Chapitre 5.
Méthodes de Simulation

5.1.Principe de la simulation.

L'idée de la simulation statistique ou méthode de Monte Carlo est très simple. Elle consiste à simuler sur ordinateur (analogique ou numérique) le processus i.e. réaliser des expériences artificielles, puis en effectuer un traitement statistique (voir schéma du chapitre 1). A l'entrée du système, on décrit les facteurs aléatoires en générant des variables aléatoires et des événements selon des lois de probabilités données, et compte tenu du mode de fonctionnement du système. On obtient en sortie des estimations statistiques des caractéristiques (mesures de performance) nécessaires aux diverses décisions de l'utilisateur. Les caractéristiques du processus réel et simulé ne coïncident pas tout à fait. On peut cependant estimer la précision de leur proximité, et l'améliorer (par exemple en fixant le nombre de simulations) à l'aide des méthodes de statistique mathématique.

Soit à approcher la grandeur μ qu'il est difficile d'évaluer par une méthode mathématique directe. Considérons alors un espace $(\Omega,\Im,P)$, où $\Omega=\{\omega\}$ est l'espace des événements élémentaires, $\Im$ la σ-algèbre des événements, et P une probabilité sur $\Im$. On suppose qu'il existe une variable aléatoire X, c'est-à-dire une fonction $X=f(\omega)$, $\Im$-mesurable, telle que son espérance mathématique $E(X)$ approche suffisamment la grandeur μ (avec une précision acceptable). L'algorithme de calcul de $f(.)$ doit être plus simple (en complexité algorithmique, temps machine,...) que celui du calcul de la grandeur μ à l'aide des méthodes connues : pour que la simulation ait un sens. Générons un échantillon $\omega_1,\omega_2,...,\omega_n$ d'éléments de Ω de distribution P, alors pour n suffisamment grand, et en vertu de la loi des grands nombres :

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}f(\omega i)=\frac{1}{n}\sum_{i=1}^{n}X_i=E(X)=\mu \qquad \text{(p.s.)}$$

Par conséquent, pour n suffisamment grand, on peut utiliser l'approximation :

$$\mu\approx\mu_n=\frac{1}{n}\sum_{i=1}^{n}X_i$$

La précision d'une telle approximation peut être estimée à l'aide du théorème central limite: pour n suffisamment grand, μ_n est asymptotiquement de loi normale de moyenne $\mu=E(X)$ et de variance $\sigma^2=Var(X)/n$ .On peut alors déterminer un intervalle de confiance de niveau $1-\alpha$ (α petit) pour la grandeur μ :

$$\mu_n-u_{\alpha/2}\sigma/\sqrt{n}\prec\mu\prec\mu_n+u_{\alpha/2}\,\sigma/\sqrt{n}$$

où $\Phi(u_\alpha)=1-u_{1-\alpha}$, $\Phi(.)$ étant la fonction de répartition de la loi normale standard $N('0,1)$ de moyenne nulle et de variance unité. On constate ainsi que la précision de la méthode de Monte Carlo est de l'ordre de $1/\sqrt{n}$. En d'autres termes, pour diminuer de 10 fois la

précision, il faut disposer de 100 fois plus d'observations dans l'échantillon de la grandeur μ. Ceci explique le fait que la simulation par la méthode de Monte Carlo ne soit pas d'une trop grande précision par rapport à d'autres méthodes. Nous présentons plus loin un tableau comparatif qui montre que cet inconvénient peut être compensé par d'autres critères de performance (par exemple le gain en temps machine pour l'intégration numérique).

Exemple 1. (Expérience de Buffon pour le calcul de π). Considérons le plan (xOy) hachuré de droites parallèles à l'axe Ox et distantes d'une longueur L, et dans lequel on jette une aiguille de longueur l.

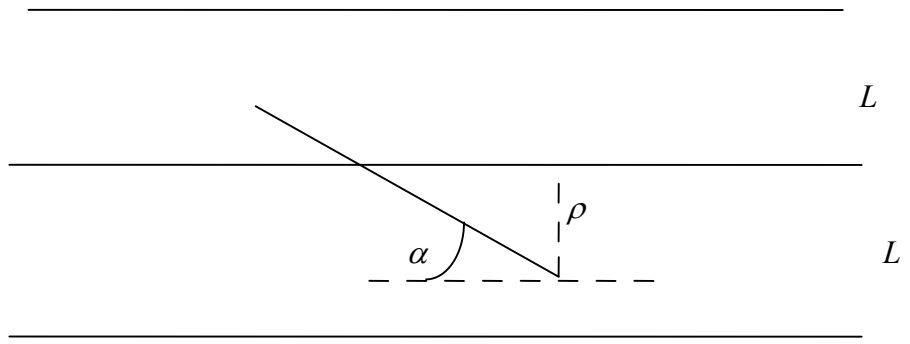


Figure 3.

Le jet de l'aiguille peut être assimilé au jet d'un point aléatoire (α,ρ) dans le rectangle de côtés L et π, $0\le\alpha\le\pi$, $0\le\rho\le L$.

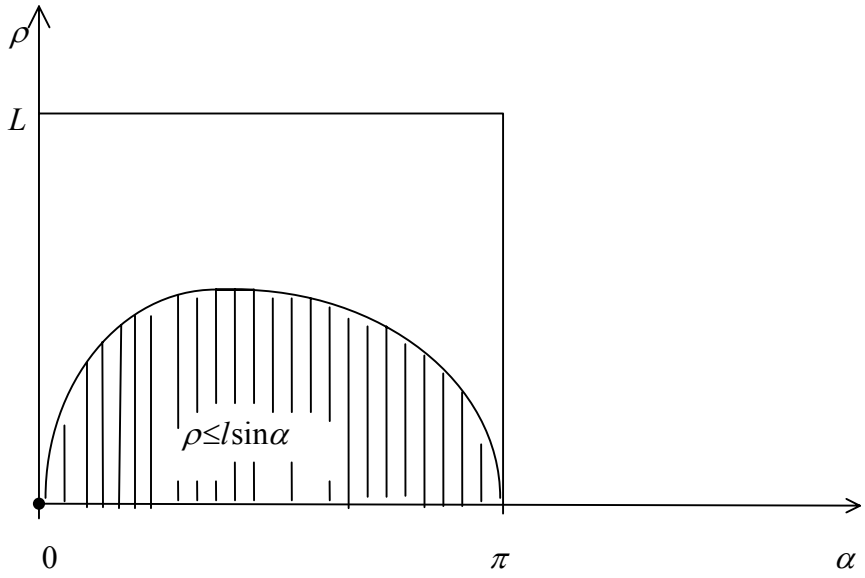


Figure 4.

L'événement A=« l'aiguille coupe au moins une des parallèles » est équivalent sur la seconde figure à l'événement «α et ρ sont tels que $\rho\le l\sin\alpha$». Par conséquent, la probabilité de l'événement A est la probabilité pour que le point aléatoire (α,ρ) tombe dans la partie hachurée (sur la figure 4):

$$P(A)=\frac{S(A)}{S}$$

où S est la surface du rectangle : $S=\pi L$ et $S(A)$ la surface sous la courbe s, ρ=l sin α . Soit,

$$S(A)=\int_0^{\pi} l\sin\alpha d\alpha=2l$$

Par conséquent, $$P(A)=\frac{2l}{\pi L}$$

Il s'agit d'obtenir un échantillon de taille n du couple aléatoire (α,ρ), soit (α_1,ρ_1), (α_2,ρ_2) , ..., (α_n,ρ_n). En vertu de la loi des grands nombres, la probabilité $P(A)$ peut être approchée par la fréquence relative de l'événement A

$$P(A)\approx\frac{n(A)}{n}, \text{ pour } n \text{ suffisamment grand.}$$

Ici $n(A)$ est le nombre de fois où l'aiguille coupe l'une des lignes. On peut alors déduire la valeur a posteriori de π à partir de la relation approximative:

$$\frac{2l}{\pi L}\approx\frac{n(A)}{n} \qquad \text{ou ; encore} \qquad \pi\approx\frac{2l}{L}\frac{n}{n(A)}$$

A l'aide de cette expérience, le naturaliste Buffon en 1717 a pu simuler le calcul de π en procédant à 16000 lancers de l'aiguille. Le tableau ci-dessous résume d'autres expériences postérieures. Au lieu de procéder à des lancers réels, il est plus simple aujourd'hui de réaliser cette expérience sur ordinateur en créant des échantillons artificiels indépendants:

$\alpha_1,\alpha_2,..,\alpha_n$ issu d'une loi uniforme sur $(0,\pi)$

$\rho_1,\rho_2,...,\rho_n$ issu d'une loi uniforme sur $(0,L)$

Expérimentateur	Année	nombre de lancers de l'aiguille	valeur expérimentale de π
Wolf	*1850*	*5 000*	*3,1596*
Smith	*1855*	*3 204*	*3,1553*
Fox	*1894*	*1 120*	*3,1419*
Lasarini	*1901*	*3 408*	*3,1415929*

Exemple 2. Soit à calculer l'espérance mathématique d'une variable aléatoire X, de fonction de répartition $F(x)=P(X\le x)$. Pour fixer les idées, supposons que X est la durée de vie d'un équipement, $R(x)=1-F(x)$ sa *fonction de fiabilité* (ou *probabilité de survie*, parfois notée

$\overline{F}(x)$). On cherche à estimer la durée de vie moyenne (connue en fiabilité par *MTTF* ou *MTBF*), qui est égale à $T_0 = E(X)$.

L'approche directe consisterait à procéder à des expérimentations physiques de laboratoire qui peuvent revenir très cher ou à des observations du système en exploitation, ce qui demanderait beaucoup de temps. Il est plus simple de procéder par simulation en commençant par créer artificiellement une suite de nombres aléatoires $\{x_1, x_2, ..., x_n\}$ à l'aide d'un ordinateur et pouvant être considérés comme la réalisation d'un échantillon issu de la variable aléatoire X.

En vertu de la loi des grands nombres, et des résultats de la statistique, la moyenne empirique

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} xi \qquad (5.1)$$

est un «bon» estimateur du MTTF (absence de biais, convergence, efficacité). On utilisera alors l'approximation:

$$MTTF = E(X) \approx \bar{x}$$

Exemple 3. Soit p la probabilité d'atteindre une cible au cours d'un tir. Si l'on procède à 10 tirs sur la cible, quelle est la probabilité de faire mouche un nombre pair de fois? Il est connu que la probabilité de faire mouche $2k$ 2k fois s'obtient à partir de la loi binomiale:

$$P_{2k} = C_{10}^{2k} p^{2k} (1-p)^{10-2k} \qquad 1 \le k \le 5 \qquad (5.2)$$

d'où la probabilité cherchée :

$$P = P\{2,4,6,8,10\} = \sum_{k=1}^{5} C_{10}^{2k} p^{2k} (1-p)^{10-2k} \qquad (5.3)$$

Supposons que l'on ne dispose pas de tables de la loi binomiale ou que cette formule soit inconnue. La probabilité P peut être calculée au moyen de deux procédés:

(i) On procède à des séries réelles de 10 tirs chacune sur la cible. Pour chaque série de 10 tirs, on relève la fréquence relative des issues paires $f_{2k}^{(i)}$, où i est le numéro de la série, $i=1,2,...,n$. Les probabilités (5.2) peuvent être estimées par: $\frac{1}{n}\sum_{i=1}^{n} f_{2k}^{(i)}$. La probabilité cherchée (1.3) s'obtient alors par sommation des valeurs paires.

$$P \approx \sum_{i=1}^{5} \left\{ \frac{1}{n} \sum_{i=1}^{n} f_{2k}^{(i)} \right\} \qquad (5.4)$$

(ii) Il est plus simple pour éviter des tirs réels de simuler ces expériences sur ordinateur. On génère d'abord les valeurs d'une variable aléatoire de loi uniforme sur l'intervalle $[0,1]$. Il est clair que $P(X \le p)=p$. A chaque étape de la simulation, on comparera donc la réalisation de la variable aléatoire $f_{2k}^{(i)}$ avec la probabilité p. Si $X \le p$, on considère que le tir a atteint la cible, dans le cas contraire, c'est l'échec. On effectue ainsi des séries de 10 expériences artificielles, et on fait le décompte des succès «pairs». Si le nombre de séries est suffisamment élevé, la fréquence obtenue sera proche de la valeur donnée par la formule (5.3).

Ces deux exemples illustrent assez bien le principe de la méthode de Monte Carlo. D'habitude, par méthode de Monte Carlo, on entend en général la résolution de problèmes déterministes (calcul d'intégrales, résolution d'équations différentielles) à partir de nombres au hasard.

Exemple 4: Calcul d'une intégrale par la méthode de Monte Carlo

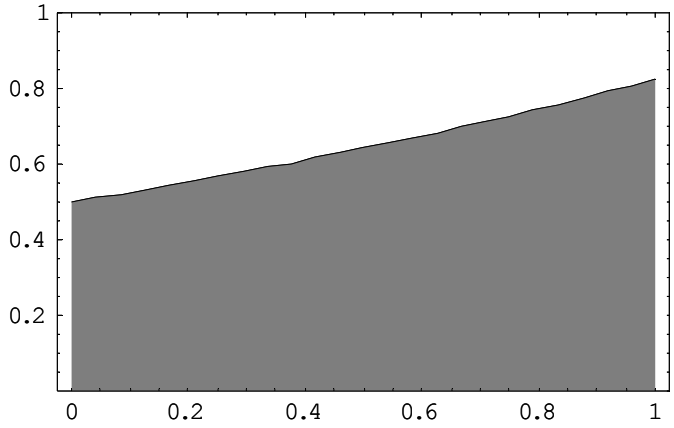
Soit à calculer l'intégrale définie : $I=\int_a^b f(x)dx$

où f(x) est une fonction connue, intégrable sur (a,b) vérifiant $c \le f(x) \le d$. On peut toujours se ramener au calcul de l'intégrale :

$$I=\int_0^1 f(t)dt\ ,\ 0 \le f(t) \le 1\ ,\ t \in [0,1].$$

sur l'intervalle $[0,1]$ moyennant une certaine transformation.

Soit par exemple à évaluer l'intégrale $I=\int_0^1 0.5e^{0.5}dx$. Il s'agit de déterminer l'aire comprise entre la courbe $y=f(t)$, l'axe des abscisses et les droites $t=0$ et $t=1$. C'est la surface hachurée sur le graphe suivant.



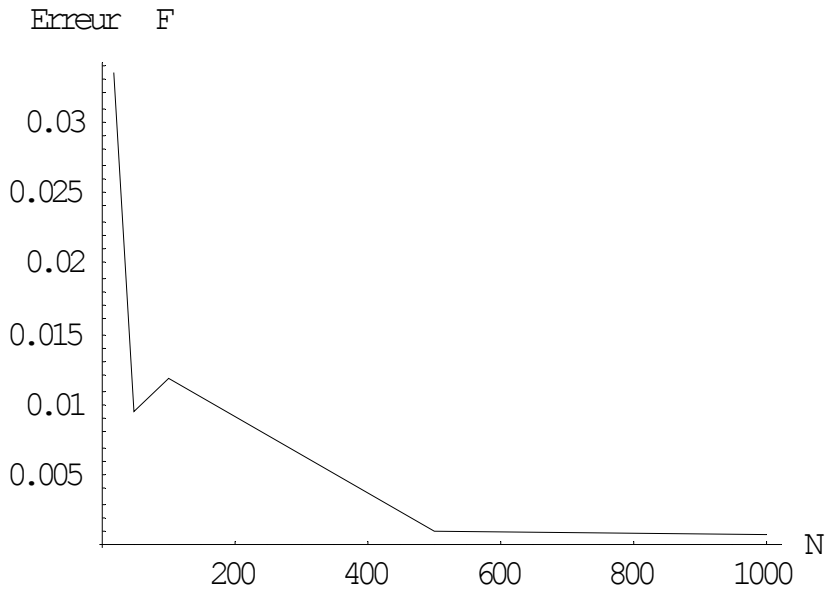
L'idée consiste à ramener un problème mathématique pur de calcul d'intégrale (on suppose qu'on est profane et qu'on ne sait pas calculer cette intégrale par les méthodes traditionnelles d'analyse). On imagine alors une expérience aléatoire qui consiste à jeter un point aléatoire dans le carré unité 1×1. On note que la valeur de l'intégrale I (la partie hachurée S) n'est

autre que la probabilité $p(S)$ pour ce point aléatoire de tomber dans la surface S. En vertu de la loi des grands nombres, cette probabilité peut-être estimée par la fréquence des points qui tombent dans la surface S.

La valeur approchée par une méthode numérique de quadrature classique (de type Trapèze) : J=0.648721. La table suivante montre les résultats de plusieurs expériences de simulation en faisant varier la taille de l'échantillon :

Taille de l'échantillon N	Valeur simulée U	Erreur F=|J-U|
20	*0.682202*	*0.0334803*
50	*0.639142*	*0.00957883*
100	*0.636924*	*0.0117974*
500	*0.64764*	*0.00108133*
1000	*0.649483*	*0.000762002*

Le graphe ci-dessous montre l'évolution de l'erreur lorsque la taille de l'échantillon N augmente.



Une méthode alternative est la suivante. Considérons la fonction :

$$\Phi(x,y)=\begin{cases}1, & si\ y<f(x)\\ 0, & si\ y\geq f(x)\end{cases}$$

L'égalité $\Phi(x,y)=1$ signifie que le point de coordonnées (x,y) se trouve dans la surface hachurée de la figure ci-dessus. On peut alors écrire :

$$\int_0^1 f(t)dt=\int_0^1\int_0^1 \Phi(x,y)dxdy$$

ou encore

$$\int_0^1 f(t)dt = E\{\Phi(\xi,,\eta)\}$$

où ξ et η sont des variables aléatoires indépendantes uniformément distribuées sur $[0,1]$.

On peut simuler la valeur de l'intégrale I de la manière suivante. Générons une suite aléatoire $\{U_n\}$ issue de la loi $U[0,1]$, et calculons successivement les valeurs de la fonction Φ

$$\begin{aligned} \Phi_1 &= \Phi(U_1, U_2) \\ \Phi_2 &= \Phi(U_3, U_4) \\ \Phi_3 &= \Phi(U_5, U_6) \\ \Phi_4 &= \Phi(U_7, U_8) \ \ldots\ldots \end{aligned} \tag{5.5}$$

En vertu de la loi des grands nombres, on aura l'approximation :

$$I = \int_0^1 f(t)dt \approx \frac{1}{n}\sum_{i=1}^{n} \Phi_i$$

pour n suffisamment grand.

