Exercices - Fiche 1

* * *

Exercice 1. *EMLyon, 2022.* On effectue une série de n lancers indépendants d'une pièce de monnaie telle qu'on obtient pile avec la probabilité $p \in]0, 1[$. On introduit la variable aléatoire T correspondant au nombre de lancers effectués lorsque, pour la première fois, on a obtenu autant de fois pile que face, et qui vaut 0 si cet événement ne se réalise pas.

1. Écrire une fonction PYTHON `simule(p,n)` qui prend en argument le réel p et le nombre de lancers n , et qui renvoie la valeur de T .
2. Qu'affiche le script PYTHON suivant ? Comment interpréter les différents résultats ?

```
for i in range(1,4):
    L=np.zeros(3)
    for j in range(3):
        m=0
        for k in range(1000):
            m=m+simule(i/4,10**4)
        L[j]=m/1000
    print(L)
```

```
[1.466 1.462 1.394]
[ 62.332  59.506 107.704]
[1.572 1.506 1.444]
```

```
1. def simule(p,n):
    nb_pile=0; nb_face=0; k=1
    if rd.rand()<p:
        nb_pile+=1
    else:
        nb_face+=1
    while nb_pile != nb_face and k<n:
        if rd.rand()<p:
            nb_pile+=1
        else:
            nb_face+=1
        k+=1
    if nb_pile!=nb_face:
        return 0
    else:
        return k
```

2. Le script affiche trois simulations de l'espérance de T , dans chacun des cas : $p = \frac{1}{4}$, $p = \frac{1}{2}$ et $p = \frac{3}{4}$. Chaque ligne correspond à une des valeurs de p .

Pour $p = \frac{1}{4}$ et $p = \frac{3}{4}$, les estimations de l'espérance sont similaires. Pour $p = \frac{1}{2}$, ce n'est en revanche pas le cas. Comme l'estimation de $\mathbb{E}(T)$ est faite à partir de 1000 simulations de T , on s'attend pourtant à trouver des valeurs similaires dans le cas où $\mathbb{E}(T)$ existe. Ceci suggère que T n'a pas d'espérance dans le cas où $p = \frac{1}{2}$, par opposition aux deux autres cas.

Exercice 2. *ESSEC/HEC, Sujet 0 2023.* Pour $n \in \mathbb{N}^*$, on note :

- ★ U_n le vecteur colonne de $\mathbb{R}^n$ dont toutes les composantes sont égales à 1,
 - ★ $H_n = I_n - \frac{1}{n} U_n {}^t U_n$, où I_n désigne la matrice identité de taille n .
1. On considère l'application $\Phi : \mathcal{M}_n(\mathbb{R}) \rightarrow \mathcal{M}_n(\mathbb{R})$ telle que $\Phi(A) = -\frac{1}{2} H_n A H_n$ pour toute matrice $A \in \mathcal{M}_n(\mathbb{R})$. Écrire une fonction PYTHON `Phi(A)` qui retourne $\Phi(A)$ lorsque le tableau numpy `A` représente la matrice A .
 2. On importe la bibliothèque `linalg` de numpy en exécutant `import numpy.linalg as al` et on saisit une matrice `A`. Donner une expression comportant la fonction `al.eig` qui vaut `True` si $\text{Sp}(\Phi(A)) \subset \mathbb{R}_+$, et `False` sinon.

```

1. import numpy.linalg as al
   import numpy as np
   def Phi(A):
       n=len(A)
       U=np.ones((n,1))
       H=np.eye(n)-1/n*np.dot(U,np.transpose(U))
       return -1/2*np.dot(H,np.dot(A,H))

2. L=al.eig(A)
   min(L[0])>=0

```

Exercice 3. *Edhec 2022.* Écrire une fonction PYTHON `k(A)` qui prend en argument un tableau numpy `A` représentant une matrice carrée, et qui retourne le réel $\max\{|\lambda|, \lambda \in \text{Sp}(A)\}$.

```

import numpy as np
def k(A):
    return max(abs(al.eig(A)[0]))

```

Exercice 4. Soit $p \in]0, 1[$. On considère une suite (X_n) de variables aléatoires indépendantes suivant toutes la loi $\mathcal{B}(p)$.

