TP6 : Probabilités finies et simulations

1. La fonction `rand()`

La fonction `rand()` est générateur de nombres pseudo-aléatoires suivant la loi uniforme sur [0,1]. La syntaxe est simple :

```
-->rand()
 ans  =
    0.7011980
```

On peut aussi générer des matrices de distribution uniforme sur [0,1] en spécifiant les dimensions :

```
-->rand(2,3)
 ans  =
    0.0683740    0.6623569    0.1985144
    0.5608486    0.7263507    0.5442573
```

Si l'on donne en argument une matrice de taille $n \times m$ alors la fonction `rand()` retourne une matrice de même taille et de distribution uniforme sur [0,1] :

```
-->A=zeros(2,4)
 A  =
    0.    0.    0.    0.
    0.    0.    0.    0.

-->rand(A)
 ans  =
    0.4826472    0.5015342    0.6325745    0.0437334
    0.3321719    0.4368588    0.4051954    0.4818509
```

Remarque :
Pour éviter d'avoir la même suite de nombres aléatoires à chaque mise en route de *Scilab*, on peut utiliser l'instruction :

```
rand("seed",getdate("s"))
```

qui permet de réinitialiser le générateur de nombres aléatoires.

2. Loi uniforme sur $[\![a, b]\!]$

Propriété : Si $(a, b) \in \mathbb{Z}$ (avec $a < b$) alors l'instruction :

```
a+floor((b-a+1)*rand())
```

génère un nombre aléatoire suivant la loi uniforme sur $[\![a, b]\!]$.

Exemple (à connaître par cœur), lancer d'un dé équilibré à 6 faces :

```
1+floor(6*rand())
```

Exemple (à connaître par cœur), loi de Bernoulli $\mathcal{B}\left(\frac{1}{2}\right)$:

```
floor(2*rand())
```

Remarques :

(a) Une autre méthode consiste à utiliser la fonction `grand` :

```
grand(n, p, 'uin', a, b)
```

génère une matrice de taille $n \times p$ dont les coefficients indépendants suivent une loi uniforme sur $[\![a, b]\!]$.

(b) La fonction `grand` permet de simuler beaucoup d'autres lois (voir mémo).

Exercice :
Que fait la suite d'instructions suivante ?

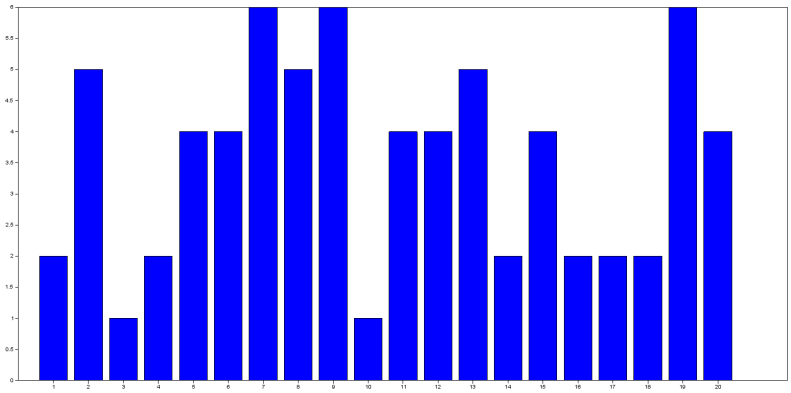
```
x=floor(2*rand(100,1))
mean(x)
sum(x)
```

3. Graphiques et simulations (bar-histplot)

L'instruction `bar(x,y)` représente le diagramme en rectangles des éléments du vecteur ligne `y` (en ordonnée) en fonction des éléments du vecteur ligne `x` (en abscisse).

Exemple à tester et comprendre :

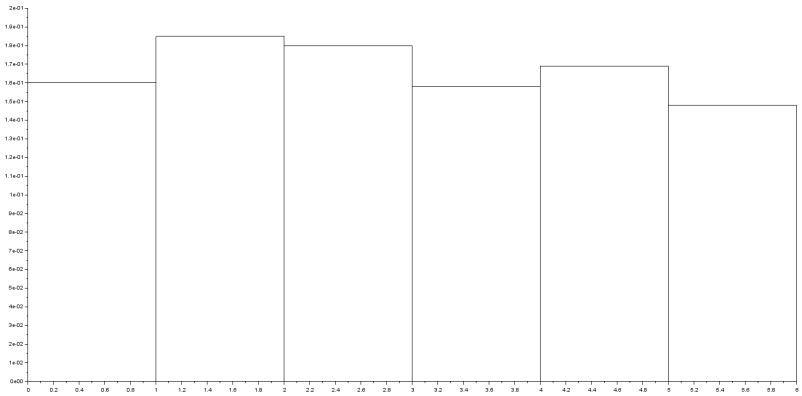
```
x=1:20;
y=1+floor(6*rand(x));
bar(x,y)
```



L'instruction `histplot(x,y)` représente l'histogramme des fréquences d'apparition des éléments du vecteur `y` par classes représentées par le vecteur `x`.

