DS 10

THÈME : INFORMATIQUE

La présentation, la lisibilité, l'orthographe, la qualité de la rédaction, la clarté et la précision des raisonnements entreront pour une part importante dans l'appréciation des copies. Les candidats sont invités à encadrer dans la mesure du possible les résultats de leurs calculs. Ils ne doivent faire usage d'aucun document : l'utilisation de toute calculatrice et de tout matériel électronique est interdite.

Exercice I

Les nombres métalliques

La suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ est définie par $F_0 = 0$, $F_1 = 1$, et la relation de récurrence

$$F_n = F_{n-1} + F_{n-2} \quad \text{pour} \quad n \geqslant 2.$$

- *Le nombre d'or*

1. a) Écrire un programme `Fibo` qui prend en argument n et renvoie la matrice ligne :

$$\begin{bmatrix} F_0 & F_1 & F_2 & \dots & F_{n-1} & F_n \end{bmatrix} \in \mathcal{M}_{1,n+1}(\mathbb{R}).$$

b) Utiliser la fonction précédente pour créer une seconde fonction `Fiboq` qui prend en argument n et renvoie

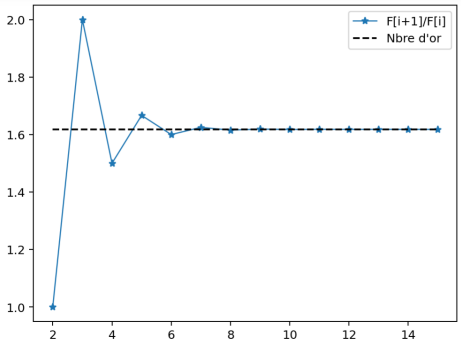
$$\begin{bmatrix} F_2/F_1 & F_3/F_2 & F_4/F_3 & \dots & F_{n-1}/F_{n-2} & F_n/F_{n-1} \end{bmatrix} \in \mathcal{M}_{1,n-1}(\mathbb{R}).$$

En reprenant les résultats sur les suites récurrentes linéaires d'ordre 2, on montre que

$$\frac{F_{n+1}}{F_n} \xrightarrow[n\to\infty]{} \varphi = \frac{1+\sqrt{5}}{2} \quad \text{(nombre d'or)}.$$

Editeur

```
# Illustration de la convergence vers le nombre d'or
Q=Fiboq(15)
A=np.arange(2,16)
plt.plot(A,Q,'-*',label="F[i+1]/F[i]")
p=(1+np.sqrt(5))/2*np.ones(14)
plt.plot(A,p,'k--',label="Nbre d'or")
plt.legend()
plt.show()
```



- *Les nombres métalliques*

Pour $p \in \mathbb{N} \setminus \{0;1\}$, on définit maintenant la suite $\left(F_n^p\right)_{n\in\mathbb{N}}$ de p-bonacci par $F_0^p = \dots = F_{p-2}^p = 0$, $F_{p-1}^p = 1$ et la relation de récurrence

$$F_n^p = F_{n-1}^p + F_{n-2}^p + \dots + F_{n-p}^p \quad \text{pour} \quad n \geqslant p.$$

On admet que la suite $(F_{n+1}^p/F_n^p)_n$ est convergente et on note :

$$\frac{F_{n+1}^p}{F_n^p} \xrightarrow[n\to\infty]{} \varphi_p \quad \text{(le } p\text{-ième nombre de métal)}.$$

L'objectif de la suite est d'obtenir une approximation de φ_p. On propose deux méthodes.

2. *Approximation via un calcul matriciel*
À p fixé, on définit maintenant la matrice colonne

$$U_n^p = \begin{bmatrix} F_n^p \\ F_{n+1}^p \\ \vdots \\ F_{n+p-1}^p \end{bmatrix} \in \mathcal{M}_{p,1}(\mathbb{R}).$$

a) Déterminer une matrice $A_p \in \mathcal{M}_p(\mathbb{R})$, ne dépendant que de p telle que pour tout $n \in \mathbb{N}$, $U_{n+1}^p = A_p U_n^p$.
b) Écrire un programme `matrice` qui prend en argument p et renvoie la matrice A_p. En déduire un programme qui prend en arguments p et n puis en renvoie la matrice ligne U_n^p.
c) En déduire un programme python qui prend en argument p et donne une approximation de φ_p.
On pourra prendre $n = 100$.

