DUTANG Christophe
MARCHAL Thibault
N'GUYEN Tan

25/04//08

Mersenne Twister et Translation irrationnelle du tore : comparaison et mise en oeuvre pratique

Sommaire :

Introduction

Dans le monde de la banque et des assurances, depuis ces dernières années, la gestion du risque prend une importance considérable. En effet, l'environnement réglementaire est profondément bouleversé. Les normes comptables internationales IAS/IFRS (en 2005), dont le but est de rendre transparente la situation financière des entreprises, incitent à évaluer l'ensemble de l'actif et du passif à leur juste valeur, à savoir à sa valeur de marché (et non plus selon la valeur historique amortie). Les directives Bâle II (pour les banques) et Solvabilité II (pour les sociétés d'assurances) imposent également un certain nombre de contraintes afin de contrôler le risque de faillite. De plus, le marché des actifs dérivés est en plein développement. On assiste à une multiplication des produits dérivés (produits hybrides, dérivés climatiques, …) et des sous-jacents (actions, inflation, températures, …) et à une intensification du volume des transactions. *(sources [8] dans les références)*

Par conséquent, il est primordial pour les assurances et les banques d'évaluer le risque avec précision. Pour ce faire, ces sociétés doivent être capables de prendre à compte toutes les évolutions possibles de la valeur de leurs éléments d'actif, passif, et des sous-jacents risqués à l'aide d'outils de simulation performants. L'ordinateur doit donc être capable de créer du hasard. De nombreuses sources de hasard, dont le but est de générer des nombres aléatoirement, ont ainsi été implémentées sur ordinateur. Néanmoins, comme nous allons le voir, ces sources d'aléa sont plus ou moins performantes.

Notre travail va ici consister à comparer la performance de deux générateurs de nombres aléatoires connus : l'algorithme du Tore et le générateur Mersenne Twister (MT). Ces deux générateurs sont réputés de par leurs qualités. Ils surpassent de nombreux générateurs, tels que le générateur Rnd d'Excel. Un exemple qui illustre bien cette supériorité est le calcul de $E(\exp(\sigma X))$, où σ un réel grand, et X une loi uniforme sur [0,1]. On fixe σ =100. Alors que la valeur théorique de cette espérance est $\frac{1}{\sigma}(\exp\sigma - 1) \approx 2.688117E+41$, on obtient après 1 000 000 simulations de chaque générateur, les valeurs suivantes :

Rnd	Tore	MT
2.71177E+41	2.68775E+41	2.67896E+41

Nous ferons dans une première parie la présentation théorique de ces générateurs, avec les avantages et inconvénients respectifs de leurs familles. Puis, nous comparerons les caractéristiques des deux générateurs à travers différents tests. Et enfin, nous passerons aux applications, afin d'apprécier plus concrètement la qualité des deux générateurs, et verrons lequel des deux générateurs est le plus approprié suivant les différentes simulations.

1. Présentation théorique des générateurs

1.1 Introduction sur les générateurs de nombres aléatoires

1.1.1 Lesquels utiliser en simulation ?

Un générateur de nombres aléatoires est un dispositif capable de produire une séquence de nombres, dont on peut considérer que les nombres sont obtenus au hasard.

La seule méthode connue aujourd'hui permettant d'obtenir des nombres véritablement aléatoires consiste à mesurer certains phénomènes physiques, tels que les émissions de particules d'une source radioactive. Cependant, cette méthode nécessite des moyens coûteux qui la rendent impropre à la simulation.

C'est pourquoi, dans les simulations, on a recours à des générateurs purement déterministes. Le but de ces générateurs est d'imiter le mieux possible le hasard, car les sorties d'un tel générateur ne sont pas aléatoires et s'approchent seulement des propriétés idéales des sources complètement aléatoires.

Nous verrons qu'il existe deux types d'algorithme générateur de nombres aléatoires de loi uniforme : les générateurs pseudo-aléatoires et les générateurs quasi-aléatoires.

1.1.2 Importance de ces générateurs

Les générateurs de lois uniformes vont servir à l'ensemble des simulations sur ordinateur. En effet, il est possible par inversion de la fonction de répartition de simuler n'importe quelle loi à partir de la loi uniforme. Concrètement, si $U \sim U[0;1]$, alors $X = F^{-1}(U)$ va suivre une loi de fonction de répartition F.
Lorsque la loi possède une fonction de répartition non inversible, d'autres procédés se servant de la loi uniforme existent. Par exemple, pour la loi normale, on utilisera les méthodes de Box-Muller, ou encore d'inversion de Moro *(détaillées dans la **section 2.2**)*.

Ainsi, ces générateurs sont à la base de la simulation, car ils permettent de simuler l'ensemble de nos modélisations faisant intervenir un caractère aléatoire.
Quelques exemples : l'évolution de cours ou de taux modélisés par des processus continus (faisant intervenir des mouvements browniens, et donc des lois normales), ou encore des décès parmi un portefeuille d'assurés (loi uniforme), ou bien le nombre d'accidents de voiture d'un assuré (souvent modélisé par des lois de Poisson composées).

Dans ce travail, nous allons nous intéresser particulièrement aux algorithmes de Mersenne Twister et du tore. MT et Tore n'appartiennent pas à la même famille de générateurs : il va donc être intéressant dans un premier temps de comparer les caractéristiques théoriques de leurs familles, avant de passer aux tests de comparaison et applications.

1.2 Mersenne Twister (MT19937), un générateur pseudo-aléatoire

Mersenne Twister appartient à la famille des générateurs pseudo-aléatoire. Nous allons voir dans cette partie quels inconvénients et avantages rencontrent les générateurs de cette famille, et en quoi Mersenne Twister est une exception, et pourquoi il est bien plus performant que la plupart des générateurs de sa famille. Nous nous intéresserons donc également dans cette partie aux générateurs congruentiels (la sous famille la plus connue des générateurs pseudo-aléatoires, dont MT ne fait pas partie) pour mieux voir en quoi MT est un générateur particulier et efficace.

1.2.1 Les générateurs pseudo-aléatoire

Les générateurs pseudo-aléatoires fabriquent des nombres U_k apparemment i.i.d de loi $U[0;1]$ selon le schéma récurent et déterministe de la forme suivante :
$U_k = g(s_k)$ avec $s_k = f(s_{k-1})$ et $s_0 \in S$, où $f : S \to S$ (appelée fonction de transfert) et $g : S \to [0;1]$ (fonction de sortie) sont déterministes.

La fonction f caractérise la dynamique du générateur. Chaque générateur pseudo-aléatoire a sa propre dynamique.

La fonction g, quant à elle, a pour but de convertir les nombres pseudo-aléatoires générés (qui appartiennent à l'ensemble S) en réels appartenant à $[0;1]$, et représentant ainsi les réalisations de la loi uniforme sur $[0;1]$. Ainsi, les générateurs pseudo-aléatoires peuvent avoir une fonction de sortie g construite de la même manière, seuls les paramètres variant.

Enfin, le choix de la graine s_0 (ou état initial) est important. En effet, les générateurs pseudo-aléatoires étant purement déterministes, lorsque la graine est fixée, on obtient invariablement la même séquence.
Dans le cas où on souhaite approximer une espérance ou tout autre indicateur statique (l'ordre de tirage des simulations n'ayant pas d'importance) et où l'état S n'est pas trop grand, on peut engendrer les graines successives avec un générateur pseudo-aléatoire auxiliaire pour obtenir l'ensemble des évolutions possibles du modèle.
Autrement, la plupart du temps, la graine doit être déterminée « au hasard » de la manière la plus aléatoire possible. La plupart du temps, les graines aléatoires sont déterminées à partir d'un état du système de l'ordinateur (tel que l'heure).

Comme l'espace des états S est fini, l'algorithme ne peut renvoyer qu'un nombre fini de valeurs distinctes, et comme la dynamique est déterministe, le générateur retrouve le même état interne au bout d'un certain nombre d'itérations. Ensuite, les mêmes séquences sont à nouveau générées. En d'autres termes, les générateurs pseudo-aléatoires sont périodiques.

1.2.2 Les générateurs congruentiels

Un générateur congruentiel est un générateur de nombres pseudo-aléatoires, dont l'algorithme, est basé sur un module.

Un générateur pseudo-aléatoire étant défini à partir de sa dynamique et de sa fonction de sortie, le générateur congruentiel est défini par les étapes ***(1), (2), (3)*** (ci-dessous).

- La dynamique d'un générateur congruentiel est définie par :
  - le choix d'une graine $X_0 \in N^*$ ***(1)***
  - le calcul récursif $X_{n+1} = (kX_n + p) \bmod m$ ***(2)***

  où k, m et p sont des entiers positifs, et m le module, et mod l'opérateur « reste de la division euclidienne de X_{n+1} / m ».

- On remarque que les valeurs générées X_k par la dynamique ci-dessus sont les m entiers naturels allant de 0 à m-1.
  Ainsi, un générateur congruentiel peut utiliser l'une des trois fonctions de sortie g suivantes :
  $U_k = g(X_k) = \frac{X_k}{m}$, $U_k = g(X_k) = \frac{X_k}{m-1}$ ou $U_k = g(X_k) = \frac{X_k + 0.5}{m}$ ***(3)***
  La plupart des générateurs congruentiels retiennent la 3ème approche, car les valeurs possibles pour U_k sont $\{1/(2m),...,1-1/(2m)\}$, qui est symétrique autour de ½.

- La famille congruentielle est composée des générateurs congruentiels linéaires (GCL), et des générateurs congruentiels multiplicatifs (GCM), dont les dynamiques sont définies comme suit (où a, k, m sont des entiers, p entier non nul)
  - GCL : $X_{n+1} = (kX_n + p) \bmod m$
  - GCM : $X_{n+1} = (kX_n) \bmod m$

  Dans ce type d'algorithme, la qualité du générateur va entièrement dépendre du choix de *k*, *p* et *m* car on ne peut pas se satisfaire d'un générateur qui produit des suites plus ou moins aléatoires, sans aucune raison apparente, selon le choix de X_0.

1.2.3 Le générateur Mersenne Twister MT19937

Les Mersenne Twister sont des générateurs récents, proposés pour la première fois par Matsumoto et Nishimura en 1998. L'idée originale des auteurs est de définir la récurrence du générateur, non pas à partir des opérations arithmétiques classiques sur les entiers (comme pour la plupart des générateurs courants), mais à partir des opérations d'arithmétique matricielle dans le corps fini $N_2 = \{0,1\}$.

1.2.3.1 Calculs préalables

Pour comprendre la dynamique de l'algorithme de MT, il faut préalablement acquérir un certain nombre de notions sur les opérations de bits, et poser un certain nombre de fonctions.

1. Tout d'abord, l'ensemble des entiers représentables en machine est de la forme N_{2^ω}, où ω désigne le nombre de bits de l'ordinateur. Tout entier $X \in N_{2^\omega}$, de décomposition binaire $\sum_{i=0}^{\omega-1} x_i 2^i$, est stocké sous la forme d'un « vecteur de bits » : $X = (x_{\omega-1},...,x_0)$.

2. Décalage de bits :
- décalage de v bits vers la droite : $X \gg v = (0,...,0,x_{\omega-1},x_{v+1})$
- décalage de v bits vers la gauche : $X \ll v = (x_{\omega-v-1},x_0,0,...,0)$

3. Opérateurs bit à bit :
Soit $X = \sum_{i<\omega} x_i 2^i$ et $Y = \sum_{i<\omega} y_i 2^i$. Les opérateurs bit à bit sont définis par :
$X \oplus Y = \sum_{i<\omega} (x_i \oplus y_i) 2^i$ et $X \otimes Y = \sum_{i<\omega} (x_i \otimes y_i) 2^i$
avec $x_i \oplus y_i = (x_i + y_i) \bmod 2$ et $x_i \otimes y_i = (x_i \times y_i) \bmod 2$

4. On pose $A(x) = (x \gg 1) \oplus 0$ si $x \bmod 2 = 0$ (x est pair)
$A(x) = (x \gg 1) \oplus a$ si $x \bmod 2 = 1$ (x est impair)

5. On pose $\underline{M}_r = (1,...,1,0,...,0)$ où les r premiers bits valent 1, les autres 0
$\overline{M}_r = (0,...,0,1,...,1)$ où les r derniers bits valent 1, les autres 0

1.2.3.2 Définition du générateur MT19937 (Mersenne Twister)

- La dynamique de Mersenne Twister est définie avec ces 2 étapes :

1. Opération de récurrence
$X_{k+n} = X_{k+m} \oplus A\left((X_{k+1} \otimes \underline{M}_r) \oplus (X_k \otimes \overline{M}_r)\right)$
La dynamique de MT est donc basée sur un schéma récurent d'ordre n dans l'ensemble des entiers machines. Pour $k \geq 0$ le terme X_{k+n} est construit à partir de X_k, X_{k+1} et X_{k+m} ($0 \leq m \leq n$).

