

Sommaire.

1 Introduction

- Générateur Congruentiel Linéaire
- Suites de Fibonacci "retardées"
- Registres à décalage à rétroaction
- Cryptography Network Security

2 Initialisation d'un générateur

3 Mise en oeuvre


4 Tester un générateur

- Tests Mathématiques
- Sample moments
- Visual Analysis

5 Références

Simulation

- La simulation est l'imitation du fonctionnement d'un processus ou d'un système du monde réel dans le temps.
- Pour simuler quelque chose, il faut d'abord développer un modèle ; ce modèle représente les caractéristiques ou les comportements/fonctions clés du système ou du processus physique ou abstrait sélectionné.
- Le modèle représente le système lui-même, tandis que la simulation représente le fonctionnement du système dans le temps.



Les simulations informatiques utilisent un modèle mathématique du système réel. Dans un tel modèle, nous utilisons des variables pour représenter les mesures numériques clés des entrées et des sorties du système, et nous exprimons les relations mathématiques entre les entrées et les sorties.

Lorsque la simulation traite de l'incertitude, le modèle comprendra des variables incertaines, dont les valeurs ne sont pas sous notre contrôle, ainsi que des variables de décision ou des paramètres que nous pouvons contrôler.

Les variables incertaines sont représentées par des générateurs de nombres aléatoires qui renvoient des valeurs échantillons provenant d'une distribution représentative des valeurs possibles. Une simulation comprend plusieurs centaines ou milliers d'essais.

L'idée de hasard

Depuis un siècle, s'est développée une science de la production contrôlée de l'aléa.
En fait, trois sciences :

- 1 celle du hasard faible, utile pour les simulations et la modélisation informatique ;
- 2 celle du hasard moyen, qui convient en cryptographie ;
- 3 celle du hasard fort, née de la théorie mathématique de l'information et de la calculabilité.

Le hasard faible

- Créer un nuage de points destiné à colorier en gris une partie d'image (et plus généralement dans tout problème de texture).
- Simuler la circulation automobile ou les molécules d'un gaz dans un réservoir, etc.
- Modéliser des interactions entre particules élémentaires.

Nécessité de méthodes rapides produisant des millions de bits aléatoires,... même si ceux-ci ne le sont qu'assez superficiellement.

Les fonctions **random** des langages de programmation donnent ce genre de hasard peu coûteux en temps de calcul... mais dangereux. En cas d'expérimentations intensives, le risque de surprises désagréables est grand.

Sommaire.

- 1 Introduction
 - Générateur Congruentiel Linéaire
 - Suites de Fibonacci "retardées"
 - Registres à décalage à rétroaction
 - Cryptography Network Security
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre
- 4 Tester un générateur
- 5 Références

Générateur Congruentiel Linéaire : LCG

Rappels

1 La relation de congruence : \equiv .

Les deux nombres x_i et $a.x_{i-1}$ sont dits congrus modulo 2^{n-1} lorsque divisés par 2^{n-1} , ils fournissent le même reste.

$$x_i \equiv a.x_{i-1} \Rightarrow x_i = q 2^{n-1} + r \quad a.x_{i-1} = q' 2^{n-1} + r.$$

2 Le bit de poids fort, (**Most Significant Bit, ou MSB**) est le bit, dans une représentation binaire donnée, ayant la plus grande valeur (celui de gauche dans la représentation positionnelle habituelle).

Exemples :

Le nombre 9 s'écrit en binaire **1001**. Le MSB (en gras) contribue pour 8 unités à la valeur totale du nombre. Le nombre 25 s'écrit en binaire **11001**.

3 Le bit de poids faible (**Least Significant Bit, ou LSB**) est pour un nombre binaire le bit ayant dans une représentation donnée la moindre valeur (celui de droite dans la représentation positionnelle habituelle).

Générateur Congruentiel Linéaire : LCG

Le générateur de nombres pseudo-aléatoires PRNG (Pseudo Random Number Generator) le plus utilisé (D. H. Lehmer 1949) :

$$x_i = a.x_{i-1} \pmod{m}$$

- a : coefficient de multiplication,
- x_0 : valeur d'initialisation,
- m : modulo, fixe l'intervalle dans lequel des nombres entiers vont être engendrés.

Une variante de ce type de générateur peut se présenter sous la forme :

$$x_i \equiv (ax_{i-1} + c) \pmod{m}, \quad \text{with } 0 < x_i < m.$$

- c : l'incrément - constante additive,

Générateur Congruentiel Linéaire

Un tel générateur est noté :

$$LCG(m, a, c, x_0).$$

Propriétés :

- 1 Ce type de générateur génère une suite (pseudo-aléatoire) de nombres compris entre 0 et $m - 1$. m donne la **période** du générateur.
- 2 Si on désire produire toujours la même séquence (ce qui est pratique à des fins de tests), on rentre toujours la même valeur de x_0 (**la graine**) du générateur.
- 3 Si on préfère que la séquence soit toujours différente, on initialise x_0 avec une grandeur toujours différente, l'heure système par exemple.

Ce type de générateur est fréquemment rencontré car :

- 1 sa mise en oeuvre est très simple ;
- 2 il présente des propriétés statistiques satisfaisantes pour la plupart des applications classiques.

