

# Test et optimisation d'un générateur pseudo-aléatoire

*TIPE de Philippe GAMBETTE (TIPE d'info, concours ENS 2003)  
Commentaires en dernière page...*

## Introduction

La programmation de simulations, modélisations ou jeux demande l'existence d'une source de nombres aléatoires, tout comme les systèmes de cryptage. S'il est possible d'en créer à partir de procédés physiques, ceci requiert tout de même des installations rares et souvent encombrantes.

Alors, comment créer des listes de nombres aléatoires grâce aux ordinateurs, appareils tout à fait déterministes ? Les nombres proposés par les ordinateurs, ou autres calculatrices ne sont en fait que des nombres pseudo-aléatoires, c'est à dire des termes d'une suite  $(X_n)$  d'entiers parfaitement déterminés.

Nous nous intéresserons donc à un générateur pseudo-aléatoire simple, qui fournit une suite d'entiers répartis uniformément dans un intervalle  $[0, m[$ , et indépendants. Agir sur les paramètres de ce générateur permettra d'optimiser l'uniformité et l'indépendance des nombres obtenus, en rejetant les générateurs qui ne permettent pas de respecter ces deux critères.

## I/ Préliminaire

Il est possible d'utiliser des processus physiques afin d'obtenir des nombres vraiment aléatoires. C'est ce que proposent des sites web qui envoient à la demande des nombres aléatoires, ou des périphériques qui se fondent sur le caractère aléatoire du trajet des photons (voir annexe). Toutefois, ces méthodes présentent un inconvénient certain : si l'on utilise les nombres aléatoires pour une expérience que l'on veut refaire en changeant quelques paramètres, mais en utilisant les mêmes nombres aléatoires, on est obligé de les stocker en mémoire.

L'utilisation de suites déterminées, qui paraissent aléatoire, permet d'éviter ce problème. De plus, ces suites pseudo-aléatoires sont étudiées depuis assez longtemps pour que l'on connaisse bien leur comportement, ce qui n'est pas le cas d'une suite vraiment aléatoire qui peut prendre des valeurs inattendues qui fausseraient une simulation.

On va donc utiliser un générateur simple : le **Générateur à Congruence Linéaire** (GCL), proposé par Lehmer en 1949. On définit la suite  $(X_n)$  de la façon suivante :

Soit  $m \in \mathbb{N}$ ;  $a, c$  et  $X_0 \in ]0, m[ \cap \mathbb{N}$

On appelle  $m$  le module,  $a$  le multiplicateur,  $c$  l'incrément et  $X_0$  la graine.

On pose pour  $n \geq 0$  :  $X_{n+1} = (a X_n + c) \bmod m$

La suite  $(X_n)$  ainsi définie est périodique à partir d'un certain rang. Afin d'obtenir un générateur aléatoire performant, il faudra donc commencer par avoir une période suffisamment longue.

La formule donnant  $X_{n+k}$  en fonction de  $X_n$  est utile peut s'avérer utile, notamment pour calculer des nombres pseudo-aléatoires lors de la répétition d'une expérience : on a pour  $n \geq 0, k \geq 1$  :

$$X_{n+k} = (a^k X_n + (a^k - 1) \frac{c}{a-1}) \bmod m \quad (\text{démonstration en annexe})$$

Il est plus facile pour un ordinateur qui gère les données en binaire de calculer avec des modulo  $2^n$ . C'est pourquoi on se restreindra par la suite à une multiplicité  $m = 2^n$ .

Enfin, nous serons parfois amenés, au cours des tests, à utiliser la suite  $U_n = X_n / m \in [0, 1[$

## II/ Le test de la période

Considérons le générateur suivant : le GCL avec  $m = 65536, a = 422, c = 987, X_0 = 641$ .

On a :  $X_1 = 641, X_2 = 129, X_3 = 129, X_4 = 129 \dots$  On obtient donc une suite stationnaire, qui n'est évidemment pas pseudo-aléatoire. Il est donc indispensable de rejeter des générateurs ayant une période trop petite. On peut procéder de deux façons :

- trouver des conditions sur  $a$  et  $c$  qui assurent une période assez grande pour le GCL.
- déterminer la période du GCL choisi, et rejeter le générateur si la période est trop courte.

