NOMBRES PSEUDO-ALEATOIRES

Pierre-Louis BAYLE - Benjamin BILLET

SOMMAIRE

I. Introduction

L'aléatoire intervient dans de nombreuses disciplines scientifiques appliquées comme, par exemple, la simulation (méthode d'approximation de Monte Carlo), la modélisation, les jeux, la cryptographie et le traitement du signal.

« *Quiconque considère des méthodes arithmétiques pour produire des nombres aléatoires est, bien sûr, en train de commettre un péché.* »

John Von Neumann.

La production de hasard par des calculs est extrêmement délicate car, par définition, le hasard n'est pas déterministe, au contraire des algorithmes de génération. C'est pour cela que l'on parle de générateurs de nombres pseudo-aléatoires : les suites de nombres qu'ils produisent tendent vers le véritable hasard mais ne peuvent pas l'atteindre totalement.

Les générateurs de nombre pseudo-aléatoires sont donc des techniques permettant de produire mathématiquement, puis informatiquement, des nombres présentant certaines propriétés du hasard.

« *La génération de nombres aléatoires est trop importante pour être confiée au hasard.* »

Robert R. Coveyou.

Le développement des générateurs pseudo-aléatoires est très lié à celui de la cryptographie, l'importance militaire et économique de cette science ayant motivé de nombreuses recherches au cours de l'histoire.
Le premier générateur de nombre pseudo-aléatoire est le générateur middle-square (carré médian). Très vite, il sera suivi des premiers générateurs à congruence linéaire qui ont été améliorés jusqu'à nos jours.

Aujourd'hui, la question est ouverte quant à savoir s'il est possible de distinguer une suite pseudo-aléatoire générée algorithmiquement d'une source parfaite d'aléa, ceci sans connaître les propriétés du générateur. En cryptographie, la

plupart des spécialistes partent du principe que c'est une chose impossible avec la puissance de calcul actuelle à moins que le générateur échoue à un ou plusieurs des tests statistiques pratiqués à l’heure actuelle.

II. Prérequis

1. Sources et entropie

a. Source d'information

Une source d'information est constituée d'un alphabet source, noté S, et d'une loi de probabilité d'occurrence, notée P, de chaque élément de cet alphabet (p_i étant la probabilité d'apparition de l'élément s_i).
Dès lors, la source d'information est notée (S, P).

Source sans mémoire : La probabilité p_i de chaque évènement reste stable au cours de l'émission du message et est indépendante des autres évènements.

Exemple : 3 6 4 8 9 5 1 6 est une suite de valeurs aléatoires. Chaque valeur est indépendante (le 6 n'avait pas plus de chance de sortir après le 3 qu'après le 1).

Source markovienne : Les probabilités de chaque évènement dépendent des évènements émis précédemment.
Si l'on note p_{ij} la probabilité d'occurrence de s_i sachant que s_j vient d'être émis alors la probabilité p_i est égale à la somme des probabilités d'émission de s_i par rapport à tous les éléments de l'alphabet S.

$$p_a = \sum_j p_{aj}$$

Exemple : L'alphabet occidental. Le caractère "n" a plus de chance d'apparaître à la suite d'une voyelle qu'à la suite d'une consonne.

Exemple 2 : Soit une source d'information (S, P) munie de $S = \{a, b\}$ et $P = \{p_{aa}, p_{ab}, p_{bb}\}$ où p_{ab} est la probabilité d'occurrence de a sachant que b vient d'être émis.

$$p_a = \sum_j p_{aj}$$
$$= p_{aa} + p_{ab}$$

Source sans redondance : La distribution des probabilités de chaque évènement est uniforme.

b. Entropie

L'entropie est une mesure du désordre d'une source d'information S, notée H(S), se calcule de la façon suivante.

$$H(S) = H(P) = -\sum_{i=1}^{n} p_i \times log_2(p_i)$$

$$H(S) = H(P) = \sum_{i=1}^{n} p_i \times log_2\left(\frac{1}{p_i}\right)$$

Propriétés :

- $\forall\, S, 0 \leq H(S) \leq log_2\, n$ où n est le nombre d'éléments de S.
- $H(S)$ tend vers $\log_2 n$ lorsque le désordre est maximal (distribution uniforme).

c. Entropie conjointe

Soient $S_1(S_1, P_1)$ et $S_2(S_2, P_2)$ deux sources sans mémoire dont les évènements d'une source ne sont pas forcément indépendants de l'autre source.

On note $S_1 = \{s_{11}, \dots, s_{1n}\}, P_1 = \{p_1, \dots, p_n\}$ et $S_2 = \{s_{21}, \dots, s_{1m}\}, P_2 = \{p_2, \dots, p_m\}$.

On note la probabilité d'occurrence conjointe de s_{1i} et s_{2j} (la probabilité qu'un évènement s_{1i} apparaisse au même moment qu'un évènement s_{2j}) comme $p_{i,j} = P\big(S_1 = s_{1i} \cap S_2 = s_{2j}\big)$

On appelle l'*entropie conjointe* de S_1 et S_2 la quantité suivante :

$$H(S_1, S_2) = -\sum_{i=1}^{n}\sum_{j=1}^{m} p_{i,j} \times \log_2\big(p_{i,j}\big)$$

Si S_1 et S_2 sont rigoureusement indépendantes alors $H(S_1, S_2) = H(S_1) + H(S_2)$.

d. Entropie conditionnelle

Soient $S_1(S_1, P_1)$ et $S_2(S_2, P_2)$ deux sources sans mémoire dont les évènements d'une source ne sont pas forcément indépendants de l'autre source.

On note $S_1 = \{s_{11}, \dots, s_{1n}\}, P_1 = \{p_1, \dots, p_n\}$ et $S_2 = \{s_{21}, \dots, s_{1m}\}, P_2 = \{p_2, \dots, p_m\}$.

On note la probabilité d'occurrence conditionnelle de s_{1i} et s_{2j} (la probabilité qu'un évènement s_{1i} apparaissent au même moment qu'un évènement s_{2j}) comme $p_{i|j} = P(S_1 = s_{1i}|S_2 = s_{2j})$

On appelle *l'entropie conditionnelle* de S_1 relativement à une valeur S_2 la quantité représentant le désordre d'une source par rapport à un événement d'une autre source :

$$H(S_1|S_2 = s_{2j}) = -\sum_{i=1}^{n} p_{i|j} \times \log_2(p_{i|j})$$

Par extension on appelle entropie conditionnelle de S_1 par rapport à S_2 la quantité :

$$H(S_1|S_2) = \sum_{j=1}^{m} p_j \times H(S_1|S_2 = s_{2j}) = \sum_{i,j} p_{i,j} \log_2\left(\frac{p_j}{p_{i,j}}\right)$$

Cette notion est très importante en génération de nombres pseudo aléatoires, il est nécessaire que la suite de valeurs générées ait une entropie forte pour éviter que l'on puisse déduire des éléments de l'algorithme générateur en exploitant les éventuelles formes d'organisations dans la suite.

e. Extension de source

L'entropie seule ne suffit pourtant pas à déterminer la quantité de désordre d'un message.

En effet $H("ABCDEFABCDEF") = H("BAFEDBCACDFE")$ alors que l'on peut constater une forme d'organisation triviale pour le premier message.

On utilise alors les *extensions de source*.

Définition : Soit $S(S, P)$ une source sans mémoire. La k^{eme} extension S^k de S est le doublet (S^k, P^k) où S^k est l'ensemble des chaînes de longueur k sur l'alphabet S et où P^k est la distribution de probabilité définie par :

Pour un mot $m = s_{i1} \dots s_{ik} \in S^k$

Alors $P^k(m) = P(s_{i1} \dots s_{ik}) = p_{ij} \times \dots \times p_{ik}$

Exemple : $S = \{a, b\}$ et $P = \left\{\frac{1}{3}, \frac{1}{4}\right\}$

$S^2 = \{aa, ab, ba, bb\}$ et $P^2 = \left\{\frac{1}{16}, \frac{3}{16}, \frac{3}{16}, \frac{9}{16}\right\}$

Définition : Soit $S(S, P)$ une source markovienne. La k^{eme} extension S^k de S est le doublet (S^k, P^k) où S^k est l'ensemble des chaînes de longueur k sur l'alphabet S et où P^k est la distribution de probabilité définie par :

Pour un mot $m = s_{i1} \dots s_{ik} \in S^k$ on a $P^k(m) = p_{i_1} \times p_{i_2 i_1} \dots \times p_{i_k i_{k-1}}$ où $p_{i_k i_{k-1}}$ est la probabilité d'occurrence de s_{ik} par rapport à son prédécesseur dans la chaîne.

