# Chapitre VI

# Génération de valeurs pseudo-aléatoires

D. Knuth, " The Art of Computer Programming ", Vol. 2, Chap. 3 : "Random Numbers", pp. 1-177, 1981.

A. M. Law & W. D. Kelton, " Simulation Modeling & Analysis", McGraw-Hill, 1991. (Chap. 7-8).

S. V. Hoover, R. F. Perry, " Simulation A problem-Solving Approach ", Addison-Wesley, 1989. (Chap. 7).

# Pourquoi générer des nombres « au hasard »?

- Théorie de l'échantillonnage
- Analyse numérique (approche de Monte-Carlo)
- Algorithmes probabilistes (Sherwood, Las Vegas, Monte Carlo, etc.)
- Théorie des jeux (jeux sur ordinateur, didacticiels)
- Simulation
- etc.

# Qu’est-ce qu’un nombre « au hasard »?

- 1415913 est-il plus " aléatoire " que 999999 ?
- Les expressions "nombre au hasard " et " nombre aléatoire " sont des abus de langage.
- On doit plutôt chercher en réalité un mécanisme permettant de générer une séquence de nombres qui **paraissent** avoir un comportement aléatoire, même si une séquence déterministe est produite.

Définition : Nombre aléatoire º une v.a. dans [0,1] tel que

$$f_X(x) = \begin{cases} 1 & \text{si } x \in [0, 1] \\ 0 & \text{sinon} \end{cases}$$

**Note :** $X \sim U_{[0,1]} \Rightarrow (B-A)\,X + A \sim U_{[A,B]}$ .

- C'est impossible de générer des variées aléatoires continus car l'ensemble des valeurs possibles générées par un ordinateur est fini.
- Mais si cet ensemble est grand, on peut approximer l'ensemble continu.

# Qu'est-ce qu'un bon générateur?

- **Objectifs :**
  - générateur statistiquement équivalent à échantillonner une population qui suit une loi uniforme.
  - capacité de reproduire des suites de nombres aléatoires déjà générés.
  - génération efficace (temps de calcul).

- On voudrait générer $U_1$, $U_2$, ... tels que les $U_i$ ont l'air de suivre une loi $U_{[0,1]}$ et être indépendants.

- Les $U_i$ devraient être répartis à peu près uniformément dans l'intervalle [0,1].
  Note : Construire un tableau de fréquences dans [0,1].

- Les points ($U_i$ , $U_{i+1}$) devraient être répartis à peu près uniformément dans (0,1) x (0,1).

# Qu'est-ce qu'un bon générateur?

- Les points ($U_i$ , $U_{i+1}$, ..., $U_{i+k-1}$) devraient être répartis à peu près uniformément dans l'hypercube à k dimensions $(0,1)^k$.
- Idéalement, ces propriétés devraient être satisfaites $\forall k \geq 1$. MAIS
  Un générateur sera considéré comme satisfaisant s'il satisfait ces conditions pour k = 1, 2, ..., N, où N est assez grand.
- S'il y a des cycles dans la séquence des $U_i$, ils doivent être de période tellement longue qu'on ne puisse pas l'atteindre en pratique.

**Comment vérifier ces propriétés ?**

- Tests théoriques (analyse des générateurs à partir des paramètres numériques sans générer les $U_i$).
- Tests statistiques (tests basés sur les $U_i$ générés).

**Autres points à considérer**

- Efficacité du générateur (vitesse de calculs).
- Espace mémoire exigé - Caractère reproductible.

# Principales classes de méthodes de génération de nombres aléatoires

1°) **Mécanismes physiques**

- pile ou face, boules dans une urne, …
- on jumelle à l'ordinateur un dispositif capable d'enregistrer les résultats d'un processus aléatoire physique
  (générateur d'impulsions électriques, source radio-active).
- derniers bits de l'horloge d'un ordinateur.
- etc.

**Désavantages :**

- Méthode coûteuse et impraticable.
- Incapable de reproduire la même suite de nombres alélatoires aux différentes sessions de simulation.

# Intérêt de pouvoir reproduire plusieurs fois une même séquence

- afin de pouvoir utiliser les techniques de réduction de la variance en simulation

  ex : Comparer différentes stratégies sous les mêmes conditions(aléatoires) extérieures.

- pour faciliter la mise au point et la vérification des programmes qui utilisent ces nombres.
- afin de générer le même échantillon dans une population.
- intérêt en analyse numérique.
- etc.

# 2°) Les bons générateurs utilisés en pratique

Ce sont des fonctions déterministes !

- Ce sont des transformations séquentielles sur un ensemble de nombres choisis arbitrairement $\Rightarrow$ suite de nombres pseudo-aléatoires.
- **Avantages :** a) méthodes économiques.
  b) la suite générée est reproductible.
- **Désavantages :** Présence d'auto-corrélation entre les éléments de la suite, difficile à éliminer.

Soit S : espace d'états (entiers, ensemble fini, vecteurs, ...), un générateur est défini par :

$$f : S \rightarrow S$$

et l'état s évolue selon : $s_i := f(s_{i-1})$, $i = 1, 2, 3, \ldots$

L'état initial $s_0$ s'appelle le germe.

