Image des $Z_{i,j}$ pour $a = -2^{16} - 2^{11}, \ell = 30, N = 2^{20}$

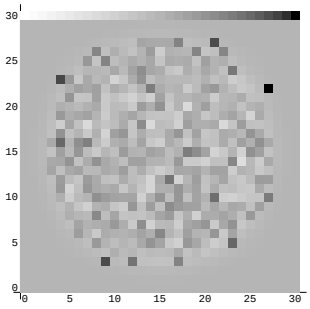

Figure 3.6: Image des $Z_{i,j}$ pour $a = 16807, \ell = 30, N = 2^{20}$

Les deux premières images, tirées de [30], montrent l'application du test pour les générateurs de Wu provenant de [44]. Les régions noires et blanches indiquent que nous devons rejeter l'hypothèse $\mathcal{H}_0$. Nous pouvons clairement voir que les valeurs u_n et u_{n-1} sont dépendantes. Par comparaison, les auteurs ont effectué le même test sur le GCLM défini par $x_n = 16807x_{n-1} \bmod (2^{31} - 1)$. Nous remarquons que la dépendance n'est plus visible.

L'Ecuyer et Simard [30] ont ensuite effectué un test plus formel en utilisant la distribution du chi-carré. Soit $\Psi = \{(i,j) : Np_{i,j} \geq 5\}$, la statistique chi-carré utilisée est

$$Q = \sum_{(i,j)\in\Psi} \frac{(C_{i,j} - Np_{i,j})^2}{Np_{i,j}}. \tag{3.17}$$

Sous l'hypothèse $\mathcal{H}_0$, Q possède approximativement une distribution chi-carré avec $|\Psi|$ degrés de liberté. Ils calculent la p-valeur du test, définie comme $p = P[Q > x|\mathcal{H}_0]$ où x est la valeur prise par un Q particulier. L'hypothèse $\mathcal{H}_0$ est rejetée lorsque p est vraiment trop près de 0. Les générateurs de Wu échouèrent aussi ce test alors que le GCLM avec $a = 16807$ le réussit bien pour les tests effectués.

3.6 Les récents résultats de Wu

Dans un travail plus récent, Wu [45] suggère sept GRM d'ordre 8 avec $m = 2^{31} - 1$. Voici les résultats obtenus au test spectral (2.22) jusqu'en dimension 16 pour les générateurs proposés dans [45], tableau 3.2.

Les générateurs G1 à G5 ne donnent pas de bons résultats par rapport au critère M_{16}. Le générateur G6 donne un résultat acceptable, toutefois le nombre de puissances de 2 parmi l'ensemble de ses coefficients est très élevé. Enfin, le générateur G7 donne un bon résultat mais ses coefficients ne sont pas de la forme $\pm 2^{p_1} \pm 2^{p_2}$.

Ainsi, pour l'instant, aucun GRM avec des coefficients de la forme $\pm 2^{p_1} \pm 2^{p_2}$ et ayant de bons résultats au test spectral n'a été proposé.

Générateur	$a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$	M_{16}
G1	$a_1 = 0, a_2 = 0, a_3 = 0, a_4 = 0$ $a_5 = -1, a_6 = 0, a_7 = 0, a_8 = 2^{16} + 2^8$	1,961 $\times 10^{-4}$
G2	$a_1 = 0, a_2 = 0, a_3 = 0, a_4 = 0$ $a_5 = -2^{11}, a_6 = 0, a_7 = 0, a_8 = 2^{23} + 2^{18}$	3,633 $\times 10^{-3}$
G3	$a_1 = 0, a_2 = 0, a_3 = 0, a_4 = 0$ $a_5 = -2^{21} + 2^{18}, a_6 = 0, a_7 = 0, a_8 = -2^{24} - 2^{10}$	4,637 $\times 10^{-3}$
G4	$a_1 = 0, a_2 = 2^{10}, a_3 = 0, a_4 = 0$ $a_5 = -2^{21}, a_6 = 0, a_7 = 0, a_8 = -2^{30} - 2^{11}$	8,123 $\times 10^{-3}$
G5	$a_1 = 2^{24}, a_2 = -2^{30}, a_3 = -2^{16}, a_4 = -2^{10}$ $a_5 = -2^{19}, a_6 = -2^6, a_7 = -2^{22}, a_8 = 2^{25} + 2^{12}$	0,1300
G6	$a_1 = 2^{24} - 2^{16}, a_2 = 2^{28} + 2^{12}, a_3 = 2^{22} + 2^{21}, a_4 = 2^{29} + 2^{12}$ $a_5 = 2^{28} - 2^{21}, a_6 = 2^{15} - 2^2, a_7 = 2^9 + 1, a_8 = -2^{20} - 2^6$	0,2751
G7	$a_1 = -280941024, a_2 = -809883983, a_3 = -1972918929$ $a_4 = -485411907, a_5 = -73766740, a_6 = -194153496$ $a_7 = -540134392, a_8 = -663034048$	0,5736

Tableau 3.2: Test spectral pour les générateurs de Wu provenant de [45]

Chapitre 4

Les générateurs de Deng et Lin

Dans un article paru en mai 2000, Deng et Lin [7] proposent des générateurs qu'ils considèrent rapides et fiables. Cet article a suscité un vif intérêt chez certains membres de la communauté scientifique qui voyaient une manière plus efficace de générer des nombres aléatoires. Mais qu'en est-il vraiment ?

L'argument principal des auteurs de l'article [7] est qu'il faut remplacer l'utilisation des GCLM par des GRM sans trop augmenter la vitesse de calcul. Nous abondons dans leur sens au niveau de la vitesse, qui est un facteur important. Les générateurs qu'ils suggèrent sont d'ordre 2 à 4 et ils suivent tous une récurrence de la forme

$$x_n = (-x_{n-1} + a x_{n-k}) \bmod m, \tag{4.1}$$

où $a < \sqrt{m}$. La multiplication de a par x_{n-k}, suivie de l'opération mod m, est faite en utilisant la factorisation approximative (section 3.1). Puis x_{n-1} est soustrait et la valeur m est ajoutée une fois si la valeur de x_n est inférieure à 0. La période de leur générateur est plus grande que celle d'un GCLM. Aucun test théorique, comme le test spectral, ou des tests statistiques ne viennent appuyer leurs propos quant aux qualités des générateurs proposés. Pour chacun des générateurs d'ordre 2 proposé dans l'article [7], nous effectuons le test spectral, expliqué au chapitre 2. Les résultats sont présentés dans le tableau 4.1.

4.1 Les résultats au test spectral pour les générateurs de Deng et Lin

La valeur a représente le coefficient du générateur testé et S_3 (2.21) donne le résultat du test spectral en dimension $t = 3$.

a	S_3	a	S_3	a	S_3
26403	0,01413	36181	0,01937	42174	0,02257
27149	0,01453	36673	0,01963	42457	0,02272
29812	0,01596	36848	0,01972	43199	0,02312
30229	0,01618	37097	0,01986	43693	0,02339
31332	0,01677	37877	0,02027	44314	0,02372
33236	0,01779	39613	0,02120	44530	0,02383
33986	0,01819	40851	0,02186	45670	0,02444
34601	0,01852	40961	0,02192	46338	0,02480
36098	0,01932				

Tableau 4.1: Test spectral des générateurs d'ordre 2 de Deng et Lin [7]

Pour $k = 2$, la valeur de S_3 est très faible. Nous montrons à la prochaine section que sans même effectuer le test spectral pour chacun des a, nous aurions pu prévoir que le comportement d'un générateur de la forme (4.1), avec $a < \sqrt{m}$, est mauvais en dimension 3.

4.2 Le calcul des bornes sur S_t

D'après la théorie sur le test spectral expliquée au chapitre 2, nous pouvons calculer S_t ainsi pour $t \leq 8$:

$$S_t = \frac{d_t^*}{d_t}, \qquad \text{où} \qquad d_t^* = \frac{1}{\gamma_t m^{k/t}}.$$

Dans le cas présent, $t = 3$, $m = 2^{31} - 1$, $k = 2$, la constante de Hermite γ_t est $2^{1/6}$ [14] et donc

$$d_3^* \approx 5,352 \times 10^{-7}. \tag{4.2}$$

Les auteurs stipulent que la valeur de a doit toujours être inférieure à $\sqrt{m}$ pour que la factorisation approximative soit possible. Ce qui n'est pas exact car la condition pour la factorisation approximative est plutôt donnée par (3.10). À l'aide de l'équation (2.25), avec $a < \sqrt{m}$, nous déduisons une borne sur la valeur de d_3,

$$d_3 \geq \left(1 + (1 + (2^{31} - 1))\right)^{-1/2} \quad \Rightarrow \quad \frac{1}{d_3} \leq (1 + 2^{31})^{1/2} \approx 46340,95.$$

Ainsi,

$$S_3 = \frac{d_3^*}{d_3} \leq 5,352 \times 10^{-7} \times 46340,95 \approx 0,0248.$$

Au tableau 4.1, cette borne est atteinte approximativement pour $a = 46338$. Peu importe la valeur choisie pour a, avec $a < \sqrt{m}$, tous les générateurs du type (4.1) se comportent très mal, en dimension 3, par rapport au test spectral.

Ce même raisonnement peut être fait mais en utilisant les indices lacunaires décrits à la section 2.5.1. Nous regardons les projections des points en dimension 3. L'ensemble I est $\{0, 1, k\}$. La borne sur d_t demeure la même, mais la valeur de d_t^* peut changer. En effet, la valeur de t est maintenant de 3 peu importe l'ordre du générateur. Pour les générateurs d'ordre 2, la borne sur S_3 demeure la même que nous regardions les indices lacunaires ou non.

Nous effectuons le même raisonnement avec $k = 3$ et $k = 4$ pour ne pas tester chacun des générateurs individuellement. Voici, sous la forme d'un tableau, les calculs et les valeurs des bornes trouvées.

$k = 3$	$k = 4$
$t = 4$	$t = 5$
$\gamma_4 = 2^{1/4}$	$\gamma_5 = 2^{3/10}$
$m = 2^{31} - 1$	$m = 2^{31} - 1$
$d_4^* \approx 8,429 \times 10^{-8}$	$d_5^* \approx 2,781 \times 10^{-8}$
$1/d_4 \leq 46340,95$	$1/d_5 \leq 46340,95$
$S_4 \leq 3,906 \times 10^{-3}$	$S_5 \leq 1,289 \times 10^{-3}$
$S_3(I) \leq 1,9224 \times 10^{-5}$	$S_3(I) \leq 1,4901 \times 10^{-8}$

Tableau 4.2: Calcul de la borne sur d_t pour les générateur d'ordre 3 et 4 de Deng et Lin [7]

Pour $t = k+1$, les bornes sur S_4 et S_5 ainsi que celles sur $S_3(I)$, pour $I = \{0, 1, k\}$, indiquent que les générateurs ont tous de mauvais résultats au test spectral. Il n'est pas nécessaire de regarder les autres dimensions, puisqu'une seule mauvaise dimension suffit à discréditer un générateur.

De plus, nous remarquons que la méthode (4.1) produit des générateurs dont le comportement est de moins en moins bon si k augmente. En effet, la borne sur d_t, avec $t = k + 1$, demeure inchangée peu importe la valeur de k car il n'y a que deux coefficients non nuls. Toutefois la valeur de d_t^* diminue avec t car la constante de Hermite augmente avec t, tout comme le rapport k/t.

4.3 Les générateurs matriciels de Deng et Lin

Dans le même article [7], Deng et Lin proposent des générateurs matriciels. Il s'agit de ℓ générateurs du type (4.1) qui évoluent en parallèle pour produire un vecteur aléatoire. Les valeurs produites par les générateurs matriciels doivent être des vecteurs aléatoires i.i.d $U(0, 1)$. Étant donné que les ℓ composantes sont du type étudié précédemment et éprouvent de sérieux problèmes vis-à-vis du test spectral, les générateurs matriciels présentés dans [7] ne sont pas recommendables.

Nous discutons à nouveau des générateurs de Deng et Lin au chapitre 7, sur les tests statistiques, et au chapitre 8, sur la vitesse des générateurs. Même si nous savons déjà que les générateurs du type (4.1) possèdent des défauts importants, nous voulons savoir s'ils peuvent tout de même être assez rapides afin de déterminer si cette méthode peut mener à des avenues intéressantes.

Chapitre 5

La méthodologie de la recherche

Dans ce chapitre, nous commençons par présenter les caractéristiques que doivent posséder les générateurs à construire. Nous posons les conditions à rencontrer et nous donnons le pseudo-code général d'un générateur. Nous poursuivons avec la démarche suivie lors de la construction des générateurs. Nous discutons des étapes successives qui nous permettent d'avoir une bonne confiance dans les générateurs que nous proposons.

5.1 Des générateurs efficaces

5.1.1 Les conditions

Dans la méthode de Wu [44], les coefficients permis sont de la forme $\pm 2^{p_1} \pm 2^{p_2}$, $\pm 2^{p_1}$ et 0. Pour les générateurs que nous voulons construire, nous ne voulons pas nous limiter à ces coefficients. Nos coefficients doivent toujours être une somme ou une différence de quelques puissances de 2 mais le nombre de puissances n'est pas limité à deux. Nous croyons, entre autres, que les nombres de la forme $\pm 2^{p_1} \pm 2^{p_2} \pm 1$ ne nécessitent pas beaucoup plus de temps de calcul et peuvent s'avérer intéressants pour améliorer l'uniformité.

Également dans la méthode de Wu [44], le module doit être de la forme $2^e - 1$, $e \in \mathbb{N}$. Ce type de module est plutôt limitatif car $2^e - 1$ doit être un nombre premier pour remplir les conditions de période maximale (théorème 2.3.1). Pour $8 < e < 64$, il y a seulement cinq nombres premiers de la forme $2^e - 1$.

L'Ecuyer et Simard [30] proposent plutôt de prendre $m = 2^e - h$ et imposent les conditions :

$$h < 2^p \text{ et } h(2^p - (h+1)2^{-e+p}) < m \tag{5.1}$$

La variable p ici, et dans la suite de cette section, est un cas général pour p_1 et p_2.

En respectant (5.1), le calcul peut se faire de la manière suivante [30]. Posons

$$x_{n-1} = A + 2^{e-p}B \quad \text{où } A = x_{n-1} \bmod 2^{e-p} \text{ et } B = \left\lfloor \frac{x_{n-1}}{2^{e-p}} \right\rfloor.$$

La variable e est la longueur en bits des nombres avec lesquels nous travaillons, en général $e = 32$ ou 64. Ainsi, B contient les p bits les plus significatifs et A contient les $e - p$ bits les moins significatifs.

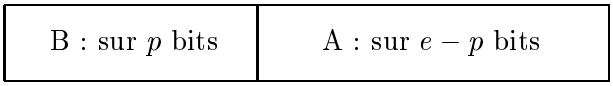

Figure 5.1: Schéma explicatif de la séparation de x_{n-1}

Le calcul de x_n se fait maintenant de la manière suivante :

$$\begin{aligned} x_n &= 2^p x_{n-1} \bmod m, \\ &= 2^p(A + 2^{e-p}B) \bmod (2^e - h), \\ &= ((2^p A \bmod (2^e - h)) + (2^e B \bmod (2^e - h))) \bmod (2^e - h), \\ &= (2^p A + Bh) \bmod (2^e - h). \end{aligned} \tag{5.2}$$

Grâce à la condition (5.1), nous savons que $2^p A + Bh$ n'excède jamais $2m$, ainsi la dernière opération mod peut se faire, en pire cas, à l'aide d'une soustraction. Tout ce calcul est effectué sans avoir recours à la multiplication (sauf pour Bh) et à la division. Seul l'emploi de décalages, d'additions et de soustractions est requis, des opérations peu coûteuses individuellement.

Les conditions (5.1) fixent une borne inférieure et une borne supérieure sur la valeur de p. Ces bornes sont données par

$$\log_2 h < p < \log_2 \left(\frac{2^e m}{h(m+1)} \right). \tag{5.3}$$

Lorsque nous faisons les recherches, nous testons p dans l'ensemble :

$$\left\{ \lceil \log_2 h \rceil, ..., \lfloor \log_2 \left(2^e m/(h(m+1)) \right) \rfloor \right\}. \tag{5.4}$$

Qu'advient-il si $m = 2^e + h$? Les conditions restent-elles les mêmes ? La méthode demeure-t-elle avantageuse ?

L'argument principal favorisant l'ajout des modules de la forme $2^e + h$ est qu'il ouvre la porte à un ensemble de nouveaux m. Toutefois, les modules de la forme $2^e + h$ possèdent un désavantage important. Pour illustrer ce désavantage, nous utilisons un exemple. Soit un ordinateur fonctionnant sur 32 bits dont le premier bit est un bit de signe. Si $m = 2^{31} - 1$, un nombre premier, alors tous les nombres entre 0 et $2^{31} - 2$ sont représentés exactement et nous nous servons du maximum des 32 bits. La période peut aller jusqu'à $2^{31} - 2$. Cependant nous ne pouvons pas prendre $m = 2^{31} + h$, $h \geq 1$, car certains nombres entre 0 et $2^{31} + h - 1$ ne sont pas représentés exactement. Pour éviter ce problème, nous prenons un nombre premier de la forme $2^{30} + h, h \geq 0$. La période n'est que de $2^{30} + h - 1$. La période la plus longue pour $2^{30} + h$ est pratiquement deux fois moins longue que celle de $2^{31} - 1$, alors que nous disposons du même nombre de bits. Ce facteur est non négligeable pour un GCLM. Toutefois, pour des GRM combinés, la période peut devenir tellement grande que le facteur deux devient négligeable dans ce cas.

Nous dérivons les nouvelles conditions sur h en reprenant le calcul fait à l'équation (5.2). Nous débutons les calculs à la troisième ligne, car c'est à partir de ce moment que le développement peut changer.

$$\begin{aligned} x_n &= ((2^p A \bmod (2^e + h)) + (2^e B \bmod (2^e + h))) \bmod (2^e + h), \\ &= (2^p A - Bh) \bmod (2^e + h). \end{aligned} \tag{5.5}$$

Pour passer de la première ligne à la suivante, il faut que $2^p A < m$. Puisque $A \leq 2^{e-p} - 1$, nous aurons $2^p A \leq 2^p(2^{e-p} - 1) = 2^e - 2^p < 2^e + h = m$. Nous avons toujours $2^p A < m$, aucune condition supplémentaire n'est nécessaire.

Pour effectuer le passage entre les deux lignes de l'équation (5.5), il faut regarder ce qui se passe avec $2^e B \bmod (2^e + h)$. On peut voir que ceci est égal à $-Bh + 2^e + h$ si $-Bh > -m$. Ceci revient à $Bh < m$, il s'agit de la même condition que dans le cas où $m = 2^e - h$. Nous laissons tomber la partie $+2^e + h$, car il reste une opération mod à effectuer et, si jamais $2^p A - Bh < 0$, alors tout redevient positif suite à l'opération mod. Il faut seulement nous

assurer que chacun des termes est borné, ce que a été fait.

Par rapport aux critères de L'Ecuyer et Simard (5.1), nous laissons tomber la première partie. La nouvelle condition devient :

$$h(2^p - (h+1)2^{-e+p}) < m. \tag{5.6}$$

L'exposant p peut se situer dans l'ensemble :

$$\left\{0, ..., \lfloor \log_2 \left(2^e m/(h(m+1))\right) \rfloor \right\}.$$

Nous croyons aussi qu'il peut être intéressant d'explorer le fait que h soit une somme ou une différence de quelques puissances de 2. Ainsi, nous évitons la multiplication de h par B. Il faut déterminer le nombre maximal de puissances de 2 pouvant composer h pour que le remplacement soit efficace. De plus, nous sommes limités par le fait que nous voulons un petit h et un m premier afin de maximiser la période. Toutes ces contraintes ne laissent pas grand choix parmi l'ensemble des m disponibles et peut-être devrons-nous laisser tomber l'idée de choisir un h qui s'exprime simplement avec des puissances de 2.

5.1.2 Notre générateur

Nous travaillons sur deux points pour améliorer le comportement du générateur. Premièrement, le type de module du générateur est généralisé. Nous ne nous limitons plus aux modules de la forme $2^e - 1$. Nous acceptons maintenant les modules de la forme $2^e - h$ ou $2^e + h$, avec h petit. Les calculs sont légèrement plus complexes avec ce type de module, mais il existe toujours une méthode pour éviter la division.

Deuxièmement, prendre des générateurs d'ordre supérieur à 1 ainsi que la combinaison de plusieurs générateurs nous laisse supposer que le côté «aléatoire» n'en sera qu'amélioré. Nous devons toujours garder en tête que nous misons sur l'amélioration de la vitesse et que, si l'ajout de plusieurs coefficients et de générateurs a certainement un effet bénéfique sur l'uniformité de la séquence, il ralentit la production des nombres.

Notre *but est la construction de générateurs du type* :

$$x_n = a_1 x_{n-1} + a_2 x_{n-2} + \ldots a_k x_{n-k} \bmod m$$

où a_i, $i = 1, \ldots, k$, est de la forme $\pm 2^{p_i} \pm 2^{q_i}$ ou $\pm 2^{q_i}$ ou 0, m est de la forme $2^e - h$, $h \geq 1$, $k \geq 1$.

Nous voulons aussi faire une recherche du côté des générateurs combinés ; dans ce cas-là, les composantes de notre générateur sont aussi de la forme exposée ci-haut. Chacun des modules est un nombre premier pour nous assurer une période maximale. Nous favorisons les coefficients de la forme 2^{p_1} ou ceux de la forme $2^{p_1} \pm 2$ et $2^{p_1} \pm 1$ puisque le calcul se fait plus rapidement avec la partie ± 2 et ± 1 que s'il s'agissait d'une puissance de 2 quelconque.

5.1.3 Le pseudo-code

Nous présentons dans cette section un pseudo-code permettant de mieux visualiser nos programmes. Chaque langage possède ses optimisations propres et, lorsque viendra le temps de parler des implantations, nous ciblerons un langage et ses spécificités.

Les valeurs de p_1, p_2, M, m et h de la figure 5.2 sont des constantes définies à l'extérieur de la procédure. La constante M représente la valeur maximale que nous pouvons représenter avec le nombre de bits dont nous disposons et elle sert comme masque pour mettre à zéro le bit de signe. Les constantes p_1 et p_2 donnent respectivement le premier exposant et le deuxième exposant composant notre coefficient de la forme $\pm 2^{p_1} \pm 2^{p_2}$. La constante m indique le module de notre générateur et h est défini selon la manière habituelle.

L'état du GCLM est gardé en mémoire entre les appels de procédures dans la variable X_n. Cette variable est globale. Nous pourrions la passer en paramètre, (c'est en général ce qui est recommandé afin de produire un programme plus «propre») mais l'ajout de la variable X_n dans l'appel de procédure augmente le temps de l'appel. Les variables z_1 et z_2 définies dans le corps de la procédure servent à conserver les résultats intermédiaires.

La commande $z_1 := ((X_n >> (31 - p_1))h$ prend les $31 - (31 - p_1) = p_1$ bits les plus significatifs et les multiplie par h. Ceci correspond à l'opération Bh dans l'équation (5.2). La ligne suivante est $z_1 := z_1 + ((X_n << p_1)\ \&\ M)$. L'opération $(X_n << p_1)$ pousse les bits de X_n de p_1 positions vers la gauche et ajoute des zéros aux p_1 positions de droite qui sont maintenant vides. L'opération $\&M$ permet de mettre à zéro le bit de signe. Enfin, nous

additionnons le premier z_1 et $((X_n << p_1)\ \&\ M)$ pour retrouver exactement l'équation (5.2). Le **if** permet de nous ramener à une valeur entre 0 et $m-1$ si jamais z_1 excède $m-1$.

Par la suite, les mêmes commandes sont reproduites en utilisant l'exposant p_2 à la place de p_1. Nous nous retrouvons avec les valeurs $z_1 = 2^{p_1} X_n \bmod m$ et $z_2 = 2^{p_2} X_n \bmod m$. Puisque nous désirons $(2^{p_1} + 2^{p_2}) X_n \bmod m$, il faut additionner z_1 et z_2 et effectuer un dernier $-m$ si nécessaire. La valeur X_n/m est retournée.

Dans ce programme, nous supposons que $a = 2^{p_1} + 2^{p_2}$. Si jamais nous avions une puissance de 2 précédée d'un moins, alors nous utiliserions l'équation suivante pour nous retrouver dans une situation connue (si $0 < a < m$) :

$$(-a) \bmod m = m - a. \tag{5.7}$$

Le programme général pour un GCLM

```
p1, p2, M, m : constante ;
Xn : entier ;
Xn = 12345 ;

procedure LCG () : reel
var z1, z2 : entier ;

begin
   z1 := ((Xn >> (31 − p1)) × h ;
   z1 := z1 + ((Xn << p1) & M) ;
   if z1 ≥ m then z1 = z1 − m

   z2 := ((Xn >> (31 − p2)) × h ;
   z2 := z2 + ((Xn << p2) & M) ;
   if z2 ≥ m then z2 = z2 − m

   if (z1 + z2 < m) then Xn := z1 + z2 ;
   else Xn := z1 + z2 − m ;
   return Xn/m ;
end ;
```

Figure 5.2: Pseudo-code pour un GCLM

5.2 La méthodologie

5.2.1 Le problème

Notre objectif principal est de maximiser la vitesse des générateurs. C'est à ce niveau que nos générateurs se démarqueront des autres. Il existe plusieurs contraintes affectant la construction des générateurs.

Tout d'abord les coefficients d'un GRM d'ordre k doivent être de la forme

$$\pm 2^{p_1} \pm 2^{p_2} \text{ ou } \pm 2^{p_1} \text{ ou } 0, \tag{5.8}$$

avec p_1 et p_2 dans l'intervalle (5.4). Le module m doit être un nombre premier près d'une puissance de 2, nous devons avoir

$$m = 2^e - h \text{ ou } m = 2^e + h, e \in \mathbb{N}. \tag{5.9}$$

Les deux conditions (5.8) et (5.9) produisent un sous-ensemble de l'ensemble de tous les GRM d'ordre k. Nous dénotons cet ensemble par $P_{2,k}$, pour *puissance de 2*.

