

Méthodes de simulation

Application aux événements rares

Régis LEBRUN

Airbus Central Research & Technology
regis.lebrun@airbus.com

27 janvier 2020

Plan

- 1 Générateurs pseudo - aléatoires
 - Générateurs uniformes

- 2 Méthodes de simulation avancées
 - Profiter de la structure de l'événement rare
 - Renoncer aux tirages indépendants
 - Renoncer aux tirages aléatoires

Simuler l'aléa sur ordinateur

Les méthodes numériques probabilistes reposent sur la génération de réalisations (ou simulation) de variables aléatoires. Mais peut-on générer des nombres aléatoires à partir d'une procédure déterministe ?

Réponse de Von Neumann (1951)

Any one who considers arithmetical methods of reproducing random digits is, of course, in a state of sin ... there is no such thing as a random number there are only methods of producing random numbers, and a strict arithmetic procedure of course is not such a method.

Il est cependant possible de construire des suites de nombres pseudo-aléatoires qui ressemblent à des réalisations indépendantes de variables aléatoires. A partir de générateurs pour des lois simples, notamment la loi uniforme sur $(0, 1)$, on peut en déduire des procédures pour simuler des variables de lois plus complexes.

Simuler l'aléa sur ordinateur

Les méthodes numériques probabilistes reposent sur la génération de réalisations (ou simulation) de variables aléatoires. Mais peut-on générer des nombres aléatoires à partir d'une procédure déterministe ?

Réponse de Von Neumann (1951)

Any one who considers arithmetical methods of reproducing random digits is, of course, in a state of sin ... there is no such thing as a random number there are only methods of producing random numbers, and a strict arithmetic procedure of course is not such a method.

Il est cependant possible de construire des suites de nombres pseudo-aléatoires qui ressemblent à des réalisations indépendantes de variables aléatoires. A partir de générateurs pour des lois simples, notamment la loi uniforme sur $(0, 1)$, on peut en déduire des procédures pour simuler des variables de lois plus complexes.

Simuler l'aléa sur ordinateur

Les méthodes numériques probabilistes reposent sur la génération de réalisations (ou simulation) de variables aléatoires. Mais peut-on générer des nombres aléatoires à partir d'une procédure déterministe ?

Réponse de Von Neumann (1951)

Any one who considers arithmetical methods of reproducing random digits is, of course, in a state of sin ... there is no such thing as a random number there are only methods of producing random numbers, and a strict arithmetic procedure of course is not such a method.

Il est cependant possible de construire des suites de nombres pseudo-aléatoires qui ressemblent à des réalisations indépendantes de variables aléatoires. A partir de générateurs pour des lois simples, notamment la loi uniforme sur $(0, 1)$, on peut en déduire des procédures pour simuler des variables de lois plus complexes.

Echantillonnages de variables aléatoires indépendantes uniforme $\mathcal{U}(0, 1)$

- Tous les algorithmes stochastiques reposent sur l'utilisation d'un tel échantillon.
- On cherche donc à construire une procédure informatique capable de produire des réalisations de variables aléatoires indépendantes et uniformément distribuées sur $[0, 1]$.
- On souhaite le faire de manière **algorithmique** et **déterministe** !

Qu'est-ce qu'un générateur de nombres pseudo - aléatoires ? I

Définition (cf [1]) : Un **générateur pseudo - aléatoire** est une structure $\mathcal{G} = (S, s_0, T, U, G)$ telle que :

- S est un ensemble **fini** d'**états**,
- $s_0 \in S$ est un **état initial** (parfois défini à l'aide d'un élément de U appelée **graine**),
- La fonction $T : S \rightarrow S$ est la **fonction de transition**,
- U est un ensemble fini de **symboles de sortie**,
- $G : S \rightarrow U$ est la **fonction de sortie** .

L'état du générateur évolue selon une relation de récurrence $s_n = T(s_{n-1})$ et le générateur produit les nombres $u_n = G(s_n)$, appelés **nombres pseudo-aléatoires**.

Quelques commentaires :

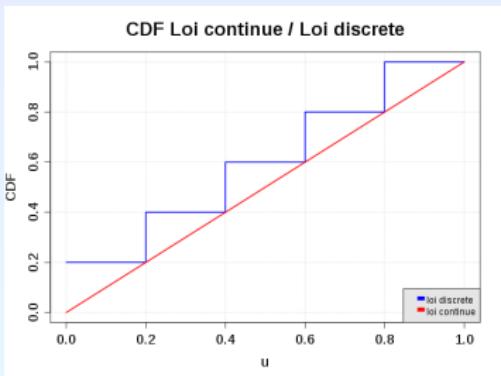
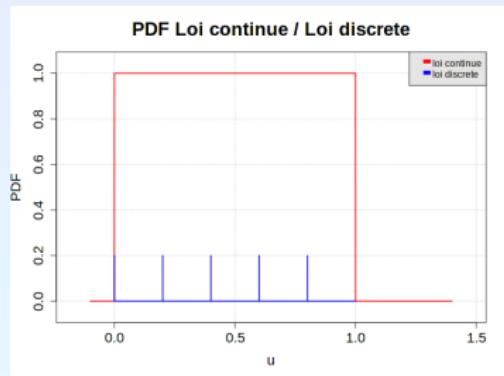
- Il n'y a rien concernant l'aléa dans la définition !
- Un générateur pseudo - aléatoire apparaît comme un système dynamique discret partiellement observé.
- G n'est pas nécessairement bijective (et même ne l'est jamais pour les générateurs de bonne qualité). Selon le type de U (double, float), on a $\text{card } U \ll \text{card } S$!
- Un générateur pseudo - aléatoire **est périodique, de période $P \leq \text{card } S$** .

Qu'est-ce qu'un générateur de nombres pseudo - aléatoires ? II

- Un tel système dynamique est supposé produire une suite de valeurs qui « ressemble» à une suite de réalisations iid d'une variable aléatoire uniforme sur $[0, 1]$, ie qui ne peut pas être différenciée d'une telle suite d'un point de vue statistique.
- La définition de ce que devrait être une suite aléatoire occupe plus de 30 pages dans [2], sans conclure de manière claire...
- L'uniformité sur $[0, 1]$ ne peut être atteinte : la représentation finie des réels par des nombres flottants ne le permet pas MAIS l'uniformité discrète sur $\Omega = \{0/m, 1/m, \dots, (m-1)/m\}$ peut être (assez bien) approchée via une propriété d'équi-repartition sur Ω , avec $m = \text{card } U$.

Qu'est-ce qu'un bon générateur de nombres pseudo - aléatoires ? |

On ne peut pas obtenir l'uniformité sur $[0, 1]$ du fait de la précision limitée de la représentation des nombres réels en machine. On essaie donc de générer une distribution uniforme discrète sur $\Omega = \{0/m, 1/m, \dots, (m-1)/m\}$, où $m = \text{card } U$.



Remark : La précision de l'approximation dépend de l'espace de symboles U !

L'indépendance entre les réalisations ne peut pas être obtenue du fait de la nature déterministe de la récurrence définissant le générateur, mais peut être approchée arbitrairement bien du fait des propriétés de mélange de T . Ce point peut être rendu plus précis à l'aide de la théorie des copules, voir le thé or ème de Mikusiński.

Validation empirique d'un générateur de nombres pseudo-aléatoires

Un générateur doit être testé de manière intensive avant d'entreprendre des simulations sérieuses (par exemple mettant en jeu la sûreté d'installations critiques)

Les tests doivent couvrir :

- L'uniformité des sorties du générateur : Kolmogorov - Smirnov, Chi - square, test sur les écarts.
- L'indépendance des sorties du générateur : test sériel, test spectral, autocorrelation

Les tests se regroupent en deux catégories :

- **Les tests empiriques** : on teste une propriété sur un échantillon de grande taille produit par le générateur
- **Les tests analytiques** : on teste une propriété sur l'ensemble complet U des valeurs produites par le générateur

Générateurs congruentiels I

Définition :

Soient les entiers $m > 0$ (le **module**), $a > 0$ (le **multiplicateur**) et c (la **constante additive**). Le générateur linéaire congruentiel associé au triplet (m, a, c) est défini par :

- $\mathcal{S} = \mathbb{Z}/m\mathbb{Z} = \{0, \dots, m-1\}$ pour $c > 0$ et $\mathcal{S} = \{1, \dots, m-1\}$ pour $c = 0$
- $s_n = T(s_{n-1}) = a.s_{n-1} + c \bmod m$
- $U = \{0, 1/m, \dots, 1 - 1/m\}$ pour $c > 0$ et $U = \{1/m, \dots, 1 - 1/m\}$ pour $c = 0$
- $u_n = G(s_n) = s_n/m$

Quand **$c = 0$** , ces générateurs sont dits **multiplicatifs**. Ils sont (étaient) souvent utilisés du fait du gain d'une addition.

