É.Bouchet ECS1 # TP6 - Simulation de variables finies

1 Tirer des nombres aléatoirement avec Scilab

1.1 Généralités

Scilab ne peut pas produire de vrai aléatoire, mais le logiciel dispose d'un générateur de nombres pseudo-aléatoire. Les générateurs pseudo aléatoires sont fondés sur des séquences déterministes, mais suffisamment imprévisibles pour paraître aléatoires.
Dans le but de pouvoir obtenir des simulations reproductibles, ces générateurs sont toujours initialisés de la même façon au démarrage de Scilab, de telle sorte que la séquence est la même d'une session à l'autre. En d'autres termes, les premiers nombres produits par la fonction rand que nous allons présenter sont toujours : 0.2113249, 0.7560439, ...
Il est possible de modifier ce réglage (des détails à ce sujet sont donnés dans la documentation), mais cela n'est pas nécessaire la plupart du temps.

1.2 La fonction rand

La fonction `rand` retourne une matrice aléatoire dont chaque élément est la réalisation d'une variable aléatoire suivant une loi donnée (par défaut une loi uniforme sur $[0,1]$, que l'on présentera formellement dans le chapitre sur les variables aléatoires à densité).
En particulier, on peut utiliser :

- Pour obtenir une valeur aléatoire :
  `r=rand()`
- Pour obtenir une matrice aléatoire de dimensions $m \times n$:
  `r=rand(m,n)`
- Pour obtenir une matrice aléatoire de même dimensions qu'une matrice A donnée :
  `r=rand(A)`

1.3 La fonction grand

La fonction **grand** ressemble à la fonction rand, mais nécessite de rentrer la loi de probabilité choisie en argument. Un grand nombre de lois usuelles sont disponibles.
On a notamment, si l'on veut des matrices de taille $m \times n$:

- Pour une loi binomiale de paramètres $N \in \mathbb{N}^*$ et $p \in [0,1]$:
  `Y = grand( m, n, "bin", N, p )`
- Pour une loi uniforme sur des entiers entre a et b :
  `Y = grand( m, n, "uin", a, b )`

1.4 Simulations de lois personnalisées

Commençons par considérer un exemple : une urne contient 6 boules blanches, 3 boules rouges et 1 boule noire. Pour simuler le tirage d'une boule, il suffit de tirer un réel uniformément entre 0 et 1 :

- si le réel obtenu est entre 0 et $0,6$ (ce qui arrive avec probabilité $0,6 = \frac{6}{10}$, qui correspond à la probabilité de tirer une boule blanche), on tire une boule blanche,
- s'il est entre $0,6$ et $0,9$ (ce qui arrive avec probabilité $0,9 - 0,6 = 0,3 = \frac{3}{10}$, qui correspond à la probabilité de tirer une boule rouge), on tire une boule rouge,
- s'il est entre $0,9$ et 1 (ce qui arrive avec probabilité $1 - 0,9 = 0,1 = \frac{1}{10}$, qui correspond à la probabilité de tirer une boule noire), on tire une boule noire.

Exercice 1.

1. En utilisant ce principe, implémenter en Scilab une fonction qui ne prend aucun argument et simule un tirage. Elle renverra la couleur de la boule sélectionnée.
2. Que se passe-t-il si on appelle cette fonction plusieurs fois successivement ?

Cette méthode s'adapte facilement au cas de n'importe quel système complet d'événements $(B_i)_{i\in[\![1,n]\!]}$. Si on pose $\forall i \in [\![1,n]\!]$, $P(B_i) = p_i$, la procédure s'adapte comme suit :

- On tire un réel u uniformément entre 0 et 1.
- Si u est compris entre 0 et p_1 (ce qui arrive avec probabilité p_1), on réalise B_1,
- Si u est compris entre p_1 et $p_1 + p_2$ (ce qui arrive avec probabilité p_2), on réalise B_2,
- Si u est compris entre $p_1 + p_2$ et $p_1 + p_2 + p_3$ (ce qui arrive avec probabilité p_3), on réalise B_3,
- ...
- Si u est compris entre $\displaystyle\sum_{i=1}^{k} p_i$ et $\displaystyle\sum_{i=1}^{k+1} p_i$ (ce qui arrive avec probabilité p_{k+1}), on réalise B_{k+1},
- ...
- Si u est compris entre $\displaystyle\sum_{i=1}^{n-1} p_i$ et $\displaystyle\sum_{i=1}^{n} p_i = 1$ (ce qui arrive avec probabilité p_n), on réalise B_n.

2 Histogrammes

Pour tracer des histogrammes, on utilise la fonction `histplot`, qui utilise deux arguments : les classes utilisées pour le diagramme et les données à représenter.
Soit c un entier ou un vecteur et d un vecteur des données, on écrit : `histplot(c,d)`

- Lorsque c est un entier, les classes de l'histogramme sont constituées de c classes de même taille, déterminées automatiquement.
- Lorsque c est un vecteur de taille n, les classes sont $[c(1), c(2)]$ puis $]c(i), c(i+1)]$ pour $i = 2, 3, \ldots, n-1$.

Remarque. Par défaut, l'histogramme est renormalisé pour avoir une aire de 1. Il faut donc faire attention en lisant les valeurs sur l'axe des ordonnées.

Pour tracer l'histogramme de simulations d'une variable aléatoire $X \hookrightarrow \mathcal{B}(6, 0.5)$, on commence par exemple par déterminer les classes (il en faut 7 ici puisque $X(\Omega)$ contient 7 valeurs) et le jeu de données à utiliser :

```
c = 7 ; d = grand(1000,1,"bin", 6, 0.5) ;
```

On peut alors tracer l'histogramme :

- Pour un histogramme normalisé : `histplot(c, d)`.
- Pour un histogramme non normalisé : `histplot(c, d, normalization=%F)`.

Exercice 2. Que se passe-t-il si on remplace la définition de c par `c = -0.5:6.5` ?

Exercice 3. Simuler 50 tirages d'une variable aléatoire $Y \hookrightarrow \mathcal{U}([\![1,5]\!])$ et les représenter sous forme d'histogramme.