Générateurs de nombres aléatoires

Algorithmes et Structures de Données II, GymInf

Juan-Carlos Barros, Yves Dethurens, Daniel Kessler et Jean-Francis Ravoux

10 juillet 2021

Table des matières

1 Introduction

1.1 Que veut-on simuler et pourquoi ?

— distributions aléatoires (bla)
— utilité directe (ex : jeus) et indirecte (ex : algos aléatoires)

1.2 Questions fondamentales et distinction entre TRNG et PRNG

Dans un article de 1955 [**VonNeumann**], Von Neumann pose les deux questions fondamentales concernant la génération de nombres aléatoires :

1. Comment peut-on produire une séquence de chiffres aléatoires ? (il est question de chiffres décimaux entre 0 et 9, mais la question est la même pour des bits 0 ou 1)
2. Comment peut-on produire des nombres réels répartis suivant une loi de distribution donnée ?

Pour la deuxième question, la réponse existe déjà à ce moment-là : en partant d'une distribution uniforme entre 0 et 1 (obtenue par exemple à partir d'une suite de bits aléatoires $(0.b_1b_2b_3\ldots)_{bin}$), en utilisant la fonction inverse de la distribution cumulée, ont peut reproduire n'importe quelle distribution continue. En effet, pour une distribution $f : I \to [0,1]$ donnée, où I est un intervalle où f est strictement positive, la distribution cumulée correspondante $c : x \mapsto \int_{-\infty}^{x} f(\xi)d\xi$ est bijective de I vers l'intervalle $[0,1]$. Son inverse $c^{(-1)}$ permettra donc de retrouver des valeurs de x distribuées selon f. Des méthodes plus efficaces existent pour des distributions particulières.

Pour la première question, deux approches sont possibles : partir d'un processus physique, ou bien utiliser une méthode arithmétique. Von Neumann considère que le hasard idéal sera trouvé dans des phénomènes nucléaires, dont on peut compter la parité en un laps de temps donné (par exemple : pair :0, impair :1), mais d'autres moyens plus abordables existent, comme nous le verrons dans la section 3.

Selon Von Neumann, "*anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin*". Cependant, même s'il était praticable d'utiliser toujours une méthode "physique", les méthodes arithmétiques ont le grand avantage d'être reproductibles : "*the real objection to this procedure is the practical need for checking computations*".

De nos jours, la considération de reproductibilité de suites aléatoires pour tester des algorithmes et méthodes numériques comportant une composante aléatoire demeure tout autant vraie. A contrario, lorsque des nombres aléatoires sont employés pour la cryptographie, on veut éviter le plus possible la reproductibilité. Dans ce dernier cas, on préfère autant que possible recourir exclusivement à la génération de "vrais" nombres aléatoires (True Random Number Generators, section 3). Cependant, ceux-ci sont coûteux en temps. C'est pourquoi dans tous les autres cas, on utilisera des méthodes arithmétiques pseudo-aléatoires (Pseudo-Random Number Generators) produisant de manière déterministe une suite de nombres uniformément distribués (section 2).

Finalement, Von Neumann nous met en garde sur le fait que dans les méthodes arithmétiques grossières qui étaient employées aux années 1950, une bonne part du "pseudo-hasard" provenait en réalité des erreurs d'arrondi, qui sont en l'occurrence très difficiles à étudier et donc diminuent la prédictibilité de la distribution des nombres produits. Des méthodes plus modernes devraient éviter cet écueil.

2 Générateurs de suites pseudo-aléatoires

2.1 Caractéristiques communes

2.1.1 la seed

Les suites de nombres pseudo-aléatoires fournissent une manière déterministe de passer d'un nombre à un autre, sans histoire (seul le nombre précédent est pris en compte). Cependant, il faut démarrer la suite avec un premier nombre : la *seed*. Celle-ci pourra être obtenue par une méthode de "vrai" hasard (cf. section 3) ou bien être imposée de manière à pouvoir reporduire une séquence pseudo-aléatoire donnée, par exemple à des fins de tests.