Propriétés :

- C'est la première famille de générateurs disposant d'une théorie mathématique,
- **Ils sont rapides**,
- Ils sont faciles à implémenter,
- **Ils sont (encore) très répandus (`drand()`* , `drand48()`* ...)**

Mais ils présentent de sérieux défauts :

- Période courte dans les implantations usuelles ($m = 2^{32} \simeq 5.10^9$);

Générateurs congruentiels II

- Mauvaise répartition en grande dimension !
- Les bits de poids faibles ont une période plus courte qu'attendu (donc ne pas prendre $y_n = s_n \bmod M$ si l'on souhaite des entiers dans $\{0, \dots, M-1\}$ à partir d'entiers dans $\{0, \dots, m-1\}$ avec $M < m$).

(*) : `drand()` est un générateur linéaire avec $m = 2^{32}$: S regroupe less entiers 32 bits.

(*) : `drand48()` est un générateur linéaire avec $m = 2^{48}$: S regroupe les entiers 48 bits. Il a pour objectif de produire des nombres flottants 32 bits (avec 23 bits de mantisse), mais peut également produire des nombres flottants 64 bits en complétant la mantisse de 52 bits par des zéros.

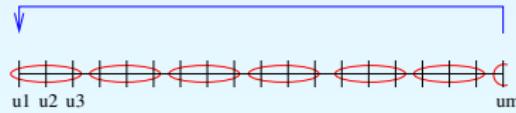
Générateurs congruentiels multiplicatifs I

Comment choisir m, a ?

- Si $m = 2^M$, $M \geq 3$, la période maximale est 2^{M-2} , atteinte par tous les états initiaux impairs si $a \bmod 8 \equiv 3$ ou 5 .
- La période $m - 1$ est atteinte seulement si m est premier. Dans ce cas, la période divise $m - 1$, et vaut $m - 1$ si et seulement si a est une racine primitive de $m - 1$, c'est à dire $a \neq 0$ et $a^{(m-1)/p} \not\equiv 1 \bmod m$ pour tout diviseur premier p de $m - 1$.
- Plus de détails dans [2].

Problèmes en grande dimension

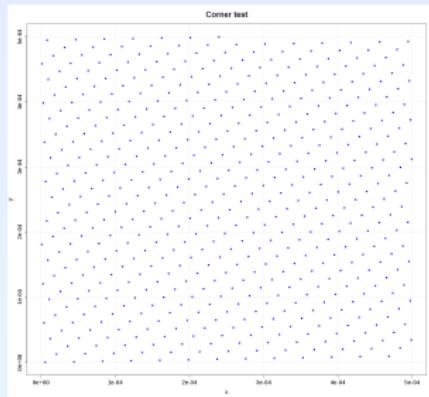
Supposons qu'on souhaite échantillonner uniformément l'hypercube $[0, 1]^d$. Une façon naturelle d'y parvenir consiste à générer les points $(u_{dn}, u_{dn+1}, \dots, u_{dn+d-1})_{n \geq 0}$.



On ne peut générer au plus que m (ou $m - 1$) points dans $[0, 1]^d$, qui seront nécessairement répartis de manière très lacunaire en grande dimension.

Dans le cas particulier des générateurs linéaires ou multiplicatifs, ces points sont également distribués selon un motif très régulier : un réseau de points.

Générateurs congruentiels multiplicatifs II



Réseau obtenu avec la fonction `random()` (librairie standard C) en dimension 2.
Equirépartition : ok...mais que dire de l'**indépendance** ?

Générateurs congruentiels multiplicatifs III

Théorème de Marsaglia, [3]

Soit $d > 0$ un entier. Soit $\alpha_0, \dots, \alpha_{d-1}$ un ensemble quelconque d'entiers tels que :

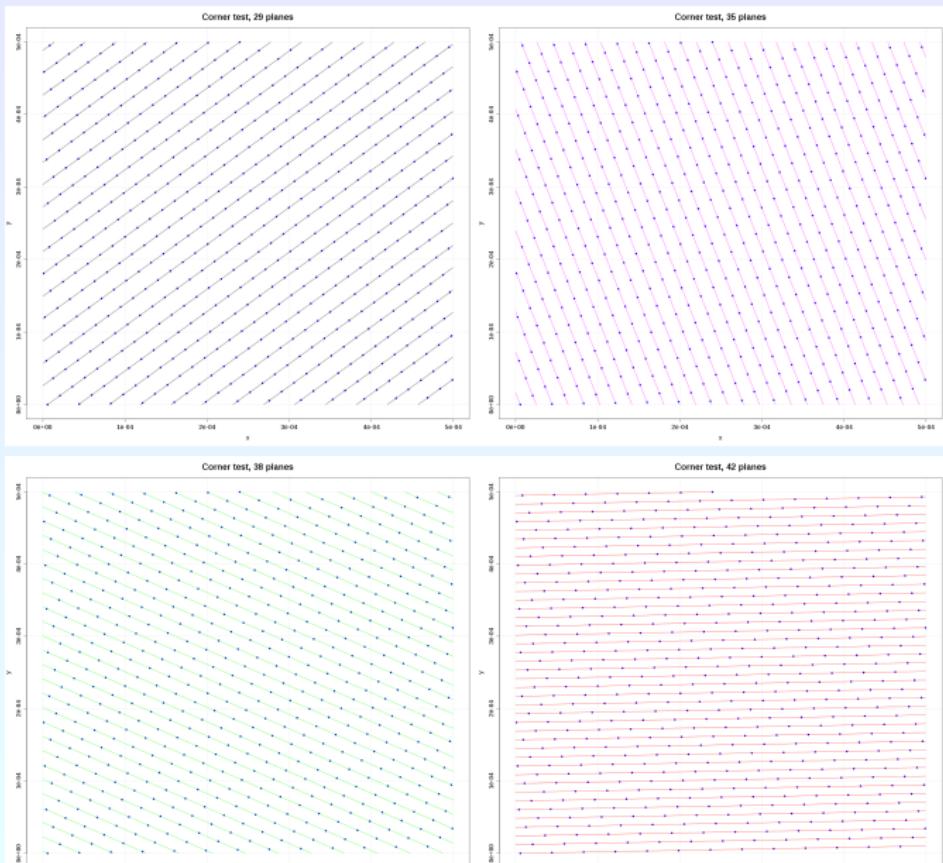
$$\alpha_0 + \cdots + \alpha_{d-1}a^{d-1} \equiv 0 \pmod{m} \quad (1)$$

alors tous les points de la forme (u_n, \dots, u_{n+d-1}) vont appartenir à un réseau d'hyperplans parallèles définis par les équations :

$$\alpha_0 x_0 + \cdots + \alpha_{d-1} x_{d-1} = 0, \pm 1, \pm 2, \dots \quad (2)$$

Il y a au plus $|\alpha_0| + \cdots + |\alpha_{d-1}|$ tels hyperplans en intersection avec $(0, 1)^d$, et il est toujours possible de choisir $\alpha_0, \dots, \alpha_{d-1}$ tels que les points tombent tous dans moins de $(d!m)^{1/d}$ hyperplans.

Générateurs congruentiels multiplicatifs IV



Générateurs congruentiels multiplicatifs V

Ces valeurs sont données pour les *méilleurs* générateurs multiplicatifs... (choix optimal

	$d = 3$	$d = 4$	$d = 5$	$d = 6$	$d = 7$	$d = 8$	$d = 9$	$d = 10$
$m = 2^{23}$	369	119	63	42	32	27	24	22
$m = 2^{32}$	2953	566	220	120	80	60	48	41
$m = 2^{52}$	300079	18131	3520	1216	582	340	227	166
$m = 2^{64}$	4801279	145055	18578	4866	1910	963	573	382

Aucun point ne tombe entre deux hyperplans consécutifs. Au moins une de ces tranches a un volume V tel que $V > \frac{1}{N}$ où N est le nombre total d'hyperplans intersectant $]0, 1[^d$.

Générateurs congruentiels multiplicatifs VI

Exemple 1 : pour $m = 2^{32}$ (simple précision) et $d = 3$, il existe une tranche de probabilité $\frac{1}{2953} = 3.10^{-4}$ qui n'est PAS échantillonnée par le générateur.

Si l'événement rare est inclu dans une telle tranche, sa probabilité sera estimée comme nulle !

Example 2 : pour $m = 2^{52}$ (double précision) et $d = 10$, il existe une tranche de probabilité $\frac{1}{166} = 6.10^{-3}$ qui n'est PAS échantillonnée par le générateur.

Si l'événement rare est inclu dans une telle tranche, sa probabilité sera estimée comme nulle !

