Travail d'Etude et de Recherche

Sujet : Générations des nombres pseudo-aléatoires

Par :

QEHOUI Hajare

MERAKCHI Younes Ayoub

Élèves en Master 1 Informatique : Secrets

Encadré par : M.Rodolfo Canto Torres

Mai 2019

REMERCIEMENTS

Nous souhaitant adresser nos remerciements les plus sincères aux personnes qui nous ont apporté leur aide et qui ont contribué à l'élaboration de ce rapport ainsi quà la réussite de cette formidable année universitaire.
Ces remerciements vont tout dabord au corps professoral et administratif de la Faculté d'iformatiques , pour la richesse et la qualité de leur enseignement et qui déploient de grands efforts pour assurer à leurs étudiants une formation actualisée.

Nous tenant à remercier sincèrement Monsieur,Rodolfo Canto Torres, qui, en tant que encadreure, s'est toujours montrés à l'écoute et très disponible tout au long de la réalisation de ce rapport, ainsi pour l'inspiration, l'aide et le temps qu'il as bien voulu nous consacrer et sans qui ce travail n'aurait jamais vu le jour.

On n'oublie pas nos parents pour leur contribution, leur soutien et leur patience.

Enfin, nous adressons nos plus sincères remerciements à tous nos proches et amis, qui nous ont toujours encouragée au cours de la réalisation de ce rapport .

Merci à tous et à toutes.

Table des matières

Table des figures

Introduction

Un générateur de nombres pseudo-aléatoires, pseudorandom number generator (PRNG) en anglais, est un algorithme qui génère une séquence de nombres présentant certaines propriétés du hasard. Par exemple, les nombres sont supposés être suffisamment indépendants les uns des autres, et il est potentiellement difficile de repérer des groupes de nombres qui suivent une certaine règle (comportements de groupe). Cependant, les sorties d'un tel générateur ne sont pas entièrement aléatoires ; elles s'approchent seulement des propriétés idéales des sources complètement aléatoires.

En mai 2008, une faille de sécurité touchant les Linux Debian concernant l'implémentation d'OpenSSL etait apparue . En défaut un morceau de code simplifié à tort qui, au lieu de générer des nombres pseudo-aléatoires de grande qualité, a généré des nombres pseudo-aléatoires de piètre qualité, les rendant prévisibles. Il en a résulté une faille de sécurité, puisque toutes les clés créées sur un système Linux Debian et les systèmes d'exploitation dérivés (dont Linux Ubuntu) ont dû être re-générés. Or, ces clés servent notamment à la sécurité des échanges commerciaux sur Internet, lorsque l'on effectue des achats en ligne, généralement lors de la saisie du code de la carte bancaire, afin d'empêcher un éventuel espion écoutant la communication de la comprendre. Cette faille pouvait autant permettre à un tiers de se faire passer pour votre banque avec un certificat usurpé, ou bien de déchiffrer vos communications cryptées avec celle-ci, et ceci s'appliquait à l'essentiel des échanges cryptés véhiculant sur Internet. et tous ça c'est à cause de générateurs pseudo-aleàtoires.

Plusieurs cryptosystémes utilisent dans ces calcules des nombres pseudo-aléatoires pour garantir des propriétes de sécurité en face de differents type de situations et attaques. Dans ce projet, on propose une introduction dans leur génération, propiétes , étude de leurs périodes et leurs implémentations.

1 Générateurs congruentiels linéaires

1.1 Définition :

Il existe de nombreuses familles de RNG : congruence linéaire, récursives multiples et des algorithmes de « fonctionnement sur ordinateur». Un générateur congruentiel linéaire est un générateur de nombres pseudo-aléatoires dont l'algorithme, introduit en 1948 par Derrick Lehmer, sous une forme réduite, pour produire des nombres aléatoires, est basé sur des congruences et une fonction de transfert du type suivant :

$$f(x) = (ax + c) \bmod m \tag{1}$$

où a est le multiplicateur, c l'incrément et m le module et x, a, c, m∈N (i.e.N est l'ensemble des entiers (positifs)) .Tel que :

$$x_n = (ax_{\mathrm{n-1}} + c) \bmod m \tag{2}$$

Le terme initial X_0 est appelé la graine (seed en anglais). C'est elle qui va permettre de générer une suite apparemment aléatoire. Pour chaque graine, on aura une nouvelle suite. Cependant, il est possible que certaines graines permettent d'obtenir une suite plus aléatoire que d'autres.

Du fait de l'opération mod (reste dans la division euclidienne de $(aX_n + c)$ par m), les termes de cette suite sont compris entre 0 et $(m - 1)$.

De plus, comme chaque terme dépend entièrement du précédent, si un nombre apparaît une deuxième fois, toute la suite se reproduit à partir de ce nombre. Or le nombre de valeurs que le nombre peut prendre étant fini (égal à m), la suite est amenée à se répéter au bout d'un certain temps. On dit qu'elle est ultimement périodique.

1.2 Propriétés :

Dans ce type d'algorithme, la qualité du générateur va entièrement dépendre du choix de a, c et m car on ne peut pas se satisfaire d'un générateur qui produit des suites plus ou moins aléatoires, sans aucune raison apparente, selon le choix de X_0 .

Pour maximiser la longueur de cycle des générateurs congruentiels linéaires il faut respecter les trois conditions :

1. l'incrément c est relativement premier pour m .
2. $(a - 1)$ est un multiple de chaque nombre premier divisant m .
3. $(a - 1)$ est un multiple de 4 lorsque m est un multiple de 4 .

1.3 Cas spéciaux :

— Lorsque c = 0, ce générateur s'appelle un générateur de congruence linéaire multiplicatif (MLCG). La relation de récurrence est donnée par :

$$x_n = (ax_{n-1}) \bmod m \tag{3}$$

La période maximale pour un MLCG est m-1, puisque 0 est un état absorbant.Vous pouvez avancer de x_n à $x_n + v$ avec la relation :

$$x_{n+v} = a^v x_n \bmod m = (a^v \bmod m) x_n \bmod m \tag{4}$$

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

1.4 Les Avantages et les inconvenients :

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

1.5 Étude de periode :

1.5.1 générateur de congruence linéaire multiplicatif (MLCG) :

Premier exemple :

Pour :

a = 2
m = 277
$x_0 = 15$

x(0)=15 ;x(1)=30 ;x(2)=60 ;x(3)=120 ;x(4)=240 ;x(5)=203 ;x(6)=129 ;x(7)=258 ;x(8)=239 ;
x(9)=201 ;x(10)=125 ;x(11)=250 ;x(12)=223 ;x(13)=169 ;x(14)=61 ;x(15)=122 ;x(16)=244 ;x(17)=211 ;
x(18)=145 ;x(19)=13 ;x(20)=26 ;x(21)=52 ;x(22)=104 ;x(23)=208 ;x(24)=139 ;x(25)=1 ;x(26)=2 ;
x(27)=4 ;x(28)=8 ;x(29)=16 ;x(30)=32 ;x(31)=64 ;x(32)=128 ;x(33)=256 ;
x(34)=235 ;.....x(86)=195 ;x(87)=113 ;x(88)=226 ;x(89)=175 ;x(90)=73 ;x(91)=146 ;x(92)=15 ;