2. Opération de tempering
Cette opération consiste à mélanger les bits de X_{k+n}, afin d'augmenter encore l'imprédictibilité des valeurs générées. X_{k+n} est transformé de la manière suivante :

$Y \leftarrow X_{k+n}$
$Y \leftarrow Y \oplus (Y \gg u)$
$Y \leftarrow Y \oplus ((Y \ll s) \otimes b)$
$Y \leftarrow Y \oplus ((Y \ll t) \otimes c)$
$Y \leftarrow Y \oplus (Y \gg l)$

- La fonction de sortie utilisée est donnée par $U_k = g(Y_k) = \frac{Y_k + 0.5}{2^\omega}$. U_k est bien à valeurs dans [0 ;1]

- Il y a au moins deux variantes connues de cet algorithme, différant seulement de par la longueur en bits des nombres (ce qui modifie la longueur de la période). Une variante avec des nombres de longueur 64-bits existe. Néanmoins, nous nous intéressons uniquement à la variante la plus récente et la plus communément utilisée : le générateur Mersenne Twister de Matsumoto et Nishimura, créé en 1998, se basant sur des nombres de longueur 32-bits. Ce générateur, appelé MT19937 génère les nombres U_k, construits comme précédemment, avec les paramètres :
  - <u>paramètres de récurrence :</u> $w = 32$, $n = 624$, $r = 31$, $m = 397$, $a = 2567483615$
  - <u>paramètres de tempering :</u> $u = 11$, $s = 7$, $t = 15$, $l = 18$, $b = 2636928640$, $c = 4022730752$

  Ce choix permet de maximiser la période, alors égale à $T_{MT} = 2^{w\,n-r} - 1 = 2^{19937} - 1$.

  La période étant de la forme $M_n = 2^n - 1$, il s'agit d'un Mersenne number. Plus précisément, comme $2^{19937} - 1$ est un nombre premier, la période est un Mersenne prime.

- Ainsi, Mersenne Twister est bien un générateur pseudo-aléatoire, et a une construction à peu près similaire aux générateurs congruentiels. En effet, on retrouve la récurrence du générateur basée sur une période. Néanmoins, la différence est que la dynamique de MT est construite non pas à partir d'opérations arithmétiques classiques sur les entiers, mais à partir d'opérations sur les bits.

1.2.4 Avantages, inconvénients

1.2.4.1 De la famille des générateurs congruentiels

Nous nous intéressons à la famille des générateurs congruentiels, car ils sont révélateurs des différents problèmes que peuvent rencontrer les familles des générateurs pseudo-aléatoires.

On considère qu'on travaille sur des générateurs congruentiels, où les paramètres k, p et m ont été correctement choisis. Ces générateurs sont pourvus de certains avantages par rapport aux autres générateurs, mais ont plusieurs inconvénients. En voici la liste.

AVANTAGES	INCONVENIENTS
1+ facilité d'implémentation	1- calcul modulo m
	2- limitation de la période
	3- problèmes d'équi-répartition multidimensionnelle

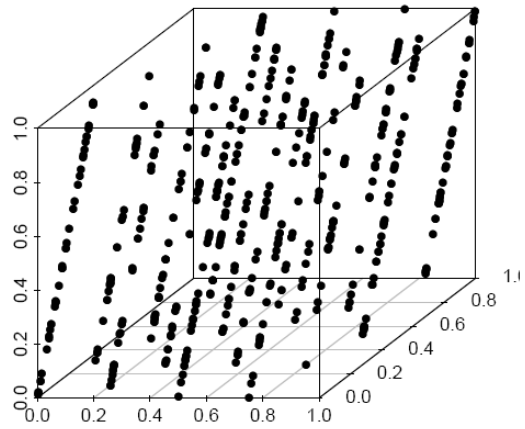
<u>1+ Facilité d'implémentation</u>
Il est très facile d'implémenter dans un programme un générateur congruentiel, de par la simplicité de la fonction de transfert et de la fonction de sortie.
Ainsi, les générateurs congruentiels peuvent être une bonne option dans un système sophistiqué, où la quantité de mémoire disponible est très gravement limitée ; de même pour un environnement tel que les jeux vidéo.

<u>1- Calcul modulo m</u>

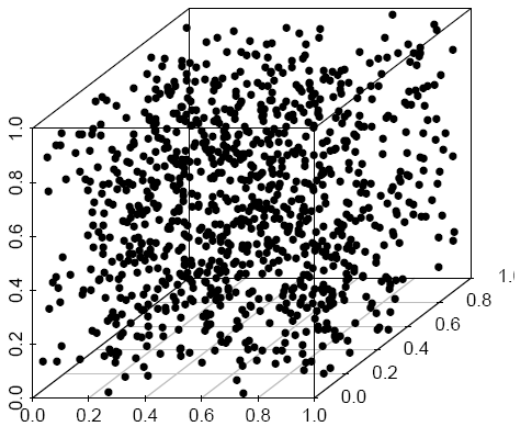
Les générateurs congruentiels font intervenir un calcul modulo m, et donc a priori une division euclidienne, ce qui peut avoir un coût de calcul important dans le cadre d'une utilisation fréquente du générateur.
*(Ce problème peut néanmoins être résolu : cf section **1.2.4.2, point 1**+)*

2- Limitation de la période :
Un générateur congruentiel a une période de longueur m. En effet, du fait de l'opération mod (reste dans la division euclidienne de X_{n+1} par m), les termes de cette suite sont compris entre 0 et m - 1. De plus, comme chaque terme dépend entièrement du précédent, si un nombre apparaît une deuxième fois, toute la suite se reproduit à partir de ce nombre. Or le nombre de valeurs que le nombre peut prendre étant fini (égal à m), la suite est amenée a se répéter au bout d'un certain temps. Donc, si m petit, la suite se répètera rapidement. Un bon générateur congruentiel devra donc avoir une période élevée pour résoudre ce problème.
*(Ce problème peut donc être résolu : cf section **1.2.4.2 point 2**+)*

3- Problèmes d'équi-répartition multidimensionnelle :
Ce problème est de loin le plus grave. Par ailleurs, il ne peut être résolu chez les générateurs congruentiels et chez la plupart des générateurs pseudo-aléatoires.
On remarque un sérieux problème en générant les points dans l'espace (pour tout n>=2). Ces points sont des vecteurs composés d'une suite de n valeurs générées. Si on se place dans l'espace de dimension n=3, on voit bien que les points sont situés sur des plans (*cf 1er graphique*). Or, en théorie, les points devraient occuper le cube au hasard (*cf 2ème graphique*).



1er graphique



2ème graphique

Cela implique qu'avec les générateurs congruentiels, les vecteurs de taille n qu'il est possible de simuler seront souvent similaires (dans l'ordonnancement de leurs composantes par exemple).

1.2.4.1 Comparaison avec MT19937

MT19937, le générateur Mersenne Twister de référence, corrige la plupart des inconvénients des générateurs pseudo-aléatoires. Nous prenons ici les générateurs congruentiels comme référence des générateurs pseudo-aléatoires. Néanmoins, il est pourvu de quelques inconvénients dont les générateurs congruentiels sont dépourvus.

PROBLEMES CORRIGES	INCONVENIENTS
1+ calcul modulo m	1- difficile à implémenter
2+ limitation de la période	2- initialisation de la graine ?
3+ problèmes d'équi-répartition dimensionnelle	

1+ calcul modulo m :
La période du générateur MT est du type $2^n - 1$, ce qui s'adapte parfaitement à l'architecture binaire de l'ordinateur *(cf section **1.2.3.1**, point **1.**)*. Ainsi, il n'y a pas besoin de faire intervenir de division euclidienne, ce qui fait économiser du temps.

2+ limitation de la période :
MT19937 possède une période de longueur colossale, valant $2^{19937} - 1 \approx 4.315425 \times 10^{6001}$.
Ainsi, le générateur ne pourra jamais retrouver le même état interne au bout d'un certain nombre d'itérations. En pratique, il y a peu de raison de trouver de plus grandes périodes, comme aucune des applications ne requière pas 2^{19937} combinaisons uniques.

3+ problème d'équi-répartition multidimensionnelle
Ce problème est extrêmement minimisé, de par deux interventions.
D'une part grâce à la récurrence du générateur à partir d'opérations sur des bits, et non pas sur des entiers, ce qui diminue la dépendance des termes dans la récurrence.
D'autre part, grâce à la procédure de tempering, qui fait un ultime mélange des bits avant d'envoyer un nouveau réel dans le segment d'unité. Ainsi, cette opération diminue encore le lien entre la k+m^{ème} génération de nombre aléatoire (i.e X_{k+m}), la k+n^{ème}, la k^{ème} , et la k+1 ^{ème}, rendant ainsi les suites de n nombres générées encore plus indépendantes.
Ces opérations permettent par conséquent une meilleure équi-répartition multidimensionnelle.
On verra dans la partie 2. que MT est équi-répartie en dimension 623 !

1- difficile à implémenter :
George Marsaglia a critiqué l'extrême complexité d'implémentation de MT dans les programmes informatiques. Il a notamment donné plusieurs exemples de générateurs de nombres aléatoires qui sont moins complexes, mais qui possèdent une période significativement plus grande. Par exemple, un simple « complimentary-multiply-with-carry » générateur possédant une période 10^{33000} plus grande peut être significativement plus rapide, et maintenir une uniformité équivalente ou meilleure.

2- graine de 624 chiffres ?
MT19937 possède une graine de 624 chiffres. Il est donc très sensible au choix de l'état initial. S'il contient trop de bits nuls, la suite générée conservera cette tendance sur plus de 10000 simulations.
Néanmoins, le problème a été remédié depuis 2002. Il a été proposé de construire la graine $(X_0, ..., X_{n-1})$ selon une récurrence qui assure une bonne diffusion des bits du registre :
$X_i = 1812433253 \times \left(\left(X_{i-1} \oplus X_{i-1} \gg 30 \right) + i \right)$ pour $i = 1, ..., n-1$
et $X_0 \in N_{2^w}$ est fixé arbitrairement.
Ainsi, les 624 graines sont déterminées dorénavant à partir d'une unique graine.

Les générateurs pseudo-aléatoires, à l'exception du Mersenne Twister, souffrent comme on l'a vu d'un certain nombre de défauts importants, le plus grave étant la faiblesse de l'équi-répartition dimensionnelle, ce qui limite leur utilisation pratique. Ainsi, on n'utilise que très peu ce type de générateurs pour les simulations.
On fait ainsi la plupart du temps appel à la deuxième grande famille de générateurs de nombres aléatoires : les générateurs quasi-aléatoire, dont l'algorithme du tore fait partie.

1.3 Algorithme du Tore, un générateur quasi-aléatoire

L'algorithme du tore appartient à la deuxième grande famille des générateurs de nombres aléatoires : les générateurs à discrépance faible, ou plus communément appelé, les générateurs quasi-aléatoires. Avant de définir l'algorithme du Tore, nous allons définir ce qu'est un générateur quasi-aléatoire, et donner quelques exemples de générateurs, afin de comprendre la logique du Tore. Puis nous étudierons les avantages et inconvénients souvent rencontrés dans cette famille, et comparerons avec le Tore.