Définition : Soient M un message de taille n et S_{M^k} la source dont les probabilités correspondent aux occurrences des $k - uplets$ consécutifs de M alors :

$$H(S_{M^k}) \leq \log_2 \left\lceil \frac{n}{k} \right\rceil$$

(où $\left\lceil \frac{n}{k} \right\rceil$ représente la partie entière supérieure de $\frac{n}{k}$)

Ainsi, dans notre exemple cité plus haut ("ABCDEFABCDEF"), si l'on regroupe les éléments de l'alphabet en doublets, $S = \{AB, CD, EF\}$ et $P = \left\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$, l'entropie, qui était de $2{,}585$ pour $S = \{A, B, C, D, E, F\}$ chute à $1{,}585$.

2. Nombres premiers

a. Algorithme d'Euclide et d'Euclide étendu

L'algorithme d'Euclide permet de calculer le pgcd de deux entiers positifs. Très simple, il se présente sous la forme récursive suivante :

```
r := a div b (a et b étant les deux nombres dont le pgcd doit être calculé)
Si r = 0
        Retourner b
Sinon
        calculer pgcd (b, r) //Appel récursif
Fin Si
```

L'algorithme d'Euclide étendu introduit, en plus du calcul du pgcd, le calcul des nombres de Bézout et se présente sous la forme récursive suivante :

```
q := a div b
r := a mod b
Si r = 0
        u = 0
        v = 0
        Retourner b, u, v
Fin Si

d, u, v := calculer pgcd(b, r, u, v)
u := v
v := u - q × v

Retourner d, u, v
```

Une démonstration de l'algorithme en fonctionnement est fournie en annexe.

III.Test de primalité Miller-Rabin

Soit n un nombre impair et soient s et t tels que $n-1 = t \times 2^s$ avec t impair.

$\forall\, a < n$, on a :

$a^{n-1} - 1 = a^{t2^s}$

$= (a^t - 1)(a^t + 1)(a^{2t} + 1) \dots \left(a^{(2^{s-1})t} + 1\right)$

$= (a^t - 1) \sum_{i=0}^{s-1} \left(a^{t2^i} + 1\right)$

Si n est premier alors $a^{n-1} - 1 \equiv 0\ [mod\ n]$ (d'après le théorème de Fermat) ce qui sous-entend que :

- soit $a^t - 1 \equiv 0\ [mod\ n]$
- soit $\sum_{i=0}^{s-1} \left(a^{t2^i} + 1\right) \equiv 0\ [mod\ n]$

On dit qu'un nombre n réussit le test de Miller-Rabin si :

$a^t - 1 \neq 0\ [mod\ n]$

$\sum_{i=0}^{s-1} (\, a^{t2^i} + 1) \neq 0\ [mod\ n]$

Si n est impair et non premier, moins de $\frac{n-1}{4}$ nombres a échouent au test de Miller-Rabin. De même, si $a \in [\![1, \dots, n-1]\!]$, la probabilité devient inférieure à $\frac{1}{4}$.

Ainsi, l'algorithme suivant permet de rechercher un nombre probablement premier :

- On tire au hasard un nombre impair n.
- On tire au hasard k nombres a_i distincts tel que $1 < a_i < n$.
- On applique le test de Miller-Rabin pour chaque a_i.
- Si aucun a_i ne réussit le test de composition, on en déduit que n est premier. La probabilité d'erreur est inférieure à 4^{-k}.
- Sinon on recommence avec $n + 2$ jusqu'à trouver un nombre premier.

IV. Généralités

Il existe une très grande quantité de générateurs pseudo-aléatoires que l'on regroupe en familles.

1. Période

La périodicité se traduit par le fait que le générateur retrouvera un même état interne deux fois. Après cela, il entrera forcément dans un cycle et la taille de ce cycle (nombre d'éléments générés avant démarrage d'un nouveau) est appelée *période*.

Bien entendu, un générateur non périodique est théoriquement possible mais nécessite une mémoire croissante pour stocker les états antérieurs et ne pas y retourner.

2. Graine

La *graine* (ou *germe*) est une composante que l'on retrouve chez chaque générateur. Il s'agit de l'état initial du générateur, état que nous retrouverons lorsqu'une période sera terminée.

Une même graine produira la même suite, il faut donc choisir judicieusement la graine. Idéalement, celle-ci devrait être aléatoire mais comme l'algorithme de génération ne peut pas s'initialiser lui-même, cela sous-entend que le hasard de la graine doit être déterminée par un autre moyen. On utilise généralement des moyens physiques comme, par exemple, un temps d'accès disque, le temps écoulé depuis le démarrage de l'application ou les mouvements d'une souris (on parle alors d'*accumulation d'entropie*).

Les générateurs pour lesquels la graine est initialisée par ces biais sont appelés « générateurs hybrides ». Ceux-ci, en plus de sélectionner une graine depuis un système aléatoire ou chaotique, vont régulièrement modifier la graine pour éviter que la périodicité ne nuise à la qualité du hasard généré.

3. Qualité des générateurs

Les générateurs de nombres pseudo-aléatoires sont aussi définis par d'autres paramètres qui permettent d'en mesurer la qualité. En effet, la génération de nombre est assimilée à une source d'information qui doit se rapprocher le plus possible d'une source sans redondance et sans mémoire, ce qui sous-entend une forte entropie et une distribution uniforme des probabilités. De plus, l'efficacité de l'algorithme se mesure dans sa capacité à fonctionner correctement quelle que soit la graine ou les paramètres utilisés.

Un générateur de nombres pseudo-aléatoires est qualifié de cryptographique lorsque sa qualité est suffisante pour justifier son application en cryptographie. En effet, de nombreux algorithmes de cryptage se basent sur des séquences aléatoires et sont donc directement impactés par les défauts potentiels du générateur.

La qualité des générateurs se mesure donc, d'une part, au choix de ses paramètres et, d'autre part, au moyen de tests statistiques qui vont éprouver l'aléatoire du générateur pour vérifier que celui-ci tend vers le véritable hasard.

4. Un générateur historique : carré médian

La méthode dite du carré médian (middle-square) fut proposée en 1946 par John Von Neuman et est considérée comme la première tentative de générer des nombres pseudo-aléatoires. Elle consiste en une opération très simple opérée sur un nombre de $2k$ chiffres. On élève au carré ce nombre et l'on prend comme terme suivant de la séquence pseudo aléatoire, le nombre formé des $2k$ chiffres du milieu du résultat précédent.

n	x_n	x_{n+1}
0	12	0**14**4
1	14	0**19**6
2	19	0**36**1
3	36	1**29**6
4	…	…

Dans la méthode originale, Von Neumann utilisait des nombres de 10 chiffres. Toutefois, la période du carré médian est faible. La qualité des sorties dépend de la graine, or « 0000 » produit toujours la même séquence et constitue donc un *état absorbant* de l'algorithme. Von Neumann en était conscient, mais craignait que des retouches à priori nécessaires n'apportent d'autres vices cachés.

Sur l'ordinateur ENIAC qu'il utilisait avec sa méthode, il obtenait une génération 200 fois plus rapide que les résultats obtenus avec des cartes perforées. Selon Von Neumann, les générateurs basés sur du matériel ne pouvaient pas fonctionner correctement car ils ne stockaient pas les résultats (et on ne pouvait donc pas les vérifier).

La méthode de Von Neumann montra vite ses limites lors d'applications utilisant des méthodes statistiques comme celle de Monte Carlo. De fait le générateur middle-square n’est plus vraiment utilisé.

V. Générateurs à congruence linéaire

1. Principe général

Cette famille de générateurs, introduite en 1948 par D. H. Lehmer sous une forme réduite, est une des plus utilisées à l'heure actuelle. Elle repose sur une simple formule de récurrence :

$$x_{n+1} = (a \times x_n + b)\ mod\ m$$

Ici, x_0 représente la graine du générateur et son choix est généralement effectué par des opérations dites d'accumulation d'entropie qui consistent à exploiter des données physiques variables (temps entre deux accès disques, clics de souris, saisie clavier, tailles de fichiers, bruit d'une résistance, température d'un composant, etc.) pour modifier régulièrement la graine du générateur

Les termes de la suite sont compris entre 0 et $m - 1$. Comme le nombre de valeurs est fini, la suite est amenée à se répéter au bout d'un certain temps, on dit qu'elle est *ultimement périodique*.

Note : Si $b = 0$ on parle de *générateur multiplicatif*.

Choix des constantes

La qualité du générateur dépend directement des constantes a, b et m. Par exemple, si l'on pose :

$$\begin{cases} a = 25 \\ b = 16 \\ m = 256 \end{cases}$$

On obtient :

- avec $x_0 = 10$, la suite : 10, 10, 10, 10, 10, ...
- avec $x_0 = 12$, la suite : 12, 60, 236, 28, 204, 252, 172, 220, 140, 188, 108, 156, 76, 124, 44, 92, **12, 60, 236, 28, 204, 252, 172**, ...