La période est : 92

Le résultat de l'execution

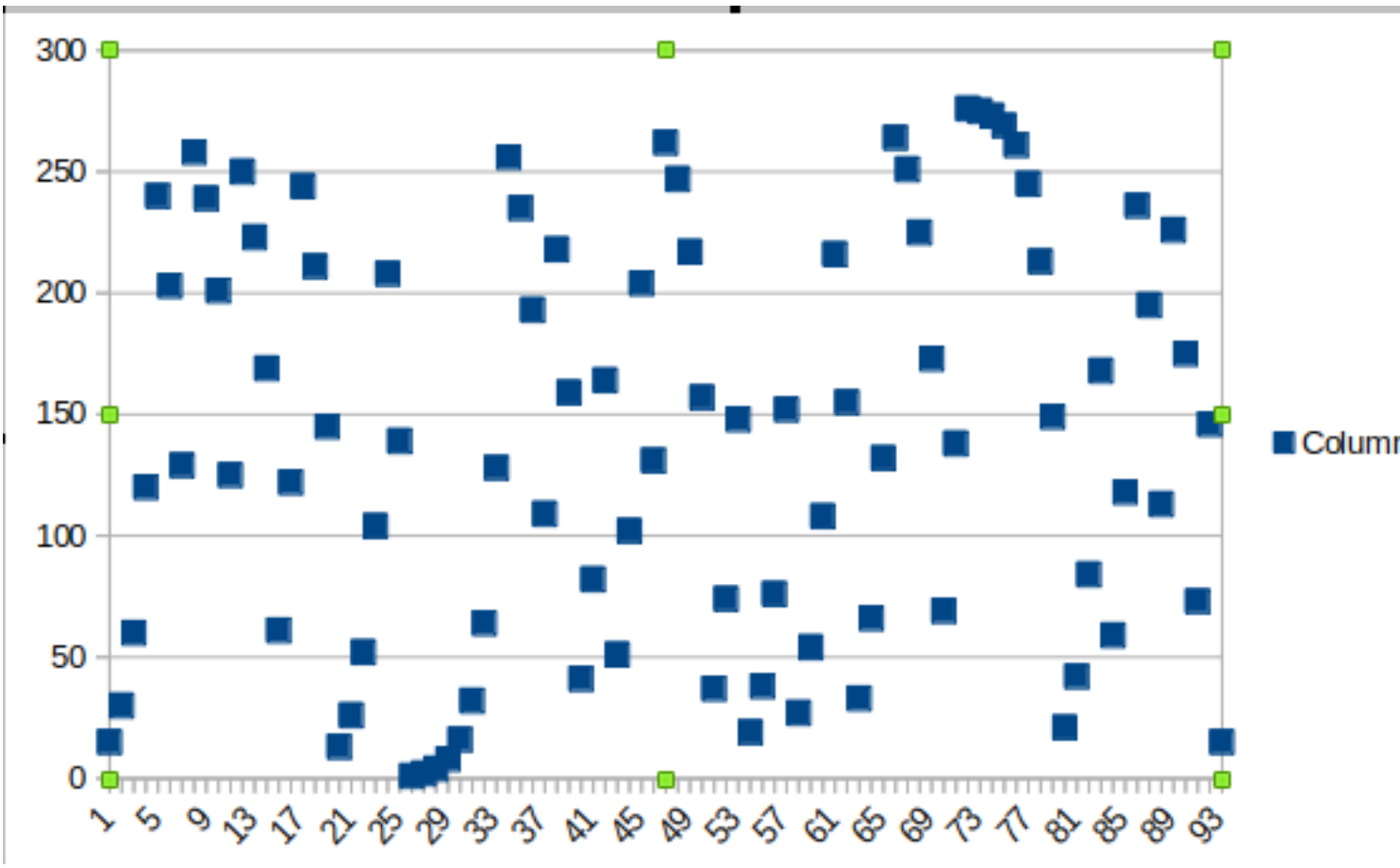

FIGURE 1 – La courbe d'un exemple de Générateurs congruentiels linéaires (d'une période)

Deuxième exemple :

Pour :

$m = 2^{12} = 4096$

$x_0 = 15$

a = 1025 a = 14 a = 486 a = 2048 a = 17 a = 10

le resultat de l'execution donne :

5	15	15	15	15	15
3087	210	3194	2048	255	150
2063	2940	3996	0	239	1500
1039	200	552	0	4063	2712
5	2800	2032	0	3535	2544
3087	2336	416	0	2751	864
2063	4032	1472	0	1711	448
1039	3200	2688	0	415	384
5	3840	3840	0	2959	3840
3087	512	2560	0	1151	1536
2063	3072	3072	0	3183	3072

1039	2048	2048	0	863	2048
5	0	0	0	2383	0
3087	0	0	0	3647	0
2063	0	0	0	559	0
1039	0	0	0	1311	0
...	...	...	...	...	...
La période :					
4	inf	inf	inf	256	inf

le graphe de ces 6 test :

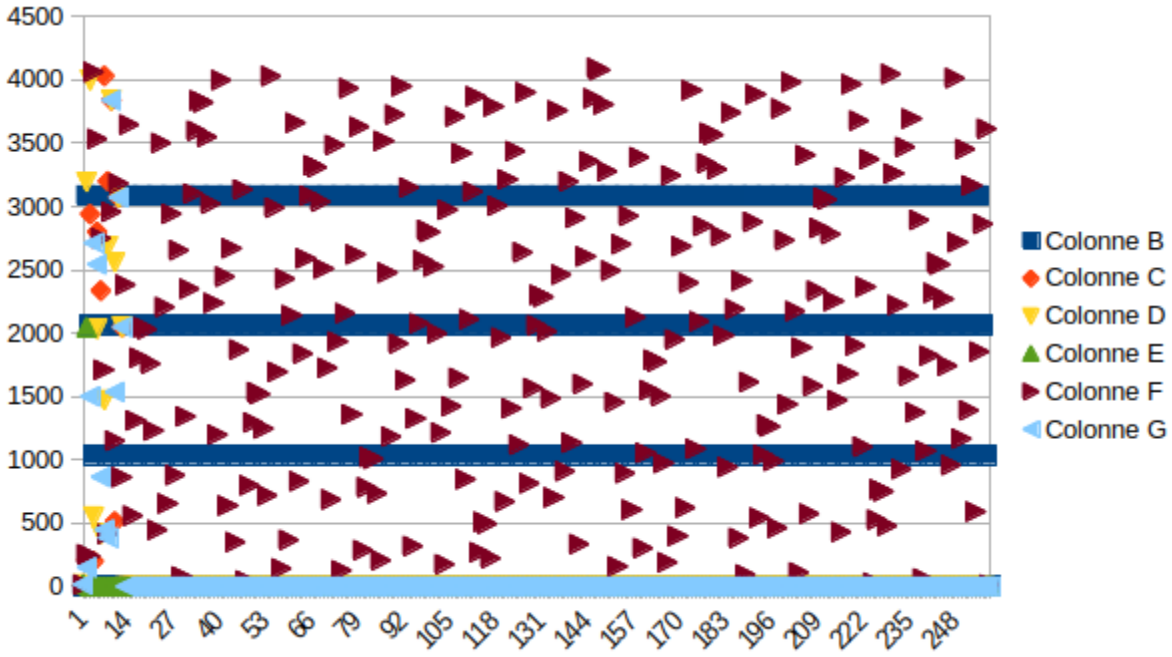

FIGURE 2 – La courbe d'un exemple de Générateurs congruentiels linéaires

-la colonne B désigne a= 1025
-la colonne C désigne a= 14
-la colonne D désigne a= 486
-la colonne E désigne a= 2048
-la colonne F désigne a= 17
-la colonne G désigne a= 10

On remarque si on doit choisire une valeur pour a parmit ces 6 valeur il faut bien choisir a=17 (puisque c'est la seule qui donne des valeur sur tout l'intervale).

1.5.2 générateur de congruence linéaire (LCG) :

Premier example :

Choisir les valeurs a, c et m « au hasard » n'est pas une bonne idée. Prenons un exemple, soit le générateur congruentiel linéaire employant les valeurs :

a = 25
c = 16
m = 256
$x_0 = 10$

l'éxécution du programme nous donne :

```
x( 0 ) = 10
x( 1 ) = 10
la periode est : 1
```

Donc on aura la suite : 10, 10, 10, 10, ...

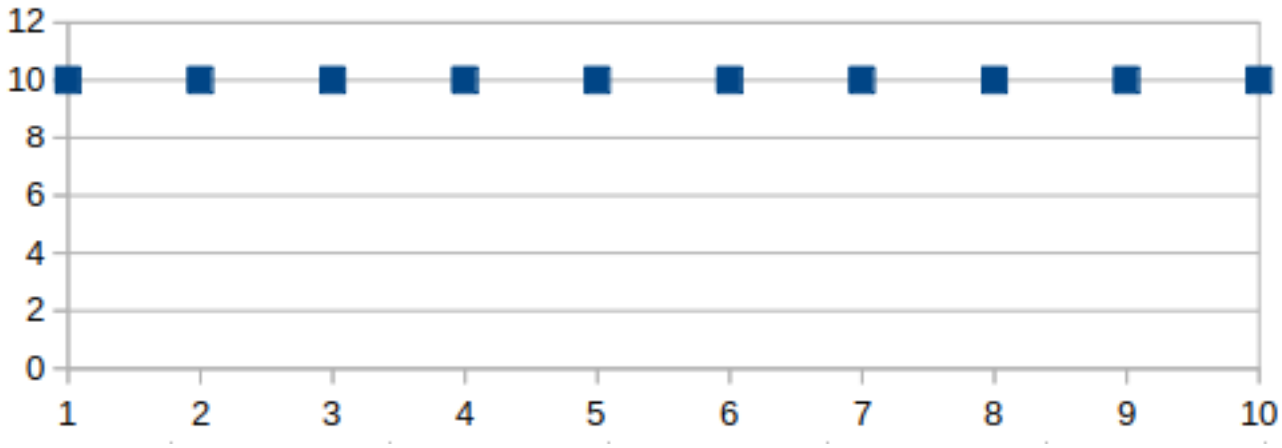

FIGURE 3 – Le graphe dun exemple de Générateurs congruentiels linéaires (Mauvais Coefficients) -plusieur périodes