Exemple à tester et comprendre :

```
clf // clf permet d'effacer la fenêtre graphique
x=1:1000;
y=1+floor(6*rand(x));
histplot(0:6,y)
```



4. Exercices

Exercice 1. (Matrice d'entiers aléatoires)
Sans utiliser la fonction `grand`, créer une fonction `alea(a,b,n,p)` qui prend en argument des entiers `a`, `b`, `n` et `p` et qui retourne une matrice de taille `n`×`p` dont tous les coefficients indépendants suivent la loi uniforme sur $[\![a, b]\!]$.

Exercice 2. (Tirages dans une urne)
On considère une urne U contenant $\frac{1}{4}$ de boules rouges, $\frac{1}{3}$ de boules vertes et $\frac{5}{12}$ de boules bleues. On effectue $n \in \mathbb{N}^*$ tirages avec remises dans cette urne et on note X_i la variable aléatoire égale à 1 si le $i^{ème}$ tirage donne une boule rouge, 2 si c'est une boule verte et 3 si c'est une boule bleue.

1. Créer un programme qui demande à l'utilisateur un nombre entier naturel n et qui simule ensuite l'expérience en affichant les valeurs de X_i.
   (On pourra générer un nombre aléatoire suivant la loi uniforme sur $[\![1, 12]\!]$*.)*
2. Calculer mathématiquement l'espérance de X_1 ?
3. Modifier le programme de la question 1. pour qu'il affiche la moyenne des X_i. Que constatez-vous pour n grand ?

Exercice 3. (Loi de Bernoulli et loi binomiale)

1. Créer une fonction `Bern` qui prend en argument un nombre $p \in]0, 1[$ et retourne ensuite la valeur prise, lors d'une réalisation, par une variable aléatoire suivant une loi de Bernoulli $\mathcal{B}(p)$.
2. Créer à la suite une fonction `Bino` qui prend en argument un entier $n \in \mathbb{N}$ et un nombre $p \in]0, 1[$ et qui affiche ensuite la valeur prise par une variable S suivant la loi binomiale $\mathcal{B}(n, p)$.
3. Ajouter à la suite du programme des instructions pour demander à l'utilisateur n et p et effectuer ensuite 10000 réalisations de S puis afficher l'histogramme de ces réalisations.
   (On pourra concaténer les valeurs prises par S dans une variable T, puis afficher `histplot(-1:n,T)`).

Exercice 4. (Le Duc de Toscane)
Au début du XVII^{ème} siècle, le Duc de Toscane a observé que lorsque l'on lance trois dés à 6 faces, on obtient plus souvent une somme égale à 10 que 9, alors qu'il y a autant de façons d'obtenir 10 que 9. Le Duc ayant fait part de ce problème à Galilée ce dernier rédigea un mémoire sur les jeux de hasard en 1620 dans lequel il expliqua ce phénomène.

1. Écrire une fonction `toscane(n)` qui simulera `n` lancers de trois dés, calculera la somme, comptera et affichera le nombre et le pourcentage des réalisations égales à 9 ainsi que le nombre et le pourcentage des réalisations égales à 10.
2. Expliquer ce phénomène.

Exercice 5. (Le lièvre et la tortue)
Un lièvre et une tortue partent d'un point O d'abscisse 0 sur un axe gradué et se déplacent uniquement vers la droite. On lance un dé équilibré à 6 faces. Si le résultat vaut $k \in [\![1, 5]\!]$ alors la tortue avance d'une unité et si le résultat vaut 6 le lièvre avance de 6 unités. Le gagnant est celui qui arrive en premier au point d'abscisse 6.