Générateurs congruentiels multiplicatifs VII

Volume des tranches non explorées par les générateurs multiplicatifs :

	$d = 3$	$d = 4$	$d = 5$	$d = 6$	$d = 7$	$d = 8$	$d = 9$	$d = 10$
$m = 2^{23}$	2.7e-3	8.4e-3	1.6e-2	2.4e-2	3.1e-2	3.7e-2	4.2e-2	4.5e-2
$m = 2^{32}$	3.4e-4	1.8e-3	4.5e-3	8.3e-3	1.2e-2	1.7e-2	2.1e-2	2.4e-2
$m = 2^{52}$	3.3e-6	5.5e-5	2.8e-4	8.2e-4	1.7e-3	340	2.9e-3	6.0e-3
$m = 2^{64}$	2.1e-8	6.9e-6	5.4e-5	2.0e-4	5.2e-4	1.0e-3	1.7e-3	2.6e-3

Tests analytiques I

Le test des écarts

Ce test a pour objectif de tester si un générateur pseudo-aléatoire associé à une mesure μ produit des valeurs U qui couvrent le support de μ de façon régulière.

Plus précisément :

- Soit $\rho = 1/\text{card } U \ll 1$ la résolution théorique optimale d'un générateur ;
- Trouver $\mu(I^*) = \max_{I=]s_i, s_j[, I \cap U=\emptyset} \mu(I)$, où $s_i, s_j \in U$;
- Vérifier que $\mu(I^*) = \mathcal{O}(\rho)$: le générateur ne laisse pas d'écarts trop importants dans le support de μ .

On remarque qu'un générateur linéaire de période maximale vérifie parfaitement ce test, dont la valeur d'écart est ρ , ou $\rho/(1-\rho)$ pour les générateurs multiplicatifs.

Tests analytiques II

Le test spectral

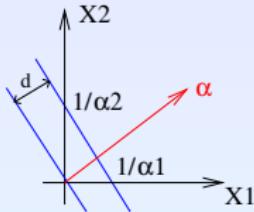
Ce test est conçu pour détecter les non-uniformités dans la répartition multidimensionnelle de U dans $[0, 1]^d$. Il a été conçu après une période d'utilisation intensive des générateurs linéaires aux propriétés médiocres (tel que le fameux RANDU fournit par les ordinateurs IBM des années 80). Le test est défini par :

- Calculer la quantité :

$$\nu_d = \min \left\{ \|\alpha\|_2 \mid \alpha \in \mathbb{Z}^d, \sum_{i=0}^{d-1} \alpha_i a^i \equiv 0 \pmod{m} \right\} \quad (3)$$

pour $d = 2, \dots, d_{max}$. Cette quantité $(\nu_d)^{-1}$ est liée à l'écart maximum entre deux hyperplans consécutifs dans une famille d'hyperplans parallèles qui couvre les points en dimension d générés par le générateur étudié.

Tests analytiques III



- Calculer la quantité normalisée :

$$\mu_d = \frac{\pi^{d/2} \nu_d^d}{\Gamma(d/2 + 1)m} \quad (4)$$

- Le test est réussi si $\mu_d > 0.1$ for $d = 2, \dots, 6$.

Expérimentalement, ce test a permis de détecter très efficacement les mauvais générateurs linéaires des bons générateurs.

Autres générateurs apparentés aux générateurs linéaires

Les générateurs linéaires ne sont pas meilleurs que les générateurs multiplicatifs

- **Le théorème de Marsaglia s'étend aux générateurs linéaires généraux** : même si les détails changent, les générateurs linéaires ne sont pas adaptés aux applications en grande dimension, à moins d'utiliser des valeurs **énormes** pour m ($m > 10^{1500}$), pour lesquelles on peut espérer travailler en dimension $d = 100$ sans trop de crainte.
- **Les générateurs qui chainent différents générateurs linéaires souffrent des même défauts** : c'est seulement une façon d'atteindre de plus grandes valeurs de m , mais partant de $m = 2^{32}$ on ne peut atteindre une résolution suffisante sans utiliser un nombre considérable d'étages dans le chainage des générateurs. Ainsi, la combinaison de k générateurs linéaires de module m_1, \dots, m_k a une période maximale de $P = \frac{(m_1-1)\dots(m_k-1)}{2^{k-1}} \simeq 2^{32(k-1)}$.

Mieux que les générateurs linéaires

SIMD - oriented Fast Mersenne Twister (SFMT), d'après [5]

Trois des propriétés les plus importantes d'un bon générateur sont :

- une période énorme ;
- une bonne répartition des points en grande dimension ;
- une génération efficace des nombres pseudo-aléatoires.

Le générateur SIMD - oriented Fast Mersenne Twister satisfait à ces contraintes avec une période de $2^{19937} - 1 \simeq 10^{6001}$ (qui peut facilement être étendue à $2^{216091} - 1 \simeq 10^{65050}$), une équirépartition prouvée en dimension jusqu'à 382 (resp. 4077) lorsqu'on utilise ces générateurs pour produire des nombres en double précision.

Détails du SFMT I

L'espace d'état S de ce générateur est $[(\mathbb{Z}/2\mathbb{Z})^w]^N$: un élément s de S est vu comme un vecteur de dimension N (x_0, \dots, x_{N-1}) sur $(\mathbb{Z}/2\mathbb{Z})^w$. Chaque x_k est appelé *register*. La fonction de transition T est linéaire et très creuse et peut être vue comme une récurrence linéaire à N termes sur $(\mathbb{Z}/2\mathbb{Z})^w$ de la forme suivante :

$$T : ((x_n)_0, \dots, (x_n)_{N-1}) = ((x_{n-1})_1, \dots, (x_{n-1})_{N-1}, z)$$

with $z = g(((x_{n-1})_0, \dots, (x_{n-1})_{N-1}))$ (5)

avec g une application linéaire creuse. Pour SFMT, $w = 128$ de façon à stocker les x_k dans des registres SIMD (Single Instruction, Multiple Data) des microprocesseurs modernes, et la fonction g est telle que :

$$g(x_0, \dots, x_{N-1}) = x_0 \mathbf{A} + x_M \mathbf{B} + x_{N-2} \mathbf{C} + x_{N-1} \mathbf{D} \quad (6)$$

Les paramètres M , N , \mathbf{A} , \mathbf{B} , \mathbf{C} et \mathbf{D} ont été choisis de façon à avoir une période multiple de $2^{19937} - 1$ et de fournir une équirépartition des points en grande dimension, ce qui conduit à l'expression suivante, pour $x \in (\mathbb{Z}/2\mathbb{Z})^{128}$:

- $N = 156$ and $M = 122$,
- $x\mathbf{A} = (x \lll 8) \wedge x$,

Détails du SFMT II

- $x\mathbf{B} = (x \gg 32) \oplus (BFFFFFFF\ 6\ BFFAFFFF\ DDFEBCB7F\ DFFFFFFEF)$,
- $x\mathbf{C} = (x \gg 128)$,
- $x\mathbf{D} = (x \ll 32)$.

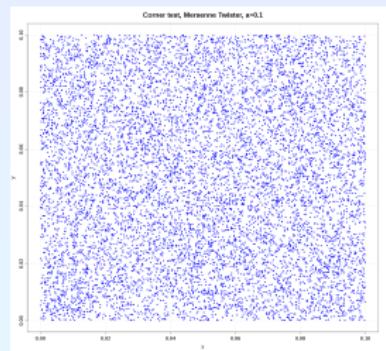
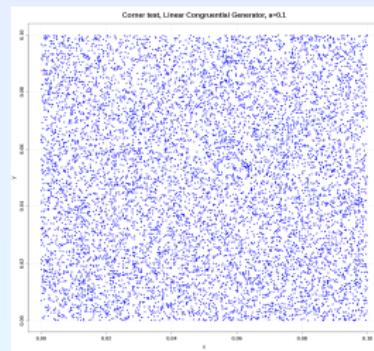
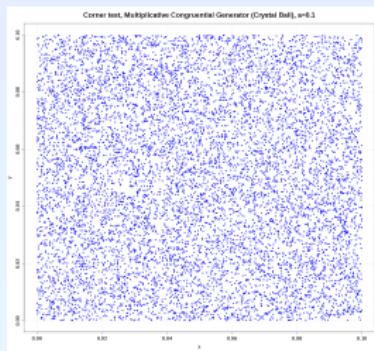
$BFFFFFFF\ 6\ BFFAFFFF\ DDFEBCB7F\ DFFFFFFEF$ est un nombre de 128 bits représenté comme 4 blocs de 8 nombres en base 16 (où les chiffres sont $(0, \dots, 9, A, \dots, F)$) :
 $\mathbb{Z}/2^{128}\mathbb{Z} \sim [(\mathbb{Z}/2^4\mathbb{Z})^8]^4$.

En utilisant des instructions SIMD, les calculs précédents peuvent être implémentés de façon très concise et efficace .

L'initialisation se fait en prenant un état (x_0, \dots, x_{N-1}) quelconque puis le générateur évalue $z = g(((x_0, \dots, (x_{N-1}))$ lorsque nécessaire.