Deuxième exemple :

avec $x_0 = 50$ l'éxecution du programme nous donne :

```
x( 0 )=50
x( 1 )=242
x( 2 )=178
x( 3 )=114
x( 4 )=50
la période est : 4
```

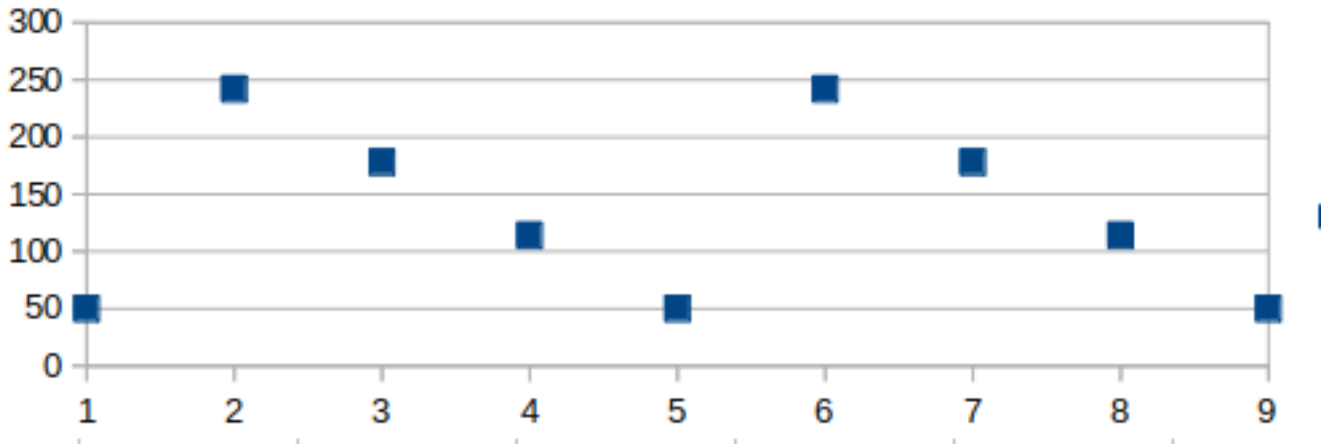

FIGURE 4 – Le graphe dun exemple de Générateurs congruentiels linéaires (2 période)

Il est clair que ces suites ne peuvent être considérées comme aléatoires.
On voit donc bien que l'on doit choisir avec précaution les paramètres du générateur si l'on espère obtenir des nombres qui s'approchent de l'aléa parfait.

Troisième exemple :générateur de Robert Sedgewick

pour :

- a = 31415821
- c = 1
- m = 10^8
- $x_0 = 10$

le résultat est incroyable :
Puisque le programme ecrit les resultats dans un fichier le fichier avait une taille très très grande ce qui a saturer la mémoire de l'ordinateur.
une nouvelle approche est suivi on va recupéré les donnés dans un graphe directement.

Une partie de résultat :
17-34068958-86184519-21875100-25957101-38694922-43160963-87795624-8167305-91932406-10995327-24868468-39232229-83695010-52753211-33951232-27241473-39544334-18508215-69469516-79612637-53329978-42781939-37656920-58131321-75029542-61183983-57995044-21191125-89788626-.....

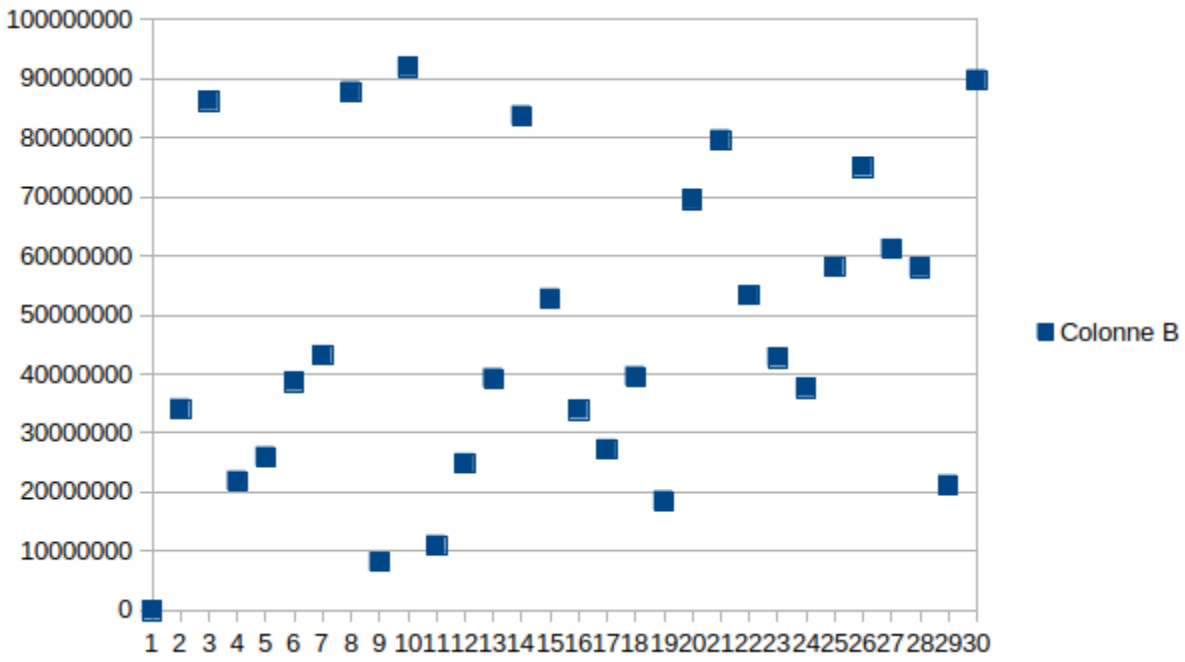

FIGURE 5 – Le graphe dun exemple de Générateurs congruentiels linéaires (partie d'une période)

temps pour trouver la période :

```
donner un x( 0 ) = 17
temps = 7.762638
la period est : 100000000
```

Dautres générateurs congruentiels linéaires quon cite dans le tableau suivant :

Algorithme	A	C	M
RANDU	$65539(2^{16}+3)$	0	2^{31}
Générateur du Turbo Pascal	129	907633385	2^{32}
Standard minimal*	16807	0	$2^{31}-1$
Générateur de Marsaglia*	69069	0	2^{32}
Générateur de KnuthETLewis *	1664525	1013904223	2^{32}

Tableau d'exemples des générateurs congruentiels linéaires

* Générateur de bonne qualité

2 Générateurs récursifs multiples

2.1 Définition :

Une généralisation des générateurs congruentiels linéaires est constituée de plusieurs générateurs récursifs. Ils sont basés sur les récurrences suivantes :

$$x_n = (a_1 \mathrm{x_{n-1}} + ... + a_k \mathrm{x_{n-k}} c) \bmod m, u_n = x_n/m. \tag{5}$$

où k est un entier fixe. L'ordre k et le module m sont des entiers positifs, alors que les coefficients $a_1, ..., a_k$ sont dans $\{-(m-1), ..., m-1\}$. Par conséquent, le nième terme de la séquence dépend du k précédent.

2.2 Cas spéciaux :

- Un cas particulier de ce type de générateurs est quand :

$$x_n = (x_{n-37} + x_{n-100}) \bmod 2^{30} \tag{6}$$

Ce générateur a été inventé par Knuth (2002) et est généralement appelé "$Knuth - TAOCP - 2002$" ou simplement "$Knuth - TAOCP$" .

En pratique, on prendra plutot $u_n = (x_n + 1)/(m + 1)$, ou encore $u_n = x_n/(m + 1)$ si $x_n > 0$ et $u_n = m/(m + 1)$ sinon, mais la structure demeure essentiellement la meme.

- Si $k = 1$, nous retrouvons le générateur à congurence linéare classique,avec $c = 0$.

L'état à l'étape n est

$$s_n = x_n = (\mathrm{x_{n-k+1}}, ..., x_n)^T \tag{7}$$

2.3 Propriétés :

Espace d'états Z_m^K, de cardinalité m_k. La periode maximale est $\rho = m_{k-1}$. pour m premier. On associe au MRG le polynome caracteristique :

$$P(z) = z^k a_1 z^{k-1} - ... - a_k = -\sum_{j=1}^{K} a_j z^{k-j} \tag{8}$$

ou $a_0 = -1$.
Pour $k > 1$, pour avoir une periode maximale, il est possible de montrer qu'il suffit davoir au moins deux coefficients non nuls, dont a_k. Ainsi, la recurrence la plus economique a la forme :

$$x_n = (a_r x_{nr} + a_k x_{nk}) \bmod m \tag{9}$$

avec 0< r < k .

