1 Lois usuelles

1.1 Simulation

Exercice 1

a) Une souris se déplace le long d'un corridor infini. À chaque instant, elle a deux chances sur trois d'aller vers la droite, et une chance sur trois d'aller vers la gauche, de façon indépendante.

Compléter le programme python suivant pour qu'il simule cette expérience et renvoie la position de la souris après n déplacements.

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

n=int(___)
P=___
print("La souris se trouve à la position",P,".")
```

b) On effectue l'expérience suivante : On lance une pièce de monnaie équilibrée jusqu'à tomber sur pile. À chaque fois que l'on tombe sur face, on dépose une boule rouge dans une urne, et une boule verte lors de l'obtenition du premier pile. On effectue alors un tirage dans cette urne.

Compléter le programme python suivant pour qu'il simule cette expérience.

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

L=___
print("On obtient pile au bout de",L,"lancers.")

C=___
if C==1:
    print("La boule tirée est verte.")
else:
    print("La boule tirée est rouge.")
```

c) L'expérience est à présent la suivante : On lance un dé équilibré à six faces, puis une pièce de monnaie équilibrée autant de fois que le score indiqué sur le dé. La partie est gagnée si on a une majorité de pile.

Compléter le programme python suivant pour qu'il simule cette expérience.

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

D=1+___
P=___
if ___:
    print("La partie est gagnée,")
else:
    print("La partie est perdue,")
print("avec",P,"pile sur",D,"lancers.")
```

d) À une agence, le nombre d'arrivée en une heure suit une loi de Poisson de paramètre 15 clients/h. Ces clients ont le choix entre 2 guichets, qu'ils choisissent uniformément et indépendamment.

Compléter le programme python suivant pour qu'il simule cette expérience.

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

N=___
A=___
B=___
print(A,"clients se sont rendus au guichet A, contre",B,"au guichet B,")
print("pour",N,"clients au total.")
```

1.2 Représentation des lois

On rappelle que la commande `hist(Data,classes)` de la librairie `matplotlib.pyplot` permet de tracer l'histogramme correspondant anx données `Data` regroupées dans des `classes`, décrites par une liste de séparateurs.

Exercice 2
Compléter, pour chacune des lois suivantes, le programme `python` ci-dessous, pour qu'il trace l'histogramme de la loi correspondante, de manière harmonieuse.

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

N=___
Data=rd.___(___,size=N)
debut=___
fin=___
classes=np.arange(debut,fin+2)-.5

plt.clf()
plt.hist(Data,classes,edgecolor='black',density=True)
```

a) $\mathscr{U}(\llbracket 1, 10 \rrbracket)$
b) $\mathscr{B}(\frac{1}{2})$
c) $\mathscr{B}(\frac{1}{3})$
d) $\mathscr{B}(5, \frac{1}{2})$
e) $\mathscr{B}(100, \frac{1}{3})$
f) $\mathscr{G}(\frac{1}{10})$
g) $\mathscr{G}(.9)$
h) $\mathscr{G}(\frac{1}{2})$
i) $\mathscr{P}(1)$
j) $\mathscr{P}(2)$
k) $\mathscr{P}(10)$
l) $\mathscr{B}(100, \frac{1}{10})$

1.3 Bonus : marche aléatoire

Exercice 3
Souvenez-vous de la souris qui va à droite ou a à gauche ? On propose le programme suivant.