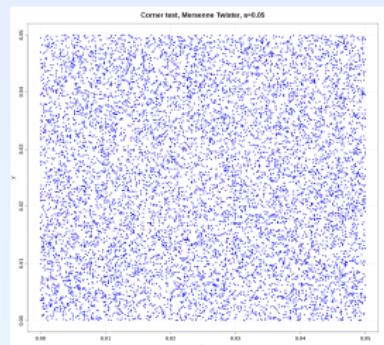
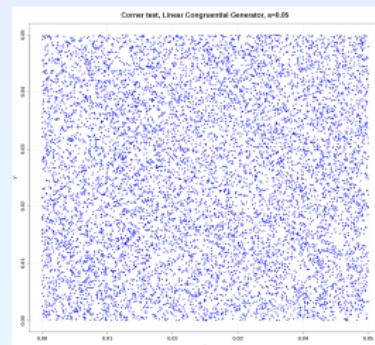
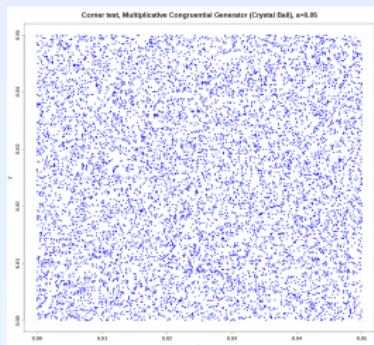
Remarque : le registre x_k est un nombre $w = 128$ bits donc il ne correspond pas à un unique nombre pseudo-aléatoire u !

- Si U représente des entiers 32 bits *int32*, alors x_k contient 4 réalisations,
- Si U représente des entiers 64 bits *int64*, alors x_k contient 2 réalisations.

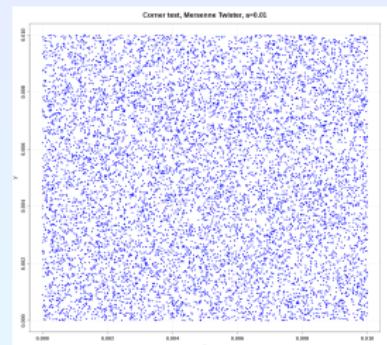
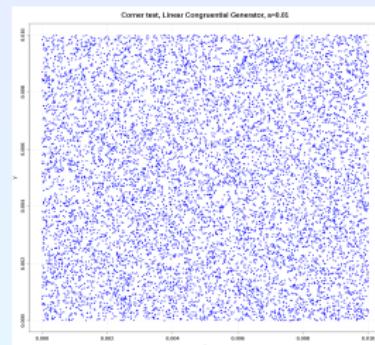
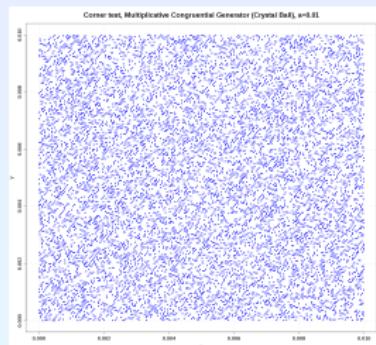
Corner test : 10^4 realizations in $[0, 0.1]^2$ 

Comparison between Crystal Ball, `random()` and the Mersenne Twister generator.

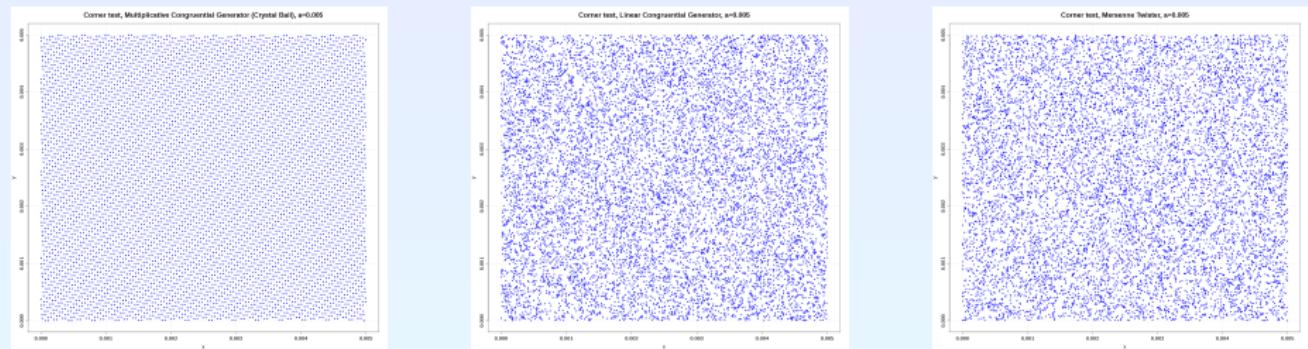
Corner test : 10^4 réalisations dans $[0, 0.05]^2$



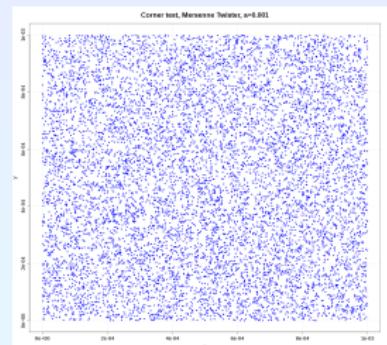
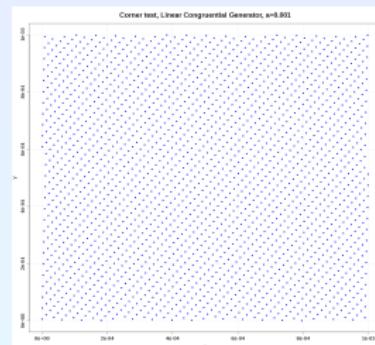
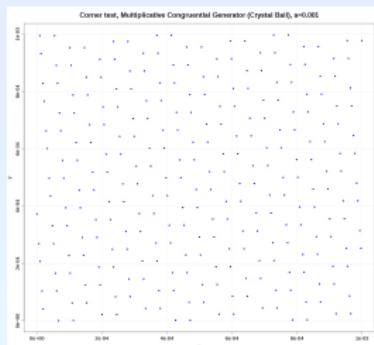
Comparaison entre Crystal Ball, `random()` et le Mersenne Twister generator.

Corner test : 10^4 réalisations dans $[0, 0.01]^2$ 

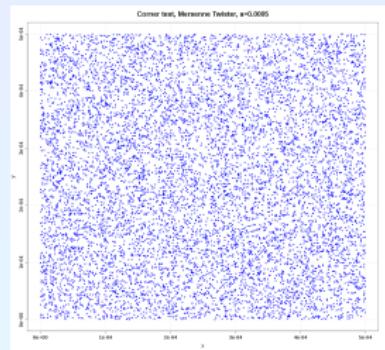
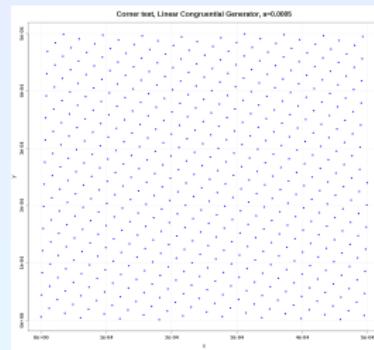
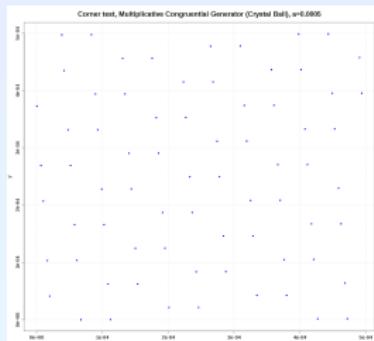
Comparaison entre Crystal Ball, `random()` et le Mersenne Twister generator.

Corner test : 10^4 réalisations dans $[0, 0.005]^2$ 

Comparaison entre Crystal Ball, `random()` et le Mersenne Twister generator.

Corner test : 10^4 réalisations dans $[0, 0.001]^2$ 

Comparaison entre Crystal Ball, `random()` et le Mersenne Twister generator.

Corner test : 10^4 réalisations dans $[0, 0.0005]^2$ 

Comparaison entre Crystal Ball, `random()` et le Mersenne Twister generator.