# 2°) Les bons générateurs utilisés en pratique

Transformation en U [0,1] par g : $S \rightarrow [0,1]$

$U_i \mapsto g(s_i)$

**Période** du générateur : Min p

$\exists n_o > 0, \forall i \geq n_o, s_{i+p} = s_i$.

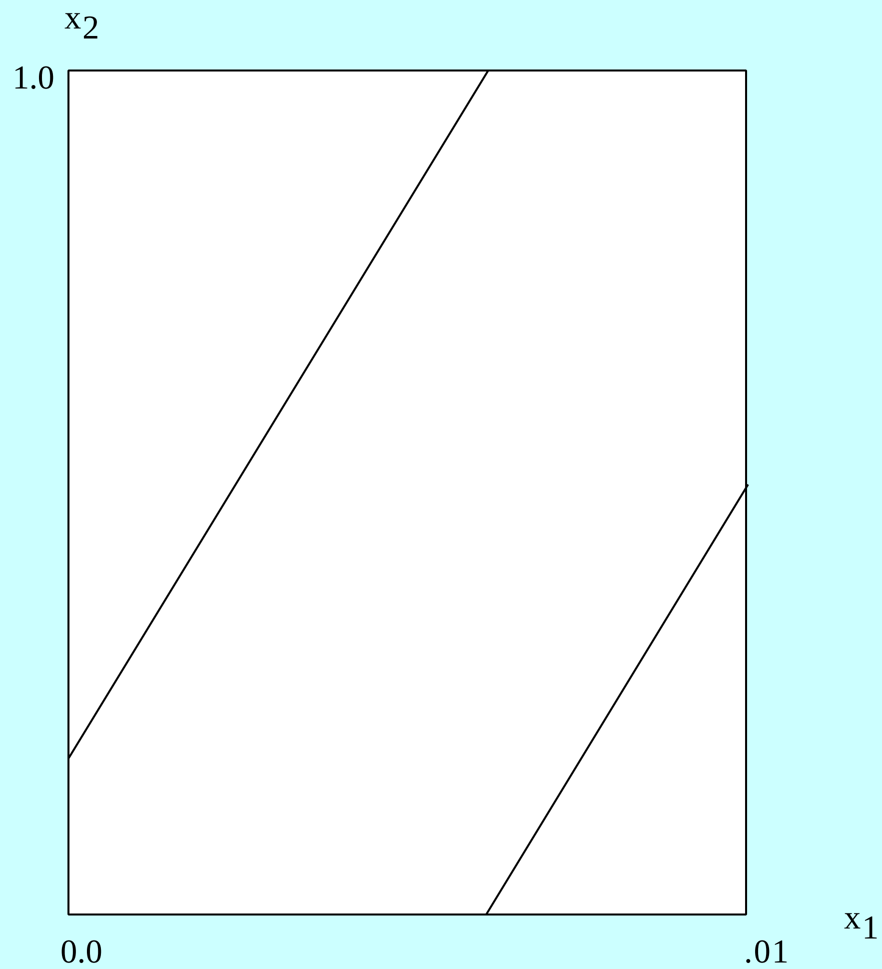
- Bon générateur $\equiv$ bon choix de f, g et de $s_o$.

a) bon support théorique + tests empiriques.

b) implantation : efficacité (vitesse et mémoire), portabilité.

c) possibilité de subdiviser la séquence en sous-séquences disjointes.

# Comportement du générateur de Turbo-Pascal (version 3.0)



# Comportement d’un générateur

D. Knuth, 1981, vol. 2, p. 4-5.

- Il propose une fonction f tellement compliquée que personne ne pouvait comprendre le programme à moins qu'il soit amplement documenté.
- Il s'est ensuite aperçu lors de l'implantation que

$$f\ (6065038420) = 6065038420$$
$$\Downarrow$$
$$p = 1$$

Morale : Il ne faut pas choisir au hasard un générateur de nombres pseudo-aléatoires. Il faut s'appuyer sur des principes théoriques.

# Principaux types de générateurs
# 1. Méthode du carré médian

L'état $s_i$ est un entier.

Pour obtenir $s_i$, on met $s_{i-1}$ au carré et on prend les chiffres du milieu.

Exemple : $s_{i-1} = 8234$ $(s_{i-1})^2 = 67798756$

$s_i = 7987$

Défauts majeurs :

- Période habituellement très courte.
- On risque même de tomber (c'est souvent le cas) sur un état *absorbant* (par exemple 0) !

$\Rightarrow$ C'est donc une méthode à <u>oublier</u>.