2.1.2 la période

Toutes les séquences pseudo-aléatoires sont en fait périodiques. Si par exemple on produit des nombres constitués de 32 bits, au maximum au bout de 2^{32} itérations on retombera sur la "seed" choisie au départ. Dans l'idéal, on ne voudrait pas y retomber avant. Une bonne méthode de génération de nombres pseudo-aléatoires devra donc garantis une période longue.

2.2 Générateurs linéaires congruents (LCG)

... (à remplir par JF)

2.3 Mersenne Twister et les LFSR

La méthode la plus employée de nos jours est en fait le **Mersenne Twister** [**MT**]. Une de ses particularités est de garder en mémoire à chaque étape non pas 1 mais 624 nombres de 32 bits. Chaque itération est un *twist* de ces 624 nombres, dont les variations permettent d'atteindre une période de $2^{19937} - 1$ (qui est un nombre de Mersenne, d'où le nom du procédé). Si on n'utilise que le premier des 624 nombres à chaque itération, on verra que le nombre considéré peut être parfois répété (mais l'état global des 624 nombres aura changé).

L'algorithme du Mersenne Twister est assez compliqué et hors de la portée de ce travail. Notons cependant qu'il se base sur les **Linear-Feedback Shift Register**, une classe d'algorithmes dont un autre représentant, le **xorshift**, est aussi employé pour la

génération de nombres pseudo-aléatoires. Il est pour l'instant moins courrant mais plus efficace que le Mersenne Twister. [1]

2.4 Vrai chaos déterministe

2.4.1 Pseudo-aléatoire avec suites chaotiques ?x

Le problème des générateurs basés sur la théorie des nombres, c'est qu'ils produisent des séquences périodiques, qui possèdent des propriétés qui rend la suite en partie prévisible, parce que les nombres générés sont dépendants de ceux qui les précèdent, ce qui ne se produit jamais dans une vraie suite aléatoire !

D'autres générateurs, basés sur la théorie du chaos, sont imprévisibles (par définition, voir plus bas), mais il est généralement plus difficile de garantir que ceux-ci ont une période longue, et ils sont moins répandus que les générateurs basés sur la théorie des nombres.

Le générateur de nombres pseudo-aléatoires présenté ci-dessous (Saito & Yamaguchi [**SY**]) utilise les deux théories pour produire des suites qui ressemblent à de vraies suites aléatoires non périodiques. Son coût computationnel est élevé, mais les séquences générées peuvent par exemple servir de référence pour des tests qualitatifs.

2.4.2 Point de départ : le décalage de Bernouilli

La relation $\alpha_{n+1} = (2\alpha_n)\ mod\ 1$ définit une suite où $\alpha_n \in [0; 1[$. En arrondissant les termes de cette suite, on obtient une séquence binaire pseudo-aléatoire.

Exemple : $\alpha_0 = 0.3$ donne la suite $\{\ 0.3\,;\ \mathbf{0.6}\,;\ 0.2\,;\ 0.4\,;\ 0.8\,;\ \mathbf{0.6}\,;\ \ldots\ \}$

ou la séquence 01001100110011001... qui est périodique et donc très prévisible.

Mais si α_0 est **irrationnel**, cette suite devient **non périodique**. Cela signifie qu'une infinitésimale variation de α_0 provoquera un changement radical à un moment de la séquence. Cette forte sensibilité aux conditions initiales est une caractéristique des fonctions chaotiques : au bout d'un certain temps, un phénomène chaotique devient imprévisible. Une loi déterministe va évidemment être prévisible si ses paramètres sont entièrement connus. Mais si les conditions initiales contiennent une part d'incertitude (par exemple une imprécision, même minime), alors un tel processus chaotique ne permet plus de prévision à long terme.

2.4.3 Sensibilité aux conditions initiales

On voit (cf. Table 1) que les deux suites, malgré un point de départ presque identique, divergent complètement à partir de n = 20. Cela montre le caractère chaotique de la structure : sans la connaissance de la huitième décimale de α_0, impossible de prévoir le comportement de cette suite au-delà du 20e terme.

On sait dès lors que l'observation des 20 premiers termes ne permet pas d'en déduire la suite.