Exemples

Exemple 1

$$x_{n+1} = (3 * x_n + 1) \bmod(16) \quad LCG(16, 3, 1, x_0 = 2)$$

$$\left. \begin{array}{l} ax_0 + b = 7 \bmod(16) \quad \left(\begin{array}{l} \frac{7}{16} = 0 \quad r = 7 \end{array} \right) \\ ax_1 + b = 22 \bmod(16) \quad \left(\begin{array}{l} \frac{22}{16} = 1 \quad r = 6 \end{array} \right) \\ ax_2 + b = 19 \bmod(16) \quad \left(\begin{array}{l} \frac{19}{16} = 1 \quad r = 3 \end{array} \right) \\ \dots \end{array} \right\} \text{séquence : } 2, 7, 6, 3, 10, 15, 14, 11, 2, 7, 6, \dots$$

Exemple 2 : Soit a, c et m tels que :

$$x_{n+1} = (25 * x_n + 16) \bmod (256)$$

- Pour $x_0 = 12$, les nombres produits sont : 60, 236, 28, 204, 252, 172.... : tous les termes de la suite sont pairs.
- Pour $x_0 = 11$, les nombres produits sont : 35, 123, 19, 235, 3, 91, 243.. : tous les termes de la suite sont impairs.
- Pour $x_0 = 10$, les nombres produits sont : 10, 10, 10, 10....

La formule est simple mais le choix des trois paramètres ne doit pas être fait à la légère pour ne pas retrouver les défauts graves de l'exemple précédent.

Choix des paramètres

$$LCG(m, a, c, x_0).$$

Choix du module m

Le module m constituant une borne supérieure à la période, on a intérêt à le prendre le plus grand possible, par exemple on peut choisir le plus grand entier représenté sur la machine.

Pour une machine 32 bits, on peut prendre $m = 2^{31}$ (le 32^{ème} digit étant réservé au signe (fortran)).

On pourra prendre $m = 2^{32}$ (C) en travaillant avec des entiers positifs ("unsigned int").

Critères (D. Knuth) que doivent remplir a , c et m :

- ① c et m doivent être premiers entre eux,
- ② $a - 1$ doit être un multiple de p , pour tout p nombre premier diviseur de m ,
- ③ $a - 1$ doit être un multiple de 4 si m est un multiple de 4,
- ④ si m est une puissance de 2, le bit de poids faible des nombres produits vaut alternativement 0 et 1 (ce n'est d'ailleurs pas le seul cas où cela se produit).

Exemple 2 le premier critère n'est pas respecté ($m = 256, c = 16 \rightarrow m = 16b$).

Exemple 3 : Respecte scrupuleusement les critères (R. Sedgewick) :

$$x_{n+1} = (31415821 * x_n + 1) \bmod 10^8$$

- * 1 and 100 000 000 sont relativement premier (nombres entiers dont le plus grand diviseur commun est 1).
- * décomposition en nombres premiers de 100 000 000 : $2^6 \times 5^6$
- * 31 415 821 - 1 = 31 415 820 est divisible par tous les facteurs de 100 000 000 (2 and 5).
- * 31 415 820 est divisible par 4 (100 000 000 est divisible par 4).
- * Tous les nombres entre 0 et 99 999 999 vont sortir et chacun une fois tous les 100 000 000 de tirages.

Pour $x_0 = 0$, la suite de valeurs aléatoires générées :

1, 31415822, 40519863, 62952524, 25482205, 90965306, 70506227, 6817368, 12779129, 29199910, 45776111, 9252132, 22780373, 20481234, 81203115,

On note la progression du dernier numéro qui ne laisse rien au hasard. Etre sûr que tous les chiffres sortent ne garantit pas qu'ils seront suffisamment aléatoires.

Générateurs portables

Park & Miller (1988) (Standard Minimal)

$$x_{n+1} = 16807 * x_n \bmod (2^{31} - 1)$$

- Générateur standard convenablement testé
- Code est portable sur toutes les machines (drand48 en ANSI C)
- Cas où b est nul donc il faut absolument éviter de partir avec $x_0 = 0$.
- La période de ce générateur est $2^{31} - 2$, ce qui est suffisamment long pour la plupart des applications.
- On peut lui reprocher d'avoir un terme multiplicatif un peu petit, mais à part pour les cas pointus, il est généralement suffisant.
- Enfin, ce générateur a passé avec succès toute une batterie de tests depuis son invention par Lewis et al. en 1969 même s'il a quelques difficultés avec le test du Chi2.

- Générateur de Knuth & Lewis :

$$x_{n+1} = (1664525 * x_n + 1013904223) \mod 2^{32}.$$

- Générateur de Marsaglia

$$x_{n+1} = 69069 * x_n \mod 2^{32}.$$

- Générateur de Lavaux & Jenssens

$$x_{n+1} = (31167285 * x_n + 1) \mod 2^{48}.$$

- Générateur de Haynes :

$$x_{n+1} = (6364136223846793005 * x_n + 1) \mod 2^{64}.$$

Sommaire.

- 1 Introduction
 - Générateur Congruentiel Linéaire
 - Suites de Fibonacci "retardées"
 - Registres à décalage à rétroaction
 - Cryptography Network Security
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre
- 4 Tester un générateur
- 5 Références

Suites de Fibonacci "retardées"

Dans les congruences considérées précédemment, x_n ne dépend que du nombre précédent x_{n-1} .

