Ecole
Internationale
des Sciences
du Traitement
de l'Information

CHAPITRE 4 : SIMULATION DE VARIABLES ALEATOIRES

Table des matières

N.B. Les exercices sont proposés au fur et à mesure du chapitre. Pour chaque simulation, il est demandé un algorithme et un programme en Scilab.

1. GENERATEUR DE NOMBRES ALEATOIRES

1.1. Les postulats

Tous les langages disposent d'un générateur pseudo-aléatoire, appelé **rand** dans Scilab. C'est une fonction qui retourne un nombre 'au hasard' entre 0 et 1 et dont on admet qu'elle vérifie les postulats suivants :

1. $0 \le a < b \le 1$, $\text{Proba}(\text{rand} \in]a,b]) = b-a$
2. Les appels successifs de rand sont des variables aléatoires indépendantes

<u>Conséquences</u>

- $\forall k \in \mathbb{N}^*$, soient $(R_1,\dots,R_k)$, k appels successifs de rand. Pour tout rectangle $D=]a_1,b_1]\times\dots\times]a_k,b_k]$, $0 \le a_i < b_i \le 1$ $i=1,\dots,k$

$$\text{Proba}[(R_1,\dots,R_k) \in D] = (b_1-a_1)\dots(b_k-a_k).$$

Par exemple pour k=2, la probabilité qu'un point dont les coordonnées sont des appels successifs de `rand` tombe dans un rectangle est la surface de ce rectangle.

- La probabilité pour que `rand` tombe sur une valeur particulière est nulle

$$\forall a \in [0,1], \text{Proba}[\texttt{rand}=a]=0 ,$$

$$\Rightarrow \text{Proba}(\texttt{rand} \in [a,b[)=\text{Proba}(\texttt{rand} \in [a,b])= \text{Proba}(\texttt{rand} \in]a,b[).$$

Il y a un paradoxe à admettre à la fois que `rand` peut prendre une infinité de valeurs distinctes et ne prend jamais aucune valeur particulière. En pratique la situation est légèrement différente. Premièrement, pour l'ordinateur, il n'existe qu'un nombre fini de valeurs. Les valeurs de `rand` sont toujours calculées à partir d'entiers répartis dans {0,1,…,M}, où M est de l'ordre de 10^8 au moins. Pour retourner un réel entre 0 et 1, il suffit de diviser par M. En pratique, seul un nombre fini de valeurs peuvent être atteintes. Ceci contredit les postulats de définition de `rand` mais ne constitue pas un inconvénient majeur dans la mesure où M est très grand. Deuxièmement la valeur retournée par un générateur est une fonction de la (ou les) précédente(s) valeurs(s) (d'où générateur pseudo-aléatoire). En fait on a à faire à une suite récurrente sur un ensemble fini donc périodique. On considère ici que la suite est aléatoire dans la mesure où la période est très grande. Il existe différent types d'algorithmes pour générer des suites de nombres pseudo-aléatoire qui ont tous besoin d'une 'graine' pour initialiser la première valeur de la suite. Il est donc indispensable de faire figurer une instruction de randomisation (une seule fois) en début de session. Dans Scilab, cette instruction est

```
rand("seed",n).
```

Un petit test
Quitter et rouvrir Scilab
Générer cinq nombres aléatoires et noter le vecteur obtenu
Quitter et rouvrir Scilab
Générer à nouveau cinq nombres aléatoires et comparer les deux vecteurs

Nous allons donc considérer que la fonction `rand` simule la réalisation d'une loi uniforme sur [0,1]. A partir de cette fonction il possible de simuler n'importe quelle loi de probabilité.

1.2. Simulations triviales

Des postulats précédents découlent immédiatement des simulations simples de lois usuelles.

Loi de Bernoulli
Dans le cas d'une loi de Bernoulli B(p), la probabilité d'un succès (codé X=1) est p et la probabilité d'un échec (codé X=0) est 1-p. Pour simuler une telle loi, il suffit d'utiliser le premier postulat d'où l'algorithme :

```
Alea ← rand
Si Alea<p
   X ← 1
Sinon
   X ← 0
```

Exercice 1 : Simuler une loi géométrique G(p)

Loi uniforme continue

> Soit X une variable aléatoire de loi uniforme sur [0,1], alors la variable
> $$Y=aX+b$$
> suit une loi uniforme sur [b,a+b] où a>0.