1. https ://en.wikipedia.org/wiki/Xorshift

	suite irrationnelle				suite rationnelle	
n	α_n	valeur	ϵ_n	$\overline{\epsilon}_n$	valeur	$\overline{\alpha}_n$
0	$\pi/4$	**0.78539816**	1	1	**0.78539823**	$\frac{355}{452}$
1	$\pi/2-1$	0.57079632	1	1	0.57079646	$\frac{258}{452}$
2	$\pi-3$	0.14159265	0	0	0.14159292	$\frac{64}{452}$
3	$2\pi-6$	0.28318530	0	0	0.28318584	$\frac{128}{452}$
4	4π–12	0.56637061	1	1	0.56637168	$\frac{256}{452}$
5	8π–25	0.13274122	0	0	0.13274336	$\frac{60}{452}$
6	16π–50	0.26548245	0	0	0.26548672	$\frac{120}{452}$
7	32π–100	0.53096491	1	1	0.53097345	$\frac{240}{452}$
8	64π–201	0.06192982	0	0	0.06194690	$\frac{28}{452}$
9	128π–402	0.12385965	0	0	0.12389380	$\frac{56}{452}$
10	256π–804	0.24771931	0	0	0.24778761	$\frac{112}{452}$
11	512π–1608	0.49543863	0	0	0.49557522	$\frac{224}{452}$
12	1024π–3216	0.99087727	1	1	0.99115044	$\frac{448}{452}$
13	2048π–6433	0.98175455	1	1	0.98230088	$\frac{444}{452}$
14	4096π–12867	0.96350910	1	1	0.96460176	$\frac{436}{452}$
15	8192π–25735	0.92701820	1	1	0.92920353	$\frac{420}{452}$
16	16384π–51471	0.85403641	1	1	0.85840707	$\frac{388}{452}$
17	32768π–102943	0.70807283	1	1	0.71681415	$\frac{324}{452}$
18	65536π–205887	0.41614566	0	0	0.43362831	$\frac{196}{452}$
19	131072π–411774	0.83229132	1	1	0.86725663	$\frac{392}{452}$
20	262144π–823549	0.66458264	1	1	0.73451327	$\frac{332}{452}$
21	524288π–1647099	0.32916528	0	0	0.46902654	$\frac{212}{452}$
22	1048576π–3294198	0.65833057	1	1	0.93805309	$\frac{424}{452}$
23	2097152π–6588397	0.31666114	0	1	0.87610619	$\frac{396}{452}$
23	4194304π–13176794	0.63332228	1	1	0.75221238	$\frac{340}{452}$
24	8388608π–26353589	0.26664456	0	1	0.50442477	$\frac{228}{452}$
25	16777216π–52707178	0.53328913	1	0	0.00884955	$\frac{4}{452}$
26	33554432π–105414357	0.06657826	0	0	0.01769911	$\frac{8}{452}$
27	67108864π–210828714	0.13315653	0	0	0.03539823	$\frac{16}{452}$
28	134217728π–421657428	0.26631307	0	0	0.07079646	$\frac{32}{452}$
29	268435456π–843314856	0.53262615	1	0	0.14159292	$\frac{64}{452}$

TABLE 1 – Exemple comparé, avec $\alpha_0 = \frac{\pi}{4}$ et $\overline{\alpha}_0 = \frac{355}{452}$, deux valeurs très proches

Problème : si on s'intéresse à la suite irrationnelle, parfaitement aléatoire en apparence, le problème est qu'il faut connaître le nombre irrationnel de départ (ici $\alpha_0 = \frac{\pi}{4}$) avec une précision croissante. Cela demandera un temps de calcul de plus en plus important, et ne ne sera plus possible pour de très grandes valeurs de n.

Il faut pouvoir gérer des valeurs exactes pour α_n !

2.4.4 Racines irrationnelles du 3e degré

Saito et Yamaguchi [**SY**] proposent une solution à ce problème : en choisissant α_0 comme racine d'un polynôme du 3e degré f_0 dont les coefficients sont entiers, et qui a une unique racine réelle.