Avec OpenTURNS I

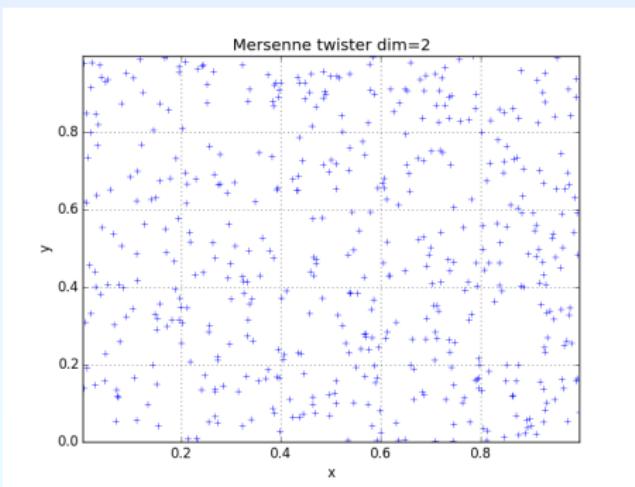
Listing 1 – mersenne.py

```
from openturns import *
from openturns.viewer import *
from time import *

# Taille de l'échantillon
size = 500
# Accès direct, flux 1D
t0 = time()
data = RandomGenerator.Generate(size)
t1 = time()
print("t=", size / (t1 - t0), "s")
# Accès via une distribution, flux 1D ou nD
dim = 2
distribution = ComposedDistribution([Uniform(0.0, 1.0)]*dim)
t0 = time()
data = distribution.getSample(size/2)
t1 = time()
print("t=", size / (t1 - t0), "s")
# Accès sous forme de DOE, flux 1D ou nD
doe = MonteCarloExperiment(distribution, size)
data = doe.generate()
t1 = time()
```

Avec OpenTURNS II

```
print("t=", size / (t1 - t0), "s")
graph = Graph("Mersenne twister dim=" + str(dim), "x", "y", True, "")
graph.add(Cloud(data))
view = View(graph)
view.save("mersenne.png")
view.close()
Show(graph)
```



Avec OpenTURNS III

en utilisant DistFunc : $98 \cdot 10^6$ réalisations/s et en utilisant Uniform() : $33 \cdot 10^6$ réalisations/s

References |

- ① Jerry Banks (ed.), "Handbook of Simulation : Modelling, Estimation and Control", Wiley, 1998.
- ② Donald E. Knuth, "The Art of Computer Programming, Vol. 2 : Seminumerical algorithms, 3rd Ed.", Addison - Wesley, 1998.
- ③ George Marsaglia, "Random numbers fall mainly in the planes", Proc N.A.S, 1968.
- ④ G. H. Hardy, E. M. Wright, "An introduction to the theory of numbers, 4th ed.", Oxford University Press, 1975.
- ⑤ Mutsuo Saito, Makoto Matsumoto, "SIMD-oriented Fast Mersenne Twister : a 128-bit Pseudorandom Number Generator", research report, Dep. of Math, Hiroshima University, 2006.
- ⑥ M. Matsumoto, T. Nishimura, "Mersenne twister : a 623-dimensionally equidistributed uniform pseudo-random number generator". ACM Transactions on Modeling and Computer Simulation 8 (1), 3 - 30, 1998.
- ⑦ Luc Devroye, "Non-Uniform Random Variate Generation", Springer, 1986.
- ⑧ Wolfgang Hörmann, "A Note on the Quality of Random Variates Generated by the Ratio of Uniform Method", ACM Transaction on Modelling and Computer Simulation, Vol 4, 1.

References II

- ⑨ Jurgen A. Doornik, "An Improved Ziggurat Method to Generate Normal Random Samples", working paper, <http://www.doornik.com/research.html>.
- ⑩ B. A. Wichmann, I. D. Hill, "Algorithm AS 183 : An Efficient and Portable Pseudo-Random Number Generator", *Applied Statistics*, 31 (2), 188 - 190, 1982.
- ⑪ A. I. McLeod, "A Remark on Algorithm AS 183 : An Efficient and Portable Pseudo-Random Number Generator", *Applied Statistics*, 34 (2), 198 - 200, 1985.
- ⑫ B. D. McCullough, "Microsoft Excel's 'Not The Wichmann-Hill' random number generators", *Computational Statistics and Data Analysis*, 52, 4587 - 4593, 2008.
- ⑬ Online Crystall Ball documentation, http://docs.oracle.com/cd/E12825_01/epm.111/cb_statistical/frameset.htm?ch02s05.html, Oracle, 05-29-2012.

Plan

1 Générateurs pseudo - aléatoires

- Générateurs uniformes

2 Méthodes de simulation avancées

- Profiter de la structure de l'événement rare
- Renoncer aux tirages indépendants
- Renoncer aux tirages aléatoires

Méthodes de Monte Carlo multiniveaux I

Dans de nombreux cas, l'événement rare s'écrit :

$$E = \{f(\mathbf{X}) \geq s\}$$

où s est un seuil tel que $p = \mathbb{P}(E) \ll 1$ et f est un logiciel de simulation coûteux. Par ailleurs, on dispose souvent d'une collection d'approximations $f_1, \dots, f_k = f$ de f de précision et de coût d'évaluation croissants. Par exemple, les fonctions f_i correspondent à des résolutions sur des maillages plus grossiers d'une équation aux dérivées partielles et f_k est la solution associée au maillage le plus fin.

On peut alors écrire :

$$p = \mathbb{E} [\mathbb{1}_{f(\mathbf{X}) \geq s}] = \mathbb{E} [\mathbb{1}_{f_1(\mathbf{X}) \geq s}] + \sum_{i=2}^k \mathbb{E} [\mathbb{1}_{f_i(\mathbf{X}) \geq s} - \mathbb{1}_{f_{i-1}(\mathbf{X}) \geq s}] \quad (7)$$

Méthodes de Monte Carlo multiniveaux II

Rappel sur la simulation par variable de contrôle : on approche p par :

$$\hat{p}_N = \frac{1}{N} \sum_{i=1}^N \left\{ \mathbb{1}_{f(\mathbf{x}_i) \geq s} - \lambda (\mathbb{1}_{g(\mathbf{x}_i) \geq s} - q) \right\} \quad (8)$$

où g est telle que la variable aléatoire $\mathbb{1}_{f(\mathbf{x}) \geq s}$ est très corrélée avec la variable aléatoire $\mathbb{1}_{g(\mathbf{x}) \geq s}$, $q = \mathbb{E} [\mathbb{1}_{g(\mathbf{x}) \geq s}]$ est connue et λ est un réel choisi pour minimiser la variance de \hat{p}_N (λ peut être estimé au fil des simulations).

La méthode de Monte Carlo multiniveaux consiste à estimer p par :

$$\tilde{p}_{N_1, \dots, N_k} = \frac{1}{N_1} \sum_{j_1=1}^{N_1} \mathbb{1}_{f_1(\mathbf{x}_{j_1}) \geq s} + \sum_{i=2}^k \frac{1}{N_i} \sum_{j_i=1}^{N_i} \left\{ \mathbb{1}_{f_i(\mathbf{x}_{j_i}) \geq s} - \mathbb{1}_{f_{i-1}(\mathbf{x}_{j_i}) \geq s} \right\} \quad (9)$$

où les vecteurs aléatoires \mathbf{X}_{j_i} sont indépendants.

La simulation multiniveaux à 2 niveaux correspond donc à la simulation par variable de contrôle avec $\lambda = 1$ et q inconnue donc devant être estimée.

Méthodes de Monte Carlo multiniveaux III

On note :

- C_1, V_1 le coût d'évaluation et la variance de $\mathbb{1}_{f_1(x_{j_1}) \geq s}$
- C_i, V_i le coût d'évaluation et la variance de $\mathbb{1}_{f_i(x_{j_1}) \geq s} - \mathbb{1}_{f_{i-1}(x_{j_1}) \geq s}$

Le coût total de l'estimation de p par $\tilde{p}_{N_1, \dots, N_k}$ est alors $\sum_{i=1}^k N_i C_i$ et la variance associée $\sum_{i=1}^k \frac{V_i}{N_i}$. Pour un coût total donné, la variance est minimisée en prenant $N_i \propto \sqrt{V_i/C_i}$.

