TP 06 – Simulations V.A. Finies

I Simulation de lois usuelles

Les lois usuelles peuvent être directement simulées grâces aux commandes suivantes, présentes dans le module `random` de la bibliothèque `numpy`, à importer de la manière suivante :

```
import numpy.random as rd
```

Les commandes sont alors les suivantes.

- `rd.random()` : nombre aléatoire de [0,1[
- `rd.randint(a,b)` : loi uniforme sur [a,b[avec a et b des entiers (b exclu)
- `rd.binomial(n,p)` : loi binomiale de paramètres n et p

Pour simuler plusieurs variables aléatoires d'un coup, on peut ajouter aux fonctions `rd.randint(a,b)` et `rd.binomial(n,p)` un troisième argument qui correspond au nombre de simulations voulu. Par exemple, pour obtenir 10 simulations d'une loi uniforme sur $[\![1,6]\!]$, on peut taper

```
Entrée [1]: rd.randint(1,7,10)
Out [1]: array([3, 5, 6, 1, 4, 5, 3, 3, 3, 4])
```

Exercice 1 *Les deux questions de cet exercice sont indépendantes.*

1. Écrire une commande qui renvoie, après exécution, un entier aléatoire entre 1 et 10.

```
rd.randint(1,11)
```

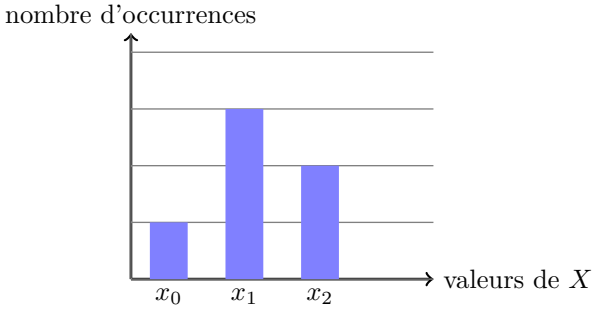
2. Écrire un programme qui renvoie, après exécution, une liste de 15 entiers pris au hasard entre 2 et 20.

```
#Version 1
L = []
for k in range(15):
    L.append(rd.randint(2,21))
print(L)

#Version 2
rd.randint(2,21,15)
```

II Diagrammes en barres

On va maintenant voir comment tracer les diagrammes en barres obtenus quand on réalise un grand nombre de simulations des lois précédentes. Un diagramme en barres pour une variable aléatoire X est un diagramme de la forme :



On utilisera la commande `plt.bar(x,y)` de la bibliothèque `matplotlib` où

- x est la liste des valeurs de X ;
- y est la liste des hauteurs de barres correspondantes.

Si $x = [x_0, x_1, \dots, x_n]$, alors $y = [y_0, y_1, \dots, y_n]$ avec

$$y_i = \text{"nombre de fois où } X \text{ vaut } i \text{ parmi un certain nombre de simulations"}.$$

Ainsi, pour tracer un histogramme, il faudra commencer par importer la bibliothèque suivante :

```
import matplotlib.pyplot as plt
```

III Moyenne, variance, écart-type

Ici X désigne une matrice contenant **plusieurs simulations** d'une variable aléatoire. Les commandes suivantes, de la bibliothèque `numpy` permettent de calculer une espérance, variance et écart-type empirique.

- `np.mean(X)` : moyenne (proche de l'espérance théorique quand il y a beaucoup de simulations.)
- `np.var(X)` : variance empirique
- `np.std(X)` : écart-type empirique

Exercice 2 On considère une urne contenant 5 boules rouges (numérotées 1,2,3,4,5) et 4 boules jaunes (numérotées 6,7,8,9). On réalise 8 tirages **avec remise** dans cette urne. On note

- Pour tout $k \in \{1, \dots, 8\}$, X_k la variable aléatoire donnant le numéro obtenu au k-ième tirage.
- R la variable aléatoire égale au nombre total de boules rouges obtenues.
- J la variable aléatoire égale au nombre total de boules jaunes obtenues.

