Stéphane Overlen 8 juin 2016

TRAVAUX PRATIQUES IT SOIR 2ÈME ANNÉE

Génération de nombres aléatoires

BUT

Générer des nombres aléatoires par différentes méthodes afin de déterminer la plus efficace.

INTRODUCTION

Les nombres aléatoires générés par un ordinateur ne sont généralement pas aléatoires : on se rend compte au bout d'un certain nombre d'itérations que la série de nombres se répète, ce qui en fait une série de nombres ***pseudo-aléatoires***.

Mais nous nous contentons souvent de ces derniers car :

- il est difficile d'obtenir de « vrais » nombres aléatoires
- dans la plupart des situations les nombres sont suffisament aléatoires pour être utilisés
- les programmes gégérateurs sont particulièrement adaptés à une implantation informatique, donc plus facilement et plus efficacement utilisables.

Les méthodes pseudo-aléatoires sont souvent employées dans des tâches telles que la simulation, la cryptographie, etc. Les générateurs à congruence linéaire sont donc très répandus.

LE PROGRAMME

Dans un soucis de modularité, le code a été écrit et enregistré dans plusieurs scripts Octave. Le programme s'exécute en écrivant la ligne de commande suivante :
>> overlen

Sept choix s'offrent alors à vous, sélectionnez la partie du TP que vous voulez tester :

1) Algorithme K
2) Nombres pseudo-aléatoires par congruence linéaire
3) Nombres pseudo-aléatoires par congruence linéaire avec période
4) 12'000 nombres aléatoires affichés en 2D et 3D
5) Générateur de Stoll-Kirckpatrick
6) Calcul de pi
7) Quitter le programme

I) L'ALGORITHME K

Même si ce n'était pas demandé dans l'énoncé, j'ai d'abord implémenté ***l'algorithme K*** (choix numéro **1**) fourni en exemple dans l'énoncé du TP. Ca m'a permis de la tester, cependant les tests que j'ai effectués avec ne sont *pas très concluants* : cet algorithme est supposé utiliser une séquence aléatoire pour trouver la prochaine valeur de X. Selon la valeur de X que j'entre au début du programme, j'obtiens parfois une série de quelques nombres parfaitement « aléatoires », parfois quelques nombres identiques, parfois un seul nombre (car le nombre d'itérations est également « aléatoire »).

```
X =  975321
X =  975321
X =  975321
X =  975321
X =  975321
X =  975321
X =  975321
```

```
X =  543636
X =  456364
X =  820364
X =  184364
X =  815636
X =    2.6208e+05
X =    2.6208e+05
```

```
Entrez une
X =  62145
-----------
1)Algorithm
```

Après quelques recherches sur internet, il apparaît que les valeurs testées lorsque le générateur a été créé se sont répétées après plus de 7000 valeurs, et la période était supérieure à 3000 valeurs.

II) GÉNÉRATEURS CONGRUENTIELS LINÉAIRES

La forme générale de ces générateurs est la suivante :

$$X_{i+1} = (aX_i + c) \bmod m$$

Nous verrons à chaque étape quelles valeurs donner aux paramètres ***a***, ***c*** et ***m*** pour obtenir le générateur correspondant à nos attentes.

II.1) Cette congruence est de la forme $X_{i+1} = f(X_i)$. À partir du moment où le résultat de X_{i+1} est un modulo (***mod m***), toutes les valeurs à partir de X_1 se situeront obligatoirement entre ***0*** et ***m***. Cela signifie qu'après un certain nombre d'itérations, X_{i+1} sera calculé avec un X_i pour la deuxième fois, et aura donc la même valeur que précédemment. Pareil pour le suivant, et ainsi de suite. Nous obtenons alors une suite de valeurs « aléatoires » périodique.

Pour un exemple simple, prenons les paramètres suivants :

N = 10
a = 1
c = 1
m = 4
$X_0 = 0$

Voici le résultat de mon implémentation (choix numéro ***2***) :

X_0 = 0
Y = 1
Y = 2
Y = 3
Y = 0
Y = 1
Y = 2
Y = 3
Y = 0
Y = 1
Y = 2

Les nombres se suivent car j'ai pris des paramètres simples pour une explication simple. Toutefois, nous constatons que dès que ***$X_{i+1} = 4 \bmod 4 = 0$***, à l'itération suivante, on se retrouve dans le cas de figure ***$X_i = X_0$*** et donc ***$X_{i+1} = X_1$***, quelle que soit la valeur de ***i***, etc.