```
1  import numpy as np
2  import numpy.random as rd
3  import matplotlib.pyplot as plt
4
5  S=10000     #nombre de souris
6  Sshow=10    #nombre de trajectoires à tracer
7  N=100       #nombre de pas
8  p=1/2       #probabilité d'aller à droite
9
10 X=np.zeros([N,S])
11 for t in range(1,N):
12     X[t]=X[t-1]+2*rd.binomial(1,p,size=S)-1
13
14 T=range(N)
15 B=N*(2*p-1)
16 R=5*np.sqrt(N*p*(1-p))
17 C=np.arange(B-R,B+R+2)-.5
18
19 plt.clf()
20 fig, (histo, walk) = plt.subplots(2, sharex=True)
21 histo.hist(X[N-1],C)
22 walk.plot(X[:,range(Sshow)],T)
```

a) Quel est l'effet de la ligne 12 ? Que contient donc le tableau X à la fin de l'exécution du programme ?
b) Quel est l'effet de la ligne 21 ? D'après les lignes 14-17, quelle est l'amplitude du déplacement escompté ? Le démontrer.
c) Quel est l'effet de la ligne 22 ?
d) Tester ce programme.

2 Exercices Divers

2.1 Tirages dans une urne

Exercice 4
On lance n fois une pièce de monnaie dont la probabilité de tomber sur pile est de $p = \frac{1}{3}$. On note X le nombr ede fois où on est tombé sur "pile" lors de cette série de lancer. On relance alors X fois la pièce, et on note Y le nombre de fois où on est tombé sur face lors de cette seconde série de lancers.

a) i. Donner la loi de X, et la loi de Y conditionnellement à X.

ii. Compléter le programme suivant pour qu'il permette de simuler l'expérience décrite :

```
def expe(n,p):
    X=___
    Y=___
    return [X,Y]
```

iii. Quel est l'effet des fonctions suivantes ?

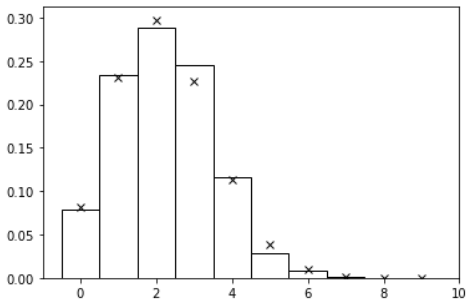
```
def fact(n):
    y=1
    for i in range(1,n+1):
        y=y*i
    return y
def binom(n,k):
    return fact(n)/(fact(k)*fact(n-k))
```

```
def loibinom(n,p):
    P=np.zeros(n)
    for i in range(n):
        P[i]=binom(n,i)*p**i*(1-p)**(n-i)
    return P
```

iv. Les deux programmes précédents étant exécutés dans python, on exécute les lignes suivantes :

```
n=10
p=1/3
q=1-p

N=1000
Data=np.array([expe(n,p)])
for i in range(N-1):
    Data=np.append(Data,[expe(n,p)],axis=0)

plt.clf()
X=np.arange(n)
classes=np.arange(n+1)-.5
P=loibinom(n,p*q)
plt.plot(X,P,'x',color="black")
plt.hist(Data[:,1],classes,density=True,color="white",edgecolor="black")
```

et on obtient le graphique suivant : Quelle hypothèse peut-on faire sur la loi de Y ?



b) i. Montrer que pour tout $0 \leqslant i \leqslant j \leqslant n$,

$$\binom{n}{j}\binom{j}{i} = \binom{n}{j}\binom{n-i}{j-i}$$

ii. Démontrer l'hypothèse formulée à la question aiv.

2.2 L'arbre aux corbeaux

Exercice 5

Le grand chêne du village est connu pour être le lieu de rendez-vous des corvidés du coin. On suppose que chaque jour, le nombre de corbeaux se perchant sur l'arbre centenaire suit une loi de Poisson, avec une moyenne de 25 corbeaux. L'ornithologue local estime que 56% des corbeaux sont des femelles (les autres étant bien évidemment des mâles, la société corvidée n'étant pas très progressiste sur les questions de genre).

a) **Étude mathématique**

On convient de noter C le nombre de corbeaux sur l'arbre un jour donné, F le nombre de femelles, et M le nombre de mâles.

i. Rappeler la formule donnant la loi de C.

ii. En moyenne, combien y aura-t-il de corbeaux sur l'arbre ? Avec quel écart-type ?

iii. Reconnaître la loi de F et M conditionnellement à $[C = n]$, pour $n \in \mathbb{N}$.

iv. En déduire que F et M suivent tous les deux une loi de Poisson, dont on déterminera les paramètres.

v. Le nombre de corbeaux mâles est-il indépendant du nombre de corbeaux femelles ?

b) **Simulation informatique**

i. Complétez le programme suivant pour qu'il calcule les probabilités $\mathbb{P}(X = k)$ d'une loi de Poisson de paramètre `L` donné, pour $k \in [\![0, s]\!]$ (s étant aussi donné) (on suppose les librairies nécessaires chargées).

