Générateur de nombres pseudo-aléatoires ou comment produire de l'aléatoire avec un système déterministe ?

Duzés Florian[1]

[1] *Étudiant de master Cryptologie et Sécurité Informatique*

Document rédigé le January 13, 2025.

Résumé

Étude sur les générateurs de nombres pseudo-aléatoires dans le cadre de l'Unité d'enseignement optionnelle Numerics. Ce document retrace brièvement l'histoire de la création de ces outils, leur multiplicité et leur usage.

1. Introduction

J'aime jouer aux jeux de société. Et je ne suis pas le seul, car il semble que neuf Français sur dix [1] s'adonnent à cette activité régulièrement. C'est une activité stimulante, qui permet de développer nos liens sociaux et nous offre des occasions de nous challenger intellectuellement. Ainsi, la défaite permet d'établir un bref classement des capacités de chacun. Pourtant, au regard de celle-ci, a-t-on vraiment perdu face à un adversaire plus fort ou face à un hasard capricieux ? Cette remarque prend une ampleur différente si des sommes d'argent sont engagées dans la partie...

Aujourd'hui, il est possible de jouer aux cartes sur un ordinateur. Or, un ordinateur est un système déterministe, obéissant à une suite finie d'instructions. Comment, dans ce contexte, peut-on garantir que la distribution des cartes ou d'autres éléments aléatoires reste équitable et imprévisible ?

Cette question nous amène à explorer le concept des générateurs de nombres pseudo-aléatoires. Ces outils sont essentiels pour simuler ou générer de l'aléa. Ils sont utilisés dans divers domaines, allant de la cryptographie aux simulations scientifiques, en passant par les jeux vidéo et les applications statistiques.

2. Nature d'un générateur de nombre pseudo-aléatoire

2.1. Définition

> **Donald E.Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms***
>
> Les séquences générées de manière déterministe sont généralement appelées séquences pseudo-aléatoires ou quasi-aléatoires [...], étant entendu qu'elles ne sont aléatoires qu'en apparence. De toute façon, le fait d'être « apparemment aléatoire » est peut-être tout ce que l'on peut dire d'une séquence aléatoire.

Un générateur de nombres pseudo-aléatoires (*PRNG*) est un algorithme qui génère une séquence de nombres présentant certaines propriétés du hasard. [3]

Un PRNG génère des suites de nombres qui ne peuvent pas satisfaire complètement les critères mathématiques qualifiant les suites aléatoires. On les appelle suites pseudo-aléatoires, car leurs propriétés s'approchent seulement des propriétés idéales des suites aléatoires parfaites.

Si l'on souhaite obtenir des nombres aléatoires, il faut qu'ils soient générés d'une source chaotique. La qualité de la source dépend de la qualité du chaos produit par la source. Écouter le bruit électronique d'une résistance ou observer un mur de lampes à lave [4] sont des solutions actuellement utilisées.

2.2. Caractéristiques

> **Clifford A. Pickover, *Computers, Pattern, Chaos and Beauty***
>
> La génération de nombres aléatoires est trop importante pour être laissée au hasard.

Voici les caractéristiques d'un PRNG :

1. Déterminisme
2. Périodicité
3. Longueur de la période
4. Uniformité
5. Indépendance
6. Qualité statistique
7. Efficacité calculatoire
8. Sécurité
9. Portabilité

Ces caractéristiques découlent du caractère déterministe de nos algorithmes. Les autres points découlent du domaine de l'analyse de suites numériques. Du domaine des suites numériques car, avec un PRNG et une entrée (graine), si l'on utilise la sortie comme nouvelle entrée, on obtiendra une suite numérique que l'on pourra analyser.

Et comme c'est déterministe, pour une même graine, la séquence générée sera toujours identique.

La périodicité et la longueur de cette période permettent de qualifier la qualité d'un PRNG. Dans le cadre de la cryptologie, on souhaite utiliser de longues périodes pour réduire la prévisibilité du terme suivant.