Le théorème suivant donne une condition nécessaire et suffisante sur la période  $\lambda$  du GCL :

$$\begin{array}{l} c \text{ impair} \\ a \equiv 1 \pmod{4} \end{array} \iff \lambda = 2^n$$

**Démonstration** : Voir annexe

Mais il est possible d'obtenir un bon générateur sans atteindre la période maximale. On doit donc être capable de déterminer exactement la période de tout générateur pseudo-aléatoire.

En outre, les générateurs à congruence linéaire sont assez simples et peuvent être facilement implémentés sur des ordinateurs 32 bits, mais ceci impose un module inférieur à  $2^{31}$ . De même, la période sera aussi inférieure à  $2^{31}$  : elle pourra donc être dépassée après quelques minutes de calcul sur un ordinateur. C'est pourquoi on pourra utiliser des générateurs de période plus élevée : les générateurs récursifs multiples de la forme  $X_{n+1} = (a_1 X_{n-1} + \dots + a_k X_{n-k} + b) \bmod m$ , ou les combinaisons de GCL, ou GRM. On doit donc être, là aussi, capable de déterminer la période du générateur en calculant ses premiers termes.

Un algorithme évident pour trouver la période d'un générateur de la forme  $X_{n+1} = f(X_n)$  serait de comparer chaque valeur de  $X_n$ , aux  $n$  valeurs précédentes, jusqu'à ce qu'il existe  $n$  et  $T$  tels que  $X_n = X_{n+T}$ . Mais cet algorithme a une complexité en mémoire en  $O(n)$  (stockage des  $X_i$ , pour  $i < n$ ). Sa complexité en temps est en  $O(n^2)$ .

Il existe un algorithme plus performant pour trouver la période de  $(X_n)$  : **l'algorithme de Pollard**. Cet algorithme astucieux s'applique bien grâce à la structure particulière des générateurs pseudo-aléatoires : d'une part, ceux-ci sont tels qu'on ne retrouve pas deux fois le même terme dans une même période ; d'autre part, ils sont périodiques à partir d'un certain rang, ce qui est géré sans problème par l'algorithme.

On pose  $A_0 = B_0$ . On définit alors  $(A_n)$  et  $(B_n)$  par la relation de récurrence :  $A_{n+1} = f(A_n), B_{n+1} = f(f(B_n))$ . Alors, si  $A_i = B_i$ , on calcule  $A_n$  et  $B_n$  jusqu'au rang  $i+k$  tel que  $A_{i+k} = B_{i+k}$ .  $k$  est alors la période du générateur.

Cette algorithme peut être compris en remarquant l'analogie avec deux coureurs cyclistes  $A$  et  $B$  tournant autour d'une piste circulaire,  $B$  allant deux fois plus vite que  $A$ . Alors  $B$  rattrapera  $A$  à chaque tour au même endroit.

Cet algorithme nous permettra d'**éliminer les générateurs à trop petite période**.

Toutefois, un générateur à grande période n'est pas nécessairement un bon générateur.

### **III/ Le test d'uniformité**

Considérons par exemple le GCL avec les paramètres suivants :  $a=1$ ,  $c=1$ ,  $m=1024$ ,  $X_0=0$ . On obtient :  $X_1=1$ ,  $X_2=2$ , ...,  $X_i=i \bmod 1024$ . La période est 1024. Or ce générateur n'est clairement pas aléatoire. Si l'on examine les 500 premières valeurs, par exemple, elles ne sont pas uniformément distribuées dans le segment  $[0,1024[$ . En effet, le segment  $[501,1024[$  ne contient aucun des termes. Pour réaliser intuitivement un test d'uniformité, on peut donc calculer les  $n$  premiers termes de la suite  $(U_n)$  définie par  $U_n=X_n/m$  et découper le segment  $[0,1[$  en  $k$  intervalles de longueur égale.

Par exemple, pour  $k=20$ , et la suite  $(U_n)$  associée au GCL avec  $a=1024$ ,  $c=675$ ,  $m=8977$  et  $X_0=9$ , dans les 1000 premiers termes, on en trouve 39 entre 0,45 et 0,5 et 61 entre 0,55 et 0,6.

Cet écart avec le résultat théorique (50 termes dans chaque intervalle de longueur 0,5) est suspect. Mais, si l'on avait trouvé exactement 50 termes dans chaque intervalle, cela aurait paru trop ordonné.

Il faut donc trouver une façon d'éliminer des générateurs ayant une trop grande ressemblance, ou une trop grande divergence avec le comportement idéal attendu. C'est le test du  $\chi^2$  qui nous permet d'effectuer ce tri.