On peut accroître la période en choisissant une relation où x_n dépend en plus d'une autre valeur précédente. La relation la plus simple, à cet effet, est :

$$x_{n+1} = (x_n + x_{n-1})[mod\ m].$$

En choisissant $x_0 = x_1 = 1$, la récurrence

$$x_{n+1} = (x_n + x_{n-1})[mod\ m]$$

définit la "suite de Fibonacci".

Remarque :

On sait que lorsque $n \rightarrow \infty$ le rapport x_{n+1}/x_n de deux termes successifs tend vers le nombre d'or. Elle ne peut conduire à un bon générateur de nombres aléatoires.

Suites de Fibonacci "retardées"

Par contre on peut généraliser :

$$x_{n+1} = (x_{n-l} + x_{n-k}) \quad [mod \ m]$$

Les nombres k et l sont appelés "**lags**" (i.e "retards" ou "décalages").

On montre que si $m = 2^\alpha$ est une puissance de deux, et si l et k sont tels que le trinôme : $x^k + x^l + 1$ est un polynôme premier (i.e. n'a aucun autre polynôme diviseur que lui-même ou une constante) dans le corps des entiers modulo 2, alors la période est $2^{\alpha-1}(2^k - 1)$ ($k > l$) et le nombre de cycles différents est $2^{(\alpha-1) \times (k-1)}$.

Les premières valeurs du couple (l, k) qui vérifient ces propriétés sont (24,55), (38,89), (37,100), (30,127), (83,258), etc.

Pour k et l suffisamment grands, tous les tests statistiques s'avèrent en général excellents, bien supérieurs aux résultats des congruences linéaires.

Générateur de Mitchell et Moore

Mitchel et Moore ont proposé le générateur suivant :

$$x_n = (x_{n-24} + x_{n-55}) \bmod m.$$

Si m est une puissance de 2, ce générateur ne nécessite qu'une addition. Il a en plus l'avantage d'avoir une période extrêmement longue : un multiple $2^{55} - 1$.

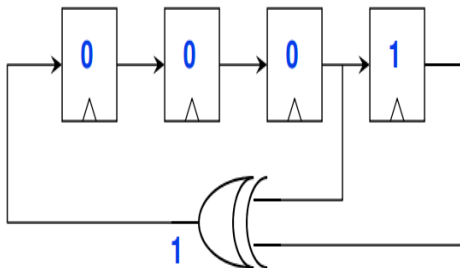
De ce fait, il n'est pas possible avec les moyens de calculs actuels de faire tourner le générateur de manière à voir une période complète.

Donc la vérification pratique que ce générateur ne se "coince" pas est impossible. Mais par bonheur, on peut le démontrer (l'étude théorique est particulièrement ardue). C'est pourquoi on dispose de peu de preuves théoriques de ses performances statistiques. Cependant, tous les tests effectués depuis son invention en 1958 ont toujours été concluants.

Sommaire.

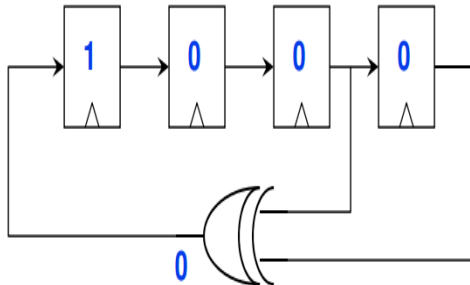
- 1 Introduction
 - Générateur Congruentiel Linéaire
 - Suites de Fibonacci "retardées"
 - Registres à décalage à rétroaction
 - Cryptography Network Security
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre
- 4 Tester un générateur
- 5 Références