Tirage d'importance adaptatif I

La technique de tirage d'importance, encore appelée méthode de Monte Carlo pondérée, est basée sur la remarque suivante :

$$p = \mathbb{E} [\mathbb{1}_{f(\mathbf{X}) \geq s}] = \mathbb{E} \left[\frac{p_{\mathbf{X}}(\mathbf{X})}{r_{\mathbf{X}}(\mathbf{X})} \mathbb{1}_{f(\mathbf{X}) \geq s} \right] \quad (10)$$

où \mathbf{X} est distribué selon la densité $r_{\mathbf{X}}$ dans la dernière espérance. Elle consiste à estimer p par :

$$\hat{p}_N = \frac{1}{N} \sum_{i=1}^N \frac{p_{\mathbf{X}}(\mathbf{X}_i)}{r_{\mathbf{X}}(\mathbf{X}_i)} \mathbb{1}_{f(\mathbf{X}_i) \geq s} \quad (11)$$

où \mathbf{X} est tiré selon la loi de densité $r_{\mathbf{X}}$ au lieu de la densité initiale $p_{\mathbf{X}}$. La variance de \hat{p}_N peut être fortement réduite par un choix judicieux de $r_{\mathbf{X}}$, et même annulée si $r_{\mathbf{X}}(\mathbf{x}) = r^*(\mathbf{x}) = \frac{\mathbb{1}_{f(\mathbf{x}) \geq s} p_{\mathbf{X}}(\mathbf{x})}{p}$. Le problème est que r^* nécessite de connaître p ... Mais on peut chercher à approcher r^* par une séquence de densités de probabilités $r^0 = p, r^1, \dots, r^k$. La méthode procède alors de la manière suivante :

Tirage d'importance adaptatif II

- On part d'une population de taille N_0 tirée selon $r^0 = px$. On sélectionne la fraction $\alpha \in]0, 1[$ conduisant aux valeurs les plus élevées de $Y_{j_0} = f(\mathbf{X}_{j_0})$. Cette population permet de construire une première densité d'importance \hat{r}^1 par reconstruction à noyau.
- A l'étape $i \geq 1$, on tire une population N_i selon r^i et on évalue $Y_{j_i} = f(\mathbf{X}_{j_i})$. Si la proportion de la population Y_{j_i} dépassant s est suffisante, on stoppe l'adaptation en posant $r^\infty = r^i$, sinon on sélectionne la fraction $\alpha \in]0, 1[$ conduisant aux valeurs les plus élevées de $Y_{j_i} = f(\mathbf{X}_{j_i})$ pour construire r^{i+1}
- On estime p par tirage d'importance selon la loi r^∞ .

Tirage d'importance adaptatif III

Remarques importantes :

- Les différents tirages sont *couplés* du fait que la mesure d'importance de l'étape i est une fonction des tirages des étapes précédentes. Ce couplage est cependant asymptotiquement négligeable dans la variance totale de l'estimateur.
- La loi reconstruite \hat{r}^i doit être telle que $\text{Var}\left[\frac{p_{\mathbf{X}}(\mathbf{X}_i)}{\hat{r}^i(\mathbf{X}_i)}\right] < \infty$. Un moyen de garantir cette propriété est de prendre pour \hat{r}^i une mixture de poids $(\epsilon, 1 - \epsilon)$ entre $p_{\mathbf{X}}$ et la reconstruction à noyaux de l'étape i .
- La densité \hat{r}^i est attirée par les modes de $Y = f(\mathbf{X})$. Si la population initiale est trop faible et qu'un des modes n'est pas échantillonné, il ne sera pas découvert par la suite...

Tirage par sous-ensembles (ou tirages particulaires) I

Cette méthode s'appuie sur la décomposition suivante :

$$p = \mathbb{P}(f(\mathbf{X}) \geq s) = \mathbb{P}(f(\mathbf{X}) \geq s_0) \prod_{i=1^k} \mathbb{P}(f(\mathbf{X}) \geq s_i | f(\mathbf{X}) \geq s_{i-1}) \quad (12)$$

où $s_0 = -\infty < s_1 < \dots < s_k = s$. On a alors $\mathbb{P}(f(\mathbf{X}) \geq s_0) = 1$ et si p est de l'ordre de 10^{-k} , en choisissant judicieusement les seuils intermédiaires s_1, \dots, s_{k-1} on peut se ramener à l'estimation de k probabilités conditionnelles $\mathbb{P}(f(\mathbf{X}) \geq s_i | f(\mathbf{X}) \geq s_{i-1})$ toutes de l'ordre de 10^1 dont beaucoup plus accessibles à une estimation par la méthode de Monte Carlo.

Deux questions se posent :

- Comment choisir les seuils s_i ?
- Comment échantillonner selon la loi de \mathbf{X} sachant $f(\mathbf{X}) \geq s_i$?

La méthode de tirage par sous-ensembles consiste à faire évoluer une population de taille N de façon à pouvoir estimer chacune des probabilités conditionnelles à l'aide de cette population. Les seuils sont construits itérativement comme des quantiles empiriques de la variable $Y = f(\mathbf{X})$ évaluée sur la population. On obtient ainsi :

- On se donne $\alpha \in]0, 1[$ et $N > 0$ et on pose $s_0 = -\infty$, $p_0 = 1$

Tirage par sous-ensembles (ou tirages particulaires) II

- On tire un échantillon $(\mathbf{X}_1, \dots, \mathbf{X}_N)$ de taille N selon la loi de densité $p_{\mathbf{X}}$ (les N particules) et on calcule l'échantillon $(Y_k = f(\mathbf{X}_k))_{k=1,\dots,N}$ correspondant. On pose $s_1 = \min(s, Y_{(\lceil \alpha N \rceil)})$ où $Y_{(\lceil \alpha N \rceil)}$ est le quantile empirique de Y de niveau α .
- Tant que $s_i < s$, faire :

Estimation On estime la probabilité $p_i = \mathbb{P}(f(\mathbf{X}) \geq s_i)$ par $p_i = (1 - \alpha)p_{i-1}$

Sélection On extrait les $N - \lceil \alpha N \rceil$ points $\tilde{\mathbf{X}}_k^i = \mathbf{X}_k^i$ tels que $Y_k^i \geq s_k$. Par construction, ces $N - \lceil \alpha N \rceil$ points sont distribués selon la loi de \mathbf{X} sachant que $Y \geq s_i$ **et sont indépendants si les \mathbf{X}_k le sont**

Clonage On clone $\lceil \alpha N \rceil$ de ces points par échantillonnage uniforme (chaque $\tilde{\mathbf{X}}_k^i$ est choisi avec la même probabilité). On obtient ainsi une population de N points distribués selon la loi de \mathbf{X} sachant que $Y \geq s_i$ **non indépendants**

Vérification On fait évoluer N chaînes de Markov partant de chacun des $\tilde{\mathbf{X}}_k^i$ et telles que leur loi stationnaire est la loi de \mathbf{X} sachant que $Y \geq s_i$ pour constituer la population $\mathbf{X}_1^{i+1}, \dots, \mathbf{X}_N^{i+1}$

- Si $s_i > s$, on ajuste l'estimation de p par :

$$\hat{p}_N = p_i \times \frac{1}{N} \sum_{j=1}^N \mathbb{1}_{f(\mathbf{X}_j^i) \geq s} \quad (13)$$

On remarque les faits suivants :

Tirage par sous-ensembles (ou tirages particulaires) III

- Le nombre d'itérations de l'algorithme est aléatoire, d'espérance $\lfloor p/(1 - \alpha) \rfloor + 1$
- Les probabilités conditionnelles intermédiaires sont estimées de manière déterministe par $(1 - \alpha)$

Cet algorithme a d'abord été proposé dans la communauté mécanicienne probabiliste sous le nom de Subset Sampling, avant d'être redécouverte sous le nom de Sequential Splitting par la communauté mathématique.

Dans le cas où les seuils sont fixés à priori, sous l'hypothèse que les étapes de diversification parviennent à restaurer l'indépendance au sein de la population, l'algorithme produit un estimateur non biaisé de p et asymptotiquement Gaussien. A nombre de seuils intermédiaires fixé, la variance est minimale lorsque toutes les probabilités conditionnelles sont égales, ce qui motive le fait de choisir des quantiles empiriques de même niveau dans la version adaptative.

Par ailleurs, la variance asymptotique s'écrit :

$$\text{Var} [\hat{p}_N] \simeq \frac{p}{N} \left(\frac{\alpha \log p}{(1 - \alpha) \log(1 - \alpha)} \right) \quad (14)$$

ce qui motive à prendre α le plus petit possible. Dans le contexte adaptatif, cela revient à prendre $\alpha = 1/N$.

Tirage par sous-ensembles (ou tirages particulaires) IV

L'algorithme résultant a été proposé par A. Guyader en 2012 (Last Particle Algoritm). Par rapport à l'algorithme précédent, les modifications sont :

- On prend $\alpha = 1/N$
- A l'étape de sélection, on garde toutes les particules sauf celle de plus petite valeur Y
- La particule éliminée est remplacée par l'état final d'une chaîne de Markov partant d'une des particules retenues choisie selon la loi uniforme sur ces particules
- L'arrêt se produit à l'itération M telle que $M = \max\{i : s_i \leq s\}$ et l'estimateur vaut alors $\hat{p}_N = \left(1 - \frac{1}{N}\right)^M$

L'analyse de la convergence de cet algorithme donne que M est distribué selon la loi de Poisson $\mathcal{P}(-N \log p)$, d'où la loi discrète exacte de \hat{p}_N :

$$\mathbb{P}\left(\hat{p}_N = \left(1 - \frac{1}{N}\right)^m\right) = \frac{p^N(-N \log p)}{m!} \quad (15)$$

dont on peut tirer un interval de confiance exact. Cette méthode atteint la variance optimale parmis toutes les méthodes de cette classe.