# Principaux types de générateurs
# 2. Méthode de Fibonacci

$s_i = (s_{i-1} + s_{i-2}) \text{ MOD } m$

**Défaut** : forte <u>corrélation</u> entre les valeurs successives.

# Principaux types de générateurs
# 3. Générateurs à congruence linéaire (GCL)

C'est ce qui est le plus utilisé et le mieux connu.
L'état s est un entier entre 0 et m-1, et on a :

$$f(s) = (as + c) \text{ MOD } m;$$
$$g(s) = s / m;$$

| | | |
|---|---|---|
| m : | modulo $(m > 0)$ | |
| a : | multiplicateur | $(0 < a < m)$ |
| c : | incrément | $(0 \leq c < m)$ |

L'évolution se fait donc selon

$$s_i := (a\, s_{i-1} + c) \text{ MOD } m$$

et on peut sauter directement de $s_i$ à $s_{i+j}$ :

$$s_{i+j} = [a^j s_i + (a^j - 1)\, c/(a-1)] \text{ MOD } m$$

# Principaux types de générateurs
# 3. Générateurs à congruence linéaire (GCL)

**Note** : On sait que

$$(x + y) \text{ MOD } m = ((x \text{ MOD } m) + (y \text{ MOD } m)) \text{ MOD } m$$

$$(x * y) \text{ MOD } m = ((x \text{ MOD } m) * (y \text{ MOD } m)) \text{ MOD } m$$

- Les $U_i = s_i / m$ ne prennent pas leurs valeurs partout dans le continuum [0,1], mais seulement aux points rationnels 0, 1/m, 2/m, ..., (m - 1)/m.

  On a ainsi Prob $[(k-1)/m < U_i < k/m] = 0$.

- La période p ne peut pas dépasser m.

- Lorsque p = m, le GCL est dit de période maximale.

  Dans ce cas, toutes les valeurs entières de 0 à m -1 sont prises exactement une fois par cycle. Cela garantit l'uniformité.

# Principaux types de générateurs
# 3. Générateurs à congruence linéaire (GCL)

- Un GCL est dit

  **mixte** lorsque $c > 0$ et

  **multiplicatif** lorsque $c = 0$.

**THÉORÈME**

Un GCL **mixte** ($c > 0$) est de **période maximale** ($p = m$) $\Leftrightarrow$

(a) m et c sont relativement premiers; (c et m n'ont pas de facteurs communs)

b) tout nombre premier qui divise m divise aussi a-1;

(c) si 4 divise m, alors 4 divise a-1.

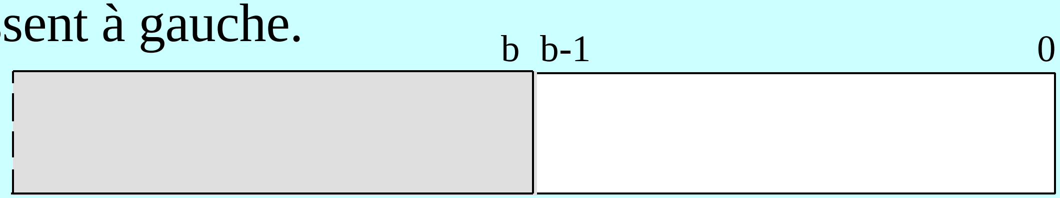
# Exemple

- $m = 8$, $a = 5$, $c = 3$, $s_o = 7$

  $s_i = (5s_{i-1} + 3) \text{ MOD } 8$

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $s_i$ | 7 | 6 | 1 | 0 | 3 | 2 | 5 | 4 | … |
| $U_i$ | 7/8 | 3/4 | 1/8 | 0 | 3/8 | 1/4 | 5/8 | 1/2 | ... |

- En pratique, on essaie de choisir **m le plus grand possible**.
- Souvent, on prend $m = 2^b$ où b est le nombre de bits utilisés pour représenter un entier positif dans la machine.
- L'opération modulo devient alors très facile : il suffit de laisser tomber les bits qui dépassent à gauche.



# Exemples (suite)

- Pour avoir $p = m$, il suffit alors de prendre $c$ impair et $a$ -1 multiple de 4.
- **Exemples :**

  $s_i := (69069\ s_{i-1} + 1)\ \mathrm{MOD}\ 2^{32}$ (<u>VAX</u>)

  $s_i := (314159269\ s_{i-1} + 453806245)\ \mathrm{MOD}\ 2^{31}$

- **Attention :**

  une longue période est une condition *nécessaire*, *mais pas suffisante* pour un bon générateur.

# Principaux types de générateurs
# 4. GCL multiplicatifs(GCLM): c=0

- Les GCLM sont les plus utilisés en pratique.

  Avantage : pas d'addition à effectuer.

- Période maximale : p = m-1 (s = 0 est exclus).
- **Théorème :**

Un GCLM (c= 0) est de période maximale (p = m-1) si (conditions suffisantes):

(a) m est premier et

(b) m-1 est le plus petit entier *k* tel que $(a^k - 1) \text{ MOD } m = 0$.

- **Exemples :**

$s_i := 16807\, s_{i-1}$ MOD $(2^{31} - 1)$ IBM, APL, IMSL, SIMPascal, ...)

$s_i := 630360016\, s_{i-1}$ MOD $(2^{31} - 1)$ (SIMSCRIPT II.5, ... )

Heureux hasard : $2^{31}-1 = 2147483647$ est premier !

# Principaux types de générateurs
# 4. GCL multiplicatifs(GCLM): c=0

- Si le modulo m est une puissance de 2, on ne peut pas atteindre la période "maximale".
- THÉORÈME : Si $c = 0$ et $m = 2^b$, alors $p \leq m/4$.
- **Exemple :** $s_i := 65539\, s_{i-1} \text{ MOD } 2^{31}$

  Ce générateur, appelé RANDU, fut longtemps utilisé sur les systèmes IBM.
  Il l'est encore à certains endroits. Il est malheureusement très très mauvais.

