T.P. VIII - Variables aléatoires

Code Capytale : `d3ff-2870060`

I - Ce qu'il faut savoir

II - Fonction de répartition

Solution de l'exercice 1. [D'après Ecricome - 2016 - Exercice 3]

1.

```
import numpy as np
import matplotlib.pyplot as plt

X = np.arange(0, 23)
U = np.zeros((23, 1))
for k in range(3, 23):
    U[k] = 1/2 * ((4/5)**(k-2) - (2/3)**(k-2))

plt.figure()
plt.plot(X, U, '+')
plt.show()
```

2.

```
import numpy as np
import matplotlib.pyplot as plt

X = np.arange(0, 23)
U = np.zeros((23, 1))
for k in range(3, 23):
    U[k] = 1/2 * ((4/5)**(k-2) - (2/3)**(k-2))

plt.figure()
plt.plot(X, U, '+')
plt.show()
```

3. Comme $\mathbf{P}(X \leqslant m) + \mathbf{P}(X > m) = 1$, on cherche un réel m pour lequel $\mathbf{P}(X \leqslant m) = 0{,}5$. On obtient graphiquement $m \simeq 7$. $\square$

Solution de l'exercice 2.

1.

```
import numpy as np
import matplotlib.pyplot as plt

def F(x):
    if x < 1:
        return 0
    else:
        return 1 - 1/x**2 - 2 * np.log(x)/x**2
```

2.

```
A = np.zeros((300, 1)) # A matrice à
1 ligne, 300 colonnes ne contenant que des 0
B = np.zeros((300, 1)) # B matrice à
1 ligne, 300 colonnes ne contenant que des 0

for i in range(0, 300): # i varie entre 0 et 299
    A[i] = - 2 + 7 * i / 300
    B[i] = F(A[i])

# A contient [-2, -2 + 7/300, -2 + 2 * 7/300, ...., -2 + 299 * 7/
# B contient [F(-2), F(-2 + 7/300), ..., F(-2 + 299 * 7/300)]

plt.figure()
plt.plot(A, B) # Trace B en fonction de A
plt.show()
```

Cette suite d'instructions permet d'obtenir un tracé du graphe de F. $\square$

III - Loi discrète uniforme

Solution de l'exercice 3. [D'après Ecricome - 2018 - Exercice 3]

```
import numpy.random as rd
def simulation():
    tirage1 = rd.randint(1, 4)
    if tirage1 < 3:
        res1 = 1
        tirage2 = rd.randint(1, 5)
        if tirage2 == 1:
            res2 = 1
        else:
            res2 = 0
    else:
        res1 = 0
        tirage2 = rd.randint(1, 4)
        if tirage2 < 3:
            res2 = 1
        else:
            res2 = 0
    return res1 + res2
```

□

Solution de l'exercice 4.

```
import numpy.random as rd

def  simulT():
    T = 0 # nombre de boules noires déjà tirées

    # randint(1, 3) : renvoie 1 ou 2 avec même probabilité
    # modélise le lancer de la pièce
    if rd.randint(1, 3) == 1: # si la pièce est tombée sur Pile
        # on effectue 2 tirages dans l'urne U
        # on numérote 1 la boule noire et 2, 3, 4 les boules blan
        for k in range(1, 3): # k varie entre 1 et 2 : 2 tirages
            # randint(1, 5) : entier aléatoire entre 1, 2, 3, ou
            # numéro de la boule tirée
            if rd.randint(1, 5) < 2: # si l'entier vaut 1 i.e. si
                T = T + 1 # on ajoute 1 au nombre de boules noire
    else : # si la pièce est tombée sur Face
        # on effectue 2 tirages dans l'urne V
        # on numérote 1, 2 les boules noires et 3, 4 les boules bl
        for k in range(1, 3): # k vair enter 1 et 2 : 2 tirages
            if rd.randint(1, 5) < 3: # validé si le numéro vaut 1
                T = T + 1 # on ajoute 1 au nombre de boules noires
    return T

print("Une simulation de la variable aléatoire T donne : ", simulT
```

□

IV - Loi discrète non unifome

Solution de l'exercice 5.

1.

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

A = np.zeros((101, 1)) # A contient que des 0 : stocke les valeurs
for k in range(0, 101): # k varie de 0 à 100
    t = rd.randint(1, 5)
    # t vaut 1, 2, 3 ou 4, chacun avec même probabilité
    if t <= 2:
        A[k] = 1 # le saut est de 1 unité
    elif t == 3:
        A[k] = 2 # le saut est de 2 unités
    else:
        A[k] = 3 # le saut est de 3 unités

print(A)
```

2.

```
import matplotlib.pyplot as plt

X = np.arange(0, 101) # X = [0, 1, 2, ..., 100]
# A = [s_0, s_1, s_2,...., s_100]
Y = np.cumsum(A)
# Y = [s_0, s_0 + s_1, s_0 + s_1 + s_2,...., s_0 + s_1 + ... + s_
# Y est la liste des positions de la puce
```

```
plt.figure()
plt.plot(X, Y) # trace la position de la puce en fonction du temps
plt.show()
```

Le graphique représente donc la position de la puce au cours du temps.

□

Solution de l'exercice 6. [D'après BCE ESCP - 2020 - Exercice 4]

```
import numpy.random as rd
k = 1 # puce en position k - 1
hasard = rd.randint(1, k+2) # nombre entre 1 et k+1

while hasard <= k: # pas de retour en 0 avec proba k/k+1
    k = k + 1 # se deplace vers la droite
    hasard = rd.randint(1, k+2) # retire au hasard

print("U a pris la valeur :", k)
```

□

Solution de l'exercice 7.

1.

```
def geom():
    X = 1
    while rd.random() <= 3/5 : # tant qu'il y a des échecs
        X = X + 1
    return X
```

2.

```
def simulZ():
    X1 = geom()
    X2 = geom()
    if X1 > X2:
        Z = X1
    else:
        Z = X2
    return Z
```

□

V - Variables aléatoires à densité

Solution de l'exercice 8.

1. `rd.random()` renvoie des réels entre 0 et 1. Donc `4 * a2 * rd.random()` renvoie des réels entre 0 et $4a^2$. Comme `rd.random()` ne favorise aucun intervalle, il en va de même de `rd.random() * 4 * a2` donc `rd.random() * 4 * a**2` qui simule donc une loi uniforme sur $[0, 4a^2]$.
On pourrait démontrer rigoureusement cette assertion en calculant les fonctions de répartition.

2.

```
np.sqrt(rd.random() * 4 * a**2)
```

3.

```
import numpy as np

X = np.zeros((100, 1)) # X contient que des 0

a = 12

for i in range(0, 100):
    X[i] = np.sqrt(rd.random() * 4 * a**2)
Tn = 3/(4*100) * np.sum(X)

print(Tn)
```

□

Solution de l'exercice 9.

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

Y = rd.exponential(1, (1, 50)) # Y = [Y_1, Y_2, ...., Y_50]
X = Y + np.ones((1, 50)) # X = [Y_1 + 1, Y_2 + 1,...., Y_50 + 1]
S = np.sum(X)/50

print(S)
```

□