L'aléatoire et les ordinateurs

Pierre-André Zitt

17 janvier 2017

> Random number generation is too important to be left to chance.
>
> R. Coveyou

> Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin.
>
> J. von Neumann

Objectifs

— Comprendre les objectifs de la génération de nombres aléatoires

— Comment tester un générateur ?

— Comprendre l'exemple du générateur linéaire.

1 Introduction : l'expérience du nombre de séries.

On demande à N personnes de « simuler » 15 tirages de pile ou face successifs.

Question pratique

Le tirage est-il raisonnable ? Ressemble-t-il à ce qu'on obtiendrait en lançant une pièce équilibrée 15 fois de suite ?

Pour répondre à cette question, on peut :

1. Choisir une certaine quantité dépendant des tirages : nombre de piles, longueur de la première série de piles, nombre de séries, ...
2. Calculer sa distribution théorique (quand les tirages proviennent de pile ou face équilibrés consécutifs),
3. Comparer la distribution théorique et la distribution observée.

Prenons un exemple simple. Pour $i = 1, ..., N$ on note $L_i = 1$ la longueur de la plus longue suite de piles consécutifs dans le tirage proposé par la i^e personne : pour le tirage

FPPPFPFPPFPPFPF

par exemple, la plus longue suite est de longueur 3.

La quantité L_i prend ses valeurs entre 0 et 15. Si le tirage provient d'une pièce équilibrée on peut déterminer la loi de L_i en calculant les probabilités d'obtenir chaque valeur. On obtient le tableau suivant :

ℓ	≤ 2	3	4	5	≥ 6
$\mathbf{P}[L = \ell]$	0.32	0.30	0.19	0.10	0.09

Question mathématique

Comment comparer la distribution observée et la distribution théorique ?

Le meilleur résultat serait obtenu en utilisant un *test statistique d'adéquation* comme le test du χ^2. En première approche, remarquons simplement que d'après le tableau, il y a une probabilité $p \approx 0.62$ que la plus longue suite soit de longueur inférieure ou égale à 3. Cette proportion est-elle respectée sur les N individus, *modulo* les flucutations d'échantillonage ?

On note $X_i = 1$ si $L_i \leq 3$, $X_i = 0$ sinon, et $S_n = \sum X_i$. Si les tirages sont parfaits, X_i suit une loi de Bernoulli de paramètre p. Soit α un niveau de risque (par exemple 0.05) et z_α le quantile d'ordre $1 - \alpha/2$ de la loi normale : si Z suit une loi normale alors $\mathbf{P}[|Z| \leq z_\alpha] = 1 - \alpha$.

Théorème

Un intervalle de fluctuation Soit $e = z_\alpha \sqrt{p(1-p)}/\sqrt{N}$. Alors

$$\mathbf{P}[S_N/N \in [p - e, p + e]] \approx 1 - \alpha.$$

Application numérique : pour $N = 30$ personnes, on trouve les intervalles suivants.

α	0.05	0.01
Intervalle sur les proportions	$[0.45, 0.80]$	$[0.40, 0.86]$
Intervalle sur les effectifs	$[13, 24]$	$[12, 26]$

Autrement dit, dans 95% des cas, au moins 6 tirages sur les 30 donnent une série de longueur supérieure ou égale à 4. Dans 99% des cas, au moins 4 tirages sur les 30 sont dans ce cas.

En pratique les êtres humains ont tendance à produire des suites « trop réparties » : en équilibrant trop les piles et les faces, en les alternant trop souvent, ...

— Nous sommes de très mauvais générateurs de nombres aléatoires ;

— ce manque de qualité peut se mettre en évidence par des tests statistiques.

2 Pourquoi l'aléatoire ?

L'ordinateur est habituellement vu comme une machine entièrement déterministe (un programme répété dans les mêmes conditions donnera le même résultat). Dans quels cadres peut-on avoir besoin que l'ordinateur produise des résultats aléatoires ?

La simulation De nombreux phénomènes réels, même quand ils sont déterministes, sont trop compliqués pour être représentés fidèlement dans un modèle informatique. Il est alors naturel de prendre un modèle aléatoire. Si l'on veut étudier le modèle il faut savoir le simuler.