On choisit tout d'abord  $C$  catégories, par exemple,  $C$  segments de  $[0,1[$ . Soit alors  $N \in \mathbb{N}$ , avec  $N$  assez grand (de préférence supérieur à  $5C^2$ ). On note  $P_s$ , la probabilité pour  $X_n$  de tomber dans la  $s$ -ième catégorie (soit, si l'on travaille avec les  $C$  segments de même longueur,  $1/C$ ), et  $Y_s$ , le nombre réel des termes  $X_n$  dans la  $s$ -ième catégorie.

On définit alors la variance de la manière suivante : 
$$V = \sum_{s=0}^C \frac{(Y_s - N P_s)^2}{N P_s} .$$

Quelles valeurs doit prendre  $V$  pour que la fonction aléatoire soit satisfaisante ? Si  $V$  est trop petit, cela signifie que la fonction aléatoire se rapproche trop des valeurs attendues, et cela paraît suspect. Au contraire, si  $V$  est trop grand, cela montre que les valeurs dites aléatoires sont réparties de façon trop discontinue. Alors, où fixer les limites ?

On se rapporte à un tableau montrant, en fonction de  $C$ , la valeur moyenne de  $V$  pour des générateurs vraiment aléatoires. Ce tableau, établi par Donald E. Knuth, est disponible en annexe.

On interprète ensuite les résultats obtenus pour le générateur pseudo-aléatoire de la façon suivante : la colonne dans laquelle se trouve  $V$  permet de déterminer si le générateur est accepté, presque suspect, suspect, ou rejeté. Il faudra toutefois prendre garde à considérer ces résultats avec précaution. Il est par exemple utile, voire indispensable de réaliser le test d'uniformité sur plusieurs valeurs de  $N$ , afin de ne pas tomber sur une valeur exceptionnelle de  $N$  qui fournit une variance satisfaisante.

Pourquoi est-il primordial d'obtenir des termes uniformément répartis dans  $[0,1[$  pour  $U_n$  ? Si l'on utilise le générateur pseudo-aléatoire pour un jeu de hasard, il est évident qu'il ne doit pas y avoir de segment privilégié dans  $[0,1[$ , sinon, un tricheur pourrait exploiter cette faille du hasard.

On peut aussi visualiser cette nécessité d'uniformité de façon concrète en étudiant l'exemple du calcul de l'intégrale d'une fonction  $f$  sur  $[0,1[$  de manière approchée.

La méthode la plus simple est de se fixer une subdivision de  $[0,1[$  de pas  $1/q$  et de calculer :

$$\sum_{k=1}^n \frac{1}{q} f\left(\frac{k}{q}\right)$$

Mais il est plus judicieux d'effectuer ce calcul en prenant une subdivision au hasard de  $[0,1[$ .

En effet, la première méthode pourrait conduire à négliger certaines variations très brutales de  $f$ , si par exemple  $f$  tendait vers l'infini pour une valeur de  $[0,1[$ , tout en restant intégrable sur  $[0,1[$ . La seconde méthode, au contraire, permet parfois de les détecter. En effet, si l'un des points aléatoires de calcul se trouve à l'intérieur du « pic » de  $f$ , cela permettra d'obtenir une bonne précision.

Il faut donc que les points aléatoires proposés par le générateur puissent atteindre tout le segment  $[0,1[$  donc la suite pseudo-aléatoire doit bien être uniformément distribuée, mais pas trop, au risque de se rapprocher de la première méthode.

### **III/ Le test d'indépendance**

L'indépendance des variables est aussi une partie intrinsèque de la définition de caractère aléatoire.

En effet, dans un jeu de hasard ou un système de cryptographie qui utiliserait un générateur pseudo-aléatoire ne passant pas le test d'indépendance, on pourrait deviner un terme en étudiant les termes précédents.

On peut donc réaliser une fois de plus le test du  $\chi^2$ , sur des catégories choisies.

Le test des séquences croissantes décroissantes, par exemple, se base sur deux catégories :  $X_n > X_{n-1}$  et  $X_n < X_{n-1}$  équiprobables.

Le test du poker se base sur plusieurs catégories : paire, brelan, carré, full...

Il faut donc, comme dans le test d'uniformité, calculer la variance correspondant à ces catégories, puis la comparer aux valeurs de référence avant de conclure.