3. *Par dichotomie*
On admet que φ_p est l'unique solution x plus grande que 1 strictement telle que

$$x^p - x^{p-1} - \cdots - 1 = 0.$$

a) Écrire une fonction `fm` qui prend en arguments un réel x, un entier p et renvoie la somme $x^p - x^{p-1} - \cdots - 1$.
b) Compléter cet algorithme de dichotomie, qui prend en argument p et donne une approximation de φ_p à 10^{-3}.

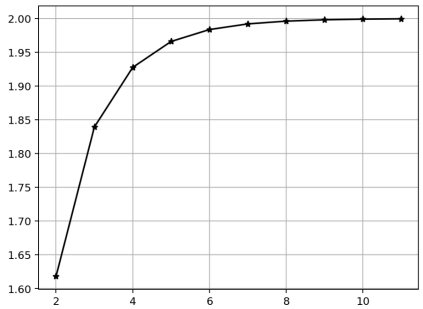
Editeur

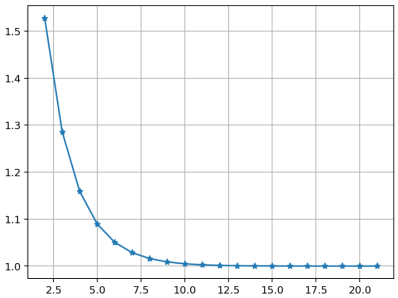
```
def dicho(p):
    a= ....
    b=10
    erreur=b-a
    while erreur> .... :
        c=(a+b)/2
        if fm(a,p)*fm(c,p)>0:
            a=  ....
        else :
            b= ....
        erreur=  ....
    return a
```

c) Que peut-on conjecturer sur le comportement de la suite $(\varphi_p)_{p\geqslant 2}$ à l'aide des deux codes suivants ?

Editeur

```
n=10
E=np.zeros(n)
for i in range(n):
    E[i]=dicho(i)
Ab=np.arange(2,n+2)
plt.plot(Ab,E,'*-')
plt.grid()
plt.show()
```





Editeur

```
n=20
G=np.zeros(n)
for i in range(n):
    G[i]=(2-dicho(i))*2**(i+2)
Ab=np.arange(2,n+2)
plt.plot(Ab,G,'*-')
plt.grid()
plt.show()
```

Exercice II

Séries lacunaires

Pour tous $n \in \mathbb{N}^*$ et $\mathscr{D}$ une partie de $\mathbb{N}^*$, on pose

$$h_n(\mathscr{D}) = \sum_{k=1}^{n} \frac{\mathbf{1}_{\mathscr{D}}(k)}{k} \quad \text{où} \quad \mathbf{1}_{\mathscr{D}}(k) = \begin{cases} 1 & \text{si } k \in \mathscr{D} \\ 0 & \text{sinon.} \end{cases}$$

On pose aussi $\mathscr{D}_1 = \{2k-1 \mid k \in \mathbb{N}^*\}$.

1. **a)** Écrire une fonction `indiD1` qui prend en argument $k \in \mathbb{N}^*$ et renvoie $\mathbf{1}_{\mathscr{D}_1}(k)$.
b) Donner la partie $\mathscr{D}_2$ caractérisée par son indicatrice $\mathbf{1}_{\mathscr{D}_2}$ codée par :

Editeur

```
def indiD2(k):
    r=round(np.sqrt(k)) # round(x) renvoie l'entier le plus proche de x
    if r*r==k:
        return 1
    return 0
```

2. Écrire une fonction `sommeh` qui prend en arguments n et une fonction `indiD` codant une partie $\mathscr{D}$ et renvoie $h_n(\mathscr{D})$.
3. Que permet de conjecturer le code suivant?