Exemples : dynamique moléculaire ; modèles économiques multi-agents ; modèles de croissance (de villes, d'arbres, ...) pour des univers virtuels, pour de la génération de textures...

L'optimisation combinatoire Pour répondre à des questions *déterministes* il peut être utile de faire des choix *aléatoires.* : pour savoir si un grand entier est premier ou non, pour savoir si une position est bonne ou non dans un jeu (go, jeu de cartes, ...), pour résoudre approximativement des problèmes d'optimisation combinatoire (voyageur de commerce, ...)

Le calcul déterministe pour des problèmes complexes Dans le même ordre d'idées, une des applications les plus courantes des générateurs aléatoires est l'utilisation de *méthodes de Monte-Carlo* pour calculer des intégrales / des espérances de quantités aléatoires (en finance, en physique, ...)

La cryptographie C'est un domaine omniprésent : chaque communication entre appareils électroniques utilise de nombreux protocoles cryptographiques. Les algorithmes de chiffrement et déchiffrement sont déterministes mais la quasi totalité de ces algorithmes nécessite d'engendrer des nombres aléatoires.

En pratique, on procède toujours en deux étapes :

1. on construit un générateur de nombres (pseudo-)aléatoires *uniformes* : une fonction rand qui renvoie un nombre « aléatoire » dans un ensemble continu (souvent l'intervalle $[0,1]$) ou discret (souvent l'ensemble $\{0,1\}$).
2. À partir d'une source de hasard uniforme, on la transforme pour simuler la variable aléatoire d'intérêt.

Ici on ne s'intéressera qu'à la première étape.

Question pratique

Comment programmer un ordinateur pour qu'il renvoie une suite (pseudo)-aléatoire uniforme ?

3 Les sources aléatoires uniformes en informatique

3.1 Les deux modèles de hasard « uniforme »

Les deux modèles usuels d'aléatoire uniforme sont :

— la loi uniforme discrète ; dans le cas le plus simple où il n'y a que deux possibilités c'est la loi du jeu de pile ou face équilibré ;

— la loi uniforme continue (sur $[0,1]$).

Rappelons que les nombres réels peuvent être écrits en base 2 plutôt qu'en base 10 (on parle de développement dyadique) :

Théorème

Écriture dyadique Soit $u \in [0,1)$. Il existe une unique suite $(x_n)_{n\geq 1}$ qui vérifie les conditions suivantes :

1. $x_n \in \{0,1\}$,
2. x_n n'est pas constante égale à 1 à partir d'un certain rang, autrement dit x_n prend la valeur 0 une infinité de fois,
3. $u = \sum_n x_n 2^{-n}$.

La suite x_n est appelée décomposition dyadique de u, c'est l'écriture du réel u en base 2. On note $u = (0.x_1x_2\cdots x_n\cdots)_2$.

Fixons un début de développement binaire, par exemple $(0.00101)_2$. Les réels dont le développement dyadique commence ainsi sont ceux compris entre $(0.00101)_2 = 1/8 + 1/32$ et $(0.00110)_2 = 1/8 + 1/16$. La probabilité pour une loi uniforme de tomber dans cet intervalle est $1/32 = 1/2^5$. Ce raisonnement est valable pour toute suite de 5 symboles dans $\{0,1\}$:

autrement dit, en regardant les 5 premiers chiffres du développement dyadique d'un réel pris uniformément dans $[0,1]$, chacune des 2^5 suites possibles est choisie avec la même probabilité.

Ceci donne plus généralement une correspondance remarquable entre l'aléatoire discret et l'aléatoire continu :

Théorème

Soit U une variable aléatoire de loi uniforme sur $[0,1]$. Soit $(X_n)_{n\in\mathbb{N}}$ la réprésentation dyadique de U. Alors $(X_n)_{n\in\mathbb{N}}$ est une suite de variables de Bernoulli indépendantes de paramètre $1/2$.