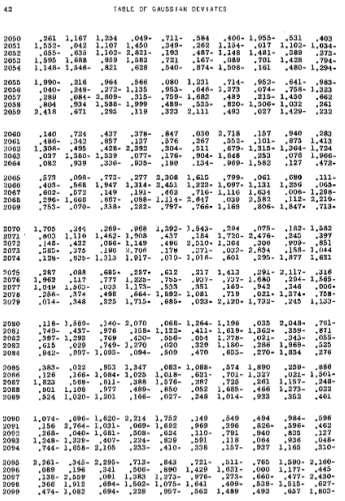
L'uniformité et l'indépendance des nombres générés permettent de distinguer un bon d'un mauvais PRNG. Les nombres doivent être uniformément distribués sur l'intervalle de sortie et statistiquement indépendants les uns des autres. Ces caractéristiques garantissent que, lors d'une application dans le cadre de simulations ou de tests statistiques, les nombres obtenus sont représentatifs et non biaisés.

Enfin, l'efficacité calculatoire d'un PRNG, sa sécurité et sa portabilité servent à déterminer son usage. Dans le cas des GPS, on souhaite un PRNG qui produise une suite de nombres très rapidement pour permettre de synchroniser deux appareils et calculer leur distance dans l'espace [3].

3. Un bref historique

3.1. Origines et premières itérations

À l'origine, lorsque des scientifiques avaient besoin de nombres aléatoires, ils tiraient des nombres d'une urne équilibrée, tiraient des cartes ou lançaient manuellement des dés [2]. En 1927, une table de (41 600) nombres aléatoires est publiée pour la toute première fois [6]. C'est le premier travail de simplification et d'accessibilité qui est offert au monde scientifique. Par la suite, une multitude de tables vont fleurir pour améliorer les travaux et les études utilisant