- Il est recommandé, pour un GCLM, de choisir m premier.

On choisit habituellement le plus grand nombre premier représentable sur la machine.
On aura évidemment $a < m$ et $s < m$.

Mais lors du calcul de f(s), le produit as dépasse en général le plus grand entier représentable sur la machine. Même en double précision, ça peut parfois déborder.

Que faire ? Il faut implanter la multiplication en " logiciel ". On essaie en général d'obtenir une implantation portable dans un langage de haut niveau.

# Tests statistiques sur les générateurs

- Il s'agit d'utiliser des tests d'hypothèse afin de tester

  $H_o$ : Notre générateur est bon.

  Il s'agit de tester si les $U_i$ ont vraiment l'air d'être des v.a. i.i.d. $U_{[0,1]}$.

- N'utilisez jamais un générateur à des fins sérieuses à moins de le tester vous-même, ou de savoir que c'est l'un des bons générateurs testés et suggérés dans la littérature.

- Les générateurs fournis sur les systèmes sont souvent très mauvais.

# A) Indicateur de la valeur du générateur

$X \in U[0,1]$ $\Rightarrow$ $E[X] = 0.5$, $Var(X) = 1/12$ et $E[X_1 X_2] = E[X_1] = 0.25$

où $X_1$ et $X_2$ sont indépendantes.

Lorsqu'un grand échantillon de nombres aléatoires a été généré, des estimés précis de la moyenne, la variance, le facteur d'auto-corrélation de la population peuvent être obtenus et comparés aux valeurs théoriques.

<u>Exemple</u> [Hoover, Perry, 1989] :

$s_i = [25173\, s_{i-1} + 13849] \bmod 65536$ et $s_o = 23311$

10,000 nombres ont été générés.

On a obtenu $\overline{X} = 0.5019$ et $1 / 10000\ \sum_{i=1, 2, \ldots 9999} x_i\, x_{i+1} = 0.25$

$s^2 = 0.08412$

# B) Tests d'indépendance entre les éléments générés

**1°)** **Test basé sur un coefficient de corrélation**

Soit $$C = \frac{n \sum_{i=1, 2, \ldots n-1} x_i\, x_{i+1} - [\sum_{i=1, 2, \ldots n} x_i\,]^2}{n \sum_{i=1, 2, \ldots n} x_i^2 - [\sum_{i=1, 2, \ldots n} x_i\,]^2}$$

on peut montrer qu'en présence d'une population normale, où les $\{x_i\}$ sont indépendants,

$$E\,[C] = -\,1 / (n - 1) \qquad \text{et} \qquad \sigma^2_C = \{n(n-3) / (n+1)\}^{1/2} / (n-1)$$

[Knuth, 69, Vol. 2] Pour des nombres aléatoires uniformes, cette approximation est bonne lorsque n est élevé.

Exemple précédent :

$C = -0.002741$ ce qui donne $Z = [C\text{-}E\,[C]\,] / \sigma_C = -0.274$

Si $Z \in N(0,1)$, Prob $(|Z| \leq 1.96) = 0.95$

$\therefore$ $-\,0.274$ n'est pas significatif.

# B) Tests d'indépendance entre les éléments générés

**2º)** **Tests des séquences croissantes (Run Test)**

- Ce test est basé sur le nombre de séquences croissantes et décroissantes.

Soit R = le nombre de séquences dans l'échantillon,

$$E[R] = (2n-1) / 3 \text{ et } Var(R) = (16n - 29) / 90$$

et pour n élevé (grand échantillon), R est approximativement normale.

| Ex : | 0.12 | .14 | .65 | .18 | .33 | .77 | .89 | .72 |
|---|---|---|---|---|---|---|---|---|
| | .66 | .43 | .70 | .81 | .94 | .98 | .03 | |

$\Rightarrow$ ++ - +++ - + - ++++ - $\Rightarrow$ **R = 8**

**Exemple précédent :** R = 6602 séquences

E(R) = 6666.3 Var(R) = 1777.5 $Z = (6602 - 6666.3) / \sqrt{1777.5} = -1.52$

\ **Non significatif**

# B) Tests d'indépendance entre les éléments générés

- On examine les $\{U_i\}$ pour trouver les sous-séquences croissantes.

ex : .855 .108 .226 .032 .132 .056 .545 .642

.103 .108 .201 .809 .901 .151

\- On compte le nombre de sous-séquences de chaque longueur :

$$r_i = \begin{cases} \text{\# sous-séquences de longueur } i, & i = 1, 2, 3, 4, 5 \\ \text{\# sous-séquences de longueur} \geq 6, & i = 6. \end{cases}$$

ex. : $r_1 = 2$, $r_2 = 2$, $r_3 = 1$, $r_4 = 0$, $r_5 = 1$, $r_6 = 0$.

# B) Tests d'indépendance entre les éléments générés

- Il s'agit de calculer

$$R = \sum_{i=1,\,2,\,\ldots\,6} \sum_{j=1,\,2,\,\ldots\,6} a_{ij}\,(r_i - nb_i)\,(r_j - nb_j) / n$$

Le calcul des constantes $a_{ij}$ et $b_i$ est effectué dans Knuth, 1981, pp.65-68, Vol. 2.