Une erreur frequente, commise en particulier par les informaticiens peu au fait des statistiques, est de considerer $m = 2^e$. Utiliser une puissance de 2 pour m permet en effet de facilement calculer le produit $ax \bmod m$, et est parfois d ecrit comme efficace, ce qui est vrai du point de la rapiditée d'execution. Les effets sur la periode sont pourtant dommageables, vu que :

pour $k = 1$ et $e \geq 4, on\ a(\rho \leq 2^{e2})$.

pour k > 1, on a $\rho \leq (2^k - 1)^{2^e - 1}$.

Exemple *: Si* $k = 7$ *et* $m = 2^{31} - 1$ *, la période maximale est* $(2^{31} - 1)^7 - 1 \approx 2^{217}$ *. Mais pour* $m = 2^{31}$ *on a :* $\rho \leq (2^7 - 1)2^{31} - 1 < 2^{37}$ *, i.e.*2^{180} *fois plus petit !.*

Pire, si nous nous interessons au i^{em} bit le moins significatif, pour $k = 1$, la période de $xn \bmod 2^i$ ne peut pas depasser $max(1, 2^{i-2})$ Pour $k > 1$, la période de $xn \bmod 2^i$ ne peut pas depasser $(2^k - 1)2^{i-1}$.

2.4 étude de periode

Premier example :

comme vous voyer degré 3 et le modulo aussi 3 on choisie des valeur aléatoire pour x_1, x_2, x_3
puis il y a le polynôme primitive qui est généré a partir de degré et modulo
comme notre but c'est calculer
$x_n = -a_0 * x_(n-1) - a_1 * x_(n-2) - ..... - a_n * x_(n-k) mod p$

il faut choisir un m (dans l'exemple c'est : 6) la fonction calcule tout les valeur de $x_n x_n + 1 .... jusqu' x_m$
par exemple pour m=6
le résultat 1 2 1 1 2 2
les trois premier valeur sont les valeur de x_1 x_2 x_3 que nous venons d'entre

$x_4 = 1$
$x_5 = 2$
$x_6 = 2$

```
n (degré de polynome)=3
p (modulo) = 3
x0 = 1
x1 = 2
x2 = 1
le polynome pémitive :[1 1 2 ]
choisissez un  nombre suppérieur à 3 afin de calculer x(votre nombre)
6
les valeurs des x0>>x(6):[1 2 1 1 2 2]
```

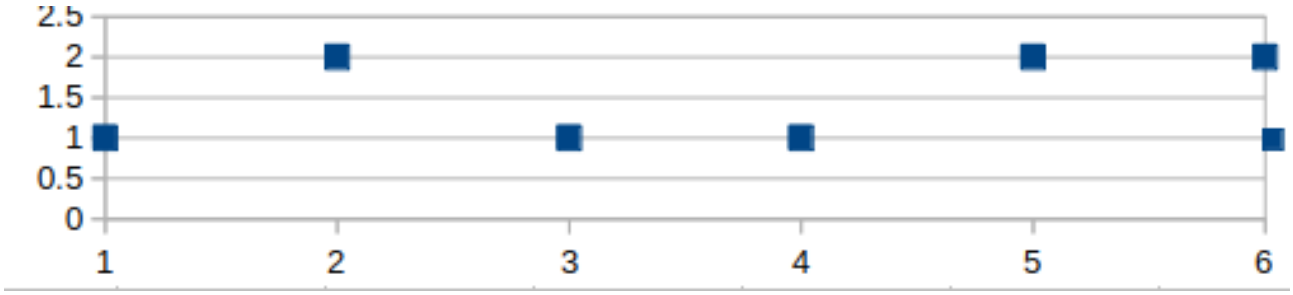

FIGURE 6 – exemple 1 : courbe d'un test de Générateurs récursifs multiples (d'une période)

Le deuxième exemple est plus intéressant :

les même valeur dans les deux exécutions mais le résultat est différent grâce au polynôme primitif

```
   n (degré de polynome)=5
p (modulo) = 2
x0 = 1
x1 = 2
x2 = 0
x3 = 1
x4 = 2
le polynome pémitive :[1 0 0 1 0 1 ]
choisissez un  nombre suppérieur à 5 afin de calculer x(votre nombre)
15
les valeurs des x(0)>>x(15):[1 0 0 1 0 1 1 1 0 0 1 0 1 1 1 ]
  n (degré de polynome)=5
p (modulo) = 2
x0 = 1
x1 = 2
x2 = 0
x3 = 1
x4 = 2
le polynome pémitive :[1 1 0 1 1 1  ]
choisissez un  nombre suppérieur à 5 afin de calculer x(votre nombre)
15
les valeurs des x(0)>>x(15):[1 0 0 1 0 1 0 0 0 1 0 1 0 0 0  ]
```

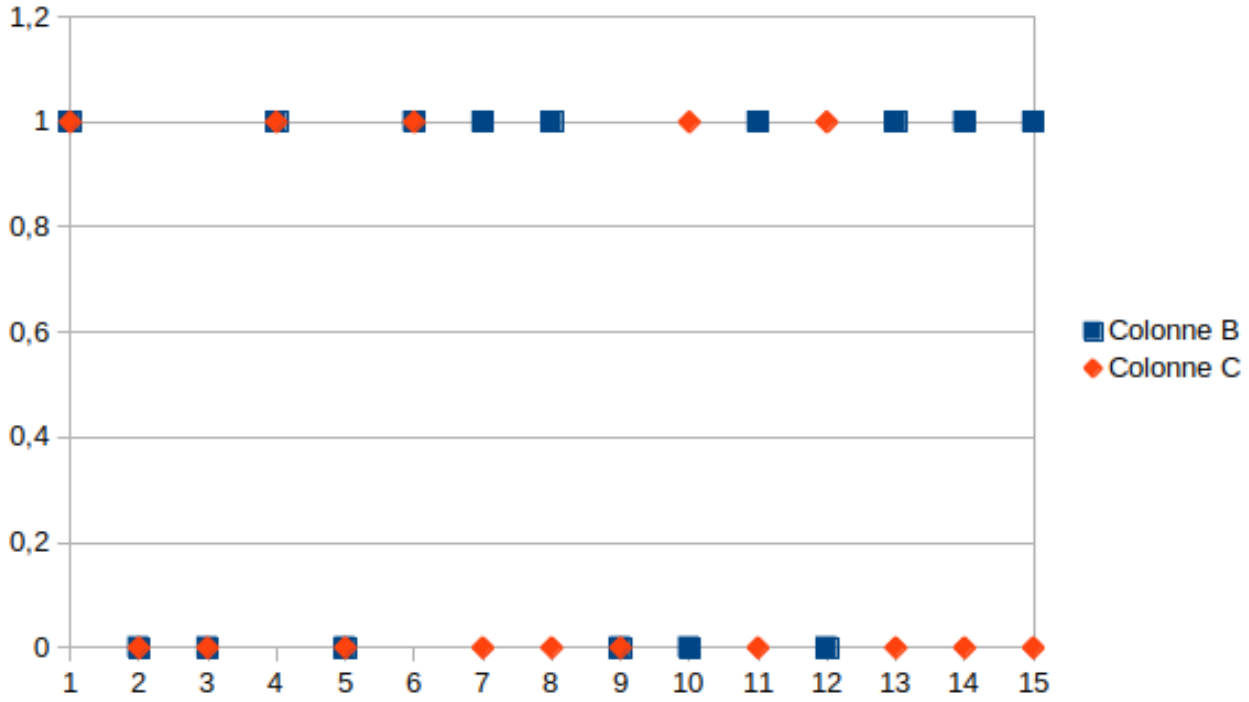

FIGURE 7 – Graphe d'un test de Générateurs récursifs multiples (avec polynome premitive différente)

On peut utiliser les résultats comme des séquence de nombres plus ou moins longue qui a lair dêtre aléatoire, alors quelle est en réalité déterministe pour chiffrer.