Algorithmes de Quasi-Monte Carlo I

Le calcul d'espérance est un calcul d'intégrale pour lequel il existe d'autres outils théoriques que la loi forte des grands nombres pour justifier la convergence, telle l'inégalité de Koksma-Hlawka :

$$\left| \int_{[0,1]^d} f(u) du - \frac{1}{N} \sum_{i=1}^N f(x_i) \right| \leq V(f) D_N \quad (16)$$

pour toute fonction f définie sur $[0, 1]^d$ de variation de Hardy-Krause $V(f)$ finie et pour toute suite de points $(x_i)_{i=1,\dots,N}$ de $[0, 1]^d$ de discrépance D_N .

Celà conduit aux méthodes dite Quasi-Monte Carlo, l'intégrale définissant l'espérance étant approchée par la moyenne des valeurs prises par l'intégrande sur un ensemble de points. **La différence fondamentale est que la suite de point n'est pas une suite aléatoire**

Algorithmes de Quasi-Monte Carlo II

La **discrépance** D_N d'une suite de points dans $[0, 1]^d$ est définie par :

$$D_N = \sup_{Q \subset [0,1]^d} \left| \frac{\text{card } Q}{N} - \text{vol}(Q) \right| \quad (17)$$

Il s'agit de comparer la proportion de points tombant dans le sous-ensemble Q de $[0, 1]^d$ au volume de Q en norme sup.

La **variation de Hardy-Krause** $V(f)$ d'une fonction est définie en plusieurs étapes :

- Soit f une fonction définie sur $[0, 1]^d$. On note :

$$\Delta_{h_k}(f, x) = f(x_1, \dots, x_k + h_k, \dots, x_d) - f(x_1, \dots, x_k, \dots, x_d) \quad (18)$$

puis, de façon récursive :

$$\Delta_{h_1, \dots, h_k}(f, x) = \Delta_{h_k}(\Delta_{h_1, \dots, h_{k-1}}(f, x)) \quad (19)$$

- On se donne des subdivisions π_k définies par $0 = t_k^1 < \dots < t_k^{N_k+1} = 1$ sur chaque dimension $k = 1, \dots, d$ et les pas associés $h_k^i = t_k^{i+1} - t_k^i$.

Algorithmes de Quasi-Monte Carlo III

- La variation au sens de Vitalli de f est définie par :

$$V^n(f) = \sup_{(\pi_1, \dots, \pi_d)} \sum_{k=1}^d \sum_{i_k=1}^{N_k} \left| \Delta_{h_1, \dots, h_k} \left(f, (x_1^k, \dots, x_k^{i_k}) \right) \right| \quad (20)$$

- La variation au sens de Hardy Krause de f est définie par :

$$V(f) = \sum V^d(f) \quad (21)$$

où la somme est faite sur toutes les faces de tous les sous-intervalles de $[0, 1]^d$ de dimension inférieure ou égale à d .

Remarques :

- Cette grandeur est difficile à calculer en pratique
- Des fonctions dérivables peuvent être à variation infinie
- Elle est imposée par f , on ne peut que chercher à réduire D_N pour approcher l'intégrale, d'où la notion de **suite à discrépance faible**

Suites à discrépance faible I

- On sait construire des suites (x_1, \dots, x_N) telles que D_N est majorée au moins par $C \frac{(\log N)^d}{N}$, où la constante C dépend de la suite mais pas de N
- On sait minorer D_N :

$$D_N \geq C(d) \frac{(\log N)^{(d-1)/2}}{N} \quad (22)$$

- On conjecture que

$$D_N \geq C(d) \frac{(\log N)^{d-1}}{N} \quad (23)$$

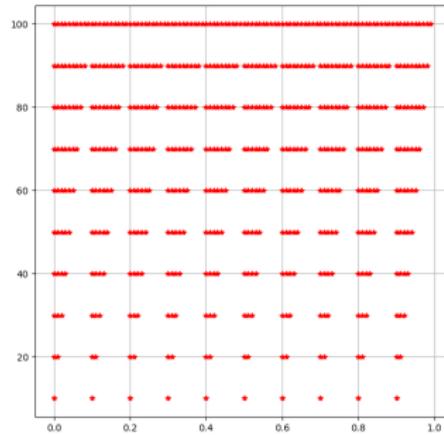
pour toute suite finie, et qu'il existe des suites avec

$$D_N \geq C(d) \frac{(\log N)^d}{N} \quad (24)$$

Suites à discrépance faible II

Exemples de suites à faible discrépance :

- Suite de Van Der Corput ($d = 1$) : on se donne une base b , puis on construit x_1, \dots, x_N de la manière suivante. Pour $n \in \{1, \dots, N\}$, si $n = \sum_{i=0}^{L-1} n_i b^i$ est l'écriture de s en base b , avec $0 \leq n_i \leq b - 1$, on pose $x_n = \sum_{i=0}^{L-1} n_i b^{-i-1}$. Il s'agit du nombre de $[0, 1]$ dont les chiffres en base b sont dans l'ordre inverse de ceux de n .



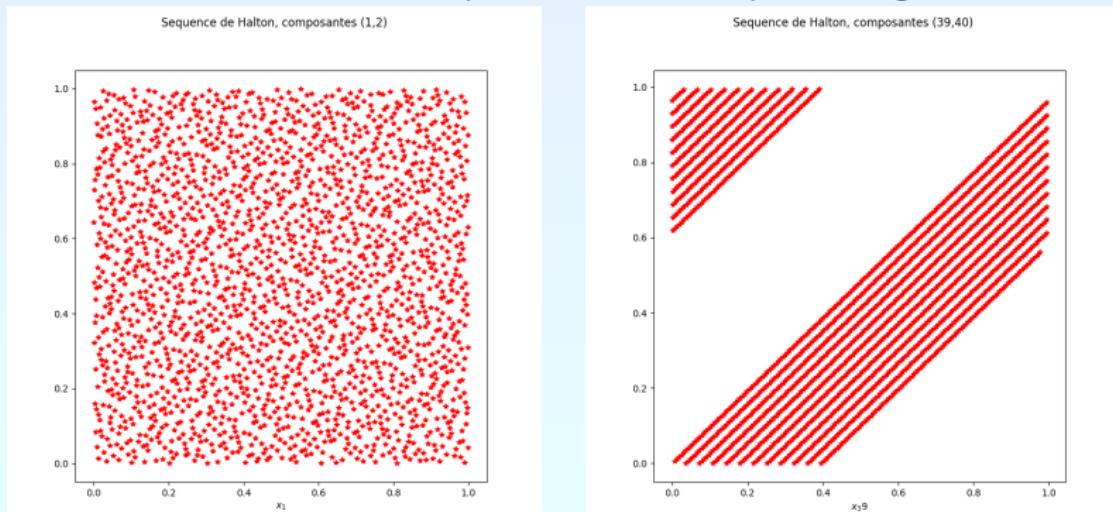
Suites à discrépance faible III

- Suite de Halton ($d > 1$) : on pose $x_n = (x_n^{p_1}, \dots, x_n^{p_d})$ où $p_1 = 2, p_2 = 3, \dots, p_d$ = le d -ème nombre premier, $x_n^{p_j}$ étant le n -ème terme de la suite de Van Der Corput associé à la base p_j .

Remarque pour la grande dimension, comme le j -ème nombre premier p_j vérifie :

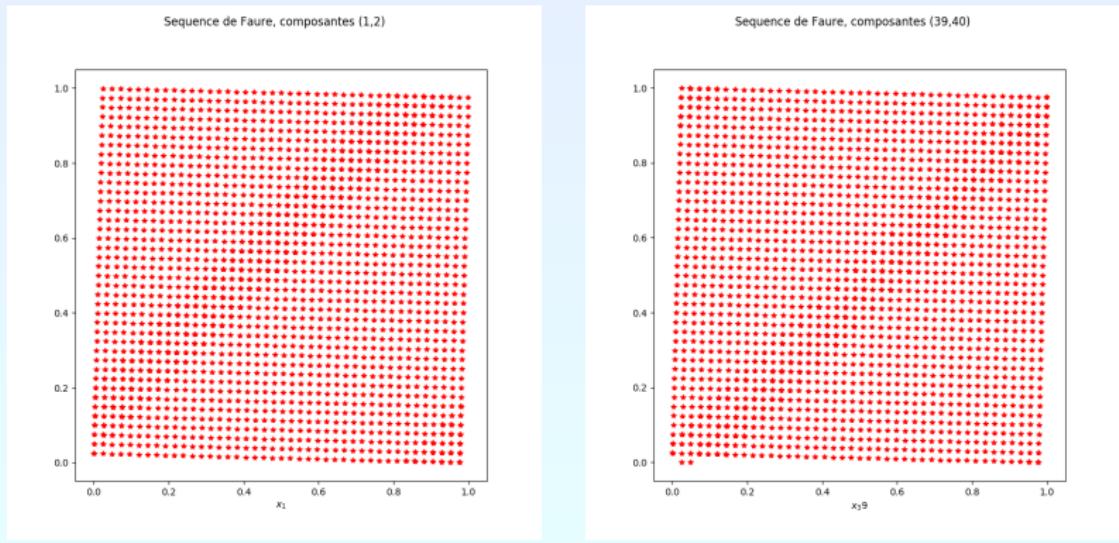
$$j(\log j + \log \log j - 1) < p_j < j(\log j + \log \log j) \quad (25)$$

la j -ème composante sera peuplée séquentiellement par des fractions de grand dénominateur, laissant des trous importants si N n'est pas assez grand.