Réciproquement si $(X_n)_{n\in\mathbb{N}}$ est une suite iid de Bernoulli de paramètre $1/2$, alors la série $\sum_n \frac{X_n}{2^n}$ converge presque sûrement vers une variable $U = (0.X_1X_2\cdots)_2$, dont la loi est uniforme sur $[0,1]$.

Cette correspondance théorique parfaite entre les deux modèles d'aléatoire élémentaires discrets et continus ne se traduit malheureusement pas par une correspondance pratique : les générateurs de bits aléatoires usuels ne donnent pas nécessairement de bons réels aléatoires, et réciproquement.

3.2 Ce qu'on peut attendre d'un générateur de nombres aléatoires

Un bon générateur aléatoire devrait être indistingable du générateur idéal (la « suite de variables de Bernoulli indépendantes »). Cela se traduit par le fait qu'il devrait vérifier les propriétés suivantes :

Équirépartition Pour tout intervalle $[a,b] \subset [0,1]$, la fraction du nombre de points qui tombe dans $[a,b]$ doit « converger » vers $b-a$.

$$\frac{1}{n}\sum_{k=1}^{n} \mathbf{1}_{u_i\in[a,b]} \approx (b-a).$$

Équirépartition d'ordre supérieur Pour toute pavé $P = \prod_{i=1}^{D}[a_i,b_i]$,

$$\frac{1}{n}\sum \mathbf{1}_{(u_{nD+1},\ldots,u_{nD+D})\in P} \approx \prod_i (b_i-a_i).$$

Et plus généralement... Pour tout k et toute fonction « test » $f : [0,1]^k \to \mathbb{R}$, on peut comparer $f(U_1,\ldots,U_k)$ pour le générateur aléatoire et pour une vraie suite de variables uniformes pour obtenir un test du générateur.

Notons que ces propriétés idéales ne seront jamais vraiment atteintes. La formulation de l'équirépartition écrite ci-dessus n'est par exemple jamais vérifiée si on interprète $\approx$ comme une limite quand n tend vers l'infini, puisqu'un générateur informatique ne peut renvoyer qu'un nombre fini de valeurs possibles, donc certains intervalles $[a,b]$ ne seraont jamais visités. On demandera alors plutôt que chacune des valeurs possibles apparaisse avec la même fréquence.

Pour certaines utilisations en cryptographie on demande également :

Imprévisibilité Même en ayant accès à tous les nombres $(u_1,\ldots,u_n)$, il n'est pas possible de prévoir u_{n+1}.

Naturellement vérifier ces critères peut demander des algorithmes plus complexes et plus gourmands en temps de calcul et en mémoire ; suivant les applications envisagées il faut faire des choix...

3.3 « Vrai » hasard

Une façon d'obtenir un nombre aléatoire est de le « capter » dans l'environnement, en enregistrant les fluctuations thermiques, les mouvements de la machine, les motifs d'interférence radio...

Cette méthode est souvent (relativement...) lente ; elle est utilisée principalement pour les besoins en cryptographie, et est intégrée dans des composants électroniques dédiés.

Les nombres fournis ainsi sont également souvent utilisés pour initialiser ou réinitialiser les générateurs pseudo-aléatoires décrits plus bas.

3.4 Pseudo-aléatoire : générateurs à congruence linéaire

Les « générateurs aléatoires » des langages/environnement de programmation classiques sont *pseudo-aléatoires*. Ce sont des processus *déterministes*, qui partagent certains aspects des suites aléatoires. L'exemple classique est le *générateur à congruence linéaire*.

Soit a, c, m trois entiers. Soit x_n la suite d'éléments de $\{0, ..., m-1\}$ définie par la relation de récurrence

$$x_{n+1} = ax_n + c \mod m$$

et soit $u_n = x_n/m$. Alors pour des valeurs bien choisies de a, c, m, la suite (u_n) « ressemble » à une suite de variables uniformes sur $[0, 1]$.

Exercice 1 — Exemple. Pour $m = 8$ on considère la récurrence linéaire $u_{n+1} = au_n + c$ avec $a = 4$, $c = 1$.