II.2 & II.3) J'ai implémenté deux générateur congruentiels linéaires : avec et sans période. Commençons par le plus simple, celui sans période (choix numéro **2**).

1- l'utilisateur choisit ses paramètres : nombre d'itérations ***N***, ***A***, ***C***, le modulo ***M***, puis le germe ***X_0***.

2- pendant ***N*** itérations, application de l'équation pour générer les nombres aléatoires avec affichage du résultat.

Pour le générateur avec période (choix numéro **3**), nous avons par la suite :

3- pendant ***N*** itérations, on incrémente la période si la dernière valeur du vecteur ***Y*** est différente de la valeur de ***Y*** courant.

4- on affiche la période.

Dans ma première implémentation, la période était calulée en comparant chaque valeur générée à ***X_1***. Cette méthode avait l'avantage d'utiliser moins de mémoire en évitant de stocker les valeurs de ***Y*** dans un vecteur. Elle fonctionnait toujours très bien car j'utilisais des petits paramètres. Cependant, j'ai tout de même opté pour l'implémentation actuelle car elle affichait des erreurs lorsque les nombres générés devenaient trop grands.

J'ai testé ce générateur avec générateur avec ***m=10*** comme demandé dans la consigne. Afin d'obtenir un cycle maximal, il faut veiller à afficher toutes les valeurs en ***0*** et ***m-1 = 9***.

Voici quelques paramètres qui fonctionnent (peu importe la valeur de X0) :

a=1 ; c=1 **a=1 ; c=3** **a=1 ; c=7** **a=1 ; c=9**

On constate ensuite que si ***a=1*** et ***c*** paire, la période est divisée par 2. Pour finir, l'augmentation de ***a*** fait diminuer la période.

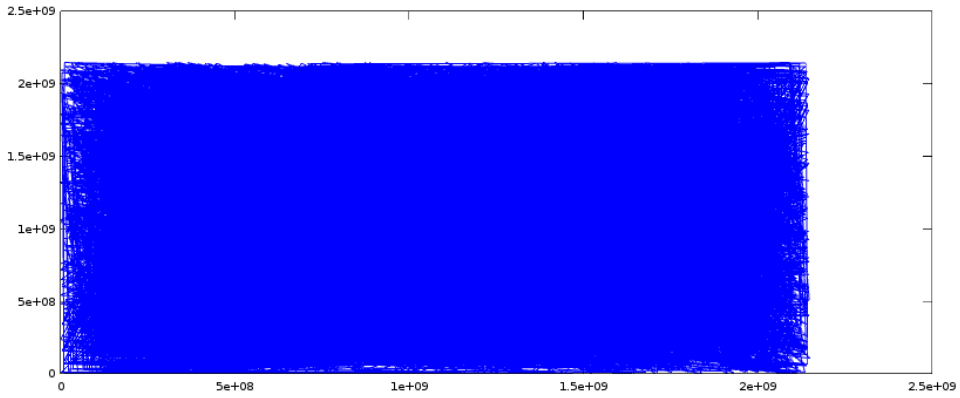
II.4) Le générateur ***RANDU*** développé par ***IBM*** (choix numéro **4**) utilisait les paramètres suivants :

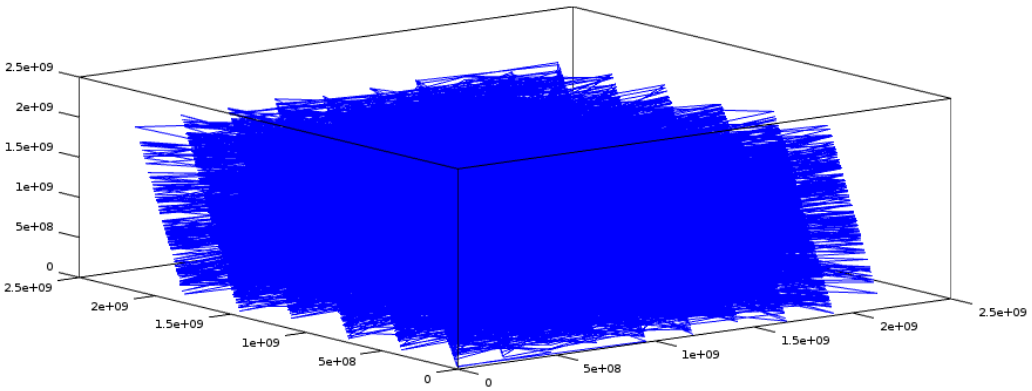
a=65539 ; **c=0** ; **m=2^{31}** ; **X_0=123456789**

La fonction ***visu3D*** est donc identique à l'implémentation du générateur congruentiel linéaire précédemment expliquée. La différence se trouve dans l'affichage des courbes ***2D*** et ***3D*** grâce à la fonction ***affichage(itérations, vecteur, numéro de figure)***.