2.5 Iplémentation

1. une fonction pour généré un polynome irreductible aléatoire : pour cela on a utilisé la bibliothéque NTL (c++) on lui passe au parametre le degré et le modulo et elle nous donne une polynome irreductible aléatoire.
2. -on récupére le polynome
   tester si l'ordre([X])= $p^n - 1$
   * fonction qui calcule x^i mod $f(x)$ pour i alant de 1 à $p^n - 1$ pour tous les valeur de i de 1 a $p^n - 2$ le calcule doit donner une valeur différente de 1 pour i = $p^n - 1$ la valeur du calcule que doit étre donné est 1 si ce n'est pas le cas alors il faut chercher un autre polynome irréductible (recommencé depuis le début).

   on récupére les coefficients $(-a_1, -a_2, ..., -a_k)$ puis on fait le calcule de

$$x_n = (a_1 \mathrm{x}_{\mathrm{n}-1} + ... + a_k \mathrm{x}_{\mathrm{n}-\mathrm{k}}) \bmod m \tag{10}$$

3 Générateurs congruentiels inverses

3.1 Définition :

Les générateurs congruentiels inverses sont un type de générateur de nombres pseudo-aléatoires congruentiels non linéaires, qui utilisent l'inverse multiplicatif modulaire (s'il existe) pour générer les nombres dans une séquence. L'avantage des méthodes non linéaires est qu'elles ne produisent pas de structure en réseau et que leurs sorties se comportent plutôt comme des nombres «véritablement» aléatoires, même sur toute la période. La formule standard pour un générateur congruentiel inversif, modulo certains premiers q est :

$$X_0 = Graine \tag{11}$$

$$X_{i+1} \equiv (ax_i^{-1} + c) \bmod q \ , si \ X_0 \neq 0 \tag{12}$$

$$X_{i+1} \equiv c \ , si \ X_0 = 0 \tag{13}$$

3.2 Propriétes :

Un tel générateur est désigné symboliquement par ICG (q, a, c, graine) et est dit être un ICG avec les paramètres q, a, c et graine de semence.

La séquence $(x_n)_{n\geq 0}$ doit avoir $x_i = x_j$ après un nombre fini d'étapes et puisque l'élément suivant ne dépend que de son prédécesseur direct, $x_i + 1 = x_j + 1$ ... etc. La longueur maximale que peut avoir la période T pour une fonction modulo q est T = q. Si le polynôme $f(x) = x^2 - cx - a \in Fq[x]$ (anneau polynomial sur F_q) est primitive, la séquence aura alors la longueur maximale. Ces polynômes sont appelés polynômes de période maximale inverse (IMP).

La condition suffisante pour une période de séquence maximale est un choix approprié des paramètres a et c , Eichenauer-Herrmann, Lehn, Grothe et Niederreiter ont montré que les générateurs congruentiels inversifs possèdent de bonnes propriétés d'uniformité, en particulier en ce qui concerne la structure du réseau et les corrélations en série.

3.3 étude de periode

Premier example :

un mauvais choix de coefficient ça nous a amené a avoir une suite de nombre avec une période minimale (mauvais choix puisuqe on veut avoir une période maximal)

```
donner un M= 37
donner un a= 15
donner un c= 0
donner un x(0)= 10
```

les résultats donné :

```
x(0)=10
x(1)=20
x(2)=10
la periode est : 2
```

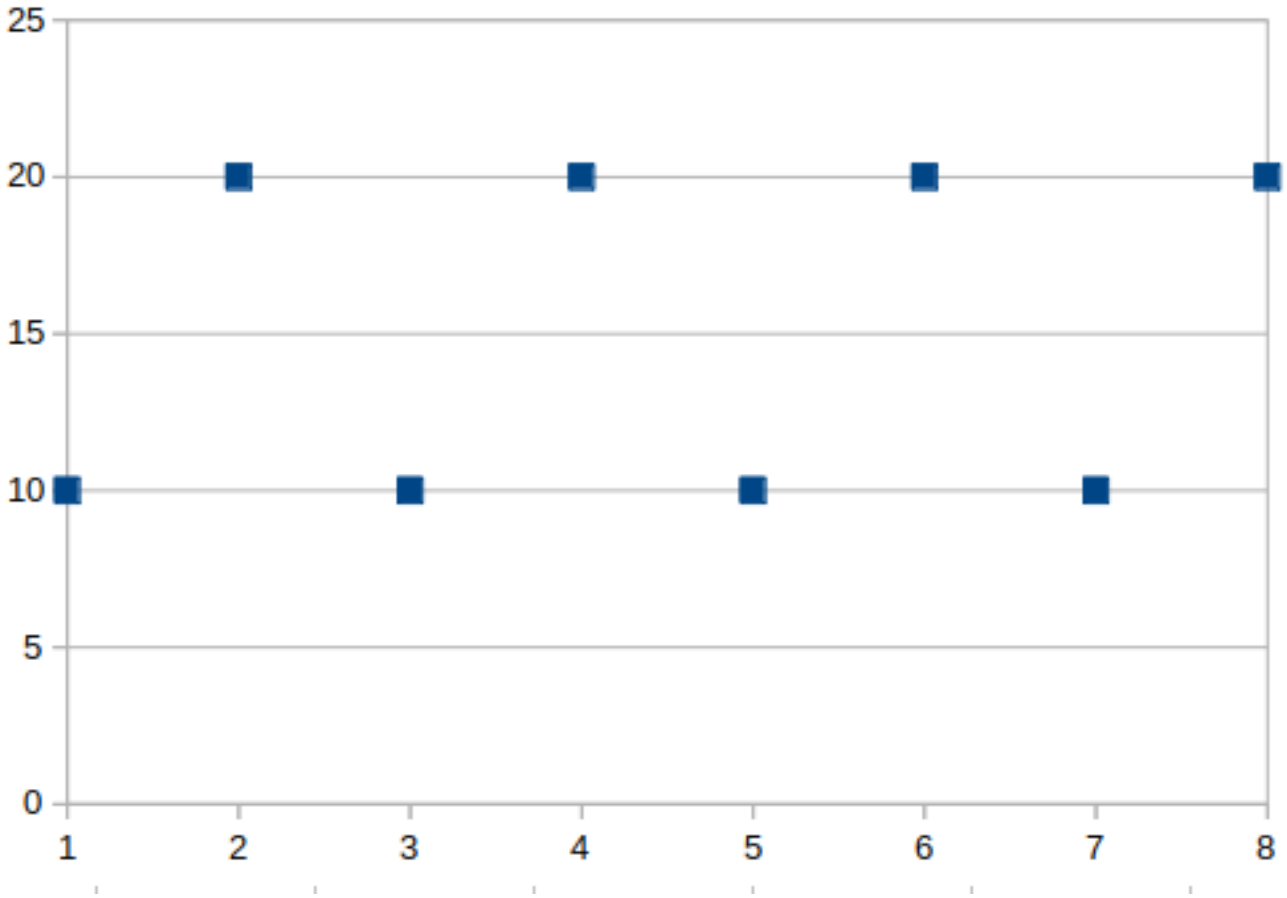

FIGURE 8 – Mauvais choix de coefficients :Graphe de Générateurs congruentiels inverses (plusieur périodes)

Deuxième example :

un bon choix de coefficients ça nous a amené a avoir une suite de nombre avec une période Maximale M.

```
  donner un M= 37
donner un a= 56
donner un c= 15
donner un x(0)= 15
```

les résultats donné :

```
x(0)=15
x(1)=36
x(2)=33
x(3)=1
x(4)=34
x(5)=21
x(6)=30
x(7)=7
x(8)=23
....
....
x(34)=19
x(35)=16
x(36)=0
la periode est :37
```