On considère par ailleurs une variable aléatoire N telle que $N - 1$ suit la loi de Poisson de paramètre $\lambda > 0$ fixé, et on introduit la variable aléatoire S donnée par $S = \sum_{i=1}^N X_i$.

Écrire une fonction PYTHON `S(lam,p)` qui prend en argument les valeurs `lam` et `p` respectives de λ et p , et qui retourne une simulation de S .

```

import numpy.random as rd
def S(lam,p):
    N=rd.poisson(lam)+1
    return np.sum(rd.binomial(1,p,N))

```

Exercice 5. Pour tout $n \in \mathbb{N}$, on pose $I_n = \int_0^1 \frac{x^n}{(1+x)^2} dx$, et $J_n = \int_0^1 \frac{x^n}{1+x} dx$.

1. Montrer que $I_0 = \frac{1}{2}$ et $I_1 = \ln 2 - \frac{1}{2}$.

2. Montrer que pour tout $n \in \mathbb{N}$, $I_{n+2} + 2I_{n+1} + I_n = \frac{1}{n+1}$.
3. Compléter la fonction PYTHON suivante I(n) pour qu'elle retourne la valeur de I_n en fonction d'un entier n en entrée.

```
import numpy as np
def I(n):
    a=1/2; b=np.log(2)-1/2
    for k in range(..., ...):
        aux=a
        a=...
        b=...
    if n==0:
        return(...)
    else:
        return(...)
```

4. Établir que pour tout $n \in \mathbb{N}^*$, $I_n = nJ_{n-1} - \frac{1}{2}$.
5. Montrer que pour tout $n \in \mathbb{N}$, $J_n + J_{n+1} = \frac{1}{n+1}$.
6. Compléter la fonction PYTHON suivante I2(n) pour qu'elle retourne la valeur de I_n à partir de l'entrée n .

```
import numpy as np
def I2(n):
    J=np.log(2)
    for k in range(..., ...):
        J=...
    if n==0:
        return 1/2
    else:
        return ...
```

```
3. import numpy as np
def I(n):
    a=1/2; b=np.log(2)-1/2
    for k in range(2,n+1):
        aux=a
        a=b
        b=1/(k-1)-2*b-aux
    if n==0:
        return(a)
    else:
        return(b)
```

```
6. import numpy as np
def I2(n):
    J=np.log(2)
    for k in range(1,n):
        J=1/k-J
    if n==0:
        return 1/2
    else:
        return n*J-1/2
```

Exercice 6. Compléter le script PYTHON ci-dessous, qui calcule et affiche une valeur approchée de l'intégrale $\int_0^1 \ln(1+x^2) dx$ avec la méthode des rectangles.

```
import numpy as np
n=...
S=0
for k in range(..., ...):
    S=...
print("Approximation de l'intégrale :", ...)
```

On utilise l'approximation par la méthode des rectangles à droite, qui est aussi la somme de Riemann : on sait que

$$\frac{1}{n} \sum_{k=1}^n \ln \left(1 + \left(\frac{k}{n} \right)^2 \right) \xrightarrow{n \rightarrow +\infty} \int_0^1 \ln(1+x^2) dx.$$

```
import numpy as np
n=1000
S=0
for k in range(1,n+1):
    S=S+np.log(1+(k/n)**2)
print("Approximation de l'intégrale :", S/n)
```

Exercice 7. Pour tout n entier tel que $n \geq 3$, on considère la fonction $f_n : x \mapsto x^n + x - n$.