Editeur

```
N=7
a=np.pi**2/6
R=np.zeros(N-1)
for p in range(1,N):
    R[p-1]=sommeh(10**p,indiD2)/a
print(R)

[0.82745633 0.94214581 0.98070235 0.99395102 0.99807922 0.99939238]
```

4. On admet qu'il existe $\ell \in \mathbb{R}$ tel que :

$$\forall n \in \mathbb{N}\setminus\{0;1;2\}, \qquad -\frac{1}{n} \leqslant \ell - \left(h_n(\mathscr{D}_1) - \ln(\sqrt{n})\right) \leqslant \frac{1}{n-2}.$$

Proposer un programme qui donne une approximation de ℓ à 10^{-3}-près.
5. Que fait le calcul suivant où $\alpha \in \mathbb{N}^*$?

Editeur

```
def mystere(n,alpha):
    k=1
    p=1
    s=0
    while k<=n:
        s+= 1/k
        p=p+1
        k=p**alpha
    return s
```

Exercice III
Marches aléatoires

On dit qu'une variable aléatoire X suit une loi de Rademacher de paramètre $p \in]0;1[$, noté $X \hookrightarrow \mathscr{R}(p)$ si

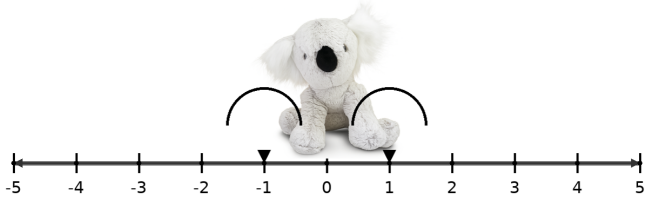
$$\begin{cases} X(\Omega) = \{-1,1\} \\ \mathbf{P}(X=1) = p \quad \text{et} \quad \mathbf{P}(X=-1) = 1-p \end{cases}$$

De plus, on dit qu'une variable aléatoire Z suit une loi de Rayleigh de paramètre σ si Z est à densité et une densité est donnée sur par :

$$f(r) = \frac{r}{\sigma^2} e^{-r^2/(2\sigma^2)}, \quad r \geqslant 0 \quad \text{et} \quad f(r) = 0 \quad \text{sinon.}$$

1. **a)** Écrire un programme qui prend en argument p et simule une variable $X \hookrightarrow \mathscr{R}(p)$.
   b) Inversement, si on ne dispose que d'une fonction `mystere` qui simule une loi de Rademacher dont le paramètre p nous est inconnu, proposer un programme permettant d'avoir une approximation de p.
2. **a)** Soient Z, une variable aléatoire qui suit une loi de Rayleigh de paramètre σ et H la restriction de la fonction de répartition de Z à $\mathbb{R}^+$. Justifier que H est une bijection et donner la loi de $H^{-1}(U)$ où $U \hookrightarrow \mathscr{U}(]0;1[)$.
   b) En déduire un programme qui simule Z.

A. Marches sur une droite



Mascotte se déplace sur une droite graduée. On suppose qu'à chaque seconde, Mascotte se déplace d'une unité sur la droite avec probabilité p, et d'une unité sur la gauche avec probabilité $1-p$. On suppose que les déplacements sont indépendants les uns des autres.

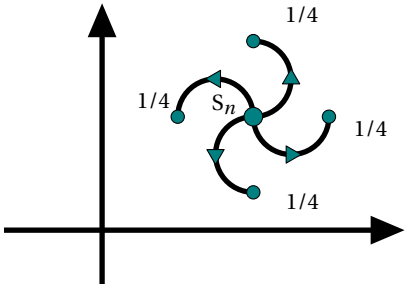
3. Soient $a, b \in \mathbb{Z}$ avec $a < 0 < b$. Écrire un programme qui prend en arguments a, b et p puis renvoie T la durée (en seconde) où pour la première fois, Mascotte quitte l'intervalle $[a;b]$.

B. Marches dans le plan

On se place maintenant dans le cas où Mascotte se déplace dans le plan gradué.