1.2.1 Les générateurs quasi-aléatoires / à discrépance faible

Ces générateurs s'appuient sur la construction de suites à discrépance faible.

- Rappelons d'abord la définition de cette notion.

Définition 1 : *Soit* $x = (x_n)_{n\geq 1}$ *une suite de points de* $[0;1]^n$. *Soient* λ_n *la mesure de Lebesgue sur* $[0;1]^n$ *et A un sous pavé quelconque de* $[0;1]^n$. *On appelle* ***discrépance locale d'ordre k de x par rapport à A*** *la quantité :*

$$D_k(A,x) = \frac{1}{k} Card\{ i \in \{1,...,k\} \; tq. x_i \in A \} - \lambda_n(A)$$

Définition 2 : *Soit* $x = (x_n)_{n\geq 1}$ *une suite de points de* $[0;1]^n$. *Soient* λ_n *la mesure de Lebesgue sur* $[0;1]^n$, *A un sous pavé quelconque de* $[0;1]^n$, *et P l'ensemble des sous-pavés de* $[0;1]^n$. *On appelle* ***discrépance d'ordre k de x par rapport à A*** *la quantité :*

$$D_k^{\infty}(x) = Sup\{ |D_k(A,x)| tq. A \in P \} - \lambda_n(A)$$

La discrépance est donc un outil pour pouvoir juger de l'équi-répartition multidimensionnelle d'une suite : plus la discrépance est faible, mieux celle-ci est répartie sur son intervalle de définition. Si la discrépance est nulle, alors la suite est équi-répartie.

Définition 3 : *Soit* $x = (x_n)_{n\geq 1}$ *une suite de points de* $[0;1]^n$ *et P tel que défini ci-dessus. La suite x est dite* ***équirépartie****, ou uniformément distribuée, si et seulement si pour tout pavé A de P, la limite quand* $k \to +\infty$ *de la discrépance locale d'ordre k selon A est égale à 0.*

Définition 4 : *Soit* $x = (x_n)_{n\geq 1}$ *une suite de points de* $[0;1]^n$.

On dit que x est à ***discrépance faible*** *si :* $D_k^{\infty}(x) = O_k \to \infty\left(\frac{(\ln k)^n}{k}\right)$

Ainsi, les générateurs quasi-aléatoires sont des suites à discrépance faible, vérifiant $D_k^{\infty}(x) = O_k \to \infty\left(\frac{(\ln k)^n}{k}\right)$.

1.2.2 Quelques exemples de générateurs quasi-aléatoires

Les générateurs quasi-aléatoires (ou suites à discrépance faible) les plus connues sont les suites de Van Der Corput, de Halton, de Fauré, et de Sobol, ainsi que l'algorithme du tore.

Les suites de Halton, Fauré, et Sobol découlent toutes des suites de Van Der Corput. En effet, pour obtenir ces suites, on a besoin de convertir des entiers naturels dans une base b (fixe ou variable), puis réaliser l'étape d'inversion décrite ci-dessous. L'algorithme du Tore est un cas à part. Dans cette partie, nous décrirons donc seulement comment est défini une suite de Van Der Corput, afin de donner une idée quant à la construction des générateurs quasi-aléatoires.

Une **suite** $(X_n)_{n\leq k}$ **de Van Der Corput de base b** , où b est un nombre entier, est obtenue par :

- conversiondes entiers naturels n jusqu'à k (nombre de valeurs désirées), dans la base b : $n = \sum_{i=0}^{m} a_i \times b^i$ (m le plus petit entier tel que les a_i sont nuls $\forall\, i > m$)
- inversion par rapport à la virgule décimale des nombres obtenus en base b et calcul en base décimale. Le terme généré X_n est ainsi $X_n = \sum_{i=0}^{m} a_i \times b^{-i-1}$

1.2.3 La translation irrationnelle du Tore

La translation irrationnelle du Tore, plus communément appelé algorithme du Tore, se distingue des algorithmes précédents (section **1.2.2**). En effet, sa construction ne se base pas sur la conversion de nombres en base b, mais sur les nombres premiers.

Ce générateur multidimensionnel donne à la n^{ème} réalisation de la d^{ème} variable aléatoire uniforme à simuler la valeur u_n, définie par :

$u_n = n\sqrt{p_d} - \lfloor n\sqrt{p_d} \rfloor$,

où p_d est le d^{ème} nombre premier, et $\lfloor . \rfloor$ désigne l'opérateur partie entière.

1.2.4 Avantages, inconvénients

Nous nous intéressons ici aux avantages et inconvénients pouvant être rencontrés par la famille quasi-aléatoire, et voyons à chaque fois ce qu'il en est pour l'algorithme du Tore.

Inconvénient : souvent, algorithmes complexes

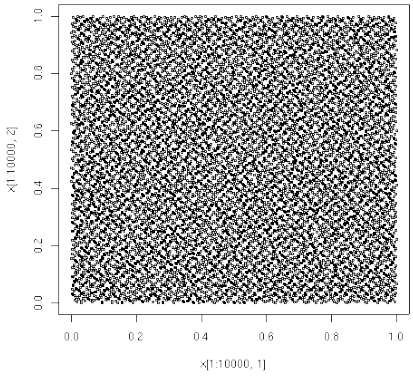
La majorité des générateurs quasi-aléatoires ont une construction complexe, et nécessite de mettre en œuvre des algorithmes relativement lourds : c'est ainsi le cas des suites de Van Der Corput, de Halton, de Fauré, et de Sobol. Outre la complexité de la programmation qu'elle induit, cette complexité pénalise les temps de calcul, qui peuvent être longs.

L'algorithme du Tore est une exception à la règle : en effet, comme on l'a vu, sa construction est d'une très grande simplicité *(cf section **1.2.3**)* : les étapes de calcul sont peu nombreuses, et les calculs sont assez basiques.

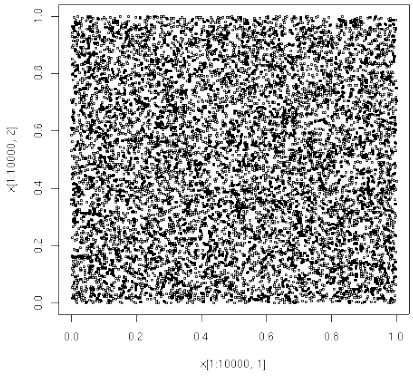
Avantage : bonnes propriétés d'équirépartition

Les générateurs quasi-aléatoires sont par définition de discrépance faible. Leur discrépance est bien meilleure que celle des suites pseudo-aléatoires (à l'exception de Mersenne Twister). Ainsi, ces algorithmes produisent des suites qui sont bien meilleures que les suites de nombres pseudo-aléatoires : leur équirépartition en dimension n ($n \geq 2$) est donc meilleure.

Les 2 graphiques ci-dessous montrent la répartition de points en dimension 2 dans un plan $[0;1]^2$, où les points sont des vecteurs composés d'une suite de 2 valeurs générées. Le 1er graphique rassemble les 10000 points générés par un générateur quasi-aléa : les points se répartissent quasi-uniformément. Le 2ème ceux générés par un pseudo-aléa : on aperçoit clairement des régions à faible et haute densité de points.



1er graphique



2ème graphique

Ainsi, ils remplacent les algorithmes pseudo-aléatoires utilisés traditionnellement dans les simulations de Monte Carlo, et sont ainsi à la base des simulations de Quasi-Monte Carlo. Grâce à leur bonne propriété d'équirépartition, ce type d'algorithme est particulièrement utilisé pour le calcul numérique d'intégrales (et donc d'espérance par exemple) (*cf **section 3.1***).

Nous avons vu dans la partie précédente que les générateurs quasi-aléatoires, dont l'algorithme du Tore, sont meilleurs que la plupart des générateurs pseudo-aléatoires. Mersenne Twister, un générateur pseudo-aléatoire, semble faire exception à ce constat. C'est pourquoi nous allons dans le reste de notre travail comparer la qualité des générateurs du Tore et Mersenne Twister, afin de voir lequel est le meilleur suivant les différentes simulations.

Nous avons choisi R comme logiciel de programmation.
Nous avons créé le package « torus », contenant les algorithmes du tore, tore mélangé, et le Rnd d'Excel. Il est disponible sur Internet.

2. Aspects pratiques

2.1 Caractéristiques algorithmes

De nombreux tests vérifiant les propriétés statistiques de séquences générées, et donc la qualité des générateurs, existent déjà. Les jeux de tests les plus connus sont Diehard, TestU01, ou encore Kiveliovitch et Vialar. Ceux-ci vérifient les propriétés d'uniformité, d'équidistribution, d'indépendance, et d'imprédictibilité. A noter que Mersenne Twister passe avec succès la plupart de ces tests, incluant les Diehard tests, mais pas certains des tests de TestU01 les plus rigoureux.

Dans cette partie, nous nous intéresserons qu'à quelques uns de ces tests, pour mettre en lumière les qualités (et les faiblesses) des algorithmes MT19937 et du tore.

2.1.1 Test d'équirépartition

On dit qu'un générateur est équi-réparti en dimension n, si les points qu'ils génèrent remplissent correctement l'espace de dimension n, sans qu'on puisse constater que certaines zones de l'espace aient moins de points que d'autres. Ici, un point $X = (x_1,...,x_n)$ en dimension n est un vecteur de n composantes x_i, où les x_i sont des simulations successives du générateur. Plus la dimension n est grande, plus il est difficile pour un générateur de remplir l'espace correctement. Plus le générateur est équi-réparti dans des dimensions élevés, meilleur il est.

Nous réaliserons d'abord un test basique en dimension 1, puis en réaliserons en dimensions n ($n \geq 2$).

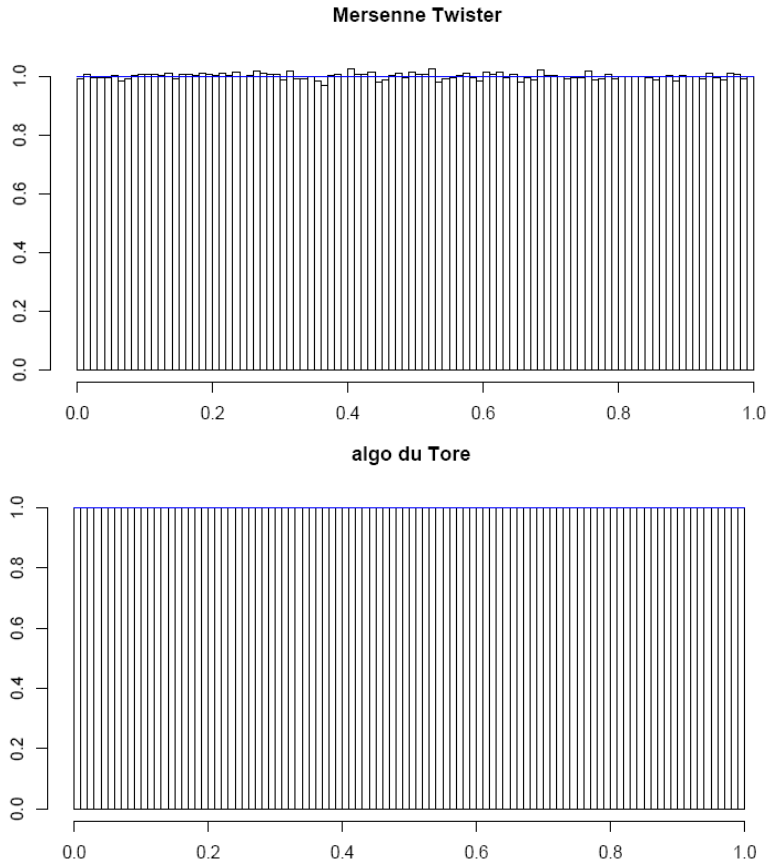
2.1.1.1 en dimension 1

Le test ci-dessous est assez basique, et assez peu révélateur de la qualité du générateur.
En effet, il suffit pour s'en convaincre de penser au générateur sortant les valeurs d'une suite $(x_n)_{n\geq 0}$ définie par : $x_n = (0.1 \times n) \bmod n \quad \forall\, n \geq 0$. Pour un nombre de simulations multiple de

10, si on coupe notre intervalle [0;1] en 10 intervalles égaux [0;0.1[, [0.1;0.2[, …, [0.9;1], les fréquences seront parfaites mais cette suite n'a rien d'aléatoire !
Par contre, ne pas réussir un tel test de manière satisfaisante serait problématique.