Nous pouvons voir que ces deux suites sont, non seulement, d'un aléatoire douteux mais qu'en plus leur période est inférieure à m (elles ne balayent pas l'ensemble des valeurs possibles entre 0 et $m-1$).

Régles de Knuth

Hugo Foulon a écrit, en 1985, dans sa thèse nommée *Les aléas du hasard* qu'un générateur à congruence linéaire était de bonne qualité s'il respectait les règles de Knuth.

En effet, pour qu'un générateur possède une période égale à m, Donald E. Knuth a établi qu'il fallait que :

- c soit premier avec m (pgcd de c et m égal à 1).
- pour tout nombre premier p divisant m, $a-1$ soit un multiple de p.
- $a-1$ soit un multiple de 4 si m est un multiple de 4.

Note : Ces règles ne s'appliquent pas forcément sur les générateurs multiplicatifs.

Knuth remarque aussi que si m est une puissance de 2 alors le bit de poids faible des nombres produits vaut alternativement 0 ou 1. Toutefois, ce n'est pas le seul cas où cela peut se produire.

Enfin, il est important de préciser que les règles de Knuth permettent d'assurer que le générateur est de période égale à m mais pas de garantir la qualité globale des nombres générés.

Exemple : $x_{n+1} = \left(31\,415\,821 x_n + 1\right) mod\ 100\,000\,000$ (citée par R. Sedgewick)
Cette suite respecte les critères évoqués précédemment mais, pour $x_0 = 0$ on obtient :

1, 31415822, 40519863, 62952524, 25482205, 90965306, 70506227, 68173688, 12779129, 29199910, 45776111, 9252132, 22780373, 20481234, 81203115, ...

Quelques générateurs congruentiels linéaires

Algorithme	a	B	M	
RANDU	$65539\ (2^{16}+3)$	0	2^{31}	Un très mauvais générateur implanté sur les IBM System/370.
Générateur du Turbo Pascal	129	907633385	2^{32}	On sait que $X_{n+1} > X_n$ dans 1 cas sur 129.
Standard minimal*	16807	0	$2^{31}-1$	
Générateur de Marsaglia*	69069	0	2^{32}	Utilisé sur le VAX.
Générateur de Knuth&Lewis*	1664525	1013904223	2^{32}	
Générateur de Haynes*	6364136223846793005	0	2^{64}	

$*$ *Générateur de bonne qualité*

2. Extension matricielle

Il existe une extension matricielle des générateurs à congruence linéaire. On parle alors de générateurs à congruence matricielle :

$$X_{n+1} = (A \times X_n + B)\ mod\ m$$

Ici, X_n et X_{n+1} sont des vecteurs de dimension k tandis qu'A et B sont des matrices de taille $k \times k$.

Ces générateurs sont souvent utilisés sous une forme réduite c'est-à-dire sans le membre B. Nous les retrouverons un peu plus loin lorsque nous traiterons les générateurs GFSR.

3. Générateur de Tausworthe et LFSR

Le générateur de Tausworthe est une extension du générateur congruentiel linéaire qui consiste à ne plus utiliser seulement x_{n-1} pour fabriquer un nouvel élément x_n mais plutôt un ensemble de valeurs précédentes :

$$x_n = (a_1 \times x_{n-1} + \cdots + a_k \times x_{n-k})\ mod\ m \text{ avec } n \geq k$$

A cette récurrence on associe le polynôme caractéristique suivant :

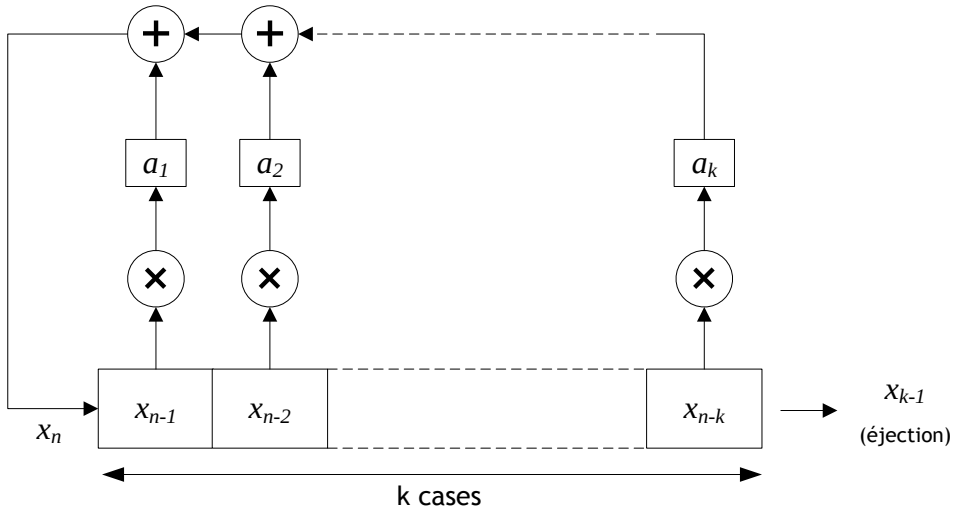
$$f(x) = x_k - a_1 \times x_{k-1} - \cdots - a_k$$

La période du générateur est de $m^k - 1$ si et seulement si le polynôme $f(x)$ est primitif (tous les coefficients du polynôme sont premiers entre eux).

Le LFSR est simplement un générateur de Tausworthe où $m = 2$, c'est-à-dire adapté à une utilisation informatique (les x_i deviennent des bits) car les opérations modulo 2 peuvent se traduire directement par un opérateur XOR (ou exclusif) :

$$x_n = a_1 \times x_{n-1} \oplus \ldots \oplus a_k \times x_{n-k}$$

De ce fait, les coûteuses opérations modulo 2 et de mémorisation des valeurs antérieures pourront se traduire par l'utilisation de composants électroniques spécialisés appelés "registres à décalage linéaire" (Linear Feedback Shift Registers) dont le fonctionnement peut être résumé par le schéma suivant :



On parle ici de registre à décalages car il s'agit d'un ensemble de k cases duquel l'élément de queue (dernier élément) est éjecté lors de l'ajout d'un nouvel élément de tête (premier élément).

De plus, il est possible de réduire les calculs en fixant certains a_i à 0 pour ne conserver que deux termes antérieurs. On parle alors de registre LFSR $R(k,r)$ dont la récurrence se traduit sous la forme suivante :

$x_n = a_r \times x_{n-r} \oplus a_k \times x_{n-k}$ avec $r < k \leq n$, $a_{i \neq r,k} = 0$ et $a_r = a_k = 1$

La version généralisée de ceci s'appelle une congruence additive :

$$x_n = (a_r \times x_{n-r} + a_k \times x_{n-k}) \; mod \; m$$

4. Générateur de Fibonacci

La méthode de Fibonacci est une méthode dérivée des générateurs à congruence additive, basée sur la suite de Fibonnacci :

$$\mathcal{F}_n = (\mathcal{F}_{n-1} + \mathcal{F}_{n-2}) \; mod \; m \text{ avec } \mathcal{F}_0, \mathcal{F}_1 \text{ fixés (graines)}$$

$\mathcal{F}_0$	$\mathcal{F}_1$	$\mathcal{F}_2$	$\mathcal{F}_3$	$\mathcal{F}_4$	$\mathcal{F}_5$	$\mathcal{F}_6$	$\mathcal{F}_7$	$\mathcal{F}_8$	$\mathcal{F}_9$	$\mathcal{F}_{10}$	$\mathcal{F}_{11}$	$\mathcal{F}_{12}$	$\mathcal{F}_{13}$	$\mathcal{F}_{14}$
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377

$\mathcal{F}_{15}$	$\mathcal{F}_{16}$	$\mathcal{F}_{17}$	$\mathcal{F}_{18}$	$\mathcal{F}_{19}$	$\mathcal{F}_{20}$	$\mathcal{F}_{21}$	$\mathcal{F}_{22}$	$\mathcal{F}_{23}$	$\mathcal{F}_{24}$	$\mathcal{F}_{25}$	…
610	987	1597	2584	4181	6765	10946	17711	28657	46368	75025	…

La suite de Fibonacci originale.

$$\mathcal{F}_n = \frac{1}{\sqrt{5}}\left(\varphi^n - \left(-\frac{1}{\varphi}\right)^n\right) \text{ où } \varphi \text{ est le nombre d'or}$$

Expression fonctionnelle pour $\mathcal{F}_0 = 0, \mathcal{F}_1 = 1$

Evidemment, la qualité de la génération est médiocre car les nombres consécutifs sont fortement corrélés.

5. GFSR

Il est possible de représenter le fonctionnement du générateur de Tausworthe sous forme matricielle :

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ x_5 & x_4 & x_3 & x_2 & x_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} mod \; m \text{ avec } k = 5$$

Tout comme le générateur de Tausworthe est une généralisation des générateurs à congruence linéaire, il est possible d'effectuer la même généralisation au niveau des générateurs à congruence matricielle.