Comme nous l'avons précisé au chapitre 2, les GRM doivent obtenir un résultat satisfaisant au test spectral pour être utilisés. Nous désignons par T_{sp} l'ensemble de tous les GRM qui obtiennent un un bon résultat au *test spectral*.

La réussite des tests statistiques représente également un critère important. Les générateurs proposés doivent les passer sans problème pour renforcer la confiance que nous avons envers eux. Nous appelons *ei* l'*ensemble* des tests statistiques d'*intérêt* auquel nous soumettons les générateurs. Nous dénotons pas ST_{ei} l'ensemble de tous les générateurs qui réussissent les *tests statistiques ei*.

En terme d'optimisation, notre problème de construction d'un générateur G peut s'énoncer ainsi :

$$\begin{aligned} \max \quad & \text{la vitesse de } G \\ \text{sujet à} \quad & G \in \left(P_{2,k} \cap T_{sp} \cap ST_{ei}\right). \end{aligned} \tag{5.10}$$

En fait, nous ne résolvons pas exactement ce problème (trouver un maximum global). Nous ne faisons que trouver une bonne solution par une combinaison d'ingéniosité et de recherche aléatoire. À la sous-section suivante, nous décrivons les étapes nécessaires pour résoudre approximativement ce problème.

5.2.2 La démarche

Puisque les ensembles $P_{2,k}$, T_{sp} et ST_{ei} contiennent un très grand nombre de générateurs, il est impossible d'énumérer l'intersection des ensembles. L'ensemble $P_{2,k}$ est différent des deux autres en ce sens où nous savons au premier abord, lors de la construction, si un générateur précis en fait partie.

La technique dont nous nous servons consiste premièrement à construire aléatoirement des générateurs provenant de $P_{2,k}$. Pour ce faire, nous fixons un module m respectant la condition (5.9). Les modules m sont choisis de façon à ce que m soit premier et que la décomposition en facteurs premiers de $(m^k - 1)/(m - 1)$ soit connue. Cette décomposition est nécessaire pour calculer la période maximale de notre générateur provenant du théorème 2.3.1. La décomposition constitue un calcul très ardu et n'est connue que pour certains m et k.

Nous générons au hasard les coefficients pour qu'ils respectent la condition (5.8). Une fois en possession d'un générateur de la classe $P_{2,k}$, nous lui faisons passer le test spectral. Si le test est réussi avec succès, c'est-à-dire une valeur de M_T (2.22) supérieure à 0,3 dans le cas d'un MRG, alors nous conservons ce générateur.

Comme nous ne pouvons pas être assurés de trouver des générateurs dans la classe $P_{2,k} \cap T_{sp}$, nous augmentons nos chances en permettant aux coefficients de pouvoir contenir jusqu'à trois puissances de 2. La variable λ indique le nombre maximal de puissances de 2 que peut contenir un coefficient a. Lorsque nous avons pu trouver des générateurs $G \in (P_{2,k} \cap T_{sp})$ avec $\lambda = 3$, nous fixons $\lambda = 2$ et enfin nous tentons de fixer certains coefficients à zéro. Notre but lors de cette étape est la construction de générateurs qui allient de bons résultats au test spectral et un nombre de puissances de 2 minimal. Cette première étape est répétée jusqu'à ce que nous ayons une quantité suffisamment grande de générateurs.

Entre le test spectral et les tests statistiques, se trouve une étape intermédiaire. Nous laissons de côté une partie des générateurs retenus lors de la première étape. Cette étape est nécessaire car les tests statistiques sont coûteux en temps de calcul et il est impossible de tester un trop grand éventail de générateurs. Nous retenons les générateurs qui à vue d'oeil peuvent maximiser la vitesse. Cette étape est faite de manière heuristique.

La deuxième véritable étape est composée des tests statistiques. Nous avons maintenant en notre possession un ensemble assez restreint de générateurs auxquels nous ajoutons des générateurs existants. Nous ajoutons un générateur qui utilise la méthode de décomposition approximative [19] et un autre basé sur la méthode en point flottant [23]. Ceci est fait dans le but de s'en servir comme point de comparaison. Les deux générateurs ajoutés obtiennent de bons résultats au test spectral. Nous appliquons également les tests statistiques sur un des générateurs de Wu [44] ainsi que sur un des générateurs de Deng et Lin [7]. Ces deux derniers générateurs n'ont jamais été soumis à des tests statistiques, sauf pour le test d'indépendance des bits pour le générateur de Wu [30], et nous voulons déterminer comment ils se comporteront.

La dernière étape consiste à tester la vitesse de tous les générateurs retenus et à comparer cette vitesse avec des générateurs existants et réputés rapides [19, 23]. Si nous n'arrivons pas à battre les générateurs existants, alors cette nouvelle technique de décomposition en puissance de 2 n'apporte rien de nouveau. Cependant, si nous réussissons alors nous avons des générateurs aussi bons théoriquement et empiriquement que les autres générateurs mais plus rapides, un avantage non négligeable.

Nous avons représenté de manière schématique nos étapes de recherche de générateurs à la figure 5.3.

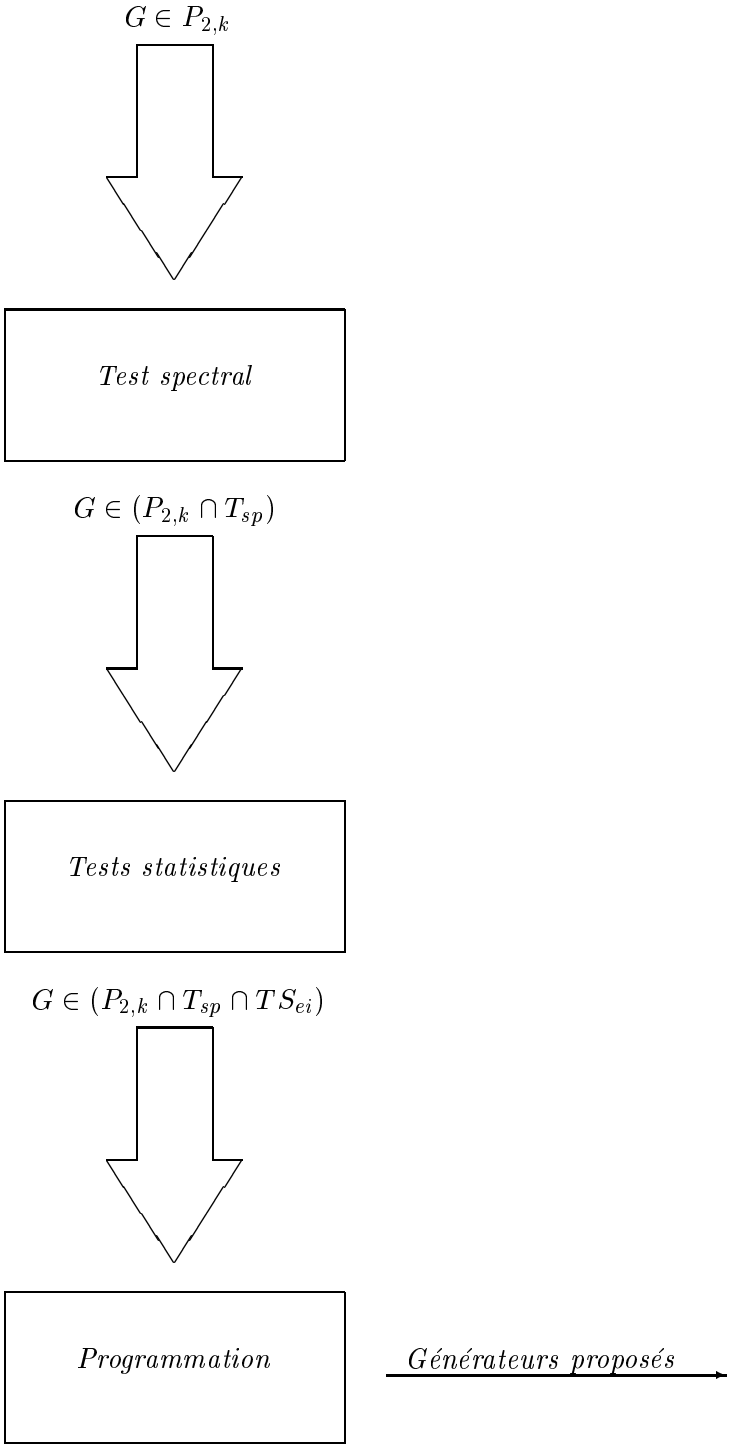

Figure 5.3: Schéma illustrant la démarche

Chapitre 6

Le choix des paramètres

Nous commençons par faire une recherche exhaustive sur des GCLM. Nous savons qu'il existe des problèmes avec les GCLM ayant un a de la forme $\pm 2^{p_1} \pm 2^{p_2}$. Cette recherche n'est pas là pour fournir des générateurs utilisables, mais bien pour observer le comportement de petits générateurs par rapport au test spectral, lorsque nous devons respecter la condition (5.4).

À partir des GCLM, nous élaborons une équation permettant de trouver le nombre de coefficients disponibles de la forme $\pm 2^{p_1} \pm 2^{p_2}$ ou 2^{p_1} et qui répondent à la condition (5.4). Nous expliquons ce que nous apporte une telle équation. Nous ne nous attardons pas longtemps à l'étude et la recherche de GCLM puisque, comme nous l'avons déjà précisé, les GCLM ne sont pas des générateurs acceptables de nos jours. Leur période est clairement trop courte et lorsqu'ils possèdent des coefficients du type $\pm 2^{p_1} \pm 2^{p_2}$, ils échouent de simples tests statistiques [30].

Par la suite, nous nous tournons du côté de recherches qui regardent au hasard une partie seulement de tous les générateurs disponibles : il s'agit des recherches aléatoires. Avec les recherches aléatoires, nous trouvons des GRM d'ordre 5 à 8. Un générateur d'ordre 5 avec $m \approx 2^{31}$ donne approximativement une période de 2^{155}, ce qui est suffisant pour les simulations contemporaines. Nous commençons par trouver des générateurs qui obtiennent un bon résultat au test spectral en permettant à tous les coefficients d'avoir jusqu'à deux ou même trois puissances de 2. Nous nous assurons ainsi d'avoir des générateurs qui respectent la condition (5.4). Lorsque cela est acquis, nous diminuons tranquillement le nombre de puissances de 2 permises pour déterminer une borne inférieure sur le nombre minimal de puissances tout en conservant un résultat acceptable au test spectral.

Le cheminement fait avec les GRM non combinés est repris avec des GRM combinés, sauf que l'ordre de nos générateurs individuels utilisés dans les combinaisons est plutôt de 3 ou 4. Enfin nous comparons les GRM combinés et non combinés quant à la borne inférieure sur le nombre minimal de puissances de 2.

6.1 Les méthodes de recherche

Nous utilisons deux types de méthodes de recherche, soit exhaustive ou aléatoire. Une recherche exhaustive signifie que nous testons l'ensemble des coefficients qui répondent à un critère précis. Ce type de recherche est en général long. Le temps de recherche sera raisonnable si nos critères sont assez restrictifs. Nous nous rendrons compte, plus loin dans ce chapitre, que des recherches exhaustives sont trop longues pour des GRM d'ordre supérieur à 2, c'est-à-dire pour des GRM ayant un certain potentiel.

Une recherche aléatoire implique que nous regardons uniquement une portion des coefficients. Nous fixons au préalable le nombre approximatif de générateurs à regarder. Un des désavantages de la recherche aléatoire est que nous pouvons passer à côté d'un bon générateur et perdre notre temps à tester deux fois le même, sans le savoir.

6.2 La recherche exhaustive

Pour des GCLM, nous effectuons une recherche exhaustive des coefficients. Nous faisons préalablement une analyse théorique du nombre de coefficients disponibles avant de nous aventurer dans des recherches trop intensives. Cette analyse nous montre que pour des GCLM avec un module inférieur à 2^{64}, le nombre de coefficients n'est pas excessivement grand et qu'il est possible de tous les tester.

6.2.1 L'analyse théorique du nombre de coefficients

L'analyse théorique que nous faisons ici est liée à l'analyse que nous faisons par la suite de l'algorithme qui génère les coefficients au hasard, ainsi qu'à l'algorithme qui génère exhaustivement tous les coefficients. Pour l'instant, nous désirons faire le décompte du nombre de coefficients pour savoir si une recherche exhaustive est possible. Par la suite, nous trouvons une méthode pour énumérer tous les coefficients. Ces deux étapes vont de pair de façon évidente. Comprendre comment nous arrivons à énumérer l'ensemble des coefficients qui sont de la forme $\pm 2^{p_1} \pm 2^{p_2}$ ou 2^{p_1}, avec p_1 et p_2 bornés, est important, pour ne pas tester deux fois le même générateur. Cette analyse est faite dans ce chapitre, puisqu'elle comporte un lien avec la recherche exhaustive. Elle nous sert également au chapitre 9, lorsque nous déterminons si notre algorithme, qui génère nos coefficients au hasard pour la recherche aléatoire, est efficace. Il existe souvent deux ou trois façons différentes de produire un même nombre avec deux puissances de 2. Ainsi, il existe une probabilité non nulle de tester deux fois le même générateur, ceci rend évidemment moins efficace notre méthode de recherche aléatoire. Ce que nous déterminons est la proportion théorique du temps que nous pouvons passer à tester deux ou trois fois le même générateur. Pour connaître cette proportion, il faut analyser théoriquement la structure des nombres de la forme $\pm 2^{p_1} \pm 2^{p_2}$ ou 2^{p_1}, et savoir en quelle proportion ils sont présents sur un intervalle précis.

Nous effectuons notre analyse en déterminant, tout d'abord, combien il y a de nombres qui peuvent être construits lorsque nous posons les valeurs des bornes inférieures et supérieures sur p_1 et p_2 (5.4).

Soit les valeurs I_{p_1}, S_{p_1}, I_{p_2}, et S_{p_2}, posées respectivement pour la borne inférieure sur p_1, la borne supérieure sur p_1, la borne inférieure sur p_2 et la borne supérieure sur p_2. Sur le graphique suivant, nous représentons les différents ordonnancements pour les bornes inférieures et supérieures.

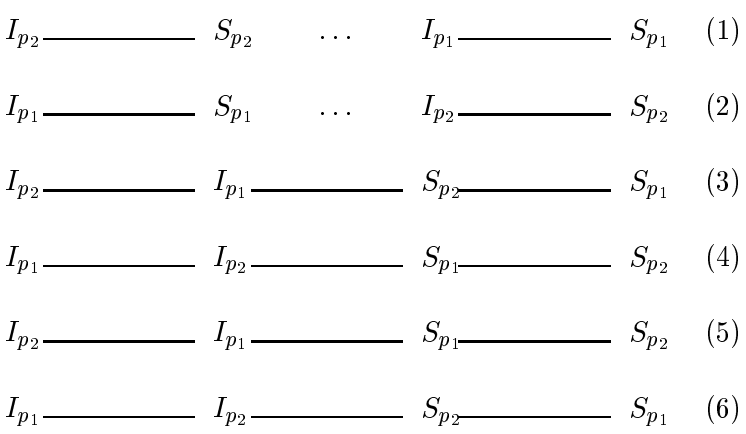

Figure 6.1: Illustration des positions possibles pour les bornes sur les exposants

Une ligne horizontale pleine indique que tous les nombres compris entre les limites, par exemple de I_{p_1} à S_{p_1} pour la ligne 1, peuvent être utilisés comme exposants. Des pointillés indiquent qu'il peut y avoir des nombres non inclus, si par exemple $S_{p_2} < I_{p_1}$. Toutes les possibilités y sont représentées.

Ces énumérations de nombres peuvent être groupées deux à deux pour les analyser plus facilement puisque dans les calculs que nous effectuons, il n'y a pas de différence entre p_1 et p_2. Les cas 1 et 2 sont identiques, de même 3 et 5, et 4 et 6.

Pour les recherches exhaustives, nous faisons l'hypothèse que $I_{p_1} = I_{p_2}$ et $S_{p_1} = S_{p_2}$ et nous les dénotons par I et S simplement. Puisqu'il s'agit d'une recherche exhaustive, nous ne voulons en aucun cas limiter la plage de recherche admissible. Ce cas constitue une simplification des cas 3 et 4. Dans le cas d'une recherche aléatoire, toutes les possibilités sont intéressantes.

Afin de bien comprendre comment les nombres sont formés, sans permettre de répétitions, nous débutons par un exemple. Soit $I = 3$ et $S = 7$. Nous pouvons jouer avec les signes assignés à chacune des puissances de 2. Mais afin de faire une énumération systématique, nous alignons les nombres en ordre croissant en ne nous souciant que des nombres positifs. Les mêmes nombres négatifs peuvent être formés en inversant les signes. À la fin, nous doublons simplement la quantité de coefficients.

Le plus petit nombre que nous pouvons former est 2^3, la puissance associée au plus petit exposant disponible. Le prochain nombre est 2^4 et suivent $2^4+2^3, 2^5, 2^5+2^3, 2^5+2^4, 2^6-2^3, 2^6, 2^6+2^3, 2^6+2^4, 2^6+2^5, 2^7-2^4, 2^7-2^3, 2^7, 2^7+2^3, 2^7+2^4, 2^7+2^5, 2^7+2^6,\ 2^7+2^7$. Nous avons 19 nombres différents et nous pouvons facilement nous rendre compte qu'aucun nombre n'a été oublié. Une preuve formelle est faite à ce sujet pour le cas général.

Nous remarquons une certaine structure dans la manière dont les nombres sont construits. Présentés de la façon suivante, ceci devient encore plus évident.

2^3	2^4	2^5	2^6	2^7
	2^4+2^3	2^5+2^3	2^6+2^3	2^7+2^3
		2^5+2^4	2^6+2^4	2^7+2^4
		2^6-2^3	2^6+2^5	2^7+2^5
			2^7-2^4	2^7+2^6
			2^7-2^3	2^7+2^7
1	2	4	6	6

Tableau 6.1: Énumération des $a_i = 2^{p_1} \pm 2^{p_2}$ avec $3 \leq p_2 \leq p_1 \leq 7$

Nous plaçons les nombres en ordre croissant et nous changeons de colonne à chaque fois que nous rencontrons une nouvelle puissance de 2 simple. Les valeurs encadrées représentent le nombre de valeurs dans la colonne associée. Pour les colonnes 2, 3 et 4, le nombre de coefficients présents est donné par la formule suivante :

$$f_k = 2(k-I), \text{ où } k \text{ est la valeur de la puissance étudiée.}$$

Théorème 6.2.1 Soit $K = \{I+1, ..., S-1\}$, un sous-ensemble des valeurs valides des exposants des puissances de 2. Dans l'intervalle $[2^k, 2^{k+1}), k \in K$, les nombres que nous pouvons exprimer en additionnant ou en soustrayant une ou deux puissances de 2 sont :

- 2^k,
- $2^k + 2^I, 2^k + 2^{I+1}, ..., 2^k + 2^{k-1}$,
- $2^{k+1} - 2^{k-2}, 2^{k+1} - 2^{k-3}, ..., 2^{k+1} - 2^I$.

Preuve : La preuve se divise en deux parties. Tout d'abord nous montrons qu'aucun nombre n'est répété et enfin qu'aucun nombre n'a été oublié.

1) Aucune répétition

Il est clair que les nombres des deux premières lignes sont tous différents. L'exposant de la deuxième puissance augmente d'un nombre à l'autre. De même, les nombres de la troisième ligne sont tous différents entre eux et ils sont aussi placés en ordre croissant puisque l'exposant de la deuxième puissance diminue. Ainsi, si nous prouvons que $2^k + 2^{k-1} < 2^{k+1} - 2^{k-2}$, nous nous assurons que tous les nombres sont différents.

Divisons les deux côtés de l'équation par 2^{k-2}. Nous obtenons

$$2^2 + 2 = 6 < 2^3 - 1 = 7.$$

Il n'y a aucune répétition.

2) Aucun oubli

Supposons qu'il existe un nombre de la forme $\pm 2^n \pm 2^m$ qui se situe dans l'intervalle $[2^k, 2^{k+1})$ et qui n'a pas été considéré. L'exposant n doit être différent de m. Si les deux signes sont égaux alors le total est soit 2^{m+1} ou -2^{m+1}. Aucun nombre négatif n'est présent dans l'intervalle et l'unique puissance de 2 présente dans $[2^k, 2^{k+1})$ a déjà été comptée. Si, par contre, les signes des deux puissances sont contraires, alors le total est zéro et le nombre zéro n'est pas dans $[2^k, 2^{k+1})$.

Ainsi $n \neq m$ et supposons que $n > m$ sans perte de généralité. Le signe associé à la puissance 2^n est nécessairement positif pour que la somme (différence) des deux puissances soit positive. Terminons la preuve en la séparant par cas.

- $n \leq k - 1$. Il est impossible de se retrouver dans l'intervalle $[2^k, 2^{k+1})$ en ajoutant une puissance 2^m à 2^n où $m < n \leq k - 1$ car

$$2^m + 2^n < 2^{k-1} + 2^{k-1} = 2^k.$$

- $n = k$. Le signe de la deuxième puissance doit être positif, s'il est négatif nous sortons de l'intervalle. Le plus petit exposant que nous puissions prendre est I et si nous dépassons $k - 1$, nous sortons à nouveau de l'intervalle. Tous ces exposants ont déjà été comptés.

- $n = k+1$. Le signe de la deuxième puissance doit être négatif pour revenir dans l'intervalle. À nouveau, les exposants disponibles vont de I à k. Il n'y a pas d'exposant plus petit que I de disponible et si nous allons au-delà de k, nous sortons de l'intervalle. Nous devons cependant nous arrêter à $k-2$, pour ne pas compter deux fois le même nombre. Car

$$\begin{aligned} 2^{k+1} - 2^{k-1} &= 4 \cdot 2^{k-1} - 2^{k-1} = 3 \cdot 2^{k-1} = 2^k + 2^{k-1} \text{ déjà compté,} \\ 2^{k+1} - 2^k &= 2^k \text{ déjà compté aussi.} \end{aligned}$$

Ainsi pour $n = k+1$, toutes les possibilités ont été comptées.

- $n \geq k+2$. Puisque $m < n$, m peut aller de I à $n-1$. Tous ces exposants sont trop petits pour nous ramener dans l'intervalle $[2^k, 2^{k+1})$.

La preuve est maintenant complète. □

Corollaire 6.2.1 Dans l'intervalle $[2^k, 2^{k+1}), k = I+1, ..., S-1$, il y a exactement $2(k-I)$ coefficients différents qui peuvent être exprimés comme la somme ou la différence de deux puissances de 2.

Preuve : Pour montrer l'exactitude de la formule, il suffit d'additioner le nombre de coefficients se trouvant sur chaque ligne du théorème 6.2.1. La première en compte un, la deuxième en a, $(k-1) - I + 1 = k - I$. Il y en a $(k-2) - I + 1 = k - I$ sur la troisième ligne. Au total, il y a $2(k-I)$ coefficients. □

Le corollaire nous donne le nombre de coefficients par intervalle. Pour connaître combien il y en a au total, nous faisons une somme sur les différents intervalles.

$$\begin{aligned} \sum_{k=I+1}^{S-1} 2(k-I) &= 2\sum_{k=I+1}^{S-1} k - 2I(S-1-(I+1)+1), \qquad (6.1) \\ &= 2\Big(\sum_{k=1}^{S-1} k - \sum_{k=1}^{I} k\Big) - 2I(S-I-1), \\ &= 2\left(\frac{S(S-1)}{2} - \frac{I(I+1)}{2}\right) - 2I(S-I-1), \\ &= (S^2 - S) - (I^2 + I) - 2IS + 2I^2 + 2I, \\ &= S^2 - S + I^2 + I - 2IS. \end{aligned}$$

Le théoreme 6.2.1 est valide seulement sur les intervalles $[2^k, 2^{k+1})$ pour les puissances $k = I+1, ..., S-1$. Il faut donc considérer le premier et le dernier intervalle. Entre $[2^I, 2^{I+1})$, il n'y a qu'un seul nombre 2^I. Aucune preuve n'est nécessaire. À partir de 2^S et au-delà, il faut aussi compter certains coefficients. Il s'agit uniquement de coefficients de la forme $2^S + 2^k$ avec $k = I, I+1, ..., S$. Tous ces exposants pour k sont acceptés, au total nous avons $S-I+2$ nouveaux coefficients.

En additionnant la somme du nombre de coefficients avec les dernières quantités dénombrées, nous trouvons la formule :

$$\begin{aligned} g(I,S) &= (S^2 - S + I^2 + I - 2IS) + (1) + (S - I + 2), \\ &= S^2 - 2IS + I^2 + 3, \\ &= (S-I)^2 + 3. \end{aligned} \tag{6.2}$$

Il faut tout multiplier par deux pour avoir l'ensemble des coefficients positifs et négatifs.

Proposition 6.2.1 Le nombre de coefficients de la forme $\pm 2^{p_1} \pm 2^{p_2}$ ou $\pm 2^{p_1}$, où p_1 et $p_2 \in [I, S]$ est donné par la formule suivante :

$$f(I,S) = 2(S-I)^2 + 6. \tag{6.3}$$

De plus, $f(I,S) \in O((S-I)^2)$.

Preuve : La preuve de cette proposition découle directement du théorème 6.2.1 et du corollaire 6.2.1 ainsi que du développement précédent. □

Nous pouvions nous attendre à une formule quadratique. Cependant tout ce raisonnement a été très utile afin de comprendre comment nous allons énumérer nos coefficients avec l'algorithme. De plus, posséder une formule exacte nous permettra de vérifier que notre programme de recherche n'a oublié aucun coefficient. Nous aurions pu trouver cette formule pour un nombre λ de puissances de 2. Cependant, l'analyse exacte, déjà assez compliquée dans le cas où $\lambda = 2$, aurait nécessité encore plus de calculs.