Interprétation graphique. Considérons un point $(\xi,,\eta)$ jeté au hasard dans le carré $[0,1]\otimes[0,1]$. (cf. figure 1). La probabilité de tomber en dessous de la courbe $y = f(t)$ est égale à

$$p = \frac{surface\ \ hachurée}{surface\ du\ carré} = \frac{J}{1} = J$$

soit la valeur cherchée de l'intégrale: $p = I$. Soit l'expérience suivante: on choisit un couple de nombres au hasard $(\xi,,\eta)$ de loi $U[0,1]$, et on vérifie la condition: $f(\xi) > \eta$. Si cette condition est remplie, le point aléatoire $(\xi,,\eta)$ est en dessous de la courbe et $\Phi(\xi,,\eta) = 1$. Après avoir répéter cette expérience n fois, on obtient la suite de valeurs (5.5), et la moyenne empirique :

$$\frac{1}{n}\sum_{i=1}^{n} \Phi_i = \frac{M}{n}$$

représente la fréquence relative des points qui se trouvent en dessous de la courbe (M représente le nombre de fois où l'inégalité $f(\xi) > \eta$ est remplie). Par suite, en vertu de la loi des grands nombres :

$$\frac{M}{n} \xrightarrow[n\to\infty]{p} p = I$$

De manière générale, soit à calculer l'intégrale multiple

$$I=\int_B f(x)dx$$

Si le domaine B est fini, on peut l'inclure dans un parallélépipède R de côtés parallèles aux axes des coordonnées. Représentons l'intégrale I sous la forme

$$I=\int_B f(x)dx=\frac{1}{v(R)}\int_B f(x)v(R)dx=\frac{1}{v(R)}\int_{\Re} f(x)\chi_B(x)v(R)dx$$

où $v(R)$ est le volume du parallélépipède R, et $\chi_B(x)$ est l'indicateur du domaine B: $\chi_B(x)=1$ si $x\in B$, $\chi_B(x)=0$ sinon. L'intégrale I peut être comprise comme l'espérance mathématique de la fonction $\Phi(U)=f(U)\chi_B(U)v(R)$, U de loi uniforme sur le parallélépipède R. Par suite, selon la loi des grands nombres, la moyenne empirique des observations de la variable aléatoire $\Phi(U)$ converge vers son espérance mathématique

$$\frac{v(R)}{n}\sum_{i=1}^{n} f(xi)\chi_B(xi)\underset{n\to\infty}{\to} E\{\Phi(U)\}=\int_B f(x)dx$$

où x_i est la suite des valeurs échantillonnées, équidistribuées sur R et indépendantes.

On utilise souvent une variante plus simple sur un domaine arbitraire B. Soit $p_\xi(x)$ une fonction de densité qui ne s'annule en aucun point du domaine B, par exemple la densité normale non dégénérée. On peut alors représenter l'intégrale sous la forme :

$$I=\int_B f(x)dx=\int_B \frac{f(x)}{p_\xi(x)}p_\xi(x)dx=\int_{-\infty}^{\infty}\frac{f(x)\chi_B(x)}{p_\xi(x)}p_\xi(x)dx=E\{\Phi(\xi)\}$$

où $\Phi(\xi)=f(\xi)\chi_B(\xi)/p_\xi(\xi)$ et ξ est de densité p_ξ. A l'aide d'une suite générée selon la densité de ξ on approche I par la moyenne empirique des valeurs de $\Phi(\xi)$:

$$I\approx\frac{1}{n}\sum_{i=1}^{n}\frac{f(\xi i)\chi_B(\xi i)}{p_\xi(\xi i)}$$

Remarques :

1. Fiction ou réalité, il semble que l'appellation Monte Carlo" soit inspirée de l'intérêt qu'aurait eu Metropolis (concepteur d'algorithme d'optimisation stochastique) pour les jeux de hasard qui font la réputation des casino de la ville de Monte Carlo (principauté de Monaco).

2. Les méthodes de Monte Carlo sont évidemment moins précises que les procédures numériques d'intégration classiques (méthodes de quadrature). Le temps de calcul s 'améliore considérablement cependant dans le cas d'intégrales multiples (relativement aux méthodes de quadratures). (cf . note annexe).

5.2.Nombres Aléatoires.

5.2.1. Introduction. L'outil de base de la simulation est la source capable de produire des nombres aléatoires (ou nombres au hasard), c'est-à-dire une suite U_1, U_2,..., U_n, ... de variables aléatoires indépendantes et uniformément distribuées sur l'intervalle [0,1]. La suite $\{U_i\}$ est ainsi interprétée comme une série de réalisations d'une variable aléatoire U de loi uniforme sur [0,1] (on notera $U \in U[0,1]$). On peut à partir d'une telle suite générer toute autre suite de variables aléatoires de loi arbitraire, ou encore tout processus ou fonction aléatoire. Les nombres aléatoires sont utilisés également pour les jeux vidéos, ainsi qu'en cryptographie.

En pratique, on utilise des procédés physiques (mécaniques) ou algorithmiques permettant de produire de telles suites aléatoires. Ces dernières sont en fait pseudo-aléatoires, c'est-à-dire qu'elles sont déterministes, mais possèdent des propriétés statistiques identiques à celles que posséderait une suite réellement aléatoire.

5.2.2.Générateurs physiques.

L'exemple le plus simple, connu de tous ceux qui sont familiers du calcul des probabilités, est le jeu de « Pile » ou « Face » permettant de générer une variable aléatoire de Bernouilli qui prend les valeurs 0 ou 1 avec des probabilités identiques, égales à 1/2. Le jet d'un dé « parfait » permet de générer une variable aléatoire à valeurs dans {1,2,3,4,5,6} avec des probabilités identiques, égales à 1/6. La suite de nombres au hasard peut être obtenue également à l'aide d'une roulette telle qu'on peut en voir dans les fêtes foraines.

On peut utiliser également des dispositifs mécaniques plus sophistiqués tels que la machine servant au tirage du loto ou de Kora+. Ainsi, pour générer une variable binaire on a longtemps utilisé des sources de particules radioactives ; un compteur dénombre les particules détectées durant un temps Δt on considère alors la variable binaire $Y=1$ si le nombre de particules détectées durant le temps Δt est pair; $Y=0$ si ce nombre est impair. Un autre procédé utilise le niveau de bruit d'une valve électronique. La valeur du voltage u(t) est un processus aléatoire qu'on observe à des temps discrets $t_1,t_2,...,t_i,...$ On choisit un niveau de section c de manière « adéquate », par exemple tel que $P(U_i=1)$ soit le plus proche possible de 1/2, et on pose:

$$U_i=\begin{cases} 0, & \text{si} \quad u(t_i) \le c, \\ 1, & \text{si} \quad u(t_i) \ge c, \end{cases}$$

La logique de création des U_i peut être plus complexe afin de réduire le biais ou garantir d'autres propriétés statistiques de la loi U[0,1].

Voici à titre d'exemples quelques procédés physiques utilisés pour la génération de nombres aléatoires : sources radioactives (voir le principe ci-dessus), effet quantique dans les semi-

conducteurs, turbulence de l'air, microphone, camera vidéo ; souris ou horloge d'un ordinateur etc…

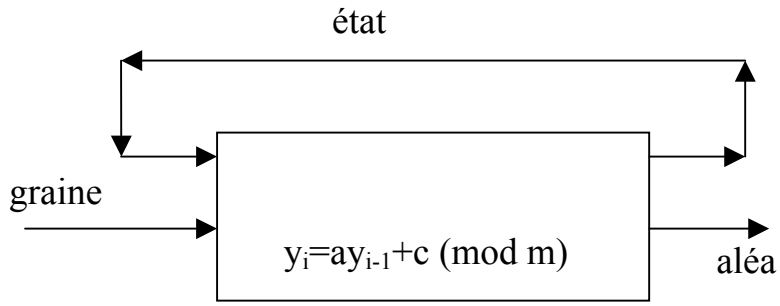
L'intérêt actuel de telles sources reste plutôt motivé par d'autres objectifs que la simulation. C'est par exemple le cas en cryptographie où les différentes méthodes de cryptage nécessitent la génération de nombres aléatoires produits par de telles sources. (voir par exemple http://world.std.com).

Les nombres produits à l'aide de telles sources sont appelés parfois nombres vraiment aléatoires (par opposition aux nombres pseudo-aléatoires du paragraphe suivant). Cependant on a toujours détecté des erreurs également dans de telles suites.

5.2.3. Générateurs algorithmiques ou pseudo-aléatoires.

Les suites produites par des procédés physiques ont été longtemps (et sont toujours) acceptées comme aléatoires. Cependant ils sont coûteux et les suites obtenues ainsi présentent des biais et dépendances. Bien que produisant également des nombres « pseudo-aléatoires », les générateurs algorithmiques ont l'avantage d'être simples à réaliser sur ordinateur. C'est en ce sens que la simulation a été l'une des toutes premières applications des premières générations d'ordinateurs dès 1940. Il existe une multitude de générateurs algorithmiques de nombres au hasard. Certains sont mieux testés que d'autres. La complexité d'un générateur ne conduit pas forcément à une suite plus « aléatoire » que celle d'un générateur plus simple. On conseille d'ailleurs en général d'utiliser un algorithme simple et bien assimilé. L'exception reste peut-être la cryptographie où on doit imposer une exigence supplémentaire : l'algorithme ne doit pas pouvoir être « casser » par l'adversaire.

Lorsqu'on génère une suite $\{U_i\}$ par un procédé algorithmique, on doit avant tout s'assurer que l'algorithme choisi possède une bonne forme mathématique simple et facilement programmable sur ordinateur. L'algorithme doit donc être récursif : $U_{i+1}=f(U_i, U_{i-1},...,U_{i-k})$ donnée à valeurs dans [0,1], qui peut avoir une forme plus ou moins « chaotique ». Pour les ingénieurs s'intéressant à la cryptographie ou à la simulation, le générateur pseudo-aléatoire est un objet décrit par un automate fini dont l'état initial est la graine (la semence, la racine, la clef).



La suite générée à l'aide de l'algorithme ci-dessus sera dans tous les cas **pseudo-aléatoire** car produite par un moyen déterministe. Les algorithmes récurrents possèdent un autre inconvénient qui réside dans le fait que la suite $\{U_i\}$ sera toujours périodique. C'est pourquoi, l'utilisation pseudo-aléatoire d'une telle suite ne pourra se faire que sur une succession d'épreuves inférieure à la période. La question qui se pose alors est de savoir déterminer une

période suffisamment grande pour garantir un nombre raisonnable de termes de la suite. Malgré les inconvénients cités, les nombres pseudo-aléatoires obtenus à partir de procédés récurrents sont très commodes pour la simulation:

¶ l'algorithme est facilement programmable, et nécessite un faible espace mémoire.

¶ toute suite de nombres peut être répétée autant de fois qu'on le désire, et à partir de n'importe quel rang; il suffit pour cela de donner un nombre initial U_0.

¶ une telle suite peut être testée sur l'uniformité une seule fois afin de pouvoir l'utiliser avec un niveau acceptable de confiance.

¶ rapidité pour garantir une utilisation multiple des nombres générés.

¶ l'étude des performances de telles suites récursives peut être réalisée à l'aide de certaines méthodes mathématiques développées (théorie des nombres, théorie des probabilités,…).

¶ La plupart des langages informatiques et logiciels sont dotés d'un sous-programme (ou routine) de génération de suites aléatoires uniformes sur [0,1] .

Table 1. Exemple de routines et fonctions utilisées dans les langages et logiciels usuels.

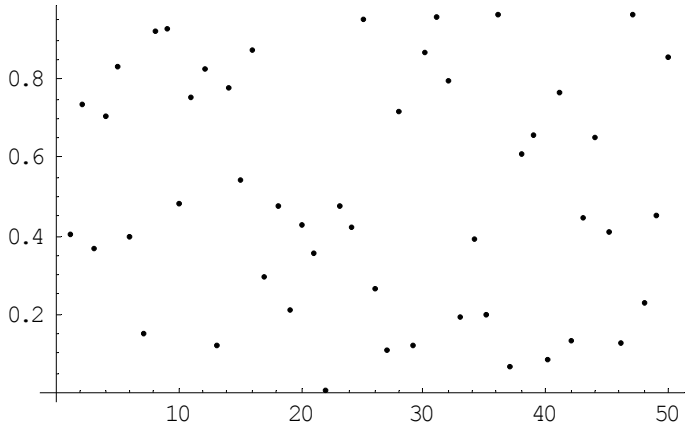
Langage	Routine ou macro	Type de nombre produit	Observation (Type de machine ou autre…)
FORTRAN	RANDU ; RAND	U∈[0,1]	IBM 360/370
FORTRAN 77	GOSCAF RANF	U∈[0,1]	NAG Library CDC Cyber 174 Vax (machine 8 à 64 bits)
PASCAL ou C	RANDOM RANDOM(n)	U∈[0,1] entier§n	
C++	rand ( )	Nombre entre 0 et RAND_MAX	
	srand		
	RANROT		
BASIC	RND ou RAND	U∈[0,1]	
MATLAB	rand randn rand(n,m)	U∈[0,1] X∈N(0,1) Matrice aléatoire	
MATEMATIKA	Random[ ] Random[Integer] Random[Real,a,b] Random[Integer,a,b] SeedRandom[ ]	U∈[0,1] U=0 ou 1 réele[a,b] entiere[a,b] initialisation	Avec proba= 1/2
Visual Basic	Rnd		
WORK PLACE	RANDOM NUMBERS	U∈[0,1]	
EXCEL	ALEA	U∈[0,1]	
BSAFE (boîte à outils cryptographiques)	MD5 Random SHA1 Random		

UNIX	rand() random() rand48()		
MAPLE	randomize uniform rand rand[r] rand(a..b) rand(n) randvector(n,entries=p)	Initialisation $U \in [0,1]$ Réel aléatoire Entier aléatoire entiere(a,b) ‹ rand(0..n-1) vecteur aléatoire	à 12 chiffres de dimension n
	MakeRandom (r)	entier	
	MakeRandom(a.b)	Entier entre a et b	
	RandUniform(a.b)	U[a,b]	
STATISTICA	Rnd(x) Uniform(x) Normal(x)	réel$\in$U[0,x[U$\in$U[0,x[réeleN(0,x^2)	 x=écart-type
ORACLE	Package DBMS_RANDOM		

Exemple 5: Ce petit programme avec MATEMATIKA vous liste une séquence de 50 nombres pseudo-aléatoires et les représente dans le plan.

```
B=Table[Random[],{50}]
ListPlot[b]
```

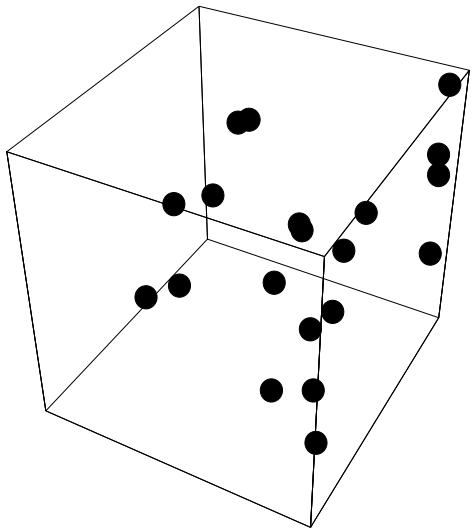
0.40419	0.735378	0.369251	0.70511	0.833578	0.397475	0.151053	0.925697
0.928252	0.481479	0.757547	0.82988	0.124692	0.777978	0.544763	0.878684
0.299862	0.478652	0.211158	0.430693	0.358442	0.005522	0.479885	0.426198
0.954252	0.270145	0.110634	0.721088	0.120675	0.87267	0.959581	0.795391
0.192522	0.39119	0.202034	0.965511	0.067830	0.613212	0.657272	0.086827
0.767969	0.13456	0.446113	0.65134	0.409527	0.129038	0.966229	0.229936
0.455275	0.858893						



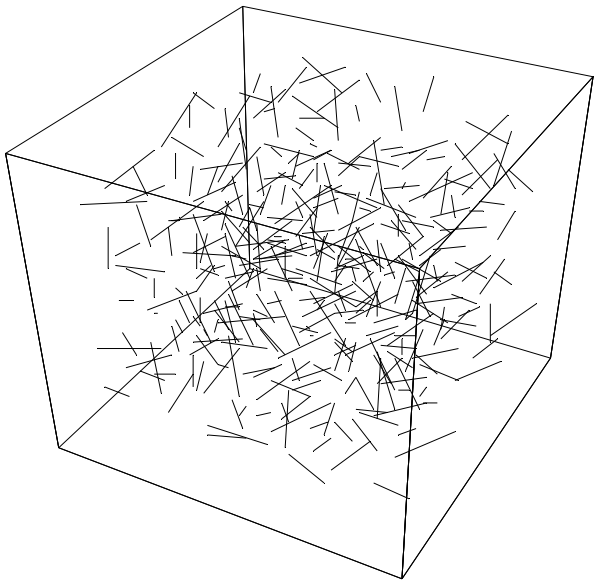
Exemple 6: En raison de l'importance des nombres aléatoires, la plupart des développeurs de logiciels prévoit dans leurs produits des générateurs de nombres au hasard adaptés à

différentes situations. En guise d'illustrations, l'exercice 9 de la série montre comment générer des nombres aléatoires en Java.

Exemple 5': Le graphe suivant représente 20 points aléatoires dans l'espace 3D (3 dimensions)



Exemple 5 '' : Le graphique suivant représente des vecteurs aléatoires en 3D.



5.2.4.Générateurs usuels.

Nous allons voir maintenant comment les générateurs algorithmiques ci-dessus ont été construits.

5.2.4.1.Générateurs binaires

Au lieu de tirer directement une variable uniformément distribuée U[0,1], on peut adopter l'approche suivante qui consiste à représenter tout nombre dans un système de base b, par une suite de chiffres. La méthode consiste alors à tirer successivement chacun des chiffres de la suite avec une approximation suffisante. Cette approche a l'avantage d'exploiter le

Hardware spécifique de l'ordinateur utilisé. En système binaire de base b=2, tout nombre U peut être représenté par une suite de chiffres pouvant prendre les valeurs 0 ou 1 :

$$U=z_1.2^{-1}+z_2.2^{-2}+...+z_n.2^{-n}+.... \quad (5.6)$$

On propose ainsi de générer les valeurs d'une variable aléatoire binaire telle que:

$$\mathbf{P}(z_i=0)=\frac{1}{2}=\mathbf{P}(z_i=1) \quad (5.7)$$

Ceci peut être réalisé sur ordinateur à l'aide d'un procédé physique ou algorithmique. On peut montrer que la variable U construite à l'aide de (5.6) et vérifiant (5.7) est de loi uniforme U[0,1]. La propriété (5.7) doit être comprise au sens où chaque terme z_i de la suite (5.6) apparaît avec une proportion asymptotique de ½. La représentation (5.6) ne peut être réalisée pratiquement sur ordinateur qui ne peut exprimer les nombres que sous forme de « mots » d'un nombre limité de chiffres (ou de bits).