➢ **histogrammes :**

Après 1.000.000 de simulations, nous obtenons les graphiques suivants :



Les p-value des test du Chi2 (expliqué dans **annexe 1**) valent 1 pour les deux générateurs.

➢ **Conclusion :**
On ne peut pas départager l'algorithme du tore et MT19937 avec ce test.

2.1.1.2 en dimension n (n>1)

➢ **test de Marsaglia :**

- Principe :

Le test peut être généralisé en dimension n. Toutefois, en pratique, on ne peut l'appliquer que pour des dimensions relativement faibles, car en dimension n, il faut tenir compte de n! combinaisons.

En dimension 3, le test est :

On considère les 3-uplets (x, y, z) formés par 3 simulations successives de notre générateur. Nous dirons qu'un triplet est du type xyz lorsque $x < y < z$, et ainsi de suite pour les 3! = 6 façons dont peuvent s'ordonner 3 nombres distincts. Il est clair que les probabilités de ces divers ordonnancements sont exactement égales pour des simulations indépendantes d'une variable aléatoire uniformément répartie, et on peut donc s'attendre à ce que les fréquences de ces 6 ordres possibles soient à peu près égales (à 1/6).

Néanmoins, si le générateur n'est pas bien équi-réparti, alors certaines séquences arriveront plus fréquemment que d'autres, et on aura des proportions valant par exemple 1/12, 3/12, 2/12, 1/12, 3/12, 2/12.

- Résultats :

On applique ici le test en dimension n=3, ce qui implique qu'il faut tenir compte de 3! = 6 combinaisons. On a simulé pour chaque générateur 6000 points. On obtient :

	xyz	xzy	yxz	yzx	zxy	zyx	
tore	998	999	1003	1004	993	1003	ecart : 0.0146667
MT19937	975	992	1024	1010	986	1013	ecart : 0.2883333

Pour chaque triplet, le nombre empirique se rapproche du nombre théorique (=1000).

La fonction $ecart$ permet ici de comparer la qualité du tore et de MT.
On a posé la fonction $ecart = \frac{1}{6}\sum_{i=1}^{6} (f_{th} - f_{emp})^2$, f_{emp} étant la fréquence empirique obtenue avec notre générateur, et f_{th} la fréquence théorique (valant 1/6 $\forall i \in 1,...,6$).

➢ tracé de plans :

- Principe :

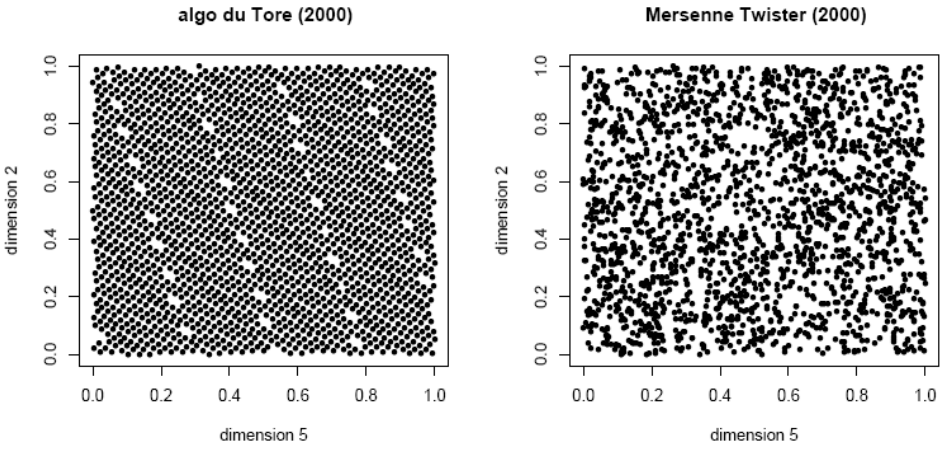
On prend un générateur donné. On génère M vecteurs X. Chaque vecteur est formé de n composantes, où n est la dimension de l'espace, et où les composantes sont en fait n simulations successives de notre générateur. On a donc $X \in [0,1]^n$

Ensuite, on choisit 2 composantes de X qu'on extrait : concrètement, on crée M autres vecteurs Y composés de 2 composantes x_i et x_j de notre vecteur X (x_i et x_j fixés tels que $1 \le x_i \le n, 1 \le x_j \le n$). On a donc $Y \in [0,1]^2$

Il suffit ensuite de tracer dans un plan $[0,1]^2$ les points Y. Si le générateur n'est pas bien équi-réparti en dimension n, alors, pour certaines composantes x_i et x_j, les M points Y du plan ne rempliront pas bien l'espace.

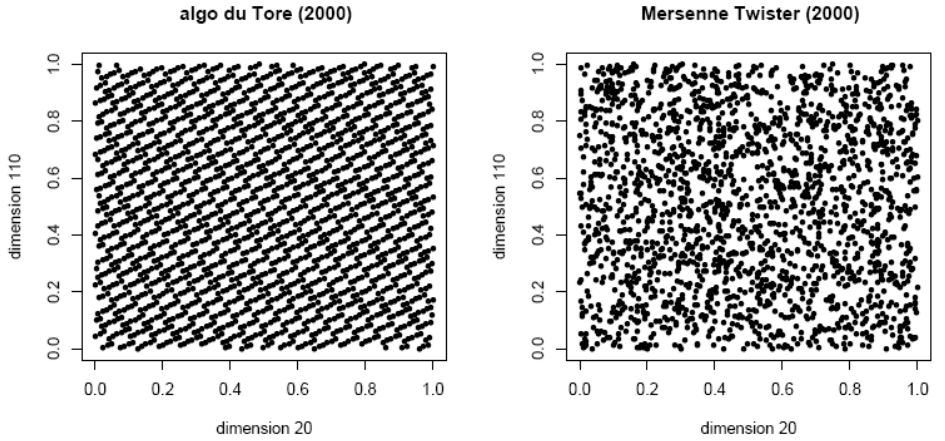
- Résultats :

Ici, on génère 6000 points de dimension 6, et on s'intéresse à x_i=2 et x_j=5 :



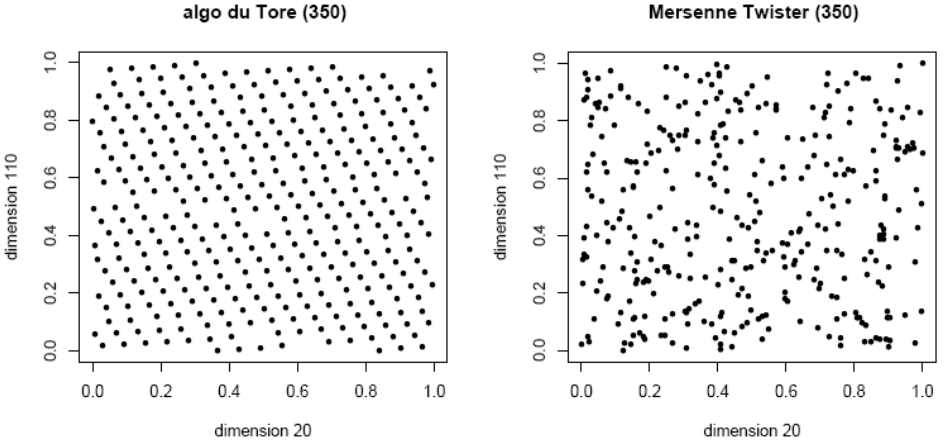
graphiques 1

Ici, on génère 6000 points de dimension 200, et on s'intéresse à x_i=20 et x_j=200 :



graphiques 2

Ici, on génère 200 points de dimension 350, et on s'intéresse à x_i=20 et x_j=200 :



graphiques 3

➢ **Conclusion :**

Avec beaucoup de simulations : cf graphiques 1 et 2

Les deux générateurs possèdent tous deux de très bonnes qualités d'équi-répartition, dans des dimensions petites et élevés *(200 par exemple)*, lorsque le nombre de simulations est grand *(ici 6000)*. En effet, le test de Marsaglia en dimension 3 est réussi pour les deux générateurs, et sur les graphiques ci-dessus, les points générés remplissent bien les plans, toutes les zones du plan ayant une densité de points comparables.

Avec peu de simulations : cf graphiques 3

Néanmoins, on se rend compte que l'algorithme du Tore est moins « aléatoire » que MT19937, et sa répartition dans l'espace est plus construite. Cela se voit mieux avec un faible nombre de simulations *(ici 350)*. En effet, le Tore remplit mieux l'espace que MT19937 sur les 3èmes graphiques, et le test de Marsaglia est plus concluant pour le tore. Par conséquent, le tore est mieux équi-réparti.

Remarques :

Pour le tore, ces résultats très satisfaisants s'expliquent de par sa nature, car il appartient à la famille des générateurs quasi-aléatoires : le tore a par définition une discrépance faible, et donc de bonnes propriétés d'équi-répartition.

Par contre, un constat plus intéressant est l'excellente équi-répartition de MT19937 dans des dimensions élevées. Cela est assez remarquable de la part d'un générateur pseudo-aléatoire, famille réputée pour ses problèmes d'équi-répartition multidimensionnelle. Dans des dimensions élevées (n=200), les points de Mersenne Twister continuent d'occuper « également » l'espace. En guise de remarque, il a été prouvé que Mersenne Twister est équi-réparti dans 623 dimensions.

2.1.2 Test d'indépendance / dépendance terme à terme

Les bonnes qualités d'équi-répartition des deux générateurs, se caractérisant par un bon remplissage de l'espace dans des dimensions assez élevées, permettent seulement de conclure à l'indépendance globale des générateurs. On pourrait en effet croire que la bonne équi-répartition implique que les composantes des points sont générées sans aucune dépendance les unes avec les autres, mais cela est faux.

Nous allons donc réaliser deux tests pour étudier la dépendance terme à terme des générateurs. Nous allons constater que le tore souffre d'une dépendance terme par terme, puis allons proposer une solution pour remédier à ce problème.

2.1.2.1 Les tests

➢ **Les corrélogrammes**

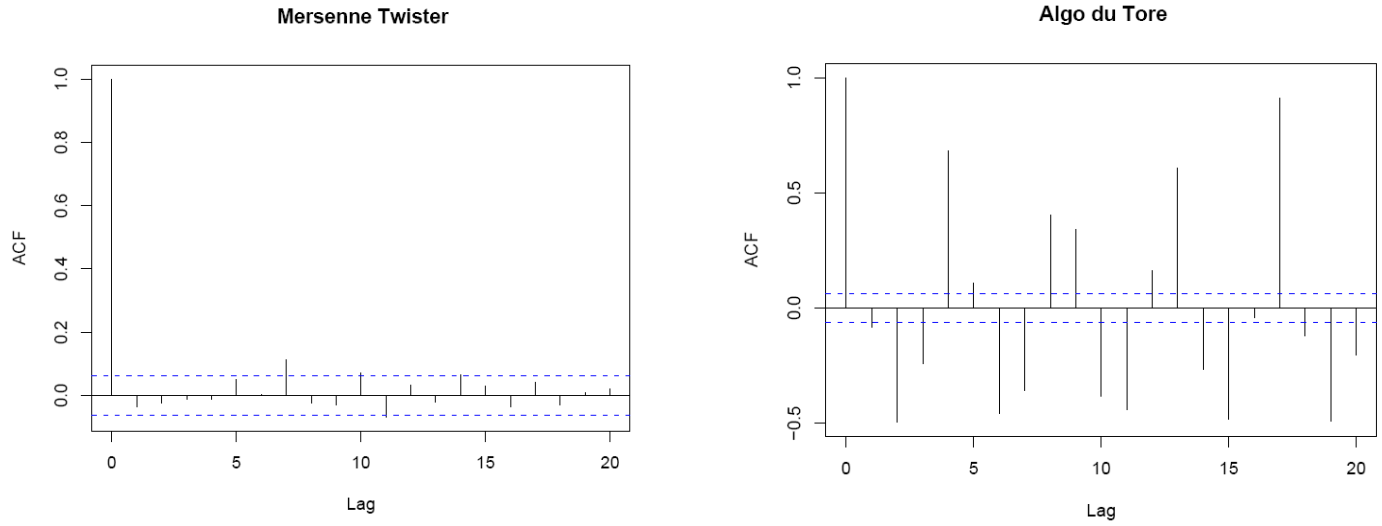
Soit $(u_n)_{n\geq 0}$ la suite générée par un générateur de nombres aléatoires. On génère N simulations avec notre générateur : on obtient donc une suite de longueur N.