```
def fact(n):
    f=1
    for i in range(1,n+1):
        f=f*i
    return f
```

```
def Poisson(L,s):
    P=np.zeros(s+1)
    for k in range(s+1):
        P[k]=___
    return P
```

ii. Complétez le programme suivant pour qu'il simule, pendant `n` jours, le nombre de corbeaux sur l'arbre (sous la forme d'un tableau à trois lignes, la première correspondant au nombre total de corbeaux, la seconde au nombre de femelles, et la dernière au nombre de mâles).

```
def corbeaux(n):
    T=np.zeros([3,n])
    for i in range(n):
        T[0,i]=___
        T[1,i]=___
        T[2,i]=___
    return T
```

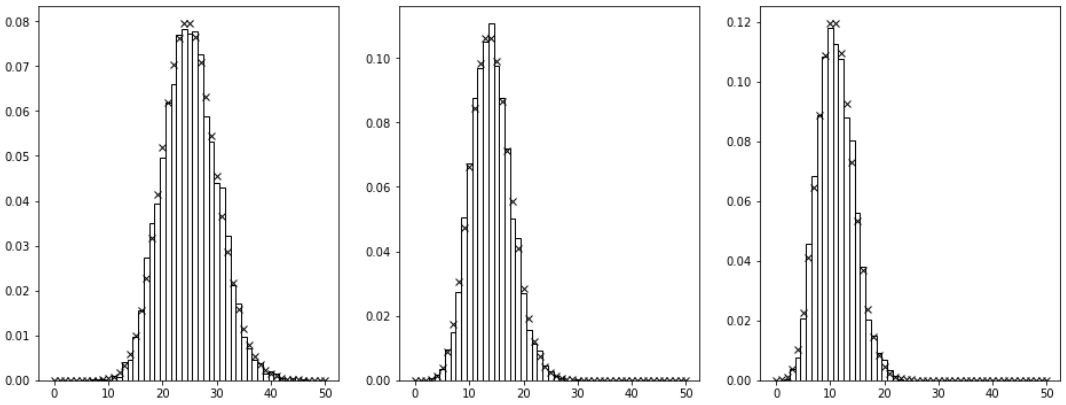
iii. On lance à présent le programme suivant :