Exercice 2 : Simuler une loi uniforme sur [-1,1]

Loi uniforme discrète

> Soit X une variable aléatoire de loi uniforme sur [0,1], alors la variable
> $$Y=E[kX]$$
> où $k\in N$ et E[.] désigne la partie entière, suit une loi uniforme {0,1,…,k-1}.

Exercice 3 : Simuler le lancer d'un dé

1.3. Validation de simulation

Dans les paragraphes suivants, nous allons voir quelques méthodes pour simuler des variables aléatoires de loi connue. Chaque simulation devra être validée, c'est-à-dire qu'il faudra vérifier si la simulation correspond bien à la loi simulée. Pour cela, il est nécessaire de simuler non pas une réalisation de la variable aléatoire simulée X mais un échantillon (*i.e* plusieurs réalisations indépendantes) $x_1,x_2,\ldots,x_k$. Il est alors possible de montrer que pour k suffisamment grand la moyenne (`mean`) donne une valeur approchée de l'espérance et la variance empirique (`variance`) une valeur approchée de la variance théorique (Statistiques ING2)

$$\bar{x}=\frac{1}{k}\sum_{i=1}^{x}x_i\approx E(X) \quad \text{et} \quad s^2=\frac{1}{k}\sum_{i=1}^{x}(x_i-\bar{x})^2\approx \text{var}(X)$$

Voici donc deux premiers indicateurs pour vérifier si une simulation est valide. On utilisera un troisième indicateur graphique, l'histogramme (`histplot`) de l'échantillon dans le cas continu ou le diagramme bâton (`diagramme`) dans le cas discret, qui doit avoir la même forme caractéristique que la fonction de densité de la variable simulée.

Evidemment, plus l'échantillon est de grande taille, plus les valeurs calculées sur l'échantillon sont proches des valeurs théoriques. Il est donc essentiel qu'une boucle de simulation soit la plus rapide possible. Il faut en éliminer toutes les opérations coûteuses et inutiles et faire le moins de boucles possibles. Pour comparer l'efficacité des algorithmes, on peut utiliser les commandes **tic** (début) et **toc** (fin) qui permettent de mesurer le temps CPU écoulé. Pour vous rendre compte de la nécessité d'optimiser vos programmes, comparer les deux méthodes suivantes pour assigner une matrice 1000×1000 de valeurs aléatoires :

```
A=[];
tic()
for i=1:1000,
    for j=1:1000,
        A(i,j)=rand();
    end;
end;
toc()
```

```
tic()
A=rand(1000,1000);
toc()
```

2. SIMULATION PAR INVERSION

2.1. Méthode d'inversion pour lois discrètes

Soit X une variable aléatoire à valeurs $\{x_i, i\in\{1,\dots,n\}$ ou $i\in\mathbb{N}$ où $x_i<x_{i+1}\}$ telle que $P[X=x_i]=p_i$, $i>0$. La fonction de répartition correspondante est définie par

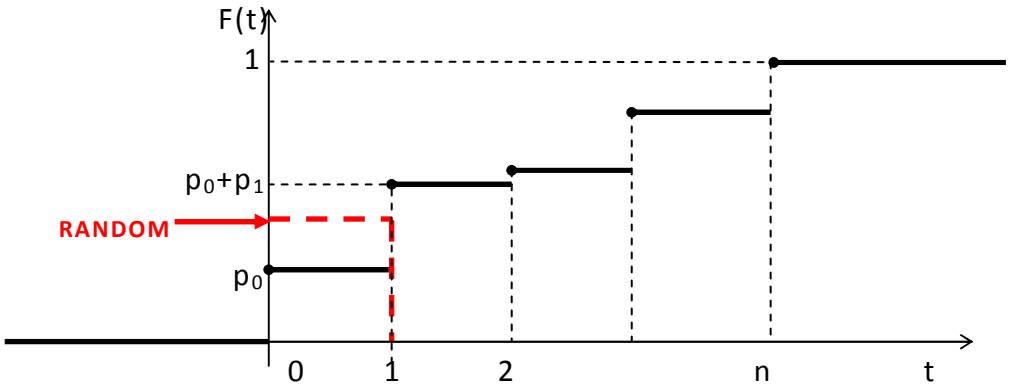
$$F(x)=\begin{cases}0 & \text{si } x<x_1\\ p_1+\dots+p_i=F_i & \text{si } x_i\le x<x_{i+1}\end{cases}.$$

L'algorithme de simulation par inversion est l'algorithme naturel de choix entre plusieurs éventualités. Il suppose que l'on connaisse explicitement les F_i.