1. Montrer que pour tout $n \geq 3$, l'équation $f_n(x) = 0$ admet une unique solution sur $]1, +\infty[$. On notera cette solution v_n .
2. On admet que pour tout $n \geq 3$, on a $1 \leq v_n \leq 2$. L'objectif de cette question est d'approcher la valeur de v_n par dichotomie à $\varepsilon = 10^{-2}$ fixé près pour $n = 10$. Recopier et compléter le programme ci-dessous.

```
import numpy as np
def f(x,n):
    return(...)
n=10 ;eps=...
a=...;b=...
while b-a ... :
    c=...
    if f(c,n) ... :
        ...
    else:
        ...
print("Valeur approchée de v_n :", ...)
```

2. On sait que $f_n(1) < 0$, $f_n(2) > 0$ et la fonction f_n est croissante. On obtient donc la dichotomie suivante pour trouver v_n .

```

import numpy as np
def f(x,n):
    return(x**n-x-n)
n=10 ; eps=10**(-2)
a=1;b=2
while b-a > eps:
    c=(a+b)/2
    if f(c,n)>0 :
        b=c
    else:
        a=c
print("Valeur approchée de v_n :",c)

```

Exercice 8. On appelle indice de nilpotence d'une matrice nilpotente M le plus petit entier p tel que la matrice M^p est nulle. Compléter la fonction `indnilp` suivante, qui permet de calculer l'indice de nilpotence d'une matrice nilpotente donnée en entrée.

```

import numpy as np
def indnilp(M):
    p=0
    A=M
    while ...:
        p=...
        A=...
    return p

```

On remarque qu'on ne peut pas utiliser la commande `A!=0` car `A` est un tableau (`array`), et `0` est un entier (`int`).

```

import numpy as np
def indnilp(M):
    p=0
    A=M
    while np.sum(abs(A))>0:
        p=p+1
        A=np.dot(M,A)
    return p

```

Exercice 9. Soit X une variable aléatoire suivant la loi binomiale de paramètres n et p . On note X^* la variable aléatoire centrée réduite associée à X .

1. Compléter la fonction `SimulBin` ci-dessous, qui prend en entrée les paramètres n et p et un réel x pour qu'elle renvoie une estimation de $\mathbb{P}(X^* \leq x)$.

```

import numpy.random as rd
def SimulBin(n,p,x):
    N=10**5
    B=rd.binomial(n,p,N)
    C=... # version centrée réduite de B
    S=0

```

```

for i in range(N):
    if ...:
        S=...
return...

```

2. Soit Z une variable suivant la loi normale centrée réduite. La fonction `SimulNorm` suivante, qui prend en entrée un réel x , estime la probabilité $\mathbb{P}(Y \leq x)$.

```

import scipy.special as sp
def SimulNorm(x):
    return sp.ndtr(x)

```

Compléter le script suivant pour qu'il compare les estimations des fonctions de répartition de X^* et de Z .

```

import matplotlib.pyplot as plt
n=...
p=...
N=50
X = np.linspace(-2,2,N)
Y = np.zeros(N)
Z = np.zeros(N)
for i in range(N):
    Y[i]=...
    Z[i]=...
plt.plot...
plt.plot...
plt.show()

```

À quels résultats s'attend-on suivant les valeurs de n et de p ?

```

1. import numpy as np
def SimulBin(n,p,x):
    N=10**5
    B=rd.binomial(n,p,N)
    C=(B-n*p)/(np.sqrt(n*p*(1-p)))
    S=0
    for i in range(N):
        if C[i]<=x:
            S=S+1
    return(S/N)

```

2. On sait que si $X_n \leftrightarrow \mathcal{B}(n, p)$ et $X_n^* = \frac{X_n - np}{\sqrt{np(1-p)}}$, alors (X_n^*) converge en loi vers une variable aléatoire Z qui suit la loi $\mathcal{N}(0, 1)$, d'après le théorème central limite. Ainsi, on s'attend à ce que les fonctions de répartition de X_n^* et de Z soient très proches si n est grand.