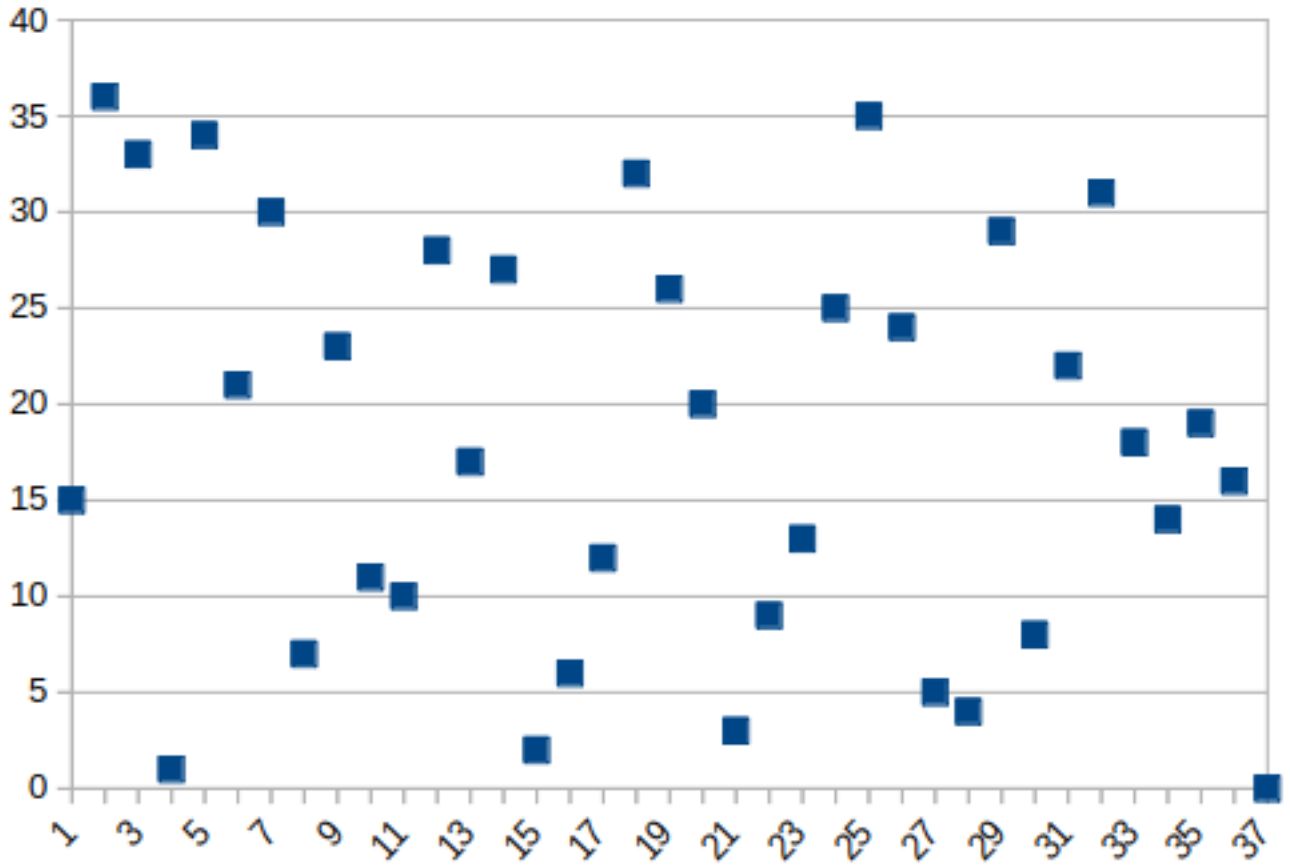

FIGURE 9 – graphe : Générateurs congruentiels inverses résultat (pirode maximal)

4 Générateur congruentiel inverse explicite

4.1 Définition :

Cette méthode est due à Eichermann et Herrmann. Elle est basée sur la récurrence :

$$\overline{\mathrm{x_n = an + c}} \bmod m = (an + x)^{m-2} \bmod m \tag{14}$$

la valeur de sortie est donnée par :

$$U_n = \frac{Z_n}{m} \tag{15}$$

Si m est premier, la longueur de la période est $\rho = m$. Cet algorithme satisfait au test en série dimensionnel pour tout $s \geq m - 2$. Il a un comportement optimal sous le test du réseau. Il vérifie une forte propriété de non-linéarité comme le générateur inverse récursif .

4.2 Propriétés :

Ces générateurs répondent aux propriétés suivantes :

1. Aucune structure de réseau non triviale n'est observée
2. Forte propriété de non-linéarité
3. Le test en série s-dimensionnel est satisfait, pour $s \leq d$ où d est une valeur spécifique.
4. un grand choix de paramètres est autorisé

Le principal inconvénient de ces deux générateurs inverses est qu'un calcul d'un nombre aléatoire prend une multiplication de $O(log\ m)$, de sorte que l'algorithme n'est pas rapide .

4.3 étude de periode

Premier example :

un mauvais choix de coefficients :

```
donner un M = 7
donner un a = 3
donner un c = 0
donner un x(0) = 0
```

les résultats donné :

```
x(0)=2
x(1)=6
x(2)=2

la periode est :2
```

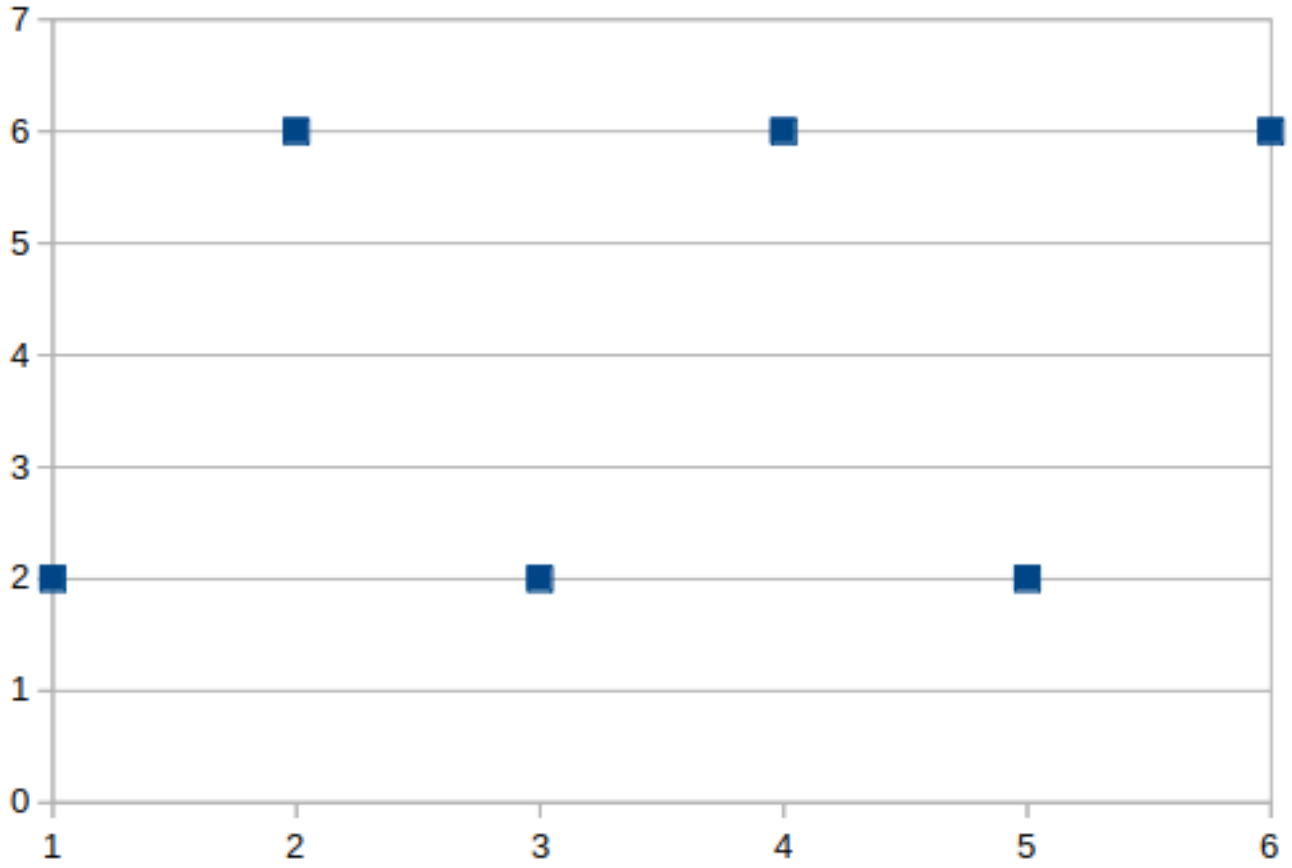

FIGURE 10 – Mauvais application (Graphe) : Générateur congruentiel inverse explicit (plusieur période)

Deuxième example :

des fois meme que le choix de coefficients est bon on trouvra le probleme d'inverse (puisque il y a certain nombre qui n'ont pas d'inverse)

```
donner un M= 101
donner un a= 156
donner un c= 5
donner un x(0)= 61
```

les résultats donné :

```
x(14)=70
x(15)=6
x(16)=60
x(17)=18
x(18)=74
x(19)=26
x(20)=77
x(21)=50
x(22)=83
....
....
x(46)=36
x(47)=75
x(48)=55
pas d'inverse a x(49)

la periode est :49
```

on voit que le calcule c'est arreter a un moment car le nombre n'avais pas d'inverse.

les résultats donné :