1. Réaliser **une** simulation de $X_1, X_2, \dots, X_8$. *Il s'agit de reconnaître une loi usuelle.*

```
for k in range(1,9):
    Xk = rd.randint(1,10)
```

2. Réaliser **une** simulation de R. *Il s'agit de reconnaître une loi usuelle.*

```
R = rd.binomial(8, 5/9)
```

3. Réaliser **une** simulation de J. *Il s'agit de reconnaître une loi usuelle.*

```
J = rd.binomial(8, 4/9)
```

4. Calculer la moyenne et la variance empirique de X_1 et R. Comparer avec les valeurs théoriques du cours.

```
#On réalise un grand nombre de simulations de la v.a X1
X1 = rd.randint(1,10, 10000)
#On calcule sa moyenne empirique
moyemp=np.mean(X1)
print('La moyenne empirique de X1 est', moyemp)
#On compare avec sa moyenne théorique
moytheo = (9+1)/2
print('La moyenne theorique de X1 est', moytheo)
#On calcule sa variance empirique
varemp=np.var(X1)
print('La variance empirique de X1 est', varemp)
#On compare avec sa variance théorique
vartheo = (9**2-1)/12
print('La var theorique de X1 est', vartheo)
```

```
#On réalise un grand nombre de simulations de la v.a X1
R = rd.binomial(8, 5/9, 10000)
#On calcule sa moyenne empirique
moyemp=np.mean(R)
print('La moyenne empirique de R est', moyemp)
#On compare avec sa moyenne théorique
moytheo = 8*(5/9)
print('La moyenne theorique de R est', moytheo)
#On calcule sa variance empirique
varemp=np.var(R)
print('La variance empirique de R est', varemp)
#On compare avec sa variance théorique
vartheo = 8*(5/9)*(1-5/9)
print('La var theorique de R est', vartheo)
```

Exercice 3 On considère la même situation que dans l'Exercice 2.

1. Réaliser 1000 simulations de X_1. On appelera la liste créée `X1`.

```
X1 = rd.randint(1,10, 1000)
```

2. Définir la liste x des valeurs de X_1 (abscisses du diagramme).

```
x = range(1,10)
```

3. Écrire une fonction `nombre(L,k)` qui, étant donnée une liste L et un nombre k, renvoie le nombre de fois où k apparaît dans L.

```
#Version 1
def nombre(L,k):
    c = 0
    for x in L:
        if x==k:
            c=c+1
    return c

#Version 2
def nombre(L,k):
    c=0
    for i in range(len(L)):
        if L[i] == k :
            c=c+1
    return c
```

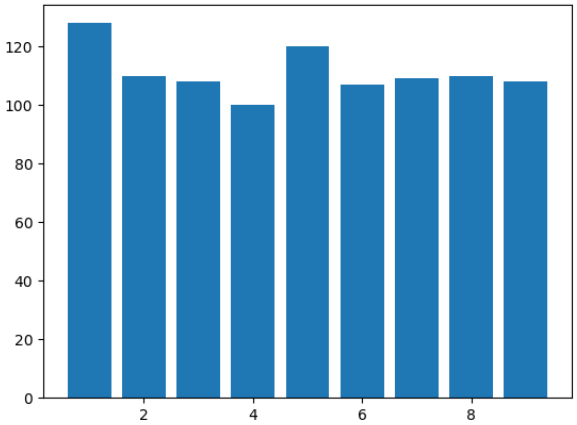
4. Compléter alors le programme suivant pour définir la liste y des hauteurs des barres.

```
y = [0]*len(x) # liste nulle de même taille que x

for i in range(len(y)):
    y[i] = nombre(X1, i+1) #attention au decalage
print(y)
```