- Pour une valeur de n élevée, ($\geq$ 4000)

$$R \approx \chi^2\,(6) \text{ avec 6 degrés de liberté;}$$

sous $H_o$ : les $U_i$ sont des v.a. I.I.D.

# C) Test pour vérifier l'adéquation entre l'échantillon généré et une population uniforme

- Il n'est pas suffisant de rechercher un générateur de période maximale.
- Habituellement, on ne génère pas une séquence complète sans cycle dont la longueur est égale à celle de la période.
- Il est important de vérifier que l'échantillon $U_1, U_2, \ldots, U_n$ est réparti uniformément dans l'intervalle [0,1].

## 1°) <u>Test de Kolmogorov-Smirnov.</u>

- Pour tester si les $U_i$ suivent une loi $U_{[0,1]}$, on tire un échantillon $U_1, U_2, \ldots U_n$ et on compare la fonction de répartition empirique avec celle d'une $U_{[0,1]}$.
- Le test statistique est : $D = \max \{D^+, D^-\}$ où

$$D^+ = \max_{1 \leq i \leq n} \{i / n - F^*(U_i)\}$$

$$D^- = \max_{1 \leq i \leq n} \{F^*(U_i) - (i-1) / n\}$$

# C) Test pour vérifier l'adéquation entre l'échantillon généré et une population uniforme

Lorsque n est élevé, nous avons des approximations des valeurs critiques :

$D_{.05} = 1.36 / \sqrt{n}$ et $D_{.01} = 1.63 / \sqrt{n}$ .

Exemple précédent :

n = 10,000 et D = 0.007612

$D_{.05, n = 10,000} = 0.0136$

∴ Test non significatif : on ne peut pas conclure que le générateur ne correspond pas à celui d'une distribution uniforme.

# C) Test pour vérifier l'adéquation entre l'échantillon généré et une population uniforme

## 2º) Test du Khi-deux

### Test du Khi-deux à 1 dimension

- On divise l'intervalle [0,1] en k sous-intervalles égaux.

- On s'attend à obtenir n/k observations de $U_i$ dans chaque sous-intervalle.

- Le test statistique est :

  $(k / n) \sum_{j=1, 2, \ldots k} (f_j - n / k)^2$ où $f_j$ = # observations dans le $j^{ième}$ intervalle.

  Cela suit approximativement une loi $\chi_{k-1}^2$ degrés de liberté sous l'hypothèse d'une répartition uniforme.

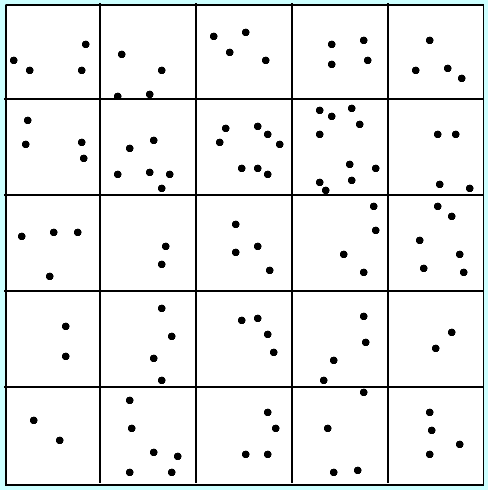
- On rejette $H_o$ si $\chi_{k-1}^2$ prend une valeur trop grande.

# C) Test pour vérifier l'adéquation entre l'échantillon généré et une population uniforme

Test du Khi-deux à d dimensions

- On divise [0,1] en k sous-intervalles égaux, ce qui divise l'hypercube $[0,1]^d$ en $k^d$ sous-hypercubes égaux.

- Exemple : d= 2, k = 5



# C) Test pour vérifier l'adéquation entre l'échantillon généré et une population uniforme

- On regroupe les $U_i$ en d-tuples :

$$(U_1, \ldots, U_d), (U_{d+1}, \ldots, U_{2d}), \ldots$$

- On compte le nombre de d-tuples observés dans chaque sous-hypercube $(j_1, j_2, \ldots, j_d)$,

  soit $f_{j_1, j_2, \ldots, j_d}$

- On calcule $(k^d / n) \sum_{j_1=1, 2, \ldots k} \ldots \sum_{j_d=1, 2, \ldots k} (f_{j_1, j_2, \ldots, j_d} - n / k^d)^2$.
- Cela suit approximativement une loi $\chi_{k^d-1}^2$ sous $H_o$.
- On rejette $H_o$ si $\chi_{k^d-1}^2$ est trop grand.
- Si $k^d$-1 est très grand, on peut approximer par une loi normale.
- Ce test permet de tester si les d-tuples sont répartis uniformément dans $[0,1]^d$.
- Toutefois , la puissance de ce test est faible.

# C) Test pour vérifier l'adéquation entre l'échantillon généré et une population uniforme

Tests répétitifs

- Pour chacun des tests ci-haut, on peut répéter le test plusieurs fois, sur des sections différentes de la séquence $\{U_i\}$.
- On obtient ainsi une série de valeurs; on peut tester si celles-ci sont réparties selon la loi à laquelle on s'attend sous $H_o$, en utilisant par exemple un test de Kolmogorov-Smirnov.