Ainsi, on peut facilement déterminer le polynôme f_1 dont α_1 est la racine, et ainsi de suite.

Exemple : on pose $f_0(x) = x^3 + x–1$ et $f_n(x) = x^3 + b_n x^2 + c_n x–d_n$, avec $f_n(\alpha_n) = 0$, et $\epsilon_n = (\alpha_n)$.

n	(b_n, c_n, d_n)	$f_n(x)$	α_n	ϵ_n
0	$(0, 1, -1)$	$x^3 + 0x^2 + 1x - 1$	0.682328	1
1	$(3, 7, -3)$	$x^3 + 3x^2 + 7x - 3$	0.364656	0
2	$(6, 28, -24)$	$x^3 + 6x^2 + 28x - 24$	0.729311	1
3	$(15, 139, -67)$	$x^3 + 15x^2 + 139x - 67$	0.458622	0
4	$(30, 556, -536)$	$x^3 + 30x^2 + 556x - 536$	0.917245	1
5	$(63, 2347, -2003)$	$x^3 + 63x^2 + 2347x - 2003$	0.834490	1
6	$(129, 9643, -6509)$	$x^3 + 129x^2 + 9643x - 6509$	0.668979	1
⋮	⋮	⋮	⋮	⋮

2.4.5 Calcul de la séquence

Alors les relations ci-dessous permettent d'éviter le calcul des α_n :

$a_n = 1$ $\qquad b_{n+1} = 2b_n + 3\epsilon_n$

$c_{n+1} = 4c_n + (4b_n + 3)\epsilon_n$ $\qquad d_{n+1} = 8d_n + (4c_n + 2b_n + 1)\epsilon_n$

$1 + 2b_n + 4c_n + 8d_n < 0 \;\Leftrightarrow\; \alpha_n > \frac{1}{2} \;\Leftrightarrow\; \epsilon_n = 1$

La production de la séquence pseudo-aléatoire $\{\epsilon_n\}_{n\in\mathbb{N}}$ consiste donc à déterminer les termes de la suite $\{(b_n, c_n, d_n, \epsilon_n)\}_{n\in\mathbb{N}}$. Celle-ci se construit entièrement avec des opérations élémentaires, sans avoir à extraire les racines α_n qui restent implicites.

Selon les auteurs, en choisissant bien les coefficients du polynôme initial f_0, on peut vérifier que la séquence binaire produite est presque uniformément distribuée sur l'intervalle [0 ;1].

3 Générateurs de "vraies" suites aléatoires

3.1 Processeur désarmé

Le processeur est incapable de générer de vrais nombres aléatoires puisqu'il est intrinsèquement déterministe. Tout est prévisible dans son fonctionnement. Si on lui donne

plusireus fois une même instruction et des mêmes données, il donnera toujours la même réponse.

Malgré cela, on a vu qu'il était capable de propager l'entropie (source de hasard) grâce à des algorithmes. Il lui font juste une graine (seed) pour s'assurer qu'il ne fournira pas toujours la même suite de nombres.

Si la graine n'est pas un vrai nombre aléatoire, l'ensemble de la suite pourrait être deviné et cela est intolérable, notamment dans les protocoles sécuritaires de chiffrage.

Il nous faudrait donc une graine vraiment imprévisible pour assurer une source de nombres aléatoires fiables mais comment la générer ?

3.2 Source d'entropie interne

Pour pallier ce problème informatique, il existe des algorithmes qui combinent de nombreuses sources d'entropie dans l'état interne de l'ordinateur pour en tirer un résultat pseudo aléatoire.

On peut citer, par exemple, HAVEGE (HArdware Volatile Entropy Gathering and Expansion) qui est implémenté dans le noyau Linux pour créer des listes de graines utilisables par les algorithmes PRNG.

HAVEGE combine des milliers de paramètres de l'état interne (horloge, registres, mémoire cache, branch predictor, etc) puis en tire un nombre qui n'est pas véritablement aléatoire mais impossible à prédire car dépendant de trop nombreux paramètres qu'un hackeur ne pourrait pas geler ou deviner en temps réel. Pour plus de détails : voir ici.

