# GENERATION DE NOMBRES ALEATOIRTES

## Plan

## I- Introduction

## II-Générateur pseudo aléatoire sous scilab

### 1- générateurs congruentiels linéaires

**- définition et caractéristique**

**- choix des paramètres**

### 2- la fonction RAND

- **définition et caractéristique**
-
- **domaine d'application**
- **qualité et rapidité**
- **application avec Rand**

1. **Introduction**

Produire des nombres sans suite apparente est une nécessité dans de nombreux domaines de l'informatique. C'est le cas des jeux, du traitement du signal, de la synthèse d'images pour ne citer que ceux là.

On ne dispose malheureusement pas dans un ordinateur de vraie source de hasard. Tout est synchronisé sur des horloges très stables et les calculs sont fait avec régularité et fiabilité.

Alors comme le matériel est d'un secours bien pauvre dans ce domaine, c'est vers le logiciel qu'il faut se tourner pour produire des séquences de nombres qui ont toutes les apparences du hasard. Mais un logiciel, quelqu'il soit, ne peut prendre les informations nécessaires pour générer le nouveau nombre que dans ce qu'il a déjà calculé (si on lui donne accès à une base de données externe, autant enregistrer une vraie séquence aléatoire produite par un phénomène physique quelconque...).

On appelle traditionnellement une telle suite une séquence pseudo aléatoire (Pseudo Random Sequence for Jack AllGood).

# II. Générateur pseudo aléatoire sous scilab :

## 1-GENERATEURS CONGRUENTIELS LINEAIRES

Il s'agit de l'algorithme le plus utilisé pour produire des nombres aléatoires depuis qu'il a été inventé en 1948 par D.H Lehmer. C'est la suite :

$$x_{n+1} = (a * x_n + b) \bmod c$$

Il est difficile de croire qu'une formule si simple puisse produire des nombres suffisamment au hasard et pourtant ....

Si on désire produire toujours la même séquence (ce qui est pratique à des fins de tests), on rentre toujours la même valeur de x0 (la graine du générateur).
Si on préfère plutôt que la séquence soit toujours différente, on initialise x0 avec une grandeur toujours différente, l'heure système par exemple (ce que fait la fonction randomise () du C).

Dans tous les cas, les nombres de la suite sont compris entre 0 et c-1.

Imaginons à titre d'exemple qu'on choisisse a, b et c tels que :
$x_{n+1} = (25 * x_n + 16) \bmod 256$

Pour x0 = 12, les nombres produits sont : 60, 236, 28, 204, 252, 172 ....
Ils sont tous pairs !!

Pour x0 = 11, les nombres produits sont : 35, 123, 19, 235, 3, 91, 243. Ils sont tous impair ! ! !

Pour x0 = 10, les nombres produits sont : 10 ,10 ,10 ,10 ....cela produit toujours la même valeur

Conclusion : certes La formule est simple mais le choix des trois paramètres doit répondre a certains critères.

Si on choisi b non nul, il est toujours possible d'obtenir une période de longueur c (donc pas de risque de blocage puisqu'on va retrouver tous les nombres entre 0 et c-1 dans la suite). D. Knuth fait la démonstration des critères que doivent remplir a, b et c pour cela :

- b et c doivent être premiers entre eux
- a-1 doit être un multiple de p, pour tout p nombre premier diviseur de c
- a-1 doit être un multiple de 4 si c est un multiple de 4.
- si c 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).

# 2-La fonction rand

Cette fonction utilise le générateur congruentiel linéaire précèdent avec les valeurs de paramètres suivantes :

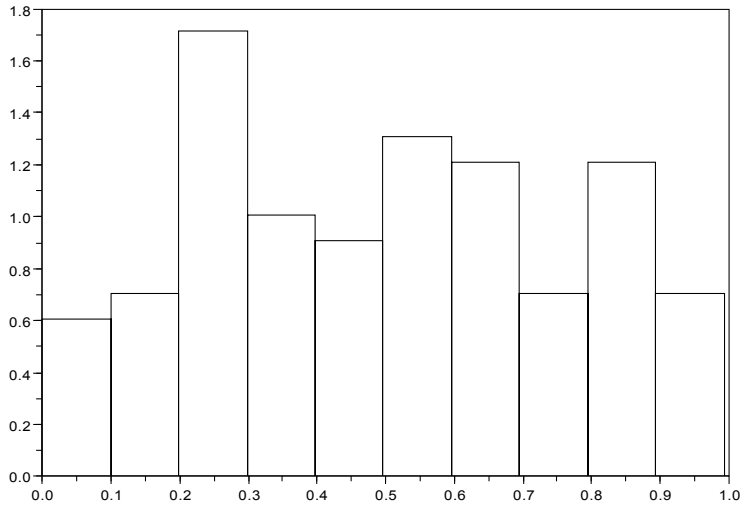
// $c = 2^{31}$
a = 843314861

b = 453816693 //

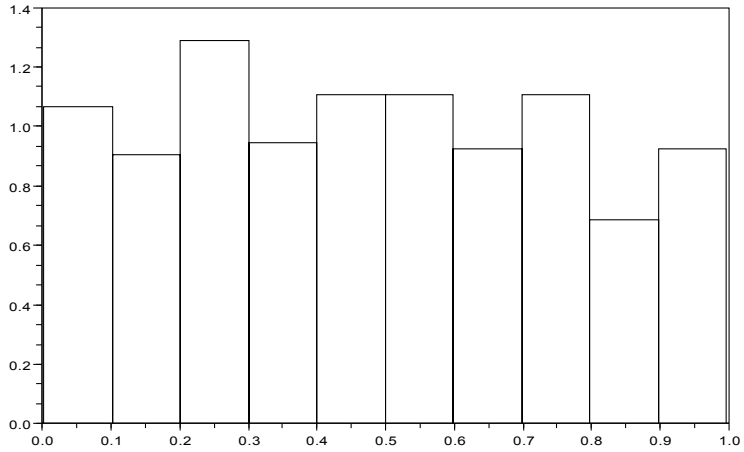
Domaines d'application : cette fonction peut être utilisée pour des applications non critiques comme pour un lecteur mp3, la génération d'un terrain, etc.

On vérifie que cette fonction génère des nombres pseudo –aléatoires :

```
  >x=rand(100,1),
-->mean(x)
         ans   0.4980215
  -->histplot(10,x)
```




```
-->histplot(10,y)
```




Application avec la fonction rand :

Pour effectuer différentes simulations, on peut programmer une fonction prenant le paramètre $N$ et qui effectue $N$ tirages successifs. Cette fonction renvoie alors $X_N$ et $V_N$ :

```
Function [XN, VN]= Urne_de_Polya(N)
  // simulation de N tirages d'une "Urne de Polya" :
  //
  VN = 1 ; V_plus_R = 2 ; XN = 0.5
  for i=1:N
    u = rand()   // tirage d'une boule
    V_plus_R = V_plus_R + 1   // ça fait une boule de plus
    if (u <= XN) then  // on a tire une boule verte : on a XN probabilité
                       //de tomber sur une boule verte (et 1 - XN sur une
                       //boule rouge)

      VN = VN + 1
    end
    XN = VN / V_plus_R // on actualise la proportion de boules vertes
  end
```