Le h[ème] terme de notre corrélogramme s'écrit : $\rho_h = \dfrac{\sum_{k=1}^{n}(u_k - \bar{u})(u_{k+h} - \bar{u})}{\sum_{k=1}^{n}(u_k - \bar{u})^2}$

où $\bar{u}$ désigne la moyenne empirique de la suite $(u_k)_{k\geq 0}$.

La fonction d'auto-corrélation ρ_h mesure donc les corrélations moyennes associées à des termes de la suite séparés par un intervalle de temps égal à h. La corrélation est une mesure de dépendance entre deux termes. Donc, plus ρ_h est éloigné de 0, plus la dépendance moyenne entre les termes x_k et x_{k+h} est forte.

On obtient les graphiques suivants :



➢ Le gap test

- Principe

Le gap test permet également d'étudier la dépendance terme à terme des générateurs. L'idée de ce test est de comparer les fréquences théoriques de la longueur des écarts (=gaps) avec les fréquences observées sur les simulations effectuées.

De manière pratique, on génère N valeurs avec notre générateur. On obtient une suite $(u_k)_{0\leq k\leq N}$. Le gap test localise les longueurs des écarts entre les groupes de valeur se situant sur un intervalle donné. Prenons par exemple le cas d'une suite de valeurs u_j, u_{j+1}, …, u_{j+r} à chaque fois situées en dehors de notre intervalle, et u_{j-1} et u_{j+r+1} situés sur l'intervalle. Alors, cette suite est appelée un « gap de longueur r ».

Nous choisissons comme intervalle l'ensemble des valeurs comprises entre 0 et p. Donc, la probabilité théorique qu'un gap ait une longueur de r est $p(1-p)^r$. Ici, on fixe p=0,5. Ainsi, notre proportion attendue de gap de longueur 0 (respectivement 1, 2, 3, 4, …) est 0.5 (respectivement 0.5^2, 0.5^3, 0.5^4, 0.5^5…).

- Résultats

Après avoir fait 1.000.000 de simulations pour chaque générateur :

longueur des gaps	0	1	2	3	4	5	6	…	18
fréquence théorique	0.5	0.25	0.125	0.0625	0.03125	0.01563	0.00781		1.9E-06
Mersenne Twister	0.4995	0.24936	0.12563	0.06297	0.03114	0.01554	0.00793		2E-06
Tore	0.17157	0.65685	0.17157	-	-	-	-		-

➢ **Conclusion**

MT passe avec succès les tests des corrélogrammes et gap test : les nombres générés par MT sont donc quasiment indépendants les uns des autres.
Par contre, les résultats sont beaucoup moins satisfaisants pour le tore. Les ρ_h sont souvent éloignés de 0, et la fréquence des gaps qu'on trouve avec le tore sont très éloignés de la réalité. Le tore souffre donc d'une dépendance terme à terme.

Pour pallier à ce problème du tore, nous allons introduire le tore mélangé, défini ci-dessous, qui va avoir de bien meilleurs résultats.

2.1.2.2 Le tore mélangé

➢ **Définition :**

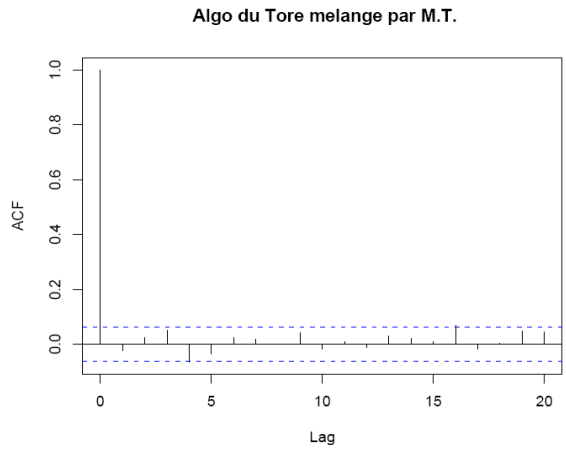
Le tore mélangé est une adaptation du tore qui consiste à mélanger les valeurs générées du tore avant de les utiliser.

Notons (u_n) la suite générée par le nombre premier p. Au lieu d'utiliser le nombre u_n lors du n^{ème} tirage de la loi uniforme sur [0;1], le tore mélangé utilise $u_{\varphi(n)}$ où $\varphi(n) = \lfloor \alpha \times N \times u + 1 \rfloor$.
$\alpha \geq 1$ (généralement $\alpha = 10$), u est la réalisation d'une variable aléatoire de loi uniforme, [.] désigne l'opérateur partie entière. Ainsi, $\varphi(n)$ à valeurs dans $1,...,\alpha N$.
Plus précisément, u correspond à la valeur générée par l'algorithme de notre choix.

Pour la génération u, nous avons retenu le générateur MT19937 (comme nous aurions pu retenir tout autre générateur pseudo-aléatoire avec une période importante).

➢ **Amélioration des résultats :**

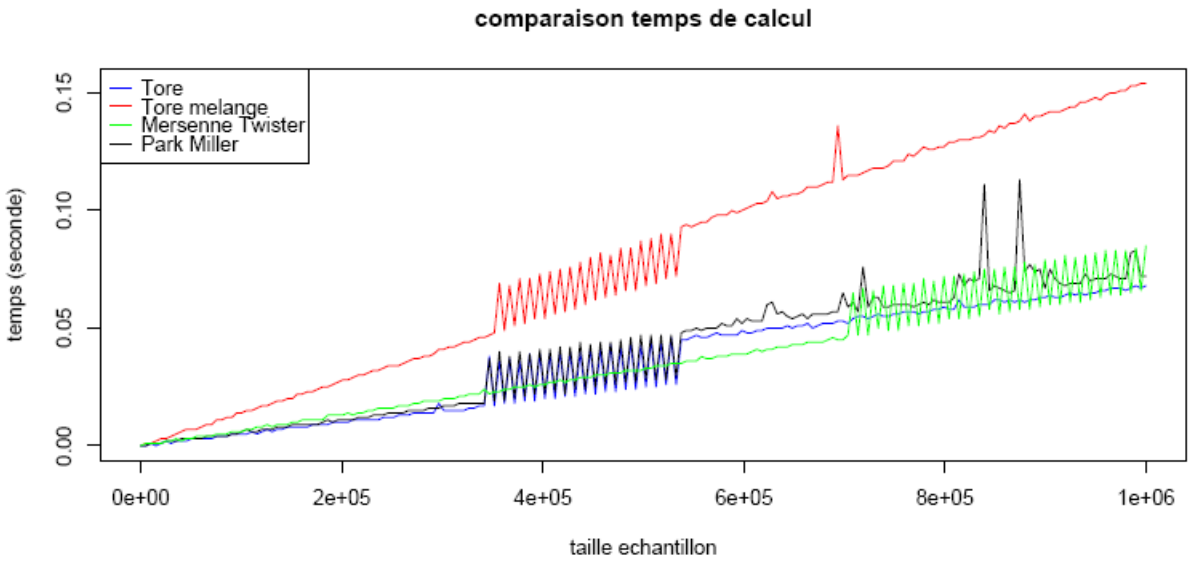
longueur des gaps	0	1	2	3	4	5	6	…	18
fréquence théorique	0.5	0.25	0.125	0.0625	0.03125	0.01563	0.00781		1.9E-06
Tore mélangé	0.50008	0.25094	0.12457	0.06193	0.03119	0.01569	0.0078		1.8E-06
Tore	0.17157	0.65685	0.17157	-	-	-	-		-



Les résultats sont concluants : le tore mélangé remédie effectivement au problème de dépendance terme à terme du tore.

2.1.3 Test de rapidité

Nous avons effectué les simulations sur un ordinateur ayant les caractéristiques suivantes : Macbook 2,2 GHz – 2 Go RAM – Mac OS X Leopard 64bit.



Ici, le tore mélangé est le tore mélangé par Mersenne Twister, et Park Miller est le Rnd d'Excel

On remarque que le Rnd d'Excel est à peu près aussi rapide que le tore et Mersenne Twister, mais il devient plus long lorsque le nombre de simulations est élevé. Le tore et MT ont des temps de simulation comparables. Le tore mélangé est systématiquement plus long que le tore.

<u>Remarque sur Tore et Tore mélangé:</u>

Comparé à la plupart des générateurs de nombres aléatoires, les calculs de l'algorithme du Tore sont peu nombreux et d'une très grande simplicité. Cet algorithme peut être résumé en

deux lignes ! Ainsi, la simplicité de ce générateur induit une programmation relativement légère, ce qui optimise les temps de calculs.

Le Tore mélangé est plus long à simuler que le tore car il nécessite une opération supplémentaire. C'est le seul inconvénient du tore mélangé par rapport au tore.

Remarque sur MT19937 :

Mersenne Twister a des calculs beaucoup plus complexes, et on pourrait donc croire que les temps de calculs en seraient gravement accrus.

Néanmoins, il n'en est rien, car la récurrence de ce générateur est basé à partir d'opérations sur les bits, et non d'entiers (à la différence du tore, et de la plupart des générateurs de nombres aléatoires). En effet, les opérateurs de bits sont très rapides, car l'ordinateur ne perd pas du temps à traduire les opérations arithmétiques en opérations de bits. Ainsi, le générateur exploite pleinement l'architecture binaire de l'ordinateur, ce qui réduit considérablement les temps de calculs.

2.2 Simulation de lois

Maintenant que nous avons étudié les caractéristiques du tore et de Mersenne Twister pour simuler la loi uniforme [0;1], nous allons voir comment ces générateurs se comportent pour simuler différentes lois : la loi exponentielle, et la loi normale. En effet, il est possible de simuler n'importe quelle loi à partir de la loi uniforme.

Le but de cette partie est de voir si un des deux générateurs est davantage efficace que l'autre pour simuler différentes lois.

2.2.1 La loi exponentielle, une loi à fonction de répartition inversible

Dans cette partie, nous avons choisi la loi exponentielle, comme nous aurions pu choisir toute autre loi à fonction de répartition (FdR) continue et inversible : par exemple, la loi de Pareto $P(a,\alpha)$, et la loi de Weibull $W(\tau,a)$. Les remarques de cette partie sont valables pour l'ensemble de ces lois.

- **Méthode de la transformée inverse**

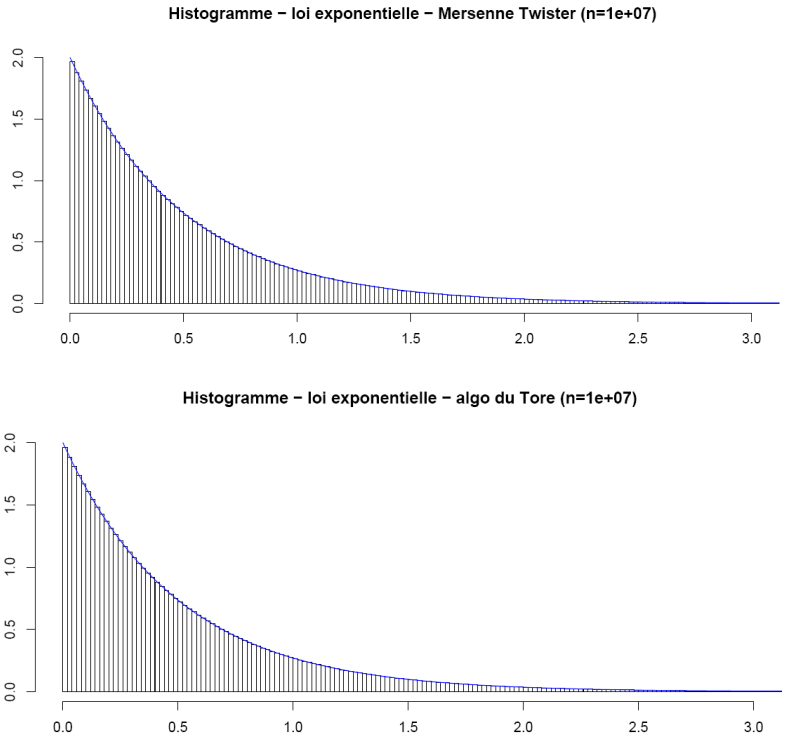
Dans le cas de distributions entières, cette méthode pourrait s'appliquer mais n'est pas performante, car trop coûteuse en temps et calcul. On ne la détaillera donc pas ici. Pour chacune des lois entières les plus connues, il existe des méthodes de simulation particulières.