Soit une machine binaire donnant des mots de k bits chacun. Elle ne pourra exprimer que 2^k mots différents sur [0,1]

$$0,\ \frac{1}{2^k},\frac{2}{2^k},....,\frac{2^{k-1}}{2^k}$$

qui sont les réalisations possibles du nombre pseudo-aléatoire $\approx{U}$. S'il y a équidistribution, chacun de ces nombres $\approx{U}$ est de la forme:

$$\approx{U}=\sum_{i=1}^{k} z_i.2^{-i}$$

avec une probabilité identique 2^{-k} (en fait la fréquence asymptotique). On montre que $E(\approx{U})Ø1/2$ et $var(\approx{U})Ø1/12$ qui sont respectivement l'espérance mathématique et la variance d'une variable uniforme sur [01].

Pour une machine décimale $m=10^b-1$.

5.2.4.2.Générateur de Fibonacci. Il est de la forme

$$y_{i+1}=y_i+y_{i-j} \pmod m$$

Ce générateur est juste cité ici pour l'histoire, car il a été constaté qu'il ne produisait pas de nombres pseudo-aléatoires satisfaisants, notamment en cryptographie.

5.2.4.3.Générateurs congruentiels.

C'est en fait toute une famille de générateurs définis par

$$y_i=(\lambda y_{i-1}+C) \bmod m$$

où λ, C et m sont des entiers. Ce procédé signifie que y_i est le reste de la division de $\lambda y_{i-1}+C$ par m, où m est un nombre entier de grande taille (généralement une puissance de 2 pour les machines binaires (où $m=2^{k-1}$, k étant la longueur d'un mot machine en bits, le premier bit étant réservé au signe); λ est un nombre entier compris entre 0 et m-1 ; y_0 est appelé la racine

(ou le germe) du générateur. En guise de nombre pseudo-aléatoire, on utilise les nombres $U_i=y_i/m$ qui sont équidistribués sur [0,1] lorsque $i\to\infty$.

Pour C=0, on obtient le générateur multiplicatif (de Lehmer) qui imite bien l'effet d'une roulette. Pour C ∫ 0, on obtient le générateur mixte.

La plupart des routines et fonctions de la table 1 utilisent des générateurs congruentiels. A titre d'exemple, la table 2 donne les valeurs des paramètres du générateur λ, m,C, y_0 utilisées dans ces langages.

Table 2.Exemples de valeurs des paramètres du générateur congruentiel pour les langages usuels.

Langage	Fonction ou routine	m	λ	C	Type de machine
FORTRAN	RANDU	2^{31}	$2^{16}+3$	0	IBM 360/370
FORTRAN NAG	G05caf	2^{59}	13^{13}	0	
FORTRAN IMSL	GGUBT	$2^{31}-1$	397204094	0	(machine 32-bits)
		$2^{35}-31$	5^5	0	(machine 36 bits)
FORTAN 77		2^{32}	69069	1	VAX /VMS (32 bits)
FORTRAN 77	RANF( )	2^{48}	44485709377909	0	CDC Cyber 174 (60 bits)
NAG Library	G05CAF	$_2{}^{59}$	5^{13}	0	Machine 60 bits
SIMULA		67099547	2^{13}	0	
GPSS, SIMSCRIPT, SLAM, SIMAN, ARENA	$2^{31}-1$ou 2^{48}				
JAVA	Java.util.random	2^{48}	Voir exercices		
Visual basic		2^{24}	1140671485		
Excel			Voir exercices		

Pour éviter les cycles (récurrence de la même séquence de nombres aléatoires), le générateur doit avoir une période la plus grande possible (cela dépend du choix de λ, m et de la racine).

Exemple7 : (i) si $m=2^b$ c ∫ 0, alors $p_{max}=m=2^b$. Il faut pour cela que c soit premier avec m et λ=1+4k.

(ii) si $m=2^b$ c=0, alors $p_{max}=m/4=2^{b-2}$. Il faut pour cela que la racine soit paire, et λ=1+4k. ou λ=5+8k

(iii) si m est premier et c=0, alors p_{max}=m-1. Il faut que λ soit tel que le plus petit k tel que λ^k-1 est divisible par m soit égal à k=m-1.

La période est souvent déterminée expérimentalement.

5.2.4.4.Générateurs registre mobiles ou à décalage.(shift-register).

Ces générateurs utilisent un hardware spécial utilisant une séquence de bits pseudo-aléatoires en enregistrant les d derniers bits $b_{i-1},...,b_{i-d}$ tels que $b_i=f(b_{i-1},...,b_{i-d})$ où f(.) est une certaine fonction de $\{0,1\}^d \rightarrow \{0,1\}$. Le générateur est de la forme:

$$U_i=0.b_{iL}b_{iL+1}...b_{iL+M}$$

où L et M sont des entiers tels que L>M>0, et la fonction f est généralement choisie

$b_i=(a_1b_{i-1}+...+a_db_{i-d}) \bmod 2$ où $a_1,...,a_d$ sont des constantes binaires.

Lorsqu'on utilise un algorithme récurrent, on doit également s'assurer que la suite produite $\{U_i\}$ possède bien des propriétés voisines de celle d'une suite aléatoire et uniforme sur [0,1].

5.2.4.5. Générateurs cryptographiquement sûrs. Dans les applications liées à la sécurité des systèmes (militaires, système de paiement électronique…), on exige que le générateur soit crypto graphiquement sûr. Cela exclut les générateurs évoqués ci-dessus et disponibles dans les langages et logiciels peu sûrs contre les attaques et qui peuvent être décryptés aisément. L'algorithme générateur doit être dans ce cas conçu de telle manière que l' « adversaire » ne puisse deviner le bit suivant avec une probabilité supérieure à (1/2)+e où e décroît exponentiellement avec un certain paramètre de sécurité s (en général la longueur du générateur). Il existe d'autres manières de définir de tels générateurs. (http://world.std.com).

5.2.5. Validation des générateurs.

Enumérons quelques critères permettant de vérifier que la suite générée $\{U_i\}$ est bien aléatoire et issue d'une variable aléatoire Y de la loi uniforme U[0,1].

D'abord, si une telle suite est bien issue de la loi uniforme sur [0,1], alors en vertu de la loi des grands nombres, la moyenne empirique :

$$\bar{U}=\frac{1}{n}\sum_{i=1}^{n}U_i$$

doit converger en probabilité vers 1/2. On obtient ainsi la première exigence :

Test 1. Si la suite $\{U_i\}$ est tronquée à la valeur n (assez grande), alors $\bar{U}$ doit être voisin de 1/2.

De la même manière, on obtient l'exigence suivante sur la variance empirique.

Test 2 $$S^2=\frac{1}{n}\sum_{i=1}^{n}(U_i-\bar{U}) \xrightarrow{p} 1/12$$

Pour la loi uniforme sur [0,1], on doit avoir **P**{U∈[a,b]}=b-a, ∀ [a,b]⊂[0,1]. En vertu de la loi des grands nombres, la proportion de valeurs U_i dans [a,b)

$$\lim_{n\to\infty} \frac{V[a,b)}{n} = \mathbf{P}\{U\in[a,b)\}=b-a$$

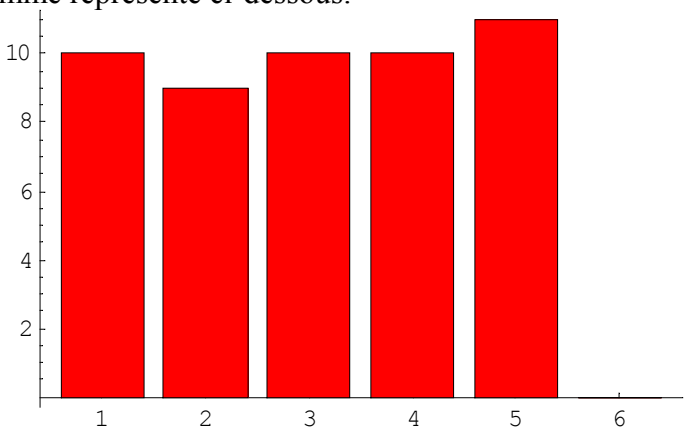
On partage donc l'intervalle [0,1] en m intervalles $I_j=[\frac{j-1}{m},\frac{j}{m})$, j=1,2,..,m (m<<n). Soit V_j le nombre de points U_i qui tombent dans l'intervalle I_j. On a alors l'exigence sur les fréquences :

Test 3. $V_j/n\approx 1/m$, ∀ j=1,2,…,m

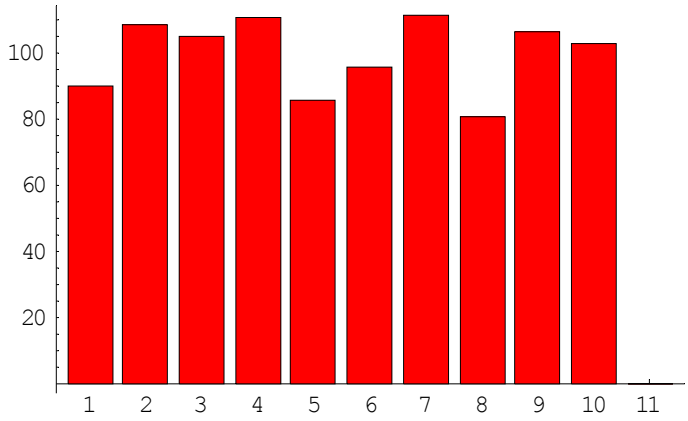
Exemple 7: Le tableau ci-dessous donne 50 nombres pseudo-aléatoires obtenus à l'aide d'un générateur congruentiel (voir exemple 5 et exercice 8).

0.40419	0.735378	0.369251	0.70511	0.833578	0.397475	0.151053	0.925697
0.928252	0.481479	0.757547	0.82988	0.124692	0.777978	0.544763	0.878684
0.299862	0.478652	0.211158	0.430693	0.358442	0.005522	0.479885	0.426198
0.954252	0.270145	0.110634	0.721088	0.120675	0.87267	0.959581	0.795391
0.192522	0.39119	0.202034	0.965511	0.067830	0.613212	0.657272	0.086827
0.767969	0.13456	0.446113	0.65134	0.409527	0.129038	0.966229	0.229936
0.455275	0.858893						

Pour vérifier l'uniformité, appliquons le test 3 en partageant l'intervalle [0,1] en 5 intervalles. L'histogramme représente la distribution des fréquences des intervalles (0;0.2)-(0.2 ;0.4)-(0.4 ;0.6)-(0.6 ;0.8)-(0.8 ;1) : les effectifs respectifs sont 10,9,10,10,11. Le test 3 stipule que si les nombres sont vraiment aléatoires, il devrait y avoir une proportion de 1/5 (ou 20%) de points dans chacun des intervalles, soit en moyenne 10 points par intervalles. Ceci semble être le cas sur l'histogramme représenté ci-dessous.



Ce second exemple visualise la distribution de 1000 nombres aléatoires sur 10 intervalles.
d={90,109,105,111,86,96,112,81,107,103}



On peut utiliser d'autres critères spécifiques, par exemple

Test 4. Pour toute fonction intégrable au sens de Riemann sur [0,1], on doit avoir :

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n} f(Ui)=\int_0^1 f(x)dx$$

Test 5. Si $U\in U[0,1]$, alors $\mathbf{E}(U)\pm\sigma_U = E(U)\pm\sigma_U = \frac{1}{2}+\frac{1}{2\sqrt{3}}$ et la proportion de points dans l'intervalle $(0.21132;0.78868)$ doit être voisine de $2\sigma_U = 0.5773602$.

On peut également utiliser des tests statistiques (Khi-deux, Kolmogorov,…), ainsi que certains tests spécifiques de l'uniformité et d'indépendance (voir plus loin).

En particulier, on peut tenter de vérifier que $\{U_i\}$ est k-équidistribuée, c'est-à-dire la distribution empirique de $(U_i,\ldots,U_{i+k-1})$ converge vers la loi uniforme sur $[0,1]^k$, $\forall k$.

Test 6 (Test du Khi-deux). C'est un test qui porte sur la distribution de la séquence.

Soit une partition $I_1+I_2+\ldots+I_k$ de l'ensemble de définition de la variable à tester, ici $U\in U[0,1]$.

Soit $p_i=P\{U\in I_i\}$, $\sum_{i=1}^{k} p_i = 1$. Notons par V_i le nombre d'observations qui tombent dans l'intervalle I_i, i=1,2,…,k, où $n=\sum_{j=1}^{k} V_j$ est le nombre total d'observations.

Si l'hypothèse nulle est vraie, alors $\frac{V_i}{n} \to p_i$ et la statistique

$$\aleph_2 = \sum_{i=1}^{k}\frac{(V_i - np_i)^2}{np_i}$$

suit asymptotiquement ($n\to\infty$) une loi du khi-deux à k-1 degrés de liberté. On rejette l'hypothèse nulle pour les grandes valeurs de χ^2 observé, i.e. $\chi^2 > \chi^2_{\alpha,p}$ où $P\{\chi_2 > \chi^2_{\alpha,p}\}=\alpha$.

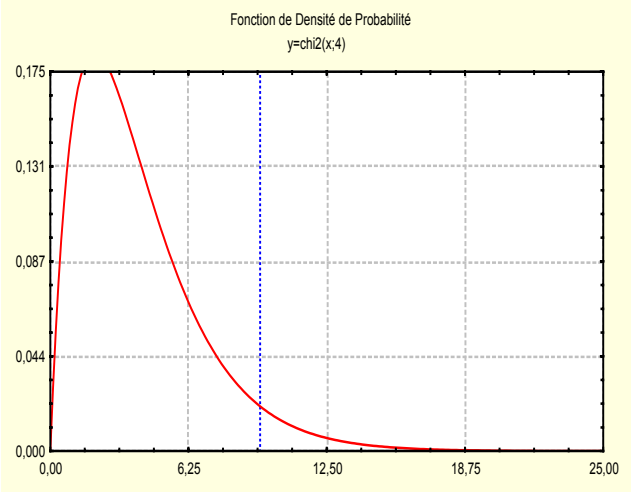
Pour le test de la loi uniforme, $p_i = |I_i|$, la longueur de l'intervalle constituant la ième classe.

Pour le test d'une loi de fonction de répartition F(x), on a $p_i=F(b_i)-F(a_i)$ où a_i et b_i sont les extrémités inférieure et supérieure respectivement de l'intervalle I_i. De plus, le degré de liberté doit être pris égal à k-m-1, où m est le nombre de paramètres estimés, de préférence à l'aide de la méthode du maximum de vraisemblance ou du minimum du Khi-deux. En pratique, les classes $I_1,I_2,\ldots,I_k$ doivent être choisies telles qu'elles contiennent au moins cinq observations en queue de distribution. Voir annexe 2 .

Exemple 7(suite): Reprenons l'exemple 7 ci-dessus. Pour confirmer notre observation graphique, on utilise le test du khi-deux.

$$\chi^2_{obs}=\sum_{i=1}^{5}\frac{(V_i-50p_i)^2}{50p_i}=\frac{(10-10)^2}{10}+\frac{(10-9)^2}{10}+\frac{(10-10)^2}{10}+\frac{(10-10)^2}{10}+\frac{(10-11)^2}{10}=0.2$$

Prenons pour seuil de confiance a=5%. Dans la table de la loi du khi-deux, on lit la valeur $\chi^2_{0.05;4}$ où $P\{\chi^2 > \chi^2_{0;05,4}\}=0.05$ ou bien $P\{\chi^2 < \chi^2_{0;05,4}\}=0.95$. Sur le graphique, la valeur $\chi^2_{0.05;4}$ est celle pour laquelle la surface sous la courbe à gauche de la ligne en pointillés est égale à 0.05 ; la surface à droite (sous la courbe) est elle égale à 0.95.



On trouve $\chi^2_{0.05;4} \approx 9.48840$ et puisque la valeur observée $\chi^2_{obs} = 0.02 < 9.48840$, alors on peut accepter l'hypothèse selon laquelle la suite est bien une suite aléatoire .

Test 7 (Test de Kolmogorov –Smirnov). C'est aussi un test de fréquences. On l'utilise lorsque la loi à tester possède une fonction de répartition continue F(x). La statistique du test est

$$D_n = \max_x | F_n(x) - F(x) |$$

où $F_n(x)$ est la fonction de répartition empirique. On montre que

$$\lim_{n \to} P\{D_n \sqrt{n} < x\} = K(x) = 1 - 2 \sum_{k=1}^{\infty} (-1)^{k-1} e^{-2x^2 k^2}$$

La loi limite est tabulée. En particulier K(1,36)=0,95.

Exemple (suite): Le test K-S dans sa version originale nécessite une comparaison des valeurs théoriques et empiriques, observation par observation, quoiqu'il soit possible de modifier le test pour des données groupées. Considérons un petit exemple pour illustrer le principe du test à partir des 10 premières observations seulement.

i	x_i ordonnées	$\hat{F}(x) = \frac{i}{n}$	$F_0(x_i)$	$F_0(x_i) - \frac{i}{n}$	$\hat{F}(x) = \frac{i}{n}$
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Exercice : Soit la suite de nombres générée à l'aide d'une suite récurrente. Peut-on les utiliser en guise de suites de nombres aléatoire uniformes entre 0 et 36 ?

$$x_1 = 23, x_2 = 18, x_3 = 1, x_4 = 16, x_5 = 2, x_6 = 3, x_7 = 20, x_8, x_9 = 7, x_{10} = 0$$

On cherche à tester

$$H_0 : F(x) = F_0(x) \quad \text{contre} \quad H_1 : F(x) \neq F_0(x)$$

où

$$F_{\grave{a}}(x) = \begin{cases} 0, & x \leq 0, \quad x \leq 0 \\ \frac{x}{36}, & 0 < x \leq 36 \\ 1, & x > 36 \end{cases}$$

i	x_i ordonnées	$\hat{F}(x) = \frac{i}{n}$	$F_0(x_i)$	$\frac{i}{n} - F_0(x_i)$	$F_0(x_i) - \frac{i-1}{n}$

1	0	0.1	0	0.1	0
2	1	0.2	0.027777	0.1722222	-0.07232
3	2	0.3	0.05555	0.244444	-0.1444
4	3	0.4	0.083333	0.316666	-0.2166
5	4	0.5	0.11111	0.388889	-0.28889
6	7	0.6.7	0.194444	0.405555	-0.30555
7	16	0.8	0.444444	0.255555	-0.155
8	18	0.9	0.5	0.3	-0.2
9	20	1.0	0.555555	0.34445	-0.2444
10	23	0	0.638888	0.361112	-0.2612

La valeur observée de la statistique $D_{10}=1.28247$ et $\sqrt{10}D_{10}=1.28247$.
Les tables de la loi de Kolmogorov-Smirnov (version papier, surtout à des fins pédagogiques les derniers temps) ou les logiciels comportant un module de calcul des valeurs de cette loi (version logicielle à des fins pédagogiques ou de recherche à l'époque actuelle) ont deux manières d'indiquer la technique de décision :

(i) Les tables indiquent par exemple que $P(\sqrt{10}D_{10}>1.2824)\approx 0.0628$. C'est à l'utilisateur d'apprécier si le seuil de confiance de à 6.28% est acceptable ou non.
(ii) Supposons que l'utilisateur se fixe un seuil est de 5% , on accepte ???

