**Feuille de TP n° 10**

# Probabilités avec Scilab

**A – Simulation et représentation de variables aléatoires réelles discrètes**

Ce TP accompagne le chapitre 6 (Informatique et Algorithmique) : **Probabilités avec Scilab**.
Créer le dossier `..\ECS1B_TPInfo\TP10\` et faites-en le répertoire courant de Scilab. Tous les scripts devront être sauvegardés dans ce dossier.

## I Entrainement

### 1) Simulation de variables aléatoires réelles discrètes

*Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.*
John von Neumann

#### a) La fonction `rand`

• La fonction `rand` de Scilab simule un nombre tiré aléatoirement entre $0$ et $1$. Exécuter trois fois consécutivement la commande `rand()` dans la console. Constater que vous avez tous les trois mêmes valeurs...

• Fermer Scilab et exécuter trois fois consécutivement la commande `rand()` dans la console. Constater que vous avez tous les trois mêmes valeurs... `0.2113249`, `0.7560439` et `0.0002211`. Pas très aléatoire tout ça... les raisons sont expliquées en détail dans le cours. On parle de nombres pseudo-aléatoires.

• L'algorithme utilisé par Scilab pour simuler des nombres pseudo-aléatoires peut être modifié. Il suffit de changer la graine (ou le germe) de l'algorithme, c'est-à-dire sa terme initial, en lui affectant la date (en seconde) de la machine. Cela permet d'intégrer une petite dose de hasard dans le générateur. Exécuter `n=getdate('s'); rand('seed',n);` puis exécuter trois fois consécutivement la commande `rand()` dans la console. Comparer avec vos camarades. Qu'en pensez-vous ?

• Exécuter les commandes suivantes dans la console :

```
-->rand(1,10);
-->rand(5,1);
-->rand(7,6);
```

• Exécuter la commande `rand('seed',0);` puis `rand()` à nouveau. Que s'est-il passé ?

#### b) Simulations avec `rand`

• Soient `a` et `b` des entiers tels que `a` $<$ `b`. Justifier brièvement que le nombre renvoyé par la commande `X=a+floor((b-a+1)*rand())` peut être assimilé à la réalisation d'une variable aléatoire de loi uniforme sur $[\![\texttt{a}, \texttt{b}]\!]$. Exécuter les commandes suivantes dans la console :

```
-->a=-6; b=10; X=a+floor((b-a+1)*rand())
-->X=a+floor((b-a+1)*rand(9,5))
```

Écrire une commande qui simule le résultat de 20 lancers successifs d'un dé équilibré à six faces. Tester-la plusieurs fois dans la console.

• Soit `p` un réel de $]0, 1[$. Justifier brièvement que le nombre renvoyé par la commande `rand()<p` peut être assimilé à la réalisation d'une variable aléatoire de loi de Bernoulli de paramètre `p`. Exécuter les commandes suivantes dans la console :

```
-->X=(rand()<0.1)
-->X=(rand(4,7)<0.8)
```

• Soient p un réel de $]0,1[$ et n un entier naturel non nul. Justifier brièvement que le nombre renvoyé par la commande `sum(rand(1,n)<p)` peut être assimilé à la réalisation d'une variable aléatoire de loi binomiale de paramètres n et p. Exécuter les commandes suivantes dans la console :

```
-->X=sum(rand(1,20)<0.9)
-->X=sum(rand(1,20)<0.2)
```

Écrire une commande qui simule le nombre de 6 obtenus en lançant 100 fois un dés équilibré à six faces. Tester-la plusieurs fois. Commenter les résultats obtenus.

• Soient p un réel de $]0,1[$. Justifier brièvement que le nombre renvoyé par les instructions `X=1; while (rand()>=p); X=X+1; end; X` peut être assimilé à la réalisation d'une variable aléatoire de loi géométrique de paramètre p. Exécuter les commandes suivantes dans la console :

```
-->X=1; while (rand()>=0.9); X=X+1; end; X
-->X=1; while (rand()>=0.1); X=X+1; end; X
```

Écrire une commande qui simule le nombre de lancers de dés nécessaires pour obtenir le premier 6. Tester-la plusieurs fois.

### c) Simulations avec `grand`

• Exécuter les commandes suivantes dans la console :

```
-->U=grand(5,7,'uin',-1,2)
-->X=grand(9,1,'bin',1,0.6)
-->Y=grand(1,12,'bin',50,0.6)
```

```
-->Z=grand(4,4,'geom',0.3)
-->T=grand(6,2,'poi',5)
```

• Simuler une variable aléatoire de loi de Poisson de paramètre 9.

## II Représentation graphique

### 1) Représentation graphique de lois théoriques

#### a) Diagrammes en bâtons

• Écrire un script contentant le programme suivant :

```
n=input('Entrer un entier naturel n strictement positif : ')
p=input('Entrer un réel p compris entre 0 et 1 : ')
u=(1-p)^n; U=[u];
for  k=1:n
     u=u*(n-k+1)/k;
     u=u*p/(1-p);
     U=[U,u];
end
```

Enregistrer ce script sous le nom `LoiBin.sce` et exécuter-le dans la console (avec un `clf();` avant).