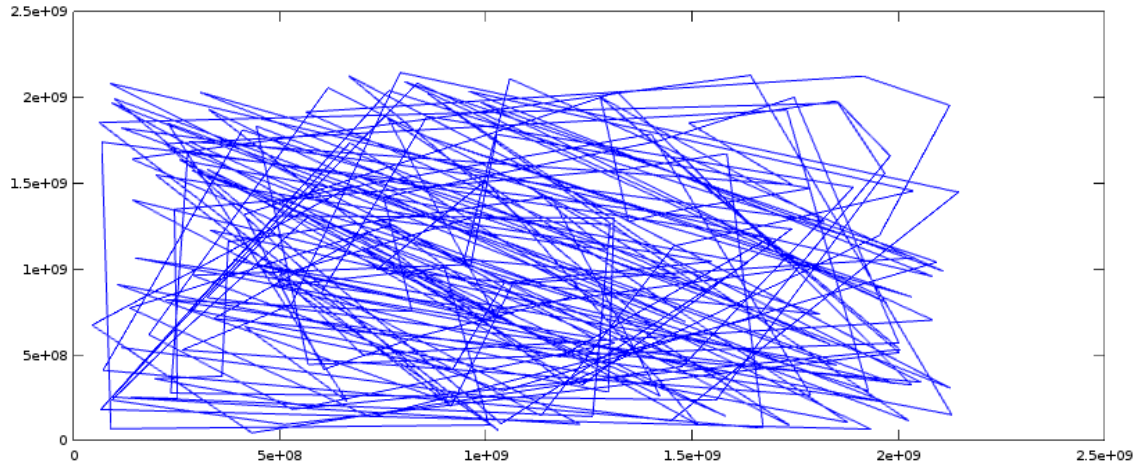
Les vecteurs ***Z*** et ***A*** récupèrent respectivement les valeurs de ***$Y(X_i-1)$*** et ***$Y(X_i-2)$*** avec ***0 < i < N***, puis on enlève les 2 premières valeurs du vecteur Y pour que sa taille corresponde à celles de ***Z*** et ***A***. J'aurais pu éviter, c'est uniquement pour plus de simplicité et pour la clareté du code.

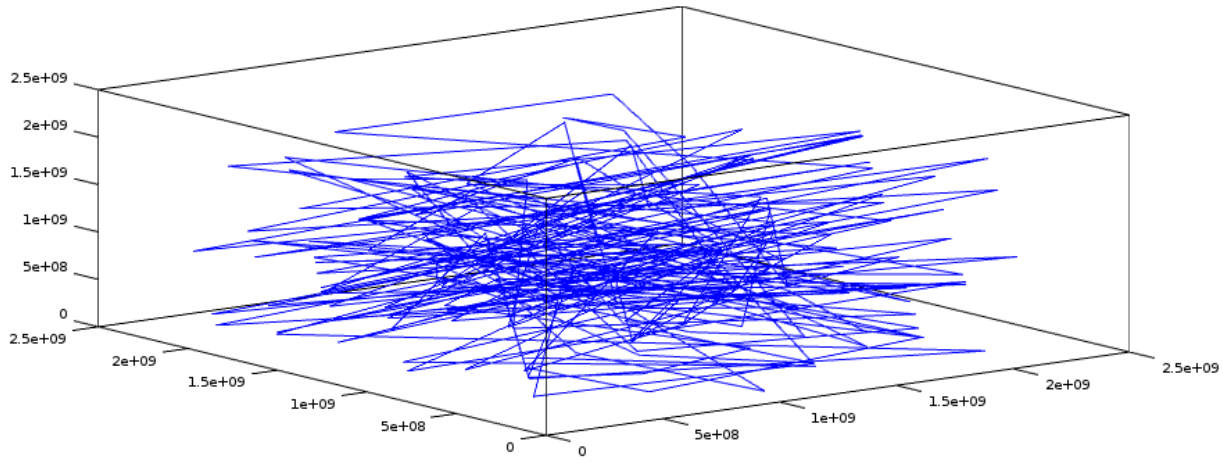
Voici les courbes affichées :





Il est cependant plus facile de constater les résultats du générateur avec un nombre d'itérations réduit. Voici donc le même générateur avec ***N=200*** :





On a déjà un net aperçu de l'efficacité de ce générateur. Pour plus de précisions, testons ces paramètres avec le générateur congruentiel linéaire avec affichage de la période (choix numéro **3**) : on constate que la période est supérieure au nombre d'itérations ***N=12'000*** et que les valeurs affichées sont bien « aléatoires ». Devant la quantité de valeurs générées, on peut dire que le générateur est très efficace.