Nous fournissent indiquent que

Test 8 (d'uniformité).

Le test de Khi-deux peut être utilisé pour tester l'uniformité d'un vecteur de dimension k, $(U_{ki},\ldots,U_{ki+k-1})$.On partage dans ce cas $[0,1]^k$ en domaines identiques, et on teste les fréquences d'occurrence des réalisations du vecteur ci-dessus dans les différents intervalles

Test 9. (d'auto corrélation). Il vise à tester la dépendance entre les nombres aléatoires d'une même séquence. Il est basé sur le coefficient d'auto corrélation empirique r_{im} entre m nombres partant du ième nombre. Considérons la sous-suite de notre suite de nombres au hasard : $U_i, U_{i+m}, U_{i+2m},\ldots, U_{i+(M+1)m}$. La valeur M est en général le plus grand entier tel que i+(M+1)m§n où n est le nombre total de nombres aléatoires de la séquence. On cherche donc à tester une sous-suite de longueur M+2. Il n'y a pas indépendance si l'auto corrélation est non nulle. Le test est donc de la forme :

$$H_0 : \quad r_{im}=0 \quad \text{contre} \quad H_1 : \quad r_{im} \int 0$$

La statistique de test est naturellement le coefficient d'auto corrélation empirique de r_{im}, que nous noterons $\hat{\rho}_{im}$ .Cet estimateur suit approximativement une loi normale pour les grandes valeurs de M, si seulement les valeurs $U_i, U_{i+m}, U_{i+2m},\ldots, U_{i+(M+1)m}$ sont non corrélées. Par conséquent, sous l'hypothèse nulle (indépendance) la statistique $Z=\dfrac{\hat{\rho}_{im}}{\sqrt{Var(\hat{\rho}_{im})}}$ suit une loi normale standard de moyenne nulle et de variance unité, pour les grandes valeurs de M. En pratique, on utilise une modification de la statistique de test

$$\hat{\rho}_{im}=\frac{1}{M+1}\left[\sum_{i=0}^{M} U_{i+km}U_{i+(k+1)m}\right]-0.25 \qquad \text{avec} \qquad Var(\hat{\rho}_{im})=\frac{\sqrt{13M+7}}{12(M+1)}$$

On rejette l'hypothèse nulle au seuil a si Z est en dehors de l'intervalle $\left[-u_{\alpha/2};u_{\alpha/2}\right]$.

Autres critères souhaités : Rapidité pour l'utilisation multiple de nombres aléatoires standardisation, période suffisamment large. Ces critères suffisants pour les applications usuelles de la simulation peuvent être accompagnés d'autres exigences dans deux cas principaux :

(i) les applications « sensibles » (nucléaire, santé,…), où l'on peut être amenés à être plus exigeant pour la validation des générateurs.
(ii) Les applications liées à la sécurité (cryptologie) : outre l'exigence sur la « fiabilité » du générateur, on peut exiger qu'il soit « cryptographiquement sûr ».

5.2.6. Liens utiles

1. Simulation et modélisations discrètes ;
   http://ina.eivd.ch/ina/Collaborateurs/cez/Mod im/modsim.htmhttp
2. Douillet, ensait, http://193.43.37.48/~douillet/cours/oprea/node3.html
3. http://crypto.mat.sbg.ac.at/~ste/dipl/node10.htm
4. Cryptographic random numbers , http://world.std.com.
5. Générateurs pseudo-aléatoires en C++,
http://www.agner.org/random/random.htm
5. http://java.sun.com/j2se/1.3/docs/api/java/util/Random.html
6. http://support.microsoft.com/support/kb/articles/Q231/8/47.ASP
7. http://support.microsoft.com/directory
8. http://www.iro.umontreal.ca/~lecuyer

5.2.7.Exercices.

Exercice 1. Quelle est la différence entre générateurs aléatoires et pseudo-aléatoires ?

(i) Générer une suite de 10 nombres pseudo-aléatoires à l'aide du générateur congruentiel de paramètres : U_0=22 ; λ=2,C=2 et m=100.
(ii) **(iii) I**ndiquer comment tester la qualité de la suite ainsi générée.
(iii) **(vi)** Utiliser cette suite pour simuler l'intégrale suivante : $\int_1^2 x^2 dx$.

Exercice 2: **Méthodes des carrés moyens.**

C'est le procédé algorithmique le plus ancien et dû à *Von Neumann*.

(i) choisir le terme initial de la suite U_0.

(ii) calculer $a=U_0^2$.

(iii) Poser U_1 =suite des chiffres situés au milieu de la partie décimale de a.

(iv) Poser $a=U_1^2$; retour en (iii)

Le nombre devra à chaque itération comporter autant de chiffres significatifs que le nombre initial U_0.

Exemple. $U_0=0.2481$; $a=U_0^2=0.06155361\Rightarrow U_1=0.1553$; $a=U_1^2=0.02411809\Rightarrow U_2=0.4118$; etc...

Ce procédé est très simple et facile à programmer. Son inconvénient est qu'on obtient beaucoup plus de petites valeurs qu'il n'en faut. De plus, si à une étape donnée l'un des nombres est nul, c'est toute la suite qui sera constituée de termes nuls.
Pour s'assurer que la suite obtenue s'accorde bien avec la loi uniforme sur U[0,1], on utilisera les critères énumérés précédemment. Vérifions par exemple l'uniformité sur l'échantillon de n nombres obtenus à l'aide du générateur de Von Neumann (En fait, il faudrait un échantillon plus grand). On a, $\overline{U}=0.570055$ qui est bien voisin de $1/2$.

Exercice 3 : Utiliser la méthode congruentielle pour générer une séquence de nombres aléatoires de 10 chiffres telle que $x_{n+1}\equiv x_n+3)\bmod 10$ et $x_0=2$. Quelle est la période ?

Sol : 2,5,8,1,4,7,0,3,6,9,2

Exercice 4 : (i) Soit un générateur congruentiel avec $\lambda=13$, $m=2^6$. Déterminer la période maximale pour $X_0=1$ puis 3. Générer une période de chacune des suites pour $X_0=1,2,3,4$ et comparer.

(ii) Trouver la période maximale pour le générateur congruentiel avec $\lambda=7^5$, $m=2^{31}-1, c=0$.

Générer 3 nombres aléatoires lorsque la racine est $X_0=123457$.

Exercice 5 : L'automate de régulation d'un carrefour a été programmé pour fonctionner 50% du temps au vert, 10% du temps à l'orange et 40% du temps au rouge.

(i) Générer une séquence de nombres aléatoires.
(ii) Utiliser cette séquence pour générer une séquence de couleurs.

Exercice 6 : Générer cinq observations de chacune des lois suivantes :

(i) la loi uniforme sur l'intervalle [-10 ; 20].
(ii) La loi triangulaire de densité $f(x)=2x$ *si* $x\in[0,1]$; $f(x)=0$ sinon.
(iii) La loi de densité $f(x)=\frac{x-10}{50}$ *si* $x\in[10,20]$; $f(x)=0$ sinon.
(iv) La loi normale de moyenne $m=2$ et de variance $\sigma^2=4$;
(v) La loi d'Erlang d'ordre 2 de moyenne 4.

Exercice 7. Cet exercice reprend l'exemple du début de ce chapitre pour illustrer la simulation par la méthode de Monte Carlo du calcul d'une intégrale. Le petit programme suivant calcule

une valeur approchée de I par la méthode de Monte Carlo avec une séquence de N=100 nombres aléatoires générés selon une *méthode congruentielle*.

```
J = NIntegrate@0.50 Exp@0.5 xD, 8x, 0, 1<D
a = Table@Random@D, 8100<D;
c = Table@0.5 Exp@0.5 a@@iDDD, 8i, 100<D;
U = N@Sum@c@@iDD, 8i, 1, 100<D ê 100D
F = Abs@J - UD
```

Pour comparer, la première ligne fournit la valeur approchée par une méthode numérique de quadrature classique (de type Trapèze) : J=0.648721. Les résultats sont donnés dans la table. Le tracé de la courbe est réalisé par le programme suivant

```
<< Graphics`FilledPlot`
FilledPlot@0.5 Exp@0.5 xD, 8x, 0, 1<, PlotRange → 80, 1<, Frame → TrueD
```

Exercice 4: Soit X une variable aléatoire de loi exponentielle de paramètre λ=2.

(i) Générer une suite de n=5 observations artificielles de cette loi (en utilisant la suite donnée dans le tableau N°1). Evaluer la moyenne arithmétique de ces 5 observations. Comparer avec l'espérance mathématique de X. Qu'en déduisez-vous ?

(ii) (TP N°2) Rédiger un programme (en C) permettant de recommencer l'expérience pour n=10,20,30,50. Tracer sur un même graphique l'histogramme expérimental et la courbe théorique.

Exercice 5 : Ecrire le programme générateur des lois de probabilité suivantes :

(i) loi uniforme sur [a,b].

(ii) loi de Weibull de densité $f(x)=1-e^{-(\lambda x)^{\beta}}, x\geq 0$.

Exercice 6 : Ecrire le programme générateur de la loi géométrique de trois manières :

(i) Utiliser le programme usuel de la méthode d'inversion. Donner une interprétation.

(ii) Résoudre explicitement l'équation en X, et remarquer qu'on peut utiliser le générateur de la loi exponentielle.

(iii) Utiliser un algorithme similaire à celui de la loi binômiale.

Comparer les performances des trois algorithmes.

Exercice 7 : Générer trois observations pour chacune des lois suivantes :

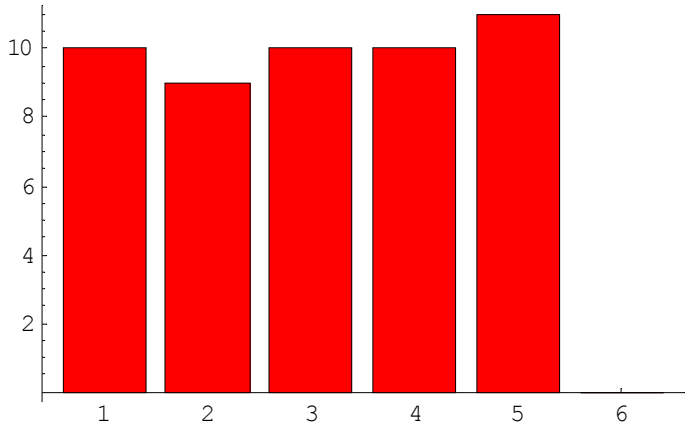
(iv) la loi normale (utiliser les deux méthodes et comparer).

(v) La loi d'Erlang (ou Gamma) de paramètres k=2,λ=2.

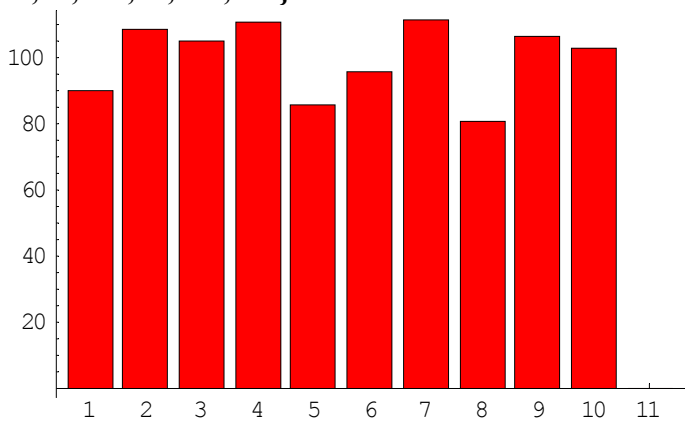
Exercice 8: Ecrire un programme permettant de générer 50 nombres aléatoires, les représenter dans le plan, puis tester leur uniformité à l'aide du test 3. Confirmer ou infirmer la visualisation graphique à l'aide d'un test de khi-deux.

Solution : Ce petit programme en MATEMATIKA reprend l'exemple 2 . On applique le test 3 en partageant l'intervalle [0,1] en 5 intervalles. L'histogramme représente la distribution des fréquences des intervalles.

```
b=Table[Random[],{50}]
ListPlot[b]
C=RangeCounts[b,{0.2,0.4,0.6,0.8,1}]
BarChart[c]
```
C={10,9,10,10,11}



Ce second exemple montre la distribution de 1000 nombres aléatoires sur 10 intervalles.
d={90,109,105,111,86,96,112,81,107,103}



Exercice 9: Générer des suites aléatoires à partir du langage C . Tester l'uniformité et l'indépendance.

Exercice 10 : Les générateurs suivants sont utilisés dans des langages bien connus.Tester leurs validité.

(i) **Java**. C'est un générateur utilisé pour implémenter la méthode nextDouble dans la classe java.util.Random de la bibliothèque Java (voir exercice 11). Il est basé sur

une réccurence linéaire de période 2^{48}, mais chaque sortie est obtenue de la manière suivante :

$$x_{i+1}=\left(25214903917x_i+11\right) \bmod 2^{48}$$
$$U_i=\left(2^{27}\lfloor x_{2i}/2^{22}\rfloor+\lfloor x_{2i+1}/2^{21}\rfloor\right)/2^{53}$$

(ii) **Unix**. Le générateur rand48 de Unix utilise la même récurrence, mais avec la sortie plus simple $U_i=x_i/2^{48}$.

(iii) **Microsoft Visual Basic**. $x_{i+1}=\left(1140671485x_i+12820163\right) \bmod 2^{24}$; $U_i=x_i/2^{24}$.

(iv) **Microsoft Excel**. Il est implémenté directement à partir des nombres U_i en arithmétique flottante $U_{i+1}=\left(9821.0U_i+0.211327\right) \bmod 1$.

(vi) **LCG16807**. Générateur à récurrence linéaire utilisé dans de nombreux langages de simulation , par exemple ARENA ou AutoMod(avec le multiplicateur 742938285) :

$$x_{i+1}=16807x_i \bmod (2^{31}-1) \quad ; \quad U_i=x_i/(2^{31}-1)$$

(vii) **MRG32k3a**. a remplacé le précédent. Il combine deux MRG (générateur récursif multiple d'ordre 3) et sa période est de longueur voisine de 2^{191}.

Exercice 11: Génération de nombres aléatoires en JAVA. Exercice facultatif si vous êtes intéressés par ce langage.(http://www.es.geneseo.edu/~baldwin/reference/random.html).

Dans Java, il existe diverses bibliothèques (packages) permettant de générer des nombres aléatoires pour différentes utilisations : jeux, simulation, cryptographie. Cela peut se faire dans la classe Random dont les instances sont des objets générateurs de nombres aléatoires.

Générer des suites aléatoires en utilisant les instructions énumérées ci-dessous, puis tester leur uniformité et indépendance. Concevez une Applet java permettant de tester l'uniformité et l'indépendance des suites aléatoires ainsi générées. Voir dans un premier temps l'une des Applet java : Random Numbers ou Uniform Applet sur le site http://ubmail.ubalt.edu/~harsham/simulation/sim.htm

Table 3. Nombres aléatoires en JAVA

Package	Utilisation et description
java.lang	StrictMath.randomNumberGenerator
	Math.randomNumberGenerator

java.math	BigInteger.(int numBits,Random rnd) Construit un grand entier aléatoire, uniformément distribué entre 0 et $2^{numBits}-1$
	BigInteger.(int bitLength, int certainty, Random rnd) Construit un grand entier aléatoire de longueur spécifiée bitLength qui est probablement premier
Java.util	Collections.r
	Collections.shuffle(List list, Random rnd) Permute aléatoirement la liste spécifiée en utilisant une source aléatoire spécifiée
java.security	java.security.SecureRandom Accés à un générateur de nombres pseudo-aléatoires forts.
Net.vattp.security	ESecureRandom Implémente des nombres aléatoires cryptographiquement sûrs.
RngPack	Autre package de générateurs de nombres aléatoires pour java, peu recommandé pour la cryptographie
randomX	Version récente de la classe java.util.Random :utilise des générateurs congruentiels, mais aussi des générateurs combinés (randomLEcuyer) ; randomHotbits ne génére pas des nombres aléatoires à l'aide d'un algorithme, mais plutôt à partir d'Internet (HotBitsserver) : donc, pas de racine à spécifier

Voici en guise d'illustration les nombres aléatoires utilisant la classe java.util.Random**.** Cela signifie que toute instruction commence par l'instruction :

```
import java.util.Random ;        ou        import java.util.* ;
```

qui assure l'accès à la classe « Random » dans la bibliothèque Java « java.util ». L'instruction

```
Random generator=new Random( ) ;
```

initialise le générateur avec le temps actuel pour racine (résolution de une milliseconde). L'instruction suivante par contre permet de spécifier la racine 19580427 :

```
Random generator2=new Random(19580427)
```

Pour générer un entier à partir d'un objet Random, envoyer le message « nextInt ». Il renvoie l'entier suivant du générateur de la séquence aléatoire : ces entiers sont uniformément distribués dans le domaine des entiers java. Par exemple, si on suppose que « generator » est une instance de Random

```
Int r=generator.nextInt( ) ;
```

Si on souhaite générer un entier compris entre 0 et une certaine borne supérieure. Par exemple on souhaite sélectionner un symbole au hasard parmi un ensemble de n symboles (ou indice dans une liste de n éléments). Les indices de cette liste dans Java sont numérotés de 0 à n-1. Si on attribue un paramètre entier n à « nextInt », on obtient en retour un entier de la distribution uniforme entre 0 et n-1 ; Si on utilise l'objet de générateur de nombre aléatoire pour générer la liste aléatoire d'indices, on utilisera l'instruction

```
Int randomIndex=generator.nextInt( n ) ;
```

Enfin, pour générer un nombre aléatoire réel uniformément distribué sur [0,1], on utilisera

```
Double r=generator.nextDouble( ) ;
```

Chapitre 5.
Méthodes de Simulation

5.1.Principe de la simulation.