Par ailleurs, lorsque p est proche de 0 ou de 1, l'approximation par la loi normale est moins bonne, et on lui préfère l'approximation par la loi de Poisson. On s'attend à ce que dans ce cas, l'écart entre les courbes soit plus important.

```
import matplotlib.pyplot as plt
n=100
p=0.3
N=50
X = np.linspace(-2,2,N)
Y = np.zeros(N)
Z = np.zeros(N)
for i in range(N):
    Y[i]=SimulNorm(X[i])
    Z[i]=SimulBin(n,p,X[i])
plt.plot(X,Y, label='Norm')
plt.plot(X,Z, label='Bin')
plt.show()
```

Exercice 10. On admet que $\int_0^1 \sqrt{1-x^2} dx = \frac{\pi}{4}$.

1. Soient X une v.a.r. qui suit la loi $\mathcal{U}([0,1])$ et Y la v.a.r. définie par $Y = \sqrt{1-X^2}$. Calculer $\mathbb{E}(Y)$ et $\mathbb{V}(Y)$, qu'on notera v .
2. Expliquer pourquoi la fonction PYTHON suivante permet d'obtenir une approximation de $\frac{\pi}{4}$.

```
import numpy as np
import numpy.random as rd
def approx(n):
    X=rd.rand(n)
    return(np.mean(np.sqrt(1-X**2)))
```

3. À l'aide du théorème central limite, montrer qu'en choisissant $n \geq t_\alpha^2 v 10^4$, où t_α est l'unique réel tel que $\Phi(t_\alpha) = 1 - \frac{\alpha}{2}$, `approx(n)` fournit une approximation de $\frac{\pi}{4}$ à 10^{-2} près avec un risque inférieur à α . Dans la suite, on choisira $\alpha = 0.05$, donc $t_\alpha \simeq 1,96$.
4. Compléter le programme suivant avec le bon choix de valeur de n pour qu'il renvoie une estimation de la probabilité que l'écart entre `approx(n)` et $\frac{\pi}{4}$ soit supérieur à 10^{-2} .

```
v=...
n=...
N=10000 # nombre de simulations
A=np.zeros(N)
for i in range(N):
    A[i]=...
S=0
for i in range(N):
    if abs(A[i]-np.pi/4)>10**(-2):
        S=S+1
print("Estimation de la probabilité :", ...)
```

1. Comme $Y^2 = 1 - X^2$, la v.a.r. Y admet un moment d'ordre 2. Par ailleurs, par le théorème de transfert, on a $\mathbb{E}(Y) = \int_0^1 \sqrt{1-x^2} dx = \frac{\pi}{4}$. Par ailleurs,

$$\mathbb{V}(Y) = \mathbb{E}(Y^2) - \mathbb{E}(Y)^2 = 1 - \mathbb{E}(X^2) - \mathbb{E}(X)^2 = 1 - \frac{1}{3} - \frac{\pi^2}{16} = \frac{2}{3} - \frac{\pi^2}{16}.$$

2. Il s'agit d'une approximation de $\int_0^1 \sqrt{1-x^2} dx$ par la méthode de Monte-Carlo : on considère une suite (X_n) de v.a.r. indépendantes de loi $\mathcal{U}([0, 1])$, on sait alors d'après la loi des grands nombres que

$$\frac{1}{n} \sum_{i=1}^n X_i \xrightarrow[n \rightarrow +\infty]{\mathbb{P}} \mathbb{E}(X) = \int_0^1 \sqrt{1-x^2} dx.$$

```
4. v=2/3-np.pi**2/16
n=int(1.96**2*v*10**4)+1
N=10000 # nombre de simulations
A=np.zeros(N)
for i in range(N):
    A[i]=approx(n)
S=0

for i in range(N):
    if abs(A[i]-np.pi/4)>10**(-2):
        S=S+1
print("Estimation de la probabilité :", S/N)
```