À chaque seconde, Mascotte se déplace d'une unité soit vers la gauche, la droite, en haut ou en bas de manière équiprobable. Ainsi, si on considère pour tout $i \in \mathbb{N}^*$

$$X_i = \begin{cases} (1,0) & \text{avec probabilité } 1/4 \\ (-1,0) & \text{avec probabilité } 1/4 \\ (0,1) & \text{avec probabilité } 1/4 \\ (0,-1) & \text{avec probabilité } 1/4 \end{cases}$$

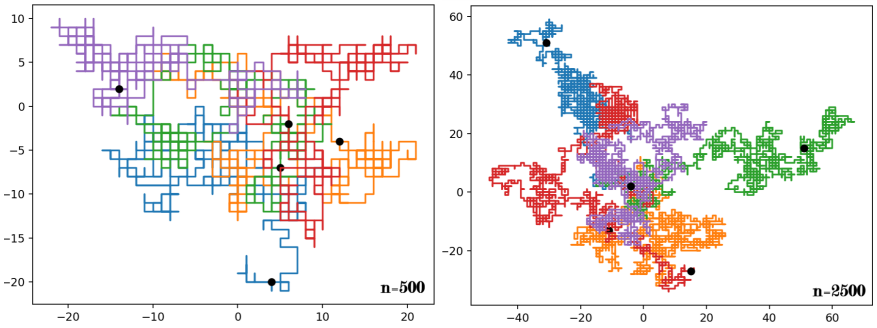


alors le vecteur aléatoire $S_n = \sum_{i=1}^{n} X_i$ donne la position de Mascotte à l'instant $n \in \mathbb{N}^*$. On suppose les variables X_i indépendantes.

4. **a)** Simuler X_i (on pourra s'aider de la loi de Rademacher et renvoyer X_i sous forme d'un tableau `np.array()`).
   b) En déduire une fonction avec un argument qui simule le vecteur aléatoire S_n.
5. On identifie $\mathbb{R}^2$ et $\mathscr{M}_{1,2}(\mathbb{R})$. En déduire une fonction avec un argument qui simule la matrice aléatoire

$$\begin{bmatrix} S_0 & S_1 & S_2 & \dots & S_n \end{bmatrix} \in \mathscr{M}_{2,n+1}(\mathbb{R}).$$

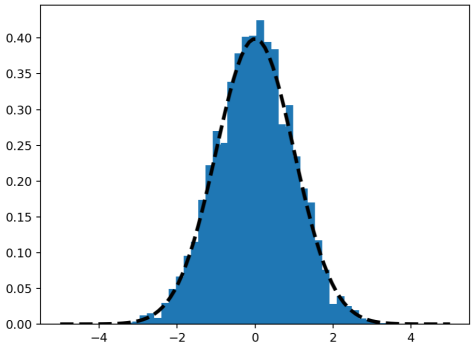
Ci-après, quelques trajectoires.



6. Quel théorème est illustré par le code suivant ? (justifier précisément).

Editeur

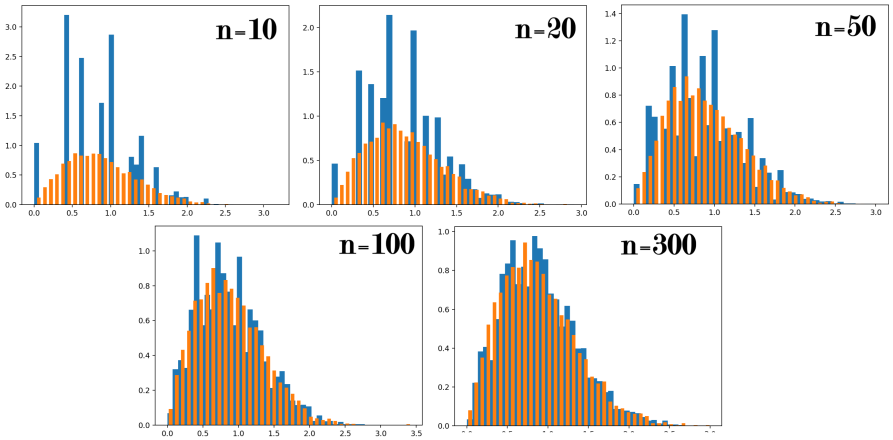
```
n=100, p=1/2, m=5000
Ech=np.zeros(m)
for i in range(m):
    s = simuS(p,n)
    Ech[i] = (s[0] + s[1]) / np.sqrt(n)
plt.hist(Ech,25,density=True)
x=np.linspace(-5,5,100)
y=(2*np.pi)**(-1/2)*np.exp(-x**2/2)
plt.plot(x,y)
plt.show()
```