L'idée de la simulation statistique ou méthode de Monte Carlo est très simple. Elle consiste à simuler sur ordinateur (analogique ou numérique) le processus i.e. réaliser des expériences artificielles, puis en effectuer un traitement statistique (voir schéma du chapitre 1). A l'entrée du système, on décrit les facteurs aléatoires en générant des variables aléatoires et des événements selon des lois de probabilités données, et compte tenu du mode de fonctionnement du système. On obtient en sortie des estimations statistiques des caractéristiques (mesures de performance) nécessaires aux diverses décisions de l'utilisateur. Les caractéristiques du processus réel et simulé ne coïncident pas tout à fait. On peut cependant estimer la précision de leur proximité, et l'améliorer (par exemple en fixant le nombre de simulations) à l'aide des méthodes de statistique mathématique.

Soit à approcher la grandeur μ qu'il est difficile d'évaluer par une méthode mathématique directe. Considérons alors un espace $(\Omega,\Im,P)$, où $\Omega=\{\omega\}$ est l'espace des événements élémentaires, $\Im$ la σ-algèbre des événements, et P une probabilité sur $\Im$. On suppose qu'il existe une variable aléatoire X, c'est-à-dire une fonction $X=f(\omega)$, $\Im$-mesurable, telle que son espérance mathématique $E(X)$ approche suffisamment la grandeur μ (avec une précision acceptable). L'algorithme de calcul de $f(.)$ doit être plus simple (en complexité algorithmique, temps machine,...) que celui du calcul de la grandeur μ à l'aide des méthodes connues : pour que la simulation ait un sens. Générons un échantillon $\omega_1,\omega_2,...,\omega_n$ d'éléments de Ω de distribution P, alors pour n suffisamment grand, et en vertu de la loi des grands nombres :

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n}f(\omega i)=\frac{1}{n}\sum_{i=1}^{n}X_i=E(X)=\mu \qquad \text{(p.s.)}$$

Par conséquent, pour n suffisamment grand, on peut utiliser l'approximation :

$$\mu\approx\mu_n=\frac{1}{n}\sum_{i=1}^{n}X_i$$

La précision d'une telle approximation peut être estimée à l'aide du théorème central limite: pour n suffisamment grand, μ_n est asymptotiquement de loi normale de moyenne $\mu=E(X)$ et de variance $\sigma^2=Var(X)/n$ .On peut alors déterminer un intervalle de confiance de niveau $1-\alpha$ (α petit) pour la grandeur μ :

$$\mu_n - u_{\alpha/2}\sigma/\sqrt{n} \prec \mu \prec \mu_n + u_{\alpha/2}\ \sigma/\sqrt{n}$$

où $\Phi(u_\alpha)=1-u_{1-\alpha}$, $\Phi(.)$ étant la fonction de répartition de la loi normale standard $N('0,1)$ de moyenne nulle et de variance unité. On constate ainsi que la précision de la méthode de Monte Carlo est de l'ordre de $1/\sqrt{n}$. En d'autres termes, pour diminuer de 10 fois la précision, il faut disposer de 100 fois plus d'observations dans l'échantillon de la grandeur μ. Ceci explique le fait que la simulation par la méthode de Monte Carlo ne soit pas d'une trop grande précision par rapport à d'autres méthodes. Nous présentons plus loin un tableau comparatif qui montre que cet inconvénient peut être compensé par d'autres critères de performance (par exemple le gain en temps machine pour l'intégration numérique).

Exemple 1. (Expérience de Buffon pour le calcul de π). Considérons le plan (xOy) hachuré de droites parallèles à l'axe Ox et distantes d'une longueur L, et dans lequel on jette une aiguille de longueur l.

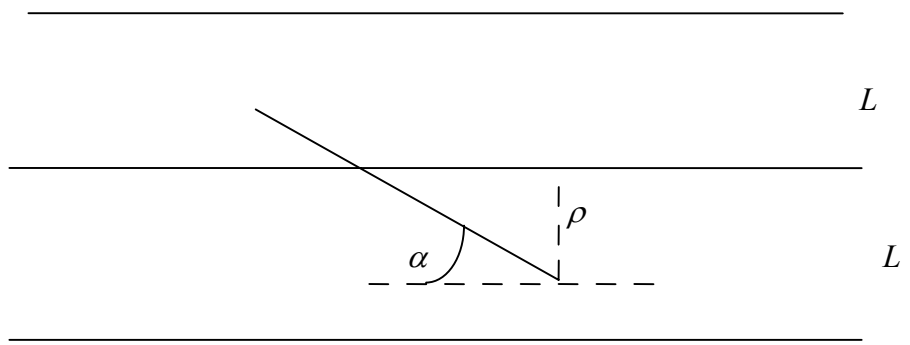


Figure 3.

Le jet de l'aiguille peut être assimilé au jet d'un point aléatoire (α,ρ) dans le rectangle de côtés L et π, $0\le\alpha\le\pi$, $0\le\rho\le L$.

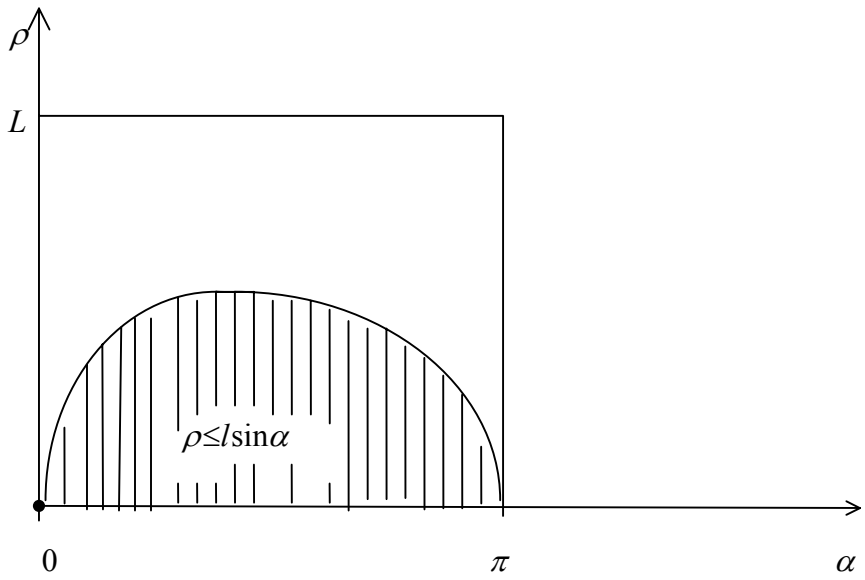


Figure 4.

L'événement A =« l'aiguille coupe au moins une des parallèles » est équivalent sur la seconde figure à l'événement « α et ρ sont tels que $\rho \le l \sin\alpha$ ». Par conséquent, la probabilité de l'événement A est la probabilité pour que le point aléatoire (α,ρ) tombe dans la partie hachurée (sur la figure 4):

$$P(A)=\frac{S(A)}{S}$$

où S est la surface du rectangle : $S=\pi L$ et $S(A)$ la surface sous la courbe s, ρ=l sin α . Soit,

$$S(A)=\int_0^{\pi} l sin\alpha d\alpha = 2l$$

Par conséquent, $$P(A)=\frac{2l}{\pi L}$$

Il s'agit d'obtenir un échantillon de taille n du couple aléatoire (α,ρ), soit (α_1,ρ_1), (α_2,ρ_2) , ..., (α_n,ρ_n). En vertu de la loi des grands nombres, la probabilité $P(A)$ peut être approchée par la fréquence relative de l'événement A

$$P(A)\approx\frac{n(A)}{n}$$, pour n suffisamment grand.

Ici $n(A)$ est le nombre de fois où l'aiguille coupe l'une des lignes. On peut alors déduire la valeur a posteriori de π à partir de la relation approximative:

$$\frac{2l}{\pi L}\approx\frac{n(A)}{n} \qquad \text{ou ; encore} \qquad \pi\approx\frac{2l}{L}\frac{n}{n(A)}$$

A l'aide de cette expérience, le naturaliste Buffon en 1717 a pu simuler le calcul de π en procédant à 16000 lancers de l'aiguille. Le tableau ci-dessous résume d'autres expériences postérieures. Au lieu de procéder à des lancers réels, il est plus simple aujourd'hui de réaliser cette expérience sur ordinateur en créant des échantillons artificiels indépendants:

$\alpha_1,\alpha_2,..,\alpha_n$ issu d'une loi uniforme sur $(0,\pi)$

$\rho_1,\rho_2,...,\rho_n$ issu d'une loi uniforme sur $(0,L)$

Expérimentateur	Année	nombre de lancers de l'aiguille	valeur expérimentale de π
Wolf	*1850*	*5 000*	*3,1596*
Smith	*1855*	*3 204*	*3,1553*

Fox	*1894*	*1 120*	*3,1419*
Lasarini	*1901*	*3 408*	*3,1415929*

Exemple 2. Soit à calculer l'espérance mathématique d'une variable aléatoire X, de fonction de répartition $F(x)=P(X\leq x)$. Pour fixer les idées, supposons que X est la durée de vie d'un équipement, $R(x)=1-F(x)$ sa *fonction de fiabilité* (ou *probabilité de survie*, parfois notée $\overline{F}(x)$). On cherche à estimer la durée de vie moyenne (connue en fiabilité par *MTTF* ou *MTBF*), qui est égale à $T_0=E(X)$.

L'approche directe consisterait à procéder à des expérimentations physiques de laboratoire qui peuvent revenir très cher ou à des observations du système en exploitation, ce qui demanderait beaucoup de temps. Il est plus simple de procéder par simulation en commençant par créer artificiellement une suite de nombres aléatoires $\{x_1,x_2,...,x_n\}$ à l'aide d'un ordinateur et pouvant être considérés comme la réalisation d'un échantillon issu de la variable aléatoire X.

En vertu de la loi des grands nombres, et des résultats de la statistique, la moyenne empirique

$$\overline{x}=\frac{1}{n}\sum_{i=1}^{n}x_i \qquad (5.1)$$

est un «bon» estimateur du MTTF (absence de biais, convergence, efficacité). On utilisera alors l'approximation:

$$MTTF=E(X)\approx\overline{x}$$

Exemple 3. Soit p la probabilité d'atteindre une cible au cours d'un tir. Si l'on procède à 10 tirs sur la cible, quelle est la probabilité de faire mouche un nombre pair de fois? Il est connu que la probabilité de faire mouche $2k$ 2k fois s'obtient à partir de la loi binomiale:

$$P_{2k}=C_{10}^{2k}p^{2k}(1-p)^{10-2k} \qquad 1\leq k\leq 5 \qquad (5.2)$$

d'où la probabilité cherchée :

$$P=P\{2,4,6,8,10\}=\sum_{k=1}^{5}C_{10}^{2k}p^{2k}(1-p)^{10-2k} \qquad (5.3)$$

Supposons que l'on ne dispose pas de tables de la loi binomiale ou que cette formule soit inconnue. La probabilité P peut être calculée au moyen de deux procédés:

(ii) On procède à des séries réelles de 10 tirs chacune sur la cible. Pour chaque série de 10 tirs, on relève la fréquence relative des issues paires $f_{2k}^{(i)}$, où i est le numéro de la

série, $i=1,2,...,n$. Les probabilités (5.2) peuvent être estimées par: $\frac{1}{n}\sum_{i=1}^{n} f_{2k}^{(i)}$. La probabilité cherchée (1.3) s'obtient alors par sommation des valeurs paires.

$$P \approx \sum_{i=1}^{5} \left\{ \frac{1}{n} \sum_{i=1}^{n} f_{2k}^{(i)} \right\} \tag{5.4}$$

(ii) Il est plus simple pour éviter des tirs réels de simuler ces expériences sur ordinateur. On génère d'abord les valeurs d'une variable aléatoire de loi uniforme sur l'intervalle $[0,1]$. Il est clair que $P(X \le p) = p$. A chaque étape de la simulation, on comparera donc la réalisation de la variable aléatoire $f_{2k}^{(i)}$ avec la probabilité p . Si $X \le p$, on considère que le tir a atteint la cible, dans le cas contraire, c'est l'échec. On effectue ainsi des séries de 10 expériences artificielles, et on fait le décompte des succès «pairs». Si le nombre de séries est suffisamment élevé, la fréquence obtenue sera proche de la valeur donnée par la formule (5.3).

Ces deux exemples illustrent assez bien le principe de la méthode de Monte Carlo. D'habitude, par méthode de Monte Carlo, on entend en général la résolution de problèmes déterministes (calcul d'intégrales, résolution d'équations différentielles) à partir de nombres au hasard.

Exemple 4: Calcul d'une intégrale par la méthode de Monte Carlo

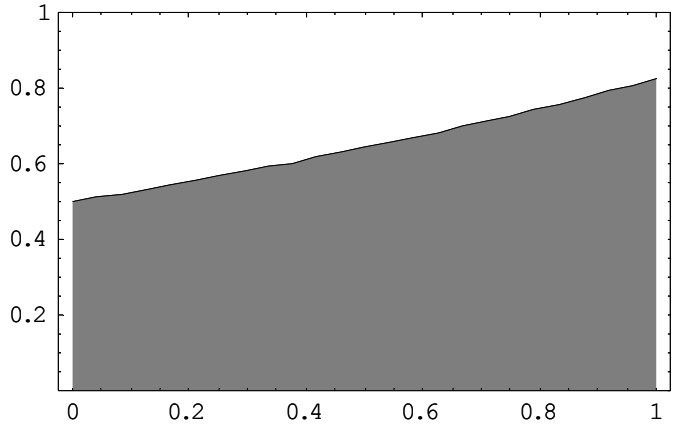
Soit à calculer l'intégrale définie : $I = \int_a^b f(x)dx$

où f(x) est une fonction connue, intégrable sur (a,b) vérifiant $c \le f(x) \le d$. On peut toujours se ramener au calcul de l'intégrale :

$$I = \int_0^1 f(t)dt \ , 0 \le f(t) \le 1 \ , t \in [0,1].$$

sur l'intervalle $[0,1]$ moyennant une certaine transformation.

Soit par exemple à évaluer l'intégrale $I = \int_0^1 0.5e^{0.5}dx$. Il s'agit de déterminer l'aire comprise entre la courbe $y = f(t)$, l'axe des abscisses et les droites $t = 0$ et $t = 1$. C'est la surface hachurée sur le graphe suivant.

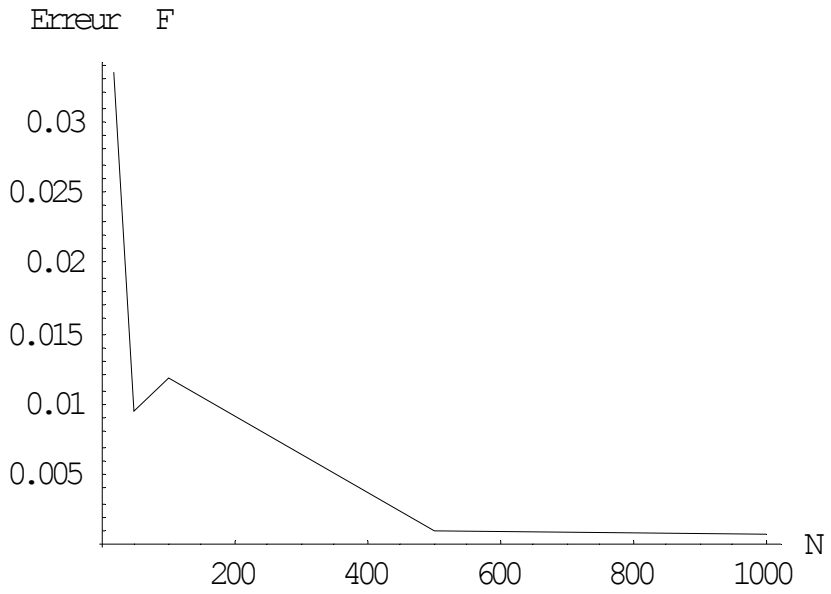


L'idée consiste à ramener un problème mathématique pur de calcul d'intégrale (on suppose qu'on est profane et qu'on ne sait pas calculer cette intégrale par les méthodes traditionnelles d'analyse). On imagine alors une expérience aléatoire qui consiste à jeter un point aléatoire dans le carré unité 1×1. On note que la valeur de l'intégrale I (la partie hachurée S) n'est autre que la probabilité $p(S)$ pour ce point aléatoire de tomber dans la surface S. En vertu de la loi des grands nombres, cette probabilité peut-être estimée par la fréquence des points qui tombent dans la surface S.

La valeur approchée par une méthode numérique de quadrature classique (de type Trapèze) : $J=0.648721$. La table suivante montre les résultats de plusieurs expériences de simulation en faisant varier la taille de l'échantillon :

Taille de l'échantillon N	Valeur simulée U	Erreur F=|J-U|
20	*0.682202*	*0.0334803*
50	*0.639142*	*0.00957883*
100	*0.636924*	*0.0117974*
500	*0.64764*	*0.00108133*
1000	*0.649483*	*0.000762002*

Le graphe ci-dessous montre l'évolution de l'erreur lorsque la taille de l'échantillon N augmente.



Une méthode alternative est la suivante. Considérons la fonction :

$$\Phi(x,y)=\begin{cases}1, & si\ y<f(x)\\ 0, & si\ y\geq f(x)\end{cases}$$

L'égalité $\Phi(x,y)=1$ signifie que le point de coordonnées (x,y) se trouve dans la surface hachurée de la figure ci-dessus. On peut alors écrire :

$$\int_0^1 f(t)dt=\int_0^1\int_0^1 \Phi(x,y)dxdy$$

ou encore

$$\int_0^1 f(t)dt=E\{\Phi(\xi,,\eta)\}$$

où ξ et η sont des variables aléatoires indépendantes uniformément distribuées sur $[0,1]$.

On peut simuler la valeur de l'intégrale I de la manière suivante. Générons une suite aléatoire $\{U_n\}$ issue de la loi $U[0,1]$, et calculons successivement les valeurs de la fonction Φ

$$\begin{aligned}\Phi_1&=\Phi(U_1,U_2)\\ \Phi_2&=\Phi(U_3,U_4)\\ \Phi_3&=\Phi(U_5,U_6)\\ \Phi_4&=\Phi(U_7,U_8)\ \ldots\ldots\end{aligned}\qquad (5.5)$$

En vertu de la loi des grands nombres, on aura l'approximation :

$$I=\int_0^1 f(t)dt\approx\frac{1}{n}\sum_{i=1}^{n}\Phi i$$

pour n suffisamment grand.