```
1  Cmoy=25
2  f=56/100
3  m=1-f
4  plt.clf()
5  plt.figure(figsize=(16, 6))
6  n=10*365
7  s=int(Cmoy+5*np.sqrt(Cmoy))
8  X=np.arange(s+1)
9  P=[Poisson(Cmoy,s),Poisson(Cmoy*f,s),Poisson(Cmoy*m,s)]
10 classes=np.arange(s+1)-.5
11 Data=corbeaux(n)
12 for i in range(3):
13     plt.subplot(1,3,i+1)
14     plt.plot(X,P[i],'x')
15     plt.hist(Data[i],classes,density=True)
```

(la commande `subplot` permet simplement de séparer la fenêtre graphique en trois).

Voici le contenu de la fenêtre graphique obtenue.

a. Quel est le contenu des trois lignes du tableau `P` défini ligne 9 ?



b. Quel semble être le rôle de a variable `s` ? Comment expliquer la formule utilisée à la ligne 7 ?

c. Quel est le contenu de `Data` défini ligne 11 ?

d. Que représentent les barres du graphique, tracéees à la ligne 15 ?

e. Que représentent les croix du graphique, tracéees à la ligne 14 ?

f. En quoi le graphique confirme-t-il l'étude mathématique effectuée précédemment ?

2.3 Un exercice en plus

Exercice 6

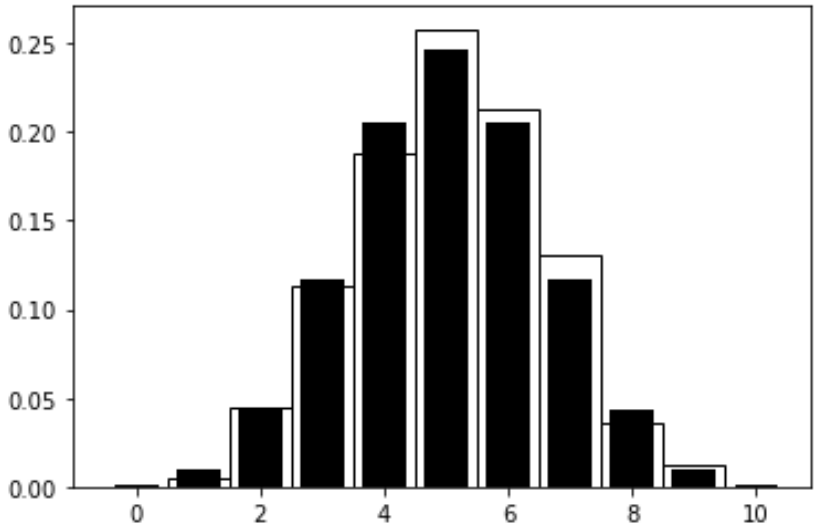
a) Compléter les fonctions suivantes pour que la dernière, `binom(n,k)`, calcule le coefficient binomial $\binom{n}{k}$.

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

def fact(n):
    f=1
    for i in range(1,n+1):
        f=___
    return f

def binom(n,k):
    B=fact(___)/(fact(___)*fact(___))
    return B
```

b) On lance alors le programme suivant :

```
N=10
p=1/2

Pb=np.zeros([2,N+1])
for k in range(N+1):
    Pb[0,k]=k
    Pb[1,k]=binom(N,k)*p**k*(1-p)**(N-k)

plt.clf()

Nsim=1000
Data=rd.binomial(N,p,size=Nsim)
S=np.arange(N+1)-.5
plt.hist(Data,S,density=True,color="white",edgecolor="black")

plt.bar(Pb[0],Pb[1],color='black',width=.7)
```

Voici une capture d'écran de la fenêtre graphique après exécution du programme :



Comment expliquez-vous ce graphique ? Qu'est-ce qui est tracé en barres vides, et qu'est-ce qui est tracé en barres pleines ? Comment expliquer que les barres semblent de hauteurs similaires, mais pas identiques ?

c) Quelle valeur changer si l'on veut une meilleure correspondance entre les deux histogrammes ?

d) (bonus) n'hésitez pas à expérimenter en changeant les valeurs de `N`, `p` et `Nsim`.

3 Pour aller plus loin : constructions des lois usuelles

3.1 L'outil de base : la fonction `random`

La fonction `random()` du module `numpy.random` renvoit un réel R, pris « uniformément au hasard » * entre 0 et 1 ($R \in [0,1[$ pour être précis). En pratique, cela signifie que, pour tout $0 \leqslant a \leqslant b < 1$,

$$\mathbb{P}(R \in [a,b]) = b - a. \tag{1}$$

```
R=rd.random()
```

3.2 Loi de Bernoulli

On peut aisément simuler une variable B suivant une loi de Bernoulli de paramètre $p \in]0,1[$ (qui donc répond 0 ou 1 avec probabilité p) en simulant un R uniforme comme précédemment, et en définissant :

$$B = \begin{cases} 1 & \text{si } R \leqslant p, \\ 0 & \text{sinon.} \end{cases}$$

```
def bernoulli(p):
    R=rd.random()
    if R<=p:
        B=1
    else:
        B=0
    return B
```

Exercice 7

a) Justifier que, si R vérifie la propriété (1), la variable B ainsi définie a bien pour support $\{0,1\}$ et pour loi $\mathbb{P}(B=1) = p$ (la loi de Bernoulli).

b) Vérifier que la fonction précédente simule bien une variable aléatoire B suivant une loi de Bernoulli de paramètre p.

c) S'inspirer de la fonction précédente pour en construire une, nommée `piece`, qui réponde « pile » ou « face » au lieu de 0 ou 1.