Ainsi, les éléments manipulés par le générateur ne sont plus des valeurs mais des vecteurs de taille L : $X_i = \{x_{i,j}\} = \{x_{i,0}, \dots, x_{(i,L-1)}\}$. La récurrence du générateur se formule donc ainsi :

$$X_n = (A_1 \times X_{n-1} + \cdots + A_k \times X_{n-k}) \; mod \; m \text{ avec } n \geq k$$

La période d'un tel générateur est de m^{k-1}.

La représentation matricielle de ce nouveau générateur est sensiblement la même que pour un générateur de Tausworthe, mais étendue à des vecteurs.

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \end{bmatrix} = \begin{bmatrix} 0 & I & 0 & 0 & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \\ A_5 & A_4 & A_3 & A_2 & A_1 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} mod \; m \text{ avec } k = 5$$

Le générateur GFSR (General Feedback Shift Registers) est une implémentation de ceci, avec $m = 2$. Il a été mis au point dans le but de pouvoir manipuler des ensembles de bits (des mots binaires) plutôt qu'un seul.
De facto, la valeur décimale u_n associée à un mot de L bits X_n s'obtient par la relation suivante :

$$u_n = \frac{\sum_{j=0}^{L-1} x_{n,j} \times 2^j}{2^L}$$

Tout comme le générateur de Tausworthe, le GFSR peut être réduit en $R(k, r)$:

$$X_n = A_r \times X_{n-r} \oplus A_k \times X_{n-k} \text{ avec } r < k \leq n, \; A_{i \neq r,k} = 0 \text{ et } A_r = A_k = I_{L \times L}$$

Ou, en version généralisée :

$$X_n = \left(A_r \times X_{(n-r)} + A_k \times X_{n-k}\right) mod \; m$$

6. TGFSR

En 1992, Makoto Matsumoto et Yoshiharu Kurita ont apportés des modifications au GFSR pour en augmenter significativement la période. Le terme A_k, qui était auparavant une matrice identité $L \times L$ est remplacé par une matrice particulière de même taille qui va « twister » la récurrence.

$$X_n = \left(X_{n-(k-r)} + A \times X_{n-k}\right) mod\ 2$$

r est une constante et A est une matrice $L \times L$. Le générateur atteint une période maximale de $2^{kL} - 1$ si et seulement si le polynôme caractéristique de la matrice A, noté $\phi(t)$, est primitif pour $t = t^k + t^r$.

La matrice A la plus connue (décrite par Matsumoto et Kurita) est :

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 \\ a_L & a_{L-1} & a_{L-2} & \dots & a_1 \end{bmatrix}$$

7. Une application pratique : Mersenne Twister

Mersenne Twister a été mis au point en 1997 par Makoto Matsumoto et Takuji Nishimura. Il se base sur un TGFSR dont la récurrence est modifiée comme ceci :

$$X_n = \left(X_{n-(k-r)} + A \times (X^u_{n-k} | X^l_{n-k+1}\right) mod\ 2 \text{ avec } 0 \leq r \leq k$$

On définit implicitement une constante p telle que $0 \leq p \leq L - 1$ qui représente un nombre de bits. X^u_{n-k} représente les $L-p$ premiers bits de X_{n-k} tandis que X^l_{n-k+1} représente les p derniers bits de X_{n-k+1}. L'opérateur « | » représente ici la concaténation des deux vecteurs de bits.

Etant basé sur un TGFSR, certains des éléments du Mersenne Twister doivent suivre les même règle. C'est notamment le cas de la matrice A.
On remarque aussi que si $p = 0$, on obtient la même récurrence qu'un TGFSR. En cela, Mersenne Twister est une généralisation de ce générateur. Si, en plus, A est égal à une matrice identité alors le générateur se comporte comme un GFSR.

La période de Mersenne Twister est inférieure à celle d'un TGFSR. En effet, celle-ci est de $2^{kL-p} - 1$.

Pour améliorer l'uniformité de la distribution des valeurs générées, les concepteurs de Mersenne Twister ont proposés d'y ajouter une opération de « tempering » décrite sous la forme d'une matrice $L \times L$ supplémentaire :

$$X_n = (T \times X_{n-(k-r)} + A \times (X_{n-k}^u | X_{n-k+1}^l))\ mod\ 2$$

Informatiquement, la matrice T est assimilée à la fonction $x \mapsto z = x \times T$ ce qui peut se traduire par :

$$\begin{aligned} y &= x \oplus (x \gg u) \\ y &= y \oplus \big((y \ll s)\text{AND } b\big) \\ y &= y \oplus \big((y \ll t)\text{AND } c\big) \\ z &= y \oplus (y \gg l) \end{aligned}$$

Ici, s et t sont des constantes entières et b et c sont des vecteurs de L bits fixés.

MT19937

L'implémentation la plus connue du Mersenne Twister est le MT19937. Ses paramètres sont :

$L = 32 \quad k = 623$

$p = 31 \quad r = 397$

$A(x) = (x \gg 1) \oplus \begin{cases} 0 \text{ si } x\ mod\ 2 = 0 \\ 2567483615 \text{ si } x\ mod\ 2 = 1 \end{cases}$

Et, pour le tempering :

$L = 11 \quad k = 7 \quad b = 2636928640$

$p = 18 \quad r = 15 \quad c = 4022730752$

La période de ce générateur est $2^{19937} - 1$, qui est un *nombre premier de Mersenne* (un nombre premier s'écrivant sous la forme $2^p - 1$, p étant premier). Ce générateur possède une excellente équidistribution dans 623 dimensions (voir Test Spectral).

MT19937 est très sensible en ce qui concerne son initialisation. Ses concepteurs préconisent d'exploiter la récurrence suivante pour déterminer les valeurs initiales :

$X_i = 1812433253 \times \left((X_{i-1} \oplus X_{i-1} \gg 30) + i\right)$ avec $i = [\![1, \dots, n-1]\!]$ et X_0 fixé arbitairement.

8. Avantages / défaut

Le principal avantage des générateurs à congruence linéaire réside dans leur rapidité de calcul et dans leur aléatoire de qualité suffisante pour des applications courantes (jeux, etc.) à partir du moment où les paramètres sont correctement choisis.
De plus, leur période (notamment en ce qui concerne les GFSR) est extrêmement élevée.

Toutefois, leurs faiblesses se font sentir sur des applications plus critiques où il est important que l'on ne puisse pas prédire les termes suivants de la suite générée comme c'est le cas en cryptographie.
En effet, des algorithmes permettent de retrouver le polynôme caractéristique d'un générateur à congruence linéaire. Par exemple, l'algorithme de Berkelamp-Massey est en mesure de réaliser cette opération en n'ayant connaissance que d'un ensemble de $2k$ valeurs précédentes générées.

Malgré toutes les règles à suivre, un générateur pseudo aléatoire doit être considéré comme un programme informatique : il n'est correct que jusqu'à ce qu'un bug soit détecté.

VI. Générateurs cryptographiques

Les générateurs cryptographiques sont décrits comme sûr pour des applications cryptographiques. Il est généralement très difficile d'en prédire la sortie car ceux-ci se basent souvent sur les algorithmes à clé publique existant en cryptographie.

Ces algorithmes se basent sur des fonctions dîtes "à sens unique". Ces fonctions sont conçues de telles sorte que :

- L'obtention de l'image de x à partir de x soit très simple.
- La recherche de x à partir de l'image de x est un problème mathématique réputé difficile, ou informatiquement trop complexe (= trop lent).

1. Algorithme Blum-Blum-Shub (BBS)

Blum Blum Shub (BBS) est un algorithme de génération de nombres pseudo aléatoires proposé en 1986 par Lenore Blum, Manuel Blum et Michael Shub, d'où son nom. En soi, le générateur n'est pas approprié pour toutes les applications, car il est relativement lent. Cependant, son niveau de sécurité élevé en fait un bon candidat pour des applications cryptographiques.

a. Principe

On calcule la sortie de l'algorithme BBS en itérant la suite :

$$x_{n+1} = (x_n)^2 \ mod \ m \text{ avec } x_0 \text{ la graine}$$

$m = p \times q$ est le produit de deux grands nombres premiers congrus à 3 modulo 4, c'est-à-dire que $p = 4 \times k_1 + 3$ et $q = 4 \times k_2 + 3$.

La sortie de l'algorithme est le bit le moins significatif de x_n, mais peut aussi dans certaines variantes, être issus des derniers bits de x_n.