Interprétation graphique. Considérons un point $(\xi,,\eta)$ jeté au hasard dans le carré $[0,1]\otimes[0,1]$. (cf. figure 1). La probabilité de tomber en dessous de la courbe $y=f(t)$ est égale à

$$p=\frac{surface\ \ hachurée}{surface\,du\ carré}=\frac{J}{1}=J$$

soit la valeur cherchée de l'intégrale: $p=I$. Soit l'expérience suivante: on choisit un couple de nombres au hasard $(\xi,,\eta)$ de loi $U[0,1]$, et on vérifie la condition: $f(\xi)>\eta$. Si cette condition est remplie, le point aléatoire $(\xi,,\eta)$est en dessous de la courbe et $\Phi(\xi,,\eta)=1$. Après avoir répéter cette expérience n fois, on obtient la suite de valeurs (5.5), et la moyenne empirique :

$$\frac{1}{n}\sum_{i=1}^{n}\Phi i=\frac{M}{n}$$

représente la fréquence relative des points qui se trouvent en dessous de la courbe (M représente le nombre de fois où l'inégalité $f(\xi)>\eta$ est remplie). Par suite, en vertu de la loi des grands nombres :

$$\frac{M}{n}\xrightarrow[n\to\infty]{p}p=I$$

De manière générale, soit à calculer l'intégrale multiple

$$\mathrm{x}\ I=\int_B f(x)dx$$

Si le domaine B est fini, on peut l'inclure dans un parallélépipède R de côtés parallèles aux axes des coordonnées. Représentons l'intégrale I sous la forme

$$I=\int_B f(x)dx=\frac{1}{v(R)}\int_B f(x)v(R)dx=\frac{1}{v(R)}\int_{\Re} f(x)\chi_B(x)v(R)dx$$

où $v(R)$ est le volume du parallélépipède R, et $\chi_B(x)$ est l'indicateur du domaine B: $\chi_B(x)=1$ si $x\in B$, $\chi_B(x)=0$ sinon. L'intégrale I peut être comprise comme l'espérance mathématique de la fonction $\Phi(U)=f(U)\chi_B(U)v(R)$, U de loi uniforme sur le parallélépipède R. Par suite, selon la loi des grands nombres, la moyenne empirique des observations de la variable aléatoire $\Phi(U)$ converge vers son espérance mathématique

$$\frac{v(R)}{n}\sum_{i=1}^{n} f(x_i)\chi_B(x_i) \underset{n\to\infty}{\to} E\{\Phi(U)\} = \int_B f(x)dx$$

où x_i est la suite des valeurs échantillonnées, équidistribuées sur R et indépendantes.

On utilise souvent une variante plus simple sur un domaine arbitraire B. Soit $p_\xi(x)$ une fonction de densité qui ne s'annule en aucun point du domaine B, par exemple la densité normale non dégénérée. On peut alors représenter l'intégrale sous la forme :

$$I=\int_B f(x)dx = \int_B \frac{f(x)}{p_\xi(x)} p_\xi(x)dx = \int_{-\infty}^{\infty} \frac{f(x)\chi_B(x)}{p_\xi(x)} p_\xi(x)dx = \mathbf{E}\{\Phi(\xi)\}$$

où $\Phi(\xi)=f(\xi)\chi_B(\xi)/p_\xi(\xi)$ et ξ est de densité p_ξ . A l'aide d'une suite générée selon la densité de ξ on approche I par la moyenne empirique des valeurs de $\Phi(\xi)$:

$$I \approx \frac{1}{n}\sum_{i=1}^{n} \frac{f(\xi_i)\chi_B(\xi_i)}{p_\xi(\xi_i)}$$

Remarques :

1. Fiction ou réalité, il semble que l'appellation Monte Carlo" soit inspirée de l'intérêt qu'aurait eu Metropolis (concepteur d'algorithme d'optimisation stochastique) pour les jeux de hasard qui font la réputation des casino de la ville de Monte Carlo (principauté de Monaco).

2. Les méthodes de Monte Carlo sont évidemment moins précises que les procédures numériques d'intégration classiques (méthodes de quadrature). Le temps de calcul s 'améliore considérablement cependant dans le cas d'intégrales multiples (relativement aux méthodes de quadratures). (cf . note annexe).

5.2.Nombres Aléatoires.

5.2.1. Introduction. L'outil de base de la simulation est la source capable de produire des nombres aléatoires (ou nombres au hasard), c'est-à-dire une suite U_1, U_2,..., U_n, ... de variables aléatoires indépendantes et uniformément distribuées sur l'intervalle [0,1]. La suite $\{U_i\}$ est ainsi interprétée comme une série de réalisations d'une variable aléatoire U de loi uniforme sur [0,1] (on notera $U\in U[0,1]$). On peut à partir d'une telle suite générer toute autre suite de variables aléatoires de loi arbitraire, ou encore tout processus ou fonction aléatoire. Les nombres aléatoires sont utilisés également pour les jeux vidéos, ainsi qu'en cryptographie.

En pratique, on utilise des procédés physiques (mécaniques) ou algorithmiques permettant de produire de telles suites aléatoires. Ces dernières sont en fait pseudo-aléatoires, c'est-à-dire qu'elles sont déterministes, mais possèdent des propriétés statistiques identiques à celles que posséderait une suite réellement aléatoire.

5.2.2.Générateurs physiques.

L'exemple le plus simple, connu de tous ceux qui sont familiers du calcul des probabilités, est le jeu de « Pile » ou « Face » permettant de générer une variable aléatoire de Bernouilli qui prend les valeurs 0 ou 1 avec des probabilités identiques, égales à 1/2. Le jet d'un dé « parfait » permet de générer une variable aléatoire à valeurs dans {1,2,3,4,5,6} avec des probabilités identiques, égales à 1/6. La suite de nombres au hasard peut être obtenue également à l'aide d'une roulette telle qu'on peut en voir dans les fêtes foraines.

On peut utiliser également des dispositifs mécaniques plus sophistiqués tels que la machine servant au tirage du loto ou de Kora+. Ainsi, pour générer une variable binaire on a longtemps utilisé des sources de particules radioactives ; un compteur dénombre les particules détectées durant un temps Δt on considère alors la variable binaire $Y=1$ si le nombre de particules détectées durant le temps Δt est pair; $Y=0$ si ce nombre est impair. Un autre procédé utilise le niveau de bruit d'une valve électronique. La valeur du voltage u(t) est un processus aléatoire qu'on observe à des temps discrets $t_1,t_2,...,t_i,...$ On choisit un niveau de section c de manière « adéquate », par exemple tel que $P(U_i=1)$ soit le plus proche possible de 1/2, et on pose:

$$U_i=\begin{cases}0, & \text{si} \quad u(t_i)\le c,\\ 1, & \text{si} \quad u(t_i)\ge c,\end{cases}$$

La logique de création des U_i peut être plus complexe afin de réduire le biais ou garantir d'autres propriétés statistiques de la loi U[0,1].

Voici à titre d'exemples quelques procédés physiques utilisés pour la génération de nombres aléatoires : sources radioactives (voir le principe ci-dessus), effet quantique dans les semi-conducteurs, turbulence de l'air, microphone, camera vidéo ; souris ou horloge d'un ordinateur etc…

L'intérêt actuel de telles sources reste plutôt motivé par d'autres objectifs que la simulation. C'est par exemple le cas en cryptographie où les différentes méthodes de cryptage nécessitent la génération de nombres aléatoires produits par de telles sources. (voir par exemple http://world.std.com).

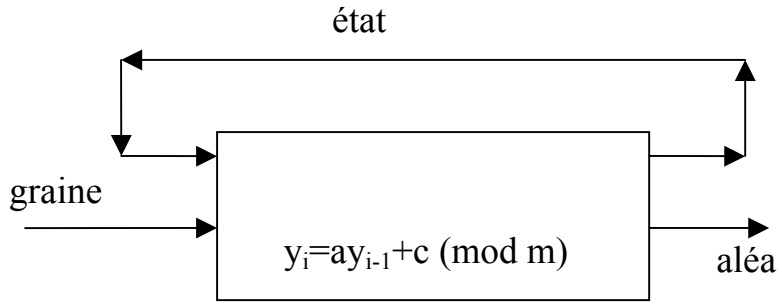
Les nombres produits à l'aide de telles sources sont appelés parfois nombres vraiment aléatoires (par opposition aux nombres pseudo-aléatoires du paragraphe suivant). Cependant on a toujours détecté des erreurs également dans de telles suites.

5.2.3. Générateurs algorithmiques ou pseudo-aléatoires.

Les suites produites par des procédés physiques ont été longtemps (et sont toujours) acceptées comme aléatoires. Cependant ils sont coûteux et les suites obtenues ainsi présentent des biais et dépendances. Bien que produisant également des nombres « pseudo-aléatoires », les générateurs algorithmiques ont l'avantage d'être simples à réaliser sur ordinateur. C'est en ce sens que la simulation a été l'une des toutes premières applications des premières générations d'ordinateurs dès 1940. Il existe une multitude de générateurs algorithmiques de nombres au hasard. Certains sont mieux testés que d'autres. La complexité d'un générateur ne conduit pas forcément à une suite plus « aléatoire » que celle d'un générateur plus simple. On conseille d'ailleurs en général d'utiliser un algorithme simple et bien assimilé. L'exception

reste peut-être la cryptographie où on doit imposer une exigence supplémentaire : l'algorithme ne doit pas pouvoir être « casser » par l'adversaire.

Lorsqu'on génère une suite $\{U_i\}$ par un procédé algorithmique, on doit avant tout s'assurer que l'algorithme choisi possède une bonne forme mathématique simple et facilement programmable sur ordinateur. L'algorithme doit donc être récursif : $U_{i+1}=f(U_i, U_{i-1},...,U_{i-k})$ donnée à valeurs dans [0,1], qui peut avoir une forme plus ou moins « chaotique ». Pour les ingénieurs s'intéressant à la cryptographie ou à la simulation, le générateur pseudo-aléatoire est un objet décrit par un automate fini dont l'état initial est la graine (la semence, la racine, la clef).



La suite générée à l'aide de l'algorithme ci-dessus sera dans tous les cas **pseudo-aléatoire** car produite par un moyen déterministe. Les algorithmes récurrents possèdent un autre inconvénient qui réside dans le fait que la suite $\{U_i\}$ sera toujours périodique. C'est pourquoi, l'utilisation pseudo-aléatoire d'une telle suite ne pourra se faire que sur une succession d'épreuves inférieure à la période. La question qui se pose alors est de savoir déterminer une période suffisamment grande pour garantir un nombre raisonnable de termes de la suite. Malgré les inconvénients cités, les nombres pseudo-aléatoires obtenus à partir de procédés récurrents sont très commodes pour la simulation:

- l'algorithme est facilement programmable, et nécessite un faible espace mémoire.

- toute suite de nombres peut être répétée autant de fois qu'on le désire, et à partir de n'importe quel rang; il suffit pour cela de donner un nombre initial U_0.

- une telle suite peut être testée sur l'uniformité une seule fois afin de pouvoir l'utiliser avec un niveau acceptable de confiance.

- rapidité pour garantir une utilisation multiple des nombres générés.

- l'étude des performances de telles suites récursives peut être réalisée à l'aide de certaines méthodes mathématiques développées (théorie des nombres, théorie des probabilités,…).

- La plupart des langages informatiques et logiciels sont dotés d'un sous-programme (ou routine) de génération de suites aléatoires uniformes sur [0,1] .

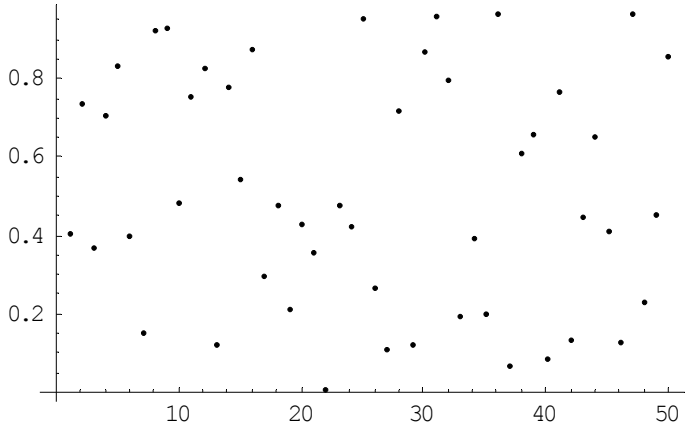
Table 1. Exemple de routines et fonctions utilisées dans les langages et logiciels usuels.

Langage	Routine ou macro	Type de nombre produit	Observation (Type de machine ou autre…)
FORTRAN	RANDU ; RAND	U∈[0,1]	IBM 360/370
FORTRAN 77	GOSCAF RANF	U∈[0,1]	NAG Library CDC Cyber 174 Vax (machine 8 à 64 bits)
PASCAL ou C	RANDOM RANDOM(n)	U∈[0,1] entier§n	
C++	rand ( )	Nombre entre 0 et RAND_MAX	
	srand		
	RANROT		
BASIC	RND ou RAND	U∈[0,1]	
MATLAB	rand randn rand(n,m)	U∈[0,1] X∈N(0,1) Matrice aléatoire	
MATEMATIKA	Random[ ] Random[Integer] Random[Real,a,b] Random[Integer,a,b] SeedRandom[ ]	U∈[0,1] U=0 ou 1 réele[a,b] entiere[a,b] initialisation	Avec proba= 1/2
Visual Basic	Rnd		
WORK PLACE	RANDOM NUMBERS	U∈[0,1]	
EXCEL	ALEA	U∈[0,1]	
BSAFE (boîte à outils cryptographiques)	MD5 Random SHA1 Random		
UNIX	rand() random() rand48()		
MAPLE	randomize uniform rand rand[r] rand(a..b) rand(n) randvector(n,entries=p)	Initialisation U∈[0,1] Réel aléatoire Entier aléatoire entiere(a,b) ‹ rand(0..n-1) vecteur aléatoire	à 12 chiffres de dimension n
	MakeRandom (r)	entier	
	MakeRandom(a.b)	Entier entre a et b	
	RandUniform(a.b)	U[a,b]	
STATISTICA	Rnd(x) Uniform(x) Normal(x)	réel∈U[0,x[U∈U[0,x[réeleN(0,x^2)	 x=écart-type
ORACLE	Package DBMS_RANDOM		

Exemple 5: Ce petit programme avec MATEMATIKA vous liste une séquence de 50 nombres pseudo-aléatoires et les représente dans le plan.

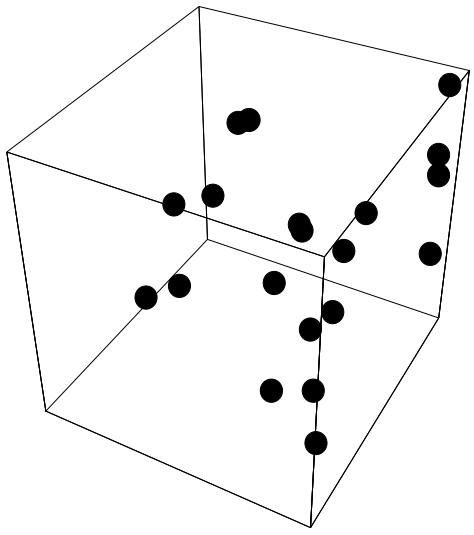
```
B=Table[Random[],{50}]
ListPlot[b]
```

0.40419	0.735378	0.369251	0.70511	0.833578	0.397475	0.151053	0.925697
0.928252	0.481479	0.757547	0.82988	0.124692	0.777978	0.544763	0.878684
0.299862	0.478652	0.211158	0.430693	0.358442	0.005522	0.479885	0.426198
0.954252	0.270145	0.110634	0.721088	0.120675	0.87267	0.959581	0.795391
0.192522	0.39119	0.202034	0.965511	0.067830	0.613212	0.657272	0.086827
0.767969	0.13456	0.446113	0.65134	0.409527	0.129038	0.966229	0.229936
0.455275	0.858893						

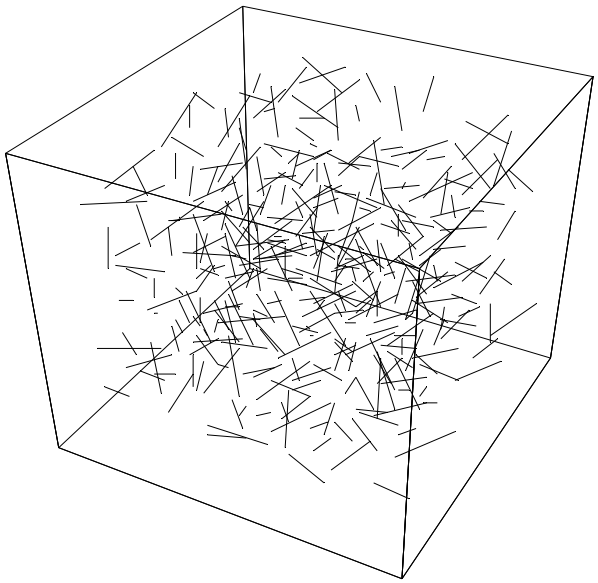


Exemple 6: En raison de l'importance des nombres aléatoires, la plupart des développeurs de logiciels prévoit dans leurs produits des générateurs de nombres au hasard adaptés à différentes situations. En guise d'illustrations, l'exercice 9 de la série montre comment générer des nombres aléatoires en Java.

Exemple 5': Le graphe suivant représente 20 points aléatoires dans l'espace 3D (3 dimensions)



Exemple 5 ’’ : Le graphique suivant représente des vecteurs aléatoires en 3D.



5.2.4.Générateurs usuels.

Nous allons voir maintenant comment les générateurs algorithmiques ci-dessus ont été construits.

5.2.4.1.Générateurs binaires

Au lieu de tirer directement une variable uniformément distribuée U[0,1], on peut adopter l’approche suivante qui consiste à représenter tout nombre dans un système de base b, par une suite de chiffres. La méthode consiste alors à tirer successivement chacun des chiffres de la suite avec une approximation suffisante. Cette approche a l’avantage d’exploiter le Hardware spécifique de l’ordinateur utilisé. En système binaire de base b=2, tout nombre U peut être représenté par une suite de chiffres pouvant prendre les valeurs 0 ou 1 :

$$U=z_1.2^{-1}+z_2.2^{-2}+...+z_n.2^{-n}+.... \quad (5.6)$$

On propose ainsi de générer les valeurs d’une variable aléatoire binaire telle que:

$$\mathbf{P}(z_i=0)=\frac{1}{2}=\mathbf{P}(z_i=1) \quad (5.7)$$

Ceci peut être réalisé sur ordinateur à l’aide d’un procédé physique ou algorithmique. On peut montrer que la variable U construite à l’aide de (5.6) et vérifiant (5.7) est de loi uniforme U[0,1]. La propriété (5.7) doit être comprise au sens où chaque terme z_i de la suite (5.6) apparaît avec une proportion asymptotique de ½. La représentation (5.6) ne peut être réalisée pratiquement sur ordinateur qui ne peut exprimer les nombres que sous forme de « mots » d’un nombre limité de chiffres (ou de bits).