6.2.2 Les recherche exhaustives pour des GCL

Nous effectuons une recherche exhaustive avec des m premiers qui se trouvent près d'une puissance de 2. Nous prenons m de la forme $2^e - h, e \in \mathbb{N}$, $h > 0$ et petit. Dans cette première recherche, seuls les coefficients répondant au critère (5.4) sont admis.

m	a	M_8	Nombre de GCLM testés
$2^{10} - 3$	$2^6 + 2^5$	$S_4 = 0,61335$	78
$2^{11} - 9$	$2^7 + 2^5$	$S_4 = 0,63808$	24
$2^{12} - 3$	$2^{10} - 2^6$	$S_8 = 0{,}61243$	134
$2^{13} - 1$	$2^9 - 2^3$	$S_7 = 0{,}61526$	292
$2^{14} - 3$	$2^{11} + 2^7$	$S_6 = 0{,}59553$	206
$2^{15} - 19$	$2^9 - 2^5$	$S_5 = 0{,}62596$	56
$2^{16} - 15$	$2^9 - 2^6$	$S_5 = 0{,}56599$	134
$2^{17} - 1$	$2^8 + 2^6$	$S_8 = 0{,}60654$	516
$2^{18} - 5$	$2^8 + 2^7$	$S_8 = 0{,}63067$	294
$2^{19} - 1$	$2^{12} - 2^7$	$S_4 = 0{,}61317$	652
$2^{20} - 3$	$2^{14} - 2^{10}$	$S_6 = 0{,}65235$	518
$2^{21} - 9$	$2^{11} + 2^4$	$S_8 = 0{,}63821$	344
$2^{22} - 3$	$2^{11} - 2^8$	$S_5 = 0{,}67191$	654
$2^{23} - 15$	$2^{19} + 2^7$	$S_6 = 0{,}64322$	456
$2^{24} - 3$	$2^{22} - 2^{13}$	$S_7 = 0{,}57320$	806
$2^{25} - 39$	$2^{13} + 2^6$	$S_6 = 0{,}65436$	344
$2^{26} - 5$	$2^{14} + 2^7$	$S_3 = 0{,}64638$	806
$2^{27} - 39$	$2^{14} - 2^{11}$	$S_7 = 0{,}60230$	518
$2^{28} - 57$	$2^{14} - 2^{10}$	$S_6 = 0{,}62749$	518
$2^{29} - 3$	$2^{23} + 2^8$	$S_5 = 0{,}58805$	1256
$2^{30} - 35$	$2^{20} + 2^{15}$	$S_7 = 0{,}63630$	654
$2^{31} - 1$	$2^{16} + 2^{11}$	$S_2 = 0{,}63942$	1804
$2^{60} - 93$	$2^{39} - 2^{29}$	$S_8 = 0,63816$	4238
$2^{61} - 1$	$2^{38} - 2^{28}$	$S_8 = 0,42765$	7204
$2^{62} - 57$	$2^{50} + 2^{31}$	$S_5 = 0{,}60936$	5208
$2^{63} - 25$	$2^{39} - 2^{23}$	$S_7 = 0{,}62100$	5624
$2^{64} - 59$	$2^{37} - 2^{29}$	$S_6 = 0{,}62093$	5414

Tableau 6.2: Recherches exhaustives pour des GCLM avec $a = \pm 2^{p_1} \pm 2^{p_2}$

La première colonne donne les valeurs de m pour lesquelles nous avons fait des recherches. Les h correspondent aux plus petites valeurs pour que le m formé soit un nombre premier. La deuxième colonne fournit la valeur de a ayant réussi le mieux au test spectral. À la troisième

colonne, nous retrouvons les résultats au test spectral. Le critère M_8 (2.22) indique que nous avons effectué le test spectral jusqu'en dimension 8 et S_T précise dans quelle dimension le générateur a été le moins bien performant. Enfin, la dernière colonne établit combien de générateurs nous avons testés pour chacun des m avant d'arriver au a optimal.

Wu [44] a présenté quatre GCLM dont deux avec $m = 2^{31} - 1$ et deux autres avec $m = 2^{61} - 1$. Nous les avons décrits à la section 3.4. Nous supposons qu'il s'agit des meilleurs résultats qu'il a obtenus. Pour $m = 2^{31} - 1$, nous arrivons au même résultat comme vainqueur au test spectral pour M_8. Cependant, le deuxième meilleur coefficient est $2^{21} - 2^{15}$ avec $M_8 = S_7 = 0{,}60650$ alors que Wu arrivait à 0,57386 avec $a = -2^{16} - 2^{11}$. Pour $m = 2^{61} - 1$, nous arrivons aussi à de meilleurs résultats, toujours en ce qui a trait au test spectral, avec $a = 2^{38} - 2^{28}$ et $M_8 = S_8 = 0{,}42765$. Le deuxième est $a = 2^{38} + 2^{28}$ avec $M_8 = S_6 = 0{,}41687$. Le meilleur résultat de Wu était 0,37807 avec $a = 2^{42} - 2^{31}$.

Tous les m possèdent au moins un coefficient qui produit un M_8 oscillant autour de 0,6, sauf pour $m = 2^{61} - 1$. Comme nous l'avons vu lorsque $h = 1$, la multiplication du coefficient avec x_{n-1} suivie de l'opération mod revient à un échange de blocs de bits. Cette grande simplicité dans l'opération est peut-être l'explication de ce moins bon résultat au test spectral. Cependant, il existe d'autres m avec $h = 1$, par exemple $m = 2^{31} - 1$ ou $m = 2^{19} - 1$, pour lesquels le résultat du test spectral est très bon. Lorsqu'il n'y a aucune restriction sur le coefficient, il est possible de trouver un générateur avec $m = 2^{61} - 1$, qui répond bien au test spectral [23].

Cette recherche exhaustive nous permet de conclure que de façon générale, il est possible de trouver des coefficients du type $\pm 2^{p_1} \pm 2^{p_2}$ qui produisent un bon générateur selon le test spectral, mais ceci n'est pas une loi absolue. Il s'agit de résultats expérimentaux, nous n'avons pas dérivé cette conclusion de théorèmes.

6.2.3 Une recherche exhaustive pour des GRM

Est-il possible de faire des recherches exhaustives de GRM utiles, c'est-à-dire d'ordre supérieur à 5 ? Par exemple, pour $m = 2^{31} - 1$, il y a 1806 coefficients pour un GCLM. Ainsi, pour un

GRM d'ordre 5 avec le même module, nous aurions autour de 2×10^{16} générateurs à tester. Sachant que tester les 1806 générateurs pour le GCLM a requis 2 secondes, il en prendrait approximativement 2×10^{13} secondes, ce qui correspond environ à 7000 siècles !

Ainsi, la recherche exhaustive n'est pas envisageable, même si nous réduisons de beaucoup la taille de notre espace de recherche par rapport aux recherches traditionnelles, qui considèrent tous les a tels que $0 < a < m$.

6.3 La recherche aléatoire

6.3.1 Les générateurs récursifs multiples

La technique dont nous nous servons pour déterminer le nombre minimum de coefficients non nuls consiste à fixer tous les coefficients à deux ou trois puissances de 2, puis à diminuer graduellement le nombre de puissances permises par coefficient. À un certain moment, les résultats obtenus au test spectral se sont assez dégradés pour que les générateurs trouvés ne puissent être convenablement considérés comme bons et ceci nous fournit une borne sur le nombre minimal de puissances de 2. Nous proposons plusieurs générateurs ayant un nombre de puissances de 2 égal à la borne ou légèrement au-dessus. Nous savons que, plus il y aura de puissances de 2, plus les calculs sont longs.

Les recherches se basent sur les résultats par rapport à M_{16}, c'est-à-dire qu'on évalue le générateur jusqu'en dimension 16, équation (2.22). De façon générale, lorsqu'un générateur est bon jusqu'en dimension 16, il se comporte assez bien aussi en plus grandes dimensions, mais ce n'est pas une loi absolue. Limiter notre recherche en dimension 16 avait comme unique but de diminuer le temps de la recherche individuelle pour tester un plus grand nombre de générateurs. Une fois que nous avons un bon générateur suivant M_{16}, nous effectuons, bien sûr, le test spectral en plus grandes dimensions pour nous assurer de la bonne performance globale du générateur au test spectral. Pour les générateurs listés, nous donnons aussi M_{max}. La valeur max est la plus grande dimension pour laquelle nous pouvons encore effectuer le test spectral. Elle varie car pour certains générateurs, l'évaluation du test spectral en plus

grandes dimensions est plus ardue.

Les GRM d'ordre 5

Nous débutons nos recherches par des GRM d'ordre 5. La variable λ indique le nombre maximal de puissances de 2 permises pour une recherche donnée. Nous commençons avec trois puissances, puis nous nous limitons à deux. Enfin, nous testons plusieurs configurations avec un nombre minimal de puissances.

m	a_1, a_2, a_3, a_4, a_5	M_{16}, M_{max}	Période
$2^{31}-1$	$a_1 = -2^{18}+2^{15}, a_2 = -2^{28}-2^5+2^3,$ $a_3 = -2^{19}+2^{15}+2^{10}, a_4 = 2^{26}-2^{17},$ $a_5 = 2^{24}+2^{17}-2^{10}$	$S_{16} = 0{,}63972$ $M_{42} = S_{31} = 0{,}58171$	$\rho \approx 2^{155}$
$2^{31}-1$	$a_1 = 2^{27}-2^8+2^2, a_2 = 2^{25}-2^8+2^4,$ $a_3 = -2^{15}-2^7-2^3, a_4 = 2^{22}-2^{17}-2^{12},$ $a_5 = 2^{27}-2^{21}-2^{15}$	$S_{10} = 0{,}63303$ $M_{41} = S_{37} = 0{,}61813$	$\rho \approx 2^{155}$
$2^{31}-61$	$a_1 = 2^{18}-2^{12}+2^7, a_2 = -2^{25}+2^8+2^2,$ $a_3 = -2^{25}+2^9, a_4 = 2^{19}+2^{11}, a_5 = 2^{24}+2^{14}$	$S_{11} = 0{,}61139$ $M_{39} = S_{20} = 0{,}55137$	$\rho \approx 2^{155}$

Tableau 6.3: Liste des meilleurs GRM avec $k = 5$ et $\lambda = 3$

Avec $\lambda = 3$, les recherches sont extrêmement simples. Il existe une grande quantité de générateurs ayant $M_{16} > 0{,}55$. Cependant, avec trois puissances de 2 par coefficients, nous nous doutons que la vitesse de ces générateurs n'est pas avantageuse. Les résultats obtenus au test spectral servent de point de comparaison avec les résultats obtenus pour $\lambda = 2$.

m	a_1, a_2, a_3, a_4, a_5	M_{16}, M_{max}	Période
$2^{31}-1$	$a_1 = -2^{28}+2^{17}, a_2 = 2^{23}+2^{20}, a_3 = 2^{15}+2^4,$ $a_4 = -2^{10}+2^5, a_5 = 2^{20}-2^9$	$S_{12} = 0{,}60681$ $M_{44} = S_{24} = 0{,}57480$	$\rho \approx 2^{155}$
$2^{31}-1$	$a_1 = -2^{26}+2^9, a_2 = -2^{26}-2^{11}, a_3 = 2^{18}+2^3,$ $a_4 = -2^{21}+2^{19}, a_5 = -2^{24}-2^{13}$	$S_{11} = 0{,}60673$ $M_{40} = S_{11}$	$\rho \approx 2^{155}$
$2^{31}-61$	$a_1 = 2^{25}-2^6, a_2 = -2^{25}+2^{11}, a_3 = -2^{22}-2^{18},$ $a_4 = -2^{15}+2^{12}, a_5 = 2^{22}+1$	$S_6 = 0{,}58582$ $M_{40} = S_6$	$\rho \approx 2^{155}$

Tableau 6.4: Liste des meilleurs GRM avec $k = 5$ et $\lambda = 2$

Pour $\lambda = 2$, les recherches commencent déjà à être plus ardues. Le nombre de générateurs ayant $M_{16} > 0{,}55$ décroît considérablement par rapport à $\lambda = 3$. Toutefois, quelques générateurs sont retenus et nous pouvons commencer à espérer que leur vitesse est bonne. Pour la recherche suivante, nous voulons que le nombre de puissances de 2 soit minimial en conservant des résultats acceptables au test spectral.

m	a_1, a_2, a_3, a_4, a_5	M_{16}, M_{max}	Période
$2^{31}-1$	$a_1 = 2^7 - 1, a_2 = 2^{12} + 2^2, a_3 = -2^{21} - 1,$ $a_4 = 2^{26}, a_5 = 2^{16} + 2^4$	$S_7 = 0{,}46138$ $M_{41} = S_7$	$\rho \approx 2^{155}$
$2^{31}-1$	$a_1 = -2^{16}, a_2 = -2^{22} - 2^2, a_3 = -2^{26} - 2,$ $a_4 = 2^{11}, a_5 = -2^{20} - 2^8 + 2$	$S_6 = 0{,}45615$ $M_{39} = S_6$	$\rho \approx 2^{155}$
$2^{31}-1$	$a_1 = 2^{24} + 2^5, a_2 = 2^{16}, a_3 = -2^{11},$ $a_4 = -2^{28}, a_5 = -2^{21} + 2^8$	$S_6 = 0{,}44854$ $M_{41} = S_6$	$\rho \approx 2^{155}$
$2^{31}-1$	$a_1 = -2^{24} + 2, a_2 = 0, a_3 = -2^{18},$ $a_4 = -2^4, a_5 = 2^{11} - 1$	$S_6 = 0{,}21735$ $M_{40} = S_6$	$\rho \approx 2^{155}$
$2^{31}-1$	$a_1 = 2^{24} - 1, a_2 = 0, a_3 = 2^{14},$ $a_4 = 2^{19}, a_5 = -2^8 - 1$	$S_6 = 0{,}18783$ $M_{42} = S_6$	$\rho \approx 2^{155}$
$2^{31}-61$	$a_1 = -2^{25} + 2, a_2 = -2^{18}, a_3 = 2^6,$ $a_4 = -2^{14}, a_5 = -2^{23} + 2^{10}$	$S_6 = 0{,}44829$ $M_{40} = S_6$	$\rho \approx 2^{155}$

Tableau 6.5: Liste des meilleurs GRM avec $k = 5$ et un nombre minimal de puissances de 2

Dans le dernier tableau, nous remarquons que les valeurs de M_{16} se situent soit près de 0,45, ou plus bas, autour de 0,2. Pour conserver une valeur de M_{16} au-delà de 0,4, nous devons absolument, d'après les recherches effectuées, admettre jusqu'à six puissances de 2 parmi les coefficients pour l'ensemble d'un générateur. Rappelons que S_t (2.21) donne un rapport et non la véritable distance entre les hyperplans. Ainsi, un S_t de 0,4 par rapport à 0,2 indique que la plus petite distance entre les hyperplans est deux fois plus grande dans le cas où S_t =0,2. Lorsque nous avons un GRM d'ordre 5 ou d'un ordre supérieur, le nombre de points formant les hyperplans est tellement grand qu'un facteur de 2 entre les valeurs de S_t n'est pas tellement désastreux. Cependant nous ne pouvons pas utiliser cet argument indéfiniment, $S_t = 0{,}1$ indiquerait seulement une distance deux fois plus grande que $S_t = 0{,}2$ et ainsi de suite. Une bonne valeur au test spectral demeure environ 0,6, et c'est ce que réussissent à accomplir les meilleurs générateurs ne possédant aucune contrainte quant aux coefficients.

Nous avons tenté de diminuer le nombre de puissances de 2 à moins de six pour l'ensemble des coefficients. Suite à cette diminution, M_{16} se situe autour de $0,2$, ce qui demeure raisonnable. Nous avons essayé de diminuer encore plus le nombre de puissances de 2 présentes. Cette dernière diminution ne donne pas de bons résultats. La valeur M_{16} chute en bas de 0,05. Cette valeur est trop faible et les générateurs qui la produisent ne peuvent être considérés.

Nous avons testé différentes configurations pour les puissances de 2. Mais tester toutes les possibilités est trop long. Par exemple, dans la cas où nous avons à donner quatre puissances de 2 à 5 coefficients en permettant les répétitions. Ceci est égal à

$$P(5,4) = \frac{5!}{(5-4)!} = 120.$$

De plus, il est permis que certains coefficients aient une puissance de 2 plus (ou moins) 1 ou 2. Ceci ajoute encore aux possibilités. Une des contraintes est que $a_k \neq 0$, où k est l'ordre du générateur, pour que les conditions de périodes maximales soient respectées, mais le nombre de possibilités demeure très grand.

Lorsque les générateurs présentent de mauvais résultats au test spectral, le problème se situe habituellement en petites dimensions, soit d'une à trois dimensions au-delà de l'ordre du générateur. Nous ne pouvons conclure que pour tous les générateurs problématiques seules les petites dimensions sont en cause, mais les recherches tendent à démontrer que les petites dimensions sont les plus discriminantes. Pour illustrer ceci, voici les résultats au test spectral du générateur :

$$x_n = \left(- 2^{15} x_{n-1} + 2^{23} x_{n-3} - 2x_{n-4} + (-2^8 - 1) x_{n-5} \right) \bmod (2^{31} - 1).$$

t	d_t	S_t
6	1,192E-7	0,10873
7	1,861E-6	0,08622
8	1,360E-5	0,07644
9	1,360E-5	0,32853
10	2,752E-5	0,51992
11	5,654E-5	0,65319
12	2,230E-4	0,36381
13	3,782E-4	0,41654
14	4,036E-4	0,68781
15	6,979E-4	0,64862
16	1,128E-3	0,61435

Tableau 6.6: Test spectral démontrant les lacunes d'un générateur en petites dimensions

Les GRM d'ordre 6

Pour les GRM d'ordre 6, nous procédons de la même manière qu'avec les GRM d'ordre 5.

m	$a_1, a_2, a_3, a_4, a_5, a_6$	M_{16}, M_{max}	Période
$2^{31}-1$	$a_1 = -2^{22} + 2^{14} + 2^{10}, a_2 = 2^{26} - 2^9 - 2^5,$ $a_3 = 2^{14}, a_4 = 2^{31} + 2^{26} + 2^{17},$ $a_5 = -2^{27} + 2^{20} + 2^9, a_6 = -2^{30} + 2^{22} + 2^{16}$	$S_8 = 0{,}64259$ $M_{41} = S_{23} = 0{,}52684$	$\rho \approx 2^{186}$
$2^{31}-1$	$a_1 = -2^{20} - 2^{18}, a_2 = 2^{23} + 2^{19} - 2^{17},$ $a_3 = 2^6 - 2^4, a_4 = -2^{28} + 2^{18},$ $a_5 = 2^{20} + 2^{10}, a_6 = 2^{23} + 2^{13} - 2^4$	$S_{12} = 0{,}63307$ $M_{40} = S_{22} = 0{,}55533$	$\rho \approx 2^{186}$
$2^{31}-1$	$a_1 = 2^{29} - 2^{14} - 2^{11}, a_2 = 2^{25} - 2^8 + 2^4,$ $a_3 = 2^{24} - 2^2, a_4 = 2^{22} - 2^{17} - 2^{12},$ $a_5 = 2^{23} + 2^{21} + 2^{10}, a_6 = 2^{29} + 2^3$	$S_{13} = 0{,}63089$ $M_{43} = S_{19} = 0{,}57029$	$\rho \approx 2^{186}$
$2^{31}-1$	$a_1 = 2^{24} + 2^{22} - 2^3, a_2 = 2^{28} + 2^{26},$ $a_3 = -2^{17} + 2^{15} - 2^{13}, a_4 = -2^{25} - 2^{11} + 1,$ $a_5 = 2^{21} + 2^{16} - 2^{14}, a_6 = 2^8 + 2^6 - 2^2$	$S_{12} = 0{,}62643$ $M_{41} = S_{19} = 0{,}53517$	$\rho \approx 2^{186}$

Tableau 6.7: Liste des meilleurs GRM avec $k = 6$ et $\lambda = 3$

Tout comme pour $k = 5$, il est très facile de trouver de bons générateurs, par rapport au critère M_{16}, lorsque $\lambda = 3$.

m	$a_1, a_2, a_3, a_4, a_5, a_6$	M_{16}, M_{max}	Période
$2^{31}-1$	$a_1 = 2^{17}+2^4, a_2 = 2^{26}+2^{22},$ $a_3 = 2^{13}-2^{10}, a_4 = -2^{26}+2^5,$ $a_4 = 2^{24}-2^{21}, a_6 = -2^{23}+2^9$	$S_{10} = 0{,}62058$ $M_{40} = S_{18} = 0{,}47213$	$\rho \approx 2^{186}$
$2^{31}-1$	$a_1 = -2^{24}-2^{16}, a_2 = 2^{20}-2^{13},$ $a_3 = -2^{27}-2^{10}, a_4 = -2^{10}-2^7,$ $a_5 = -2^{25}+2^4, a_6 = -2^{25}+2^{20}$	$S_8 = 0{,}61172$ $M_{41} = S_{31} = 0{,}55552$	$\rho \approx 2^{186}$
$2^{31}-1$	$a_1 = 2^{23}+2^{16}, a_2 = 2^{19}-2^{12},$ $a_3 = 2^{27}+2^{15}, a_4 = -2^{10}-2^7,$ $a_5 = -2^4-1, a_6 = 2^{27}+2^{16}$	$S_{14} = 0{,}59149$ $M_{40} = S_{14}$	$\rho \approx 2^{186}$

Tableau 6.8: Liste des meilleurs GRM avec $k = 6$ et $\lambda = 2$

Pour $\lambda = 2$, nous remarquons que les résultats ne se dégradent pas de façon significative. Les bons générateurs sont plus rares, mais en cherchant un peu, nous arrivons encore à en trouver quelques-uns satisfaisant nos exigences.

m	$a_1, a_2, a_3, a_4, a_5, a_6$	M_{16}, M_{max}	Période
$2^{31}-1$	$a_1 = 2^{13}-2, a_2 = 2^{28}, a_3 = -2^{17},$ $a_4 = 2^8, a_5 - 2^{20}, a_6 = 2^{23}-2^6$	$S_7 = 0{,}50395$ $M_{42} = S_7$	$\rho \approx 2^{186}$
$2^{31}-1$	$a_1 = -2^{24}-2^5, a_2 = 2^8, a_3 = -2^{20},$ $a_4 = 2^{24}, a_5 = -2^{13}, a_6 = 2^{28}+2^{15}$	$S_7 = 0{,}50390$ $M_{41} = S_7$	$\rho \approx 2^{186}$
$2^{31}-1$	$a_1 = 2^{27}-2^9, a_2 = -2^{13}, a_3 = 2^{18},$ $a_4 = -2^5, a_5 = -2^{27}, a_6 = 2^{23}-2^{19}$	$S_7 = 0{,}50098$ $M_{41} = S_7$	$\rho \approx 2^{186}$
$2^{31}-1$	$a_1 = -2^{23}, a_2 = -2^9, a_3 = -2^{13},$ $a_4 = -2^{28}, a_5 = 2^{18}, a_6 = -2^4+2$	$S_8 = 0{,}32100$ $M_{41} = S_8$	$\rho \approx 2^{186}$
$2^{31}-1$	$a_1 = -2^{21}-1, a_2 = -2^{12}, a_3 = 2^{16},$ $a_4 = 0, a_5 = 2^7, a_6 = -2^{27}+1$	$S_7 = 0{,}21216$ $M_{41} = S_7$	$\rho \approx 2^{186}$

Tableau 6.9: Liste des meilleurs GRM avec $k = 6$ et un nombre minimal de puissances de 2

À nouveau, comme pour $k = 5$, nous listons les générateurs ayant un M_{16} supérieur à 0,4. Pour produire un $M_{16} > 0{,}4$, le générateur doit posséder au moins sept puissances de 2, selon les résultats obtenus.

Cependant, une valeur inférieure à 0,4 n'est pas nécessairement mauvaise dans le cas d'un GRM d'ordre 6. Pour un M_{16} se situant autour de 0,3, nous avons quelques générateurs,

nous n'en avons listé qu'un seul. Il contient six puissances de 2. Lorsque nous imposons à un des coefficients d'être à zéro, alors M_{16} chute environ à 0,2. Le dernier générateur du tableau précédent possède cinq puissances de 2. Les générateurs produisant une valeur de M_{16} d'environ 0,2 étaient aussi assez nombreux.

Déjà avec un seul coefficient à zéro, nous commençons à avoir assez de difficulté à trouver de bons générateurs. La valeur de M_{16} chute parfois à moins de 0,1 lors de certaines recherches. Nous n'avons donc pas imposé qu'un autre coefficient soit nul. Les résultats obtenus avec un coefficient à zéro laissent fortement présager que les générateurs avec deux coefficients nuls ne sont pas bons au test spectral.