1. Déterminer mathématiquement la probabilité que la tortue gagne une partie.
2. Créer une fonction `y=course()` qui simule une partie et retourne 1 si la tortue gagne et 0 sinon.
   (On pourra créer deux variables `t` et `l` initialisée à 0 et lancer un dé tant que `t<6` et `l<6`, puis incrémenter `l` si le dé donne 6 et incrémenter `t` sinon).
3. À la suite du programme `y=course()`, écrire une suite d'instructions demandant à l'utilisateur un entier naturel $n \in \mathbb{N}^*$ et permettant de simuler n courses. Le programme devra alors afficher la fréquence des courses gagnées par la tortue.
   Corroborer le résultat trouvé à la question 1.
4. On considère maintenant un dé à N faces. Le gagnant est alors le premier à arriver au point d'abscisse N. En notant t_N la probabilité que la tortue gagne une course, déterminer mathématiquement $\lim\limits_{N\to+\infty} t_N$.

Exercice 6. (Tirage sans remise - Permutation de $[\![1, n]\!]$) $*$
Créer un programme qui demande à l'utilisateur un entier naturel $n \in \mathbb{N}^*$ et qui affiche ensuite une permutation de $[\![1, n]\!]$.
(On pourra créer le vecteur `x=1:n` et commencer par choisir au hasard le dernier élément `k` $\in [\![1, n]\!]$ de la permutation et échanger sa place avec le dernier élément de $[\![1, n]\!]$. Puis recommencer avec l'avant dernier,

Exercice 7. (Inspiré d'EDHEC 2000)
Soit $n \in \mathbb{N}$ avec $n \geqslant 2$. On lance n fois une pièce donnant "Pile" avec la probabilité $p \in]0,1[$ et "Face" sinon. Pour tout entier naturel $k \geqslant 2$, on dira que le $k^{ème}$ lancer est un changement s'il amène un résultat différent du $(k-1)^{ème}$ lancer. On note alors X_n la variable aléatoire égale au nombre de changements survenus au cours des n lancers.

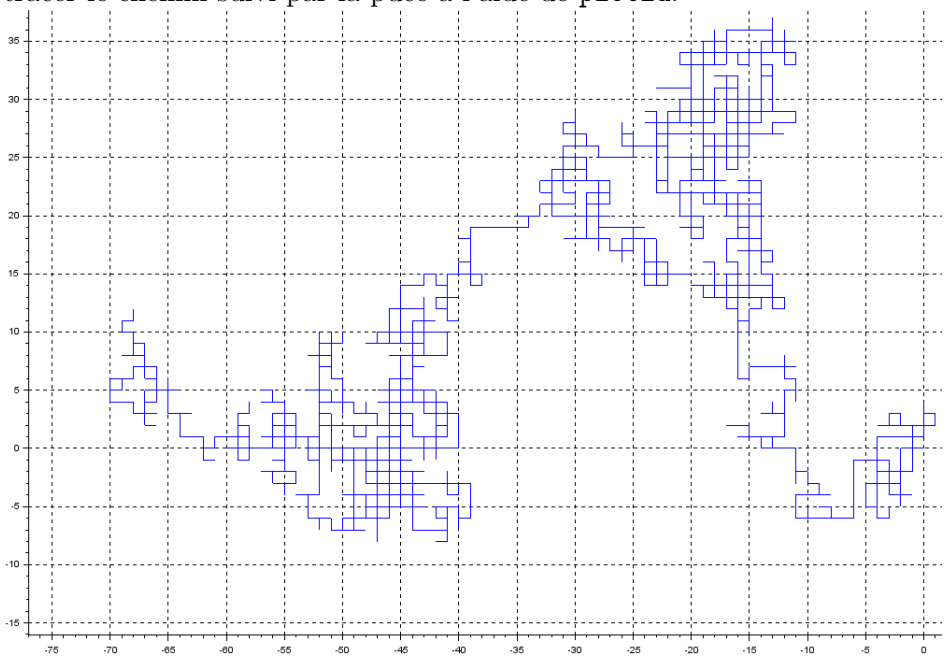
1. Écrire une fonction `y=bern(p)` qui simule une épreuve de Bernoulli $\mathcal{B}(p)$.
2. Écrire une fonction `X=CountCh(n,p)` qui calcule et retourne la valeur prise par X_n.
3. À la suite de ces fonctions ajouter une suite d'instructions qui calculera `CountCh(10000,p)` pour `p` allant de 0,1 à 0,9 avec un pas de 0,1 et qui affichera ensuite le diagramme en rectangles.

Exercice 8. (La puce et la droite - marche aléatoire 1D)
Une puce se déplace par sauts successifs sur une droite graduée. Initialement située sur l'origine, elle se déplace à chaque saut vers la gauche ou la droite de manière équiprobable.