3.3 Source d'entropie externe

On l'a dit, l'ordinateur est incapable de produire du vrai hasard. Paradoxalement, on a la situation inverse dans le monde physique : on est toujours embêté par l'incertitude des mesures et par le fait qu'on ne peut pas contrôler une valeur physique avec une précision arbitraire.

Prenons l'exemple de la température : on ne peut pas mesurer la température d'un four avec une infinité de décimales et on peut encore moins stabiliser la température du four à une valeur donnée.

Il semble donc que le monde physique pourrait nous fournir une vraie source d'entropie (hasard) inaltérable.

On va voir deux catégories de sources physiques :

— phénomènes classiques (statistiques)

— phénomènes quantiques (individuels)

3.3.1 Entropie classique

Parmis les sources d'entropie naturelle, on trouve tous les processus stochastiques où des lois microscopiques (déterministes ou pas) sont implémentées sur des myriades d'éléments ; ceci aboutit à une mesure macroscopique (comme la température) qu'il est impossible de prédire avec une précision infinie car elle dépend de trop de paramètres (état interne).

Un exemple typique est le bruit thermique correspondant aux fluctuations de la tension électrique aux bornes d'une résistance à une certaine température. En effet, le mouvement imprévisible des électrons se traduit en courant électrique et ceci affecte la tension.

Ceci a été découvert en 1927 par Johnson puis expliqué théoriquement par Nyquist (Johnson & Nyquist).

Imaginons donc qu'on mesure la variation de la tension aux bornes de la résistance, on aurait une bonne source de hasard.

Un autre exemple serait la mesure de la hauteur de l'eau dans un océan. La houle ou les vagues provoquées par le vent entraîne là aussi une impossibilité de prévoir à long terme la hauteur de l'eau. Il convient de remarquer que l'échantillonage des valeurs aléatoires ne devra pas dépasser une fréquence qui dépend de la source employée.

Mais même si la suite temporelle de valeurs de la tension ou de la hauteur de l'eau est chaotique, elle n'est pas forcément entièrement imprévisible, dumoins en théorie. C'est ce que proclament les défendeurs de la prochaine source d'entropie...

3.3.2 Entropie quantique

Les phénomènes quantiques ont l'avantage d'être intrinsèquement indéterministes. Exemples :

— source de radioactivité détectée par un compteur Geiger

— photons traversant un miroir semi-réfléchissant

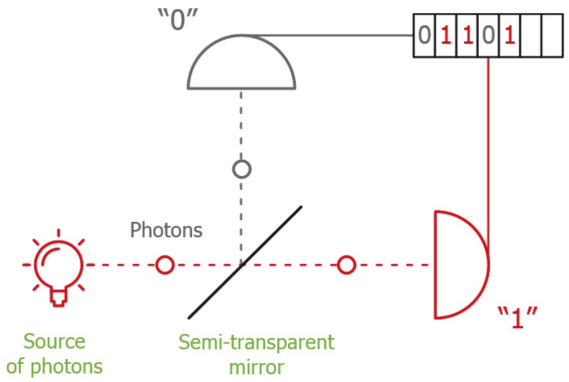
Dans ces deux cas, on ne peut pas deviner le résultat d'une mesure, même si l'on connaît parfaitement l'état interne du système car il est dans une superposition quantique d'états que seule la mesure va briser en réduisant le paquet d'onde.

3.3.3 ID Quantique

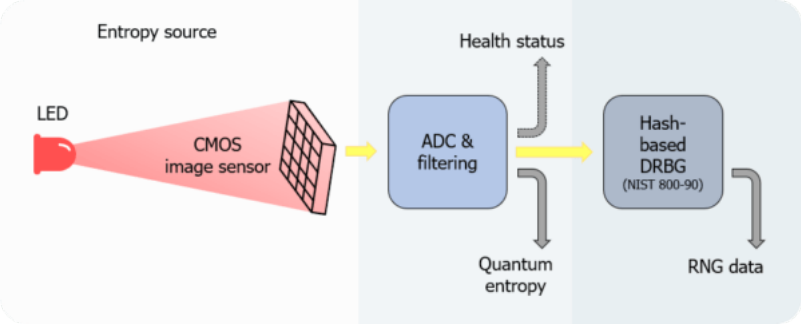
ID Quantique est une entreprise genevoise, spin off de l'université de Genève, fondée en 2001 par 3 scientifiques (Nicolas Gisin, Hugo Zbinden et l'actuel CEO Grégoire Ribordy qui nous a accordé une interview)