Les GRM d'ordre 7

Pour les GRM d'ordre 7, la démarche fut similaire à celle pour les GRM d'ordre 5 et 6.

m	$a_1, a_2, a_3, a_4, a_5, a_6, a_7$	M_{16}, M_{max}	Période
$2^{31}-19$	$a_1 = -2^{27} - 2^{13}, a_2 = 2^{24} - 2^{15} + 2^{12},$ $a_3 = 2^{26} + 2^{10} + 2^2, a_4 = -2^{17} + 2^{12} + 2^5,$ $a_5 = -2^{18}, a_6 = -2^{20} + 2^{13} + 2^9,$ $a_7 = -2^{19} - 2^9$	$S_{11} = 0{,}60951$ $M_{41} = S_{11}$	$\rho \approx 2^{217}$
$2^{31}-19$	$a_1 = -2^{27} - 2^{13}, a_2 = -2^{16} + 2^{10} + 2^6,$ $a_3 = 2^{26} + 2^{10} + 2^2, a_4 = 2^{23} - 2^{21} - 2^8,$ $a_5 = -2^{20} + 2^{12} - 2^6, a_6 = -2^{25} - 2^{23} + 2^{10},$ $a_7 = -2^{22} - 2^{18}$	$S_{12} = 0{,}60868$ $M_{39} = S_{23} = 0{,}60863$	$\rho \approx 2^{217}$
$2^{31}-61$	$a_1 = 2^{25} + 2^{24} - 2^{19}, a_2 = 2^{25} - 2^{20} - 2^6,$ $a_3 = -2^{25} - 2^{25} - 2^{13}, a_4 = -2^{22} - 2^{18} + 2^7,$ $a_5 = 2^{25} + 2^{10} + 2^2, a_6 = 2^{24} - 2^{17} - 2^{14},$ $a_7 = -2^{24} + 2^9$	$S_8 = 0{,}47693$ $M_{40} = S_8$	$\rho \approx 2^{217}$
$2^{31}-61$	$a_1 = 2^{25} + 2^{24} - 2^{19}, a_2 = 2^{25} - 2^{20} - 2^6,$ $a_3 = -2^{25} - 2^{25} - 2^{13}, a_4 = -2^{16} + 2^{10} + 2^6,$ $a_5 = 2^{25} + 2^{10} + 2^2, a_6 = 2^{24} - 2^{17} - 2^{14},$ $a_7 = 2^{23} - 2^{12}$	$S_8 = 0{,}47125$ $M_{36} = S_8$	$\rho \approx 2^{217}$

Tableau 6.10: Liste des meilleurs GRM avec $k = 7$ et $\lambda = 3$

Nous remarquons que la tendance générale qui tendait à s'imposer, à savoir qu'avec $\lambda = 3$, il est facile de trouver de bons générateurs, avec $M_{16} > 0,6$, ne tient plus avec $m = 2^{31} - 61$ et $k = 7$. Le problème semble se situer en dimension 8. Tous les générateurs répertoriés possèdent un $M_{16} = S_8$. Nous avons testé autour de 50000 générateurs et les 160 meilleurs présentent un $M_{16} = S_8$.

Comme explication, nous voyons deux possibilités : soit le critère (5.4) est trop restrictif, soit le problème est plus profond et, peu importe les coefficients, les générateurs avec $m = 2^{31} - 61$ et $k = 7$ sont mauvais. Afin de vérifier la dernière hypothèse, nous effectuons une recherche sans aucune condition sur les $a_i, i = 1, ..., 7$. Cette recherche nous permet de rejeter la dernière hypothèse puisque nous trouvons une multitude de bons générateurs.

Nous donnons ici, en exemple, les résultats au test spectral pour un bon générateur avec $m = 2^{31} - 61$, $k = 7$ et

$$a_1 = 900929394, \quad a_2 = -534251888, \quad a_3 = -44746272, \quad a_4 = 122986905,$$
$$a_5 = -541415841, \quad a_6 = -429982915, \quad a_7 = 352556536.$$

t	d_t	S_t
8	7,022E-9	0,68804
9	5,185E-8	0,72739
10	3,065E-7	0,63499
11	1,087E-6	0,68316
12	3,161E-6	0,71472
13	8,435E-6	0,68493
14	1,844E-5	0,69904
15	3,961E-5	0,65117
16	7,321E-5	0,64529

Tableau 6.11: Test spectral pour un GRM d'ordre 7 avec $m = 2^{31} - 61$

La condition (5.4) est-elle trop restrictive ? Avec $m = 2^{31} - 61$ les bornes données par (5.4) sont 6 et 25. Cela élimine une grande quantité de coefficients élevés. Rappelons que l'équation (2.25) donne une borne sur la valeur d_t. Cette borne est particulièrement utile pour démontrer la mauvaise performance d'un générateur lorsque nous travaillons avec des coefficients faibles. Regardons ce qui se passe dans le cas présent.

$k = 7$	$k = 7$
$t = 8$	$t = 8$
$\gamma_8 = 2$	$\gamma_8 = 2$
$m = 2^{31} - 61$	$m = 2^{31} - 61$
$d_8^* \approx 3,416 \times 10^{-9}$	$d_8^* \approx 3,416 \times 10^{-9}$
$1/d_8 \leq 2,6633 \times 10^8$	$1/d_8 \leq 1,7755 \times 10^8$
$S_8 \leq 0,909$	$S_8 \leq 0,606$

Tableau 6.12: Calcul de la borne sur d_t pour $m = 2^{31} - 61$ et $6 \leq p \leq 25$

Dans la première colonne, nous supposons que chaque coefficient est inférieur ou égal à $3(2^{25})$, ce qui est toujours le cas. La borne sur S_8 trouvée est près de 1. Dans la deuxième colonne, nous supposons que $a \leq 2(2^{25})$. Ceci n'est pas toujours vrai dans le cas où nous permettons trois puissances de 2 dans la représentation du coefficient. Cependant, cela est vrai pour le cas où il n'y en aurait que deux. De plus, $a = 3(2^{25})$ est un cas très limite, et nous pouvons supposer que la majorité des coefficients sont inférieurs à $3(2^{25})$. Ainsi, pour la deuxième colonne, nous trouvons $S_8 \leq 0{,}606$, cette valeur se démarque plus de 1. Cela nous laisse croire que le problème dans le cas où $m = 2^{31} - 61$ et $k = 7$ est la trop grande restriction posée par l'équation (5.4).

m	$a_1, a_2, a_3, a_4, a_5, a_6, a_7$	M_{16}, M_{max}	Période
$2^{31} - 19$	$a_1 = 2^{26} + 2, a_2 = -2^7 + 2^5, a_3 = 2^{26} + 2^6,$ $a_4 = 2^{18} + 2^8, a_5 = 2^{11} + 2^5,$ $a_6 = 2^{26} + 2^{22}, a_7 = -2^{18} - 2^{15}$	$S_8 = 0{,}55326$ $M_{41} = S_8$	$\rho \approx 2^{217}$
$2^{31} - 19$	$a_1 = 2^{26} + 2, a_2 = -2^{22} + 2^8, a_3 = 2^{12} - 2^2,$ $a_4 = 2^{24} + 2^{14}, a_5 = 2^{26} - 2^6,$ $a_6 = 2^{26} + 2^{22}, a_7 = -2^{18} - 2^{15}$	$S_9 = 0{,}55108$ $M_{40} = S_9$	$\rho \approx 2^{217}$
$2^{31} - 61$	$a_1 = 2^{25} + 2^8, a_2 = 2^{22} + 2^{17},$ $a_3 = 2^{25} - 2^{20}, a_4 = 2^{23} + 2^{13},$ $a_5 = -2^7 - 2^2, a_6 = 2^{25} + 2^{11}, a_7 = 2^{25} + 2^6$	$S_8 = 0{,}32493$ $M_{40} = S_8$	$\rho \approx 2^{217}$
$2^{31} - 61$	$a_1 = 2^{25} + 2^8, a_2 = 2^{22} + 2^{17},$ $a_3 = 2^{25} - 2^{20}, a_4 = 2^{23} + 2^{13},$ $a_5 = -2^7 - 2^2, a_6 = 2^{25} + 2^{11}, a_7 = -2^{25} + 2^2$	$S_8 = 0{,}32493$ $M_{40} = S_8$	$\rho \approx 2^{217}$

Tableau 6.13: Liste des meilleurs GRM avec $k = 7$ et $\lambda = 2$

On remarque que les résultats ne sont pas exceptionnels pour $m = 2^{31} - 61$ et $\lambda = 2$, ce que le calcul théorique avec la borne sur S_8 nous laissait entrevoir.

Pour un nombre minimal de puissances de 2, nous ne testons pas $m = 2^{31} - 61$. Les résultats avec $\lambda = 2$ n'étant déjà pas satisfaisants, on se doute que moins de puissances fera baisser la valeur de M_{16} jusqu'à ce qu'elle soit inacceptable. Nous refaisons le calcul de la borne sur S_8 en posant seulement une puissance de 2 par coefficient. Nous trouvons,

$$\frac{1}{d_8} \leq 8,878 \times 10^7 \Rightarrow S_8 \leq 0,303.$$

Ainsi dans le meilleur des cas, nous pouvons avoir $0,303$. Cependant, comme notre but est encore de réduire le nombre de puissances de 2 à peut-être six, notre borne sera encore plus faible. En conclusion, nous perdons notre temps à explorer dans la direction de $m = 2^{31} - 61$.

m	$a_1, a_2, a_3, a_4, a_5, a_6, a_7$	M_{16}, M_{max}	Période
$2^{31} - 19$	$a_1 = 2^{13} + 1, a_2 = -2^{22}, a_3 = 2^{29} + 1,$ $a_4 = 2^{17} - 2, a_5 = 2^6 + 1, a_6 = 2^{25},$ $a_7 = 2^{18} - 2^{10}$	$S_{11} = 0{,}54426$ $M_{41} = S_{21} = 0{,}54191$	$\rho \approx 2^{217}$
$2^{31} - 19$	$a_1 = -2^6 - 1, a_2 = 2^{24} - 2, a_3 = 2^{12} - 1,$ $a_4 = 2^{28}, a_5 = -2^{14}, a_6 = 2^{20} + 2,$ $a_7 = -2^{16} - 2^{10}$	$S_9 = 0{,}43849$ $M_{42} = S_9$	$\rho \approx 2^{217}$
$2^{31} - 19$	$a_1 = 2^{25} - 1, a_2 = -2^{11}, a_3 = 0,$ $a_4 = 2^{24}, a_5 = -2^{20}, a_6 = 2^7,$ $a_7 = -2^{15} + 2$	$S_8 = 0{,}18131$ $M_{41} = S_8$	$\rho \approx 2^{217}$
$2^{31} - 19$	$a_1 = 2^{25} - 1, a_2 = -2^{11}, a_3 = 0,$ $a_4 = -2^5, a_5 = -2^{20}, a_6 = -2^{16},$ $a_7 = -2^{13}$	$S_8 = 0{,}16218$ $M_{41} = S_8$	$\rho \approx 2^{217}$
$2^{31} - 19$	$a_1 = -2^{12}, a_2 = -2^{20}, a_3 = 2^{14},$ $a_4 = 0, a_5 = 2^{25}, a_6 = -2^6,$ $a_7 = 2^4 + 1$	$S_8 = 0{,}16218$ $M_{41} = S_8$	$\rho \approx 2^{217}$

Tableau 6.14: Liste des meilleurs GRM avec $k = 7$ et un nombre minimal de puissances de 2

Nous remarquons qu'il n'y a presque pas de différence au niveau de M_{16} pour les générateurs où $\lambda = 2$ et le premier générateur présenté dans le tableau ci-dessus. Les coefficients de ce premier générateur ne sont pas des puissances de 2 simples. Pratiquement tous les coefficients sont de la forme $\pm 2^{p_1} \pm 1$ ou 2. Lorsque nous désirons qu'un des coefficients soit nul et que les

autres coefficients soient seulement des puissances de 2 ou presque, les résultats sont moins bons. Nous trouvons plusieurs générateurs avec un M_{16} tournant autour de 0,18. Lorsque nous imposons à un autre coefficient d'être nul, les résultats deviennent inacceptables.

Les GRM d'ordre 8

Il est intéressant de pouvoir comparer les résultats obtenus avec $k = 8$ à ceux obtenus par Wu dans [45]. Nous commençons par faire nos recherches avec $\lambda = 3$ comme dans les cas précédents.

m	$a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$	M_{16}, M_{max}	Période
$2^{31}-1$	$a_1 = 2^{31} - 2^{24} + 2^{14}, a_2 = 2^{14} - 2^7,$ $a_3 = 2^{19} + 2^{18} - 2^{16}, a_4 = 2^{31} + 2^{11} + 2^3,$ $a_5 = -2^{24} + 2^{19} - 2^3, a_6 = 2^{27} - 2^{23} - 2^{19},$ $a_7 = -2^{24} + 2^{16} + 2^{10}, a_8 = -2^{28} - 2^{25} - 2^3$	$S_{12} = 0{,}63540$ $M_{40} = S_{31} = 0,58331$	$\rho \approx 2^{248}$
$2^{31}-1$	$a_1 = -2^{18} + 2^{12} + 2^7, a_2 = 2^{28} + 2^5 + 2^2,$ $a_3 = 2^{22} + 2^7 + 2^5, a_4 = -2^{19} + 2^8 - 2,$ $a_5 = 2^{12} - 2^9 - 2^6, a_6 = 2^{17} + 2^{12} - 2,$ $a_7 = 2^{30} - 2^{27}, a_8 = 2^{26} - 2^{24} - 2^{16}$	$S_{11} = 0{,}63017$ $M_{39} = S_{25} = 0,60711$	$\rho \approx 2^{248}$

Tableau 6.15: Liste des meilleurs GRM avec $k = 8$ et $\lambda = 3$

On trouve facilement de bons générateurs. Ceci ne constitue pas une grande surprise puisque pour $m = 2^{31} - 1$, la condition (5.4) n'est pas restrictive.

m	$a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$	M_{16}, M_{max}	Période
$2^{31}-1$	$a_1 = -2^{19} + 2^9, a_2 = 2^{28} + 2^{15},$ $a_3 = -2^{11} - 2^5, a_4 = 2^{14} + 2^9,$ $a_5 = 2^{31} - 2^{13}, a_6 = 2^{24} + 2^{22},$ $a_7 = 2^{16} + 2^4, a_8 = 2^{22} - 2^5$	$S_9 = 0{,}59292$ $M_{34} = S_{22} = 0{,}50465$	$\rho \approx 2^{248}$
$2^{31}-1$	$a_1 = 2^{17} + 2^{12}, a_2 = 2^{20} + 2^{12},$ $a_3 = -2^{24} - 2^{16}, a_4 = 2^{24} + 2^5,$ $a_5 = -2^{25} + 2^2, a_6 = 2^{28} + 2^{15},$ $a_7 = -2^{10} - 2^7, a_8 = -2^{21} + 2^7$	$S_{11} = 0{,}57073$ $M_{38} = S_{11}$	$\rho \approx 2^{248}$

Tableau 6.16: Liste des meilleurs GRM avec $k = 8$ et $\lambda = 2$

Nous trouvons de bons générateurs et ils sont meilleurs que ceux cités dans [45]. Regardons ce que nous pouvons faire en tentant de réduire le nombre total de puissances de 2.

m	$a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$	M_{16}, M_{max}	Période
$2^{31}-1$	$a_1 = -2^{11}+2, a_2 = -2^4, a_3 = -2^{15}+2^2,$ $a_4 = 2^{27}-2, a_5 = 2^8+2^2, a_6 = 2^{21}-1,$ $a_7 = -2^{25}, a_8 = 2^{30}-2^{17}$	$S_9 = 0{,}46595$ $M_{40} = S_9$	$\rho \approx 2^{248}$
$2^{31}-1$	$a_1 = -2^{11}+2, a_2 = -2^4, a_3 = -2^{15}+2^2,$ $a_4 = -2^{28}+2, a_5 = 2^8+2^2, a_6 = 2^{21}-1,$ $a_7 = -2^{25}, a_8 = 2^{30}-2^{17}$	$S_9 = 0{,}46595$ $M_{41} = S_9$	$\rho \approx 2^{248}$
$2^{31}-1$	$a_1 = -2^4, a_2 = 0, a_3 = 2^{15}, a_4 = 2^{12},$ $a_5 = 2^{22}, a_6 = 2^9, a_7 =, 2^{27} a_8 = 2^{18}-2$	$S_9 = 0{,}23295$ $M_{41} = S_9$	$\rho \approx 2^{248}$
$2^{31}-1$	$a_1 = -2^{24}, a_2 = -2^{19}, a_3 = 2^5, a_4 = -2^{12},$ $a_4 = -2^{12}, a_5 = 0, a_6 = -2^{26}, a_7 = 2^{17},$ $a_8 = 2^8-1$	$S_9 = 0{,}23295$ $M_{39} = S_9$	$\rho \approx 2^{248}$

Tableau 6.17: Liste des meilleurs GRM avec $k = 8$ et un nombre minimal de puissances de 2

Nous observons beaucoup de variance entre le M_{16} des meilleurs générateurs. Des recherches plus longues auraient peut-être permis de faire ressortir de meilleurs générateurs. Tout comme pour $k = 5$, 6 et 7, il a été impossible d'imposer deux coefficients à zéro sans faire chuter M_{16} et S_9 en dessous de 0,1.

6.3.2 Les générateurs récursifs multiples combinés

Pour les générateurs combinés, nous n'effectuons pas exactement la même démarche que pour les non combinés. Nous débutons immédiatement avec un nombre très réduit de puissances de 2. Les générateurs combinés, comme nous l'avons vu au chapitre 2, sont approximativement équivalents à des GRM. Ainsi des coefficients très simples, c'est-à-dire des coefficients nuls ou composés d'une seule puissance de 2, dans les composantes du générateur combiné, peuvent produire des coefficients, dans le GRM non combiné équivalent, où une structure simple est absente. Nous débutons par lister de bons GRM combinés ayant un M_{16} supérieur à 0,5.

Les GRM combinés d'ordre 3

Une condition nécessaire pour avoir une période maximale est qu'au moins deux coefficients soient non nuls, dont l'un d'eux est a_k où k est l'ordre du générateur. Ceci nous oblige à ne fixer qu'un seul coefficient à zéro. Pour les deux autres, nous essayons d'avoir une seule puissance de 2.

Composante 1	Composante 2	Combinaison	M_{16}, M_{max}
$m_1 = 2^{31} - 1$ $a_{11} = 0$ $a_{12} = 2^{22}$ $a_{13} = 2^7 + 1$	$m_2 = 2^{31} - 21069$ $a_{21} = 2^{15}$ $a_{22} = 0$ $a_{23} = 2^{15} + 1$	$m = 4611640770946945613$ $a_1 = 4341088847531259234$ $a_2 = 2349160800583431525$ $a_3 = 3927818590467337243$ $\rho \approx 2^{185}$	$S_{10} = 0{,}60159$ $M_{48} = S_{10}$
$m_1 = 2^{31} - 1$ $a_{11} = -2^{16}$ $a_{12} = 0$ $a_{13} = -2^9 - 2$	$m_1 = 2^{31} - 19$ $a_{21} = 0$ $a_{22} = 2^{14}$ $a_{23} = 2^{22} - 1$	$m = 4611685975477714963$ $a_1 = 512417371580141115$ $a_2 = 3586868824503337885$ $a_3 = 2818753001731361830$ $\rho \approx 2^{185}$	$S_{12} = 0{,}60578$ $M_{41} = S_{18} = 0{,}54075$
$m_1 = 2^{31} - 1$ $a_{11} = 0$ $a_{12} = 2^{19}$ $a_{13} = 2^4 - 2$	$m_1 = 2^{31} - 19$ $a_{21} = 2^7$ $a_{22} = 0$ $a_{23} = -2^{19} - 1$	$m = 4611685975477714963$ $a_1 = 4099276437917852568$ $a_2 = 512347002836586043$ $a_3 = 4355418647277921199$ $\rho \approx 2^{185}$	$S_{13} = 0{,}62621$ $M_{42} = S_{17} = 0{,}56948$
$m_1 = 2^{31} - 1$ $a_{11} = 0$ $a_{12} = 2^{14}$ $a_{13} = -2^{26} + 1$	$m_1 = 2^{31} - 19$ $a_{21} = 2^{17}$ $a_{22} = 0$ $a_{23} = 2^{11}$	$m = 4611685975477714963$ $a_1 = 1024834743160413302$ $a_2 = 1024817150974393462$ $a_3 = 264211419898751547$ $\rho \approx 2^{185}$	$S_5 = 0{,}59340$ $M_{41} = S_{17} = 0{,}57688$

Tableau 6.18: Liste des meilleurs GRM combinés avec $k = 3$, $J = 2$

Le premier générateur du tableau ci-dessus est celui proposé dans [33]. On se souvient que pour $m = 2^{31} - 61$ et $k = 7$, le fait que $h = 61$ soit si grand influençait grandement les résultats obtenus, les possibilités pour les coefficients étaient trop réduites et cela produisait de mauvais résultats. Toutefois, dans le cas du premier générateur du tableau ci-dessus, nous avons $h = 21069$ pour la deuxième composante. Selon la condition (5.4), la seule puissance que nous pouvons accepter est 2^{15}. C'est extrêmement restrictif. Cependant, nous arrivons

très facilement à trouver de bons générateurs malgré la contrainte. C'est le grand avantage des GRM combinés par rapport aux non combinés.

Nous avons essayé de trouver des générateurs avec encore moins de puissances de 2 pour l'ensemble des coefficients. Nos recherches sont demeurées infructueuses.

Nous présentons des GRM combinés d'ordre 3, mais où le générateur combiné est créé à partir de trois composantes.

Composante 1	Composante 2	Composante 3	Combinaison
$m_1 = 2^{31} - 1$ $a_{11} = 2^{11}$ $a_{12} = 0$ $a_{13} = 2^{19} - 1$	$m_2 = 2^{31} - 19$ $a_{21} = 0$ $a_{22} = 2^{15}$ $a_{23} = 2^{17}$	$m_3 = 2^{31} - 61$ $a_{31} = -2^{27}$ $a_{32} = 2^{20}$ $a_{33} = 2$	$m = 99035199407364773673068122 81$ $a_1 = 17501151578282880660065532 47$ $a_2 = 49465217317689693011702063 60$ $a_3 = 76175898592208895169338695 12$ $\rho \approx 2^{264}$ $M_{16} = S_6 = 0{,}62951$ $M_{33} = S_{25} = 0{,}59746$
$m_1 = 2^{31} - 1$ $a_{11} = 2^{4}$ $a_{12} = 0$ $a_{13} = -2^{26} + 1$	$m_2 = 2^{31} - 19$ $a_{21} = 0$ $a_{22} = -2^{25}$ $a_{23} = 2^{7}$	$m_3 = 2^{31} - 61$ $a_{31} = 2^{19}$ $a_{32} = -2^{11}$ $a_{33} = 2$	$m = 99035199407364773673068122 81$ $a_1 = 30286964758008061669069354 03$ $a_2 = 53344815602313714581215391 09$ $a_3 = 58226163671617183823534715 71$ $\rho \approx 2^{264}$ $M_{16} = S_{12} = 0{,}61745$ $M_{33} = S_{19} = 0{,}58621$
$m_1 = 2^{31} - 1$ $a_{11} = 2^{31}$ $a_{12} = 0$ $a_{13} = -2^{17} + 1$	$m_2 = 2^{31} - 19$ $a_{21} = 0$ $a_{22} = -2^{5}$ $a_{23} = -2^{25}$	$m_3 = 2^{31} - 61$ $a_{31} = 2^{16}$ $a_{32} = 2^{22}$ $a_{33} = 2$	$m = 99035199407364773673068122 81$ $a_1 = 38028991377375554671290335 30$ $a_2 = 52714050623595404073516721 83$ $a_3 = 44711979830102869263547324 58$ $\rho \approx 2^{264}$ $M_{16} = S_{10} = 0{,}61708$ $M_{33} = S_{19} = 0{,}56579$
$m_1 = 2^{31} - 1$ $a_{11} = 2^{30}$ $a_{12} = 0$ $a_{13} = 2^{19} - 1$	$m_2 = 2^{31} - 19$ $a_{21} = 0$ $a_{22} = 2^{23}$ $a_{23} = 2^{19}$	$m_3 = 2^{31} - 61$ $a_{31} = 2^{11}$ $a_{32} = 2^{9}$ $a_{33} = 2$	$m = 99035199407364773673068122 81$ $a_1 = 53283819295365028075818439 32$ $a_2 = 11422595951262764772963182 93$ $a_3 = 46308116054158711209761035 55$ $\rho \approx 2^{264}$ $M_{16} = S_{11} = 0{,}61608$ $M_{33} = S_{20} = 0{,}57355$

Tableau 6.19: Liste des meilleurs GRM combinés avec $k = 3$, $J = 3$

Les GRM combinés d'ordre 4

Composante 1	Composante 2	Combinaison	M_{16}, M_{max}
$m_1 = 2^{31} - 1$ $a_{11} = -1$ $a_{12} = 2^{19}$ $a_{13} = 0$ $a_{14} = -2^7 - 1$	$m_2 = 2^{31} - 19$ $a_{21} = -2^{23}$ $a_{22} = 0$ $a_{23} = -2^{15}$ $a_{24} = -2^{11}$	$m = 4611685975477714963$ $a_1 = 3329661293603149154$ $a_2 = 512347002836586043$ $a_3 = 2049634301948754156$ $a_4 = 2818252311624096871$ $\rho \approx 2^{247}$	$S_7 = 0{,}61852$ $M_{33} = S_{17} = 0{,}45844$
$m_1 = 2^{31} - 1$ $a_{11} = -1$ $a_{12} = -2^{20}$ $a_{13} = 0$ $a_{14} = -2^{12} - 1$	$m_2 = 2^{31} - 19$ $a_{21} = 2^9$ $a_{22} = 0$ $a_{23} = -2^{20}$ $a_{24} = -2^5$	$m = 4611685975477714963$ $a_1 = 2305843048942141420$ $a_2 = 3586991969804542877$ $a_3 = 1024694005672123510$ $a_4 = 768614814219672011$ $\rho \approx 2^{247}$	$S_8 = 0{,}61092$ $M_{36} = S_{18} = 0{,}58772$
$m_1 = 2^{31} - 1$ $a_{11} = -1$ $a_{12} = -2^{13}$ $a_{13} = 0$ $a_{14} = 2^{23} + 1$	$m_2 = 2^{31} - 19$ $a_{21} = 2^{10}$ $a_{22} = 0$ $a_{23} = -2^{20}$ $a_{24} = 2^7$	$m = 4611685975477714963$ $a_1 = 256204898702691840$ $a_2 = 4099277399990518232$ $a_3 = 1024694005672123510$ $a_4 = 3329661308643923293$ $\rho \approx 2^{247}$	$S_{15} = 0{,}58319$ $M_{31} = S_{17} = 0{,}55765$
$m_1 = 2^{31} - 1$ $a_{11} = 1$ $a_{12} = -2^{22}$ $a_{13} = 0$ $a_{14} = -2^{12} - 1$	$m_2 = 2^{31} - 19$ $a_{21} = 2^9$ $a_{22} = 0$ $a_{23} = -2^{21}$ $a_{24} = 2^{23}$	$m = 4611685975477714963$ $a_1 = 2818252601534389346$ $a_2 = 512909952785026619$ $a_3 = 2049388011344247020$ $a_4 = 2819253829277576999$ $\rho \approx 2^{247}$	$S_9 = 0{,}57570$ $M_{38} = S_{20} = 0{,}57184$

Tableau 6.20: Liste des meilleurs GRM combinés avec $k = 4$, $J = 2$

Pour les GRM combinés d'ordre 4, tout comme ceux d'ordre 3, il nous a été impossible de fixer plus d'un coefficient à zéro. Les coefficients nuls dans les composantes du générateur ont une influence importante dans le générateur non combiné équivalent. Pour $k = 4$ et deux composantes, nous concluons que nous ne pouvons fixer plus d'un coefficient à zéro, selon nos recherches.

