Informatique - Simulation de variables aléatoires discrètes

Pour simuler des expériences aléatoires discrètes, on utilise le module `numpy.random` qu'il faut importer.

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

Python

- La commande `"rd.rand()"` renvoie un nombre réel aléatoire entre 0 et 1.
- La commande `"rd.rand(n,m)"` renvoie une matrice $n \times m$ dont chaque terme est un réel aléatoire entre 0 et 1.
- Soit p un réel fixé entre 0 et 1.

La commande `"rd.rand() < p"` simule un évènement de probabilité p.

- Exercice 1 (Lois usuelles discrètes) -

1. Écrire une fonction `"Bernoulli(p)"` prenant comme paramètre un réel $p \in [0;1]$ et simulant une loi de Bernoulli de paramètre p.
2. Écrire une fonction `"Binomiale(n,p)"` prenant comme paramètre un entier n et un réel $p \in [0;1]$ et simulant une loi binomiale de paramètres n et p.
3. Écrire une fonction `"Geometrique(p)"` prenant comme paramètre un réel $p \in [0;1]$ et simulant une loi géométrique de paramètre p.

- Exercice 2 -

Le gain d'un joueur sur un jeu de grattage (prix du billet inclus) est modélisé par une variable aléatoire X dont la loi est donnée par : $X(\Omega) = \{-2,0,2,6\}$ et $P(X=-2) = \frac{1}{2}$, $P(X=0) = \frac{1}{4}$, $P(X=2) = \frac{1}{6}$, $P(X=6) = \frac{1}{12}$.

1. (a) En utilisant judicieusement la commande `"rd.rand()"`, écrire une fonction `"SimuX()"` qui renvoie une simulation de la variable X.

   (b) Quelle valeur est affichée par ce programme?

```
S = 0
for k in range(1000) :
    S += SimuX()
print( S/1000 )
```

2. (*) S'inspirer de l'exemple précédent pour simuler une loi de Poisson de paramètre λ.

Python

Soit X une variable aléatoire suivant la loi binomiale de paramètres n et p.

- La commande `"rd.binomial(n, p)"` renvoie une réalisation de X.
- La commande `"rd.binomial(n, p, k)"` renvoie un vecteur contenant `"k"` réalisations de X.

Soit Y une variable aléatoire suivant la loi géométrique de paramètre p.

- La commande `"rd.geometric(p)"` renvoie une réalisation de Y.
- La commande `"rd.geometric(p, k)"` renvoie un vecteur contenant `"k"` réalisations de Y.

Soit Z une variable aléatoire suivant la loi de Poisson de paramètre λ (noté `lbd`).

- La commande `"rd.poisson(lbd)"` renvoie une réalisation de Z.
- La commande `"rd.poisson(lbd, k)"` renvoie un vecteur contenant `"k"` réalisations de Z.

- Exercice 3 -

1. La commande `"np.mean(L)"` renvoie la moyenne empirique de la liste `L`. Donner une approximation de la valeur affichée par le programme suivant.

```
np.mean(rd.geometric(0.25 ,10000))
```

2. La commande `"np.count_nonzero(condition)"` compte le nombre de fois que la "condition" est vraie dans une liste. Quelle est la variable aléatoire simulée par la variable `"X"` dans le programme suivant?

```
R = rd.rand(1,20)
X = np.count_nonzero(R<0.25)
```


- Exercice 4 -

On lance n fois une pièce équilibrée ($n \geqslant 2$), les lancers étant supposés indépendants. On note Z la variable aléatoire qui vaut 0 si l'on n'obtient aucun pile pendant ces n lancers et qui prend pour valeur le rang du premier pile dans le cas contraire.

1. Compléter la fonction suivante permettant de simuler la variable aléatoire Z, le nombre n de lancers étant donné en paramètre.

```
def SimuZ(n) :
    for i in range(1, n+1) :
        if ........ :
            return ........
    return ........
```

2. Écrire un programme renvoyant une approximation de l'espérance de Z.

- Exercice 5 -

Deux individus A et B s'affrontent dans un jeu de Pile ou Face dont les règles sont les suivantes :

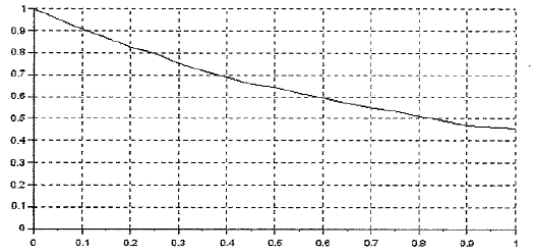
- le joueur A dispose d'une pièce amenant Pile avec la probabilité $\frac{2}{3}$ et lance cette pièce jusqu'à l'obtention du deuxième Pile ; on note X la variable aléatoire prenant la valeur du nombre de Face alors obtenus ;
- le joueur B dispose d'une autre pièce amenant Pile avec la probabilité $p \in]0,\ 1[$ et lance cette pièce jusqu'à l'obtention d'un Pile ; on note Y la variable aléatoire prenant la valeur du nombre de Face alors obtenus ;
- Le joueur A gagne si son nombre de Face obtenus est inférieur ou égal à celui de B ; sinon c'est le joueur B qui gagne.

On dit que le jeu est équilibré lorsque les joueurs A et B ont la même probabilité de gagner.

1. Écrire une fonction Python `"simule_X()"` qui simule la variable aléatoire X.
2. Écrire une fonction Python `"simule_Y(p)"` qui prend en argument un réel p de $]0;1[$ et simule la variable aléatoire Y.
3. Expliquer ce que renvoie la fonction suivante :

```
def mystere(p) :
    r = 0
    N = 10**4
    for k in range(1,N+1) :
        x = simule_X()
        y = simule_Y(p)
        if x <= y :
            r = r + 1/N
    return( r )
```

4. On trace, en fonction de p, une estimation de la probabilité que A gagne et on obtient le graphe suivant :



À la vue de ce graphe, conjecturer une valeur de p pour lequel le jeu serait équilibré.