Elle a trois principales branches d'activité (détecteurs quantiques, partage de clefs quantique QKD et Génération quantique de nombres aléatoires QRNG)

Le principe du QRNG chez ID Quantique est le suivant :



1. Une diode LASER émet régulièrement un photon.
2. Le photon émis arrive sur un miroir semi-réfléchissant
3. Deux détecteurs sont à l'affût pour signaler si le photon a traversé ou a été réfléchi par le miroir.
4. Le résultat est transformé en "1" ou en "0".
5. Un traitement logiciel (NIST 800-90, anti-aliasing, etc.) est appliqué aux bits pour assurer l'équiprobabilité.
6. Si le débit de nombres aléatoires n'est pas suffisant, on peut utiliser des PRNG se basant sur les graines produites par le QRNG pour l'augmenter.



Ce processus est particulièrement imprédictible car au moment où le photon sort de la diode LASER, il n'a aucune caractéristique qui le prédétermine à être plutôt transmis que réfléchi ou l'inverse. En effet, tous les photons sont indistinguables à la sortie d'un LASER. Même au moment où il atteint le miroir, on ne sait toujours pas s'il a traversé ou s'il a été réfléchi car les deux phénomènes se produisent en réalité en parallèle. Ce n'est qu'au moment d'être effectivement détecté dans un des deux chemins que le paquet d'onde est réduit et que le photon devient une "particule réfléchie" ou une "particule transmise". Impossible donc de recueillir des données innombrables sur l'état interne pour déduire de possibles résultats (comme ce pourrait être envisagé dans un TRNG classique) !

4 Que fait le module "random" de Python ?

Python fait appel à l'OS pour générer des "vrais" nombres aléatoires. Celui-ci implémente typiquement un algorithme HAVEGE. On peut y accéder directement à travers la classe `random.SystemRandom` ou par le module dédié `secrets`, qui, comme son nom l'indique, a pour but de fournir des nombres suffisamment aléatoires pour la cryptographie.

En guise de PRNG, le module `random` fait appel au Mersenne Twister (voir section 2.3), implémenté de manière sous-jacente en C afin d'être rapide, efficace et isolé (vu comme une "opération atomique" par l'interprétateur Python, ce qui évite des soucis dans le cadre de la programmation concurrente). La *seed* par défaut est obtenue par le générateur aléatoire de l'OS, mais elle peut aussi être initialisée "à la main" si nécessaire.

Le module `random` fournit des fonctions transformant directement les nombres générés par le Mersenne Twister de manière à émuler plusieurs distributions aléatoires usuelles (uniforme, normale, etc.) ainsi que le tirage d'une sous-collection aléatoire d'une collection donnée.

```
import random

# entier aléatoire entre 0 et 9
n = random.randrange(10)

# flottant aléatoire entre 0 et 10
x = random.uniform(0, 10)

# tirage de 2 éléments au hasard d'une liste
l = random.sample(["a", "b", "c", "d", "e"], 2)
```

5 Conclusion

Le sujet de la génération de nombres aléatoires par un ordinateur s'est avéré beaucoup plus vaste qu'on aurait pu l'imaginer. D'un côté, les méthodes arithmétiques employant des suites de nombres déterministes de grande période présentant une distribution quasi-uniforme sont en évolution permanente. De l'autre, la cryptographie nécessite sans cesse des progrès dans les moyens de construire des "vrais" nombres aléatoires à partir de sources physiques. Vraisemblablement, il y aura de quoi donner du fil à retordre à beaucoup de chercheurs pendant les prochaines décennies.