Algorithme

```
i ← 1
Alea ← rand
Tant que (Alea>Fi) faire
   i ← i+1
Fin tant que
X ← xi
```

Le nombre de tests de l'algorithme vaut i avec une probabilité p_i. On a donc tout intérêt à ranger les éventualités par ordre de probabilités décroissantes.



Exemple : Simulation de la loi de Poisson

Soit X une variable aléatoire de loi de Poisson de paramètre λ. Il n'y a pas d'expression simple pour la fonction de répartition et l'ensemble des valeurs possibles est infini. Il faut donc dans ce cas particulier calculer les valeurs de F_i au fur et à mesure. On écrit pour cela la loi de probabilité sous forme récursive :

$$\text{Prob}[X=n]=e^{-\lambda}\frac{\lambda^n}{n!}=\frac{\lambda}{n}\text{Prob}[X=n-1].$$

```
P ← e^-λ, F ← P, X ← 0
Choix ← rand
Tant que (choix>F)
   X ← X+1, P ← P*λ/X, F ← F+P
Fin tant que
```

Exercice 4 : Simuler une loi géométrique – Comparer avec la simulation de l'exercice 1

2.2. Méthode d'inversion pour lois continues

> Soient F une fonction de répartition et U une variable aléatoire de loi uniforme sur [0,1]. La variable aléatoire $X=F^{-1}(U)$ a pour fonction de répartition F.

Il suffit donc de connaître l'inverse de la fonction de répartition pour pouvoir simuler la loi. Exemple de la loi exponentielle de paramètre λ.

$\forall x\geq 0$, $F(x)=1-e^{-\lambda x}$ donc $\forall u\in[0,1]$, $F(x)=u \Leftrightarrow x=-\log(1-u)/\lambda$.

Exercice 5 : Simuler une loi exponentielle

3. SIMULATION DE LA LOI NORMALE

Il est uniquement nécessaire de savoir simuler la loi normale centrée réduite puisque si Z suit une $N(0,1)$ alors $Y=\sigma Z+\mu$ suit une $N(\mu,\sigma^2)$.

3.1. Théorème de la limite centrale

Soit $(X_n)_{n>0}$ une suite de variables aléatoires indépendantes de même loi, d'espérance α et de variance finie β^2. Alors

$$Z_n=\frac{\sqrt{n}}{\beta}(\overline{X}_n-\alpha),\ \text{où}\ \overline{X}_n=\frac{X_1+\ldots+X_n}{n}$$

converge en loi vers la loi normale $N(0,1)$.

Algorithme

```
M←0
Répéter n fois
   Xi←simulation loi(α,β²)
   M←M+Xi
Fin répéter
Z←(M-α)*√n/β
```

Problème : A partir de quel rang peut-on considérer que la convergence est atteinte ? Cela dépend de la loi utilisée. Par exemple si on utilise une suite de lois uniformes sur [0,1] alors 12 itérations suffisent en général. Cela vient du fait que $\alpha=1/2$ et $\beta^2=1/12$ donc $S=(X_1+\ldots+X_{12})/12$ suit une loi $N(1/2,1/12)$ et $X=12[S/12-1/2]$ suit une $N(0,1)$ donc pour n=12 les calculs sont simplifiés car X=S-6.

Exercice 6 : Simuler une loi N(0,1) à partir d'une loi uniforme
Simuler une loi N(0,1) à partir d'une loi exponentielle
Comparer

3.2. Méthode de Box-Muller

Cette méthode est basée sur la proposition suivante.

> Soient R un v.a. de loi exponentielle de paramètre ½ et Θ une v.a. indépendante de R de loi uniforme sur $[0,2\pi]$. On note $X=\sqrt{R}\cos\Theta$ et $Y=\sqrt{R}\sin\Theta$. Alors X et Y sont 2 v.a. indépendantes de loi normale $N(0,1)$.

Remarque : Le fait d'obtenir deux v.a. indépendantes de loi $N(0,1)$ permet de diviser par deux le nombre d'itérations à effectuer pour construire un échantillon de taille n.

Exercice 7 : Simuler une loi N(0,1) par la méthode de Box-Muller et comparer avec les simulations de l'exercice 6