• Tester `LoiBin.sce` avec différentes valeurs de n et p. Pour chaque essai, exécuter la commande `bar(0:n,U,'r')` ou `plot2d3(0:n,U,style=5)`.

• Construire le diagramme en bâtons représentant la loi du résultat d'un lancer de dé équilibré. Même question pour la loi de la somme du résultats de deux dés.

#### b) Fonctions de répartition

• Exécuter les commandes suivantes dans la console :

```
-->X=[-1,2,3,4,2,2,3,2,3,5,6,-1,3,4,2];
-->tabul(X)
-->T=tabul(X,'i')
```

```
-->T(:,2)=T(:,2)/sum(T(:,2))
-->T(:,2)=cumsum(T(:,2));
```

Que font les commandes `tabul` et `cumsum` ?

• Exécuter la commande `plot2d2(T(:,1),T(:,2))`.

• Construire la fonction de répartition d'une variable aléatoire de loi binomiale (pour différents paramètres).

• Construire la fonction de répartition de la loi du résultat d'un lancer de dé équilibré. Même question pour la loi de la somme du résultats de deux dés.

### 2) Représentation graphique de phénomènes aléatoires

• Exécuter les commandes suivantes dans la console :

```
-->p=0.7; n=10;
-->X=grand(1,1000,'bin',n,p);
-->T=tabul(X,'i');
```

```
-->T(:,2)=T(:,2)/sum(T(:,2));
-->clf(); plot2d3(T(:,1),T(:,2),style=-1)
```

Exécuter ensuite la commande `LoiBin.sce` afin de superposer le diagramme théorique de la loi $\mathcal{B}(10, 7/10)$ au diagramme empirique obtenu. Commenter. On ajoutera une légende.

• Exécuter les commandes suivantes dans la console :

```
-->T(:,2)=cumsum(T(:,2));
-->clf(); plot2d2(T(:,1),T(:,2))
```

```
-->exec('LoiBin.sce',-1)
//Prendre n=10 et p=0.7
-->plot2d2(0:n,cumsum(U),style=5)
```

• Faites la même chose pour $1000$ lancers de dés successifs (diagramme en bâtons, fonction de répartition empirique, superposition avec les courbes théoriques).

## III Exercices

*Commencer par taper* `clear` *dans la console. Pour vous y retrouver plus facilement, veuiller écrire*

- *en commentaire au début de chaque fonction le numéro du TP et de l'exercice concerné.*
- *sur cette feuille de TP, le nom que vous avez donné à la fonction (extension en* `.sci`*) ou au programme (extension en* `.sce`*).*

*Vous testerer dans la console Scilab toutes les fonctions que vous écrirer (sans oublier de les exécuter au préalable).*

**Exercice 1 (simulation et représentation de v.a. de loi géométrique).**

1) Écrire une fonction `simgeom.sci` qui prend en entrée un réel `p` de $]0, 1[$ et un entier naturel `N` strictement positif et qui renvoie un vecteur contenant `N` réalisations indépendantes de variables aléatoires de loi géométrique de paramètre `p`. On n'utilisera pas la fonction `grand`.
2) En utilisant les fonctions `tabul` et `plotd2d3`, représenter les diagrammes en bâtons de
   a) `simgeom(0.2,50)`, `simgeom(0.5,50)` et `simgeom(0.8,50)`.
   b) `simgeom(0.2,1000)`, `simgeom(0.5,1000)` et `simgeom(0.8,1000)`.
   *On pourra utiliser* `subplot(2,3,#)`*.*
   c) Comparer avec les diagrammes en bâtons théoriques.
   *La commande* `p*((1-p)*ones(1,K)).^(0:(K-1))` *renvoie un vecteur content* $\mathbb{P}(X = k)$ *pour tout* $k \in [\![1, K]\!]$ *lorsque* $X$ *est une v.a. de loi* $\mathcal{G}(p)$*. Comprenez-vous pourquoi ?*
3) En utilisant les fonctions `tabul`, `cumsum` et `plotd2d2`, représenter les fonctions de répartition empiriques de
   a) `simgeom(0.2,50)`, `simgeom(0.5,50)` et `simgeom(0.8,50)`.
   b) `simgeom(0.2,1000)`, `simgeom(0.5,1000)` et `simgeom(0.8,1000)`.
   *On pourra utiliser* `subplot(2,3,#)`*.*
   c) Comparer avec les fonctions de répartitions théoriques.
   *La commande* `plot2d2(1:K,1-(p*ones(1:K)).^(1:K))` *trace la courbe représentative de la fonction de répartition d'une v.a. de loi* $\mathcal{G}(p)$ *sur l'intervalle* $[1, K]$*. Comprenez-vous pourquoi ?*

**Exercice 2 (simulation et représentation de v.a. de loi de Poisson).**
Reprendre l'exercice précédent avec la loi de Poisson. On utilisera cette fois la fonction `grand` et on pourra tester les paramètres 1, 5 et 20.
*On se limitera aux valeurs significatives des probabilités.*

**Exercice 3.** Écrire une fonction qui prend en entrée deux entiers naturels $n$ et $p$ strictement positifs tels que $p \leqslant n$, qui simule $p$ tirages successifs et avec remise dans une urne contenant $n$ boules numérotées de $1$ à $n$ et qui renvoie un vecteur contenant les $p$ numéros tirés (dans l'ordre du tirage).