### **Conclusion**

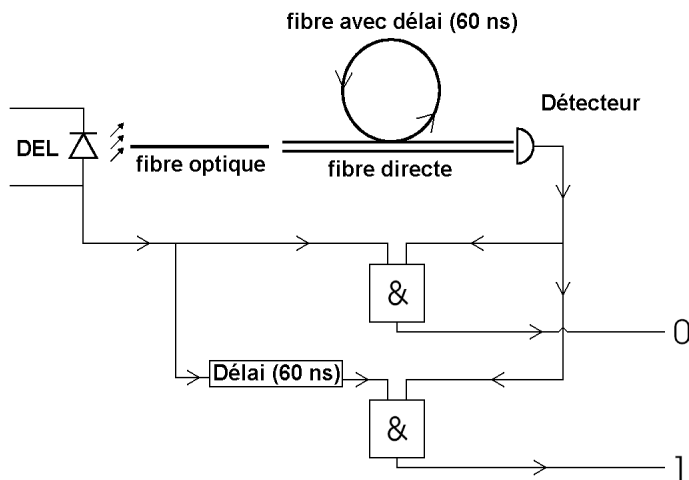
Les trois tests que nous avons vus ne permettent pas, en cas de succès, d'affirmer que le générateur testé est bon. Ils servent à éliminer les générateurs vraiment mauvais. Il est donc impossible de trouver le générateur congruentiel optimal. On peut en revanche avoir des générateurs pseudo-aléatoires satisfaisant à nos demandes. Les générateurs congruentiels linéaires qui sont bien connus, et maîtrisés, sont alors des outils très efficaces pour produire des suites de nombres pseudo-aléatoires.

## Annexes

### • *Le générateur quantique de nombres aléatoires*

Prototypé connectable sur port USB, 68x50x188 mm, réalisé en collaboration avec Deutsche Telekom. (A. Stefanov, O. Guinnard, L. Guinnard, H. Zbinden, N. Gisin)

Avantages : pratique, peu coûteux, performant...



*Schéma de fonctionnement du générateur*

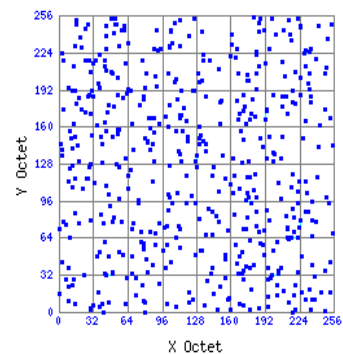


### • *Les lava-lamps*

Le site [www.lavarnd.org](http://www.lavarnd.org) propose des nombres vraiment aléatoires obtenus en prenant régulièrement des photos de lava-lamps à partir d'un webcam. Ces photos sont utilisées pour produire 512 paires de huit bits aléatoires.



*Lava-lamps*



*Paires d'octets aléatoires*

• **Démonstration de la formule du terme général  $X_n$  :**

On procède par récurrence sur  $k$ .

Initialisation :  $X_{n+1} = (a X_n + c) \bmod m = (a^1 X_n + (a^1 - 1) \frac{c}{(a-1)}) \bmod m$  donc la proposition est vraie au rang 1

Hérédité : On suppose la proposition vraie au rang  $k$ . L'est-elle au rang  $k+1$  ?

$X_{n+k} = (a^k X_n + (a^k - 1) \frac{c}{(a-1)}) \bmod m$ , donc  $\exists q \in \mathbb{Z}$  tel que  $X_{n+k} = (a^k X_n + (a^k - 1) \frac{c}{(a-1)}) + mq$ , alors

$$\begin{aligned} X_{n+k+1} &= (a X_{n+k} + c) \bmod m = (a ((a^k X_n + (a^k - 1) \frac{c}{(a-1)}) + mq) + c) \bmod m \\ &= (a^{k+1} X_n + c \frac{(a^{k+1} - a) + (a-1)}{(a-1)}) \bmod m \end{aligned}$$

donc  $X_{n+k+1} = (a^{k+1} X_n + (a^{k+1} - 1) \frac{c}{(a-1)}) \bmod m$  donc la proposition est vraie au rang  $k+1$ . Elle est donc vraie pour tout  $k > 0$ .