Principe



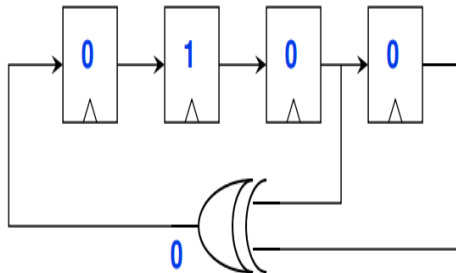
1

Principe



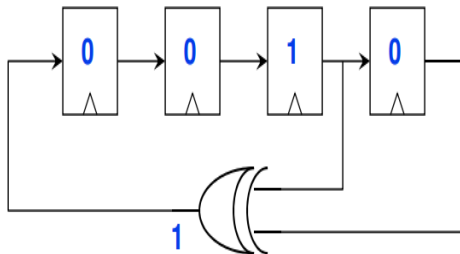
10

Principe



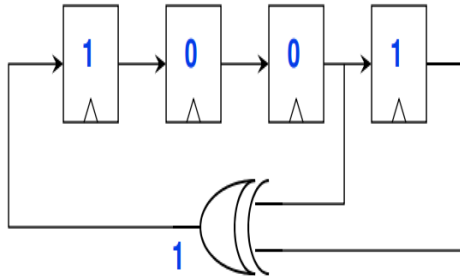
100

Principe



1001

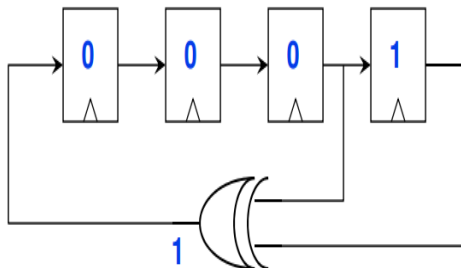
Principe



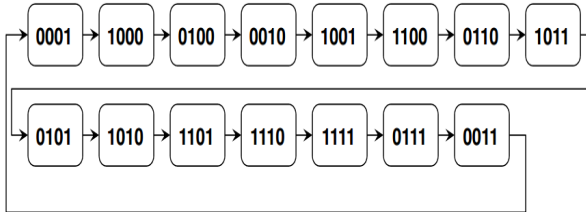
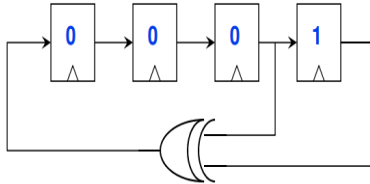
10011

Principe

Combien d'états peut-on avoir ?



Principe



15 states

Registres à décalage à rétroaction

Utilisation des registres à décalages à rétroaction appelés **Linear Feedback Shift Register (LSFR)**.

- On commence avec un registre rempli arbitrairement de 0 et de 1.
- On décale ensuite cette suite d'un cran vers la droite.
- On remplira la position tout à gauche par la somme (modulo 2) du contenu des deux registres les plus à droite (avant le décalage).



Chaque carré jaune représente un bit et le rond noir l'addition modulo 2

Exemple

Registre à 4 bits rempli pour commencer avec le motif 1111

Etape	0	1	2	3	4	5	6	7	8	9
Registre	1111	0111	0011	0001	1000	0100	0010	1001	1100	0110
Base dix	15	7	3	1	8	4	2	9	12	6

Etape	10	11	12	13	14	15
Registre	1011	0101	1010	1101	1110	1111
Base dix	11	5	10	13	14	15

- En convertissant les contenus successifs du registre en base 10, on obtient une suite pseudo-aléatoire.
- Le processus cycle quand on retrouve le motif initial.
- On a obtenu tous les motifs possibles, sauf 0000. Cela aurait été le cas pour tous les états initiaux possibles, sauf 0000.
- On a trouvé dans notre exemple un cycle de longueur maximale.

Remarque : on peut imaginer d'autres façons de remplir la position tout à gauche, par exemple, en additionnant modulo 2 les trois derniers bits, ou le dernier et l'avant dernier, etc.

Conclusions

- En général, pour un registre à n bits, il est possible d'arranger les choses pour que la longueur du cycle soit $2^n - 1$.
- Pour n grand, on obtient de bons générateurs de nombres pseudo-aléatoires.
- Typiquement, on peut utiliser $n = 31$ ou $n = 63$.
- Comme pour les générateurs congruentiels linéaires, les propriétés mathématiques de ces registres ont été beaucoup étudiées.
Par exemple, on sait beaucoup de choses sur le choix des positions à utiliser pour la rétroaction pour obtenir un cycle de longueur maximale.
Ainsi, pour $n = 31$, on peut faire la rétroaction avec la position 0 et au choix les positions 4, 7, 8, 14, 19, 25, 26 ou 29 (les positions sont numérotées de droite à gauche en commençant par 0).

Mersenne Twister

TGSFR (twisted generalised shift feedback register)

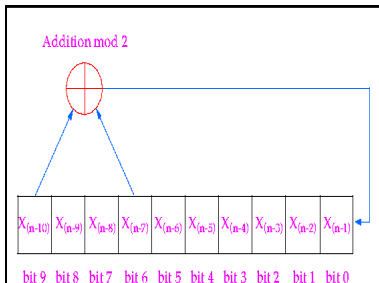


Figure 8: State Transition Diagram for Binary Shift Register

\oplus fonction OU exclusif, souvent appelée XOR (eXclusive OR) : opérateur logique qui deux opérandes, pouvant avoir chacun la valeur VRAI ou FAUX, associe un résultat qui a lui-même la valeur VRAI seulement si les deux opérandes ont des valeurs distinctes. Cet opérateur peut s'utiliser pour combiner deux bits, valant chacun 0 ou 1, en appliquant les règles la table suivante, le résultat étant lui-même la valeur d'un bit.

A	B	$R = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Exemple : 0010 (decimal 2) OR 1000 (decimal 8)
= 1010 (decimal 10))

Sommaire.

- 1 Introduction
 - Générateur Congruentiel Linéaire
 - Suites de Fibonacci "retardées"
 - Registres à décalage à rétroaction
 - Cryptography Network Security
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre
- 4 Tester un générateur
- 5 Références

Blum Blum Shub (BBS)

On rappelle qu'un résidu quadratique modulo M est le carré d'un nombre modulo M .

$$x_{n+1} = (x_n)^2 \bmod M$$

- La sortie de l'algorithme est le bit le moins significatif ou les derniers bits de x_{n+1}
- générer deux grands nombres premiers p and q
- $M = p.q$ produit de 2 nombres premiers p, q ; p, q devraient tous deux être congrus à 3 modulo 4
- La graine aléatoire x_0 et M doivent être premiers entre eux (c'est-à-dire que p et q ne doivent pas être des facteurs de x_0), et x_0 ne doit pas être 0 ou 1.

$$p = 7603, q = 7487, x_0 = 7817, M = 56923661$$

Sommaire.