Exemple : Prenons $p = 11$, $q = 19$ et $s = 3$ (la graine). Nous pouvons nous attendre à une taille de cycle assez importante par rapport à des nombres aussi petits puisque $pgcd\big(\varphi(p-1), \varphi(q-1)\big) = 2$. Le générateur crée la séquence $x_1, \dots, x_6 = 9, 81, 82, 36, 42, 92$. Il ne restera plus qu'a prendre le bit le moins significatif de chaque nombre ainsi obtenu pour générer un nombre pseudo aléatoire. Dans notre cas : 110000 = 48.

b. Conclusion

Il a été prouvé que l'algorythme BBS était cryptographiquement sûr sous l'hypothèse qu'il soit difficile de déterminer si, modulo un entier composé, un nombre est un carré ou non.

Par la suite il a également été démontré qu'il était cryptographiquement sûr, sous l'hypothèse que le problème de la factorisation soit difficile, et qu'au plus $log(log(m))$ bits de poids faible de chaque x_n soient sortis à chaque itération. Dans ce cas, pour M très grand, il sera aussi difficile de distinguer la suite produite d'une suite réellement aléatoire que de factoriser m, comme pour l'algorithme de cryptage RSA.

Après 20 ans de recherches, aucune autre méthode efficace que la factorisation de n n'a été publié pour casser Blum Blum Shub.
Les limites actuelles de factorisation concernent des nombres de 200 chiffres environ (record 2005).

1. Algorithme Blum-Micali

Inventé par Silvio Micali et Manuel Blum, l'algorithme de Blum-Micali est basé sur le même principe que l'algorithme de chiffrement à clé publique El Gamal.

a. Exponentiation (ou puissance) modulaire

Soit a un élément de $\frac{\mathbb{Z}}{n\mathbb{Z}^*}$, l'exponentiation modulaire est le morphisme défini par :

$\frac{\mathbb{Z}}{n\mathbb{Z}} \mapsto \frac{\mathbb{Z}}{n\mathbb{Z}}$

$b \to a^b \ [mod\ n]$

Algorithme de calcul

Si b = 0

Retourner 1

Sinon

d := recalculer puissance modulaire $\left(a^{\left\lfloor \frac{b}{2} \right\rfloor}, \left\lfloor \frac{b}{2} \right\rfloor, n\right)$

d := (d × d) mod n

Si b est impair

d = (d × a) mod n

Fin Si

Retourner d

Fin Si

Cet algorithme se base sur une décomposition de b en carrés successifs pour diminuer le nombre d'opérations au niveau informatique.

a. Logarithme discret

Rappel : Un générateur du groupe multiplicatif $\frac{\mathbb{Z}}{n\mathbb{Z}^*}$ est un nombre g tel que $\{g^i, i \in \mathbb{N}\} = \frac{\mathbb{Z}}{n\mathbb{Z}^*}$.

Soit a un élément de $\frac{\mathbb{Z}}{n\mathbb{Z}^*}$.

Si a est un générateur du groupe multiplicatif $\frac{\mathbb{Z}}{n\mathbb{Z}^*}$ alors la fonction d'exponentiation est associée à sa fonction de décodage.

Pour c dans $\frac{\mathbb{Z}}{n\mathbb{Z}}$, le plus petit entier positif b tel que $a^b = c\,[mod\, n]$ est appelé *logarithme discret* (ou *index*) *en base* a *de* b *modulo* n.

$\frac{\mathbb{Z}}{n\mathbb{Z}} \mapsto \frac{\mathbb{Z}}{n\mathbb{Z}}$

$c \to \log_a c\,[mod\, n]$

b. Algorithme

- Soit p un nombre premier suffisamment grand pour que le problème du logarithme discret soit difficile dans $\frac{\mathbb{Z}}{p\mathbb{Z}^*}$
- Soit $\alpha = \frac{\mathbb{Z}}{p\mathbb{Z}^*}$ une racine primitive (= générateur).

- Soit f la fonction d'exponentiation modulaire $f(x) = \alpha^x \, mod \, p$.
- Soit B la fonction à valeurs dans $\{0,1\}$ définie telle que :

  $B(x) = 1$ si $0 \leq \log_\alpha(x) \leq \frac{p-1}{2}$

  $B(x) = 0$ si $\log_\alpha(x) > \frac{p-1}{2}$
- Soit la séquence $S = \{x_0, x_1, \dots, x_k\}$ où x_0 est un élément non nul quelconque de $\mathbb{F}_p$ (la graine du générateur) et $x_i = f(x_{i-1})$ avec $i > 0$.
- Enfin, on génère la séquence $B = \{b_1, \dots, b_k\}$ où $b_i = B(x_i)$ avec $i > 0$.

c. Conclusion

Le problème du logarithme discret (noté DLP pour *Discrete Logarithm Problem*) est le calcul inverse de la puissance modulaire.
Si la puissance modulaire est calculable en un temps raisonnable comme nous avons pu le voir pour l'algorithme présenté plus haut, ce n'est pas le cas du logarithme discret.

En effet, la résolution du problème du logarithme discret se base sur le théorème suivant :

Si g est un générateur de $\frac{\mathbb{Z}}{n\mathbb{Z}^*}$, alors $\forall \, x, y \in \mathbb{N} : \alpha^x \equiv \alpha^y \, [mod \, n]$ si et seulement si $x = y \, [mod \, \phi(n)]$.

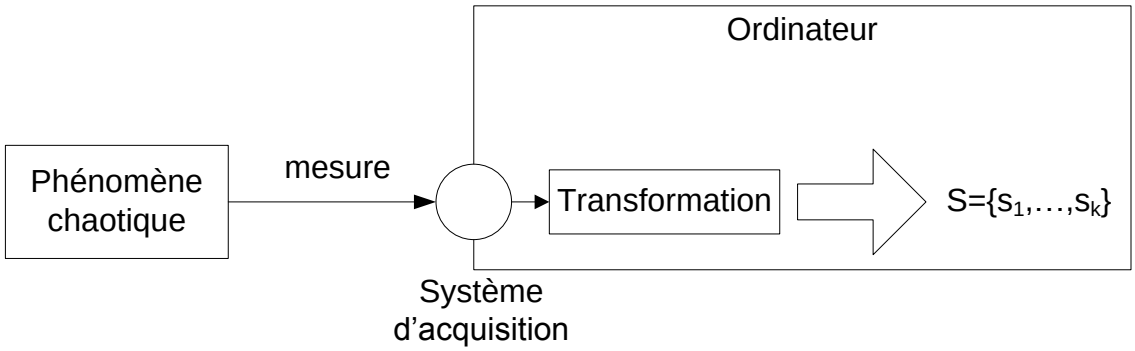
Toutefois, étant donné y, il est difficile de calculer x tel que $\alpha^y = y$. La seule méthode simple consiste en l'énumération exhaustive de tous les x possibles. Ainsi, si $n = 10^{150}$, l'énumération va demander 10^{150} opérations, ce qui est absolument impossible en temps raisonnable à l'heure actuelle.

Ainsi, l'exponentiation modulaire et son opération inverse, le logarithme discret, sont une fonction à sens unique. De fait il est impossible de prédire l'état suivant du générateur sans connaître la séquence $\{x_0, x_1, \dots, x_k\}$.

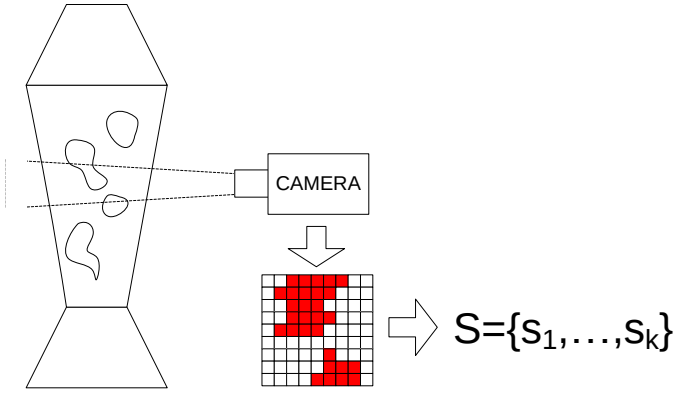
VII. Générateurs physiques

Les générateurs physiques sont des types très particuliers de générateurs de nombres. Leur aléatoire est basée sur une variable imprévisible et non déterministe (ou du moins trop chaotique pour être déterminée rapidement) qui est fournie par une source physique externe.

Nous ne rentrerons pas dans les détails ici car ces générateurs sont, d'une part, très simples à comprendre et, d'autre part, ne peuvent plus être considérés comme pseudo-aléatoires.



Tout phénomène chaotique peut faire l'affaire (lumière + capteur à photon, source radioactive + capteur de radiations, etc.).



Le générateur LavaRand.

VIII. Tests d'un générateur

Aussi important que l'étude des générateurs pseudo-aléatoires, les tests doivent permettre de mettre en évidence des problèmes liés à la distribution des valeurs de sorties du générateur. De facto, ils ont une importance capitale pour assurer qu'un générateur est de bonne qualité.