La formule étant initialisée et héréditaire, elle est vraie pour tout  $k$  de  $\mathbb{N}$

• **Table de référence des variances :**

p :	1%	5%	25%	50%	75%	95%	99%
2	0.00016	0.00393	0.1015	0.5449	1.323	3.841	6.635
3	0.0201	0.1026	0.5754	1.386	2.773	5.991	9.21
4	0.1148	0.3518	1.213	2.366	4.108	7.815	11.34
5	0.2971	0.7107	1.923	3.357	5.385	9.488	13.28
6	0.5543	1.1455	2.675	4.351	6.626	11.07	15.09
7	0.8721	1.635	3.455	5.348	7.841	12.590	16.81
8	1.239	2.167	4.255	6.346	9.037	14.07	18.48
9	1.646	2.733	5.071	7.344	10.220	15.510	20.09
10	2.088	3.325	5.899	8.343	11.39	16.92	21.67
11	2.558	3.940	6.737	9.342	12.550	18.310	23.210
12	3.053	4.575	7.584	10.34	13.7	19.68	24.72
13	3.571	5.226	8.348	11.340	14.850	21.030	26.22
>13	$C + \sqrt{2C} x_p + \frac{2}{3} x_p^2 - \frac{2}{3} + O\left(\frac{1}{\sqrt{C}}\right)$						
$x_p$	-2.33	-1.64	-0.674	0	0.674	1.64	2.33

$p$  est le pourcentage de suites aléatoires avec un  $V$  inférieur au nombre correspondant

• **Démonstration du théorème de la période :**  $\begin{matrix} c \text{ impair} \\ a \equiv 1 \pmod{4} \end{matrix} \iff \lambda = 2^n$

Quitte à changer  $X_0$ , on suppose  $X_0 = 0$ .

$\Rightarrow$  : On suppose  $c$  impair,  $a \equiv 1 \pmod{4}$ . Soit  $\lambda$ , la période du GCL.

$a \equiv 1 \pmod{4} \Rightarrow \exists q \in \mathbb{N} / a = 1 + 4q$  alors  $\exists q' \text{ impair}, k \geq 2 / a = 1 + 2^k q'$  donc  $a \equiv 1 \pmod{2^k}$  et  $a \not\equiv 1 \pmod{2^{k+1}}$  et  $2^k \geq 4 > 2$ .

**Lemme :**  $p$  premier,  $k \in \mathbb{N}$ ,  $p^k > 2$   
 $x \equiv 1 \pmod{p^k}$  et  $x \not\equiv 1 \pmod{p^{k+1}} \Rightarrow x^p \equiv 1 \pmod{p^{k+1}}$  et  $x^p \not\equiv 1 \pmod{p^{k+2}}$ .

dém :  $x \equiv 1 \pmod{p^k} \Rightarrow \exists q, p \nmid q$  tel que  $x = 1 + qp^k \Rightarrow x^p = \sum_{i=0}^p C_p^i (qp^k)^i = 1 + qp^k \left( \sum_{i=1}^p C_p^i (qp^k)^{i-1} \right)$

Or  $p \mid C_p^i$  donc  $C_p^i = pr_i$ , avec  $r_i = \frac{(p-1)!}{i!(p-i)!}$ , donc  $x^p = 1 + qp^k \left( \sum_{i=1}^{p-1} pr_i (qp^k)^{i-1} \right) + (qp^k)^{p-1}$

donc  $x^p = 1 + qp^{k+1} \left( \sum_{i=1}^{p-1} r_i (qp^k)^{i-1} \right) + q^{p-1} p^{k(p-1)-1}$

\* Si  $p=2$ ,  $p^k > 2 \Rightarrow k \geq 2$  et  $p-1=1$  donc  $q^{p-1} p^{k(p-1)-2} \in \mathbb{N}$

\* Si  $p > 2$ ,  $k \geq 1$  et  $p-1 \geq 2$  donc  $q^{p-1} p^{k(p-1)-2} \in \mathbb{N}$

Ainsi  $x^p = 1 + qp^{k+1} + qp^{k+2} \left( \sum_{i=2}^{p-1} r_i (qp^k)^{i-2} + q^{p-1} p^{k(p-1)-2} \right) = 1 + qp^{k+1} \pmod{qp^{k+2}}$  et  $p \nmid q$

donc  $x^p \equiv 1 \pmod{p^{k+1}}$  et  $x^p \not\equiv 1 \pmod{p^{k+2}}$ .