Cette première fonction permet déjà de simuler des expériences simples : par exemple, le programme suivant permet de simuler le tirage d'une boule dans une de deux urnes, l'une contenant deux boules noires et une blanche, l'autre, une noire et deux blanches :

```
T=bernoulli(1/2)   #sélection de l'une des deux urnes
if T==1:
   C=bernoulli(1/3)#tirage dans la première urne
else:
   C=bernoulli(2/3)#tirage dans la seconde urne
if C==1:
   disp("blanc!")  #traduction de la couleur
else:
   disp("noir!")
end
```

Exercice 8

a) Simuler l'expérience suivante dans `python` : Devant nous se trouvent deux urnes, la permière contient une boule rouge, et deux noires, la seconde, deux boules rouges et trois noires. On commence par lancer une pièce de monnaie. Si on obtient pile, on pioche une boule de la première urne, sinon, une boule de la seconde. Le programme renverra la couleur de la boule tirée.

b) Sachant que l'on a pioché une boule rouge, quelle est la probabilité que l'on aie obtenu pile ? Retrouver ce résultat avec la simulation `python`, en utilisant le fait que, sur un nombre grand d'épreuves, la moyenne des résultats correspond à l'espérance.

∗. En réalité, il s'agit d'un rationnel, pour des raisons évidentes de limitations de mémoire. Il n'est d'ailleurs pas choisi purement au hasard, puisqu'il dépend d'un paramètre fixé, la graine... mais à notre niveau, cela suffit bien.

3.3 Loi binomiale

La loi binomiale (n, p) correspond au nombre de succès rencontrés si l'on fait n expériences successives de Bernoulli (p). Si vous lancez 4 dés à 6 faces, le nombre de 1 obtenu suit une loi binomiale $(4, \frac{1}{6})$.

On peut aisément, à partir d'une loi de Bernoulli, simuler une loi binomiale, en en simulant plusieurs à la fois, de la façon suivante† :

```
def binomiale(n,p):
    T=np.zeros(n)
    for i in range(n):
        T[i]=bernoulli(p)
    B=int(sum(T))
    return B
```

Exercice 9

a) Vérifier que la fonction précédente simule bien une variable aléatoire X suivant une loi binomiale de paramètres (n, p).

b) Implémenter en `python` un programme simulant l'expérience suivante : à une fête foraine, vous avez trois balles pour renverser la pile de gobelets en face de vous. Hélas, la tâche est ardue, et chaque lancer n'a qu'une chance sur quatre de renverser la pile. Vous gagnez donc si au moins un de vos lancers touche la cible.

c) La pile est à présent bien plus difficile à renverser, et il vous faut au moins trois lancers réussis, mais vous avez droit à 5 lancers.

3.4 Loi géométrique

La loi géométrique (p) correspond au nombre d'expériences de Bernoulli qu'il faut réaliser pour obtenir un succès : si vous lancez une pièce de monnaie jusqu'à tomber sur "pile", le nombre de lancers effectués suit une loi géométrique $(\frac{1}{2})$.

```
def geom(p):
    G=1
    while bernoulli(p)<1:
        G=G+1
    return G
```

Exercice 10

a) Vérifier que la fonction précédente simule bien une variable aléatoire X suivant une loi géométrique de paramètre p.

b) Vous commencez avec une cagnotte de 1000€. On vous propose de lancer une pièce de monnaie, jusqu'à ce que vous obteniez un pile, et repartiez avec la cagnotte. Cependant, le montant de la cagnotte est divisé par deux à chaque fois que vous obtenez face.
Écrivez un programme qui simule cette expérience.

c) Vous vous trouvez face à une urne contenant un billet de lotterie gagnant. Vous lancez une pièce de monnaie jusqu'à obtenir pile, et piochez alors un billet dans l'urne. Cependant, le présentateur ajoute un billet perdant à l'urne à chaque fois que vous obtenez une face (ainsi, si vous avez obtenu pile au $n$ième lancé, vous tirez dans une urne contenant 1 billet gagnant et $n-1$ billets perdants).
Écrivez un programme qui simule cette expérience.