Dans le cas de distributions continues, la méthode est fondée sur la propriété qu'a la variable aléatoire $U = F_X(x)$ d'être distribuée uniformément quelle que soit la forme de la fonction de répartition $F_X(x)$. Ainsi, si $U \sim U[0;1]$, alors $X = F^{-1}(U)$ va suivre une loi de fonction de

répartition F, donc il suffit de simuler n réalisations indépendantes de la loi uniforme sur [0;1], puis d'appliquer l'inverse de la fonction de répartition à chacune de ces valeurs. Néanmoins, cette méthode ne marche que pour les lois ayant des fonctions de répartition inversibles.

Dans cette méthode, il y a une relation arithmétique directe entre la valeur uniforme générée (notée U) et la valeur de la loi (ayant pour fonction de répartition F) générée (notée X). Par exemple, pour la loi exponentielle λ, de fonction de répartition $F(x) = (1 - e^{-\lambda x}).1_{x>0}$, on a $X = F^{-1}(U) = -\frac{1}{\theta}\ln(1-U)$.

- **Résultats :**
  la densité théorique est en bleu, les histogrammes à pas fixes estiment la densité à partir de nos simulations



- **Conclusion**

La valeur uniforme et la valeur normale générées étant liées par une simple relation arithmétique *(contrairement à la méthode de Box-Muller section **2.2.2.1**)*, il n'y a pas d'interaction négative entre le générateur de nombres aléatoires et la méthode d'inversion : si le générateur est de qualité pour la loi uniforme [0;1], alors il sera de qualité pour la loi exponentielle (et autres lois à FdR inversible).

Ainsi, la comparaison de la simulation de lois exponentielles (et autres lois ayant une fonction de répartition continue et inversible) n'apporte pas d'information supplémentaire pour comparer les qualités des générateurs tore et Mersenne Twister ; en effet, cette comparaison ne peut aboutir qu'aux mêmes conclusions du test de fréquence de la loi uniforme [0;1].

2.2.2 La loi normale, un cas particulier

On s'intéresse au cas particulier de la loi normale, car elle est extrêmement importante en assurances / banques. En effet, elle est nécessaire pour simuler les mouvements browniens, souvent utilisés pour modéliser l'évolution des taux et des cours, et est donc à la base de nombreuses simulations de trajectoires.

Le problème avec la loi normale est que sa fonction de répartition n'est pas inversible. Générer la loi normale est donc moins évident que la loi exponentielle. Nous allons étudier les différentes méthodes pour simuler la loi normale, et étudier leur compatibilité avec les générateurs Tore et Mersenne Twister.

2.2.2.1 Simulation via la méthode de Box-Muller

➢ **<u>présentation</u>**

La méthode de Box-Muller (George Edward Pelham Box et Mervin Edgar Muller, 1958) consiste à générer des paires de nombres aléatoires à distribution normale centrée réduite, à partir d'une source de nombres aléatoires de loi uniforme.

La transformation prend communément deux formes.
- La forme simple transforme des coordonnées cartésiennes uniformément distribuées dans le cercle unité en des coordonnées normalement distribuées.
- La forme polaire transforme des coordonnées polaires uniformément distribuées en des coordonnées cartésiennes normalement distribuées.

<u>Forme cartésienne :</u>
Soient x et y choisis indépendamment et uniformément dans $[-1,1]$, et $s = x^2 + y^2$.
Concrètement, on fait 2 simulations successives x et y avec notre générateur, puis on réalise l'opération : $x \leftarrow 2 \times x - 1$ et $y \leftarrow 2 \times y - 1$
Si s > 1, rejetons-le et choisissons à nouveau un couple (x, y), jusqu'à ce que s appartienne à]0,1]. Pour ces points "filtrés", il suffit ensuite de calculer :

$$z_0 = x.\sqrt{\frac{-2\ln s}{s}} \quad \text{et} \quad z_1 = y.\sqrt{\frac{-2\ln s}{s}}$$

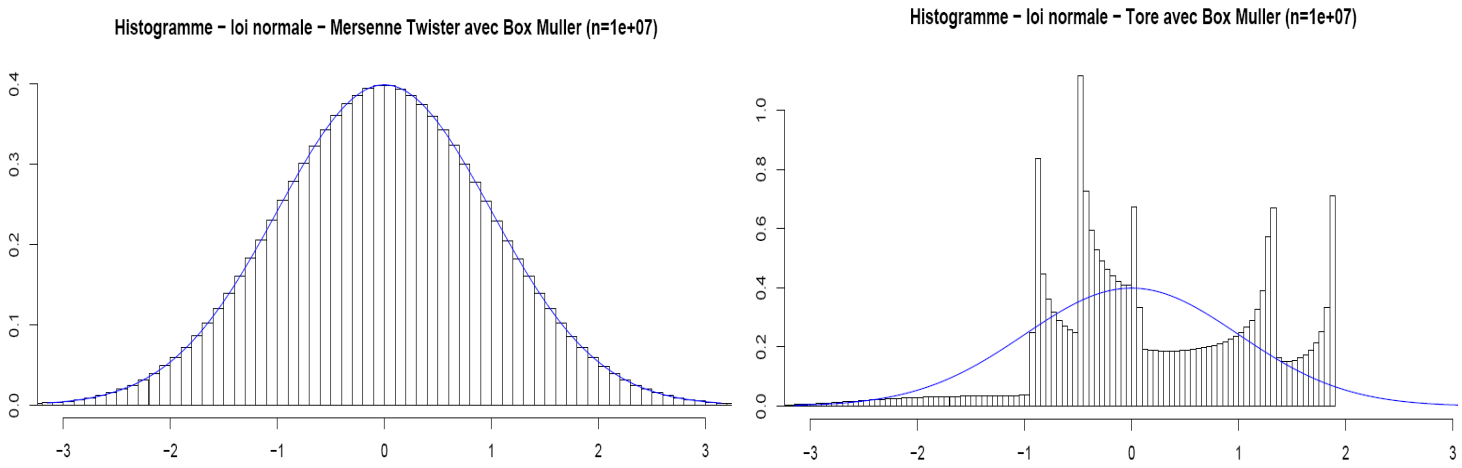
<u>Forme polaire :</u>
Soient U1 et U2 deux variables aléatoires indépendantes uniformément distribuées dans]0,1].
Soient $Z_0 = R\cos(\theta) = \sqrt{-2\ln U_1}\cos(2\pi U_2)$ et $Z_1 = R\sin(\theta) = \sqrt{-2\ln U_1}\sin(2\pi U_2)$
Alors Z_0 et Z_1 sont des variables aléatoires indépendantes suivant une loi $N(0,1)$
Cette transformation vient du fait que, dans un système cartésien à deux dimensions où les coordonnées X et Y suivent deux variables aléatoires indépendantes normales, les variables aléatoires R2 et Θ (ci-dessus) sont également indépendantes et peuvent s'écrire :
$R^2 = -2.\ln U_1$ et $\theta = 2\pi U_2$

La forme cartésienne est une méthode d'échantillonnage à rejet, qui n'utilise qu'une partie des nombres générés par la source aléatoire, mais elle est en pratique plus rapide que la forme polaire car elle est plus simple à calculer :

- la forme cartésienne n'utilise pas de fonctions trigonométriques, coûteuses en temps de calcul
- la génération de nombres aléatoires est plutôt rapide, il n'est donc pas gênant d'en gaspiller une partie. En moyenne la part de points rejetés est $(1-\pi/4) \approx 21.46\%$. On génère donc $4/\pi \approx 1.2732$ nombres aléatoires uniformes pour obtenir chaque nombre aléatoire normal.

➢ **<u>résultats (non satisfaisants)</u>**

- Au cœur de la distribution pour MT et tore



Les valeurs empiriques se confondent avec les valeurs théoriques pour MT19937.

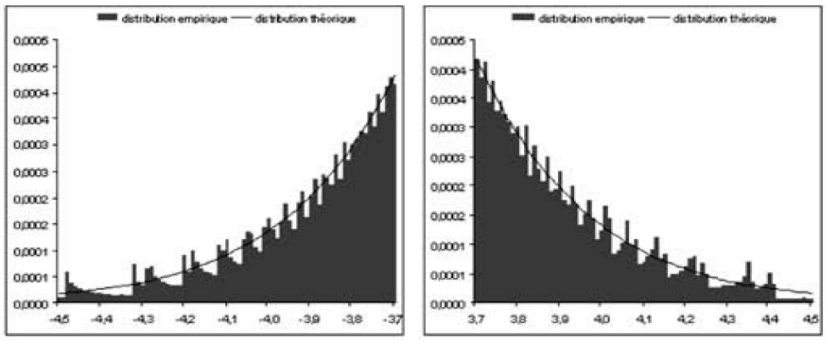
Par contre, la technique de Box-Muller est complètement incompatible avec le tore. Le problème vient de la dépendance terme à terme du tore. En effet, la méthode de Box-Muller implique la génération d'un couple de lois uniformes : donc les deux valeurs successives générées doivent être quasi-indépendantes, ce qui n'est pas le cas pour le Tore. Certes, le tore mélangé pourrait quant à lui être compatible avec Box-Muller, mais les calculs en seraient alourdis ; on préférera utiliser une méthode d'inversion, *cf section **2.2.2.2***.

- Aux valeurs extrêmes pour MT (effet Neave)

On souhaite voir ce qui se passe précisément aux valeurs extrêmes (par exemple, les valeurs situées avant le quantile 0.01% et après le quantile 0.99%).

Pour obtenir des graphiques précis à ces valeurs, il faudrait se servir de variables de contrôle permettant de simuler directement les valeurs extrêmes de nos générateurs MT et Tore. Cela permettrait de réduire considérablement les temps de calculs ; autrement, en réalisant par exemple 1.000.000 simulations, nous n'obtenons en moyenne que 200 valeurs extrêmes, ce qui est insuffisant.

Nous avons cependant choisi de faire figurer un graphique déjà publié, vu que la méthode de Box-Muller ne permet pas de départager les performances du tore et de Mersenne Twister, ces deux générateurs ayant des problèmes avec cette méthode *(cf conclusion section **2.2.2.3**)*.



On observe un échantillonnage irrégulier (en « dents de scie ») pour ces valeurs extrêmes. Cette déviation entre la distribution empirique et la distribution théorique ne peut pas être imputée aux générateurs, car ils passent les tests standards, ni à la longueur de l'échantillon, car sur le graphique, le nombre de simulations pour ces valeurs est suffisant. Ce phénomène est donc la conséquence d'une interaction non souhaitée entre l'algorithme de Box-Muller et le générateur.

2.2.2.2 Simulation via l'approximation de F^{-1}

Comme dans la section 2.2.1, on peut appliquer la méthode d'inversion à la loi normale. Bien que la fonction de répartition de celle-ci ne soit pas inversible, il est possible de l'approximer. Il existe au moins deux méthodes : Acklam et Moro. Leur construction diffère légèrement, mais l'idée est la même, et les résultats sont similaires. On n'en présentera donc ici qu'une seule : la méthode de Moro.