42 TABLE OF GAUSSIAN DEVIATES

2050	.261	1.167	1.234	.049-	.711-	.584	.400-	1.955-	.531	.403
2051	1.553-	.041	1.197	1.450	.249-	.262	1.134-	.017	1.102-	1.034-
2052	.655-	.634	1.102-	2.833-	.193	.487-	1.148	1.481-	.589	.372-
2053	1.595	1.698	.919	1.582	.731	1.467-	.080	.701	1.428	.794-
2054	1.148-	1.346-	.821	.628	.540-	.874-	1.508-	.191	.480-	1.294-
2055	1.990-	.316	.964	.586	.080	1.231	.714-	.952-	.641-	.983-
2056	.040-	.248-	.272-	1.135	.953-	.646-	1.275	.074-	.758-	1.323
2057	.289	.084-	3.809-	.315-	.759-	1.682	.485	.219-	1.450	.062
2058	.904	.934	1.588-	1.999	.489-	.535-	.826-	1.506-	1.032	.261
2059	2.418	.971	.295	.319	.973	2.111	.491	.037	1.439-	.032
2060	.140	.724	.437	.378-	.847	.030	2.718	.157	.940	.283
2061	.486-	.362	.857	.127	.876	.267	.353-	.101-	.873	1.413
2062	1.306-	.495	.428-	2.292	.204-	.511	.679-	1.318-	1.304-	1.724
2063	.037	2.580-	1.539	.077-	.179-	.904-	1.548	.293	.076	1.966-
2064	.082	.939	.336-	.935-	.180	.134-	.969-	1.582	.127	.472-
2065	.573	.098-	.772-	.277	2.308	1.615	.799-	.061	.089	.111-
2066	.405-	.568	1.947	1.314-	2.451	1.223-	1.097-	1.131	1.299	.063-
2067	.802-	.572	.149	.191-	.462	.716-	1.116	1.634	.006-	1.298-
2068	.296-	1.063	.667-	.098-	1.114-	2.847	.032	2.582	.112-	2.219-
2069	.753-	1.070-	.338-	.282-	.797-	.766-	1.168	.806-	1.847-	.713-
2070	1.705	.264	.269-	.968	1.392-	1.843-	.394	.075-	.182-	1.982
2071	.803	1.134	1.462-	1.908	.437	.184	1.720-	2.476-	.245	.397
2072	.148	.422	.086-	1.149	.496	2.510-	1.262	.208	.959-	.851
2073	.585-	.272	.196	1.706	.178	.272-	.032-	2.854	.158-	1.044
2074	.139-	1.020-	1.313	1.917-	.019-	1.018-	.601	.295-	1.877	1.831
2075	.287	.088	.685-	.287-	.612	.217	1.412	.091-	2.117-	.316
2076	1.962	.117	.777	1.225-	.755-	.937-	.727-	1.240	.294-	1.585-
2077	1.019	1.360-	.003	1.173-	.533	.351	.169-	.942	.346	.006-
2078	.284-	.574	.498	.464-	1.882-	1.081	.719	.021-	1.174-	.758-
2079	.014-	.348	.325	1.715-	.689-	.023-	2.120-	1.732-	.245	1.139-
2080	.118-	1.589-	.140-	2.070	.068-	1.264-	1.196	.035	2.049-	.761-
2081	.749-	.457-	.976	.158-	1.122-	.411-	1.619-	1.362-	.359-	.871
2082	1.587-	1.292	.769	.620-	.926-	.054	1.778-	.021-	.243-	.053-
2083	.615	.029	1.749-	1.270	.020	.329	1.180-	.288	1.868-	.525
2084	2.842-	1.997-	1.035-	.084-	.809	.470	.853-	2.270-	1.894	.276
2085	.383-	.022	.853	1.347	.083-	1.088-	.574	1.890	.259-	.888
2086	.126	1.166-	1.084-	1.025	1.018-	.621-	.701-	1.327	.021-	1.301-
2087	1.822	.598-	.011-	.388	1.576-	.997	.728	.261	1.157-	.248-
2088	.801	1.106	.977	.489-	.850	.052	1.685-	.466	1.273-	.022
2089	.524	1.020-	1.303	.166-	.027-	.248	1.014-	.933	.353	.461
2090	1.074-	.696-	1.620-	2.214	1.752	.149	.549	.494	.984-	.698
2091	.156	2.764-	1.031-	.669-	1.492	.949	.596	.826-	.596-	.162
2092	.268-	.046-	1.681-	.508-	.624	.110-	.791	.940	.835	.127
2093	1.248-	1.328-	.407-	.234-	.826	.597	.118	.064	.936	.048-
2094	.744-	1.658-	2.165	.253-	.410-	.036	.157-	.937	1.168	.310-
2095	2.261-	.345-	2.298-	.713-	.843	.721	.511-	.765	1.590-	2.166-
2096	.089	.196	.341	.506-	.890	1.428	1.621-	.000	1.177-	.445
2097	.138-	2.529	.091	1.383	1.272-	.976-	.273-	.660-	1.477-	2.490-
2098	.366	1.912	.894-	1.502-	1.075-	1.641	.409-	.539-	1.515-	.627-
2099	.474-	1.082	.894-	.238	.957-	.862	1.489	.492	.827	1.803-

Figure 1. Page prise au hasard du *A Million Random Digits with 100,000 Normal Deviates*,RAND

l'aléatoire. En 1955, une association américaine, RAND, publie la première table d'un million de nombres [7].

En parallèle, la technique pour la génération de nombres aléatoires progresse aussi et ERNIE (*Electronic Random Number Indicator Equipment*) fut achevé en 1957. Cette machine servait à tirer les numéros du loto anglais. La génération de nombres est réalisée avec l'observation du bruit produit par le passage d'un courant électrique dans des tubes à néons. ERNIE 1 était capable de générer environ 2 000 nombres par heure. En 2019, ERNIE 5 est entré en activité et observe le bruit quantique, actuellement une des méthodes les plus sûres et les plus imprévisibles pour générer des nombres aléatoires. ERNIE 5 peut générer environ 1,2 million de nombres par heure. [8]

3.2. Évolutions

L'utilisation de tables de chiffres était compliquée due au support de l'information, alors sur bande magnétique. Entrer les données est fastidieux et obtenir une table de nombres aléatoires assez grande pour réaliser plusieurs expériences différentes et indépendantes est difficile. L'utilisation d'une machine comme ERNIE n'est pas une solution valide, comment reproduire les résultats obtenus lorsqu'une machine génère des nombres aléatoirement ?