Soit une machine binaire donnant des mots de k bits chacun. Elle ne pourra exprimer que 2^k mots différents sur [0,1]

$$0,\ \frac{1}{2^k}, \frac{2}{2^k}, \ldots, \frac{2^{k-1}}{2^k}$$

qui sont les réalisations possibles du nombre pseudo-aléatoire $\tilde{\tilde{U}}$. S'il y a équidistribution, chacun de ces nombres $\tilde{\tilde{U}}$ est de la forme:

$$\tilde{\tilde{U}}=\sum_{i=1}^{k} z_i.2^{-i}$$

avec une probabilité identique 2^{-k} (en fait la fréquence asymptotique). On montre que $E(\tilde{\tilde{U}})Ø1/2$ et $var(\tilde{\tilde{U}})Ø1/12$ qui sont respectivement l'espérance mathématique et la variance d'une variable uniforme sur [01].

Pour une machine décimale $m=10^b-1$.

5.2.4.2.Générateur de Fibonacci. Il est de la forme

$$y_{i+1}=y_i+y_{i-j} \pmod m$$

Ce générateur est juste cité ici pour l'histoire, car il a été constaté qu'il ne produisait pas de nombres pseudo-aléatoires satisfaisants, notamment en cryptographie.

5.2.4.3.Générateurs congruentiels.

C'est en fait toute une famille de générateurs définis par

$$y_i=(\lambda y_{i-1}+C) \bmod m$$

où λ, C et m sont des entiers. Ce procédé signifie que y_i est le reste de la division de $\lambda y_{i-1}+C$ par m, où m est un nombre entier de grande taille (généralement une puissance de 2 pour les machines binaires (où $m=2^{k-1}$, k étant la longueur d'un mot machine en bits, le premier bit étant réservé au signe); λ est un nombre entier compris entre 0 et m-1 ; y_0 est appelé la racine (ou le germe) du générateur. En guise de nombre pseudo-aléatoire, on utilise les nombres $U_i=y_i/m$ qui sont équidistribués sur [0,1] lorsque $i\to\infty$.

Pour C=0, on obtient le générateur multiplicatif (de Lehmer) qui imite bien l'effet d'une roulette. Pour C ∫ 0, on obtient le générateur mixte.

La plupart des routines et fonctions de la table 1 utilisent des générateurs congruentiels. A titre d'exemple, la table 2 donne les valeurs des paramètres du générateur λ, m,C, y_0 utilisées dans ces langages.

Table 2.Exemples de valeurs des paramètres du générateur congruentiel pour les langages usuels.

Langage	Fonction ou routine	m	λ	C	Type de machine
FORTRAN	RANDU	2^{31}	$2^{16}+3$	0	IBM 360/370

FORTRAN NAG	G05caf	2^{59}	13^{13}	0	
FORTRAN IMSL	GGUBT	2^{31}-1	397204094	0	(machine 32-bits)
		2^{35}-31	5^5	0	(machine 36 bits)
FORTAN 77		2^{32}	69069	1	VAX /VMS (32 bits)
FORTRAN 77	RANF( )	2^{48}	44485709377909	0	CDC Cyber 174 (60 bits)
NAG Library	G05CAF	$_2{}^{59}$	5^{13}	0	Machine 60 bits
SIMULA		67099547	2^{13}	0	
GPSS, SIMSCRIPT, SLAM, SIMAN, ARENA	2^{31}-1ou 2^{48}				
JAVA	Java.util.random	2^{48}	Voir exercices		
Visual basic		2^{24}	1140671485		
Excel			Voir exercices		

Pour éviter les cycles (récurrence de la même séquence de nombres aléatoires), le générateur doit avoir une période la plus grande possible (cela dépend du choix de λ, m et de la racine).

Exemple7 : (i) si $m=2^b$ c ∫ 0, alors $p_{max}=m=2^b$. Il faut pour cela que c soit premier avec m et $\lambda=1+4k$.

(ii) si $m=2^b$ c=0, alors $p_{max}=m/4=2^{b-2}$. Il faut pour cela que la racine soit paire, et $\lambda=1+4k$. ou $\lambda=5+8k$

(iii) si m est premier et c=0, alors $p_{max}=m-1$. Il faut que λ soit tel que le plus petit k tel que λ^k-1 est divisible par m soit égal à k=m-1.

La période est souvent déterminée expérimentalement.

5.2.4.4.Générateurs registre mobiles ou à décalage.(shift-register).

Ces générateurs utilisent un hardware spécial utilisant une séquence de bits pseudo-aléatoires en enregistrant les d derniers bits $b_{i-1},...,b_{i-d}$ tels que $b_i=f(b_{i-1},...,b_{i-d})$ où f(.) est une certaine fonction de $\{0,1\}^d \rightarrow \{0,1\}$. Le générateur est de la forme:

$$U_i=0.b_{iL}b_{iL+1}...b_{iL+M}$$

où L et M sont des entiers tels que L>M>0, et la fonction f est généralement choisie

$$b_i=(a_1b_{i-1}+...+a_db_{i-d}) \bmod 2 \quad \text{où } a_1,...,a_d \text{ sont des constantes binaires.}$$

Lorsqu'on utilise un algorithme récurrent, on doit également s'assurer que la suite produite $\{U_i\}$ possède bien des propriétés voisines de celle d'une suite aléatoire et uniforme sur [0,1].

5.2.4.5. Générateurs cryptographiquement sûrs. Dans les applications liées à la sécurité des systèmes (militaires, système de paiement électronique…), on exige que le générateur soit crypto graphiquement sûr. Cela exclut les générateurs évoqués ci-dessus et disponibles dans les langages et logiciels peu sûrs contre les attaques et qui peuvent être décryptés aisément. L'algorithme générateur doit être dans ce cas conçu de telle manière que l' « adversaire » ne puisse deviner le bit suivant avec une probabilité supérieure à (1/2)+e où e décroît exponentiellement avec un certain paramètre de sécurité s (en général la longueur du générateur). Il existe d'autres manières de définir de tels générateurs. (http://world.std.com).

5.2.5. Validation des générateurs.

Enumérons quelques critères permettant de vérifier que la suite générée $\{U_i\}$ est bien aléatoire et issue d'une variable aléatoire Y de la loi uniforme U[0,1].

D'abord, si une telle suite est bien issue de la loi uniforme sur [0,1], alors en vertu de la loi des grands nombres, la moyenne empirique :

$$\bar{U}=\frac{1}{n}\sum_{i=1}^{n}U_i$$

doit converger en probabilité vers 1/2. On obtient ainsi la première exigence :

Test 1. Si la suite $\{U_i\}$ est tronquée à la valeur n (assez grande), alors $\bar{U}$ doit être voisin de 1/2.

De la même manière, on obtient l'exigence suivante sur la variance empirique.

Test 2 $$S^2=\frac{1}{n}\sum_{i=1}^{n}(U_i-\bar{U}) \xrightarrow{p} 1/12$$

Pour la loi uniforme sur [0,1], on doit avoir $\mathbf{P}\{U\in[a,b]\}$=b-a, $\forall$ $[a,b]\subset[0,1]$. En vertu de la loi des grands nombres, la proportion de valeurs U_i dans [a,b)

$$\lim_{n\to\infty} \frac{V[a,b)}{n}=\mathbf{P}\{U\in[a,b)\}=b-a$$

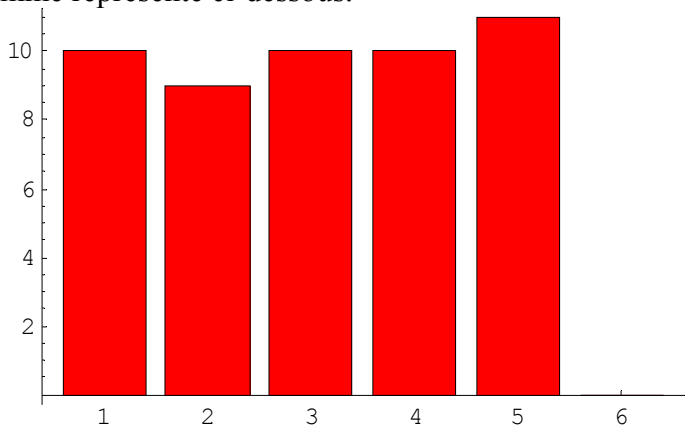
On partage donc l'intervalle [0,1] en m intervalles $I_j=[\frac{j-1}{m},\frac{j}{m})$, j=1,2,..,m (m<<n). Soit V_j le nombre de points U_i qui tombent dans l'intervalle I_j. On a alors l'exigence sur les fréquences :

Test 3. $V_j/n\approx 1/m$, $\forall$ j=1,2,…,m

Exemple 7: Le tableau ci-dessous donne 50 nombres pseudo-aléatoires obtenus à l'aide d'un générateur congruentiel (voir exemple 5 et exercice 8).

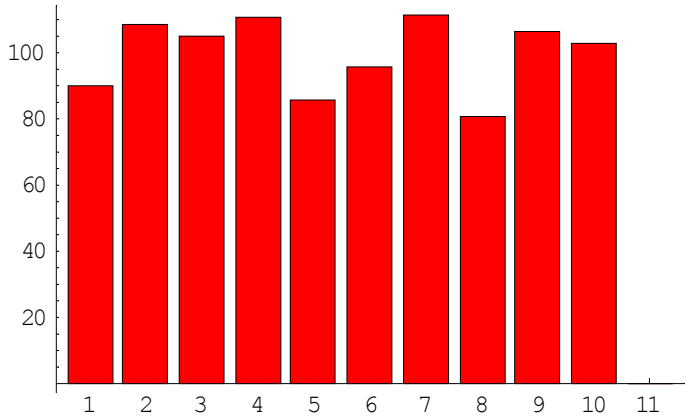
0.40419	0.735378	0.369251	0.70511	0.833578	0.397475	0.151053	0.925697
0.928252	0.481479	0.757547	0.82988	0.124692	0.777978	0.544763	0.878684
0.299862	0.478652	0.211158	0.430693	0.358442	0.005522	0.479885	0.426198
0.954252	0.270145	0.110634	0.721088	0.120675	0.87267	0.959581	0.795391
0.192522	0.39119	0.202034	0.965511	0.067830	0.613212	0.657272	0.086827
0.767969	0.13456	0.446113	0.65134	0.409527	0.129038	0.966229	0.229936
0.455275	0.858893						

Pour vérifier l'uniformité, appliquons le test 3 en partageant l'intervalle [0,1] en 5 intervalles. L'histogramme représente la distribution des fréquences des intervalles (0;0.2)-(0.2 ;0.4)-(0.4 ;0.6)-(0.6 ;0.8)-(0.8 ;1) : les effectifs respectifs sont 10,9,10,10,11. Le test 3 stipule que si les nombres sont vraiment aléatoires, il devrait y avoir une proportion de 1/5 (ou 20%) de points dans chacun des intervalles, soit en moyenne 10 points par intervalles. Ceci semble être le cas sur l'histogramme représenté ci-dessous.



Ce second exemple visualise la distribution de 1000 nombres aléatoires sur 10 intervalles.

d={90,109,105,111,86,96,112,81,107,103}



On peut utiliser d'autres critères spécifiques, par exemple

Test 4. Pour toute fonction intégrable au sens de Riemann sur [0,1], on doit avoir :

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(Ui) = \int_0^1 f(x)dx$$

Test 5. Si $U \in U[0,1]$, alors $\mathbf{E}(U) \pm \sigma_U = E(U) \pm \sigma_U = \frac{1}{2} + \frac{1}{2\sqrt{3}}$ et la proportion de points dans l'intervalle $(0.21132; 0.78868)$ doit être voisine de $2\sigma_U = 0.577362$.

On peut également utiliser des tests statistiques (Khi-deux, Kolmogorov,…), ainsi que certains tests spécifiques de l'uniformité et d'indépendance (voir plus loin).

En particulier, on peut tenter de vérifier que $\{U_i\}$ est k-équidistribuée, c'est-à-dire la distribution empirique de $(U_i,\ldots,U_{i+k-1})$ converge vers la loi uniforme sur $[0,1]^k$, $\forall k$.

Test 6 (Test du Khi-deux). C'est un test qui porte sur la distribution de la séquence.

Soit une partition $I_1+I_2+\ldots+I_k$ de l'ensemble de définition de la variable à tester, ici $U \in U[0,1]$.

Soit $p_i = P\{U \in I_i\}$, $\sum_{i=1}^{k} p_i = 1$. Notons par V_i le nombre d'observations qui tombent dans l'intervalle I_i, $i=1,2,\ldots,k$, où $n = \sum_{j=1}^{k} V_j$ est le nombre total d'observations.

Si l'hypothèse nulle est vraie, alors $\frac{V_i}{n} \to p_i$ et la statistique

$$\aleph_2 = \sum_{i=1}^{k} \frac{(V_i - np_i)^2}{np_i}$$

suit asymptotiquement $(n \to \infty)$ une loi du khi-deux à k-1 degrés de liberté. On rejette l'hypothèse nulle pour les grandes valeurs de χ^2 observé, i.e. $\chi^2 > \chi^2_{\alpha,p}$ où $P\{\chi_2 > \chi^2_{\alpha,p}\} = \alpha$.

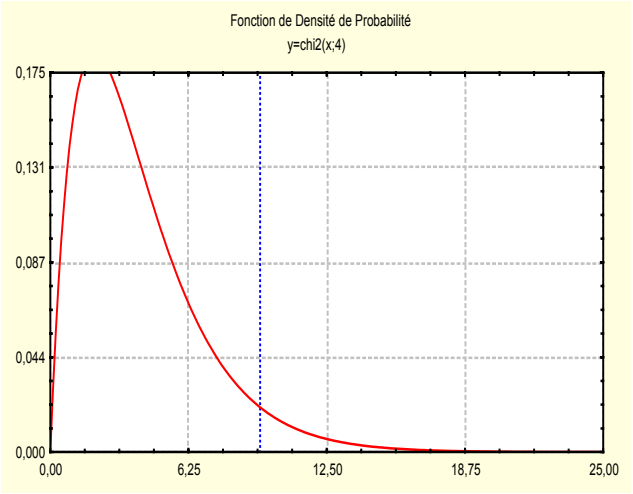
Pour le test de la loi uniforme, $p_i = |I_i|$, la longueur de l'intervalle constituant la ième classe.

Pour le test d'une loi de fonction de répartition F(x), on a $p_i = F(b_i) - F(a_i)$ où a_i et b_i sont les extrémités inférieure et supérieure respectivement de l'intervalle I_i. De plus, le degré de liberté doit être pris égal à k-m-1, où m est le nombre de paramètres estimés, de préférence à l'aide de la méthode du maximum de vraisemblance ou du minimum du Khi-deux. En pratique, les classes $I_1, I_2, \ldots, I_k$ doivent être choisies telles qu'elles contiennent au moins cinq observations en queue de distribution. Voir annexe 2 .

Exemple 7(suite): Reprenons l'exemple 7 ci-dessus. Pour confirmer notre observation graphique, on utilise le test du khi-deux.

$$\chi^2_{obs} = \sum_{i=1}^{5} \frac{(V_i - 50p_i)^2}{50p_i} = \frac{(10-10)^2}{10} + \frac{(10-9)^2}{10} + \frac{(10-10)^2}{10} + \frac{(10-10)^2}{10} + \frac{(10-11)^2}{10} = 0.2$$

Prenons pour seuil de confiance a=5%. Dans la table de la loi du khi-deux, on lit la valeur $\chi^2_{0.05;4}$ où $P\{\chi^2 > \chi^2_{0;05,4}\}=0.05$ ou bien $P\{\chi^2 < \chi^2_{0;05,4}\}=0.95$. Sur le graphique, la valeur $\chi^2_{0.05;4}$ est celle pour laquelle la surface sous la courbe à gauche de la ligne en pointillés est égale à 0.05 ; la surface à droite (sous la courbe) est elle égale à 0.95.



On trouve $\chi^2_{0.05;4} \approx 9.48840$ et puisque la valeur observée $\chi^2_{obs}=0.02<9.48840$, alors on peut accepter l'hypothèse selon laquelle la suite est bien une suite aléatoire .

Test 7 (Test de Kolmogorov –Smirnov). C'est aussi un test de fréquences. On l'utilise lorsque la loi à tester possède une fonction de répartition continue F(x). La statistique du test est

$$D_n=\max_x |F_n(x)-F(x)|$$

où $F_n(x)$ est la fonction de répartition empirique. On montre que

$$\lim_{n\to} P\{D_n\sqrt{n}<x\}=K(x)=1-2\sum_{k=1}^{\infty}(-1)^{k-1}e^{-2x^2k^2}$$

La loi limite est tabulée. En particulier K(1,36)=0,95.

Exemple (suite): Le test K-S dans sa version originale nécessite une comparaison des valeurs théoriques et empiriques, observation par observation, quoiqu'il soit possible de modifier le test pour des données groupées. Considérons un petit exemple pour illustrer le principe du test à partir des 10 premières observations seulement.

i	x_i ordonnées	$\hat{F}(x)=\frac{i}{n}$	$F_0(x_i)$	$F_0(x_i)-\frac{i}{n}$	$\hat{F}(x)=\frac{i}{n}$
1					

2 3 4 5 6 7 8 9 10					

Exercice : Soit la suite de nombres générée à l'aide d'une suite récurrente. Peut-on les utiliser en guise de suites de nombres aléatoire uniformes entre 0 et 36 ?

$$x_1=23, x_2=18, x_3=1, x_4=16, x_5=2, x_6=3, x_7=20, x_8, x_9=7, x_{10}=0$$

On cherche à tester

$$H_0: F(x)=F_0(x) \quad \text{contre} \quad H_1: F(x)\neq F_0(x)$$

où

$$F_{\grave{a}}(x)=\begin{cases} 0, & x\leq 0, \quad x\leq 0 \\ \frac{x}{36}, & 0<x\leq 36 \\ 1, & x>36 \end{cases}$$

i	x_i ordonnées	$\hat{F}(x)=\frac{i}{n}$	$F_0(x_i)$	$\frac{i}{n}-F_0(x_i)$	$F_0(x_i)-\frac{i-1}{n}$
1	0	0.1	0	0.1	0
2	1	0.2	0.027777	0.1722222	-0.07232
3	2	0.3	0.05555	0.244444	-0.1444
4	3	0.4	0.083333	0.316666	-0.2166
5	4	0.5	0.11111	0.388889	-0.28889
6	7	0.6.7	0.194444	0.405555	-0.30555
7	16	0.8	0.444444	0.255555	-0.155
8	18	0.9	0.5	0.3	-0.2
9	20	1.0	0.555555	0.34445	-0.2444
10	23	0	0.638888	0.361112	-0.2612

La valeur observée de la statistique $D_{10}=1.28247$ et $\sqrt{10}D_{10}=1.28247$.