On pourra dans un premier temps, regarder la loi de Y_i*, la variable aléatoire qui somme les deux coordonnées de* X_i*.*

7. Soit D_n, la distance entre la mascotte et l'origine. Comment simuler D_n ?

8. Commenter le code et les résultats suivants.



Editeur

```
n= ...
m=5000
E=np.zeros(m)
F=np.zeros(m)
for i in range(m):
    E[i]=simuD(n)/np.sqrt(n)     #la fonction simuD(n) simule Dn
    F[i]=np.sqrt(-np.log(rd.random()))
plt.hist(E,40,density=True)
plt.hist(F,40,density=True,rwidth=0.6)
plt.show()
```

– FIN –

DS 10 - solution

Exercice I

Les nombres métalliques

1.a)

```
def Fibo(n):
    F=np.zeros(n+1)
    # initialisation
    # On a déjà F[0]=0
    F[1]=1
    for i in range(2,n+1):
        F[i]=F[i-1]+F[i-2]
    return F

print(Fibo(5))
[0. 1. 1. 2. 3. 5.]
```

1.b)

```
def Fiboq(n):
    F=Fibo(n)
    Q=np.zeros(n-1)
    for i in range(n-1):
        Q[i]=F[i+2]/F[i+1]
    return Q

print(Fiboq(10))

[1.          2.          1.5
    1.66666667  1.6         1.625
 1.61538462  1.61904762  1.61764706]
```

2.a) Vérifier que

$$A_p = \begin{bmatrix} 0 & 1 & 0 & \cdots & \cdots & 0 \\ 0 & 0 & 1 & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \cdots & 0 & 1 \\ 1 & 1 & \cdots & \cdots & \cdots & 1 \end{bmatrix}.$$

2.b)

```
def matrice(p):
    A=np.zeros([p,p])
    for i in range(p-1):
        A[i,i+1]=1
        A[p-1,i]=1
    A[p-1,p-1]=1
    return A
```

- On teste :

```
>>> matrice(5)
array([[0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.],
       [1., 1., 1., 1., 1.]])
```

- On en déduit un programme qui prend en argument n et p et renvoie U_n^p.

```
def suiteU(n,p):
    U=np.zeros([p,1])
    U[p-1]=1
    A=matrice(p)
    for i in range(n):
        U=np.dot(A,U)
    return U
```

2.c) Pour n « grand », on peut approximer φ par F_{n+1}^p/F_n^p.

```
def approx(p):
    n=100
    U=suiteU(n,p)
    return U[p-1]/U[p-2]
```

- Tests :

```
>>> approx(2)
array([1.61803399])

>>> approx(3)
array([1.83928676])

>>> approx(4)
array([1.92756198])
```

Il existe des formules explicites pour ces trois valeurs. Par exemple

$$\varphi_2 = \frac{1+\sqrt{5}}{2}$$

$$\varphi_3 = \frac{\sqrt[3]{19+3\sqrt{33}} + \sqrt[3]{19-3\sqrt{33}} + 1}{3}.$$

3.a)

```
def fm(x,p):
    s=x**p
    for i in range(p):
        s=s-x**i
    return s

## Solution 2 pour minimiser le nombre de
    calcul des puissances

def fm2(x,p):
    s=1
    puiss=1
    for i in range(1,p):
        puiss=x*puiss
        s=s+puiss
    return x*puiss-s
```

3.b)

```
def dicho(p):
    a= 1
    b=10
    erreur=b-a
    while erreur> 10**(-3):
        c=(a+b)/2
        if fm(a,p)*fm(c,p)>0:
```

```
            a=c
        else :
            b=c
        erreur=erreur / 2
## erreur =b-a convient aussi
    return c
```