On peut donc appliquer ce lemme. Par une récurrence immédiate, on a  $a^{2^{k'}} \equiv 1 \pmod{2^{k+k'}}$  et

$a^{2^{k'}} \not\equiv 1 \pmod{2^{k+k'+1}}$  pour tout  $k'$ . Or  $a-1 \equiv 0 \pmod{2^k}$ , donc  $\frac{a^{2^{k'}} - 1}{a-1} \equiv 0 \pmod{2^{k'}}$ . En particulier,

pour  $k'=n$ ,  $X_{2^n} = 0$ . Ainsi,  $\lambda \mid 2^k$  donc  $\exists j \leq n / \lambda = 2^j$ . Si  $j < n$ ,  $\frac{a^{2^j} - 1}{a-1} \not\equiv 0 \pmod{2^{j+1}}$ . A fortiori,

$\frac{a^{2^j} - 1}{a-1} \not\equiv 0 \pmod{2^n}$  et  $c$  impair donc  $\frac{a^{2^j} - 1}{a-1} \not\equiv 0 \pmod{2^n}$  donc  $\lambda \neq 2^j$ . Ainsi,  $\lambda = 2^n$ .

$\Leftarrow$  :  $\lambda = 2^n$

• Supposons par l'absurde que  $c$  pair :  $c = 2p$ .

$$X_k = c \frac{(a^k - 1)}{(a-1)} \pmod{2^n} = 2p \frac{(a^k - 1)}{(a-1)} \pmod{2^n} = \text{donc } \forall k, X_k \not\equiv 1 \pmod{2^n} :$$

absurde car toutes les valeurs de  $\mathbb{Z}/2^n\mathbb{Z}$  doivent être atteintes.

Donc  $c$  impair.

• Supposons par l'absurde que  $a \not\equiv 1 \pmod{4}$ .

\* Soit  $a$  pair. Alors  $\frac{(a^{2^n} - 1)}{(a-1)} \equiv 0 \pmod{2^n} \Rightarrow a^{2^n} - 1 \equiv 0 \pmod{2^n} \Rightarrow a^{2^n}$  est impair : absurde !

Donc  $a$  est impair.

\* Soit  $a \equiv 3 \pmod{4}$ . Or, on montre aisément par récurrence que pour  $k > 0$ ,  $a^{2^{k-1}} \equiv 1 \pmod{2^{k+1}}$ .

Cette récurrence est initialisée par l'hypothèse  $a \equiv 3 \pmod{4}$ . On la propage, en supposant que

$a^{2^{k-2}} \equiv 1 \pmod{2^k}$ , et en en déduisant que dans ce cas, on note  $a^{2^{k-2}} = 1 + q2^k$ .