# C) Test pour vérifier l'adéquation entre l'échantillon généré et une population uniforme

Tests répétitifs

**Exemple :**

$$R = \sum_{i=1,2,\ldots 6} \sum_{j=1,2,\ldots 6} a_{ij} (r_i - nb_i)(r_j - nb_j)/n \quad \rightarrow \quad \chi^2_6$$

- Calculer R pour $U_1, \ldots, U_n$
- Calculer R pour $U_{n+1}, \ldots, U_{2n}$

...

- Calculer R pour $U_{99n+1}, \ldots U_{100n}$

- Calculer la fonction de répartition empirique de ces 100 valeurs de R,
- Comparer cette fonction de répartition empirique avec celle d'une loi $\chi^2_6$ à l'aide du test de Kolmogorov-Smirnov.
- On obtient ainsi un test beaucoup plus puissant mais plus coûteux.

# C) Test pour vérifier l'adéquation entre l'échantillon généré et une population uniforme

Conclusion :

- Il reste beaucoup de recherche à faire dans ce domaine.
- Plusieurs autres tests statistiques n'ont pas été présentés.
- Les tests théoriques ne sont pas présentés à cause de la complexité mathématique.

[Knuth, 81, pp. 75-110; Fishman, 78, pp. 358-371].

# Génération de valeurs pseudo-aléatoires selon une loi ≠ U[0,1]

- On utilise en général des valeurs produites par un générateur $U_{[0,1]}$ que l'on transforme.
- Comment choisir un algorithme de transformation ?

On cherche une méthode:

exacte : les résultats sont justes.

efficace : a) mémoire requise

b) temps d'exécution

- complexité de l'algorithme (difficulté d'implantation et de compréhension).
- robustesse de l'algorithme (exact et efficace quel que soit les valeurs des paramètres).

# Principales classes de méthodes
# A) Transformation inverse

- On veut générer la valeur d'une v.a. X dont la fonction de répartition est F.
- Soit U : $U_{[0,1]}$ , si F est inversible ($F^{-1}$ existe) alors

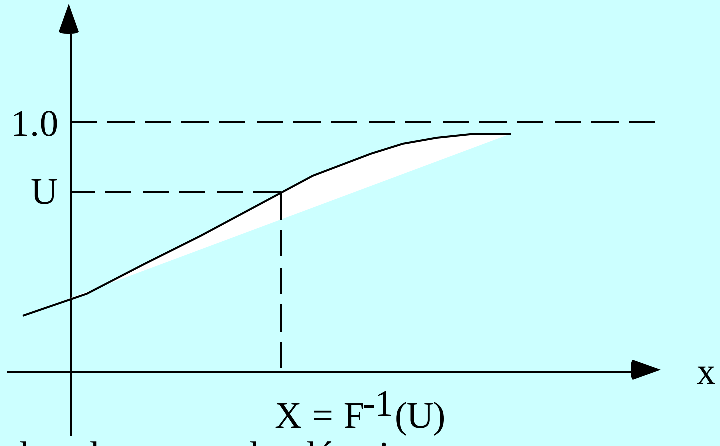
$$P\,(F^{-1}(U) \leq x) = P\,(U \leq F(x)) = F(x) = P(X \leq x).$$

$\Rightarrow$ $F^{-1}(U)$ a la même distribution que X.

$\therefore F(X) \sim U_{[0,1]}$.

- On procède alors comme suit :

1. générer U : $U_{[0,1]}$
2. poser $X = F^{-1}(U)$



# Principales classes de méthodes
# A) Transformation inverse

## Exemple I

Soit $f_X(x) = \begin{cases} 2x & 0 \le x \le 1 \\ 0 & \text{sinon} \end{cases}$

$$F_X(x) = \begin{cases} 0 & x \le 0 \\ x^2 & 0 \le x \le 1 \\ 1 & x \ge 1 \end{cases}$$

$x = F^{-1}(u) = \sqrt{u}$

# Principales classes de méthodes
# A) Transformation inverse

## Exemple II

Soit $X \sim \text{Exp}(\lambda)$,

$$F(X) = 1 - e^{-\lambda x}, \; x > 0$$

$$u = 1 - e^{-\lambda x} \Leftrightarrow \quad x = -\ln(1-u)/\lambda$$

$\Rightarrow$ Générer U et calculer X = -ln (1-U) * Moyenne.