6.4 L'analyse des résultats

Lorsque nous désirons faire des recherches de paramètres pour des générateurs, il faut penser à un critère d'arrêt. Nous aurions pu tester une plus grande panoplie de m ou effectuer des recherches plus longues, nous permettant peut-être de trouver un meilleur générateur qui nous aurait échappé jusqu'à présent. Le problème est qu'il n'y a pratiquement pas de limite au nombre de recherches. Nous estimons que les recherches faites dans le présent chapitre sont suffisantes et atteignent les buts que nous nous étions fixés.

Nous proposons de bons générateurs avec quelques m différents. Nous regardons du côté des GRM combinés pour $k = 3$ et $k = 4$. Il est toujours possible d'avoir une plus grande période mais plus elle est longue, plus l'ordre de notre générateur est grand ou plus il faut combiner de générateurs. Dans les deux cas, le générateur est plus lent.

Nous remarquons que les GRM non combinés avec des coefficients simples donnent de moins bons résultats, au test spectral, que les générateurs combinés soumis à peu près aux mêmes contraintes. Imposer plus d'un coefficient à zéro, dans un générateur non combiné ou dans une composante d'un générateur combiné, ne nous a jamais fourni de bons résultats.

Nous observons aussi que la valeur de h, provenant de $m = 2^e - h$, a un impact assez grand sur la qualité des résultats que nous obtenons avec ce m. Si h est trop grand, alors une certaine partie des grandes puissances de 2 ne peuvent être utilisées, si nous respectons le critère (5.4). Dans le cas où $m = 2^{31} - 61$, par exemple, il est pratiquement impossible de trouver de bons générateurs.

6.4.1 Une nouvelle borne sur S_t

L'équation (5.4) fournit des bornes inférieure et supérieure sur la valeur des exposants p_j, $j = 1, ..., \lambda$, dans $a = \pm 2^{p_1} \pm 2^{p_2} \pm ... \pm 2^{p_\lambda}$. La borne supérieure est donnée par

$$\left\lfloor \log_2 \left(2^e m/(h(m+1))\right) \right\rfloor = \left\lfloor e - \log_2(h) + \log_2(m/(m+1)) \right\rfloor. \tag{6.4}$$

Cette borne, dans le cas où m est grand, est équivalente à

$$\left\lfloor e - \log_2(h) \right\rfloor = e - \left\lfloor \log_2(h) \right\rfloor. \tag{6.5}$$

Si nous supposons que nos coefficients sont formés de λ puissances de 2 alors

$$a_i \leq \lambda(2^{e-\lfloor \log_2(h) \rfloor}), \quad i = 1, ..., k, \tag{6.6}$$

où k est l'ordre du générateur. L'équation (2.25) peut s'écrire ainsi

$$\frac{1}{d_t} \leq \Big(1 + \sum_{i=1}^{k} a_i^2\Big)^{1/2}. \tag{6.7}$$

De même, nous pouvons écrire l'équation (2.21)

$$S_t \leq d_t^* \Big(1 + \sum_{i=1}^{k} a_i^2\Big)^{1/2} = \frac{\Big(1 + \sum_{i=1}^{k} a_i^2\Big)^{1/2}}{\gamma_t m^{k/t}}. \tag{6.8}$$

Ce qui implique, en combinant (6.6) et (6.8), que

$$S_t \leq \frac{\Big(1 + k(\lambda 2^{e-\lfloor \log_2(h) \rfloor})^2\Big)^{1/2}}{\gamma_t m^{k/t}}. \tag{6.9}$$

Nous avons maintenant une formule générale qui donne en fonction de k, λ et $m = 2^e - h$, une borne supérieure sur la valeur de S_t.

Appliquons la nouvelle borne à un exemple. Posons $\lambda = 1, k = 7, t = 8, e = 31$, puis calculons la borne avec différentes valeurs de h.

h	$S_t \leq$
1	19,41
16	1,213
64	0,3032
128	0,1516
1024	0,0185

Tableau 6.21: Exemple de calcul avec la nouvelle borne sur S_t

Nous remarquons que la borne sur S_t est loin d'être serrée lorsque h est près de 1. Mais déjà avec $h = 64$, la borne sur S_t commence à être beaucoup plus serrée et utile. Dans le cas où $h = 1024$, nous voyons qu'il ne sert à rien de faire une recherche pour un générateur, il serait nécessairement mauvais au test spectral.

6.4.2 Les m de la forme $2^e + h$

Aucun générateur n'a été présenté avec un $m = 2^e + h$, où h est petit. Il ne s'agit pas d'un oubli. Rappelons que ce type de modules n'offre pas vraiment d'avantage, sauf celui de procurer un plus vaste éventail de m. De plus, son principal désavantage est qu'il produit une plus courte période qu'un m requerrant le même nombre de bits pour sa représentation. Comme nous trouvons de bons générateurs avec un m de la forme $2^e - h$, il n'a pas été nécessaire d'avoir recours à ceux de la forme $2^e + h$.

6.4.3 Les conclusions préliminaires

Suite aux recherches effectuées, nous concluons que les GRM combinés respectant (5.4) et produisant de bons résultats au test spectral sont beaucoup plus nombreux que les GRM non combinés. Les résultats des générateurs combinés ne semblent pas affectés par la contrainte (5.4) que nous leur imposons.

De plus, nous pouvons espérer que les GRM combinés seront plus rapides, lorsqu'au chapitre 8, nous effectuons les tests de vitesse. Nous basons cette hypothèse sur le fait que nos GRM d'ordre 6 possèdent cinq coefficients non nuls alors que nos GRM combinés avec $J = 2$ et $k = 3$ possèdent deux coefficients nuls par composante. Cependant la combinaison des composantes d'un générateur requiert un certain temps.

Il faut se méfier cependant des générateurs comptant peu de puissances de 2. Ils peuvent cacher des problèmes que le test spectral ne détecte pas, commme nous l'avons vu avec les générateurs de Wu [44]. Les tests statistiques furent plus utiles que le test spectral pour rejeter les générateurs de Wu [44].

Chapitre 7

Les tests statistiques effectués

Au chapitre 2, nous avons introduit la méthodologie des tests statistiques. Nous avons justifié la présence des tests statistiques et nous avons présenté les éléments d'information qu'ils apportent en plus. Au risque de nous répéter, nous rappelons l'hypothèse nulle $\mathcal{H}_0$ à tester : «les valeurs produites par le générateur peuvent être perçues comme des variables aléatoires indépendantes suivant la loi $U(0,1)$».

Pour tous les tests statistiques présentés par la suite, nous présentons la p-valeur, définie en (2.26), correspond à la probabilité, sous $\mathcal{H}_0$, que la statistique calculée prenne une valeur aussi grande que celle que nous observons. Dans ce mémoire, comme dans [28], une p-valeur inférieure à 0,01 ou supérieure à 0,99 sera considérée suspecte. Dans tous les résultats que nous présentons, lorsque nous indiquons ϵ, cela veut dire que la p-valeur est inférieure à 10^{-15} et si nous inscrivons $-\epsilon$, alors la p-valeur est plus grande que $1 - 10^{-15}$. Pour des p-valeurs aussi extrêmes, il sera clair que nous devons rejeter H_0.

Dans ce chapitre, nous faisons passer quelques tests à nos générateurs. Parmi ces tests, il y a le test d'indépendance des bits, test pour lequel les générateurs de Wu se sont mals comportés et dont nous avons parlé au chapitre 2. Les autres tests sont des tests bien connus dans la littérature. Nous préconisons les tests qui travaillent directement avec les bits de la sortie étant donné que nos coefficients sont près de puissances de 2. Il y a plus de risque que nos générateurs échouent ce genre de tests. Nous faisons passer les tests statistiques à tous les générateurs qui se trouvent au tableau 7.1 présenté ci-dessous.

Nous testons aussi un des générateurs de Wu, $x_n = (2^{15} - 2^{10})x_{n-1} \bmod (2^{31} - 1)$, pour tous les tests, sauf le test d'indépendance des bits, qui a déjà été fait dans [30]. Ceci est fait dans le but de pouvoir effectuer des comparaisons entre nos générateurs et celui de Wu.

7.1 Le tableau récapitulatif des générateurs testés

Dans la présente section, nous listons les générateurs dont nous testons la performance aux tests statistiques et par la suite, au chapitre 8, la vitesse. Nous leur associons ici un nom concis pour y référer plus facilement.

Nom	$\rho \approx$	Méthode	Caractéristiques
GRM96a	2^{185}	FA	$m_1 = 2147483647, a_{11} = 0, a_{12} = 63308, a_{13} = -183326$ $m_2 = 2145483479, a_{21} = 86098, a_{22} = 0, a_{23} = -539608$
GRM96b	2^{185}	FA	$m_1 = 2147483647, a_{11} = 0, a_{12} = 63308, a_{13} = -183326$ $m_2 = 2145483479, a_{21} = 86098, a_{22} = 0, a_{23} = -539608$
DL00a	2^{124}	FA	$m = 2^{31} - 1, a_1 = -1, a_2 = 0, a_3 = 0, a_4 = 22093$
GRMPFa	2^{191}	PF	$m_1 = 2^{32} - 209, a_{11} = 0, a_{12} = 1403580, a_{13} = -810728$ $m_2 = 2^{32} - 22853, a_{21} = 527612, a_{22} = 0, a_{23} = -1370589$
GRM00a	2^{155}	DP2	$m = 2^{31} - 1, a_1 = -2^{24} + 2, a_2 = 0$ $a_3 = -2^{18}, a_4 = -2^4, a_5 = 2^{11} - 1$
GRM00b	2^{186}	DP2	$m = 2^{31} - 1, a_1 = -2^{21} - 1, a_2 = -2^{12}, a_3 = 2^{16}$ $a_4 = 0, a_5 = 2^7, a_6 = -2^{27} + 1$
GRM00c	2^{217}	DP2	$m = 2^{31} - 19, a_1 = -2^{12}, a_2 = -2^{20}, a_3 = 2^{14}$ $a_4 = 0, a_5 = 2^{25}, a_6 = -2^6, a_7 = 2^4 + 1$
GRM00d	2^{248}	DP2	$m = 2^{31} - 1, a_1 = -2^4, a_2 = 0, a_3 = 2^{15}, a_4 = -2^{12}$ $a_5 = 2^{22}, a_6 = 2^9, a_7 = 2^{27}, a_8 = 2^{18} - 2$
GRM00e	2^{185}	DP2	$m_1 = 2^{31} - 1, a_{11} = 0, a_{12} = 2^{22}, a_{13} = 2^7 + 1$ $m_2 = 2^{31} - 21069, a_{21} = 2^{15}, a_{22} = 0, a_{23} = 2^{15} + 1$
GRM00f	2^{185}	DP2	$m_1 = 2^{31} - 1, a_{11} = 0, a_{12} = 2^{14}, a_{13} = -2^{26} + 1$ $m_2 = 2^{31} - 19, a_{21} = 2^{17}, a_{22} = 0, a_{23} = 2^{11}$
GRM00g	2^{264}	DP2	$m_1 = 2^{31} - 1, a_{11} = 2^{30}, a_{12} = 0, a_{13} = 2^{19} - 1$ $m_2 = 2^{31} - 19, a_{21} = 0, a_{22} = 2^{23}, a_{23} = 2^{19}$ $m_3 = 2^{31} - 61, a_{31} = 2^{11}, a_{32} = 2^9, a_{33} = 2$
GRM00h	2^{247}	DP2	$m_1 = 2^{31} - 1, a_{11} = -1, a_{12} = -2^{13}, a_{13} = 0, a_{14} = 2^{23} + 1$ $m_2 = 2^{31} - 19, a_{21} = 2^{10}, a_{22} = 0, a_{23} = -2^{20}, a_4 = 2^7$

Tableau 7.1: Tableau des noms des générateurs

Dans la colonne *méthode*, FA représente la *factorisation approximative* et PF signifie que

le générateur est programmé selon la méthode en *point flottant*, ces deux méthodes furent introduites au chapitre 2. Enfin, DP2 est une abréviation pour *décomposition en puissances de 2*, c'est-à-dire, la méthode d'implantation que nous exploitons dans ce mémoire.

7.2 Les justifications des choix des générateurs

GRM96a et GRM96b sont deux implantations du générateur proposés dans [19]. Le générateur DL00a est un des multiples générateurs proposés dans [7]. Tous les générateurs de [7] sont supposés être très rapides selon leurs auteurs. Même si cela était le cas, nous avons vu au chapitre 4 que les générateurs de [7] ne passent pas bien le test spectral. Nous voulons observer leur comportement aux tests statistiques. Le générateur GRMPFa [22] nous fournit un bon exemple de générateur qui exploite efficacement la méthode de calcul en point flottant.

Les générateurs GRM00a à GRM00h proviennent tous des recherches faites au chapitre 6. Nous conservons uniquement des générateurs qui laissent espérer une bonne vitesse de calcul par la suite. Nous favorisons les générateurs qui possèdent le plus grand nombre de coefficients nuls. Nous prévilégions aussi les modules $m = 2^e - h$ avec $h = 1$. Lorsque $h \neq 1$, nous devons effectuer en plus la multiplication Bh (5.2), qui ajoute du temps de calcul. De plus, avec $h \neq 1$, au moins une comparaison de plus est nécessaire pour nous assurer que $2^p A + Bh < m$ et soustraire m si ce n'est pas le cas.

7.3 Le test d'indépendance des bits

Les générateurs de Wu provenant de [44] ont échoué ce test dans [30]. Il s'agit donc d'un test pour lequel les générateurs avec des coefficients qui sont de la forme $\pm 2^{p_1} \pm 2^{p_2}$ peuvent être sensibles. La description de ce test a déjà été faite au chapitre 3. Voici les résultats obtenus pour chacun des générateurs. Nous présentons les résultats graphiques basés sur la variable standardisée $Z_{i,j}$ (3.16). Puis nous donnons au tableau 7.2 les p-valeurs associées au test (3.17). Les paramètres utilisés sont $N = 1, n = 2^{26}, \ell = 30, L = 300$.

Légende

$Z_{i,j} \leq -5$ $Z_{i,j} = -2$ $Z_{i,j} = 0$ $Z_{i,j} = 2$ $Z_{i,j} \geq 5$

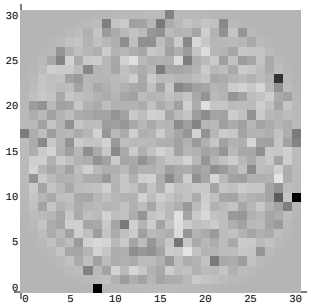

Image des $Z_{i,j}$ pour GRM96a

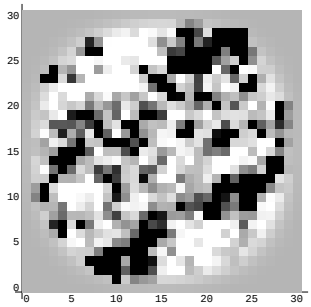

Image des $Z_{i,j}$ pour DL00a

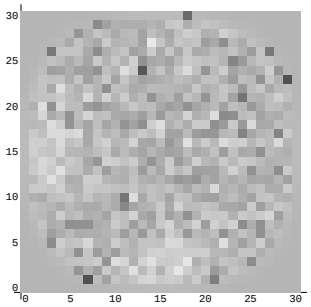

Image des $Z_{i,j}$ pour GRMPFa

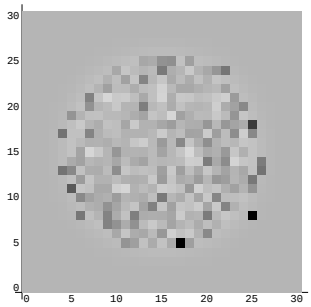

Image des $Z_{i,j}$ pour GRM00a

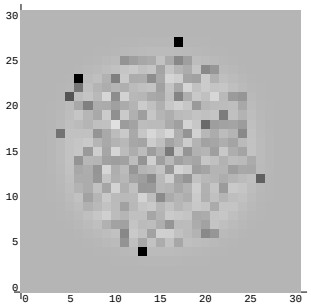

Image des $Z_{i,j}$ pour GRM00b

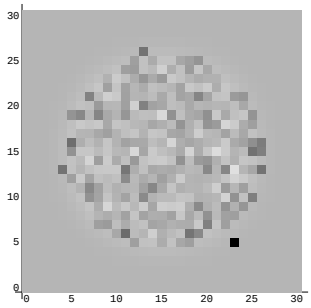

Image des $Z_{i,j}$ pour GRM00c

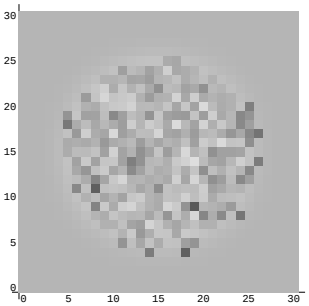

Image des $Z_{i,j}$ pour GRM00d

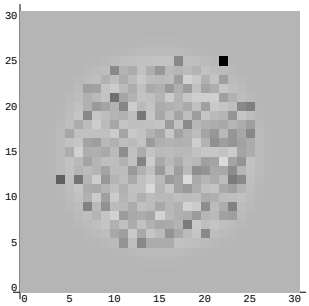

Image des $Z_{i,j}$ pour GRM00e

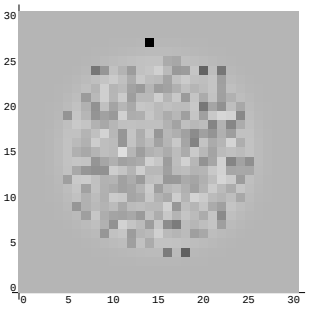

Image des $Z_{i,j}$ pour GRM00f

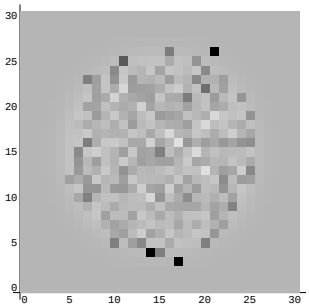

Image des $Z_{i,j}$ pour GRM00g

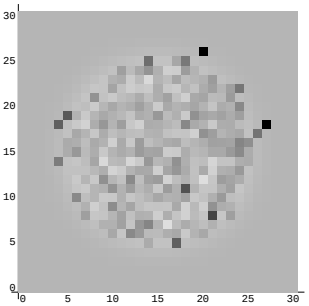

Image des $Z_{i,j}$ pour GRM00h

Figure 7.1: Image des $Z_{i,j}$ pour les générateurs testés

	p-valeur		p-valeur
GRM96a	0,53	GRM00c	0,29
DL00a	ϵ	GRM00d	0,45
GRMPFa	0,25	GRM00e	0,47
GCLMWu	ϵ	GRM00f	0,19
GRM00a	0,88	GRM00g	0,37
GRM00b	0,91	GRM00h	0,54

Tableau 7.2: Les p-valeurs pour le test d'indépendance des bits

Ce test, que les générateurs de Wu [44] échouaient, est réussi par nos générateurs. Les p-valeurs pour le test ne révèlent aucun problème. De plus, les graphiques, qui nous offrent une idée plus visuelle du comportement des générateurs, ne laissent entrevoir aucun problème. Nous voyons que DL00a et GCLMWu échouent ce test de manière évidente (la p-valeur est inférieure à 10^{-15}). Le graphique des $Z_{i,j}$ pour DL00a montre clairement qu'il y a un problème même sans regarder la p-valeur du tableau 7.2. Certains de nos générateurs présentent des carrés plus foncés, c'est-à-dire des $Z_{i,j}$ plus près de 5, sur les graphiques. Ceci est simplement dû au fait que l'approximation normale est moins bonne pour $Z_{i,j}$ lorsque $Np_{i,j}$ est petit, et n'indique pas vraiment un problème dû aux générateurs. Nous remarquons que les carrés plus foncés se situent sur le contour du cercle des carrés aux endroits où la probabilité $p_{i,j}$ devient faible. Nous ne remarquons aucune structure importante dans la couleur des carrés, ce qui serait préoccupant.

7.4 Le test d'espacement des apparitions

Le test d'espacement des apparitions, ou «appearance spacings» en anglais, provient de [36]. Les observations de notre test statistique sont les nombres u_n. Pour chaque nombre u_n, $0 < u_n < 1$, produit par le générateur, nous conservons les s bits suivant le point. Nous utilisons ensuite ces s bits pour former des blocs de L bits. La valeur des variables L et s sont fixées par l'utilisateur. Pour chaque bloc de L bits, il nous faut générer $\lceil L/s \rceil$ nombres u_n. Les derniers bits du dernier nombre ne sont peut-être pas utilisés. Les blocs de L sont ensuite réunis pour former une séquence de $L(Q+K)$ bits, où Q représente le nombre de blocs de L

bits nécessaires pour l'initialisation et K représente le nombre de blocs requis pour effectuer le test.

Pour chacun des K blocs de longueur L généré, c'est-à-dire les blocs construits expressément pour effectuer le test, nous déterminons la plus récente apparition de ce bloc dans la séquence. Par exemple, s'il s'agit du bloc k et que sa plus récente apparition est le bloc $k-i$, alors nous fixons $A_k = i$. Si nous ne pouvons pas trouver de bloc $k-i$ égal au bloc k, alors il s'agit de la première apparition du bloc k et nous posons $A_k = k$. Nous avons besoin de K blocs d'initialisation car sinon il est très probable que $A_k = k$ pour les premiers blocs des L blocs.

La moyenne du logarithme en base 2 des A_k est calculée. Nous répétons ceci N fois, pour avoir N moyennes. Théoriquement, la moyenne des A_k suit approximativement une loi normale dont la moyenne et la variance sont données dans [36]. Nous comparons les N moyennes que nous obtenons avec la moyenne théorique.

Les valeurs de N, L, s, K et Q utilisées sont données dans le tableau suivant. Pour une seule des configurations, nous utilisons un $N > 1$. Si N est grand, nous devons restreindre les valeurs L, K et Q, car nous sommes contraints par la quantité de mémoire. Ainsi, il peut être préférable de prendre un petit N pour tester une plus longue séquence, L, K et Q assez grands.

	N	s	L	K	Q
Configuration 1	1	15	15	2^{20}	2^{20}
Configuration 2	1	15	15	2^{24}	2^{24}
Configuration 3	1	20	20	2^{24}	2^{24}
Configuration 4	1	10	20	2^{24}	2^{24}
Configuration 5	32	15	15	2^{20}	2^{20}

Pour la configuration 5, nous obtenons des statistiques différentes puisque $N = 32$ plutôt que $N = 1$. Il existe plusieurs statistiques que nous pourrions calculer, nous donnons ici seulement la p-valeur de la statistique de Anderson-Darling. Des détails plus complets sur l'explication de cette statistique peuvent être trouvés dans [24].

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5
GRM96a	0,27	0,26	0,29	0,49	0,27
DL00a	0,59	0.53	0,79	0,47	0,02
GRMPFa	0,47	0,40	0,31	0,68	0,25
GCLMWu	ϵ	ϵ	ϵ	ϵ	ϵ
GRM00a	0,16	0,81	0,47	0,26	0,10
GRM00b	0,69	0,15	0,39	0,32	0,18
GRM00c	0,84	0,47	0,62	0,26	0,03
GRM00d	0,45	0,47	0,25	0,87	0,36
GRM00e	0,38	0,78	0,91	0,65	0,11
GRM00f	0,47	0,43	0,49	0,89	0,05
GRM00g	0,79	0,68	0,59	0,48	0,09
GRM00h	0,59	0,62	0,23	0,68	0,13

Tableau 7.3: Les p-valeurs pour le test d'espacement des apparitions

Nos générateurs passent ce test. Les générateurs GRM96a, DL00a et GRMPFa le réussissent aussi. À nouveau, GCLMWu échoue le test.

7.5 Le test de corrélation des bits

Tout comme pour le test d'espacement des apparitions décrit précédemment, nous formons des suites de L bits exactement de la même manière. Dans chaque suite de L bits, nous comptons le nombre de bits à 1 et la variable aléatoire X est égale à ce nombre. Nous calculons ensuite la corrélation empirique entre les valeurs des X pour des blocs successifs. Ce test provient de [24]. L'estimateur de la covariance est donné par la formule :

$$\hat{\rho} = \frac{4}{L\sqrt{n-1}} \sum_{j=1}^{n-1} (X_j - L/2)(X_{j+1} - L/2). \tag{7.1}$$

Enfin, la distribution empirique de $\hat{\rho}$ est comparée avec celle d'une loi normale (0,1), qui est la loi théorique asymptotique des $\hat{\rho}$ lorsque $n \to \infty$.