Ce sont ces problématiques, en plus de la démocratisation des ordinateurs dans les milieux scientifiques, qui ont fait avancer les travaux sur les PRNG. Le premier PRNG fut élaboré par Von Neumann en 1946 et se nomme le **carré médian** [9].

Méthode du Carré médian

Pour la graine "1111":

1. $1111^2 = 1234321 \implies 12\,'3432'\,1$
2. $3432^2 = 11778624 \implies 11\,'7786'\,24$

0n obtient "7786" en deux tours.

Ce premier PRNG s'est démocratisé assez rapidement et fut largement étudié durant les années 50. Aujourd'hui, il n'est plus utilisé, car il s'avère que c'est une méthode qui dégénère en séquence cyclique, souvent avec une période très courte [10].

4. Éventails de générateurs de nombres pseudo-aléatoires

Tous les types différents de générateurs pseudo-aléatoires mentionnés par la suite sont issus de cet ouvrage : *Monte Carlo Methods, Second Revised and Enlarged Edition* [11].

4.1. Générateurs Congruentiels

Après la méthode du carré médian, les recherches se sont portées vers l'usage des récurrences linéaires, mais la qualité du PRNG dépend excessivement de la caractérisation de ses paramètres a_j, b, P.

$$\text{Récurrence linéaire : } x_{i+1} \equiv a_0 x_i + \cdots + a_j x_{i-j} + b \pmod{P} \quad (1)$$

Lorsqu'on analyse un PRNG issu de cette méthode, on constate que les périodes sont trop courtes et présentent des propriétés statistiques insuffisantes. Pourtant, en reprenant cette idée et en la plaçant dans l'espace des multiplications; selon M.Lehmer [12], alors on obtient un **générateur congruentiel multiplicatif (MCG)** :

$$x_{i+1} \equiv \lambda \cdot x_i \pmod{P} \quad (2)$$

La forme (1) récurrence linéaire est trop générale et n'est pas satisfaisante pour des travaux statistiques. En revanche, en la déclinant en (2) on obtient plus facilement de meilleures propriétés.

En 1988, Park et Miller [13] ont proposé un PRNG en définissant P comme un premier de Mersenne et λ une racine modulo 31 :

$$x_{i+1} \equiv 7^5 \cdot x_i \pmod{2^{31} - 1} \quad (3)$$

Ce PRNG se nomme **MINSTD** et a été utilisé dans Carbon (vieille API d'Apple) et C++11 (C++ version 2011).

En poursuivant cette voie, les **Générateurs récursifs multiples (MRG)** ont été développés pour obtenir une période plus importante que les MCG. Messieurs Marsaglia et Zaman [14] ont conçu une variation des MRG nommée *Multiply with carry*. Malheureusement, il a été démontré quelques années plus tard que cette variante produit des séquences identiques à des MCG.

4.2. Générateurs à rétroaction

Les équations linéaires, c'est chouette, mais dans un contexte informatique, où tout se réduit à des bits, où les nombres sont transformés et évalués en base 2, n'y a-t-il moyen d'exploiter cette idée pour fabriquer des nombres ? Ainsi, Tausworthe a présenté un PRNG [15] qui se base sur la manipulation des registres où sont stockés les bits. Cette méthode est en réalité identique à l'équation (1) avec $P = 2$:

$$b_i = a_1 b_{i-1} + \cdots + a_j b_{i-j} \pmod{2} \quad (4)$$

Cette méthode est très rapide mais présente des soucis de sécurité. Par exemple, pour l'équation (4), on sait que sa période maximale est $2^j - 1$ (on a j bits et deux choix à chaque fois). En revanche, si l'implémentation est correctement réalisée, alors ces PRNG sont totalement utilisables.

Les itérations suivantes ont ajouté l'idée de décalage entre les bits impactés, puis enfin l'idée de "tordre" les suites de bits. En ajoutant une matrice binaire (A) dans les opérations, on peut modifier les résultats et ainsi obtenir une période maximale plus élevée :

$$y_{i+n} = y_{i+m} \oplus A y_i \; ; n > m \quad (5)$$

On peut retrouver une implémentation exploitant cette idée dans la bibliothèque *random* de Python. Elle a été conçue en 1998 par Matsumoto et Nishimura [16], utilise un premier de Mersenne et a une période de $2^{19937} - 1$: le "**Mersenne Twister**".