➢ **<u>présentation</u>**

Soit y la valeur de la loi uniforme générée. On a $y = \phi(x)$ *(d'après la section **2.2.1**).*

- on pose $z = y - 0.5$

- si $|z| \leq 0.42$, alors on approche x par $x = z.\left(\sum_{i=0}^{3} a_i z^{2i}\right) / \left(\sum_{j=0}^{4} b_j z^{2j}\right)$

- si $|z| > 0.42$, alors on approche x par $x = \varepsilon.\left(\sum_{i=0}^{8} c_i T_i(t)\right) - \varepsilon\,\frac{c_0}{2} = \varepsilon\, f(t)$

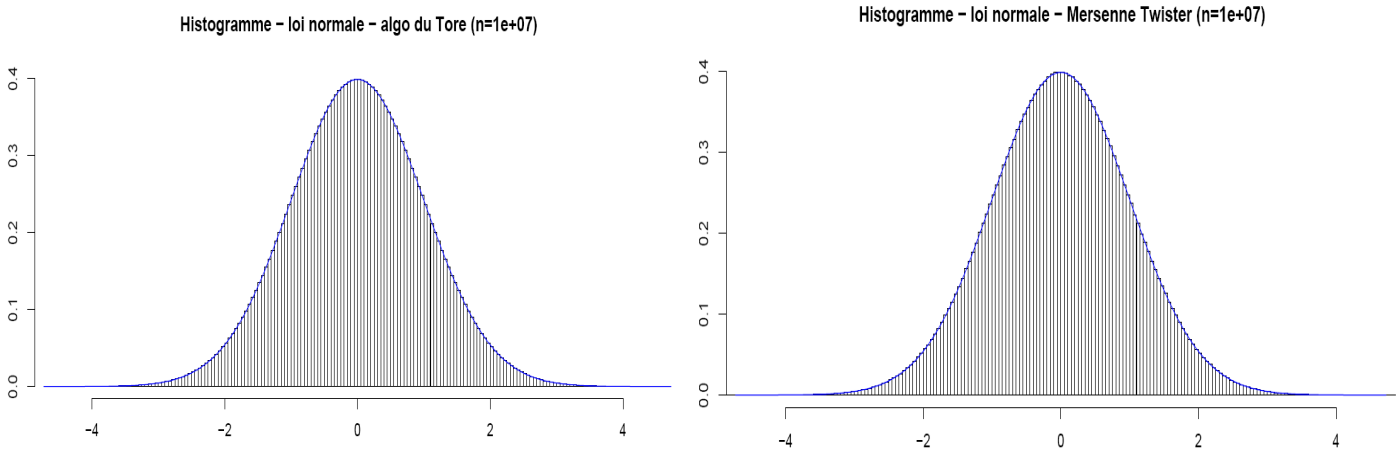
Ici, les valeurs a_i, b_i, c_i, d_i sont des réels fixés, donnés ***en annexe***.

On a posé $t = k_1\left\{2\ln\left(-\ln\left(\frac{1}{2} - |z|\right)\right) - k_2\right\}$.

La fonction $f(t) = \left(\sum_{i=0}^{8} c_i T_i(t)\right) - \frac{c_0}{2} \simeq t d_1 - d_2 + \frac{c_0}{2}$, avec d_1 et d_2 déterminés par l'algorithme suivant :

- Soit $d_{10} = 0$ et $d_9 = 0$.
- Soient d_i les réels déterminés par la relation récursive :
  $d_i = 2t d_{i+1} - d_{i+2} + c_j$ pour i=8, 7, …, 1

➢ **résultats (satisfaisants)**



Dans le cœur de la distribution, les valeurs empiriques se confondent avec les valeurs théoriques pour MT19937 et le tore.

Aux valeurs extrêmes, on observerait également, en théorie, une bonne approximation des valeurs empiriques. Il n'y a pas d'effet Neave, car la méthode d'inversion est une simple transformation arithmétique des valeurs générées uniformes en valeurs normales : les qualités d'équi-répartition en dimension 1 du tore et de MT19937 sont conservées lors de l'inversion.

2.2.2.3 Conclusion

L'algorithme de Box-Muller n'étant pas basé sur une approximation, à la différence des algorithmes d'inversion de la fonction de répartition de la loi normale, il devrait être de meilleure qualité que ceux-ci.

Néanmoins, il n'en est rien. Le tore n'est tout simplement pas compatible avec, à cause de sa dépendance terme à terme. Mersenne Twister souffre d'un effet Neave, et donc de valeurs extrêmes générées non réalistes. Or, les queues de distribution jouent un rôle fondamental dans les applications financières, car elles représentent les scénarios les plus extrêmes, donc les plus risqués et les plus redoutés par les opérateurs (les krach boursiers).

C'est pourquoi les assurances et banques préfèrent recourir aux algorithmes de Moro et d'Acklam. Pour les mêmes raisons que mentionnées dans la conclusion de la section **2.2.1**, la comparaison de lois normales simulées en assurance n'apporte pas d'information supplémentaire pour comparer les qualités des générateurs tore et Mersenne Twister, car avec Moro et Acklam, il y a une relation arithmétique directe entre la valeur uniforme générée et la valeur de la loi normale (approximée).

3. Applications

3.1 Simulation d'indicateurs statiques

➢ **Définition d'indicateur statique et méthode de quasi-Monte Carlo :**

Un indicateur statique est un indicateur où l'ordre de tirage des simulations n'a pas d'importance. Un exemple est le calcul d'intégrales, comme des espérances.

Pour le calcul de tels indicateurs, il est donc inutile de simuler à partir du tore mélangé (ce qui ne ferait qu'augmenter inutilement le temps de simulation), le tore suffit. En effet, la dépendance terme à terme des générateurs n'est pas un problème. Au contraire, cette dépendance est une force, dans le sens où elle permet aux points générés par le tore d'avoir de meilleures propriétés d'équi-répartition et de mieux remplir l'espace même avec un nombre de simulations peu importants. En théorie, la convergence sera plus rapide qu'avec des échantillons $(U_1,...,U_s)$ indépendants, qui seraient trop désordonnés, et donc qu'avec Mersenne Twister. En effet, MT19937, qui a des simulations plus aléatoires, mettra théoriquement plus de temps pour remplir l'espace.

C'est d'ailleurs pour cette raison qu'on utilise les méthodes de quasi-Monte Carlo pour le calcul d'intégrales. Leur but est d'accélérer la convergence vers la valeur de l'intégrale. Comme Monte Carlo, elle permet de calculer la valeur approchée d'une intégrale à partir d'une série de points x_1, x_2, …, x_N, avec la formule : $\int_{I^s} f(u)du \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)$ où I^s est le cube unitaire de dimension s, $I^s = [0;1]^s$, x_i un vecteur de s éléments.

La seule différence avec Monte Carlo est que dans quasi-Monte Carlo, les points x_1, x_2, …, x_N sont les termes d'une suite à discrépance faible (et non des nombres pseudo-aléatoires). Néanmoins, l'avantage de quasi-Monte Carlo n'est pas toujours aussi grand que ce qu'on aurait espéré.

Ainsi, la pratique va-t-elle vérifier la théorie ? Le tore permettra-t-il une convergence plus rapide que MT19937 ? Pour comparer les performances des deux algorithmes, nous allons comparer la valeur empirique d'un put européen en t avec sa valeur théorique.

➢ **Présentation du modèle :**

Soit un Put européen sur une action modélisée sous la probabilité risque neutre par un mouvement brownien géométrique, i.e :

$dS_t / S_t = rdt + \sigma\, dB_t$

où B_t un mouvement brownien sous la probabilité risque neutre

r le taux sans risque, σ la volatilité de l'actif risqué

Ce processus peut être discrétise comme suit :

$S_{t+\delta} = S_t \exp\left\{\left(r - \frac{\sigma^2}{2}\right)\delta + \sigma\sqrt{\delta}\,\varepsilon\right\}$ où ε suit une loi normale $N(0,1)$.

Pour rappel, un Put européen est une option sur une action S permettant de recevoir à la date d'échéance T un versement de $[K - S_T]^+$. Nous disposons d'une formule fermée pour évaluer en t le prix d'un Put européen, de prix d'exercice K et d'échéance T :

$P_t(S,K,T) = -S_0 N(-d_1) + Ke^{-rt}N(-d_2)$

où $d_1 = \frac{\ln\frac{S_t}{K} + \left(r + \frac{\sigma^2}{2}\right)(T-t)}{\sigma\sqrt{T-t}}$, $d_2 = d_1 - \sigma\sqrt{T-t}$,

et N désigne la fonction de répartition de la loi normale centrée réduite

Nous pouvons donc mesurer la performance des générateurs par le biais de l'erreur d'estimation du prix de l'option considérée. En effet, si S_T^i est le prix du titre considéré à la date T dans la i[ème] simulation de l'actif risqué, l'estimation naturel du prix du Put en t est :

$\overline{P}_t(S,K,T,n) = \frac{e^{-r(T-t)}}{n}\sum_{i=1}^{n}\left[S_T^i - K\right]^+$.

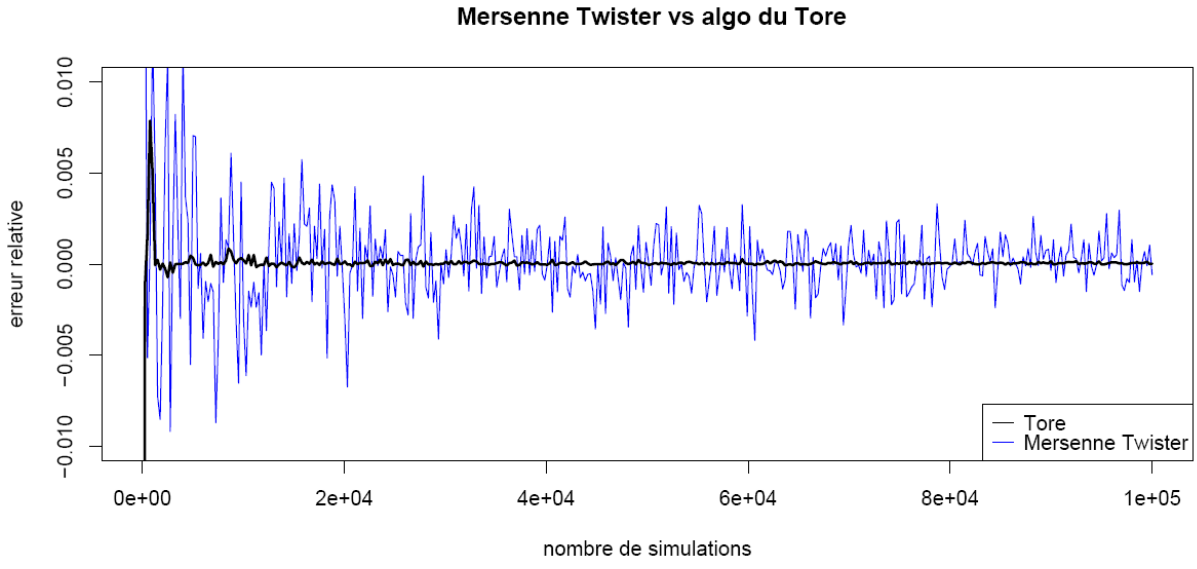
L'erreur relative d'estimation peut donc s'écrire : $\rho = \frac{\overline{P}_t(S,K,T,n) - P_t(S,K,T,n)}{P_t(S,K,T,n)}$.

➢ **Application :**

On fixe les paramètres $T = 1/2$, $S_0 = 60$, $K = 75$, $r = 8\%$, $\sigma = 10\%$.

On simule directement les S_T à partir de S_0 par $S_T = S_0 \exp\left\{\left(r - \frac{\sigma^2}{2}\right)T + \sigma\sqrt{T}\varepsilon\right\}$.

On obtient :



➢ **Conclusion :**

La convergence est beaucoup plus rapide avec le tore qu'avec MT19937. En effet, avec le tore, au bout de 1.000 simulations, l'erreur relative est inférieure à 0.1%. Il faut attendre approximativement 10.000 simulations pour que l'erreur avec MT soit inférieure à 0.1%.