1. Calculer les premiers termes de la suite en partant de $x_0 = 0$. Qu'observe-t-on ? Donner la période de la suite.
2. Que se passe-t-il pour les autres valeurs de x_0 ?
3. Mêmes questions si l'on choisit $a = 3$, $c = 1$, puis $a = 5$, $c = 1$.
4. Montrer que l'application qui envoie x_n sur x_{n+1} est bijective si et seulement si a est premier avec m.

La valeur initiale x_0 est appelée « graine » (*seed*). Comme x_n ne peut prendre qu'au maximum m valeurs, elle est nécessairement périodique à partir d'un certain rang ; on cherche usuellement à avoir une période élevée (si possible proche de m lui même).

Si $a = 1$ on a une suite arithmétique (modulo m), de période maximale si et seulement si c est premier avec m. On se doute, et on peut montrer, que ce n'est pas un très bon choix. On suppose donc $a > 1$.

Il est relativement facile de déterminer si un choix de paramètres conduit à une période maximale. Montrons-le dans le cas particulier [1] où m est une puissance de 2.

Théorème

CNS de période maximale. Soit e un entier et $m = 2^e$. La période du générateur est maximale égale à m si et seulement si c est impair et $a \equiv 1[4]$.

1. Le lecteur curieux pourra trouver l'énoncé général (pour m quelconque) et sa preuve dans *The art of Computer programming II* de D. Knuth, section 3.2.1.2, théorème A.

Lemme

Si $e \geq 2$,

$$\begin{cases} x \equiv 1[2^e], \\ x \not\equiv 1[2^{e+1}] \end{cases} \implies \begin{cases} x^2 \equiv 1[2^{e+1}], \\ x^2 \not\equiv 1[2^{e+2}] \end{cases}$$

Exercice 2 — Preuve du résultat.

1. Montrer le lemme.
2. Dans les questions 2 à 5, on suppose le générateur périodique de période m, et on fixe $x_0 = 0$. Montrer que $x_k = y_k c \mod m$, où y_k est l'entier $\frac{a^n-1}{a-1}$.
3. Montrer que c est impair.
4. Montrer que les y_k sont non-nuls modulo m pour $1 \leq k < m$, et que $y_m = 0$. En déduire que $a^m \equiv 1[m]$ puis que a est impair.
5. (Plus difficile) Montrer que a ne peut pas être congru à 3 modulo 4.
6. On suppose maintenant c impair et $a \equiv 1[4]$. Montrer que a s'écrit $2^f q + 1$ avec q impair et $f \geq 2$.
7. Montrer que pour tout $g \geq 0$, $a^{2^g} \equiv 1[2^{f+g}]$, $a^{2^g} \not\equiv 1[2^{f+g+1}]$, et en déduire qu'il existe r impair tel que
$$y_{2^g q} = r 2^g.$$
8. Montrer que la période du générateur divise m.
9. Montrer que la période vaut m.

Remarquons qu'avoir la période maximale n'est pas une garantie de qualité. Une suite de période maximale est équirépartie, mais elle ne l'est pas forcément à l'ordre supérieur. Pour s'en convaincre, on peut représenter graphiquement les points (u_n, u_{n+1}) : pour un tirage parfait on devrait obtenir des points de loi uniforme dans le carré, en pratique on observe des motifs plus ou moins nets suivant les valeurs des paramètres (voir la figure page suivante pour une illustration).

Ces défauts se retrouvent dans tous les générateurs linéaires. Pour les contourner on peut par exemple...

— Remplacer la sortie $u_n = x_n/m$ par une fonction plus compliquée/imprévisible ;

— Remplacer la récurrence linéaire par une récurrence non-linéaire (par exemple $x_{n+1} = x_n^2 \mod m$: c'est l'algorithme de Blum Blum Shub).

Les générateurs correspondants sont (parfois) meilleurs, mais souvent plus lents et plus difficiles à analyser mathématiquement.