4.3. Générateurs non linéaires

Le point négatif des PRNG basés sur la récursion linéaire est qu'ils présentent des structures de réseaux. Cette structure peut être problématique si l'on souhaite générer des clés de chiffrement pour une communication, par exemple.

Pour résoudre cette problématique, Lenore Blum, Manuel Blum et Michael Shub ont proposé leur solution [17] :

$$\textbf{Blum Blum Shub} \ : x_{i+1} = {x_i}^2 \pmod M \tag{6}$$

Avec M est le produit de deux grands nombres premiers. Le point négatif de ce PRNG est sa lenteur. En revanche, son absence de structure lui a permis d'être utilisée dans des applications cryptographiques.

4.4. Générateurs combinaison

Enfin, la dernière méthode pour concevoir un PRNG et qui permet d'améliorer les propriétés et la période d'un PRNG est de combiner deux ou plusieurs PRNG indépendants.

$$\theta_i = \frac{x_i}{30269} + \frac{y_i}{30307} + \frac{z_i}{30323} \pmod 1 \tag{7}$$

Dans cet exemple, trois générateurs congruentiels multiplicatifs sont combinés, il s'agit du PRNG de Wichmann et Hill [18].

5. Tests et étude de qualité

J'ai mentionné précédemment qu'il était possible d'évaluer les qualités statistiques d'un PRNG. Or ce que l'on a à observer ce sont des suites de nombres. Et rien d'autre qu'une suite de nombres apparemment aléatoire ne ressemble plus à une suite aléatoire, regardez π.

> ***Scientific American*, "Mathematical Games", Martin Gardner, 1965**
>
> Les mathématiciens considèrent le développement décimal de π comme une série aléatoire, mais pour un numérologue moderne, il est riche en motifs remarquables.

Pour résoudre ce problème, des algorithmes ont été mis au point pour évaluer la distribution, la présence de corrélation et d'autres éléments à évaluer.

5.1. Test Statistiques

Ces tests permettent de vérifier si les nombres générés sont uniformément distribués et s'ils présentent des corrélations indésirables. La réponse de ces tests n'est pas si une suite est ou n'est pas aléatoire. La réponse est sa probabilité d'être une distribution uniforme.

5.1.1. Tests Théorique

Le **test spectral** conçus par Coveyou et MacPherson [20]. Ce test détermine si une suite de s éléments issus d'un PRNG de classe MCG est issue de variables aléatoires indépendantes et uniformément distribuées. Le test utilise une transformée de Fourier sur chaque suite pour l'évaluer. Si la période est m alors chaque s-uples doit apparaître $\frac{1}{m^s}$.

Ce test a été amélioré en 2002 par travailler empiriquement [21]. Cette version examine la structure de réseau de la génération du PRNG, il observe la discrépance[1] maximale par rapport au nombre attendu d'occurrences d'un s-uples.

5.1.2. Tests Empiriques

Les tests empiriques consistent à observer le comportement d'un PRNG sur son comportement à grande échelle, avec une mise en perspective de toutes ces productions. Trois tests sont mentionnés ici [11] : *Test d'équidistribution*, *Test sériel* et *Test des séries ascendantes et descendantes*.

Le test d'équidistribution, où l'intervalle (0, 1) est divisé en k sous-intervalles. Le nombre de valeurs tombant dans chaque sous-intervalle est déterminé pour une séquence de nombres pseudo-aléatoires, et un test khi carré est effectué pour vérifier si les nombres sont uniformément distribués dans leurs intervalles.

[1] Discrépance : (Mathématiques) Décrit la déviation, parfois asymptotique, d'une situation considérée comme régulière.

> **Test du khi carré ou χ^2 [22]**
>
> $$\chi^2 = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$
>
> - χ^2 : valeur du khi carré,
> - O_{ij} : valeur observée,
> - E_{ij} : valeur attendue.
>
> Le test du khi-carré d'adéquation vérifie si nos échantillons correspondent à une distribution définie a priori.

