Rapport du Lab 1

Analyse du problème :

Dans ce premier TP, nous allons implémenter les algorithmes de quelques générateurs de nombres pseudo-aléatoires. Ce TP a pour vocation de comprendre le codage de ces générateurs et l’importance de leurs qualités pour les simulations stochastiques.

Nous allons nous baser sur les générateurs de nombres pseudo-aléatoires suivant :

* Le carré médian (middle-square) de John Von Neumann
* Générateur congruentiel linéaire

Conception de la solution :

1. Le principe du générateur de Von Neumann est le suivant :

Sur un nombre de 2x chiffres, on élève le nombre au carré puis on prend les x chiffres du milieu comme terme suivant de la séquence pseudo-aléatoire.

Si nous reprenons l’exemple du TP, cela donnera :

0,1234 N0 = 1234 1234 \* 1234 = 01522756

0,5227 N1 = 5227 5227 \* 5227 = 27321529

0,3215 N2 = 3215 ...

1. Générateur à Congruence Linéaire :

Il repose sur la formule de reccurence :

Xn+1 = (a\*X0 + b) mod m

Le choix des paramètres est très important pour la qualité du générateur.

Pour avoir le bon choix des paramètres a, b et m, on utilise la règle de Knuth :

Pour qu’un générateur possède la période égale à m, il faut que :

* c soit premier avec m c’est-à-dire PGCD(c,m)=1
* pour tout nombre premier p divisant m, a-1 soit un multiple de p
* a-1 soit un multiple de 4 si m est un multiple de 4

Réalisation :

L’implémentation de l’algorithme de Von Neumann est la suivante

1. void jvn\_generator(int seed)
2. {
3. int i=0, n0=seed, n1;
4. while(n0!=0) // don't forget to uncomment line i++ ;
5. //for(i=0;i<20;i++)
6. {
7. printf("%.4f \t N%d = %04d \t %04d\*%04d = %08d \n", 0.1\*n0/1000, i, n0, n0, n0, n0\*n0);
8. n1=((n0\*n0)/100)%10000;
9. n0=n1;
10. i++; // comment this line if you use the for loop
11. }
12. }

On fournit en entrée une graine, le nombre initial qui sert à initialiser le générateur. Dans ce code, la boucle while permet d’afficher toutes les séquences générées jusqu’à avoir 0, et la boucle for permet juste d’afficher les i premières séquences.

Pour le générateur LCG le code correspondant est :

void lcg\_intRand(int seed, int a, int c, int m)

{

int x = seed, i;

for(i=0;i<100;i++)

{

printf("%d\t",x);

x=(a\*x + c) % m;

}

printf("\n");

}

Tests :

1. Middle-square

Avec seed = 1234 on a :

0.1234 N0 = 1234 1234\*1234 = 01522756

0.5227 N1 = 5227 5227\*5227 = 27321529

0.3215 N2 = 3215 3215\*3215 = 10336225

0.3362 N3 = 3362 3362\*3362 = 11303044

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

0.0024 N54 = 0024 0024\*0024 = 00000576

0.0005 N55 = 0005 0005\*0005 = 00000025

2- LCG avec les paramètres : x0=5 a=5 c=1 m=16

5 10 3 0 1 6 15 12 13 2 11 8 9 14 7

4 5 10 3 0 1 6 15 12 13 2 11 8 9 14

7 4 5 10 3 0 1 6 15 12 13 2 11 8 9

14 7 4 5 10 3 0 1 6 15 12 13 2 11 8

9 14 7 4 5 10 3 0 1 6 15 12 13 2 11

8 9 14 7 4 5 10 3 0 1 6 15 12 13 2

11 8 9 14 7 4 5 10 3 0

Conclusion :

J’ai eu du mal à implémenter l’algorithme LFSR, je n’ai pas bien compris son principe de fonctionnement.