**Exercice 4.** Recommencer l'exercice précédent avec des tirages sans remise.

*Indication : on pourra commencer par se donner le vecteur `B=1:`$n$ représentant les numéros des $n$ boules, ainsi qu'un vecteur vide `T=[]` représentant les numéros piochés. Ensuite on pourra*

- *tirer uniformément un nombre `k` dans `1:length(B)`,*
- *ajouter `B(k)` à `T` et supprimer `B(k)` de `B`,*

*puis recommencer ces deux étapes à partir des vecteurs `B` et `T` ainsi modifiés.*

**Exercice 5.** On dispose d'un stock suffisant de boules rouges et d'une urne qui contient $r \geqslant 2$ boules rouges et $b \geqslant 2$ boules bleues. On tire une boule au hasard dans l'urne. Si elle est rouge, on la remet dans l'urne. Si elle est bleue, on la remplace par une rouge et on la remet dans l'urne. On recommence cette opération indéfiniment.

1) Ecrire une fonction commençant par `function R=tirage(r,b,n)` qui simule `n` tirages successifs avec cette règle et renvoie le vecteur `R` dont la valeur est `[`$\mathtt{r}_1$`,...,`$\mathtt{r}_n$`]` où, pour tout $k \in [\![1, \mathtt{n}]\!]$, $\mathtt{r}_k$ est le nombre de boules rouges à l'issue du $k^{\text{ième}}$ tirage.
   *Indication : sachant qu'il y a $\mathtt{r}_k$ boules rouges à l'issue du $k^{\text{ième}}$ tirage, la probabilité de tirer une boule rouge au $(k+1)^{\text{ième}}$ tirage est $\frac{\mathtt{r}_k}{\mathtt{r+b}}$.*
2) Écrire un programme qui demande à un entier `N` et les nombres `r,b,n`, qui simule `N` fois cette expérience et qui renvoie un vecteur `[`$\mathtt{m}_1$`,...,`$\mathtt{m}_n$`]` où, pour tout $k \in [\![1, \mathtt{n}]\!]$, $\mathtt{m}_k$ est la moyenne des valeurs prises par $\mathtt{r}_k$ au cours de ces `N` expériences.

**Exercice 6 (temps d'attente du $k^{\text{ième}}$ Pile).**

1) Écrire un programme qui prend en entrée un réel $p$ de $]0,1[$ et un entier $k \in \mathbb{N}^*$ et qui simule l'expérience aléatoire dont le résultat est le temps d'attente du $k^{\text{ième}}$ pile lors de lancers successifs d'une pièce truquée de sorte que Pile apparaît avec la probabilité $p$.
2) On réalise $N = 10000$ fois cette expérience. Calculer une valeur approchée du nombre moyen de lancers réalisés à chaque expérience.

**Exercice 7.** On dispose d'une urne contenant une seule boule de couleur rouge. On répète indéfiniment le jeu suivant : on tire uniformément une boule dans l'urne, on note sa couleur et on ajoute une nouvelle boule de couleur bleue. Pour tout $k \in \mathbb{N}^*$, au moment du $k^{\text{ième}}$ tirage, l'urne contient donc une boule rouge et $k-1$ boules bleues. On suppose que les tirages sont indépendants.
*On pourra se référer à partie C du DM n° 13.*

1) Écrire une fonction qui prend en entrée un entier $n$ strictement positif et qui renvoie un vecteur décrivant les $n$ premiers tirages selon cette règle.
   *On pourra coder un tirage rouge par 1 et un tirage bleu par 0.*
2) Modifier la fonction pour qu'elle renvoie également le nombre moyen de boules rouges tirées lors des $n$ premiers tirages.
3) On recommence le jeu en modifiant un peu les règles : désormais on remplit l'urne de sorte que, au moment du $k^{\text{ième}}$ tirage, il y ait $k^2-1$ boules bleues dans l'urne et toujours une seule boule rouge. Écrire une fonction qui prend un entier naturel $n$ non nul en entrée et qui renvoie un vecteur décrivant les $n$ premiers tirages dans l'urne selon la règle de l'énoncé ainsi que le nombre moyen de boules rouges tirées lors des $n$ premiers tirages. Commentez.