Le test sériel est un autre test empirique important. Il vérifie l'interdépendance entre les nombres pseudo-aléatoires successifs dans une séquence. L'espace est divisé en r^s partitions, et la fréquence avec laquelle les s-tuples tombent dans chaque partition est mesurée. Les nombres vraiment aléatoires sont uniformément distribués dans l'espace de dimension s, donc une valeur de χ^2 peut être calculée pour évaluer la qualité du générateur.

Le test des séries ascendantes et descendantes compare la magnitude d'un nombre pseudo-aléatoire avec le précédent. Il compte le nombre de séries ascendantes (ou descendantes) de différentes longueurs dans une séquence de nombres pseudo-aléatoires. Ce test permet de détecter des corrélations indésirables dans les séquences générées.

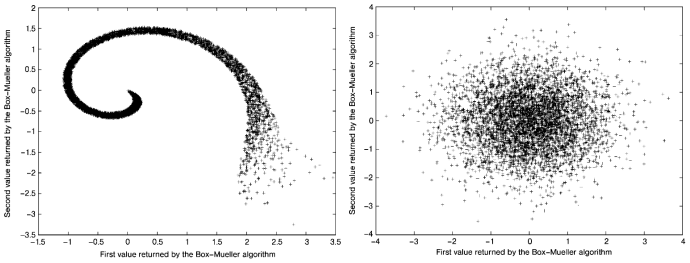


Figure 2. Extrait de *Monte Carlo Methods, Second Revised and Enlarged Edition* présentant deux distributions issus de PRNG dans un objectif de comparaisons de leur qualité

5.2. Test de sécurité

Les tests de sécurité des PRNG interviennent pour garantir que les nombres générés soient peu prédictibles et donc, ne présentent pas de vulnérabilités exploitables. En utilisant une combinaison de tests théoriques et empiriques, l'objectif est de vérifier si notre PRNG présente une distribution pseudo-uniforme avec aucune période courte, ni de corrélations entre les nombres générés. L'objectif est de se prémunir de la cryptanalyse que peuvent subir nos PRNG et ainsi offrir une faille lors de la génération de clés secrètes utilisées dans les applications sécurisées.

6. Applications des PRNG

Les PRNG sont associés au développement des ordinateurs. Leurs usages s'inscrivent dans le besoin de reproduction de l'aléatoire dans un contexte déterministe. Les trois domaines principaux qui me viennent sont :

- *La cryptographie* - besoin de l'aléatoire pour y dissimuler de l'information.
- *Le divertissement* - pour concevoir des jeux avec des comportements imprédictibles.
- *La simulation, modélisation* - dans différents domaines scientifiques comme la physique, la finance ou les statistiques.

7. Un regard sur l'horizon

Voici maintenant un siècle que ce domaine s'affine, se perfectionne et progresse. On a un cadre qui nous permet de définir les usages pour lesquels un nouveau PRNG serait applicable. Si je suis imprédictible, je serai utilisé en cryptographie, en revanche si je suis calculable rapidement, j'irai vers des usages statistiques.

SHISHUA [23] conçu en 2020, est capable de produire un gigabit de nombres par seconde. La construction de ce PRNG est optimisée pour des ordinateurs d'architecture *x86-64*. Dernièrement (ce 31 décembre 2024), une équipe internationale a mis au point un PRNG matériel [24] conçu pour générer simultanément plusieurs séquences non corrélées, avec des statistiques programmables adaptées aux besoins spécifiques des applications. Ce PRNG occupe une surface d'environ 0,0013 mm^2 et consomme une énergie de 0,57 pJ/bit.

Actuellement, l'objectif principal est d'améliorer surtout les structures de PRNG imprédictibles. Les propriétés d'imprédictibilité sont encore mal définies. Ces travaux sont importants pour pouvoir améliorer les performances d'applications cryptographiques.