- 1 Introduction
 - Générateur Congruentiel Linéaire
 - Suites de Fibonacci "retardées"
 - Registres à décalage à rétroaction
 - Cryptography Network Security
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre
- 4 Tester un générateur
 - Tests Mathématiques
 - Sample moments
 - Visual Analysis
- 5 Références

Choix de la graine

- Une graine aléatoire (aussi appelée germe aléatoire) est un nombre utilisé pour l'initialisation d'un générateur de nombres pseudo-aléatoires
- Il n'est pas nécessaire que la graine soit aléatoire
- Deux graines différentes produiront des suites de nombres aléatoires complètement différentes..

Exemple en C : `function srand - void srand(unsigned int seed);`

Exemple en Python : `random.seed ()`

Exemple en R : `set.seed()`

Initialisation d'un générateur en C

Exemple : le C 2 fonctions : **srand()** et **randomize()**.

srand(unsigned)

- Cette fonction initialise le générateur avec une valeur qui lui est fournie de manière à pouvoir générer toujours la même séquence (pratique pour tester une routine).
- Le défaut est écrit dans le prototype même de la fonction : on doit lui passer un "unsigned int".
- Dans de nombreuses implémentations du C, ce type correspond à un nombre de 16 bits. Cela signifie, qu'on ne peut produire que 65536 (2^{16}) séquences différentes possibles, et c'est fort peu
- On serait par exemple bien mal inspiré d'utiliser **srand()** pour initialiser une séquence utilisée en cryptage, car même si la période du générateur est très longue, il est très facile de passer en revue toutes les possibilités....
- C'est d'autant plus regrettable qu'il n'y a pas spécialement de problème à initialiser le générateur au moins avec un long (sur 32 bits) dans le cas du C ANSI car c'est avec cette taille de mots qu'il travaille.

Initialisation d'un générateur en C

Le plus couramment **srand** s'initialise en utilisant la fonction `int time(int*)` qui est définie dans `time.h`.

Cette fonction donne le nombre de secondes écoulées depuis le premier janvier 1970 (il est très rare qu'on lance le programme deux fois dans la même seconde).

On ne se servira pas du paramètre que l'on mettra donc à `NULL`. On l'appellera donc comme ceci : `time(NULL)` .

```
#include < stdio.h >
#include < stdlib.h >
#include < time.h > //Ne pas oublier d'inclure le fichier time.h
int main(void){
    int i = 0;
    int nombre_aleatoire = 0;
    srand(time(NULL)); // initialisation de rand
    for(i=0; i < 5; i++){
        nombre_aleatoire = rand();
        printf("%d ",nombre_aleatoire);
    }
    return 0;
}
```

Initialisation d'un générateur en Python

```
import random  
random.seed ( [x] )
```

Note : les graines ne sont pas directement accessibles, vous devez importer le module aléatoire, puis appeler la méthode par des objets statiques aléatoires.

```
#!/usr/bin/python
```

```
import random  
random.seed( 10 ) _ print "Random number with seed 10 : ", random.random()  
random.seed( 10 )  
print "Random number with seed 10 : ", random.random() _
```

Après avoir exécuté l'exemple ci-dessus on obtient

Random number with seed 10 : 0.57140259469

Random number with seed 10 : 0.57140259469

Sommaire.

- 1 Introduction
 - Générateur Congruentiel Linéaire
 - Suites de Fibonacci "retardées"
 - Registres à décalage à rétroaction
 - Cryptography Network Security
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre**
- 4 Tester un générateur
 - Tests Mathématiques
 - Sample moments
 - Visual Analysis
- 5 Références

Produire un nombre entier entre deux bornes

Les algorithmes qu'on a vus produisent des nombres pseudo-aléatoires entre deux bornes très éloignées (pour le Standard Minimal, l'intervalle est $[1, 2^{31} - 2]$).

Dans la pratique, on a plutôt besoin d'un intervalle réduit.

Comme généralement les bits de poids faibles des générateurs ne sont pas particulièrement aléatoires, la ligne C suivante effectuant un tirage entre 0 et n est vivement déconseillée :

`j = rand() % n ;`

Si n est grand devant la valeur maximale du générateur, elle peut en plus introduire un biais considérable.

Imaginons par exemple que notre générateur de base produise des nombres avec une distribution uniforme entre 0 et $2^{32} - 1$ inclus et qu'on demande un $n = 2^{32} - 2$.

- si le nombre produit par rand() est égal à 0 ou $2^{32} - 2$, au résultat on aura $j = 0$,
- si le nombre produit par rand() est égal à 1 ou $2^{32} - 1$, au résultat on aura $j = 1$,
- dans tous les autres cas, $j = \text{rand}()$.

Conclusion, 0 et 1 ont deux fois plus de chances de sortir que les autres nombres. C'est dommage, même si cet exemple est un cas extrême malgré tout assez peu gênant.

Il est donc préférable de procéder comme suit :

```
j = (int)(n*rand()/(RAND_MAX + 1));
```

- (RAND_MAX est la valeur maximale que peut produire le générateur rand(), la valeur minimale étant 0.)
- Si RAND_MAX + 1 est une puissance de 2, il est possible de réduire le calcul à des multiplications entières.
- Il est possible de faire la même chose sans multiplication ni division et en restant avec des entiers :

Sommaire.