On enchaîne avec un petit test :

```
print(approx(2),approx(3),approx(4))
print(dicho(2),dicho(3),dicho(4))
```

```
[1.61803399] [1.83928676] [1.92756198]
1.61798095703125 1.83880615234375
    1.92724609375
```

3.c) Le premier code permet de conjecturer que la suite est croissante, convergente et de limite 2. Le second code suggère plutôt

$$(2-\varphi_p)2^p \underset{p\to+\infty}{\longrightarrow} 1.$$

Ce qui peut être traduit par l'équivalent :

$$2-\varphi_p \sim 2^{-p}.$$

Exercice II

Séries lacunaires

Cet exercice est inspiré du sujet maths II 2018, partie II.

1.a) Deux propositions :

```
def indiD1(k):
    n=k+1
    if 2*np.floor(n/2)==n:
        return 1
    return 0

def indiD1bis(k):
    n=k+1
    if n%2==0:
        return 1
    return 0
```

et un petit test :

```
for k in range(1,10):
    print(k,indiD1(k),indiD1bis(k))
```

```
1 1 1
2 0 0
3 1 1
4 0 0
5 1 1
6 0 0
7 1 1
8 0 0
9 1 1
```

1.b) Le code test si l'entier est un carré d'un entier en testant si le carré de l'entier le plus proche de sa racine carrée est bien le nombre. On a donc

$$\mathscr{D}_2 = \{n^2 \mid n \in \mathbb{N}^*\}.$$

Un petit test :

```
for k in range(1,10):
    print(k,indiD2(k))
```

```
1 1
2 0
3 0
4 1
5 0
6 0
7 0
8 0
9 1
```

2.

```
def sommeh(n,indiD):
    s=0
    for k in range(1,n+1):
        s=s+indiD(k)/k
    return s
```

Attention de bien commencer par $k=1$ et non par $k=0$ avec seulement `range(n+1` car sinon on commence par diviser par 0...

3. Le code permet de conjecture que

$$\frac{h_n(\mathscr{D}_2)}{a} \underset{n\to\infty}{\longrightarrow} 1.$$

Limite que l'on peut retraduire par :

$$\sum_{k=1}^{n} \frac{1}{k^2} \underset{n\to\infty}{\longrightarrow} a = \frac{\pi^2}{6}.$$

4. On peut retraduire la propriété par

$$\left|\ell - \left(h_n(\mathscr{D}_1) - \ln(\sqrt{n})\right)\right| \leqslant \frac{1}{n-2}$$

Ainsi pour le choix $n = 2+10^3$, on a

$$\left|\ell - \left(h_n(\mathscr{D}_1) - \ln(\sqrt{n})\right)\right| \leqslant 10^{-3}.$$

Le code s'en déduit :

```
n=10**(3)+2
lapprox=sommeh(n,indiD1)-np.log(n)/2

# test
print(lapprox)
0.6351815057316696
```

5. Le code calcule $h_n(\mathscr{D}_\alpha)$ où

$$\mathscr{D}_\alpha = \{n^\alpha \mid n \in \mathbb{N}^*\}.$$

Un petit test :

```
n=5000
print(mystere(n,2),np.pi**2/6)
print(mystere(n,4),np.pi**4/90)
print(mystere(n,6),np.pi**6/945)
```

```
1.630749907490019 1.6449340668482264
1.0817841677034807 1.082323233711138
1.0172408827374828 1.017343061984449
```

Test basé sur le fait que :

$$\sum_{k=1}^{+\infty} \frac{1}{k^2} = \frac{\pi^2}{6}, \qquad \sum_{k=1}^{+\infty} \frac{1}{k^4} = \frac{\pi^4}{90}$$

$$\text{et} \quad \sum_{k=1}^{+\infty} \frac{1}{k^6} = \frac{\pi^6}{945}.$$

Exercice III
Marches aléatoires

1.a)

```
def rad(p):
    x=rd.random()
    if x<p:
        return 1
    return -1

def radbis(p):
    return 2*(rd.random()<p)-1

def radbibis(p):
    return (-1)**(rd.random()>p)
```