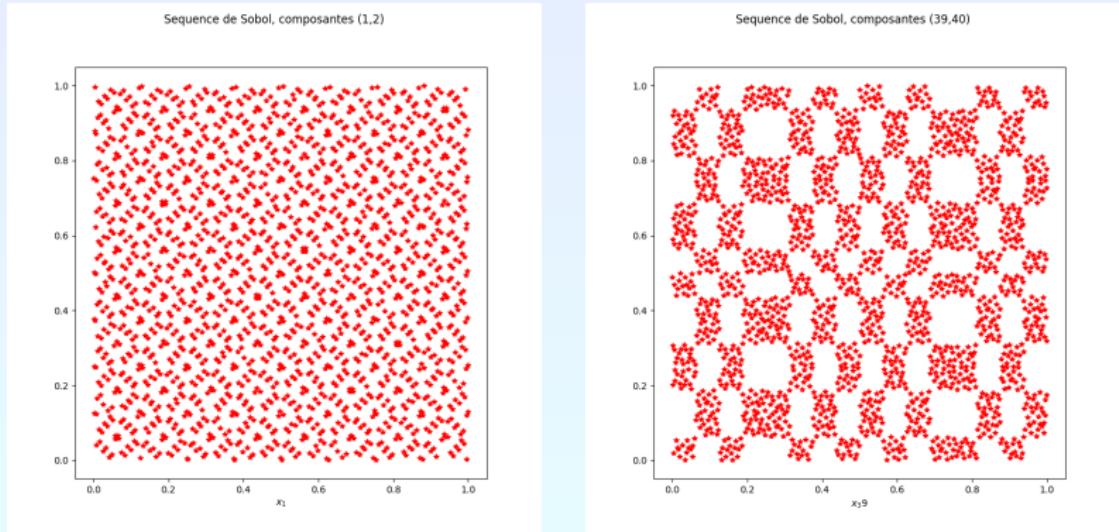
Suites à discrépance faible IV

- Suite de Faure ($d > 1$) : on pose $x_n = (\sigma_1(x_n^p), \dots, \sigma_d(x_n^p))$ où p est un nombre premier supérieur à d et $(\sigma_i)_{i=1,\dots}$ sont des permutations de $\{0, \dots, p-1\}$ en nombre au moins égal à N (donc $p! \geq N$).
Les performances de cette suite se dégradent moins vite avec la dimension que celles de la suite de Halton, car elle fait intervenir un unique nombre premier asymptotiquement négligeable devant p_d quand d augmente.



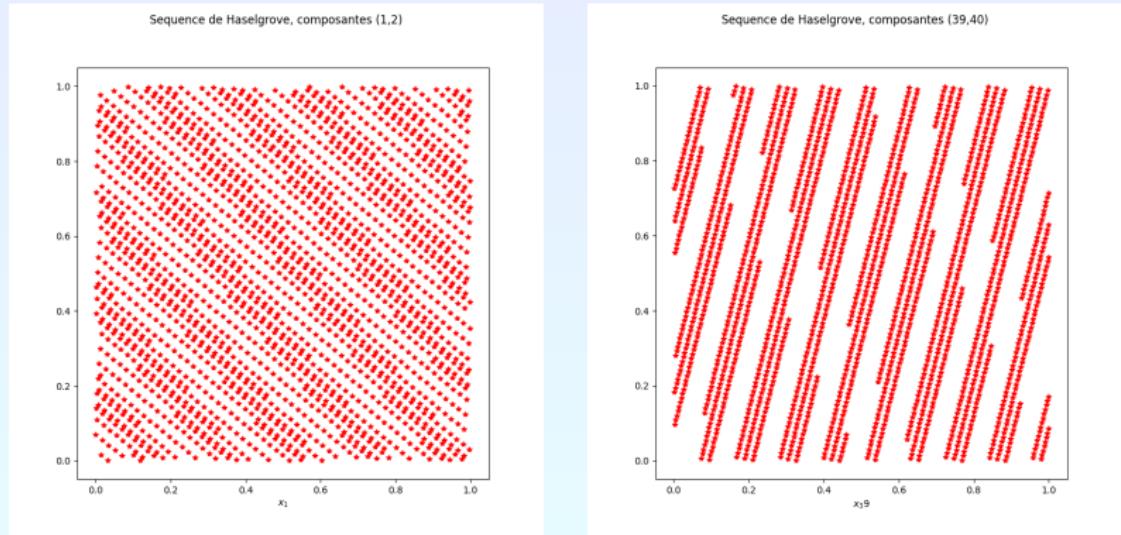
Suites à discrépance faible V

- Suite de Sobol ($d > 1$) : basé sur des polynômes irréductibles de $(\mathbb{Z}/2\mathbb{Z})^d$, produisant des points de discrépance comparable à celle des suites de Faure mais dont les performances se dégradent également avec la dimension.



Suites à discrépance faible VI

- Les suites algébriques : soient $\alpha_1, \dots, \alpha_d$ d nombres irrationnelles linéairement indépendants sur \mathbb{Q} , on pose $(u_1^n, \dots, u_d^n) = (n\alpha_1 \bmod 1, \dots, n\alpha_d \bmod 1)$. Par exemple, la suite de Haselgrove consiste à prendre $\alpha_k = \sqrt{p_k}$ où p_k est le k -ème nombre premier.



Ces suites sont faciles à générer mais présente une discrépance supérieure à celle de la suite de Halton.

Suites à discrépance faible VII

- Les suites (t, s) qui se proposent de contrôler l'équi-répartition des points sur les faces de dimensions faibles de $[0, 1]^s$ ici la dimension est s , pas d !. Soit $b \geq 2$ un entier, un intervalle élémentaire de $[0, 1]^d$ est un interval de la forme $\prod_{i=1}^s [a_i b^{-d_i}, (a_i + 1)b^{-d_i}]$ où a_i et d_i sont des entiers positifs tels que $0 \leq a_i < b^{d_i}$ pour $1 \leq i \leq s$.

Etant donnés des entiers t, m tels que $0 \leq t \leq m$, un réseau (t, m, s) en base b est un ensemble de dimension s comportant b^m points tels que tout interval élémentaire en base b de volume b^{t-m} contient exactement b^t points.

Une suite s -dimensionnelle $(\mathbf{X}_n)_{n \geq 1}$ de $[0, 1]^s$ est une (t, s) suite si le sous-ensemble $\{\mathbf{X}_n : kb^m < n \leq (k+1)b^m\}$ est un réseau (t, m, s) en base b pour tout entier $k \geq 0$ et $m \geq t$.

Revisiter les algorithmes de simulation I

- Le calcul d'espérance par échantillonnage correspond à remplacer une intégration par rapport à une mesure continue par une sommation par rapport à une mesure discrète, cette mesure discrète convergeant vers la mesure continue lorsque la discréttisation se raffine ;
- La convergence et son contrôle sont faits via la loi forte des grands nombres et le théorème de la limite centrale dans le cadre probabiliste, par l'inégalité de Koksma-Hlawka dans le contexte des suites à discrépance faible. Dans tous les cas, la distribution discrète tend vers la distribution continue au sens de la convergence faible

→ on peut voir les suites à discrépance faible comme une alternative aux générateurs pseudo-aléatoires et revisiter tous les algorithmes de simulation

C'est en particulier le cas pour l'algorithme de Monte Carlo multiniveaux, avec une amélioration de la vitesse de convergence constatée dans de nombreuses applications

- Les bornes sur la discrépance font apparaître un facteur exponentiel en la dimension, laissant supposer de mauvaises performances en grande dimension. Ce n'est pas ce qui est constaté en pratique dans de nombreux cas, d'où l'introduction d'une notion de dimension effective d'un problème

Revisiter les algorithmes de simulation II

- Il est tout à fait possible de coupler une approche par suites à discrépance faible et par échantillonnage : les méthodes **QMC randomisées**. Au lieu de considérer des points (u_1, \dots, u_d) de $[0, 1]^d$ générés via une suite à discrépance faible, on considère $(u_1 + V_1 \bmod 1, \dots, u_d + V_d \bmod 1)$ où les variables aléatoires V_1, \dots, V_d sont iid de loi commune uniforme sur $[0, 1]$. On obtient ainsi des estimateurs de très faible variance dont on peut effectivement contrôler la convergence via le théorème de la limite centrale.