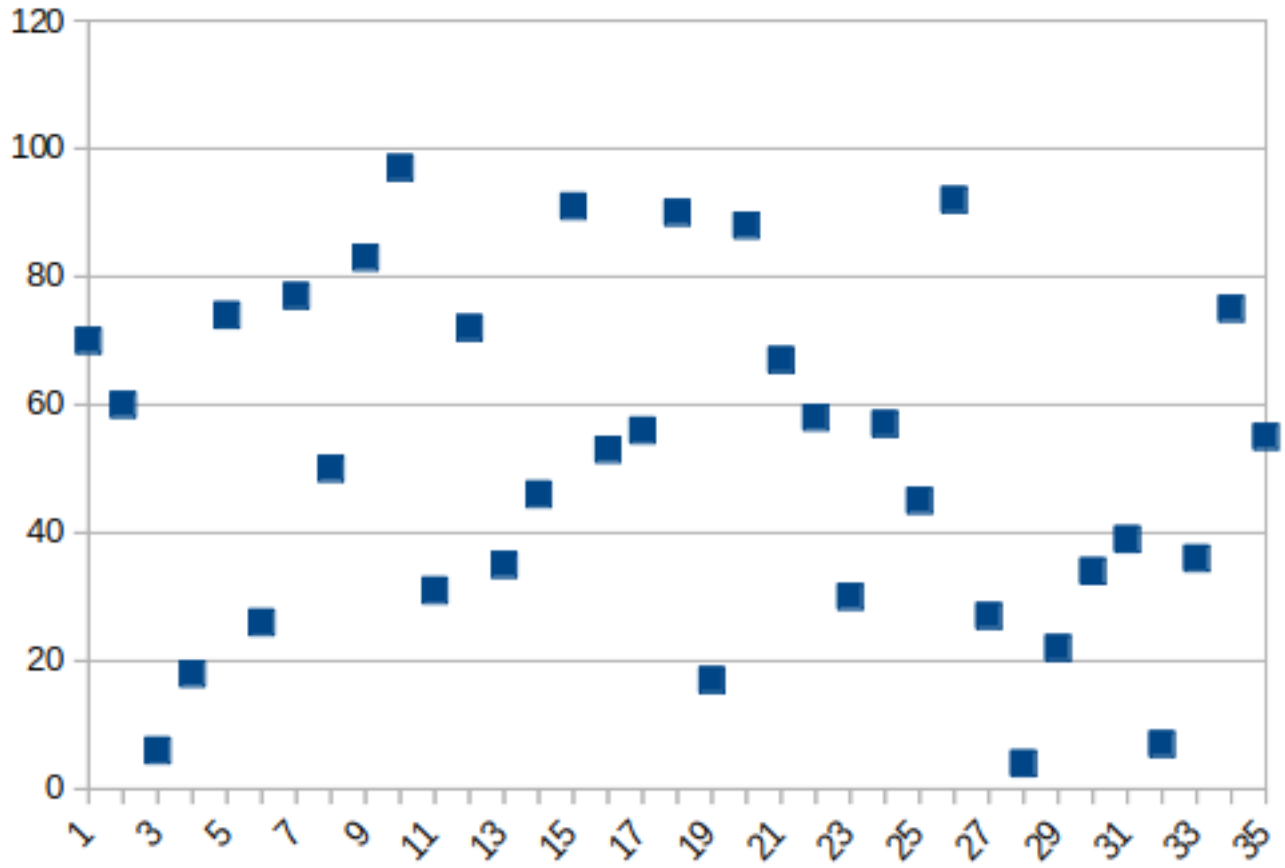

FIGURE 11 – Graphe example : Générateur congruentiel inverse explicit

Troisième example :

des fois meme que le choix de coefficients est bon on trouvra le probleme d'inverse (puisque il y a certain nombre qui n'ont pas d'inverse)

```
donner un M= 10144867
donner un c= 55485
donner un x(0)= 61
```

a = 102646 a = 146865 a = 486 a = 68446 a = 17

Le temp d'execution :

0.000442 0.000446 0.000197 0.000346 0.000278

La période :

25(pas d'inverse) 10(Pas D'inv) 60 (Pas D'inv) 235 (Pas D'inv) 10 (Pas D'inv)

le graphe des points pour a=486 (Colonne D), a=68446 (Colonne E) et a=17 (Colonne F) :

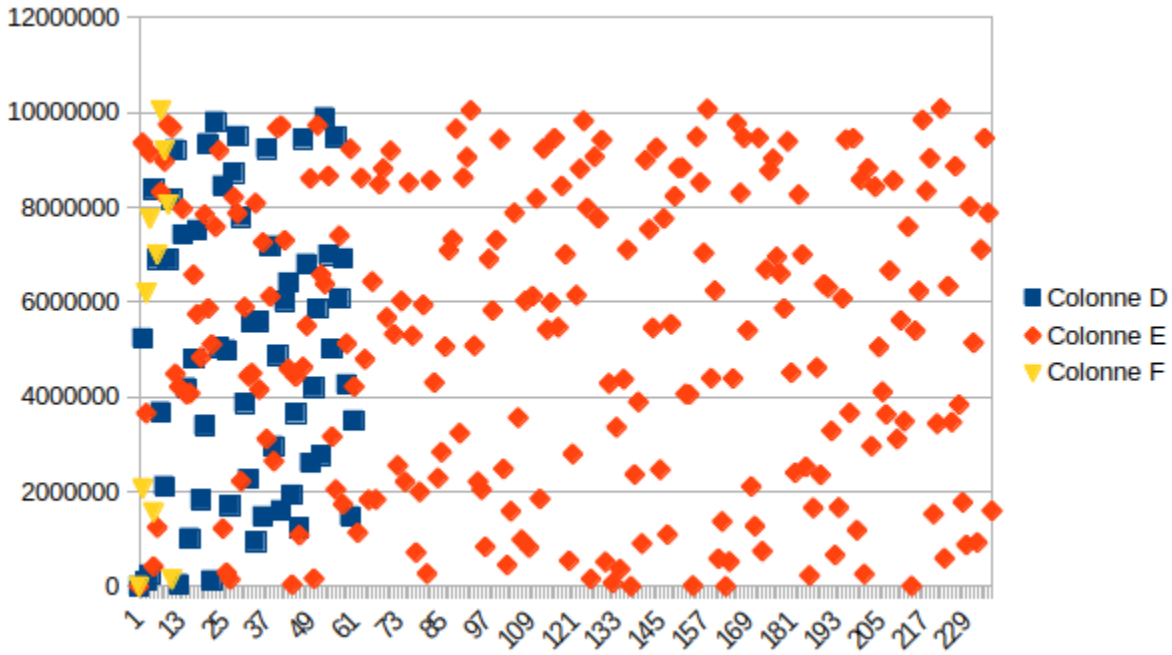

FIGURE 12 – Graphe example : Générateur congruentiel inverse explicit

4.4 Iplémentation

Nous avons programmer en C a l'aide de la bibliothèque GMP.
cette bibliothèque nous permet de calculer l'inverse facilement avec la fonction

int mpz_invert (mpz_t rop, const mpz_t op1, const mpz_t op2)

5 Générateur Inverse Numérique

5.1 Définition :

afin de trouver un générateur inversif plus rapide que l'inverse récursif ou l'inverse explicite, une nouvelle méthode appelée méthode inverse numérique a été récemment étudiée. Pour une précision k, soit F_q le corps fini d'ordre $q = 2^k$ et $F * q$ le groupe multiplicatif d'éléments non nuls de F_q.Puis considérons la récursion :

$$\varphi_{n+1} = \alpha\overline{\varphi_n} + \beta \tag{16}$$

avec $\alpha \in F_q^*$, $\beta \in F_q$, $\varphi_0 \in F_q$ et nous définissons le nombre pseudo-aléatoire inversif numérique par :

$$u_n = \sum_{j=1}^{k} c_n^{(j)} 2^{-j} \tag{17}$$

avec $c_n = (c_n^{(1)}, ..., c_n^{(k)}) \in Z_2^k$ le vecteur de coordonnées de φ_n par rapport à la base ordonnée B.

5.2 Propriétés :

La longueur de la période est $\rho = m$ si et seulement si $x^2 - \beta x - \alpha$ est un polynôme IMP sur F_q. En particulier $\rho = m$ si le polynôme est primitif.

6 Générateur Inversif Composé

6.1 Définition :

La construction d'un générateur inversé composé (CIG) repose sur la combinaison de deux générateurs inverses congruentiels ou plus selon la méthode décrite ci-dessous.

Soit $p_1....p_r$ des entiers premiers distincts, chaque $p_j \geq 5$. Pour chaque indice j,$1 \leq j \leq r$, posons $(x_n)_{n\geq 0}$, soit une séquence d'éléments de $\in F_{p_j}$ périodique de longueur p_j . En d'autres termes, $\{x_n^j | 0 \leq n \leq p_j =\in F_{p_j}\}$.

Pour chaque indice j, $1 \leq j \leq r$, on considère $T_j = T/p_j$ *o* $T = p_1....p_r$ est la durée de la séquence suivante $(x_n)_{n\geq 0}$.