Python 1

Soit X une variable aléatoire suivant la loi uniforme discrète sur $[\![m;n]\!]$: $X \hookrightarrow \mathscr{U}([\![m;n]\!])$.
La commande `"rd.randint(m, n+1)"` renvoie une simulation de X.

- EXERCICE 6 -

On dispose de n urnes, numérotées de 1 à n. Pour chaque $k \in [\![1;n]\!]$, l'urne k contient k boules numérotées de 1 à k. On choisit une urne au hasard puis on tire une boule dans cette urne. On note X_n la variable aléatoire qui prend la valeur du numéro de la boule piochée.
Écrire une fonction `"simuX(n)"` qui simule X_n pour une valeur de n donnée en paramètre.

- EXERCICE 7 -

Soit N un entier supérieur ou égal à 3. On considère une urne qui contient une seule boule noire numérotée 1 et $(N-1)$ boules blanches numérotées de 2 à N et .
On effectue des tirages sans remise jusqu'à l'obtention de la boule noire et on note X la variable aléatoire qui donne le nombre de tirages nécessaires.

1. Compléter la fonction suivante pour qu'elle simule X pour une valeur de N donnée en paramètre.

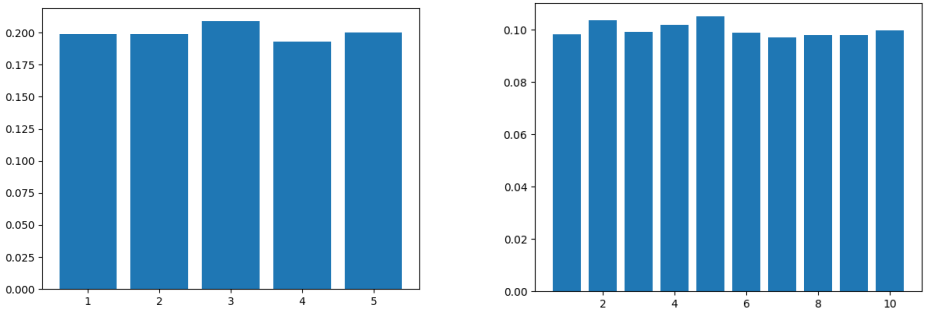
```
def simuX(N) :
    nbr_boules = N
    c = 0
    for i in range(1,N+1) :
        tirage = rd.randint(.....................)
        c = ............
        if tirage == 1 :
            return ............
        else :
            nbr_boules = ............
```

2. Si `X` et `Y` sont deux listes, la commande `"plt.bar(X,Y)` trace le diagramme en bâtons d'abscisse `X` et d'ordonnée `Y`.
Compléter le programme suivant pour qu'il trace le diagramme en bâton de X, à partir de ses fréquences.

```
import matplotlib.pyplot as plt
N = 5
freqX = np.zeros(.............)
for k in range (10000) :
    i = simuX(N)
    freqX[........] += 1
freqX =  freqX /10000

plt.bar( range(1, N+1),  freqX )  # diagramme en batons
plt.show ()
```

3. On obtient les diagrammes en bâtons ci-dessous pour $N = 5$ et $N = 10$. Que peut-on conjecturer sur la loi de X ?



- EXERCICE 8 -

Une urne contient initialement 1 boule rouge et 1 boule blanche. On effectue une succession d'épreuves selon le protocole suivant.

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

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. (a) Écrire une fonction `"tirage(x,y)"` qui simule le tirage d'une boule dans une urne contenant `"x"` boules rouges et `"y"` boules blanches et qui retourne la valeur 0 si la boule est rouge et 1 si elle est blanche.

   (b) Compléter la fonction suivante, qui simule n tirages successifs dans une urne contenant initialement 1 boule rouge et 1 boule blanche (selon le protocole décrit ci-dessus) et qui retourne la valeur de X_n :

```
def experience(n) :
    x = 1
    y = 1
    for k in range(1,n+1) :
        r = tirage(x,y)
        if r == 0 :
            x = ..........
        else :
            ...........
    return( .......... )
```

   (c) Donner l'ensemble $X_n(\Omega)$.

   (d) On souhaite estimer la loi de X_n. Pour cela, on va construire un vecteur nommé *loi* contenant les approximations de $P(X_n = 0)$, $P(X_n = 1), \ldots, P(X_n = n)$. Pour obtenir ces approximations, on va simuler m fois (m grand) l'expérience grâce à la fonction `"experience"` précédente.
   Compléter la fonction suivante pour que la variable *loi* contienne en sortie les approximations de $P(X_n = 0)$, $P(X_n = 1), \ldots, P(X_n = n)$.

```
def simulation(n,m) :
    loi = np.zeros(............)
    for k in range( ............. ) :
        Xn = experience(n)
        loi(......) = loi(......) + 1
    end
    loi = loi / .....
    return(loi)
```

2. (a) On considère les instructions suivantes et leurs sorties.
   Pour $n = 2$ et $m = 10000$:

```
>>> simulation(2,10000)
array([0.3331, 0.3317, 0.3352])
```

   Pour $n = 4$ et $m = 10000$:

```
>>> simulation(4,10000)
array([0.2051, 0.1897, 0.2053, 0.2029, 0.197 ])
```

   Que peut-on conjecturer sur la loi de X_n ?

   (b) Déterminer la loi de X_1 et vérifier votre conjecture lorsque $n = 1$.

   (c) En utilisant la formule des probabilités totales, déterminer la loi de X_2 et vérifier votre conjecture lorsque $n = 2$.