5. Enfin, on trace le diagramme en barres.

```
plt.bar(x,y)
plt.show()
```



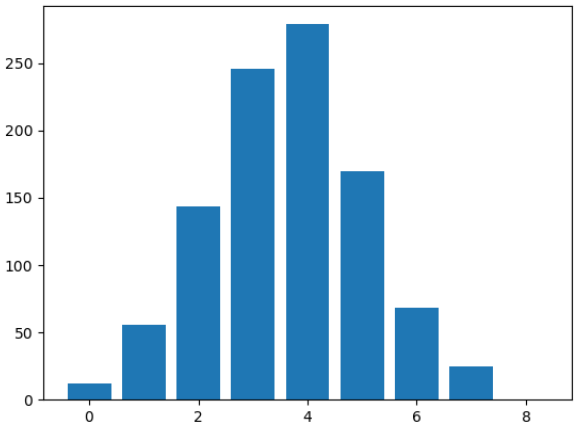
6. Réaliser le diagramme correspondant pour J.

```
#Bcp de simulations de J
J = rd.binomial(8, 4/9, 1000)

#Liste des abscisses
x = range(0, 9)

#Liste des ordonnees
y = [0] *len(x)
for i in range(len(y)):
    y[i] = nombre(J, i) #pas de decalage ici

#Tracer du diagramme en barres
plt.bar(x,y)
plt.show()
```



Exercice 4 On considère une urne contenant 5 boules rouges (numérotées 1,2,3,4,5) et 3 boules jaunes (numérotées 6,7,8).

1. Simuler un tirage dans cette urne (en s'appuyant sur le numérotage des boules et non pas la couleur).

```
rd.randint(1, 9)
```

2. On réalise 7 tirages **avec remise** dans cette urne. On note X la variable aléatoire qui vaut 0 si aucune boule jaune n'est tirée et, sinon, est égale au numéro du tirage où l'on a obtenu la première boule jaune. Écrire une fonction `SimulationX()` qui réalise une simulation de X.

```
def SimulationX():
    #Au debut, la v.a est initialisée à 0
    X = 0
    #On effectue 7 tirages dans l'urne avec remise
    for k in range(1, 8):
        tirage = rd.randint(1,9)
        #si la boule est jaune alors...
        if tirage >= 6 :
            #...on recupère le num du tirage
            X = k
            #... et on s'arrête
            return X
```

3. Construire une liste qui contient 20 simulations différentes de X.

```
L = []
for k in range (1, 21):
    L.append(SimulationX())
print(L)
```

4. Calculer la valeur moyenne de la liste précédente.

```
S = 0
for k in range(20):
    S = S + L[k]
S = 1/20*S
print(S)
```

Exercice 5 On considère une urne contenant 5 boules rouges (numérotées 1,2,3,4,5) et 3 boules jaunes (numérotées 6,7,8). On va cette fois-ci réaliser 7 tirages **sans remise** dans cette urne. On note Y la variable aléatoire égale au rang de la première boule jaune (il y en a forcément eu une).

1. Écrire une fonction `SimulationY()` qui réalise une simulation de Y.

```
def SimulationY():
    #on garde une trace du nbre de boules rouges...
    rouges = 5
    #et de boules jaunes...
    jaunes = 3
    #On effectue 7 tirages dans l'urne sans remise
    for k in range(1,8):
        #attention, le nbre de boules dans l'urne change à
chaque tirage
        tirage = rd.randint(1, rouges+jaunes+1)
        #si la boule est jaune...
        if tirage > rouges :
            #on recupere le num du tirage
            Y = k
            #et on s'arrete
            return Y
        #sinon on continue
        else :
            #et on enleve la boule rouge tiree
            rouges = rouges - 1
```

2. Construire une liste qui contient 20 simulations différentes de Y.

```
L = []
for k in range (1, 21):
    L.append(SimulationY())
print(L)
```

Exercice 6 On lance trois fois de façon indépendante une pièce donnant Pile avec la probabilité 1/3. On note X la variable aléatoire donnant le nombre de Pile obtenu au cours de ces trois lancers.

1. Pour simuler un tel lancer de pièce numériquement, on peut tirer un nombre au hasard entre 0 et 1. Si ce nombre est plus petit que 1/3, on décrète que l'on obtient Pile, sinon, cela signifie qu'on a obtenu Face. La simulation d'un lancer de cette pièce peut se faire alors grâce au programme suivant.