Les paramètres que nous utilisons pour ce test sont $n = 2^{26}, s = 30, L = 300$.

Générateur	p-valeur	Générateur	p-valeur
GRM96a	0,12	GRM00c	0,21
DL00a	$1 - 4 \times 10^{-4}$	GRM00d	0,81
GRMPFa	0,92	GRM00e	0,10
GCLMWu	ϵ	GRM00f	0,12
GRM00a	0,59	GRM00g	0,80
GRM00b	0,52	GRM00h	0,57

Tableau 7.4: Les p-valeurs pour le test de corrélation des bits

Nos générateurs passent ce test facilement, aucune valeur suspecte n'est présente. Nous pouvons remarquer que le générateur DL00a éprouve de sérieux problèmes et que le générateur GCLMWu échoue le test.

7.6 Le test de collision

Ce test, provenant de [24, 32], consiste à lancer n points dans k cases où $n \leq k$ et à observer le nombre de collisions. C'est-à-dire le nombre de points qui tombent dans une case déjà occupée. Lorsque nous sommes en dimension t, on fixe les k cases en partionnant l'intervalle $[0, 1)$ en ℓ segments égaux afin d'avoir $\ell^t = k$ cases. Ensuite chacune des cases est numérotée. Les n points sont obtenus à l'aide de notre récurrence. Nous générons, pour chacun des n points, t nombres aléatoires consécutifs pour former un point en dimension t.

Le test de collision ne peut s'appliquer que dans le cas clairsemé, c'est-à-dire où la plupart des cases ne sont pas pleines. La densité n/k doit préférablement être inférieure ou égale à 1 pour que le test soit assez sensible. Soit la variable aléatoire Y qui indique le nombre de collisions. Si n est grand, la distribution des Y suit approximativement une loi de Poisson de moyenne $n^2/2k$, sous l'hypothèse $\mathcal{H}_0$ [24]. Le test de collision est répété N fois pour obtenir N valeurs de Y, la somme des Y suit une loi de Poisson de moyenne $\lambda = Nn^2/2k$.

Le programme qui effectue le test de collision utilise une table de hashing pour calculer le nombre de collisions. La grandeur de la table est proportionnelle à n, le nombre de points. La quantité de mémoire disponible impose une limite sur la grandeur de la table de hashing.

Nous prenons pour l'ensemble des tests $n = 2^{23}$, la plus grande valeur de n en considérant la mémoire disponible. Pour que le test de collision soit assez sensible, la valeur de λ ne doit ni être trop petite, ni trop petite. Typiquement une valeur de λ entre 250 et 30000 s'est avérée correcte. Nous fixons $N = 20$ afin de répéter le test suffisamment de fois. Enfin, nous varions arbitrairement les valeurs de ℓ et t pour conserver $250 \leq \lambda \leq 30000$ et $k \approx 2^{40}$.

	N	n	ℓ	t	k	λ
Configuration 1	20	2^{23}	2^{10}	4	2^{40}	640
Configuration 2	20	2^{23}	2^{8}	5	2^{40}	640
Configuration 3	20	2^{23}	2^{6}	6	2^{36}	10240
Configuration 4	20	2^{23}	2^{5}	8	2^{40}	640
Configuration 5	20	2^{23}	2^{4}	10	2^{40}	640
Configuration 6	20	2^{23}	2^{3}	13	2^{39}	1280
Configuration 7	20	2^{23}	3	26	3^{26}	$\approx 276{,}84$

	Conf. 1	Conf. 2	Conf. 3	Conf. 4	Conf. 5	Conf. 6	Conf. 7
GRM96a	0,27	0,95	0,55	0,18	0,75	0,72	0,11
DL00a	$-\epsilon$	$-\epsilon$	$-\epsilon$	ϵ	$-\epsilon$	$-\epsilon$	$-\epsilon$
GRMPFa	0,75	0,75	0,13	0,70	0,28	0,63	0,90
GCLMWu	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ	ϵ
GRM00a	0,73	0,16	0,39	0,21	0,49	0,59	0,22
GRM00b	0,16	0,44	0,45	0,35	0,61	0,91	0,53
GRM00c	0,08	0,51	0,63	0,90	0,97	0,78	0,14
GRM00d	0,34	0,40	0,44	0,66	0,84	0,64	0,05
GRM00e	0,95	0,61	0,04	0,80	0,69	0,15	0,36
GRM00f	0,73	0,11	0,02	0,54	0,41	0,63	0,75
GRM00g	0,87	0,84	0,34	0,58	0,09	0,71	0,87
GRM00h	0,31	0,01	0,71	0,23	0,38	0,36	0,12

Tableau 7.5: Les p-valeurs pour le test de collision

Pour le générateur GCLMWu, il nous a été impossible de faire le test avec $n = 2^{23}$, une importante proportion des points se retrouvait dans les mêmes cases et le nombre de collisions était trop grand. Ceci n'est pas très bon et nous pouvons extrapoler assez aisément que la p-valeur serait de ϵ si le programme pouvait la calculer. Elle est de ϵ pour la plus grande valeur de n que nous avons pu tester.

7.7 Le test de l'espacement des anniversaires

Le test de l'espacement des anniversaires [31] est mieux connu sous le nom anglais de «birthday spacings» . Tout comme pour le test de collision, on partitionne l'intervalle $[0, 1)$ en ℓ segments égaux. Ce qui donne une partition de l'hypercube $[0, 1)^t$ en $k = \ell^t$ boîtes de grandeur égale. Les boîtes sont numérotées de 0 à $k - 1$. Par la suite, on génère nt nombres aléatoires pour construire n vecteurs de dimension t. Le vecteur V_i forme le i^e point et il est égal à $(U_{ti}, U_{ti+1}, ..., U_{t(i+1)-1}), i = 0, ..., n-1$. Pour chacun des points, nous déterminons dans quelle boîte il tombe. Nous énumérons toutes les boîtes I_i, $i = 1, ..., n$ qui recoivent un point. Nous avons au plus n boîtes qui contiennent un point car nous avons n points. Nous classons ces boîtes en ordre croissant de façon à avoir $I_1 \leq I_2 \leq ... \leq I_n$. Nous calculons les espacements $S_j = I_{j+1} - I_j, j = 1, ..., n-1$ puis nous les ordonnons pour obtenir $S_{(1)}, S_{(2)}, ..., S_{(n-1)}$. Posons maintenant la variable aléatoire Y comme étant le nombre de collisions entre les espacements, c'est-à-dire le nombre de valeurs de j pour lesquelles $S_{(j+1)} = S_{(j)}$. Les n points peuvent être vus comme les anniversaires de n personnes dans une année qui compterait k jours.

Si n est grand, la distribution des Y suit approximativement une distribution de Poisson de paramètre $n^3/4k$ sous l'hypothèse $\mathcal{H}_0$. Nous répétons ceci N fois, pour obtenir N valeurs de Y, la somme des Y doit aussi suivre une loi de Poisson, de moyenne $\lambda = Nn^3/4k$. À nouveau, comme pour le test de collision, la valeur de λ doit se situer dans l'intervalle 250 à 30000 pour que le test soit suffisamment sensible. Nous prenons $n = 500000$ pour tous les tests, sauf lorsque nous indiquons $n/2$, dans ce cas $n = 250000$. Nous prenons $k \approx 2^{50}$ et nous adaptons les valeurs de ℓ et t en conséquence. Enfin, nous posons $N = 20$.

n	ℓ	t	k	λ	GRM96a	DL00a	GRMPFa	GCLMWu	GRM00a	GRM00b
n	2^{26}	2	2^{52}	$\approx 138{,}77$	0,50	ϵ	0,81	ϵ	0,85	0,73
n	2^{17}	3	2^{51}	$\approx 277{,}56$	0,98	ϵ	0,13	ϵ	0,50	0,70
n	2^{13}	4	2^{52}	$\approx 138{,}77$	0,57	ϵ	0,64	ϵ	0,18	0,67
n	2^{10}	5	2^{50}	$\approx 555{,}11$	0,19	ϵ	0,05	ϵ	0,18	0,51
$n/2$	2^{8}	6	2^{48}	$\approx 2220{,}44$	0,50	ϵ	0,17	ϵ	0,50	0,12
$n/2$	2^{6}	8	2^{48}	$\approx 2220{,}44$	0,92	ϵ	0,31	ϵ	0,18	0,92
n	2^{5}	10	2^{50}	$\approx 555{,}11$	0,51	ϵ	0,16	ϵ	0,56	0,66
n	2^{4}	13	2^{52}	$\approx 138{,}77$	0,64	ϵ	0,92	ϵ	0,50	0,93
n	2^{2}	26	2^{52}	$\approx 138{,}77$	0,34	ϵ	0,44	ϵ	0,89	0,85

n	ℓ	t	k	λ	GRM00c	GRM00d	GRM00e	GRM00f	GRM00g	GRM00h
n	2^{26}	2	2^{52}	$\approx 138{,}77$	0,44	0,14	0,85	0,50	0,70	0,64
n	2^{17}	3	2^{51}	$\approx 277{,}56$	0,57	0,05	0,09	0,43	0,38	0,52
n	2^{13}	4	2^{52}	$\approx 138{,}77$	0,60	0,83	0,70	0,57	0,64	0,34
n	2^{10}	5	2^{50}	$\approx 555{,}11$	0,57	0,87	0,72	0,81	0,72	0,11
$n/2$	2^{8}	6	2^{48}	$\approx 2220{,}44$	0,97	0,66	0,91	0,62	0,97	0,12
$n/2$	2^{6}	8	2^{48}	$\approx 2220{,}44$	0,55	0,20	0,17	0,83	0,18	0,09
n	2^{5}	10	2^{50}	$\approx 555{,}11$	0,11	0,72	0,30	0,69	0,90	0,16
n	2^{4}	13	2^{52}	$\approx 138{,}77$	0,85	0,05	0,96	0,14	0,64	0,60
n	2^{2}	26	2^{52}	$\approx 138{,}77$	0,07	0,07	0,50	0,11	0,90	0,25

Tableau 7.6: Les p-valeurs pour le test d'espacement des anniversaires

L'ensemble des générateurs proposés passent le test d'espacement des anniversaires pour tous les paramètres essayés. Les générateurs GRM96a et GRMPFa passent ce test. Il s'agit d'un résultat auquel nous pouvions nous attendre, car ils font partie des générateurs amplement testés, au fil des ans, et réputés bons. Cependant les générateurs DL00a et GCLMWu font piètre figure à ce test. Peu importe les paramètres, ils échouent le test lamentablement.

7.8 L'analyse des résultats

Nos générateurs (GRM00*) passent bien tous les tests effectués. Nous n'avons pu déceler aucun problème flagrant relié à la forme particulière de nos coefficients.

Le générateur de Wu échoue tous les tests. Il faut se rappeler qu'il ne s'agit que d'un GCLM et que sa période est moins grande. Ainsi, lorsque nous utilisons un grand nombre de points, les tests statistiques se rendent compte plus facilement que ce générateur est trop uniforme. C'est un problème qui existe avec tous les générateurs à congruence linéaire de courte période.

Par contre, le générateur DL00a passe certains tests et en échoue d'autres. Dans son cas, il est moins limité par sa période, qui est plus longue que celle de GCLMWu. Mais comme nous l'avons démontré au chapitre 4, des problèmes existaient déjà au niveau de sa structure théorique et il n'est donc pas étonnant qu'il échoue quelques tests statistiques.

Chapitre 8

L'implantation de générateurs efficaces

La vitesse d'un générateur dépend fortement de la manière dont il est programmé. Dans les programmes que nous présentons, nous exploitons la structure particulière des coefficients, nous décomposons les coefficients en une somme (différence) de puissances de 2. Cependant, même en utilisant la technique optimisée présentée au chapitre 5, il reste des éléments de programmation à considérer pour minimiser le temps de calcul. Nous traitons, dans le présent chapitre, des différences au niveau de la programmation.

Nous testons nos programmes sur différentes architectures, avec différents compilateurs, pour déterminer s'il y a des différences importantes. Nous comparons les temps obtenus par les générateurs que nous proposons aux autres générateurs.

8.1 Les paramètres utilisés

L'emploi de variables globales est, de loin, la manière la plus efficace de conserver en mémoire les états courants du générateur. C'est la technique que nous allons utiliser même si nous convenons que ce n'est pas une technique de programmation recommandée en général

8.2 La programmation

Nous présentons ici les programmes finaux et optimaux, auxquels nous arrivons pour chacun des générateurs. Avant d'arriver aux programmes optimaux, nous avons effectué différentes tentatives pour programmer les générateurs. Entre autres, nous avons essayé d'exprimer h

par une somme ou différence de puissances de 2 pour remplacer la multiplication Bh (5.2) par des additions et des décalages. Ceci ne fut pas fructueux.

Les générateurs dont nous testons la vitesse sont les mêmes qui ont subi les tests statistiques au chapitre précédent, tableau 7.1. Les générateurs GRM96a, GRM96b, GRMPFa et DL00a sont présentés, même s'ils ne proviennent pas du fruit de notre travail. Ils servent de point de comparaison et nous croyons que le lecteur appréciera de les voir reproduits.

8.2.1 Les programmes

Les programmes des générateurs dans le langage C se retrouvent à l'annexe I. Ces programmes ont été regroupés dans un module que nous appelons `utouzin.c`. À ces fonctions de générations de valeurs aléatoires, nous avons ajouté des fonctions appelées `SetMRG...` et `WrMRG...` afin d'initialiser un générateur et d'écrire les valeurs de l'état d'un générateur. Ce module nous a servi lorsque nous avons effectué les tests statistiques au chapitre 7. En effet, les tests statistiques étant en général assez longs à faire, il est préférable d'avoir nos générateurs programmés de la façon la plus efficace qui soit.

8.2.2 Les compilateurs et les machines

Nous testons nos programmes sur trois architectures différentes. La première est un ordinateur Sun Ultra-2. Sur cette machine, nous utilisons deux compilateurs différents. Le compilateur «cc» , produit par Sun et optimisé pour leurs machines. Puis nous testons nos générateurs avec le compilateur «gcc», il provient de «GNU Compiler Collection». Les deuxième et troisième types de machines sont semblables, il s'agit d'un Pentium-III de 500 MHz et d'un AMD Athlon de 750 MHz sur lesquelles nous testons seulement le compilateur «gcc».

8.2.3 Les options d'optimisation pour les compilateurs

Avec tous les compilateurs, il existe des options d'optimisation qui permettent aux programmes de s'exécuter plus rapidement. Nous donnons ici les options que nous avons utilisées.

Ces options ont semblé produire le code le plus efficace.

Machine	Compilateur	Optimisations
Sun	cc	-fast -xtarget=ultra -xarch=v8plusa
Sun	gcc	-O3 -ffast-math -s -fomit-frame-pointer -finline-functions -funroll-loops -fexpensive-optimizations -fschedule-insns2
Pentium-III	gcc	-O3 -ffast-math -malign-double -s -fomit-frame-pointer -funroll-loops -fexpensive-optimizations -fschedule-insns2 -mwide-multiply -finline-functions
AMD Athlon	gcc	-O3 -ffast-math -malign-double -s -fomit-frame-pointer -funroll-loops -fexpensive-optimizations -fschedule-insns2 -mwide-multiply -finline-functions

Tableau 8.1: Options pour les compilateurs

8.3 Les temps de production des nombres par les générateurs

Nous présentons les temps obtenus pour chacun des générateurs. Nous avons généré 10^7 nombres pour chacun des générateurs et la somme a été calculée.

GNA	$\rho \approx$	Méthode	SUN Ultra-2		Pentium-III 500 MHz	AMD Athlon 750 MHz	Somme
			cc	gcc	gcc	gcc	
GRM96a	2^{185}	FA	18,2	33,9	6,0	3,7	4999897,05
GRM96b	2^{185}	FA	11,4	20,3	6,0	3,3	4999897,05
GRMPFa	2^{191}	PF	5,1	4,7	5,5	2,3	5001090,95
DL00a	2^{124}	FA	4,3	4,0	5,6	2,1	4999403,67
GRM00a	2^{155}	DP2	2,9	4,5	2,6	1,3	5001019,99
GRM00b	2^{186}	DP2	3,0	4,9	2,9	1,4	5000527,01
GRM00c	2^{217}	DP2	5,3	7,7	3,3	1,7	4999419,22
GRM00d	2^{248}	DP2	3,7	6,1	3,5	1,8	4999751,65
GRM00e	2^{185}	DP2	3,3	5,4	2,6	1,4	5000214,81
GRM00f	2^{185}	DP2	2,9	4,4	2,0	1,1	4999683,09
GRM00g	2^{264}	DP2	4,8	6,9	3,1	1,7	4999545,58
GRM00h	2^{247}	DP2	4,7	6,3	2,8	1,4	5001317,67

Tableau 8.2: Temps CPU, en secondes, pour générer et additionner 10^7 nombres aléatoires

8.4 L'analyse des temps de production des nombres

Nous analysons les temps de calcul entre les générateurs pour un même type d'architecture. Commençons par l'architecture Sun. Les machines Sun sont reconnues pour être très efficaces avec le calcul en point flottant. Nous remarquons qu'avec le compilateur «gcc», le générateur GRMPFa est très rapide, mais il est devancé par DL00a et GRM00f. Rappelons que DL00a n'est pas considéré comme un bon générateur selon le test spectral et sa période est inférieure aux autres générateurs de ce tableau. Ainsi, avec le compilateur «gcc» sur Sun, notre nouvelle méthode est légèrement avantageuse.

À nouveau sur l'architecture Sun, mais cette fois avec le compilateur «cc», la décomposition en puissances de 2 procure de très bons résultats. Les générateurs GRM00a à GRM00h ont des temps semblables à ceux de GRMPFa ou nettement inférieurs. Nous réussissons même à surpasser DL00a avec quelques générateurs. Que ce soit avec «cc» ou «gcc», la méthode de la factorisation approximative, représentée par GRM96a et GRM96b, n'est pas comparable avec la méthode en point flottant ou la décomposition en puissances de 2.

Sur les machines Pentium-III ou AMD Athlon, la décomposition en puissances de 2 est plus rapide que les autres méthodes avec les lesquels nous comparons nos générateurs. De tous, GRM00f est deux fois plus rapide que GRMPFa et trois fois plus rapide que GRM96b.

Analysons maintenant les temps des générateurs GRM00a à GRM00h entre eux. De GRM00a à GRM00d, l'ordre du générateur augmente. Il y a donc plus de calculs à effectuer, et il est donc normal que le temps de calcul augmente. Les générateurs GRM00e et GRM00b possèdent environ la même période et nécessitent le même temps. Cependant GRM00f, qui lui aussi a une période semblable à GRM00e et GRM00b, va plus vite que ces derniers.

En effet, pour GRM00e, $m_2 = 2^{31} - 21069$, ce qui implique que nous devons faire des multiplications par $h = 21069$. Pour GRM00f $m_2 = 2^{31} - 19$, les multiplications ne sont faites qu'avec $h = 19$, ce qui est plus facile puisque la multiplication est dans l'ordre de grandeur des nombres multipliés [3]. Aussi, GRM00f ne contient que quatre puissances de 2 parmi tous ses coefficients, alors que GRM00b en possède cinq. Le nombre total de puissances de 2 explique aussi les différences de temps entre GRM00g et GRM00h.

Chapitre 9

Les programmes

Dans ce chapitre, nous proposons des améliorations au programme `seekl` de L'Ecuyer [27] et nous analysons la performance de quelques procédures.

9.1 Quelques lacunes de seekl

Le programme `seekl` (ou la version `seeks`, qui utilise de longs entiers) permet de trouver des générateurs linéaires du type GRM ou GRM combinés selon certains critères. Ce logiciel a été conçu au départ pour trouver des coefficients de n'importe quel type. Il permet aussi de trouver des coefficients qui répondent au critère de la factorisation approximative.

Après la publication en 1997 de l'article de Wu [44], un ajout a été fait au logiciel `seekl` pour lui permettre de trouver des générateurs avec des coefficients du type $\pm 2^{p_1} \pm 2^{p_2}$. Toutefois, il n'y avait à peu près aucune latitude au niveau du type de recherche que nous pouvions exécuter.

Voyons ce qu'il nous était possible de faire comme recherche. La commande dans le fichier de données, pour effectuer des recherches à la manière de Wu, était

MaxBits NbMaxBits HighestBit

MaxBits indiquait que nous voulions un coefficient qui était une somme ou une différence de puissances de 2.
NbMaxBits donnait le nombre de puissances de 2 que nous pouvions inclure dans la somme.
HighestBit fixait la plus grande puissance de 2 permise dans notre somme ou différence.

Nous procédons ici à l'énumération des problèmes relevés avant de se lancer dans la programmation d'une nouvelle version qui permet les recherches de générateurs qui utilisent la décomposition en puissances de 2.

1. La recherche se faisait toujours de manière exhaustive. De plus, le programme ne tenait pas vraiment compte du temps limite indiqué dans le fichier. Ainsi, si nous voulions permettre au coefficient d'être composé de grandes puissances, alors toutes les puissances inférieures étaient aussi testées et cela nécessitait énormément de temps. Le programme `seekl` est fait de telle façon que si le programme ne se termine pas par lui même, alors nous n'avons aucun résultat. Puisque pour des générateurs combinés et/ou d'un ordre assez grand, le nombre de générateurs que nous pouvons tester est très grand, une recherche aléatoire s'avérait nécessaire. La recherche exhaustive demeure intéressante seulement dans les cas où le temps de calcul requis est raisonnable.

2. Toutes les bornes sur les coefficients et sur les puissances composant les coefficients étaient données par la même valeur, soit *HighestBit*. Il arrive souvent que dans le cas d'un GRM combiné, nous fixons certains coefficients à zéro. Le programme ne permettait pas ce type de critère. *NbMaxBit* et *HighestBit* étaient valables pour tous les coefficients d'un générateur.

   Dans le but d'améliorer aussi la vitesse, nous pouvons préférer avoir seulement une puissance comme coefficient au lieu de *NbMaxBits* puissances. Il nous était impossible d'imposer un critère à un coefficient en particulier sans devoir l'imposer à tous les autres.

3. Il n'existait aucune borne inférieure sur la valeur des puissances qui composent le coefficient. Le critère de L'Ecuyer et Simard [30] pose une borne inférieure que nous devons respecter si nous voulons utiliser efficacement la méthode de calcul qu'ils proposent (5.1).

9.2 Les modifications apportées

Nous présentons ici, les améliorations apportées au programme.

1. Ajouter des fonctions qui permettent une recherche exhaustive et une recherche aléatoire.
2. Modifier le format du fichier d'entrée, `entree.dat`, pour que nous puissions imposer des bornes *inférieures* et *supérieures* sur les exposants des puissances de 2.
3. Inclure la possibilité d'avoir un nombre fixé de puissances de 2 qui peut être différent pour chacun des coefficients.
4. Programmer une fonction qui donne automatiquement les bornes sur les exposants suivant le critère (5.1).
5. Ajouter l'option permettant d'avoir de petits exposants sur les puissances de 2 sans pour autant que ces exposants se situent dans l'intervalle donné ou répondent au critère (5.1).
6. Arrêter la recherche lorsque le programme a atteint un certain temps donné dans le fichier `entree.dat`. Ceci existait au préalable dans le programme `seekl` pour d'autres types de recherche.

Premièrement nous redéfinissons en partie la structure du fichier d'entrée qui contient les données pour la recherche. À ce niveau, nous tentons de conserver le plus possible le format d'origine du fichier.

Un des éléments conservé est le nom *MaxBit*. Celui-ci fut choisi en stipulant que le nombre formé de quelques puissances de 2 contenait un nombre maximum de bits à «1», qui était donné par *NbMaxBit*. Ceci n'est pas tout à fait exact dans le cas où nous soustrayons deux puissances, puisqu'alors le nombre de bits à «1» peut être plus grand. Même si ce nom n'est plus représentatif, nous le conservons pour minimiser le nombre de modifications.

Dans la prochaine section, nous présentons la structure générale d'un fichier d'entrée. Puis nous fournissons un exemple typique de fichier avec des données réelles.

9.3 La structure du fichier d'entrée

Le format du fichier est très semblable à ce qui existait auparavant. Pour plus de détails sur la signification des autres champs, voir [27]. La majorité de ce guide n'est pas nouvelle. Nous ne discutons ici que des nouveaux éléments. Les données doivent être placées ligne par ligne dans l'ordre indiqué. Des commentaires peuvent être mis sur la même ligne, à la suite des données, mais séparés par au moins un espace blanc.