8. Conclusion

À travers ce document, nous avons exploré l'évolution historique et technique des PRNG, leurs nombreuses variantes, ainsi que les tests qui permettent d'évaluer leur performance. La génération de nombres pseudo-aléatoires est un domaine qui allie science, technique et inspiration. Si les PRNG ont largement évolué depuis leurs premières itérations, leur utilisation dépend toujours de compromis entre efficacité, sécurité et qualité statistique. Comprendre leurs caractéristiques et leurs limitations est essentiel pour garantir leur bon usage, que ce soit dans les jeux, la simulation ou la cryptographie. À travers cette exploration, nous avons vu que bien que la perfection de l'aléatoire reste hors de portée, les progrès réalisés nous permettent de répondre à des besoins toujours plus complexes et exigeants.

References

1. Businesscoot. *Marché du jeux de société* https : / / www . businesscoot.com/fr/etude/le-marche-des-jeux-de-societe-france.
2. E.Knuth, D. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms* (Addison Wesley, 1997).
3. Anonymes. *Générateur de nombres pseudo-aléatoires* https://fr.wikipedia.org/wiki/G%C3%A9n%C3%A9rateur_de_nombres_pseudo-al%C3%A9atoires.
4. Cloudflare. https://fr.wikipedia.org/wiki/Lavarand.
5. Pickover, C. A. *Computers, Pattern, Chaos and Beauty* ISBN: 9780486151618. https : / / books . google . fr / books ? id = xnzCAgAAQBAJ & printsec = frontcover & redir _ esc = y # v = onepage&q&f=false (Courier Corporation, 2012).
6. J.M.Sengupta, N. B. *Table of random normal deviates* (Indian Statistical Institute, Calcutta, 1958).
7. RAND. *A Million Random Digits with 100,000 Normal Deviates* (The Free Pres, 1955).
8. Holmogorov, V. *ERNIE — an incredible computer based on military technologies* https://medium.com/serverspace-cloud/ernie-an - incredible - computer - based - on - military - technologies - ba4c3bcc4fca.
9. Anonyme. *Méthode du Carré médian* https://fr.wikipedia.org/wiki/M%C3%A9thode_du_carr%C3%A9_m%C3%A9dian.
10. Meyer, H. A. *Symposium on Monte Carlo Methods* (Wiley, 1956).
11. Malvin H. Kalos, P. A. W. *Monte Carlo Methods, Second Revised and Enlarged Edition* (Wiley-VCH, 2008).
12. Lehmer, D. H. Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery. *Annals of the Computation Laboratory of Harvard University, Vol. 26* (1951).
13. Park Stephen K.; Miller, K. W. Random Number Generators: Good Ones Are Hard To Find. *Communications of the ACM* (1988).
14. G. Marsaglia, A. Z. A new class of random number generators. *The Annals of Applied Probability* (1991).
15. Tausworthe, R. Random numbers generated by linear recurrence modulo two. *Mathematics of Computation* (1965).
16. Matsumoto, M. & Nishimura, T. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *ACM Trans. on Modeling and Computer Simulation Vol. 8* (1998).
17. Lenore Blum, M. B. e. M. S. A Simple Unpredictable Pseudo-Random Number Generator. *SIAM Journal on Computing, vol. 15* (1986).
18. Brian A.Wichmann, D. I. Algorithm AS 183: An Efficient and Portable Pseudo-Random Number Generator. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* (1982).
19. Gardner, M. Mathematical Games. *Scientific American* (Jan. 1965).
20. Et R. D.MacPherson, R. R. Fourier analysis of uniform random number generators (1967).
21. Zeitler D., M. J. & J., K. Empirical spectral analysis of random number generators. *Computing Science and Statistics* (2002).
22. *Test du khi carré* https://fr.wikipedia.org/wiki/Test_du_%CF%87%C2%B2.
23. Tyl, T. SHISHUA: The Fastest Pseudo-Random Generator In the World. *espadrine blog.* https://espadrine.github.io/blog/posts/shishua-the-fastest-prng-in-the-world.html (2020).
24. Wu, J., Salim, A. Y., Elmitwalli, E., Köse, S. & Ignjatovic, Z. *A Pseudo-random Number Generator for Multi-Sequence Generation with Programmable Statistics* 2024. arXiv: 2501.00193 [cs.CR]. https://arxiv.org/abs/2501.00193.