Cela confirme notre idée de départ : le tore (et les autres générateurs à discrépance faible) sont beaucoup plus performants pour simuler les indicateurs statiques que MT19937 (et les autres générateurs pseudo-aléatoires), qui, de par leur caractère aléatoire, remplissent moins bien l'espace.

3.2 Simulation d'indicateurs dynamiques

➢ **Définition et conséquence sur le tore :**

Contrairement aux indicateurs statiques, l'ordre de tirage des simulations a de l'importance pour estimer les indicateurs dynamiques, car l'indicateur dynamique (tel que les trajectoires de cours) varie au cours du temps et les points de la trajectoire varient en fonction des simulations successives.

Ainsi, la dépendance terme à terme du Tore risque donc de poser problème. C'est pourquoi il va être nécessaire d'utiliser le tore mélangé dans cette partie.

Lequel des deux générateurs (MT19937 et tore mélangé) sera le plus apte à simuler efficacement des trajectoires ? Lequel alliera le mieux précision et rapidité ? Pour départager le tore mélangé et MT19937, nous allons travailler sur les trajectoires d'un mouvement brownien géométrique.

➢ **Présentation du modèle :**

Il s'agit ici de comparer l'évolution de la moyenne théorique d'un cours suivant un mouvement brownien géométrique, avec l'évolution de la moyenne empirique, i.e estimée à partir des simulations. Si le générateur est bon, normalement, pour un nombre de simulations suffisamment grand, la moyenne empirique est très proche de la moyenne théorique.

On a $dS_t / S_t = \mu\, dt + \sigma\, dB_t$, qu'on peut discrétiser par $S_{t+\delta} = S_t \exp\left\{\left(\mu - \frac{\sigma^2}{2}\right)\delta + \sigma\sqrt{\delta}\,\varepsilon\right\}$

où ε suit une loi normale $N(0,1)$.

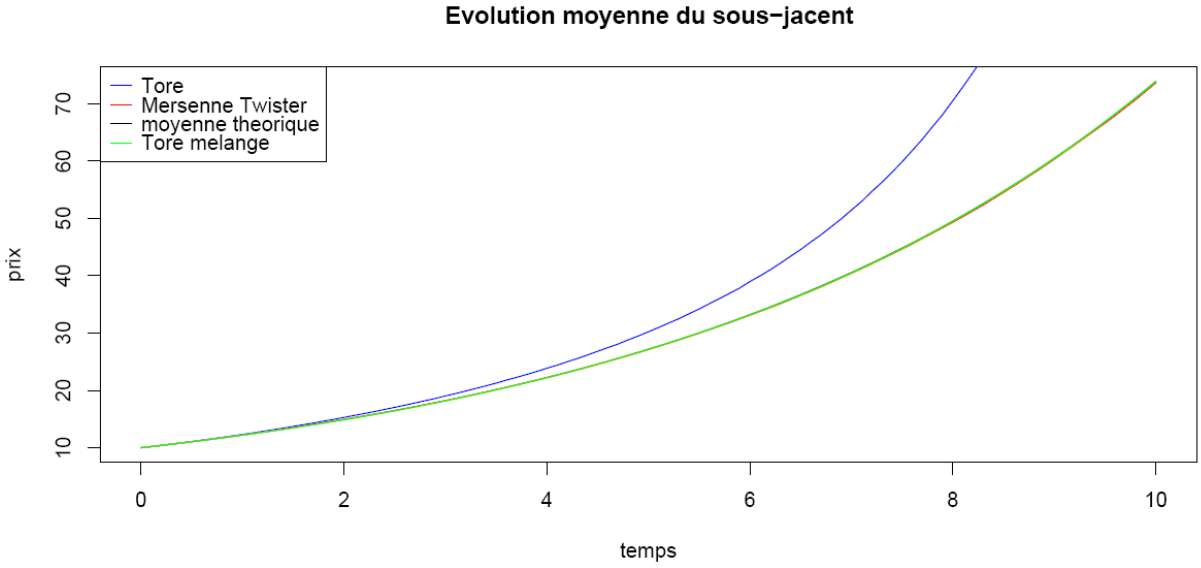
Le cours moyen $E(S_t)$ vérifie ainsi théoriquement :

$dE(S_t) / E(S_t) = \mu\, dt$, qu'on peut discrétiser par $E(S_{t+\delta}) = E(S_t) \times \exp\{\mu\,\delta\}$, $E(S_0) = S_0$

Soit S_t^i le cours du titre à la date t dans la i[ème] simulation. L'estimateur empirique du cours moyen à la date t, noté $\overline{S_t}$, est donné par : $\overline{S_t} = \frac{1}{N}\sum_{i=1}^{N} S_t^i$.

➢ **Application :**

On fixe ici les paramètres $\mu = 7.5\%$, $\sigma = 30\%$, $S_0 = 10$. Après avoir simulé 1000 trajectoires de S :



Les courbes rouge, noire, et verte sont ici confondues
Le tore généré est d'indice p=13 ; le tore est mélangé par MT19937, avec p=13

➢ **Conclusion :**

La dépendance terme à terme du tore peut engendrer une erreur importante dans la simulation de trajectoires, comme le montre le graphique : ici, l'évolution moyenne de sous-jacent est systématiquement surestimée.
En supprimant la dépendance terme à terme, le tore mélangé remédie à ce problème et permet d'obtenir des trajectoires réalistes. Il obtient des résultats identiques à MT19937 ; le problème est que le tore mélangé est plus long à simuler (*cf section **2.1.3***). MT19937 est donc plus adapté pour simuler les trajectoires.

Conclusion

Les générateurs utilisés virtuellement dans les simulations ne sont pas générées parfaitement aléatoirement, mais sont des procédés déterministes dont le but est d'imiter le mieux possible le hasard. Or, dans les assurances et banques, la gestion du risque occupe beaucoup de place, ce qui rend primordial le choix du générateur.

Deux générateurs sont fréquemment utilisés : l'algorithme du tore, qui est un générateur à discrépance faible, et l'algorithme de Mersenne Twister, qui est un générateur pseudo-aléatoire. La plupart des générateurs utilisés lors des simulations sont des quasi-aléatoires, car ils disposent d'excellentes qualités d'équi-répartition, contrairement à la plupart des générateurs pseudo-aléatoires.

Néanmoins, Mersenne Twister fait exception et dispose de bonnes qualités d'équi-répartition puisqu'il est équi-réparti en dimension 623. Le tore est mieux équi-réparti : la répartition des points du tore dans l'espace multidimensionnel se fait de manière moins aléatoire que MT19937 ; ainsi, le tore nécessite moins de simulations pour occuper complètement l'espace. Toutefois, le tore, contrairement à MT19937, souffre d'une dépendance terme à terme ; cet effet peut être annulé par l'adaptation du tore en un nouvel algorithme : le tore mélangé. L'unique problème du tore mélangé est qu'il va nécessiter un temps de simulation plus long que le tore, et par conséquent que MT19937, puisque le tore et MT19937 sont aussi rapides. Par ailleurs, le tore et MT19937 sont aussi bons l'un que l'autre pour la simulation de la loi normale, qui s'obtient dans les deux cas avec des méthodes d'inversions.

Ainsi, concrètement, le tore et MT19937 ne s'appliquent pas pour le même type de simulations. Le tore (comme les autres suites à discrépance faible) sera préféré à MT pour le calcul d'indices statiques tels que les intégrales et espérances car, grâce à ses qualités d'équi-répartition avec un faible nombre de simulations, la convergence vers l'intégrale est plus rapide. Par contre, on préférera MT pour le calcul d'indices dynamiques. En effet, la dépendance terme à terme du tore le rend inutilisable pour la simulation de trajectoires ; quant au tore mélangé, certes, il obtient d'aussi bons résultats que MT, mais son temps de simulation est plus long.

Aucun des deux algorithmes n'est meilleur que l'autre dans l'absolu : leur utilisation dépend seulement du type de simulations à effectuer.

Annexes

Annexe 1 : test des fréquences

Dans le test des fréquences, on segmente notre intervalle [0;1] en d intervalles égaux ; le $i^{ème}$ intervalle est noté A_i. On réalise M simulations X_i d'un générateur. On note N_i le nombre de simulations appartenant à A_i. Une variable de loi uniforme [0;1] a une probabilité d'appartenir à A_i égale à $\pi_i = 1/d$.

La statistique du test est $D^2 = \sum_{i=1}^{d} \frac{(N_i - M\pi_i)^2}{M\pi_i} = \sum_{i=1}^{d} \frac{(N_i - M/d)^2}{M/d}$.

D^2 est asymptotiquement distribué comme une variable de χ^2_{d-1}, d-1 étant le nombre de degrés de liberté.

L'hypothèse (H_0) est : les variables X_i sont i.i.d, de loi uniforme [0;1]
La p-valeur de ce test est donc donné par $\hat{a} = P(\chi^2_{d-1} > D^2)$.
Donc, plus la p-valeur est grande et proche de 1 pour un générateur (de même, plus la valeur de D est petite et proche de 0), plus cela traduit la qualité de la répartition du générateur dans [0;1].

Annexe 2 : Constantes de l'algorithme de Moreno

	a	b	c	k
0	2.506628238840	1	7.710887070549	0.417988642493
1	-18.615006252900	-8.473510930900	2.777201353369	4.245468688138
2	41.391197735340	23.083367437430	0.361496412926	
3	-25.441060496370	-21.062241018260	0.037341823343	
4		3.130829098330	0.002829714304	
5			0.000162571692	
6			8.017330474 * 10 ^ -6	
7			0.3840919865 * 10 ^ -6	
8			0.012970717 * 10 ^ -6	

Références

[1] Planchet F. et Thérond P, *« Simulation de trajectoires de processus continus »*

[2] Planchet F. et Thérond P, *« Outils numériques pour la simulation Monte Carlo des produits dérivés complexes »*

[3] Planchet F., Thérond P., Jacquemin J., *« Modèles financiers en assurance »,* Economica

[4] Gobet E., *Méthodes numériques*, cours ISFA

[5] Références spécifiques à la partie 1 :
http://fr.wikipedia.org/wiki/G%C3%A9n%C3%A9rateur_de_nombres_pseudo-al%C3%A9atoires
http://fr.wikipedia.org/wiki/G%C3%A9n%C3%A9rateur_de_nombres_al%C3%A9atoires
http://en.wikipedia.org/wiki/Mersenne_twister
http://en.wikipedia.org/wiki/Mersenne_prime
http://en.wikipedia.org/wiki/Random_seed
http://fr.wikipedia.org/wiki/G%C3%A9n%C3%A9rateur_congruentiel_lin%C3%A9aire
http://en.wikipedia.org/wiki/Linear_congruential_generator
http://en.wikipedia.org/wiki/Quasi-randomness

[6] Références spécifiques à la partie 2 :
http://www.apprendre-en-ligne.net/random/spectre.html
http://en.wikipedia.org/wiki/Equidistributed
http://fr.wikipedia.org/wiki/M%C3%A9thode_de_la_transform%C3%A9e_inverse
http://www.isfa.info/jwa/documentation/1226.nsf/769998e0a65ea348c1257052003eb94f/fe1a778a8b27e299c1256d6100487209/$FILE/02.Planchet.Jacquemin.Simulation%201.pdf
http://fr.wikipedia.org/wiki/M%C3%A9thode_de_Box-Muller#cite_note-0
http://www.isfa.info/jwa/documentation/1226.nsf/769998e0a65ea348c1257052003eb94f/fe1a778a8b27e299c1256d6100487209/$FILE/02.Planchet.Jacquemin.Simulation%201.pdf

[7] Références spécifiques à la partie 3 :
http://en.wikipedia.org/wiki/Quasi-Monte_Carlo_method

[8] Références spécifiques à l'introduction :
http://fr.wikipedia.org/wiki/Solvabilit%C3%A9_II
http://fr.wikipedia.org/wiki/Bale_II
http://fr.wikipedia.org/wiki/Normes_comptables_IFRS
Laurent JP [2007], *Risque de crédit* , cours ISFA