III) GÉNÉRATEUR DE STOLL-KIRCKPATRICK

Pour commencer, nous devons déjà avoir 250 valeurs « aléatoires » pour appliquer la formule du générateur :

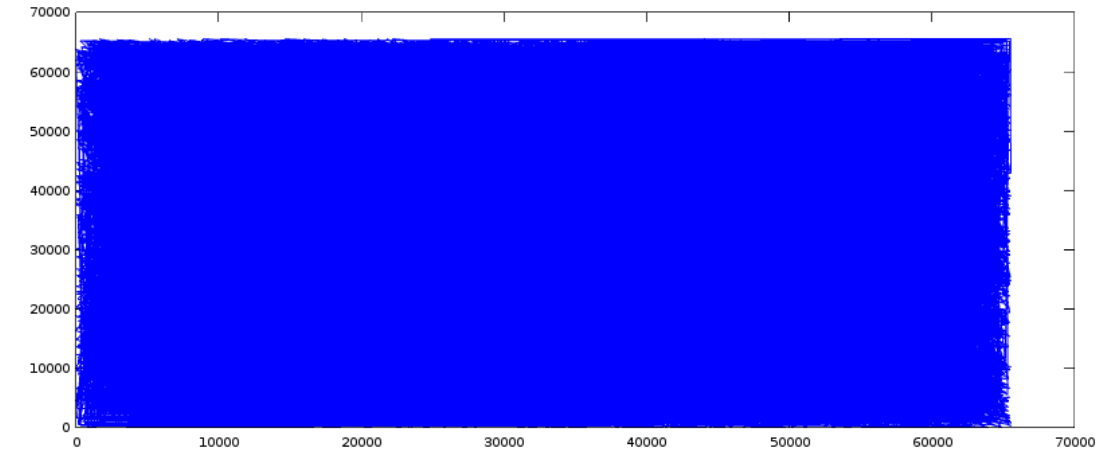
$$X_{i+1} = X_{i-249} \text{ xor } X_{i-102}$$

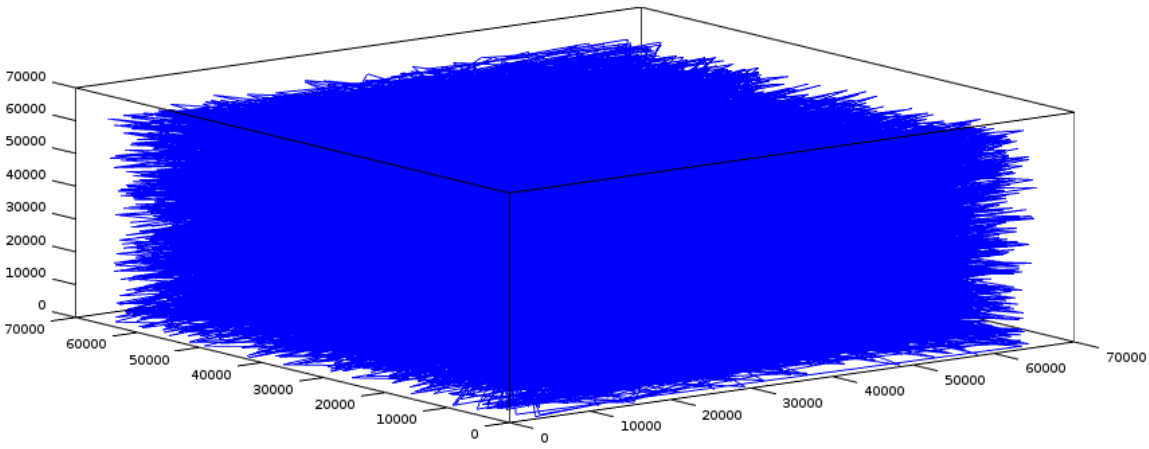
La première étape est donc d'importer les valeurs présentes dans le fichier « aleatoire250.txt » grâce à la fonction ***importdata()*** dans ***Valeurs***. J'ai ajouté

à cela une petite boucle ***if*** sur la longueur de ***Valeurs*** afin de m'assurer qu'il y a suffisament de valeurs dans le fichier pour pouvoir effectuer les calculs.