- 1 Introduction
 - Générateur Congruentiel Linéaire
 - Suites de Fibonacci "retardées"
 - Registres à décalage à rétroaction
 - Cryptography Network Security
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre
- 4 **Tester un générateur**
 - Tests Mathématiques
 - Sample moments
 - Visual Analysis
- 5 Références

Tester un générateur

Les nombres sont-ils réellement aléatoires ?

Il est impossible de donner une preuve mathématique que le générateur est un aléatoire ;
Les tests aident à détecter certaines formes de faiblesse que les générateurs peuvent avoir

Les propriétés que doivent valider les générateurs sont **uniformité** and **indépendance**

● **Uniformité**

- 1 Frequency Test
 - Kolmogorov-Smirnov test
 - Chi-Square test
- 2 Sample moments

● **Indépendance** : Five Basic Test (Chi-square analysis)

- 1 Run tests :
- 2 Autocorrelation test
- 3 Serial Test :
- 4 Poker Test :

Caractéristiques

Equiprobabilité :

Une caractéristique de la propriété *est aléatoire* est l'équiprobabilité, c'est-à-dire que chaque valeur de la suite apparaît avec la même fréquence (dans notre cas 0 et 1 apparaissent avec une fréquence 1/2). Un premier test consiste donc à vérifier ce point.

Cependant conclure que la suite est aléatoire si elle passe ce test avec succès est absurde : en effet une suite du type 0101010 etc. passe ce test sans problème, mais ne peut en aucun cas être considéré comme aléatoire. Il est donc nécessaire de prévoir plusieurs tests complémentaires (i.e avec les mêmes hypothèses H_0 et H_1 mais avec des objectifs différents, par exemple le calcul de la fréquence et la recherche de chaînes périodiques dans la suite).

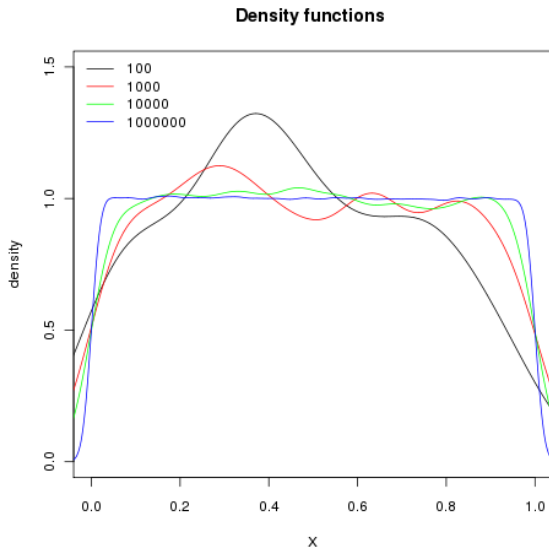
Une suite aléatoire doit bien sûr avoir quelques caractéristiques mathématiques fondamentales :

- 1 un histogramme de population de chaque valeur plat,
- 2 une corrélation nulle,
- 3 un bit d'entropie par bit de sortie.

Sommaire.

- 1 Introduction
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre
- 4 Tester un générateur
 - Tests Mathématiques
 - Sample moments
 - Visual Analysis
- 5 Références

Distributions aléatoires

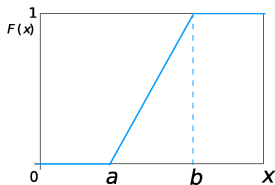
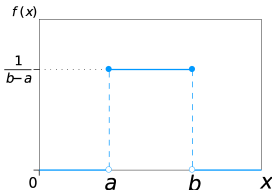


Sommaire.

- 1 Introduction
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre
- 4 Tester un générateur
 - Tests Mathématiques
 - Sample moments
 - Visual Analysis
- 5 Références

Equiprobabilité

- Uniform distribution (continuous) $\mathcal{U}([a, b])$



- Uniform distribution (continuous) $\mathcal{U}([0, 1])$

Quartiles		Moments	
1 st quartile	1/4	Variance	1/12 \approx 0.0833333
Expected value (median)	1/2	Kurtosis (normalised)	-6/5
3 rd quartile	3/4	Skewness	0

- Exemples

n = 1 000 000	Park Miller	Mitchell Moore	Mersenne Twister	True Random
Quartile 25%	0.2501178	0.2502911	0.2505255	0.2491899
Mean value	0.50003 0	0.5004514	0.5001666	0.4995804
Quartile 75%	0.7498376	0.7507262	0.7497352	0.7496738
Variance	0.08324767	0.0834187	0.08324623	0.08346218
Excess kurtosis	-1.198939	-1.200936	-1.198613	-1.200789
Skewness	-0.0005702	-0.002919674	-0.0001407255	0.002259139

Tests quantitatifs usuels (1)

Voir : Exploratory Data Analysis, NIST/SEMATECH e-Handbook of Statistical Methods,
<http://www.itl.nist.gov/div898/handbook/>

- Coefficient de corrélation de Pearson - Mean Squared Error

n = 1 000 000	Park Miller	Mitchell Moore	Mersenne Twister	True Random
Correlation	0.00286138	-0.00145309	-0.00030804	0.00266613
MSE	4.155 e-05	4.683 e-05	2.485 e-05	4.352 e-05