1. Test de premier secours

Il s'agit du test le plus simple possible. Il consiste à calculer la moyenne, la variance et le facteur d'autocorrélation de la suite $U = \{u_1 = \frac{s_1}{m}, \dots, u_n = \frac{s_n}{m}\}$ calculée à partir de la suite $S = \{s_1, \dots, s_n\}$ produite par le générateur. Chaque $u_i \in [0,1[$ puisque les $s_i \in [\![0, m-1]\!]$.

Pour une suite aléatoire, ces facteurs tendent vers des valeurs idéales qu'il suffit donc de comparer aux valeurs calculées pour la suite U.

Remarque : On peut coupler, au test de ces trois valeurs, un test d'entropie qui consiste à analyser aussi la valeur de l'entropie pour la suite (voir prérequis).

- Moyenne : $\mu = \frac{1}{n}\sum_{i=1}^{n} u_i$
- Variance : $v = \frac{1}{n}\sum_{i=1}^{n} u_i^2 - \mu^2$
- Autocorrélation : $r = \frac{1}{n}\sum_{i=1}^{n-1} u_i \times u_{i+1}$

Idéalement, on a $\mu = \frac{1}{2}, v = \frac{1}{12}, r = \frac{1}{4}$

Toutefois, ce test n'induit qu'une analyse très sommaire de la qualité du générateur. En effet, la dernière suite de l'exemple ci-dessous se rapproche assez des valeurs théoriques mais n'est absolument pas aléatoire.

Suite (n = 10)	Moyenne	Variance	Autocorrélation
0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5	0.5	0	0.225
0, 0, 0, 0, 0, 1, 1, 1, 1, 1	0.5	0.25	0.4

0, 1, 0, 1, 0, 1, 0, 1, 0, 1	0.5	0.25	0
0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9	0.45	0.0825	0.33

1. Test du χ^2 (khi-2)

a. Fonction gamma

La fonction gamma est une fonction complexe prolongeant la fonction factorielle à l'ensemble des nombres complexes (à l'exception de certains points).

Pour $z \in \mathbb{C}$ tel que $Re(z) > 0$, on définit la fonction suivante, notée par la lettre grecque Γ (gamma majuscule) :

$$\Gamma : z \mapsto \int_0^{+\infty} t^{z-1} e^{-t} dt$$

En intégrant par parties, on montre que :

$$\Gamma(z+1) = z\Gamma(z)$$

a. Loi multinomiale

La loi multinomiale est une généralisation de la loi binomiale qui concerne le nombre de succès de N épreuves de Bernoulli indépendantes. Mais la loi Binomiale ne gère que des résultats binaires (lancer d'une pièce par exemple), tandis que la loi multinomiale est applicable par exemple aux lancés de dés à six faces.

Pour rappel, la loi binomiale est définie par :

$$\mathcal{P}(X = x) = \frac{n!}{x!\,(n-x)!} p^x (1-p)^{n-x}$$

Dans le cas multinomial à m résultats possibles au lieu de deux, les variables deviennent N_i et correspondent aux probabilités p_i où $i = \{1, \dots, m\}$, avec les contraintes :

$$\sum_{i=1}^{m} N_i = n \qquad \sum_{i=1}^{m} p_i = 1$$

La fonction de probabilité s'écrit alors, sous la condition portant sur la somme des variables :

$$\mathcal{P}(N_1 = n_1, \dots, N_m = n_m) = \frac{n!}{n_1! \dots n_m!} p_1^{n_1} \dots p_m^{n_m}$$

Lorsque la variable aléatoire N_i devient assez grande, le théorème de la limite centrale montre qu'elle est raisonnablement approchée par une variable normale à laquelle correspond la variable centrée réduite :

$$\frac{N_i - np_i}{\sqrt{np_i(1-p_i)}}$$

Si ces variables étaient indépendantes, la somme de cette variable centrée réduite suivrait une loi du χ^2 à m degrés de liberté.

$$\sum_{i=1}^{m} \frac{(N_i - np_i)^2}{np_i(1-p_i)}$$

Du fait de la contrainte linéaire qui s'applique, la variable centrée réduite suit une loi du χ^2 à $m-1$ degrés de liberté qui est à la base du test du khi-2.

$$\sum_{i=1}^{m} \frac{(N_i - np_i)^2}{np_i}$$

b. Loi du χ^2

La loi du χ^2 est une loi à densité de probabilité. Cette loi est caractérisée par un paramètre appelé degré de liberté, à valeur dans l'ensemble des entiers naturels (non nuls).

Soit $X_1, \dots, X_k$, k variables aléatoires indépendantes de même loi normale centrée et réduite, alors la variable X vaut :

$$X := \sum_{i=1}^{k} X_i^2$$

Soit X une variable aléatoire suivant une loi du χ^2 à k degrés de liberté, on notera $\chi^2(k)$ la loi de X de degré k.

La densité de X, pour tout t positif, notée f_x vaut :

$$f_x(t) = \frac{1}{2^{\frac{k}{2}} \times \Gamma\left(\frac{k}{2}\right)} t^{\frac{k}{2}-1} \times e^{-\frac{t}{2}}$$

Remarque : L'espérance mathématique de X vaut k et sa variance vaut $2k$.

c. Fonctionnement

Généralités

La statistique mathématique a pour but la description d'une population dont on ne connaît qu'un nombre relativement petit d'individus et pour ce faire, on associe une loi de probabilité à cette population. Et, mis à part pour certains problèmes de physiques fondamentale ou certains problèmes élémentaires, cette loi de probabilité est toujours inconnue.

Le test du χ^2 (dit « khi-deux » ou « khi-carré ») permet, partant d'une hypothèse et d'un risque supposé au départ, de rejeter l'hypothèse si la distance entre deux ensembles d'informations est jugée excessive, permettant ainsi de comparer ladite hypothèse à une loi de probabilité.

En 1900, un mathématicien britannique, Karl Pearson, mit en évidence qu'une grande différence entre la loi théorique et la mesure réelle a plus d'importance que plusieurs petites différences. Cela a donné le test du χ^2 qui est un cas particulier de test statistique d'hypothèse qui fournit une méthode pour déterminer la nature d'une répartition, qui peut être continue ou discrète.

Dans le cas d'une répartition discrète, on regarde le nombre d'événements ayant la même valeur discrète et c'est la fréquence d'apparition d'une valeur qui constitue la mesure. Lorsque le nombre de valeurs possibles est élevé, on est généralement amené à regrouper plusieurs valeurs dans une même classe, comme pour les valeurs continues, de manière à satisfaire la règle.

Un des problèmes importants est de savoir combien de mesures au minimum il faut faire pour bien comparer la loi théorique à la réalité. Une règle empirique couramment utilisée consiste à dire que chaque classe doit contenir au moins cinq événements puisque 80 % des classes doivent satisfaire la règle des cinq éléments

tandis que les autres doivent seulement être non vides. Si l'on est en dessous, cela signifie qu'il faut regrouper les classes, à condition que leur nombre initial et le nombre total d'observations soient suffisants.

Principe du test

A la base du test, il y a une hypothèse de départ qui suppose que toutes les données considérées suivent une même loi probabiliste. Ces données réparties en classes, notées m, on va chercher à déterminer le nombre de degrés du problème, à partir du nombre de classes $m-1$, et se donner une marge d'erreur. Cette marge d'erreur est souvent 5% pour des raisons qui tiennent moins du raisonnement que de l'habitude. Ensuite, à l'aide d'une table donnée par la *Loi du* χ^2, on déduit la distance critique, fonction de $m-1$, le nombre de degrés de liberté et α la marge d'erreur (5%).

Degrès	$\alpha = 0{,}10$	$\alpha = 0{,}05$	$\alpha = 0{,}01$	$\alpha = 0{,}001$
1	2.706	3.841	6.635	10.828
2	4.605	5.991	9.210	13.816
3	6.251	7.815	11.345	16.266
4	7.779	9.488	13.277	18.467
5	9.236	11.070	15.086	20.515
6	10.645	12.592	16.812	22.458
7	12.017	14.067	18.475	24.322
8	13.362	15.507	20.090	26.124
9	14.684	16.919	21.666	27.877
10	15.987	18.307	23.209	29.588
11	17.275	19.675	24.725	31.264
12	18.549	21.026	26.217	32.909
13	19.812	22.362	27.688	34.528
14	21.064	23.685	29.141	36.123
15	22.307	24.996	30.578	37.697
16	23.542	26.296	32.000	39.252
17	24.769	27.587	33.409	40.790
18	25.989	28.869	34.805	42.312
19	27.204	30.144	36.191	43.820
20	28.412	31.410	37.566	45.315
21	29.615	32.671	38.932	46.797
22	30.813	33.924	40.289	46.268
23	32.007	35.172	41.638	49.728

24	33.196	36.415	42.980	51.179
25	34.382	37.652	44.314	52.620
30	40.26	43.77	50.89	59.70
35	46.06	49.80	57.34	66.62
40	51.81	55.76	63.69	73.40
45	57.51	61.66	69.96	80.08
50	63.17	67.50	76.15	86.66
60	74.40	79.08	88.38	99.61
70	85.53	90.53	100.43	112.32
80	96.58	101.88	112.33	124.84
90	107.57	113.15	124.12	137.21
100	118.50	124.34	135.81	149.45

En fonction du nombre de degrés de liberté et du risque d'erreur α on obtient la valeur de la distance critique qui possède la probabilité d'être dépassée.