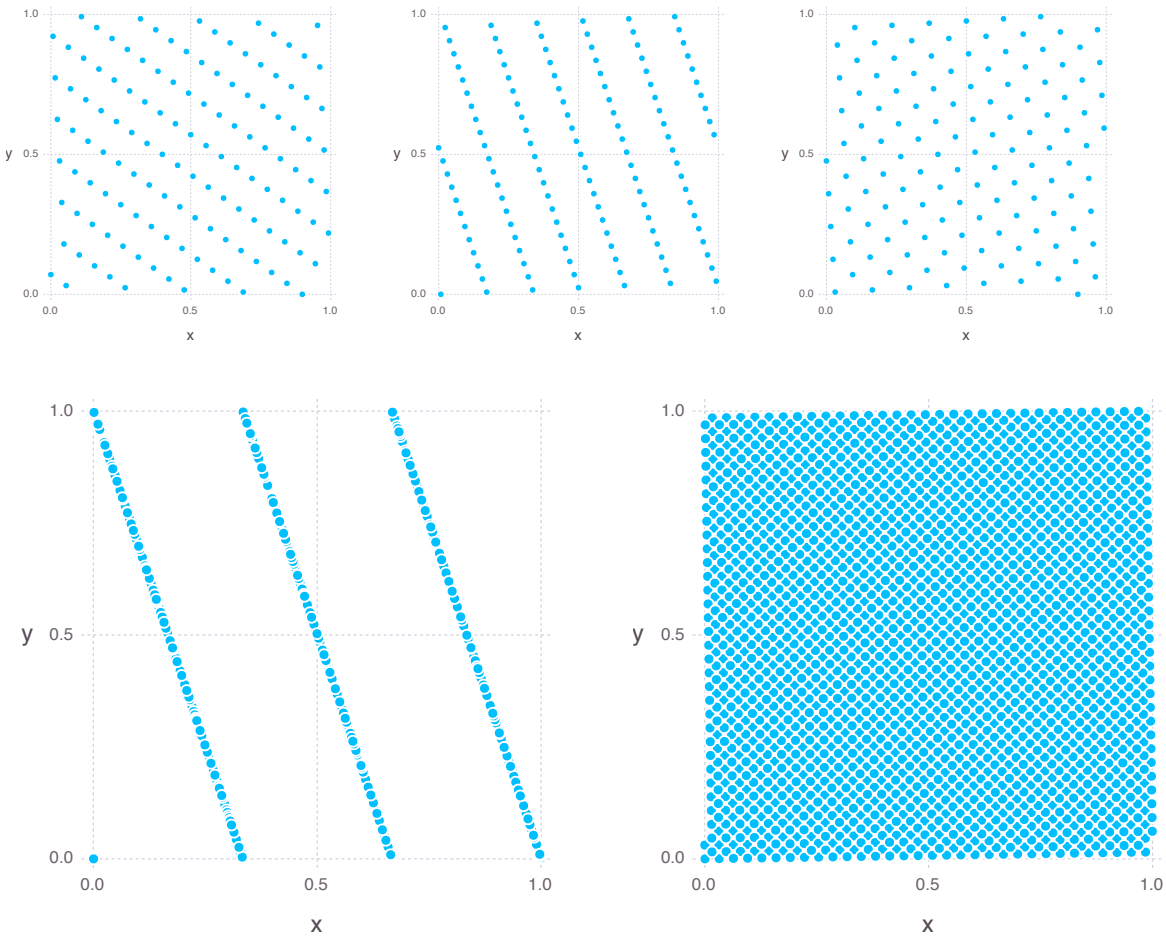
3.5 Pseudo-aléatoire : linear shift register

Plutôt que de travailler sur une suite récurrente d'entiers modulo m comme précédemment, on regarde une suite récurrente sur les mots à d lettres sur $\{0, 1\}$.

On part d'un mot à d lettres. À chaque étape :

— on renvoie le bit de gauche comme nouveau "bit aléatoire", on décale tous les autres vers la gauche et on met 0 à droite.

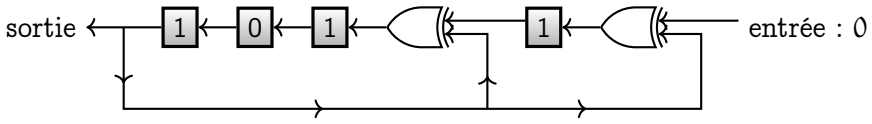
— Si le bit qui « vient de sortir » vaut 1, on inverse les bits en certaines positions fixées à l'avance.