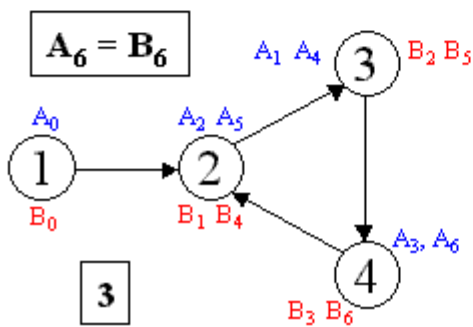
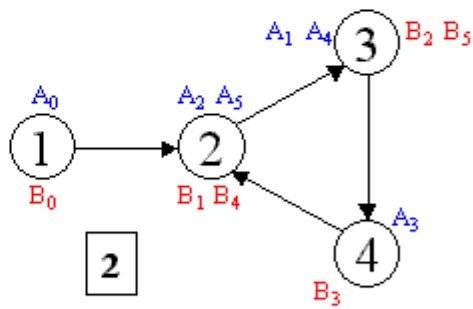
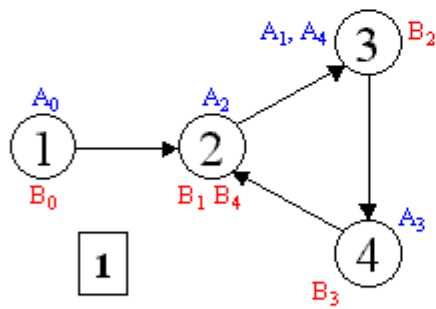
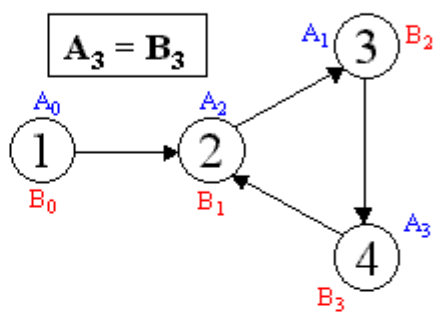
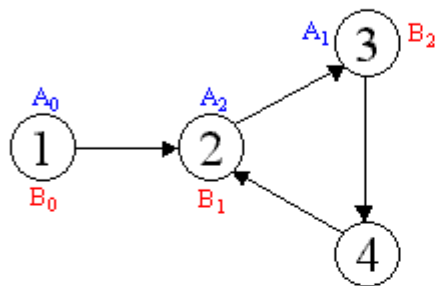
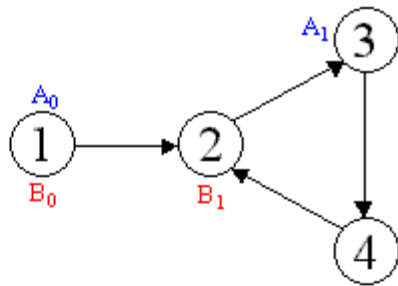
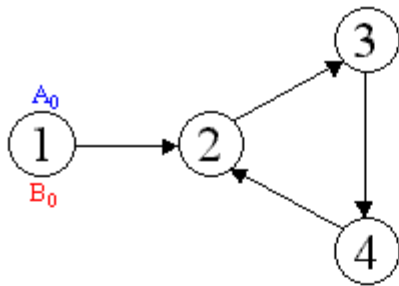
Or  $a^{2^{k-1}} = (a^{2^{k-2}} + 1)(a^{2^{k-2}} - 1) + 1$  soit  $a^{2^{k-1}} = (2^k q + 2)2^k q + 1 = (2^{k-1} q + 1)2^{k+1} q + 1$  donc

$a^{2^{k-1}} \equiv 1 \pmod{2^{k+1}}$ . La propriété est initialisée au rang 1 et se propage, donc elle est vraie pour

tout  $k > 0$ . Ainsi,  $\frac{a^{2^{k-1}} - 1}{a-1} \equiv 0 \pmod{2^k}$  donc, en particulier pour  $k=n$ ,  $X_{2^{n-1}} \equiv 0 \pmod{2^n}$  : absurde !

• On a donc bien  $a \equiv 1 \pmod{4}$ , condition nécessaire pour la maximalité de la période.

• Exemple d'application de l'algorithme de Pollard



La période est donc 3

## Commentaires

Trois ans après, avec un peu d'expériences en sciences en général et en informatique en particulier, un petit commentaire me semble indispensable.

Sur l'épreuve du TIPE tout d'abord, qui pour l'épreuve des ENS constituerait idéalement une première incursion dans le monde de la recherche scientifique... Une introduction qui annonce un peu plus clairement les (modestes) ambitions de ce TIPE, une jolie bibliographie, à la fin d'un document tapé en LaTeX, voilà donc ce qui manquait le plus, du point de vue de la forme, à mon travail (crédité de 14/20, si mes souvenirs sont bons, à l'épreuve des ENS).

Sur le fond, je me suis contenté du sens « en français » de l'indépendance et de l'uniformité, sans penser que les termes étaient parfaitement définis en mathématique. Les examinateurs m'ont fait retrouver au tableau les formules faisant intervenir des probas pour définir tout ça.

Sur le fond aussi, la référence scientifique incontournable pour l'introduction aux générateurs pseudo-aléatoires est *The Art of Computer Programming, volume 2*, par Donald Knuth. Vous trouverez de nombreux articles scientifiques récents sur le sujet écrits par Pierre L'Ecuyer de l'Université de Montréal (<http://www.iro.umontreal.ca/~lecuyer/papers.html>), notamment un test de plusieurs générateurs dans <http://www.iro.umontreal.ca/~lecuyer/myftp/papers/wsc01rng.pdf>. Et aussi bien sûr les nombreux résultats par le formidable moteur de recherche Google Scholar (<http://scholar.google.com/scholar?q=pseudo+random+generators>) qui n'existait pas encore en 2003, et l'article Wikipedia qui est maintenant assez conséquent, en anglais par exemple : [http://en.wikipedia.org/wiki/Pseudo-random\\_number\\_generator](http://en.wikipedia.org/wiki/Pseudo-random_number_generator).