Ceci fait, on calcule algébriquement la distance entre les ensembles d'informations (phénomène et loi de probabilité) par la somme suivante.

$$\sum_{i=1}^{m} \frac{(n_i - n.p_i)^2}{n.p_i}$$

On suppose l'indépendance des n valeurs considérées regroupées dans m classes. L'effectif de chaque classe notée i est une variable aléatoire définie par la loi multinomiale. La loi de probabilité testée permet de définir pour chaque classe la probabilité p_i. Il est de plus courant que les paramètres qui caractérisent la loi de probabilité (moyenne, variance, ...) soient inconnus au moment du test. Les données sont donc utilisées pour estimer ceux-ci, mais il est alors nécessaire de diminuer le nombre de degrés de liberté du nombre de paramètres estimé.
La somme représente la distance entre les données et la loi de probabilité supposée. C'est une réalisation d'une variable aléatoire qui dérive d'une *Loi du* χ^2 à $m - 1$ degrés de liberté.

Si cette distance est supérieure à la distance critique, on en conclut que l'hypothèse de départ est nulle et doit être rejetée. Le risque, ou marge d'erreur donnée au départ, est celui de donner une réponse fausse lorsque les fluctuations de l'échantillonnage sont seules en cause. Evidemment, un rejet est une réponse

négative pour les tests d'adéquation et d'homogénéité, mais pas aux tests d'indépendance.

Exemple : On lance un dé soixante fois et on obtient les résultats suivants :

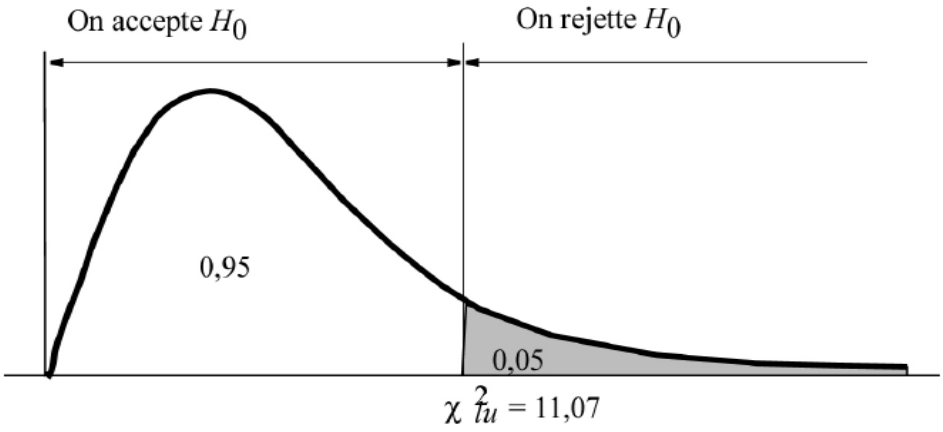
Face	1	2	3	4	5	6
Effectif	6	15	8	5	4	22

On considère un risque de 5% et on cherche à savoir si le dé est truqué.
Théoriquement, chaque face a une chance sur six d'apparaitre, on en déduit donc un effectif théorique de 10 pour chaque face.

On effectue les calculs nécessaires :

Face / Classe m	Résultat n_i	Probabilités théoriques p_i	Effectifs théoriques	$n_i - n.p_i$	$(n_i - n.p_i)^2$	$\frac{(n_i - n.p_i)^2}{n.p_i}$
1	6	1/6	10	-4	16	1.6
2	15	1/6	10	5	25	2.5
3	8	1/6	10	-2	4	0.4
4	5	1/6	10	-5	25	2.5
5	4	1/6	10	-6	36	3.6
6	22	1/6	10	12	144	14.4
Somme	60	1	60 (n)	0		**25**

Nous avons six classes, or pour le degré de liberté du test est de $m - 1$ soit cinq.
Nous sommes donc en présence d'une loi du χ^2 à 5 degrés de liberté.
Si on reprend la table de la loi du Khi deux, on trouve la valeur 11.070 associée au risque de 5%, or le χ^2 observé est égal à 25. Nous dépassons donc le seuil des 5% d'erreur, l'hypothèse de départ est rejetée, nous concluons donc que le dé est truqué.



2. Test des suites croissantes (run up test)

Le test des suites croissantes a été proposé par Donald E. Knuth et consiste à dénombrer les suites de valeurs croissantes et à les classer par longueur.

Soit $\{r_i\}_{i\in[\![1,\dots,5]\!]}$ le nombre de séquences croissantes de taille i et r_6 le nombre de séquences croissantes de taille supérieure ou égale à 6.

A partir de ces mesures, on détermine une valeur, notée R, qui sera comparée à la distribution du khi-deux avec 6 degrés de liberté (voir test du khi-deux).

$$R = \frac{1}{n-6}\sum_{i=1}^{6}\sum_{j=1}^{6} a_{i,j}(r_i - n\times b_i)(r_j - n\times b_j)$$

Les coefficients a et b sont définis par les constantes suivantes :

a	J					
i	4529.4	9044.9	13568	18091	22615	27892
	9044.9	18097	27139	36187	45234	55789
	13568	27139	40721	54281	67852	83685
	18091	36187	54281	72414	90470	111580

b
$\frac{1}{6}$
$\frac{5}{24}$
$\frac{11}{120}$
$\frac{19}{720}$

	22615	45234	67852	90470	113262	139476	$\frac{29}{5040}$
	27892	55789	83685	111580	139476	172860	$\frac{1}{840}$

3. Run test

a. Loi normale

En probabilité, on dit qu'une variable aléatoire réelle X suit une loi normale (ou loi normale gaussienne, loi de Laplace-Gauss) d'espérance μ et d'écart type σ strictement positif (donc de variance σ^2) si cette variable aléatoire réelle X admet pour densité de probabilité la fonction φ définie, pour tout nombre réel x, par :

$$\varphi(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Une telle variable aléatoire est alors dite *variable gaussienne*. On note habituellement cela de la manière suivante :

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

Sa fonction de répartition, notée Φ, est :

$$\Phi(x) = \int_{-\infty}^{x} \varphi(t)dt = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$$

a. Fonctionnement

Proche du test précédent, le run test étudie le nombre de séquences (run) croissantes et décroissantes au sein de la sortie d'un générateur pseudo-aléatoire.

Soit r_c, le nombre de séquences croissantes, $r_d$le nombre de séquences décroissantes et $R = r_c + r_d$ le nombre de séquences dans un échantillon de n valeurs pseudo-aléatoires.

Soit X variable aléatoire qui associe une probabilité à chaque nombre de séquences possible (R) pour un échantillon de n nombres.
Lorsque n est très grand, X tend à suivre une loi normale de paramètres :

$$E(X) = 2 \times \frac{r_c \times r_d}{R} + 1$$

$$\sigma^2(X) = 2 \times \frac{r_c \times r_d \times (2 \times r_c \times r_d - R)}{(R-1)R^2} + 1$$

De là on peut calculer une valeur à comparer à la valeur théorique fournie par la loi normale. Le générateur passe le test si l'écart entre la valeur réelle et la valeur théorique est inférieur à une marge d'erreur fixée à l'avance (historiquement 5%).

4. Test de Kolmogorov-Smirnov

a. Fonction de répartition empirique

Une fonction de répartition empirique est une fonction de répartition qui attribue la probabilité $\frac{1}{n}$ à chacun des n nombres dans un échantillon donné.

Soit $(x_1, \ldots, x_n)$un échantillon de variables indépendantes et suivant la même loi de probabilité à valeurs dans $\mathbb{R}$ avec pour fonction de répartition $F(x)$.

La fonction de distribution empirique $F_n(x)$ basée sur l'échantillon $(x_1, \ldots, x_n)$est une fonction en escalier définie par :

$$F_n(x) = \frac{\text{nombre d'éléments dans l'échantillon} \leq x}{n} = \frac{1}{n}\sum_{i=1}^{n} I(X_i \leq x)$$

où $I(A)$ est la fonction indicatrice de l'événement A.

Une fonction indicatrice, est une fonction définie sur un ensemble E qui explicite l'appartenance ou non à un sous-ensemble F de E de tout élément de E.
Formellement, la fonction caractéristique d'un sous-ensemble F d'un ensemble E est une fonction :

$$\mathcal{X}_F : E \to \{0,1\}$$
$$x \mapsto \begin{cases} 1 \text{ si } x \in F \\ 0 \text{ si } x \notin F \end{cases}$$

a. Fonctionnement

Le test de Kolmogorov-Smirnov est un test d'ajustement à une loi continue, qui prend en compte l'ensemble des quantiles (points essentiels pris à des intervalles réguliers verticaux d'une fonction de distribution cumulative d'une variable aléatoire).