- Chi-Square Test

$$\chi^2 = \sum_{i,j} \frac{(f_{ij} - e_{ij})^2}{e_{ij}} \quad f_{ij} : \text{empirical values} ; e_{ij} : \text{theoretical values}$$

n = 1 000 000	Park Miller	Mitchell Moore	Mersenne Twister	True Random
X-squared	149840	1500400	1498300	1502800
p-value	< 2.2e - 16	< 2.2e - 16	< 2.2e - 16	< 2.2e - 16

Tests quantitatifs usuels (2)

- Autocorrelation tests

One way to look for short-range correlations in a sequence of random numbers x_i is to use the autocorrelation function :

$$C(k) = \frac{\langle x_{i+k} x_i \rangle - \langle x_i \rangle^2}{\langle x_i x_i \rangle - \langle x_i \rangle^2}$$

where $\langle x_{i+k} x_i \rangle$ is found by forming all possible products $x_{i+k} x_i$ for a given k and dividing by the number of terms in the product. $C(k)$ should become zero when $k \rightarrow \infty$

- diehard test : Birthday spacings

Choose random points on a large interval. The spacings between the points should be asymptotically exponentially distributed. Choose m birthdays in a year of n days. List the spacings between the birthdays. If j is the number of values that occur more than once in that list, then j is asymptotically Poisson distributed. Here $n = 2^{32}$ and $m = 2^{12}$, so that the underlying distribution for j is taken to be Poisson with $\lambda = m^2/(4n) = 4$ and with 245 samples

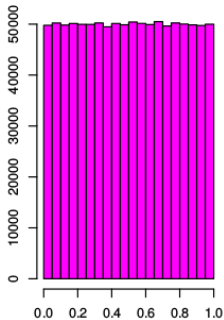
n = 1 000 000	Park Miller	Mitchell Moore	Mersenne Twister	True Random
$\lambda (= 4)$	5.734694	5.795918	4.004082	7.408163
MSE	8.167347	7.697959	3.857143	19.220408

Interprétation graphique - Histogramme

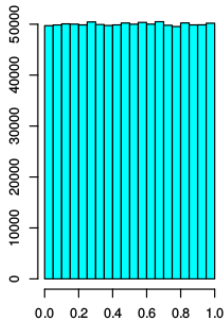
The uniform distribution is shaped like a rectangle, where each score is equally likely.

$n = 1\,000\,000$

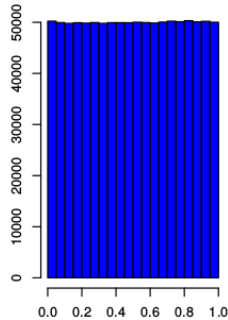
ParkMiller



MersenneTwister



MitchellMoore

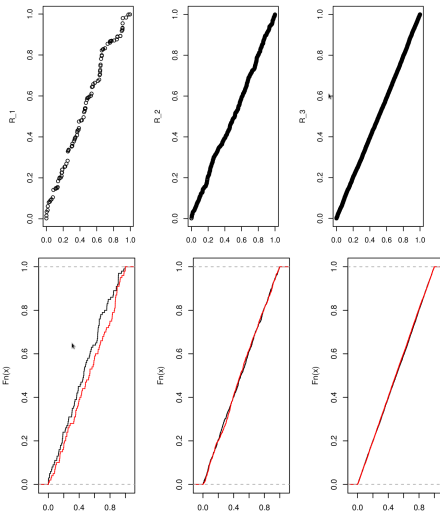


QQplot - ECDF

QQplot

$N_1 = 100$, $N_2 = 1000$, $N_3 = 10000$

Empirical Cumulative Distribution Function (ECDF)



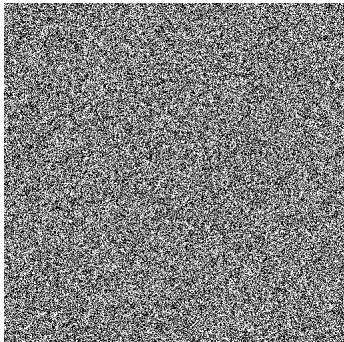
Sommaire.

- 1 Introduction
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre
- 4 Tester un générateur
 - Tests Mathématiques
 - Sample moments
 - Visual Analysis
- 5 Références

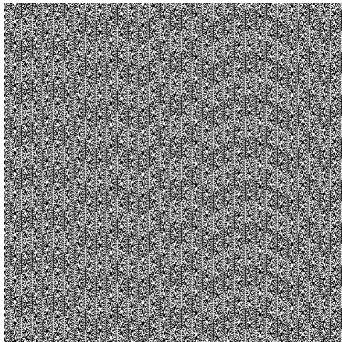
Visual Analysis

The reason : provide a very quick mechanism to test random sequences.