1. Écrire un programme qui simule une loi de Bernoulli de paramètre $\frac{1}{2}$.
2. On note p le vecteur représentant les abscisses de la puce au cours de n sauts. Compléter le programme précédent pour qu'il simule p, lorsque l'entier n est donné par l'utilisateur.
3. Compléter le programme pour qu'il donne la valeur minimale, la valeur maximale et la moyenne de p.
4. Compléter le programme pour qu'il affiche le déplacement de la puce. On prendra en abscisse `0:n` (les sauts) et en ordonnée p.

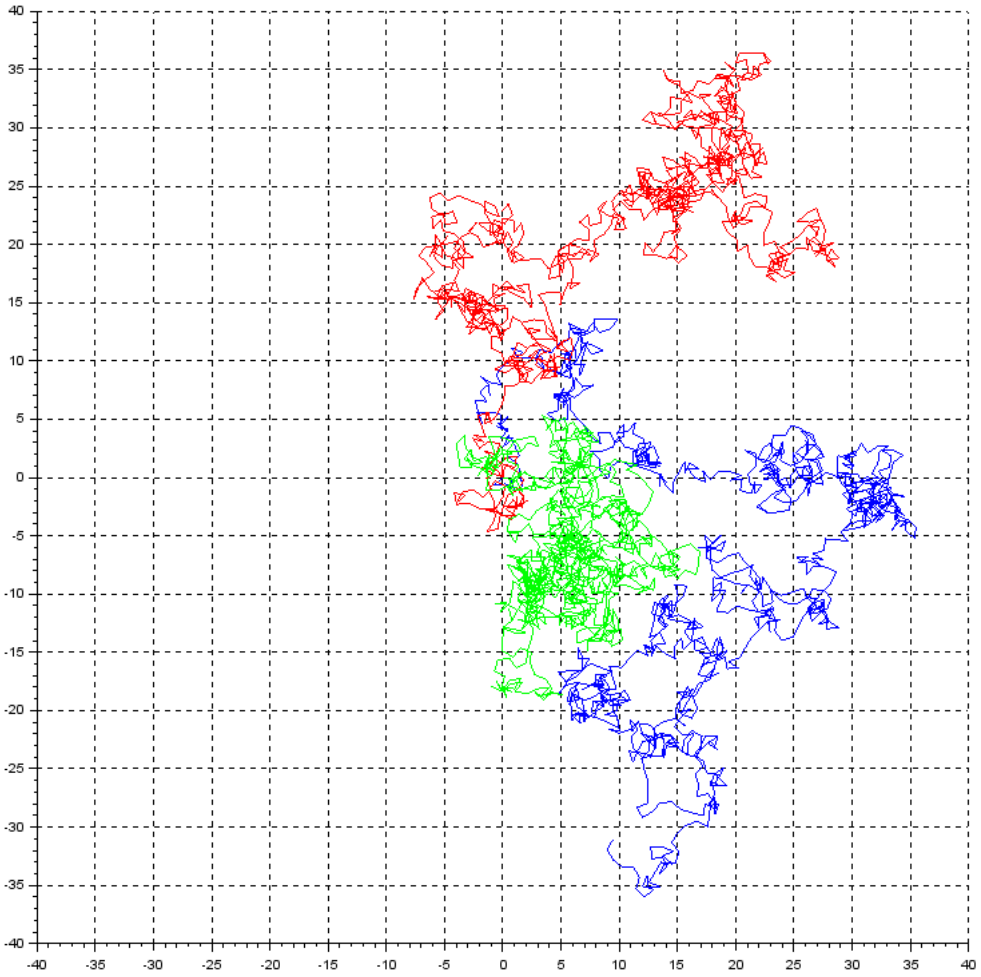
Exercice 9. (La puce et le plan - marche aléatoire 2D)
Une puce se déplace par sauts successifs sur un plan muni d'un repère orthonormal. Initialement sur l'origine, à chaque saut elle se déplace vers le Nord, le Sud, l'Est ou l'Ouest de manière équiprobable.

1. Écrire un programme qui demande à l'utilisateur un entier n et qui calcule ensuite les positions successives (x, y) de la puce à chaque saut jusqu'au $n^{ème}$.
2. Enregistrer ces positions successives dans deux vecteurs (X, Y) et tracer le chemin suivi par la puce à l'aide de `plot2d`.