ReadGenFile [<*fichier0*>]	
J	
TypeGen	} *(Répéter J fois)*
m	
k	
PerMax	
ImplemCond [<*ImplemCondMB*>]	
*F*1 [<*fichier1*>]	
*F*2 [<*fichier2*>]	
n_1	
$borneinf_{1,1}$ $bornesup_{1,1}$	
$borneinf_{1,2}$ $bornesup_{1,2}$	
$\vdots$	
$borneinf_{1,n_1}$ $bornesup_{1,n_1}$	
$\vdots$	
n_k	
$borneinf_{k,1}$ $bornesup_{k,1}$	
$borneinf_{k,2}$ $bornesup_{k,2}$	
$\vdots$	
$borneinf_{k,n_k}$ $bornesup_{k,n_k}$	
SearchMethod [*n H* H_k]	
C *Dim*(0) *Dim*(1) $\cdots$ *Dim*(*C*)	
MinMerit(1) *MaxMerit*(1) $\cdots$ *MinMerit*(*C*) *MaxMerit*(*C*)	
NbGen(1) $\cdots$ *NbGen*(*C*)	
Criterion <*Norm*>	
LatticeInfo	
LatticeType	
LaGroupSizes *Spacing*	
VerifyBB	
MaxNodesBB	
TimeLimit	
*S*1 *S*2	
ResultForm	

Tableau 9.1: Format du fichier d'entrée pour le programme `seekl` modifié

- *ImplemCond* et *<ImplemcondMB>* : Pour *ImplemCond*, les choix disponibles sont `NoCond, AppFact` ou `MaxBits`. Les deux premiers sont expliqués dans le guide [27]. `MaxBits` permet d'avoir des coefficients de la forme $\pm 2^{p_1} \pm 2^{p_2}$. Si l'option `MaxBits` est active, il faut aussi préciser comment les exposants des puissances de 2 sont choisis.
  - `LecSim` : Les exposants sont choisis selon le critère de L'Ecuyer et Simard (5.1). Dans ce cas, nous ne tenons pas compte de la valeur des bornes sur les puissances de 2 données aux lignes $borneinf_{1,1}$,...,$borneinf_{k,n_k}$.
  - `LecSimP` : Les exposants sont à nouveau choisis selon le critère de l'équation (5.1). Toutefois nous permettons aussi aux petits exposants (0,1,2) d'être présents dans la formation du coefficient. À nouveau, nous ne tenons pas compte de la valeur des bornes sur les puissances données par $borneinf_{1,1}$,...,$borneinf_{k,n_k}$.
  - `PetitExpo` : Nous nous servons ici de la valeur des bornes $borneinf_{1,1}$,...,$borneinf_{k,n_k}$ pour trouver les coefficients mais en permettant aussi d'avoir de petits exposants (0,1,2). Ce critère est nécessaire car il n'est pas possible, avec notre représentation de fichier, d'indiquer avec les bornes que nous voulons soit des grandes puissances situées dans l'intervalle donné, ou de petites puissances pas nécessairement dans ce même intervalle.
  - `Aucun` : Ce critère nous informe de prendre les bornes $borneinf_{1,1}$,...,$borneinf_{k,n_k}$ pour effectuer la recherche.
- n_i : Nombre maximum de puissances de deux qui composent le coefficient i. On doit avoir $n_i \in \{0, 1, ..., 10\}$. Nous ne permettons pas $n_i > 10$ parce qu'à ce niveau, et même avant, la méthode de décomposition en puissances de 2 devient inutile. Il vaut mieux multiplier le coefficient a_i par x_{n-i} par la méthode usuelle.
- $borneinf_{i,\lambda}$ $bornesup_{i,\lambda}$: Il s'agit des bornes inférieures et supérieures sur l'exposant de la puissance $\lambda = 1, ..., n_i$ pour le i^e coefficient. Les variables $borneinf_{i,\lambda}$ et $bornesup_{i,\lambda}$ sont en général dans $\{0, 1, ..., N\}$ où $N = \lfloor \log_2 m \rfloor$. Lorsque le critère (5.1) est appliqué, alors l'exposant ne dépasse pas N. Mais le coefficient pourrait dépasser N sans problème, s'il s'agit d'une exigence de l'usager.
- *SearchMethod* et n, H, H_k : la formulation sur cette ligne est identique à `seekl`. Toutefois elle a une signification particulière dans le cas de *MaxBit*. Pour *SearchMethod*, il y

deux choix : *Exhaust* ou *Random*. *Exhaust* permet de tester seulement des GCLM qui respectent le critère de l'équation (5.1). Nous n'avons pas jugé nécessaire de permettre une recherche exhaustive avec des paramètres plus variés. Ce qui nous intéresse dans le cas d'une recherche exhaustive est la possibilité d'essayer tous les coefficients qui peuvent produire un algorithme rapide.

Avec *Random*, la recherche de générateurs se fera de manière aléatoire. D'autres paramètres devront être précisés sur la ligne dans le cas de *Random*. Ceux-ci sont n, H et H_k. Ici, pour ne pas trop modifier l'apparence du fichier, nous les gardons comme tel. Ils servent encore à établir la grandeur de la recherche mais d'une autre façon. H dicte le nombre de coefficients différents pour chacun des a_i, $i = 1, ..., k-1$, où k est l'ordre de notre générateur. H_k donne le nombre de coefficients différents pour a_k. Une procédure récursive permet d'effectuer toutes les combinaisons entre les coefficients ensuite.

- *C Dim(0) Dim(1) ... Dim(C)* : Pour le cas *MaxBit*, on prend $C = 1$. La recherche aléatoire se fait dans une seule région. $Dim(0)$ indique la plus petite dimension pour les vecteurs à regarder. $Dim(1)$ précise la dimension où nous devons nous arrêter.
- *S1, S2* : La valeur *S1* sert dans la recherche aléatoire pour initialiser le générateur qui permet de générer au hasard des coefficients selon les critères précisés dans le fichier. Le format de la ligne est inchangé.

9.4 Un exemple de fichier d'entrée

Nous présentons un exemple pour mieux comprendre mieux comment nous pouvons construire un fichier pour la recherche de générateurs. Nous recherchons un générateur combiné ($J = 2$) où chacune des composantes est d'ordre 3. Ce fichier nous a permis de trouver le générateur GRM00f, présenté au chapitre 7.

```
FALSE                   Lecture du générateur à partir d'un fichier
2                       J
  MRG                   Type de générateur
  2 31 -1               Module[j] : m
  3                     Ordre[j] : k
  TRUE                  Période maximale
  MaxBits Aucune        Condition sur les coefficients
  Read Facteur1.dat     Facteurs de m-1
  Read Facteur2.dat     Facteurs de r
  0                     Nombre de puissances pour a1
  1                     Nombre de puissances pour a2
  1 31                  Borne inf, borne sup sur p21
  2                     Nombre de puissances pour a3
  1 31                  Borne inf, borne sup sur p31
  0 1                   Borne inf, borne sup sur p32
  Random 10 15 15 8 2   Méthode de recherche
  MRG                   Type de générateur
  2 31 -19              Module[j] : m
  3                     Ordre[j] : k
  TRUE                  Période maximale
  MaxBits Aucune        Condition sur les coefficients
  Read Facteur3.dat     Facteurs de m-1
  Read Facteur4.dat     Facteurs de r
  1                     Nombre de puissances pour a1
  4 27                  Borne inf, borne sup sur p11
  0                     Nombre de puissances pour a2
  2                     Nombre de puissances pour a3
  4 27                  Borne inf, borne sup sur p31
  0 1                   Borne inf, borne sup sur p32
  Random 10 15 15 8 2   Méthode de recherche
1 4 16                  Dimension 2 à 8
0.10 1.0                Valeur min et max de la figure de mérite
8                       Nb de générateurs à conserver
Spectral Rogers         Critère de mérite
Spectral                Information à imprimer
Full                    Type de lattice à analyser
1 1                     Indice lacunaire, espacement
NoVerify                Vérif. de la réduct. de la base avec bornes sur l'erreur
10000000                Nombre de noeuds max dans le branch and bound
2.0                     Temps limite (heure)
80000 98765             S1 S2 : germes pour les générateurs
RES                     Fichier de résultats
```

Tableau 9.2: Exemple d'un fichier d'entrée pour le programme `seeks` modifié

9.5 Les procédures ajoutées au logiciel LatMRG

Nous présentons ici les procédures ajoutées au fichier `LATIO.mod`. Ces procédures sont écrites en Modula-2 tout comme l'ensemble du logiciel LatMRG. Des procédures existantes ont été modifiées, mais la plupart de ces modifications étaient liées à la lecture du fichier d'entrée ou à l'écriture du fichier de sortie. Comme il s'agit de modifications mineures, dans la plupart des cas, nous ne croyons pas nécessaire d'en discuter ici.

```
PROCEDURE CritereLecSim (VAR BInf, BSup : LONGINT; j : INTEGER);
```

Cette procédure calcule les bornes sur les exposants des puissances de 2, composant les coefficients, en suivant les critères de l'équation (5.1).

```
PROCEDURE RechercheMaxBits (j : INTEGER);
```

Cette procédure est appelée lorsque les coefficients du générateur doivent être de la forme $\pm 2^{p_1} \pm \ldots \pm 2^{p_N}$. Elle prend en paramètre la variable j qui indique sur quelle composante du générateur combiné nous effectuons la recherche.

Cette procédure appelle, par la suite, les procédures de recherche exhaustive ou aléatoire tout dépendant du choix de l'utilisateur.

```
PROCEDURE ChoixAleatoireCoeff (j, k, N : LONGINT; t : RngStream; VAR a :
SuperInteger);
```

Cette procédure fournit un coefficient de la forme $\pm 2^{p_1} \pm \ldots \pm 2^{p_N}$. Les valeurs de p_i, $i = 1, ..., N$, se situent entre les borne BorneInf[j,k,i] et BorneSup[j,k,i]. Les tableaux BorneInf et BorneSup sont globaux.

La variable a est la valeur retournée par la procédure. Il faut aussi fournir t, qui est un générateur existant. Ce dernier permet de sortir au hasard des valeurs de p_i, $i = 1, ..., N$, qui se situent dans les intervalles déterminés.

Cette procédure ne retourne pas les valeurs de a avec la même probabilité. Certaines valeurs de a risquent de revenir plus souvent, s'il existe différentes façons de la représenter, comme dans le cas suivant : $12 = 2^3 + 2^2 = 2^4 - 2^2$. Ce défaut de la procédure ChoixAleatoireCoeff est analysée à la section suivante, afin de déterminer son impact sur les recherches.

```
PROCEDURE RechercheAleatoire (j : INTEGER);
```

Cette procédure permet de faire une recherche aléatoire de coefficients sur la composante j du générateur, en respectant les caractéristiques données dans le fichier d'entrée.

```
PROCEDURE MixAi (VAR Indices : ARRAY OF LONGINT; j : INTEGER; VAR TableauAk :
ARRAY OF SuperInteger; VAR TableauAi : ARRAY OF ARRAY OF SuperInteger);
```

La procédure `MixAi` est récursive. Dans le fichier d'entrée, les variables H et H_k sont précisées. La variable H donne le nombre de a_i, $i = 1, ..., k-1$, à tester et H_k spécifie le nombre de a_k. Ainsi, nous fixons pour chacun des a_i, $i = 1, ..., k-1$, H coefficients et H_k coefficients pour a_k, que nous plaçons dans des tableaux TableauAi et TableauAk, respectivement. Ensuite, nous testons toutes les combinaisons possibles des coefficients. Ce nombre de combinaisons est égal à $H^{k-1} \cdot H_k$.

9.6 La recherche aléatoire de coefficients

Lorsque nous désirons trouver des générateurs remplissant les conditions énumérées précédemment, nous devons nous restreindre à effectuer une recherche aléatoire car nous ne pouvons pas tester tous les coefficients possibles dans un temps raisonnable.

Tout d'abord, il est inefficace de choisir uniformément a_i dans un intervalle et de vérifier si ce choix peut s'exprimer comme une somme (différence) de deux puissances de 2 ou comme une puissance seulement. En effet, seule une petite partie des nombres répondent à cette condition. De plus, les calculs qu'il faudrait effectuer ne sont pas simples. Pour cette raison, nous travaillons plutôt à former les nombres en générant les puissances individuellement.

Pour ce faire, nous avons programmé une fonction qui trouve aléatoirement des coefficients répondant aux conditions fixées par l'utilisateur. Cette procédure s'appelle ChoixAleatoire-Coeff, comme nous l'avons vu à la section précédente.

Nous présentons un algorithme simple pour générer les coefficients. Ensuite, nous relevons le problème qui existe avec un tel algorithme et nous nous questionnons à savoir si ce problème est assez important pour devoir être corrigé.

Nous expliquons brièvement le fonctionnement de l'algorithme. Tout d'abord un test est fait pour nous assurer que les bornes précisées par l'usager sont logiques : les bornes inférieures doivent être plus petites que les bornes supérieures.

Ensuite, il faut modifier la valeur de la borne inférieure sur l'exposant, si nous devons aussi inclure les petits exposants (0,1,2). Lorsque cette modification est faite, nous choisissons le signe et l'exposant de la première puissance de 2. Cependant tout n'est pas terminé. L'intervalle d'où provient l'exposant est continu. Cependant, au départ, il est possible d'avoir deux intervalles distincts, par exemple 0 à 2 puis 5 à 26. Nous avons regroupé ces intervalles pour former l'intervalle 2 à 26, où 2 correspond à 0, 3 à 1 et 4 à 2. Il faut donc utiliser une bijection pour retrouver la vraie valeur de l'exposant. De plus, dans tous les cas, il faut ajouter la possibilité où une puissance de 2 est remplacée par 0. Enfin, une boucle sur le nombre de puissances de 2 permet de générer a.

Le problème, comme nous avons pu le remarquer, est que plusieurs coefficients du type $\pm 2^{p_1} \pm 2^{p_2}$ peuvent s'écrire de manières différentes. Par exemple, le nombre 32, peut s'écrire comme $2^4 + 2^4$, 2^5 et $2^6 - 2^5$. De même, le nombre 24, qui n'est pas une puissance de 2, peut s'écrire comme $2^4 + 2^3$ et $2^5 - 2^3$.

Lorsque nous effectuons notre recherche de coefficients, il n'est pas capital d'avoir une parfaite uniformité : c'est plutôt une propriété souhaitable pour notre générateur. Toutefois une analyse plus élaborée est nécessaire pour connaître quelle est la proportion des nombres qui peuvent revenir plus souvent. Cette proportion, si elle devient trop grande, peut nuire à notre recherche. Nous risquons de perdre notre temps à tester deux fois le même générateur.

```
procedure CoeffAleSimple( j,k,N : entier ) : entier;
var    Exposant, Signe,BorneI, i : entier;
begin

// Test sur les bornes
for i := 1 to N
  if (BorneSup[j,k,i]< 0) then return 0; end;
end;

a := 0;
for i := 1 to N

  // Ajustement des bornes
  if (ImplemC[j]=(LecSimP or PetitExpo)) then
    if (BorneInf[j,k,i]≥3) then BorneI := BorneInf[j,k,i]−4;
    else BorneI := -1;
    end;
  else BorneI := BorneInf[j,k,i]−1;
  end;

  // Signe
  Signe := 2 × DiscreteUniform(0,1)−1;
  // Exposant
  Exposant := DiscreteUniform(BorneI,BorneSup[j,k,i]);

  // Bijection entre Exposant et Vrai Exposant
  if (Exposant < BorneInf[j,k,i]) then
    if (((ImplemC[j]= (LecSimP or PetitExpo)) and (BorneInf[j,k,i] ≥ 3)) then
      Exposant := Exposant − (BorneInf[j,k,i]−3);
    elsif((ImplemC[j]=LecSim) OR (ImplemC[j]=Aucune)) then
      Signe := 0;
    end;
  end;
  if (Exposant < 0) then Signe := 0; end;

  // Mise à jour de a
  if (Signe ≠ 0) then
    a := a + Signe × 2^Exposant
  end;

end;
return a;
end;
```

Tableau 9.3: Pseudo-code pour l'algorithme qui génère les coefficients

Dans tous les exemples décrits précédemment, il s'agit de façons totalement différentes d'écrire les nombres. En effet, $2^6 - 2^5$ et $-2^5 + 2^6$ sont des représentations équivalentes et puisque

celles-ci arrivent en quantités égales, il n'est pas nécessaire de se soucier de la répétition.

En y regardant de plus près, nous remarquons que les nombres possédant deux ou trois structures différentes sont construits de la même manière. Lorsque nous avons une puissance de 2, soit 2^p, les constructions $2^{p+1} - 2^p$ et $2^{p-1} + 2^{p-1}$ donnent aussi 2^p. Ainsi trois manières différentes sont possibles pour trouver le même coefficient. Pour les nombres qui ne sont pas une puissance de deux, deux constructions différentes peuvent produire le même nombre, soient $2^{p+1} - 2^{p-1} = 2^p + 2^{p-1} = 3(2^{p-1})$. Il faut aussi se méfier du nombre zéro, produit à chaque fois que les puissances 2^p et -2^p sont générées.

9.7 L'analyse de la présence répétitive de coefficients

Le calcul du nombre exact de coefficients, lorsque p_1 et p_2 sont dans l'intervalle $[I...S]$, a été fait au début du chapitre 6. La formule est donnée par :

$$2(S - I)^2 + 6 \tag{9.1}$$

La question est : combien de fois pouvons-nous tester deux fois un même coefficient ? La proportion des nombres répétés sur l'ensemble des nombres est la quantité qui nous intéresse. Comme nous l'avons fait au chapitre 6, pour déduire la formule précédente, nous nous limitons à analyser le cas où I et S sont identiques pour toutes les puissances composants le coefficient.

Supposons toujours que p_1 et p_2 sont dans $[I...S]$. Dans les deux tableaux suivants, nous donnons les puissances simples ou les nombres de la forme $3(2^p)$ ainsi que, dans la deuxième colonne, le nombre de façons différentes de les générer.

Les puissances de deux simples

Puissance de 2	Quantité
2^I	2
2^k, $k = I + 1, ..., S - 1$	3
2^S	2
2^{S+1}	1

Au total, il y a

$$1 + 2\big((S-1) - (I+1) + 1\big) + 1 = 2(S-I), \text{ si } I \leq S+2 \tag{9.2}$$

puissances de 2 qui sont des répétitions d'autres puissances de 2.

Les nombres de la forme $3(2^p)$

Nombre de la forme $3(2^p)$	Quantité
$3(2^k),\ k = I, ..., S-2$	2
$3(2^{S-1})$	1

Au total,

$$\big((S-2) - I + 1\big) = S - I - 1, \text{ si } I \leq S+2 \tag{9.3}$$

nombres de la forme $3(2^p)$ répétés.

En considérant tous les coefficients répétés, le total est

$$2(S-I) + S - I - 1 = 3(S-I) - 1, \text{ si } I \leq S+2. \tag{9.4}$$

Seuls les coefficients positifs ont été comptés. En ajoutant les négatifs, on obtient :

$$2(S-I) + S - I - 1 = 6(S-I) - 2, \text{ si } I \leq S+2. \tag{9.5}$$

La proportion du nombre de coefficients répétés est égale à

$$\frac{6(S-I) - 2}{(2(S-I)^2 + 6) + (6(S-I) - 2)}, \text{ si } I \leq S+2. \tag{9.6}$$

En posant, $S - I = \Delta$, on retrouve

$$\frac{6\Delta - 2}{2\Delta^2 + 6\Delta + 4}, \text{ si } \Delta \geq 2. \tag{9.7}$$

Ainsi, si la différence entre S et I est faible, nous perdons notre temps souvent.

Nous présentons ci-dessous les proportions pour des différences $\Delta = S - I$ variant de 2 à 31, les écarts typiques rencontrés.

Écart (Δ)	Proportion	Écart (Δ)	Proportion
2	0.4167	17	0.1462
3	0.4000	18	0.1395
4	0.3667	19	0.1333
5	0.3333	20	0.1277
6	0.3036	21	0.1225
7	0.2778	22	0.1178
8	0.2556	23	0.1133
9	0.2364	24	0.1092
10	0.2197	25	0.1054
11	0.2051	26	0.1019
12	0.1923	27	0.0985
13	0.1810	28	0.0954
14	0.1708	29	0.0925
15	0.1618	30	0.0897
16	0.1536	31	0.0871

Tableau 9.4: Tableau des proportions de la répétition des coefficients

Ces proportions indiquent que si nous testons tous les coefficients que nous pouvons générer avec l'algorithme ChoixAleatoireCoeff, une certaine proportion du temps est perdue à tester des coefficients identiques. Toutefois, nous sommes très loin de tester l'ensemble des coefficients.

Pour un a_i, $i = 1, ..., k$, donné, nous sortons au hasard, pour une recherche, en général dix coefficients. La probabilité de prendre deux fois le même coefficient est non nulle même si nous n'avons qu'une seule façon de générer chaque coefficient. La probabilité de prendre un coefficient en particulier est faible et est égale à

$$\frac{\kappa}{2\Delta^2 + 6\Delta + 4}, \text{ si } \Delta \geq 2. \tag{9.8}$$

où κ est le nombre de façons différentes de générer ce coefficient. On voit que la probabilité de prendre un coefficient plus qu'un autre est seulement facteur de κ. Comme la probabilité est petite et κ ne dépasse pas trois, les risques de perdre notre temps à tester le même générateur sont minimes. Pour cette raison, nous conservons l'algorithme ChoixAleatoireCoeff.

Une façon de régler le problème consiste à nous assurer que toutes les possibilités sont différentes pour un même a_i. Si cela n'est pas le cas alors nous pouvons générer d'autres a_i

différents pour remplacer les dédoublements. Cette manière de procéder ne vient pas démolir tout le raisonnement que nous avons fait sur la présence répétitive de coefficients. En effet, lorsque nous faisons des recherches successives nous pourrions aussi nous retrouver avec les mêmes coefficients, mais dans ce cas là, nous ne possédons aucune manière de la savoir. Nous ne voulons pas conserver en mémoire tous les paramètres des générateurs testés.

Chapitre 10

Conclusion

Nous avons débuté notre étude des générateurs rapides par un survol des méthodes «rapides» existantes de génération de nombres alétoires. Nous avons pu remarquer que certaines méthodes, telles que celle de Wu [44] et celle de Deng et Lin [7] proposaient des avenues intéressantes mais étaient loin d'être des méthodes suffisament robustes du point de vue théorique. Nous avons démontré que la méthode de Deng et Lin présente une faille importante et il faut donc se méfier de ce genre de générateur.

Après avoir fait un tour d'horizon assez complet des GRM, nous avons été en mesure de préciser quel type de recherches nous voulions faire et quelles caractéristiques nos programmes de recherche devaient posséder.

Nous savions que d'obliger les coefficients à être de la forme $a = \pm 2^{p_1} \pm 2^{p_2}$ réduisait de beaucoup l'éventail des coefficients disponibles pour nos générateurs. Nous avons donc analysé théoriquement le nombre de coefficients et il en a résulté une formule exacte pour le nombre de coefficients de la forme $a = \pm 2^{p_1} \pm 2^{p_2}$ ou $a = \pm 2^{p_1}$. Grâce à cette formule, nous avons déterminé de façon certaine qu'une recherche exaustive pour des générateurs d'ordre supérieur à 5 était pratiquement interminable.

Notre travail a consisté ensuite à déterminer les lacunes principales du programme `seekl`, pour la recherche de générateurs avec des coefficients de la forme $a = \pm 2^{p_1} \pm 2^{p_2}$, et à les régler. Avec un nouveau programme, plus flexible, nous avons procédé à la recherche de générateurs qui réussissaient bien le test spectral en plusieurs dimensions selon les critères désirés.

Cependant nous ne voulions pas uniquement des générateurs qui passaient bien le test spectral, le nombre de puissances de 2 composant les coefficients de notre générateur devait

être minimal. Nous avons restreint nos recherches à des générateurs possédant au moins un zéro comme coefficient et une seule puissance de 2 pour chacun des autres coefficients. Les générateurs que nous proposons passent bien le test spectral dans toutes les dimensions qu'il nous a été possible de tester.

Nous nous sommes rendu compte que si le h composant $m = 2^e - h$ est trop grand et les exposants des puissances de 2 doivent être dans l'intervalle (5.4) alors les qualités théoriques de notre générateur en souffrent. Nous avons dérivé une borne sur la valeur de S_t en nous servant de bornes existantes. Cette borne nous donne une idée assez juste sur la qualité des meilleurs résultats que nous pouvons obtenir pour un $m = 2^e - h$ et k donnés.

Les tests statistiques ont démontré que nous pouvons avoir confiance dans les générateurs proposés. Le fait que les coefficients soient de la forme $a = \pm 2^{p_1} \pm 2^{p_2}$ ne vient pas altérer leur qualité.

Ensuite, nous avons programmé nos générateurs dans le langage C afin de tester leur vitesse. Il s'est avéré qu'avec des processeurs Pentium-III et AMD Athlon, nos générateurs sont de deux à trois fois plus rapides que d'autres GRM existants et ayant de bonnes propriétés théoriques. Sur l'architecture Sun, nos générateurs ne sont pas vraiment plus rapides mais ils sont loin d'être plus lents : leur vitesse est comparable. Nos générateurs se retrouvent dans un module appelé `utouzin.c` qui peut être intégré à la librairie `testu01` utilisée au laboratoire d'optimisation du DIRO.

Ainsi, les générateurs que nous proposons sont aussi bons théoriquement et statistiquement que leurs prédécesseurs du même type mais tout en étant deux à trois fois plus rapides. Ce facteur deux n'est pas négligeable pour une personne qui se sert abondamment des générateurs de nombres aléatoires, pour des simulations par exemple.

Les développements futurs

Il pourrait être intéressant d'avoir un programme qui effectue ses recherches selon les indices lacunaires tout en exigeant que les coefficients soient de la forme $a = \pm 2^{p_1} \pm 2^{p_2}$.

L'idée introduite par P.C. Wu d'utiliser des sommes ou différences de puissances de 2 comme coefficients ne se limite pas aux générateurs à congruence linéaire. Dans les générateurs non

linéaires [21], il y a également des multiplications par un coefficient. Jusqu'à maintenant, un des points qui a freiné l'émergence des générateurs non linéaires est le temps de calcul beaucoup trop long par rapport aux générateurs linéaires.

Niederreiter, dans [39, 40], propose une technique appelée *génération de vecteurs pseudo-aléatoires par la méthode matricielle récursive multiple*. Cette méthode contient beaucoup de liens avec les GRM : les coefficients sont matriciels plutôt qu'un scalaire et les états sont des vecteurs. Ainsi, il est possible d'appliquer la méthode de décomposition en puissances de 2 pour les scalaires qui composent les matrices. Il s'agit d'une autre application intéressante.

Bibliographie

[1] Blouin, F., *Mémoire de maîtrise : Générateurs à congruence linéaire généralisés.* Département d'informatique, École des gradués, Université Laval, 1989.

[2] Boyar, J., «Inferring sequences produced by linear congruential generator missing low-order bits», *Journal of Cryptography*, vol. 1, 1989, pp. 177–184.

[3] Brassard, G. et P. Bratley, *Fundamentals of Algorithmics.* Prentice Hall, 1996.