Bitmap generated with a True Random Number Generator (TRNG)



Bitmap generated with the rand() function from PHP on Microsoft Windows (PRNG)

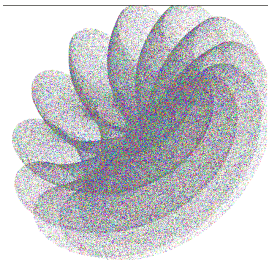


Visual Analysis

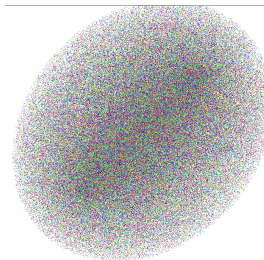
- Method which will be used to plot a sequence of numbers using spherical coordinate.
- Highlights problems of independence of a pseudo-random number sequence.
- If $s[n]$ is a set of random numbers and n represents the n^{th} random number in s , then the transformation of a triplet of points $s[n-2]$, $s[n-1]$, $s[n]$ to 3-space is as follows :

$$\theta[n] = 2\pi \times s[n-2] \quad \varphi[n] = \pi \times s[n-1] \quad r[n] = \sqrt{s[n]}$$

Randu Generator



Mersenne-Twister Generator



Conclusion



Sommaire.

- 1 Introduction
 - Générateur Congruentiel Linéaire
 - Suites de Fibonacci "retardées"
 - Registres à décalage à rétroaction
 - Cryptography Network Security
- 2 Initialisation d'un générateur
- 3 Mise en oeuvre
- 4 Tester un générateur
 - Tests Mathématiques
 - Sample moments
 - Visual Analysis
- 5 **Références**

- Knuth.D,E. *The Art of Computer Programming.Seminumerical Algorithms*.Vol.2.p.1-177. Addison-Wesley(1981)
- Park, S. K., Miller, K. W., *Random Number Generators : Good Ones Are Hard to Find*, Communications of the ACM, October 1988, Volume 31, Number10
- Marsaglia, G., [*Seeds for Random Number Generators, Communications of the ACM, Vol. 46, No 5, May 2003*
- Matsumoto M. , Nishimura T., *Mersenne Twister : [A 623- Dimensionally Equidistributed Uniform Pseudo-Random Number Generator January 1998, Keio Universit, ACM Transactions on Modelling and Computer Simulation, Volume 8, Number 1, Pages 3-30*
- Robert G. Brown *dieharder Duke University Physics Department Durham, NC 27708-0305 Copyright Robert G. Brown, 2016*

Les tests de compression

Ces tests statistiques déterminent si une suite peut être compressée de façon significative sans perte d'information, ce qui la distinguerait d'une suite aléatoire. Des programmes comme **ENT** calculent les caractéristiques mathématiques d'un échantillon produit par le générateur à tester. C'est le plus représentatif de ce genre de programme.

- Calcul de l'entropie, (taux de compression de la suite) :

Entropy = 7.999583 bits per byte.

Optimum compression would reduce the size of this 417792 byte file by 0 percent.

Une suite aléatoire n'est jamais compressible car son entropie est très proche de 8 bits par octet.

- Calcul du χ^2 (valeur très représentative de la qualité aléatoire)

En pratique, il faut être très proche de 50 %.

Chi square distribution for 417792 samples is 241.44, and randomly

would exceed this value 50.00 percent of the times.

- Calcul de la moyenne et le coefficient de corrélation :
Arithmetic mean value of data bytes is 127.5092 (127.5 = random).
La moyenne doit bien sr être très proche de 127.5.
- Calcul de π par Monte Carlo :
On utilise également chaque séquence de 6 octets pour calculer π via le Monte-Carlo. Monte Carlo value for Pi is 3.139361213 (error 0.07 percent).
Le pourcentage d'erreur doit être le plus petit possible mais peut être faussé si on utilise un petit échantillon.
- Le coefficient de corrélation :
Serial correlation coefficient is -0.000324 (totally uncorrelated = 0.0).
mesure la dépendance d'un octet à l'autre. Un flot aléatoire est proche de 0, un texte en anglais de 0.5 et des données très redondantes comme un fichier image bitmap s'approchent de 1.

Parmi les tests de compression, les plus connus sont le test universel de Maurer, le test de compression Lempel-Ziv, le test de l'entropie de Pincus, Singer

Les tests de Diehards

disponibles en fortran et C

Il s'agit de tests basés sur des manipulations de bits ou de matrices obtenues avec l'échantillon. Pour plus de détails, la sortie du programme (voir exemple ci-dessous) précise mathématiquement chaque test avant de donner les résultats. Ces tests retournent une valeur de χ^2 qui ne doit pas être 0 ou 1, ou trop proche de ces valeurs.

Le test est considéré comme réussi si le χ^2 est compris entre 0,025 et 0,975.

Ils permettent uniquement de s'assurer qu'il s'agit de bon générateurs de hasard faible, car ils ne peuvent tester si la séquence est réellement imprédictible. La version la plus récente du programme, Dieharder, propose 115 tests. à l'issus de chaque tests, trois résultats sont possibles : PASSED, si la p-value est très supérieur de 0,05% et très inférieur à 95%. WEAK si la p-value est proche de 0,05% ou 95%, FAILED sinon. Bien sr il faut garder à l'esprit que ces nombres ont été générés aléatoirement et qu'il existe donc une probabilité pour que ces tests soient rejetés à tord. Ainsi les WEAK sont en partie dus à un simple "manque de chance".

Ce test est néanmoins très gourmand, si de nos jours les machines ne mettent que moins de quelques minutes à effectuer les tests, il faut néanmoins un échantillon d'environ 10Mo pour parvenir à effectuer tous les tests.