3.5 Lancer de dé

La loi uniforme correspond au résultats d'une expérience où tous les résultats ont la même probabilité d'apparaître. Si vous lancez un dé à 6 faces, le résultat obtenu suit une loi uniforme sur $[\![1, 6]\!]$.

```
def des(F):
    D=int(1+np.floor(np.random()*F))
    return D
```

Exercice 11

a) Comprendre pourquoi cette fonction simule bien le résultat du lancer d'un dé équilibré à F faces.

b) Soirée jeux de rôles, vous avez oublié vos dés ! Écrire un programme `python` permettant à l'ogre de lancer ses 2d8 (somme de deux dés à huit faces) de dégâts.

†. On rappelle que, comme son nom l'indique, la commande `sum` fait la somme des termes entre parenthèses.

3.6 Simulation de lois quelconques

Exercice 12

On veut simuler une variable aléatoire X de fonction de répartition connue, F_X. On suppose ici que F_X est une bijection de $\mathbb{R}$ dans $[0,1]$.

a) Montrer que la variable aléatoire $U = F_X(X)$ suit une loi uniforme continue sur $[0,1]$ (c'est à dire que que sa fonction de répartition est $x \mapsto x$ sur $[0,1]$).

b) En déduire que si U est une variable aléatoire uniforme continue sur $[0,1]$, $F_X^{-1}(U)$ suit la même loi que X (on montrera que sa fonction de répartition est bien F_X).

c) en utilisant la commande `rd.random()`, qui simule une loi uniferme continue, compléter le programme suivant, pour qu'il simule une variable aléatoire dont la fonction de répartition est donnée par :

$$F_X(x) = \frac{1}{1+\mathrm{e}^{-x}}$$

(On vérifiera bien sûr qu'il s'agit bien d'une fonction de répartition.)

```
def Finv(u):
    return (___)
U=rd.random()
X=___(U)
print(X)
```

Remarque

L'hypothèse de bijectivité de F_X n'est en réalité pas nécessaire, car le raisonnement marche tout autant avec la réciproque généralisée *:*

$$\begin{array}{rccl} F_X^{-1}: & [0,1] & \longrightarrow & \mathbb{R} \\ & u & \longmapsto & \max\{x \in \mathbb{R},\ F_X(x) \leqslant u\} \end{array}$$

A Commandes

A.1 Simulations de lois

Syntaxe (librairie random)

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

Syntaxe (lois usuelles)

```
rd.random()                    #uniforme sur [0,1[
rd.randint(n)             #uniforme entière sur [[0,n-1]]
rd.binomial(n,p)    #binomiale
rd.geometric(p)           #géométrique
rd.poisson(l)             #Poisson
```

N.B. : il est possible d'ajouter un argument `size=(...)` pour simuler plusieurs variables en même temps.

A.2 Représentations graphiques

Syntaxe (graphes : librairie `matplotlib.pyplot`*)*

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

Ne pas oublier de "nettoyer" la fenêtre graphique au moyen de `plt.clf()`.

Selon les versions de python, il peut être nécessaire de demander explicitement d'afficher la fenêtre graphique par

```
plt.show()
```

Syntaxe (diagrammes en barre)

```
plt.bar(modalités,effectifs)
```

Syntaxe (histogramme)

```
plt.hist(data,n)    #classes automatiques (nombre de classes)
plt.hist(data,c)    #classes manuelles (liste des "bords")
```

Syntaxe (fréquences cumulées)

```
plt.plot(range(1,data.shape[0]+1),data.sort())
```

Syntaxe (boîte à moustache)

```
plt.boxplot(data)
```