# Principales classes de méthodes
# A) Transformation inverse

## Exemple III

Soit $X \sim$ Weibull $(\alpha, \beta)$,

$$F(x) = 1 - e^{-(x/\beta)^{\alpha}}, \quad x > 0$$

$$F^{-1}(u) = \beta\,[-\ln(1 - u)]^{1/\alpha}$$

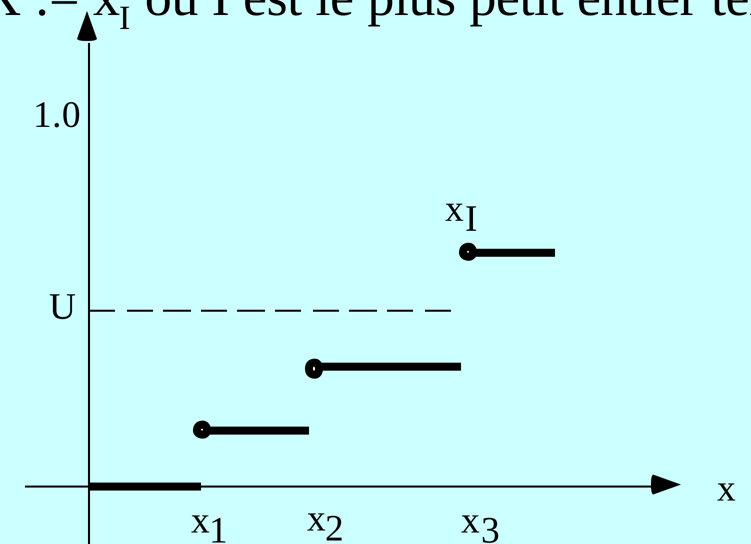
# Principales classes de méthodes A) Transformation inverse

## Exemple IV (cas discret)

Valeurs possibles : $x_1 < x_2 < x_3 < \ldots$ et $F(x) = P(X \leq x) = \; ? \; p(x_i)$
$\{i | \; x_i \leq x\}$

1°) Générer U : $U_{[0,1]}$

2°) Poser $X := x_I$ où I est le plus petit entier tel que $F(x_I) \geq U$.



# Principales classes de méthodes
# A) Transformation inverse

## Avantages et inconvénients de la méthode

- Ce n'est pas toujours facile d'évaluer $F^{-1}(U)$.
- Dans le pire cas, on utilise des tables avec interpolation.
- Méthode parfois lente et coûteuse.
- Nous avons besoin d'un seul U pour chaque X :

  cela facilite la synchronisation des valeurs aléatoires lorsqu'on fait plusieurs répétitions avec un même germe pour comparer différents systèmes ou différentes stratégies.

$\therefore$ C'est la méthode qu'il faut privilégier.

# B) Méthode d'acceptation / rejet [Von Neumann, 1951]

- Méthode à utiliser lorsque les autres méthodes ne fonctionnent pas ou ne sont pas efficaces.
- Soit $f_X(x)$ : la fonction de densité de X,

  $t(x)$ : une fonction qui majore $f_X(x)$ i.e. $t(x) \geq f_X(x)$.

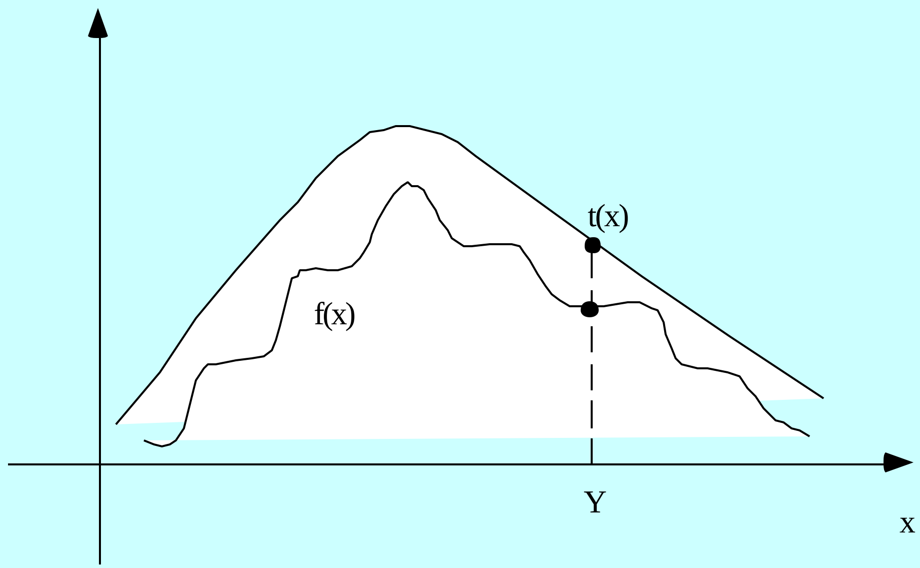
  $$\Rightarrow \quad r(x) = \frac{t(x)}{\int_{-\infty}^{\infty} t(s)\, ds} \quad \text{est une fonction de densité.}$$

- On doit choisir t(x) afin qu'il soit facile de générer des v.a. selon la loi de r(x).
- Méthode : 1°) Générer Y selon la densité r(x).

  2°) Générer U : U [0,1] indépendant de Y

  3°) Si $U \leq f(Y) / t(Y)$, poser X: = Y sinon retourner à 1.

# B) Méthode d'acceptation / rejet [Von Neumann, 1951]




**Choix de t(x) :**

- Cela doit être facile de générer des v.a. selon r (x).
- La surface entre $f_X(x)$ et t(x) doit être petite, afin de minimiser le nombre de rejets.