Entrée [2]:
```
import numpy.random as rd

x = rd.random()
if x <1/3:
    piece = "Pile"
else :
    piece = "Face"
```

En s'appuyant sur ce programme, compléter le programme suivant pour que l'exécution de `simul_X()` renvoie une réalisation de la variable aléatoire X (qui compte le nombre de Pile total pendant l'expérience).

Entrée [3]:
```
def simul_X():
    #initialisation du nbre de pile avant le debut de l'experience
    X = 0
    #On réalise trois lancers de pieces
    for k in range(1,4):
        #on realise "un" lancer
        x = rd.random()
        #si on obtient pile...
            if x>1/3:
                # la valeur de X augmente d'un
                X = X+1
    #On renvoie à la fin le nbre de pile total
    return(X)
```

Exercice 7 (Adapté d'Ecricome 2015 Maths E) On considère une urne contenant 4 boules blanches (numérotées de 2 à 4) et une boule noire (numérotée 1). On effectue des tirages sans remise dans cette urne, jusqu'à l'obtention de la boule noire. On note X la variable aléatoire qui prend pour valeur le nombre de tirages nécessaires pour l'obtention de la boule noire.

1. Que vaut $X(\Omega)$?

$$X(\Omega) = \{1, \ldots, 5\}$$

2. Recopier et compléter le programme Python pour qu'il simule la réalisation de 10 000 répétitions de cette expérience aléatoire, en rajoutant à chaque fois dans une liste la valeur prise par la variable aléatoire X.

Entrée [4]:
```
Liste=[]
for k in range(10000):
    i = 1 #nbre tirages
    M = 5 #nbre boules dans l'urne
    while rd.randint(1,M+1) >1:
        i = i+1
        M= M-1
    Liste.append(i)
```

Exercice 8 [**Ecricome 2016**] Une urne contient initialement 1 boule rouge et 1 boule blanche. On effectue une succession d'épreuves, chaque épreuve étant constituée des trois étapes suivantes :

- on pioche une boule au hasard dans l'urne,
- on replace la boule tirée dans l'urne,
- on rajoute dans l'urne une boule de la même couleur que celle qui vient d'être piochée.

Après n épreuves, l'urne contient donc $n+2$ boules. Pour tout $n \in \mathbb{N}^*$, on note X_n le nombre de boules rouges qui ont été **ajoutées** dans l'urne (par rapport à la composition initiale) à l'issue des n premières épreuves.

1. Compléter la fonction suivante, qui simule le tirage d'une boule dans une urne contenant x boules rouges (numérotées $1, 2, \ldots, x$) et y boules blanches (numérotées $x+1, \ldots, x+y$) et qui retourne la valeur 0 si la boule est rouge et 1 si elle est blanche.

```
def tirage(x,y):
    alea = rd.randint(1, x+y+1)
    if alea <= x:
        return 0
    else:
        return 1
```

2. Écrire une fonction `experience(n)` qui réalise une simulation de X_n.

```
def experience(n):
    #Variable donnant le nbre de boules rouges
    x = 1
    #Variable donnant le nbre de boules blanches
    y = 1
    #Variable donnant le nbre de boules rouges ajoutées
    X = 0
    #on réalise n épreuves
    for k in range(n):
        #on realise un tirage dans l'urne
        t = tirage(x,y)
        #si on a obtenu une boule rouge...
        if t == 0 :
            #... on ajoute une boule rouge dans l'urne
            x = x + 1
        else :
            #...sinon on ajoute une boule blanche
            y = y + 1
    #le nbre de boule rouge ajoutée à la fin vaut...
    X = x - 1
    return X
```

3. Réaliser 10 000 simulations de X_5 puis tracer le diagramme en barres correspondant.

```
## Simulations
X = np.array([experience(5) for k in range(10000)])

## Diagramme en barres
# abscisses = valeurs possibles pour X5
x = range(0,6)
y = [0]*len(x)
for i in range(0,6):
    y[i] = np.sum(X == i) # nombre de fois où X = i
plt.bar(x,y)
plt.show()
```