simulation des lois de probablités

ESSADDOUKI Mostafa (essaddouki@gmail.com), 10 mai 2018

1 Fonctions utils

1.1 Le générateur aléatoire rand

La librairie standard d'un langage d'implémentation sur ordinateur contient généralement un générateur de nombres aléatoires. L'appel à une fonction de type random fournit une suite de nombres $x_1, ..., x_n \in [0, 1]$ sensés être **n** réalisations $X1(\omega), ..., Xn(\omega)$ de **n** variables aléatoires indépendantes de **loi uniforme** sur $[0, 1]$.
Sous Scilab, la fonction rand permet de réaliser une telle simulation, à ceci près qu'elle permet de générer des matrices aléatoires au lieu de simples listes. Ainsi, l'appel à la fonction :

- **rand(m,n)** fournit une matrice aléatoire de dimension m*n, dont chaque terme est la réalisation d'une suite de m*n variables aléatoires indépendantes de loi uniforme sur [0, 1]
- **rand(a)** fournit une matrice aléatoire de même dimension que la matrice a
- **rand()** sans argument retourne un scalaire.
- **rand("uniform")** : la loi par défaut est la loi uniforme sur [0, 1].
- **rand("normal")** : la loi par défaut est la loi normale centrée réduite.

```
--> rand(3,4)
 ans  =

   0.2113249   0.3303271   0.8497452   0.068374
   0.7560439   0.6653811   0.685731    0.5608486
   0.0002211   0.6283918   0.8782165   0.6623569
--> A=[1 2 3]
 A  =

   1.    2.    3.

--> rand(A)
 ans  =

   0.7263507   0.1985144   0.5442573

--> rand()
 ans  =

   0.2320748
```

1.2 Le générateur aléatoire grand

La fonction grand dispose de générateurs aléatoires produisant des séquences de nombres qui possèdent de meilleures qualités statistiques que rand. Par conséquent, dans les situations ou la qualité statistique des séquences de nombres aléatoires est importante, il vaut mieux utiliser la fonction grand. Cette fonction

dispose par ailleurs de fonctionnalités très variées dont voici les plus classiques (les instructions ci-dessous renvoient une matrice de taille N x M de valeurs obtenues selon la loi choisie) :

- **X=grand(N,M,"uin",n1,n2)** : simule une variable aléatoire suivant la loi uniforme sur $[\![n1, n2]\!]$;
- **X=grand(N,M,"bin",n,p)** : simule une variable aléatoire suivant la binomiale de paramètres (n, p) ;
- **X=grand(N,M,"geom",p)** : simule une variable aléatoire suivant la loi géométrique de paramètre p ;
- **X=grand(N,M,"poi",lambda)** : simule une variable aléatoire suivant la loi Poisson de paramètre λ ;
- **X=grand(N,M,"def")** : simule une variable aléatoire suivant la loi uniforme sur $[0, 1[$;
- **X=grand(N,M,"unf",a,b)** : simule une variable aléatoire suivant la loi uniforme sur $[a, b[$;
- **X=grand(N,M,"gam",nu,1/b)** : simule une variable aléatoire suivant la loi gamma de paramètres (b, v) ;
- **X=grand(N,M,"bet",a,b)** : simule une variable aléatoire suivant la loi beta de paramètres (a, b) ;

```
--> grand(1,5,"uin",1,10)
 ans  =

   2.    10.    6.    9.    4.


--> X=grand(1,5,"bin",10,0.5)
 X  =

   3.    6.    8.    8.    8.
--> X=grand(1,5,"geom",0.5)
 X  =

   1.    1.    3.    3.    1.
--> X=grand(1,5,"unf",1,7)
 X  =

   4.8385801    5.753244    6.2705839    6.7569546    4.0219761
```

2 Simulation des lois

2.1 La loi de Bernoulli de paramètre p.

C'est aussi une brique de base. L'algorithme le plus simple consiste à couper l'intervalle $[0, 1]$ en deux sous-intervalles, l'un de taille p, l'autre de taille $1 - p$:

```
U = rand(1);
if(U < p) then X = 1;else X = 0;
end
```

2.2 Loi discrète uniforme sur [a,b].

On utilise l'instruction **rand(n)** qui renvoie un entier aléatoire compris entre 0 et n-1 suivant la loi uniforme sur $[\![0; n - 1]\!]$. On obtient alors une simulation de X avec l'instruction suivante :

```
X=a+rand(b-a+1)
```

2.3 Loi binomiale de paramètre (n, p)

Il suffit d'ajouter des lois de Bernoulli

```
U = rand(1, n)
Y = (U < p) ;
X = sum(Y )
```

2.4 Loi géométrique de paramètre p

Il suffit de répéter des Bernoulli jusqu'à obtenir un succès :

```
X = 1;
U = rand(1);
while(U>p)X=X+1;U=rand(1); end
disp(X)
```

2.5 Loi uniforme à densité sur un intervalle [a,b]

On utilise l'instruction **rand()** qui simule la loi uniforme sur [0 ; 1].
On obtient une simulation de **X** avec l'instruction suivante :

```
X=a+(b-a)*rand()
```