# B) Méthode d'acceptation / rejet [Von Neumann, 1951]

**Exemple**

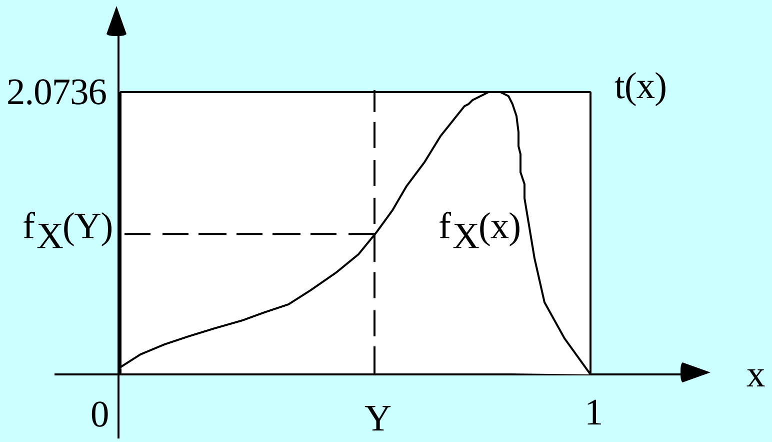
$f_X(x) = 60\, x^3\, (1-x)^2,\ 0 < x < 1$

$F_X(x)$ est très difficile à inverser,

mais,

$f_X(x) \leq 2.0736 = t(x),\ 0 < x < 1.$

On choisit alors pour r(x), la densité d'une $U_{[0,1]}$.




**REPEAT**

Générer Y : U [0,1]

Générer U : U [0,1]

**UNTIL** $2.0736 * U \leq f_X(Y)$;

X : = Y;

# C) Méthode de composition

- Lorsque F est une combinaison linéaire convexe de plusieurs distributions :

$$F(x) = \; ?\,_j p_j\, F_j(x), \qquad ?\,_j\, p_j = 1.$$

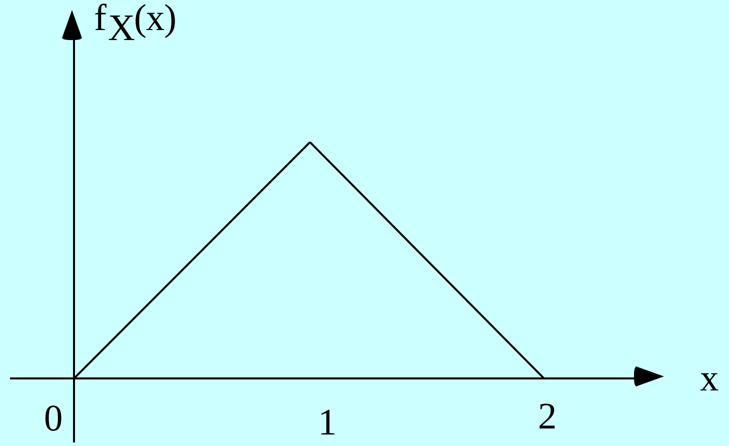
- <u>Méthode</u> :

  1) Générer J avec P (J = j) = $p_j$
  2) Générer X selon la loi spécifiée par $F_j$

- **<u>Exemple</u> :** v.a. hyperexponentielles (combinaison de v.a. exponentielles).

# C) Méthode de composition

Exemple : $f_X(x) = \begin{cases} x & 0 \leq x \leq 1 \\ 2-x & 1 \leq x \leq 2 \\ 0 & \text{sinon} \end{cases}$




Posons $f_X(x) = 0.5\, f_1(x) + 0.5\, f_2(x)$

où $f_1(x) = 2x,\ 0 \leq x \leq 1$ et $f_2(x) = 4\text{-}2x,\ 1 \leq x \leq 2$.

$\Rightarrow$ $F_1(x) = x^2,\ 0 \leq x \leq 1$ $\qquad F_2(x) = 4x - x^2 - 3,\ \ 1 \leq x \leq 2$

$F_1^{-1}(\mu) = \sqrt{\mu}$ $\qquad F_2^{-1}(\mu) = 2 - \sqrt{(1-\mu)}$

# D) Méthode de convolution

- Une v.a. peut être exprimée comme une combinaison linéaire de k autres v.a. :

$$x = \sum_{i=1,\,2,\,\ldots\,k} b_i x_i$$

- Méthode :

  1) Générer k nombres aléatoires indépendants suivant les lois spécifiques des v.a. $x_i$.

  2) Calculer x.

- Exemple : Loi hypoexponentielle ≡ somme de v.a. exponentielles.
  Loi d'Erlang

Note : Exige k valeurs de U dans U[0,1] ⇒ coût élevé.

# D) Méthode de convolution

Exemple :

$$\text{Soit } f_X(x) = \begin{cases} x & 0 \le x \le 1 \\ 1 - x & 1 \le x \le 2 \end{cases}$$

on peut montrer que $X = U_1 + U_2$ où $U_1$ et $U_2$ ~ U [0,1] sont indépendants.

si $\mu_1 = .21$, $\mu_2 = .86$, alors $x = .21 + .86 = 1.07$.

# Algorithmes particuliers pour chaque loi de probabilité

Différents algorithmes pour chaque type de loi.

- Law & Kelton
- Bratley, Fox & Schrage
- Hoover & Perry (Sections 7.5 & 7.6)
- etc.

FIN