La séquence $(x_n)_{n\geq 0}$ de nombres pseudo-aléatoires composés est définie comme la somme :

$$x_n = T_1 x_n^{(1)} + T_2 x_n^{(2)} + +T_r x_n^{(r)} \bmod T \tag{18}$$

L'approche composée permet de combiner des générateurs inverses congruentiels, à condition qu'ils aient une période complète, dans des systèmes de génération parallèles.

6.2 Les avantages de Générateur Inversif Composé :

Les CIG sont acceptés à des fins pratiques pour un certain nombre de raisons.

Premièrement, les séquences binaires ainsi produites sont exemptes de déviations statistiques indésirables. Les séquences inverses testées de manière approfondie avec une variété de tests statistiques restent stables sous la variation de paramètre.

Deuxièmement, il existe une méthode simple et stable de choix de paramètres, basée sur l'algorithme de Chou , qui garantit une durée maximale de la période.

Troisièmement, l'approche composée a les mêmes propriétés que les générateurs inverses simples, mais fournit également une longueur de période nettement supérieure à celle obtenue avec un seul générateur congruentiel inversé.

Ils semblent être conçus pour être utilisés avec des plates-formes matérielles parallèles multiprocesseurs.

Il existe un algorithme qui permet de concevoir des générateurs de composés avec une longueur de période prévisible, un niveau de complexité linéaire prévisible, avec d'excellentes propriétés statistiques des flux de bits produits. La procédure de conception de cette structure complexe commence par la définition du corps fini de p éléments et se termine par le choix des paramètres a et c pour chaque générateur de congruences inverses constituant le composant du générateur de composés.

Cela signifie que chaque générateur est associé à un polynôme de période maximale inversive fixe. Une telle condition est suffisante pour la période maximale de chaque générateur congruentiel inversé et enfin pour la période maximale du générateur de composés.

La construction de polynômes période maximale inversive est l'approche la plus efficace pour trouver les paramètres du générateur congruentiel inversé avec une durée de période maximale.

7 comparaison des générateurs de nombres aléatoires

Dans cette section, nous résumons quelques remarques sur les différents groupes de générateurs de nombres aléatoires afin de les comparer et de trouver une méthode adaptée à différents types de simulations.

Les générateurs LCG ont de fortes propriétés d'uniformité, mais leur structure en réseau peut les rendre inappropriés dans de nombreuses applications.

Les générateurs inversifs fournissent un véritable caractère aléatoire en référence au test en série s-dimensionnel et à un manque de structure de réseau. La méthode la plus prometteuse semble être la méthode numérique inverse, qui combine l'avantage de la méthode non linéaire et un algorithme rapide.

8 Bibliotheques étudiées :

8.1 GMP :

GMP est une bibliothèque libre pour l'arithmétique en précision arbitraire, fonctionnant sur des entiers signés, des nombres rationnels et des nombres à virgule flottante. Il n'y a pas de limite pratique à la précision, à l'exception de celles impliquées par la mémoire disponible dans la machine sur laquelle GMP s'exécute. GMP a un riche ensemble de fonctions, et les fonctions ont une interface régulière.

Les principales applications cibles des BPF sont les applications et la recherche en cryptographie, les applications de sécurité Internet, les systèmes dalgèbre, la recherche en algèbre informatique, etc.

GMP est soigneusement conçu pour être aussi rapide que possible, aussi bien pour les petits opérandes que pour les grands opérandes. La rapidité est obtenue en utilisant fullwords comme type arithmétique de base, en utilisant des algorithmes rapides, avec un code assemblage hautement optimisé pour les boucles internes les plus courantes de nombreux processeurs, et en mettant l'accent sur la vitesse.

La première version de GMP a été publiée en 1991. Elle est développée et maintenue en permanence, avec une nouvelle version environ une fois par an.

Depuis la version 6, GMP est distribué sous les licences doubles, GNU LGPL v3 et GNU GPL v2. Ces licences rendent la bibliothèque libre d'utilisation, de partage et d'amélioration, et vous permettent de transmettre le résultat. Les licences GNU donnent des libertés, mais imposent également des restrictions fermes à l'utilisation de programmes non libres.

Les principales plates-formes cibles de GMP sont les systèmes de type Unix, tels que GNU / Linux, Solaris, HP-UX, Mac OS X / Darwin, BSD, AIX, etc. Il est également connu que Windows fonctionne aussi bien en version 32 bits qu'en version 64 bits. mode bit.

GMP fait partie de projet GNU , qui est un système d'exploitation sous forme de logiciel libre, c'est-à-dire qu'il respecte la liberté des utilisateurs. Le système d'exploitation GNU se compose de packages GNU (programmes spécifiquement publiés par le projet GNU) ainsi que de logiciels libres publiés par des tiers. Le développement de GNU a rendu possible l'utilisation d'un ordinateur sans logiciel qui piétinerait la liberté de l'utilisateur.

8.2 FLINT :

FLINT est une bibliothèque C pour la théorie des nombres, maintenue par William Hart.

FLINT était sous licence GPL v2 + jusqu'à la version 2.5 incluse. la version de développement actuelle (et les versions ultérieures) sont sous licence LGPL v2.1 + après que les contributeurs aient accepté de changer de licence en avril 2016.

FLINT prend en charge l'arithmétique avec des nombres, des polynômes, des séries de puissances et des matrices sur de nombreux anneaux de base, notamment :

Entiers multiprécisions et rationnels
Entiers modulo n
nombres p-adiques
Champs finis (ordre premier et non premier)
Nombres réels et complexes (via la bibliothèque d'extension Arb)

Un support est également en cours de développement pour les champs de nombres algébriques (via la bibliothèque dextensions Antic).

Les opérations pouvant être exécutées comprennent les conversions, l'arithmétique, l'informatique, la factorisation, la résolution de systèmes linéaires et l'évaluation de fonctions spéciales. En outre, FLINT fournit diverses routines de bas niveau pour une arithmétique rapide. FLINT est abondamment documenté et testé.

FLINT est écrit en ANSI C et fonctionne sur de nombreuses plates-formes (y compris Linux, Mac OS X et Windows sur des configurations matérielles courantes), mais est actuellement principalement optimisé pour les processeurs x86 et x86-64. Il est conçu pour être thread-safe. FLINT dépend des bibliothèques MPIR / GMP et MPFR.

FLINT a été utilisé pour les calculs à grande échelle dans la recherche sur la théorie des nombres (par exemple : A Trillion Triangles), et convient également comme back-end à usage général pour les systèmes de calcul algébrique. Sage utilise FLINT comme package par défaut pour l'arithmétique polynomiale sur Z, Q et Z / nZ pour les petits n, et des travaux sont en cours pour utiliser FLINT dans Singular et Macaulay2.

Remarque : lors notre implémentation pour des algorithmes on avait besoin des polynomes irréductibles , donc on as etudié la bibliotheque FLINT , mais cette derniere ne permet pas de générer des polynomes irreductibles aléàtoires avec les memes coefficients c'est pour cela on as choisi la bibliotheque NTL qu'on en parle dans la suite de rapport qui nous as ramené a faire ça .

8.3 NTL :

NTL est une bibliothèque C ++ portable hautes performances fournissant des structures de données et des algorithmes permettant de manipuler des entiers de longueur signée, signés, et des vecteurs, matrices et polynômes sur des entiers et des champs finis.

Par défaut, NTL est thread-safe. Depuis la version 9.5, NTL est désormais non seulement thread-safe, mais inclut une nouvelle fonctionnalité de renforcement de thread, qui utilise plusieurs curs pour accélérer les calculs de bas niveau. Ceci est un travail en cours, et à partir de maintenant, seule l'arithmétique ZZ_pX est renforcée par les threads.

NTL est distribué sous LGPLv2.1 +

L'auteur de NTL a récemment reçu le prix commémoratif Richard Dimick Jenks ACM / SIGSAM pour l'excellence en génie logiciel appliqué à Computer Algebra pour ses travaux sur NTL.

Références

[1] D.E.KNUTH. *The Art of Computer programming, Seminumerical Algorithms, volume 2.* . Addison-Wesley, 1981

[2] Pierre-Louis BAYLE -Benjamin BILLET. *NOMBRES PSEUDO-ALEATOIRES*.

[3] Christophe Dutang et Diethelm Wuertz. *A note on random number generation* .

[4] Anne GILLE-GENEST. *Pseudo-Random Numbers Generators*. 2009

[5] Sécurité informatique et nombres aléatoires.
`https://unearaigneeauplafond.fr/securite-informatique-generateur-nombres-aleatoires/`