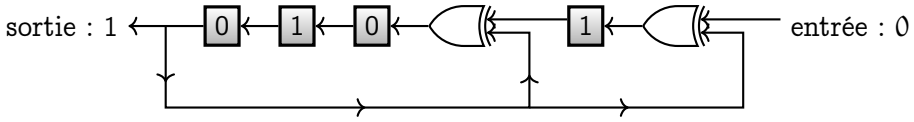
Tracé des points (u_k, u_{k+1}) sur une période du générateur. Sur la première ligne on fixe $m = 128$, et on étudie différentes valeurs pour (a, c) : $(101, 115)$, $(21, 1)$, $(17, 115)$. La qualité de l'équidistribution dans le carré est très sensible à ces choix.
Dans la deuxième ligne on fait le même constat pour $m = 2048$, $c = 1$ et deux valeurs de a : 1365 à gauche et 65 à droite.
Notons que dans tous les cas les u_k sont équirépartis dans $[0, 1]$.

FIGURE 1 – Répartition des points dans le carré pour différents générateurs à récurrence linéaire

Prenons un exemple : partons du mot à $d = 4$ lettres (1011), et supposons que quand la sortie vaut 1 on inverse les deux bits les plus à droite. Cette inversion peut être modélisée par le fait qu'à chaque étape, le 3è bit est obtenu en faisant la somme modulo 2 des anciens premiers et quatrième bit ; de même le quatrième bit est obtenu en faisant la somme modulo 2 du premier bit et de 0. On peut représenter la méthode par le schéma suivant :



où le symbole  représente une « porte "ou exclusif" », qui fait la somme modulo 2 de ses deux entrées. Après une itération, on obtient par exemple l'état suivant :



On obtient la suite de résultats

$$
\begin{array}{cccc}
1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0
\end{array}
$$

avant de retomber sur l'état initial. Là encore la méthode a une interprétation algébrique simple, ce qui permet de l'étudier en grand détail :

Théorème

Le circuit effectue la multiplication par X dans l'espace des polynômes à coefficients dans $\{0,1\}$ modulo P, où $P = X^d + \sum_{i=0}^{d-1} a_i X^i$ avec $a_i = 1$ si on inverse en position i.

Dans l'exemple $P = X^4 + X + 1$. En partant de l'état initial 1011, c'est-à-dire du polynôme $X^3 + X + 1$, on effectue la multiplication par X pour obtenir $X^4 + X^2 + X$. Comme on travaille modulo P, et que les coefficients sont eux-mêmes modulo 2, $X^4 = -X - 1 = X + 1 \mod P$. Donc $X^4 + X^2 + X = X + 1 + X^2 + X = X^2 + 1 \mod P$. L'état suivant est donc bien 0101.

Grâce à cette interprétation, on peut trouver par une étude algébrique des conditions suffisantes sur le polynôme P qui garantissent que le générateur engendre tous les $2^d - 1$ mots possibles.

C'est une combinaison de plusieurs de ces générateurs à registres qui est utilisée dans les systèmes de chiffrement A5/1 (utilisé pour la téléphonie GSM), E0 (pour Bluetooth) ou encore CSS (pour les DVD).

Le standard « Mersenne Twister », référence *de facto* pour les générateurs aléatoires (hors cryptographie), fonctionne comme une généralisation d'un LFSR. C'est celui qui est utilisé par défaut dans Python.

3.6 Bilan

Être imprévisible est finalement assez difficile, et l'étude et la création de générateurs aléatoires pour l'informatique est un sujet de recherche toujours actif actuellement. Ce besoin est particulièrement important en cryptographie : pour les besoins courants en simulation, de bonnes solutions existent en standard sur les logiciels classiques, même si la généralisation du travail sur les grandes masses de données peut demander de faire appel à de très nombreux nombres aléatoires.

L'étude de ces générateurs est un sujet d'autant plus riche qu'il est à la frontière entre l'informatique théorique, la cryptographie, l'algèbre, les probabilités et les statistiques.