[4] Bratley, P., B. L. Fox et L. E. Schrage, *A Guide to Simulation*, 2e éd. Springer-Verlag, 1987.

[5] Conway, J. H. et N. J. A. Sloane, *Sphere Packings, Lattices and Groups*, 3e éd., vol. 290. Springer-Verlag, 1999.

[6] Coveyou, R.R. et R.D. Macpherson, «Fourier analysis of random number generators», *Journal of ACM*, vol. 14, 1967, pp. 100–119.

[7] Deng, L.-Y. et D. K. J. Lin, «Random number generation for the new century», *American Statistical Association*, vol. 54, no. 2, 2000, pp. 145–150.

[8] Dieter, U., «How to calculate shortest vectors in a lattice», *Math. Comp.*, vol. 29, 1975, pp. 827–833.

[9] Fishman, G.S. et L.R. Moore, «A statistical evaluation of multiplicative congruential random number generators with modulus $2^{31} - 1$», *Journal of the American Statistical Association*, vol. 77, 1982, pp. 129–136.

[10] Fishman, G. S., *Monte Carlo : Concepts, Algorithms, and Applications.* Springer Series in Operations Research. Springer-Verlag, New York, 1996.

[11] Gottfried, B. S., *Schaum's Outline Series, Programming with Pascal.* McGraw-Hill, 1985.

[12] Grimaldi, R. P., *Discrete and Combinatorial Mathematics*, 3e éd. Addison-Wesley, 1994.

[13] IBM. *System/360 Scientific Subroutine Package, Version* III*, Programmer's Manual*. White Plains, New York, 1968.

[14] Knuth, D. E., *The Art of Computer Programming : Seminumerical Algorithms*, 3e éd., vol. 2. Addison-Wesley, 1998.

[15] Krawczyk, H. «How to predict congruential generators». Dans *Advances in Cryptology : Proceedings of CRYPTO'89*, G. Brassard, Éd., vol. 435 de *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1990, pp. 138–153.

[16] Law, A. M. et W. D. Kelton, *Simulation Modeling and Analysis*, 3 éd. McGraw-Hill, 2000.

[17] L'Ecuyer, P., «Random number generation for simulation», *Communications of the ACM*, vol. 33, no. 10, 1990, pp. 85–97.

[18] L'Ecuyer, P., «Uniform random number generation», *Annals of Operations Research*, vol. 53, 1994, pp. 77–120.

[19] L'Ecuyer, P., «Combined multiple recursive random number generators», *Operations Research*, vol. 44, no. 5, 1996, pp. 816–822.

[20] L'Ecuyer, P., «Bad lattice structures for vectors of non-successive values produced by some linear recurrences», *INFORMS Journal on Computing*, vol. 9, no. 1, 1997, pp. 57–60.

[21] L'Ecuyer, P., *Handbook on simulation, chapitre 4, Random number generation*, Jerry Banks éd. Wiley-Interscience, 1998.

[22] L'Ecuyer, P., «Good parameters and implementations for combined multiple recursive random number generators», *Operations Research*, vol. 47, no. 1, 1999, pp. 159–164.

[23] L'Ecuyer, P., «Tables of linear congruential generators of different sizes and good lattice structure», *Mathematics of Computation*, vol. 68, no. 225, 1999, pp. 249–260.

[24] L'Ecuyer, P. *TestU01, Un logiciel pour appliquer des tests statistiques à des générateurs de valeurs aléatoires.* Département d'informatique et de recherche opérationnelle, Université de Montréal, 2000.

[25] L'Ecuyer, P., F. Blouin et R. Couture, «A search for good multiple recursive random number generators», *ACM Transactions on Modeling ans Computer Simulation*, vol. 3, no. 2, 1993, pp. 87–98.

[26] L'Ecuyer, P. et R. Couture, «An implementation of the lattice and spectral tests for multiple recursive linear random number generators», *INFORMS Journal on Cumputing*, vol. 9, no. 2, 1997, pp. 206–217.

[27] L'Ecuyer, P. et R. Couture. *LATMRG : A software package for theoretical analysis of linear congruential and multiple recursive random number generators.* Département d'informatique et de recherche opérationnelle, Université de Montréal, 2000.

[28] L'Ecuyer, P. et P. Hellekalek. «Random number generators : Selection criteria and testing». Dans *Random and Quasi-Random Point Sets*, P. Hellekalek et G. Larcher, Éds., vol. 138 de *Lecture Notes in Statistics*. Springer, New York, 1998, pp. 223–265.

[29] L'Ecuyer, P. et C. Lemieux, «On the choice of quasi-random point sets with a lattice structure», *Proceedings of Monte Carlo Simulation 2000*, Juin 2000.

[30] L'Ecuyer, P. et R. Simard, «Beware of linear congruential generators with multipliers of the form $a = \pm 2^q \pm 2^r$», *ACM Transactions on Mathematical Software*, vol. 25, no. 3, 1999, pp. 367–374.

[31] L'Ecuyer, P. et R. Simard, «On the performance of birthday spacings tests with certain families of random number generators», *Mathematics and Computers in Simulation*, vol. 55, no. 1–3, 2001, pp. 131–137.

[32] L'Ecuyer, P., R. Simard et S. Wegenkittl, «Sparse serial tests of uniformity for random number generators», *Manuscript*, 1998.

[33] L'Ecuyer, P. et R. Touzin, «Fast combined multiple generators with multipliers of the form $\pm 2^p \pm 2^r$», *Proceedings of the 2000 Winter Simulation Conference*, 2000, pp. 683–690.

[34] Lehmann, E. L., *Testing Statistical Hypotheses*, 2 éd. Wiley Series in Probability and Mathematical Statistics, 1986.

[35] Marsaglia, G. «Diehard : A battery of tests of randomness». Voir `http ://stat.fsu.edu/~geo/diehard.html`, 1996.

[36] Maurer, U. M., «A universal statistical test for random bit generators», *Journal of Cryptology*, vol. 5, no. 2, 1992, pp. 89–105.

[37] McCabe, G. P. et D. S. Moore, *Introduction to the Practice of Statistics*, 2e éd. Freeman, 1996.

[38] NIEDERREITER, H., *Random Number Generation and Quasi-Monte Carlo Methods.* SIAM, Philadelphie, 1992.

[39] NIEDERREITER, H., «The multiple-recursive matrix method for pseudorandom number generation», *Finite Field and their applications*, vol. 1, 1995, pp. 3–30.

[40] NIEDERREITER, H., «Pseudorandom vector generation by the multiple-recursive matrix method», *Mathematics of Computation*, vol. 64, no. 209, 1995, pp. 279–294.

[41] PARK, S. K. et K. W. MILLER, «Random number generators : Good ones are hard to find», *Communications of the ACM*, vol. 31, no. 10, 1988, pp. 1192–1201.

[42] ROSEN, K. H., *Discrete Mathematics and its Applications*, 4e éd. McGraw-Hill, 1999.

[43] ROSS, S. M., *Initiation aux probabilités*, 3e éd. Presses polytechniques et universitaires romandes, 1988.

[44] WU, P.-C., «Multiplicative, congruential random-number generators with multiplier $\pm 2^{k1} \pm 2^{k2}$ and modulus $2^p - 1$», *ACM Transactions on Mathematical Software*, vol. 23, no. 2, 1997, pp. 255–265.

[45] WU, P.-C., «Multiplication-free multiple recursive random number generators», *Manuscript*, 1999.

Annexe I

```
long  D0, D1, D2, D3;

#define norm 4.6566128752e-10
#define m1   2147483647.0
#define B        22093.0
#define A        97201.0
#define Reste    21954.0

double DL00a ()
{
   long h, Resultat ;

   h = D0 / A; Resultat = B * (D0 - A * h) - Reste * h;
   if (Resultat < 0) Resultat += m1;

   Resultat -= D0;
   if (Resultat < 0) Resultat += m1;

   D3 = D2; D2 = D1; D1 = D0; D0 = Resultat;
   return (D0 * norm);

}
```

Tableau 10.1: Programme du générateur DL00a

```
long  z10, z11, z12, z20, z21, z22;

int M1  = 2147483647,
    M2  = 2145483479,
    A12 = 63308,
    Q12 = 33921,
    R12 = 12979,
    A13 = -183326,
    Q13 = 11714,
    R13 = 2883,
    A21 = 86098,
    Q21 = 24919,
    R21 = 7417,
    A23 = -539608,
    Q23 = 3976,
    R23 = 2071;

double  InvM = 4.656612873077393e-10;

double GRM96a ()
{
   long h, p12, p13, p21, p23;

   /* Composante 1 */
   h = z10 / Q13;    p13 = -A13 * (z10 - h * Q13) - h * R13;
   h = z11 / Q12;    p12 = A12 * (z11 - h * Q12) - h * R12;
   if(p13 < 0)  p13 += M1;  if(p12 < 0)  p12 += M1;
   z10 = z11;  z11 = z12;  z12 = p12 - p13;  if(z12 < 0) z12 += M1;

   /* Composante 2 */
   h = z20 / Q23;    p23 = -A23 * (z20 - h * Q23) - h * R23;
   h = z22 / Q21;    p21 = A21 * (z22 - h * Q21) - h * R21;
   if(p23 < 0)  p23 += M2;  if(p21 < 0)  p21 += M2;
   z20 = z21;  z21 = z22;  z22 = p21 - p23;  if(z22 < 0) z22 += M2;

   /* Combinaison */
   if (z12<z22) h = z12 - z22 + M1;
   else h = z12 - z22;
   return (h * InvM);

}
```

Tableau 10.2: Programme du générateur GRM96a

```
long  y10, y11, y12, y20, y21, y22;

#define m1     2147483647
#define m2     2145483479
#define a12    63308
#define q12    33921
#define r12    12979
#define a13   -183326
#define q13    11714
#define r13    2883
#define a21    86098
#define q21    24919
#define r21    7417
#define a23   -539608
#define q23    3976
#define r23    2071
#define InvM 4.656612873077393e-10

double GRM96b ()
{
   long h, p12, p13, p21, p23;

   /* Composante 1 */
   h = y10 / q13;    p13 = -a13 * (y10 - h * q13) - h * r13;
   h = y11 / q12;    p12 = a12 * (y11 - h * q12) - h * r12;
   if(p13 < 0)  p13 += m1;  if(p12 < 0)  p12 += m1;
   y10 = y11;  y11 = y12;  y12 = p12 - p13;  if(y12 < 0) y12 += m1;

   /* Composante 2 */
   h = y20 / q23;    p23 = -a23 * (y20 - h * q23) - h * r23;
   h = y22 / q21;    p21 = a21 * (y22 - h * q21) - h * r21;
   if(p23 < 0)  p23 += m2;  if(p21 < 0)  p21 += m2;
   y20 = y21;  y21 = y22;  y22 = p21 - p23;  if(y22 < 0) y22 += m2;

   /* Combinaison */
   if (y12<y22) h = y12 - y22 + m1;
   else h = y12 - y22;
   return (h * InvM);

}
```

Tableau 10.3: Programme du générateur GRM96b

```
double  s10, s11, s12, s20, s21, s22;

#define norm  2.328306549295728e-10
#define m1   4294967087.0
#define m2   4294944443.0
#define a12     1403580.0
#define a13n     810728.0
#define a21      527612.0
#define a23n    1370589.0

double GRMPFa ()
{
   register long   k;
   register double p1, p2;

   /* Composante 1 */
   p1 = a12 * s11 - a13n * s10;
   k = p1 / m1;   p1 -= k * m1;   if (p1 < 0.0) p1 += m1;
   s10 = s11;   s11 = s12;   s12 = p1;

   /* Composante 2 */
   p2 = a21 * s22 - a23n * s20;
   k  = p2 / m2;  p2 -= k * m2;   if (p2 < 0.0) p2 += m2;
   s20 = s21;   s21 = s22;   s22 = p2;

   /* Combinaison */
   if (p1 <= p2) return ((p1 - p2 + m1) * norm);
   else return ((p1 - p2) * norm);

}
```

Tableau 10.4: Programme du générateur GRMPFa

```
unsigned long  v1, v2, v3, v4, v5;

#define mask1a  127
#define mask3a  8191
#define mask4a  134217727
#define mask5a  1048575
#define m1      2147483647
#define norm    4.656612873077393e-10

double MRG00a()
{
   register unsigned long Resultat;

   /* Coefficient 1 */
   Resultat = m1 - (((v1 & mask1a) << 24) + (v1 >> 7));
   Resultat += v1;
   if (Resultat >= m1) Resultat -= m1;
   Resultat += v1;
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 3 */
   Resultat += (m1 - (((v3 & mask3a) << 18) + (v3 >> 13)));
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 4 */
   Resultat += (m1 - (((v4 & mask4a) << 4) + (v4 >> 27)));
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 5 */
   Resultat += (((v5 & mask5a) << 11) + (v5 >> 20));
   if (Resultat >= m1) Resultat -= m1;
   Resultat = Resultat + (m1- v5);
   if (Resultat >= m1) Resultat -= m1;

   v5 = v4; v4 = v3; v3 = v2; v2 = v1;

   v1 = Resultat;
   return (norm * v1);

}
```

Tableau 10.5: Programme du générateur GRM00a

```
unsigned long  w1, w2, w3, w4, w5, w6;

#define mask1a  1023
#define mask2a  524287
#define mask3a  32767
#define mask5a  16777215
#define mask6a  15
#define m1      2147483647
#define norm    4.656612873077393e-10

double MRG00b()
{
   register unsigned long Resultat;

   /* Coefficient 1 */
   Resultat = m1 - (((w1 & mask1a) << 21) + (w1 >> 10));
   Resultat += (m1-w1);
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 2 */
   Resultat += (m1 - (((w2 & mask2a) << 12) + (w2 >> 19)));
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 3 */
   Resultat += (((w3 & mask3a) << 16) + (w3 >> 15));
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 5 */
   Resultat += (((w5 & mask5a) << 7) + (w5 >> 24));
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 6 */
   Resultat += (m1 - (((w6 & mask6a) << 27) + (w6 >> 4)));
   if (Resultat >= m1) Resultat -= m1;
   Resultat += w6;
   if (Resultat >= m1) Resultat -= m1;

   w6 = w5; w5 = w4; w4 = w3; w3 = w2; w2 = w1;

   w1 = Resultat;
   return (norm * w1);

}
```

Tableau 10.6: Programme du générateur GRM00b

```
unsigned long  Y1, Y2, Y3, Y4, Y5, Y6, Y7;

#define mask1a  524287
#define mask2a  2047
#define mask3a  131071
#define mask5a  63
#define mask6a  33554431
#define mask7a  134217727
#define m1      2147483629
#define norm    4.6566129143e-10

double MRG00c()
{
   register unsigned long Resultat1, Resultat2;

   /* Coefficient 1 */
   Resultat1 = (((Y1 & mask1a) << 12) + 19 * (Y1 >> 19));
   if (Resultat1 >= m1) Resultat1 = m1 - (Resultat1 - m1);
   else Resultat1 = m1 - Resultat1;

   /* Coefficient 2 */
   Resultat2 = (((Y2 & mask2a) << 20) + 19 *(Y2 >> 11));
   if (Resultat2 >= m1) Resultat2 = m1 - (Resultat2 - m1);
   else Resultat2 = m1 - Resultat2;
   Resultat2 += Resultat1;
   if (Resultat2 >= m1) Resultat2 -= m1;

   /* Coefficient 3 */
   Resultat1 = (((Y3 & mask3a) << 14) + 19 * (Y3 >> 17));
   if (Resultat1 >= m1) Resultat1 -= m1;
   Resultat2 += Resultat1;
   if (Resultat2 >= m1) Resultat2 -= m1;

   /* Coefficient 5 */
   Resultat1 = (((Y5 & mask5a) << 25) + 19 * (Y5 >> 6));
   if (Resultat1 >= m1) Resultat1 -= m1;
   Resultat2 += Resultat1;
   if (Resultat2 >= m1) Resultat2 -= m1;

   /* Coefficient 6 */
   Resultat1 = (((Y6 & mask6a) << 6) + 19 * (Y6 >> 25));
   if (Resultat1 >= m1) Resultat1 = m1 - (Resultat1 - m1);
   else Resultat1 = m1 - Resultat1;

   Resultat2 += Resultat1;
   if (Resultat2 >= m1) Resultat2 -= m1;
```

```
   /* Coefficient 7 */
   Resultat1 = (((Y7 & mask7a) << 4) + 19 * (Y7 >> 27));
   if (Resultat1 >= m1) Resultat1 -= m1;
   Resultat1 += Y7;
   if (Resultat1 >= m1) Resultat1 -= m1;

   Resultat2 += Resultat1;
   if (Resultat2 >= m1) Resultat2 -= m1;

   Y7 = Y6; Y6 = Y5; Y5 = Y4; Y4 = Y3; Y3 = Y2; Y2 = Y1;

   Y1 = Resultat2;
   return (norm * Y1);

}
```

Tableau 10.7: Programme du générateur GRM00c

```
unsigned long  Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8;

#define mask1a  134217727
#define mask3a  65535
#define mask4a  524287
#define mask5a  511
#define mask6a  4194303
#define mask7a  15
#define mask8a  8191
#define mask8b  1073741823
#define m1      2147483647
#define norm    4.656612873077393e-10

double MRG00d()
{
   register unsigned long Resultat;

   /* Coefficient 1 */
   Resultat = m1 - (((Z1 & mask1a) << 4) + (Z1 >> 27));

   /* Coefficient 3 */
   Resultat += (((Z3 & mask3a) << 15) + (Z3 >> 16));
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 4 */
   Resultat += (m1 - (((Z4 & mask4a) << 12) + (Z4 >> 19)));
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 5 */
   Resultat += (((Z5 & mask5a) << 22) + (Z5 >> 9));
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 6 */
   Resultat += (((Z6 & mask6a) << 9) + (Z6 >> 22));
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 7 */
   Resultat += (((Z7 & mask7a) << 27) + (Z7 >> 4));
   if (Resultat >= m1) Resultat -= m1;

   /* Coefficient 8 */
   Resultat += (((Z8 & mask8a) << 18) + (Z8 >> 13));
   if (Resultat >= m1) Resultat -= m1;
   Resultat += (m1- Z8);
   if (Resultat >= m1) Resultat -= m1;
   Resultat += (m1- Z8);
   if (Resultat >= m1) Resultat -= m1;
```

```
   Z8 = Z7; Z7 = Z6; Z6 = Z5; Z5 = Z4; Z4 = Z3; Z3 = Z2; Z2 = Z1;

   Z1 = Resultat;
   return (norm * Z1);

}
```

Tableau 10.8: Programme du générateur GRM00d

```
unsigned long  x10, x11, x12, x20, x21, x22;

#define mask11  511
#define mask12  16777215
#define mask20  65535
#define mask22  65535
#define m1      2147483647
#define m2      2147462579
#define norm    4.656612873077393e-10

double MRGComb3 ()
{
   register unsigned long Resultat1;
   register unsigned long Resultat2;

   /* Composante 1 */
   Resultat1 = (((x11 & mask11) << 22) + (x11 >> 9))
             + (((x12 & mask12) << 7)  + (x12 >> 24));
   if (Resultat1 >= m1) Resultat1 -= m1;
   Resultat1 += x12;
   if (Resultat1 >= m1) Resultat1 -= m1;
   x12 = x11; x11 = x10; x10 = Resultat1;

   /* Composante 2 */
   Resultat1 = ((x20 & mask20) << 15) + 21069 *(x20 >> 16);
   if (Resultat1 >= m2) Resultat1 -= m2;
   Resultat2 = ((x22 & mask22) << 15) + 21069 *(x22 >> 16);
   if (Resultat2 >= m2) Resultat2 -= m2;
   Resultat1 += Resultat2;
   if (Resultat1 >= m2) Resultat1 -= m2;
   Resultat1 += x22;
   if (Resultat1 >= m2) Resultat1 -= m2;
   x22 = x21; x21 = x20; x20 = Resultat1;

   /* Combinaison */
   if (x10<= x20) return ((x10 - x20 + m1) * norm);
   else return ((x10 - x20) * norm);

}
```

Tableau 10.9: Programme du générateur GRM00e

```
unsigned long  T10, T11, T12, T20, T21, T22;

#define mask11  131071
#define mask12  31
#define mask20  16383
#define mask22  1048575
#define m1      2147483647
#define m2      2147483629
#define norm    4.656612873077393e-10

double MRG00f ()
{
   register unsigned long Resultat1;
   register unsigned long Resultat2;

   /* Composante 1 */
   Resultat1 = (((T11 & mask11) << 14) + (T11 >> 17))
             + (m1-(((T12 & mask12) << 26)  + (T12 >> 5)));
   if (Resultat1 >= m1) Resultat1 -= m1;
   Resultat1 += T12;
   if (Resultat1 >= m1) Resultat1 -= m1;
   T12 = T11; T11 = T10; T10 = Resultat1;

   /* Composante 2 */
   Resultat1 = ((T20 & mask20) << 17) + (19 *(T20 >> 14));
   if (Resultat1 >= m2) Resultat1 -= m2;
   Resultat2 = ((T22 & mask22) << 11) + (19 *(T22 >> 20));
   if (Resultat2 >= m2) Resultat2 -= m2;
   Resultat1 += Resultat2;
   if (Resultat1 >= m2) Resultat1 -= m2;

   T22 = T21; T21 = T20; T20 = Resultat1;

   /* Combinaison */
   if (T10<= T20) return ((T10 - T20 + m1) * norm);
   else return ((T10 - T20) * norm);

}
```

Tableau 10.10: Programme du générateur GRM00f

```
unsigned long  S10, S11, S12, S20, S21, S22, S30, S31, S32;

#define mask10  1
#define mask12  4095
#define mask21  255
#define mask22  4095
#define mask30  1048575
#define mask31  4194303
#define m1      2147483647
#define m2      2147483629
#define m3      2147483587
#define norm    4.656612873077393e-10

double MRG00g ()
{
   register unsigned long Resultat1;
   register unsigned long Resultat2;

   /* Composante 1 */
   Resultat1 = (((S10 & mask10) << 30) + (S10 >> 1))
             + (((S12 & mask12) << 19) + (S12 >> 12));
   if (Resultat1 >= m1) Resultat1 -= m1;
   Resultat1 += (m1 - S12);
   if (Resultat1 >= m1) Resultat1 -= m1;
   S12 = S11; S11 = S10; S10 = Resultat1;

   /* Composante 2 */
   Resultat1 = ((S21 & mask21) << 23) + (19 *(S21 >> 8));
   if (Resultat1 >= m2) Resultat1 -= m2;
   Resultat2 = ((S22 & mask22) << 19) + (19 *(S22 >> 12));
   if (Resultat2 >= m2) Resultat2 -= m2;
   Resultat1 += Resultat2;
   if (Resultat1 >= m2) Resultat1 -= m2;
   S22 = S21; S21 = S20; S20 = Resultat1;

   /* Composante 3 */
   Resultat1 = ((S30 & mask30) << 11) + (61 *(S30 >> 20));
   if (Resultat1 >= m3) Resultat1 -= m3;
   Resultat2 = ((S31 & mask31) <<  9) + (61 *(S31 >> 22));
   if (Resultat2 >= m3) Resultat2 -= m3;
   Resultat1 += Resultat2;
   if (Resultat1 >= m3) Resultat1 -= m3;
   Resultat1 += S32;
   if (Resultat1 >= m3) Resultat1 -= m3;
   Resultat1 += S32;
   if (Resultat1 >= m3) Resultat1 -= m3;
   S32 = S31; S31 = S30; S30 = Resultat1;
```

```
   /* Combinaison */
   if (S10 + S30 <= S20) return ((S10 - S20 + S30 + m1)*norm);
   else if (S10 -S20 +S30 > m1) return ((S10 - S20 + S30 - m1)*norm);
   else return ((S10 - S20 + S30) * norm);

}
```

Tableau 10.11: Programme du générateur GRM00g

```
unsigned long  M10, M11, M12, M13, M20, M21, M22, M23;

#define mask11  262143
#define mask13  255
#define mask20  2097151
#define mask22  2047
#define mask23  16777215
#define m1      2147483647
#define m2      2147483629
#define norm    4.656612873077393e-10

double MRG00h ()
{
   register unsigned long Resultat1;
   register unsigned long Resultat2;

   /* Composante 1 */
   Resultat1 = (m1- M10) + (m1 - (((M11 & mask11) << 13) + (M11 >> 18)));
   if (Resultat1 >= m1) Resultat1 -= m1;
   Resultat1 += ((M13 & mask13) << 23) + (M13 >> 8);
   if (Resultat1 >= m1) Resultat1 -= m1;
   Resultat1 += M13;
   if (Resultat1 >= m1) Resultat1 -= m1;
   M13 = M12; M12 = M11; M11 = M10; M10 = Resultat1;

   /* Composante 2 */
   Resultat1 = ((M20 & mask20) << 10) + (19 *(M20 >> 21));
   if (Resultat1 >= m2) Resultat1 -= m2;
   Resultat2 = (((M22 & mask22) << 20) + (19 *(M22 >> 11)));
   if (Resultat2 >= m2) Resultat2 =  m2 - (Resultat2 - m2);
   else Resultat2 = m2 - Resultat2;
   if (Resultat2 >= m2) Resultat2 -= m2;
   Resultat1 += Resultat2;
   if (Resultat1 >= m2) Resultat1 -= m2;
   Resultat2 = ((M23 & mask23) << 7 ) + (19 * (M23 >> 24));
   if (Resultat2 >= m2) Resultat2 -= m2;
   Resultat1 += Resultat2;
   if (Resultat1 >= m2) Resultat1 -= m2;
   M23 = M22; M22 = M21; M21 = M20; M20 = Resultat1;

   /* Combinaison */
   if (M10<= M20) return ((M10 - M20 + m1) * norm);
   else return ((M10 - M20) * norm);
}
```

Tableau 10.12: Programme du générateur GRM00h