3. **La marche aléatoire 2D**.
   Recopier, tester et comprendre le principe du code suivant.

```
x=[0]; y=[0]; a=[0]; b=[0]; u=[0]; v=[0]
clf()
plot2d(0,0,0,rect=[-30,-30,30,30],frameflag=3)
xgrid(1)
for i=1:500
    r=2*%pi*rand(); x=[x,x(i)+cos(r)]; y=[y,y(i)+sin(r)]
    s=2*%pi*rand(); a=[a,a(i)+cos(s)]; b=[b,b(i)+sin(s)]
    t=2*%pi*rand(); u=[u,u(i)+cos(t)]; v=[v,v(i)+sin(t)]
    plot2d(x,y,2); plot2d(a,b,5); plot2d(u,v,3);
    sleep(1)
end
```



Exercice 10. (La puce et le carré)
Une puce se déplace par sauts successifs sur un carré $ABCD$. Initialement en A, elle va sur l'un des deux sommets consécutifs avec une probabilité égale à $\frac{3}{8}$ dans les deux cas et elle va sur le sommet diagonalement opposé avec une probabilité égale à $\frac{1}{4}$.

1. Compléter le programme suivant pour qu'il demande à l'utilisateur un entier $n \in \mathbb{N}^*$ et qu'il simule le déplacement de la puce en affichant la suite des sommets successivement visités au cours des n déplacements. (Tester le programme.)

```
n=________
puce=["A","B","C","D"]
p=0; disp(puce(1))
for i=1:n
    //h est un nombre entier aléatoire sur [1,8] :
    h=________
    //Probabilité d'aller dans le sens trigo :
    if ____ then p=p+1; end
    //Probabilité de passer par la diagonale :
    if ____ then p=p+2; end
    //Probabilité d'aller dans le sens non trigo :
    if ____ then p=p+3; end
    disp(puce(modulo(p,4)+1))
end
```

2. Modifier le programme pour qu'il compte le nombre de passages a, b, c, d sur les sommets respectifs A, B, C et D.
3. Afficher le diagramme en rectangles de ces dernières valeurs. Conjecturer avec n grand.
4. On note respectivement An, B_n, C_n et D_n les évènements la puce se trouve sur le sommet A, B, C ou D au bout de n sauts.
   Déterminer $P(A_{n+1})$ en fonction de $P(A_n)$, $P(B_n)$, $P(C_n)$ et $P(D_n)$.
   Faire de même avec $P(B_{n+1})$, $P(C_{n+1})$ et $P(D_{n+1})$.
5. Déterminer une matrice $M \in \mathcal{M}_4(\mathbb{R})$ telle que :

$$U_{n+1} = \begin{pmatrix} P(A_{n+1}) \\ P(B_{n+1}) \\ P(C_{n+1}) \\ P(D_{n+1}) \end{pmatrix} = M \times \begin{pmatrix} P(A_n) \\ P(B_n) \\ P(C_n) \\ P(D_n) \end{pmatrix}$$

   Vérifier que M est bistochastique (pour chaque ligne/colonne la somme des coefficients vaut 1).
6. Démontrer que, pour tout $n \in \mathbb{N}$, $U_n = M^n U_0$
7. Déterminer U_0 et conjecturer, grâce à *Scilab* la probabilité que la puce se trouve sur les sommets A, B, C ou D au bout de 1000 sauts.

En bonus un code pour générer le mouvement brownien en 3D :

```
clf(); T = 5; N = 500; S = sqrt(T/N);
bruitX = grand(N,3,"nor",0, sqrt(T));
bruitY = grand(N,3,"nor",0, sqrt(T));
bruitZ = grand(N,3,"nor",0, sqrt(T));
x=zeros(N,3); y=zeros(N,3); z=zeros(N,3);
for i=1:3
    x(:,i)=S*cumsum(bruitX(:,i))
    y(:,i)=S*cumsum(bruitY(:,i))
    z(:,i)=S*cumsum(bruitZ(:,i))
end
plot2d(0,0,0,rect=[-10,-10,10,10],frameflag=3); xgrid(1);
for i = 1:N
    param3d(x(1:i,1),y(1:i,1),z(1:i,1));
    e=gce(); e.foreground=color('red');
    param3d(x(1:i,2),y(1:i,2),z(1:i,2));
    e=gce(); e.foreground=color('blue');
    param3d(x(1:i,3),y(1:i,3),z(1:i,3));
    e=gce(); e.foreground=color('green');
    sleep(100)
end
```