Le principe de mesurer l'écart maximum qui existe entre la fonction de répartition empirique observée sur n échantillons et la fonction de répartition théorique et de juger au vu de l'écart si l'échantillon est mauvais ou non.

Tout d'abord, nous allons définir un seuil d'erreur, comme pour le test du Khi deux, qui est lui aussi, généralement de 5%. A partir de ce seuil, on déterminera la distance critique $D_a(n)$ qu'un échantillon ne doit pas dépasser.

Taille de l'échantillon	Seuil critique $D_a(n)$				
	a = .20	a = .15	a = .10	a = .05	a = .01
1	.900	.925	.950	.975	.995
2	.684	.726	.776	.842	.929
3	.565	.597	.642	.708	.828
4	.494	.525	.564	.624	.733
5	.446	.474	.510	.565	.669
6	.410	.436	.470	.521	.618
7	.381	.405	.438	.486	.577
8	.358	.381	.411	.457	.543
9	.339	.360	.388	.432	.514
10	.322	.342	.368	.410	.490
11	.307	.326	.352	.391	.468
12	.295	.313	.338	.375	.450
13	.284	.302	.325	.361	.433
14	.274	.292	.314	.349	.418
15	.266	.283	.304	.338	.404
16	.258	.274	.295	.328	.392
17	.250	.266	.286	.318	.381
18	.244	.259	.278	.309	.371
19	.237	.252	.272	.301	.363
20	.231	.246	.264	.294	.356
25	.210	.220	.240	.270	.320
30	.190	.200	.220	.240	.290
35	.180	.190	.210	.230	.270
> 35	$\frac{1.07}{\sqrt{n}}$	$\frac{1.14}{\sqrt{n}}$	$\frac{1.22}{\sqrt{n}}$	$\frac{1.36}{\sqrt{n}}$	$\frac{1.63}{\sqrt{n}}$

Ensuite on va considérer un échantillon $(x_1, \dots, x_n)$ d'une loi inconnue P et partir de l'hypothèse de base H_0 que P a pour fonction de répartition F_0, ou F_0 est la fonction de répartition d'une loi continue donnée. Si l'hypothèse H_0 est correcte,

alors la fonction de répartition empirique F_n de l'échantillon doit être proche de F_0. Pour cela on va calculer :

$$D(F_0, F_n) = \max\left\{\left|F_0(X_i) - \frac{i}{n}\right|, \left|F_0(X_i) - \frac{i-1}{n}\right|\right\}$$

avec $i = 1 \dots n$

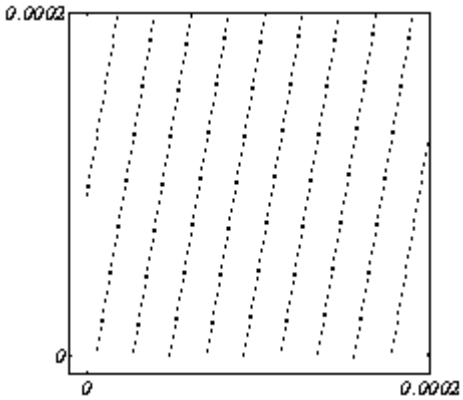
Si $D(F_0, F_n)_i > D_a(n)_i$, on considère l'échantillon comme mauvais.

Comme ce test mesure l'adéquation à une loi continue, le résultat permet d'apprécier la non-aléatoirité de l'échantillon. Aussi, si un échantillon échoue à ce test, cela signifie que son aléatoire est satisfaisant.

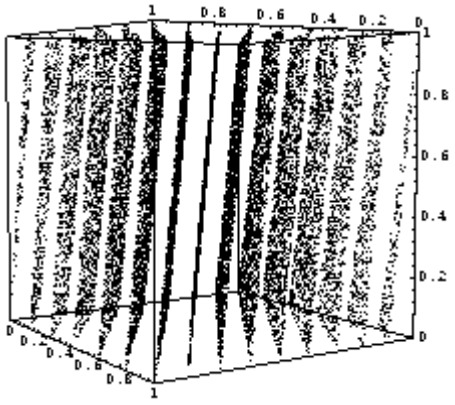
5. Test spectral

Le test spectral est décrit par Knuth comme le plus discriminant de tous. En effet, aucun générateur démontré mauvais n'a pu le réussir.

Très simple, la méthode consiste à étudier la distribution des valeurs générées dans une dimension k pour en vérifier la qualité.



Le générateur d'Apple, années 90 ($k = 2$)



Le générateur RANDU d'IBM, années 60 ($k = 3$)

Ces raies sont typiques des générateurs à congruences linéaires (on parle d'*effet Marsaglia*). Elles apparaissent forcément à un k donné ($k = 2$ pour l'exemple de gauche, $k = 3$ pour l'exemple de droite

De manière générale, le test spectral permet de déterminer l'écart d_k entre deux raies. Au plus cet écart est faible, au plus le générateur est de bonne qualité. Idéalement, l'expérience a montré qu'un bon générateur devait respecter cette relation :

$$d_k \leq m^{-\frac{1}{k}}$$

Déterminer d_k peut se faire graphiquement jusqu'à la dimension 3. Au-delà, d_k doit être déterminé par le calcul suivant et comparé à sa valeur théorique acceptable :

$$d_k = \frac{1}{v_k}$$

$$v_k = \min\left\{\sqrt{x_1^2 + x_2^2 + \cdots + x_k^2}\right\}$$

Par exemple, pour le générateur RANDU nous avons $v_2 = 23169.06$ et $v_3 = 10.86$ (au lieu de $v_3 = 812.75$ théorique) ce qui explique la si mauvaise qualité du générateur lors du test spectral en dimension 3.

6. Test par méthode de Monte Carlo

On appelle méthode de Monte-Carlo toute méthode visant à calculer une valeur numérique, et utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes. Le nom de ces méthodes, qui fait allusion aux jeux de hasard pratiqués à Monte-Carlo, a été inventé en 1947 par Nicholas Metropolis.

La méthode de Monte Carlo n'est donc pas, à proprement parler, un test. Toutefois, comme celle-ci permet de fournir des approximations qui dépendent de la qualité du générateur de nombre pseudo-aléatoires, il existe une relation directe entre la qualité de l'approximation et celle du générateur.

a. Vulgarisation

Etudions le problème de la superficie du lac.

Soit une zone rectangulaire dont les côtés sont de longueur connue. Au sein de cette aire se trouve un lac dont nous aimerions connaître la superficie.

Pour ce faire, on demande à un canonnier de tirer X coups de canons de manière aléatoire sur le terrain, puis on compte le nombre N de boulets tombés sur le terrain et l'on détermine le nombre $X - N$ de boulets tombés dans le lac.

On peut établir le rapport des superficies comme ceci :

$$\frac{S_{terrain}}{S_{lac}} = \frac{X}{X-N} \text{ et } S_{lac} = \frac{X}{X-N} \times S_{terrain}$$

La qualité de l'estimation dépend directement du nombre de tirs effectués et de la répartition des tirs. En effet, si le canonnier ne vise que le lac, l'estimation est faussée.
Cette caractéristique est à rapprocher de la qualité du générateur de nombre pseudo-aléatoire.

a. Généralisation

De manière générale, la méthode de Monte Carlo est un formidable outil d'approximation d'intégrale. Prenons le problème de l'intégration numérique :

$$\int_0^1 g(x)dx$$

Outre les diverses méthodes déterministes existantes (trapèzes, Simpson, etc.), la méthode de Monte Carlo consiste à écrire cette intégrale sous la forme $I = E[g(U)]$ avec U une variable aléatoire suivant une loi uniforme sur $[0,1]$.
Ainsi, d'après la loi forte des grands nombres, si $(U_i)_{i\in\mathbb{N}}$ est une suite de variables aléatoires indépendantes et de loi uniforme sur $[0,1]$ alors :

$$\frac{1}{n}\sum_{i=1}^{n} g(U_i) \to E[g(U)] \text{ « presque sûrement »}$$

En d'autres termes, si $u_1, u_2, \dots, u_n$ sont des nombres tirés au hasard dans $[0,1]$ alors $\frac{1}{n}\big(g(u_1) + g(u_2) + \cdots + g(u_n)\big)$ est une approximation de $\int_0^1 g(x)dx$.

Finalement, nous avons effectué la même opération que précédemment, nous avons créé un rectangle virtuel et produit une sélection aléatoire de u_i uniforme à l'intérieur du rectangle. De même, au plus il y aura d'u_i dans notre sélection au plus l'approximation sera juste.