Avant d'appliquer la formule, nous devons convertir les valeurs en binaire. Pour ce faire, j'ai utilisé la fonction ***de2bi()*** qui renvoie ***16 bits***. Puis, à l'aide de deux boucles ***for***, j'ai appliqué un ***OU-EXCLUSIF*** sur chacun des bits de X_{i-249} et X_{i-102}.

Pour finir, il faut reconvertir les valeurs binaires en décimales avec la fonction ***bi2de()*** afin d'afficher les courbes 2D et 3D ainsi que les moyennes théoriques et pratiques des valeurs. Voici les courbes résultantes :





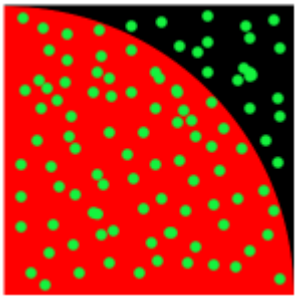
On constate que ce générateur est également très performant, cependant les valeurs générées ne dépassent pas 65535 contrairement au générateur ***RANDU***.

Ce rendu ne contient que **30'000** itérations, contrairement aux **300'000** demandées dans la consigne. Cependant, la lenteur de l'ordinateur avec lequel les tests sont effectués afin d'obtenir ce rendu ne me permet pas de tester le générateur pour un nombre d'itérations dix fois plus élevé.

CALCUL DE π

Afin de calculer les décimales de ***pi***, il faut tirer des points dont les coordonnées aléatoires se situent entre ***0*** et ***1***. Une fois que nous avons obtenu ces points, il faut calculer x^2+y^2 et déterminer si le résultat est inférieur ou égal à ***1***. Pour finir, le rapport entre les résultats inférieurs à ***1*** et le nombre total de points multiplié par 4 nous donne la valeur de ***pi***.

L'explication vient du calcul de ***pi*** effectué sur le quart de cercle suivant :



Les points verts sont des valeurs « aléatoires », ceux situés dans la zone rouge correspondent à $x^2+y^2 \leq 1$. Cela implique que :

$$\frac{points\,dans\,la\,zone\,rouge}{total\,des\,points} = \frac{pi}{4} \quad \text{et donc pi} = 4 * \frac{points\,dans\,la\,zone\,rouge}{total\,des\,points}$$

Pour trouver les valeurs aléatoires, nous utilisons les générateurs précédemment implémentés. Afin d'éviter de modifier tout mon code, j'ai réimplémenté et adapté chaque générateur afin qu'ils tiennent tous dans la fonction de calcul de ***pi***, et ce dans l'unique but de gagner du temps.

Si les nombres tirés sont tous équiprobables après un très grand nombre de tirages, le rapport doit être l'aire du quart de cercle de rayon ***1*** divisée par l'aire du carré, soit :

$$\frac{pi*R^2/4}{1} = 0.785398163397448$$

On observe que plus le nombre de valeurs choisi par l'utilisateur est élevé, plus nous obtenons de décimales de ***pi***. En effet, le nombre de valeurs détermine la précision du rapport.

Voici les résultats des tests que j'ai effectués avec un nombre d'itérations de ***N = 3'000*** :
- le générateur congruentiel avec mes valeurs retourne ***pi = 3,346***
- le générateur ***RANDU*** d'IBM retourne ***pi = 3,72267***
- le générateur de ***Stoll-Kirckpatrick*** retourne ***pi = 3,853***
- le générateur ***rand()*** de Matlab retourne ***pi = 3,176***

N = 10'000 :
- le générateur congruentiel avec mes valeurs retourne ***pi = 2,698***
- le générateur ***RANDU*** d'IBM retourne ***pi = 3,7428***
- le générateur de ***Stoll-Kirckpatrick*** retourne ***pi = 3,8688***
- le générateur ***rand()*** de Matlab retourne ***pi = 3,1492***

La méthode ***rand()*** semblant être la plus prometteuse, j'ai effectué un dernier test avec ***N = 100'000***. Le résultat est ***pi = 3.14376***.

CONCLUSION DU TP

On constate à travers ce TP qu'il y a plusieurs méthodes pour obtenir des nombres pseudo-aléatoires. Cependant, ils n'ont pas tous la même efficacité. En ce qui concerne le calcul de ***pi***, il est impressionnant de constater qu'il a pu être calculé à ***dix mille milliards de décimales*** près tandis qu'un ordinateur tel que ceux de l'HEPIA parvient difficilement à la cinquième décimale.

J'ai effectué ce travail seul et tenté de le rendre le moins « hard-codé » possible afin de réutiliser facilement le code.