Les tables de la loi de Kolmogorov-Smirnov (version papier, surtout à des fins pédagogiques les derniers temps) ou les logiciels comportant un module de calcul des valeurs de cette loi (version logicielle à des fins pédagogiques ou de recherche à l'époque actuelle) ont deux manières d'indiquer la technique de décision :

(iii) Les tables indiquent par exemple que $P(\sqrt{10}D_{10}>1.2824)\approx 0.0628$. C'est à l'utilisateur d'apprécier si le seuil de confiance de à 6.28% est acceptable ou non.

(iv) Supposons que l'utilisateur se fixe un seuil est de 5% , on accepte ???

Nous fournissent indiquent que

Test 8 (d'uniformité).

Le test de Khi-deux peut être utilisé pour tester l'uniformité d'un vecteur de dimension k, $(U_{ki},\ldots,U_{ki+k-1})$.On partage dans ce cas $[0,1]^k$ en domaines identiques, et on teste les fréquences d'occurrence des réalisations du vecteur ci-dessus dans les différents intervalles

Test 9. (d'auto corrélation). Il vise à tester la dépendance entre les nombres aléatoires d'une même séquence. Il est basé sur le coefficient d'auto corrélation empirique r_{im} entre m nombres partant du ième nombre. Considérons la sous-suite de notre suite de nombres au hasard : $U_i, U_{i+m}, U_{i+2m},\ldots, U_{i+(M+1)m}$. La valeur M est en général le plus grand entier tel que i+(M+1)m§n où n est le nombre total de nombres aléatoires de la séquence. On cherche donc à tester une sous-suite de longueur M+2. Il n'y a pas indépendance si l'auto corrélation est non nulle. Le test est donc de la forme :

$$H_0 : \quad r_{im}=0 \qquad \text{contre} \qquad H_1 : \; r_{im} \int 0$$

La statistique de test est naturellement le coefficient d'auto corrélation empirique de r_{im}, que nous noterons $\hat{\rho}_{im}$ .Cet estimateur suit approximativement une loi normale pour les grandes valeurs de M, si seulement les valeurs $U_i, U_{i+m}, U_{i+2m},\ldots, U_{i+(M+1)m}$ sont non corrélées. Par conséquent, sous l'hypothèse nulle (indépendance) la statistique $Z=\dfrac{\hat{\rho}_{im}}{\sqrt{Var(\hat{\rho}_{im})}}$ suit une loi normale standard de moyenne nulle et de variance unité, pour les grandes valeurs de M. En pratique, on utilise une modification de la statistique de test

$$\hat{\rho}_{im}=\frac{1}{M+1}\left[\sum_{i=0}^{M} U_{i+km}U_{i+(k+1)m}\right]-0.25 \qquad \text{avec} \qquad Var(\hat{\rho}_{im})=\frac{\sqrt{13M+7}}{12(M+1)}$$

On rejette l'hypothèse nulle au seuil a si Z est en dehors de l'intervalle $\left[-u_{\alpha/2};u_{\alpha/2}\right]$.

Autres critères souhaités : Rapidité pour l'utilisation multiple de nombres aléatoires standardisation, période suffisamment large. Ces critères suffisants pour les applications usuelles de la simulation peuvent être accompagnés d'autres exigences dans deux cas principaux :

(iii) les applications « sensibles » (nucléaire, santé,…), où l'on peut être amenés à être plus exigeant pour la validation des générateurs.
(iv) Les applications liées à la sécurité (cryptologie) : outre l'exigence sur la « fiabilité » du générateur, on peut exiger qu'il soit « cryptographiquement sûr ».

5.2.6. Liens utiles

9. Simulation et modélisations discrètes ; http://ina.eivd.ch/ina/Collaborateurs/cez/Mod im/modsim.htmhttp
10. Douillet, ensait, http://193.43.37.48/~douillet/cours/oprea/node3.html
11. http://crypto.mat.sbg.ac.at/~ste/dipl/node10.htm
12. Cryptographic random numbers , http://world.std.com.

5.Générateurs pseudo-aléatoires en C++,
http://www.agner.org/random/random.htm

13. http://java.sun.com/j2se/1.3/docs/api/java/util/Random.html
14. http://support.microsoft.com/support/kb/articles/Q231/8/47.ASP
15. http://support.microsoft.com/directory
16. http://www.iro.umontreal.ca/~lecuyer

5.2.7.Exercices.

Exercice 1. Quelle est la différence entre générateurs aléatoires et pseudo-aléatoires ?

(viii) Générer une suite de 10 nombres pseudo-aléatoires à l'aide du générateur congruentiel de paramètres : U_0=22 ; λ=2,C=2 et m=100.

(ix) **(iii) I**ndiquer comment tester la qualité de la suite ainsi générée.

(x) **(vi)** Utiliser cette suite pour simuler l'intégrale suivante : $\int_1^2 x^2 dx$.

Exercice 2: **Méthodes des carrés moyens.**

C'est le procédé algorithmique le plus ancien et dû à *Von Neumann*.

(i) choisir le terme initial de la suite U_0 .

(ii) calculer $a=U_0^2$.

(iii) Poser U_1 =suite des chiffres situés au milieu de la partie décimale de a.

(iv) Poser $a=U_1^2$; retour en (iii)

Le nombre devra à chaque itération comporter autant de chiffres significatifs que le nombre initial U_0.

Exemple. U_0 =0.2481 ; $a=U_0^2$ =0.06155361$\Rightarrow U_1$ =0.1553; $a=U_1^2$ =0.02411809$\Rightarrow U_2$ =0.4118; etc...

Ce procédé est très simple et facile à programmer. Son inconvénient est qu'on obtient beaucoup plus de petites valeurs qu'il n'en faut. De plus, si à une étape donnée l'un des nombres est nul, c'est toute la suite qui sera constituée de termes nuls.
Pour s'assurer que la suite obtenue s'accorde bien avec la loi uniforme sur U[0,1], on utilisera les critères énumérés précédemment. Vérifions par exemple l'uniformité sur l'échantillon de n nombres obtenus à l'aide du générateur de Von Neumann (En fait, il faudrait un échantillon plus grand). On a, $\overline{U}$ =0.570055 qui est bien voisin de 1/2 .

Exercice 3 : Utiliser la méthode congruentielle pour générer une séquence de nombres aléatoires de 10 chiffres telle que $x_{n+1}\equiv x_n+3) \bmod 10$ et $x_0=2$. Quelle est la période ?

Sol : 2,5,8,1,4,7,0,3,6,9,2

Exercice 4 : (i) Soit un générateur congruentiel avec $\lambda=13$, $m=2^6$. Déterminer la période maximale pour $X_0=1$ puis 3. Générer une période de chacune des suites pour $X_0=1,2,3,4$ et comparer.

(ii) Trouver la période maximale pour le générateur congruentiel avec $\lambda=7^5$, $m=2^{31}-1, c=0$.

Générer 3 nombres aléatoires lorsque la racine est $X_0=123457$.

Exercice 5 : L'automate de régulation d'un carrefour a été programmé pour fonctionner 50% du temps au vert, 10% du temps à l'orange et 40% du temps au rouge.

(iii) Générer une séquence de nombres aléatoires.
(iv) Utiliser cette séquence pour générer une séquence de couleurs.

Exercice 6 : Générer cinq observations de chacune des lois suivantes :

(vi) la loi uniforme sur l'intervalle [-10 ; 20].
(vii) La loi triangulaire de densité $f(x)=2x$ *si* $x\in[0,1]$; $f(x)=0$ sin*on*.
(viii) La loi de densité $f(x)=\frac{x-10}{50}$ *si* $x\in[10,20]$; $f(x)=0$ sin*on*.
(ix) La loi normale de moyenne $m=2$ et de variance $\sigma^2=4$;
(x) La loi d'Erlang d'ordre 2 de moyenne 4.

Exer**cice 7**. Cet exercice reprend l'exemple du début de ce chapitre pour illustrer la simulation par la méthode de Monte Carlo du calcul d'une intégrale. Le petit programme suivant calcule une valeur approchée de I par la méthode de Monte Carlo avec une séquence de $N=100$ nombres aléatoires générés selon une *méthode congruentielle*.

```
J = NIntegrate@0.50 Exp@0.5 xD, 8x, 0, 1<D
a = Table@Random@D, 8100<D;
c = Table@0.5 Exp@0.5 a@@iDDD, 8i, 100<D;
U = N@Sum@c@@iDD, 8i, 1, 100<D ê 100D
F = Abs@J - UD
```

Pour comparer, la première ligne fournit la valeur approchée par une méthode numérique de quadrature classique (de type Trapèze) : $J=0.648721$. Les résultats sont donnés dans la table. Le tracé de la courbe est réalisé par le programme suivant

```
<< Graphics`FilledPlot`
FilledPlot@0.5 Exp@0.5 xD, 8x, 0, 1<, PlotRange → 80, 1<, Frame → TrueD
```

Exercice 4: Soit X une variable aléatoire de loi exponentielle de paramètre $\lambda=2$.

(iii) Générer une suite de $n=5$ observations artificielles de cette loi (en utilisant la suite donnée dans le tableau N°1). Evaluer la moyenne arithmétique de ces 5

observations. Comparer avec l'espérance mathématique de X. Qu'en déduisez-vous ?

(iv) (TP N°2) Rédiger un programme (en C) permettant de recommencer l'expérience pour n=10,20,30,50. Tracer sur un même graphique l'histogramme expérimental et la courbe théorique.

Exercice 5 : Ecrire le programme générateur des lois de probabilité suivantes :

(iii) loi uniforme sur [a,b].

(iv) loi de Weibull de densité $f(x)=1-e^{-(\lambda x)^{\beta}}, x\geq 0$.

Exercice 6 : Ecrire le programme générateur de la loi géométrique de trois manières :

(iv) Utiliser le programme usuel de la méthode d'inversion. Donner une interprétation.
(v) Résoudre explicitement l'équation en X, et remarquer qu'on peut utiliser le générateur de la loi exponentielle.
(vi) Utiliser un algorithme similaire à celui de la loi binômiale.

Comparer les performances des trois algorithmes.

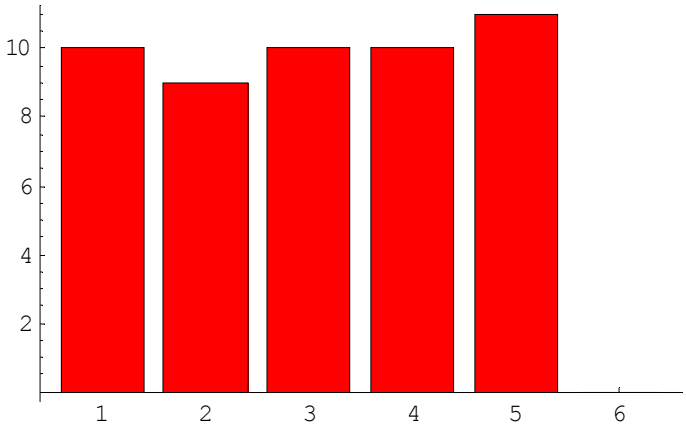
Exercice 7 : Générer trois observations pour chacune des lois suivantes :

(xi) la loi normale (utiliser les deux méthodes et comparer).
(xii) La loi d'Erlang (ou Gamma) de paramètres $k=2, \lambda=2$.

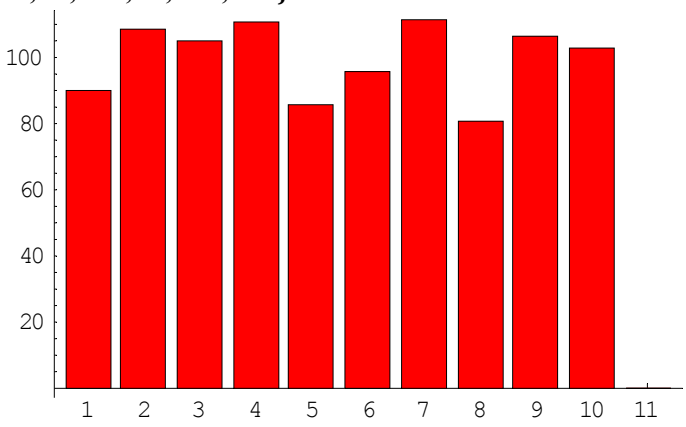
Exercice 8: Ecrire un programme permettant de générer 50 nombres aléatoires, les représenter dans le plan, puis tester leur uniformité à l'aide du test 3. Confirmer ou infirmer la visualisation graphique à l'aide d'un test de khi-deux.

Solution : Ce petit programme en MATEMATIKA reprend l'exemple 2 . On applique le test 3 en partageant l'intervalle [0,1] en 5 intervalles. L'histogramme représente la distribution des fréquences des intervalles.

```
b=Table[Random[],{50}]
ListPlot[b]
C=RangeCounts[b,{0.2,0.4,0.6,0.8,1}]
BarChart[c]
```
C={10,9,10,10,11}



Ce second exemple montre la distribution de 1000 nombres aléatoires sur 10 intervalles.
d={90,109,105,111,86,96,112,81,107,103}



Exercice 9: Générer des suites aléatoires à partir du langage C . Tester l'uniformité et l'indépendance.

Exercice 10 : Les générateurs suivants sont utilisés dans des langages bien connus.Tester leurs validité.

(ii) **Java**. C'est un générateur utilisé pour implémenter la méthode nextDouble dans la classe java.util.Random de la bibliothèque Java (voir exercice 11). Il est basé sur une réccurence linéaire de période 2^{48}, mais chaque sortie est obtenue de la manière suivante :

$$x_{i+1}=\left(25214903917x_i+11\right) \bmod 2^{48}$$
$$U_i=\left(2^{27}\left\lfloor x_{2i}/2^{22}\right\rfloor+\left\lfloor x_{2i+1}/2^{21}\right\rfloor\right)/2^{53}$$

(ii**) Unix**. Le générateur rand48 de Unix utilise la même récurrence, mais avec la sortie plus simple $U_i=x_i/2^{48}$.

(iii) **Microsoft Visual Basic**. $x_{i+1}=\left(1140671485x_i+12820163\right) \bmod 2^{24}$; $U_i=x_i/2^{24}$.

(iv) **Microsoft Excel**. Il est implémenté directement à partir des nombres U_i en arithmétique flottante $U_{i+1}=\left(9821.0U_i+0.211327\right) \bmod 1$.

(xiii) **LCG16807**. Générateur à récurrence linéaire utilisé dans de nombreux langages de simulation , par exemple ARENA ou AutoMod(avec le multiplicateur 742938285) :

$$x_{i+1}=16807x_i \mod(2^{31}-1) \quad ; \quad U_i=x_i/(2^{31}-1)$$

(xiv) **MRG32k3a**. a remplacé le précédent. Il combine deux MRG (générateur récursif multiple d'ordre 3) et sa période est de longueur voisine de 2^{191}.

Exercice 11: Génération de nombres aléatoires en JAVA. Exercice facultatif si vous êtes intéressés par ce langage.(http://www.es.geneseo.edu/~baldwin/reference/random.html).

Dans Java, il existe diverses bibliothèques (packages) permettant de générer des nombres aléatoires pour différentes utilisations : jeux, simulation, cryptographie. Cela peut se faire dans la classe Random dont les instances sont des objets générateurs de nombres aléatoires.

Générer des suites aléatoires en utilisant les instructions énumérées ci-dessous, puis tester leur uniformité et indépendance. Concevez une Applet java permettant de tester l'uniformité et l'indépendance des suites aléatoires ainsi générées. Voir dans un premier temps l'une des Applet java : Random Numbers ou Uniform Applet sur le site http://ubmail.ubalt.edu/~harsham/simulation/sim.htm

Table 3. Nombres aléatoires en JAVA

Package	Utilisation et description
java.lang	StrictMath.randomNumberGenerator
	Math.randomNumberGenerator
java.math	BigInteger.(int numBits,Random rnd) Construit un grand entier aléatoire, uniformément distribué entre 0 et $2^{numBits}$-1
	BigInteger.(int bitLength, int certainty, Random rnd) Construit un grand entier aléatoire de longueur spécifiée bitLength qui est probablement premier
Java.util	Collections.r
	Collections.shuffle(List list, Random rnd) Permute aléatoirement la liste spécifiée en utilisant une source aléatoire spécifiée
java.security	java.security.SecureRandom Accés à un générateur de nombres pseudo-aléatoires forts.
Net.vattp.security	ESecureRandom Implémente des nombres aléatoires cryptographiquement sûrs.
RngPack	Autre package de générateurs de nombres aléatoires pour java, peu recommandé pour la cryptographie

randomX	Version récente de la classe java.util.Random :utilise des générateurs congruentiels, mais aussi des générateurs combinés (randomLEcuyer) ; randomHotbits ne génére pas des nombres aléatoires à l'aide d'un algorithme, mais plutôt à partir d'Internet (HotBitsserver) : donc, pas de racine à spécifier

Voici en guise d'illustration les nombres aléatoires utilisant la classe java.util.Random**.** Cela signifie que toute instruction commence par l'instruction :

```
import java.util.Random ;        ou        import java.util.* ;
```

qui assure l'accès à la classe « Random » dans la bibliothèque Java « java.util ». L'instruction

```
Random generator=new Random( ) ;
```

initialise le générateur avec le temps actuel pour racine (résolution de une milliseconde). L'instruction suivante par contre permet de spécifier la racine 19580427 :

```
Random generator2=new Random(19580427)
```

Pour générer un entier à partir d'un objet Random, envoyer le message « nextInt ». Il renvoie l'entier suivant du générateur de la séquence aléatoire : ces entiers sont uniformément distribués dans le domaine des entiers java. Par exemple, si on suppose que « generator » est une instance de Random

```
Int r=generator.nextInt( ) ;
```

Si on souhaite générer un entier compris entre 0 et une certaine borne supérieure. Par exemple on souhaite sélectionner un symbole au hasard parmi un ensemble de n symboles (ou indice dans une liste de n éléments). Les indices de cette liste dans Java sont numérotés de 0 à n-1. Si on attribue un paramètre entier n à « nextInt », on obtient en retour un entier de la distribution uniforme entre 0 et n-1 ; Si on utilise l'objet de générateur de nombre aléatoire pour générer la liste aléatoire d'indices, on utilisera l'instruction

```
Int randomIndex=generator.nextInt( n ) ;
```

Enfin, pour générer un nombre aléatoire réel uniformément distribué sur [0,1], on utilisera

```
Double r=generator.nextDouble( ) ;
```