1.b) D'après la loi faible des grands nombres (X a une variance), on sait que la moyenne empirique d'un échantillon "relativement important" est proche de l'espérance de X. Or

$$\mathbf{E}(X) = 2p - 1.$$

En inversant la relation, on a

$$p = \frac{\mathbf{E}(X)+1}{2}.$$

Le code s'en déduit :

```
    s=0
for i in range(10000):
    s+=mystere()
print((s/10000+1)/2)
```

Un petit test :

```
def mystere():
    return  rad(0.3) # pour le test

s=0
for i in range(10000):
    s+=mystere()
print((s/10000+1)/2)
```

```
0.29200000000000004
```

2.a) Voir la méthode d'inversion qui permet de montrer que

$$Z \quad \text{et} \quad \sigma\sqrt{-2\ln(1-U)} \quad \text{avec} \quad U \hookrightarrow \mathscr{U}([0;1[)$$

ont même loi. Comme U et $1-U$ ont même loi, on peut même simplifier par

$$\sigma\sqrt{-2\ln U}.$$

2.b)

```
def Rayleigh(sigma):
    u = rd.random()
    return sigma*np.sqrt(-2*np.log(u))
```

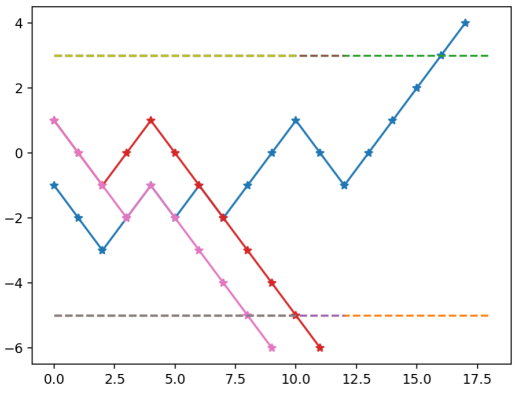
3.

```
def bords(a,b,p):
    n=1
    s=0
    while s>=a and s<=b:
        n+=1
        s+=rad(p)
    return n
```

- Une seconde possibilité pour visualiser :

```
def bordsbis(a,b,p):
    n=1
    s=0
    L=[]
    while s>=a and s<=b:
        n+=1
        s+=rad(p)
        L.append(s)
    plt.clf()
    plt.plot(L,'-*')
    plt.plot(a*np.ones(n),'--')
    plt.plot(b*np.ones(n),'--')
    plt.show()
    return n
```

On affiche alors trois trajectoires :



4.a)

```
def simuX():
    p=1/2
    if rad(p)==1:
        x=rad(p)
        return np.array([x,0])
    else :
        y=rad(p)
        return np.array([0,y])
```

4.b)

```
def simuS(n):
    s=np.array([0,0])
    for i in range(n):
        s+=simuX(p)
    return s
```

5.

```
def simuS2(p,n):
    S=np.zeros([2,n+1])
    s=np.array([0,0])
    for i in range(1,n+1):
        s+=simuX(p)
        S[0,i]=s[0]
        S[1,i]=s[1]
    return S
```

Le code pour afficher les trajectoires :

```
for i in range(5):
    S=simuS2(1/2,2500)
    plt.plot(S[0,:],S[1,:])
    plt.plot(S[0,-1],S[1,-1],'ko')
plt.show()
```

6. C'est le théorème central limite. En effet, on vérifie que pour tout indice i

$$Y_i \hookrightarrow \mathscr{R}(1/2)$$

et les variables sont mutuellement indépendantes. La variable simulée par

```
Ech[i] = (s[0] + s[1]) / np.sqrt(n)
```

est la variable

$$\frac{C_n}{\sqrt{n}} \quad \text{où} \quad C_n = \sum_{k=1}^{n} Y_k.$$

Notons que

$$\mathbf{E}(C_n) = \sum_{k=1}^{n} \mathbf{E}(Y_k) = 0$$

et par indépendance des variables

$$\mathbf{V}(C_n) = \sum_{k=1}^{n} \mathbf{V}(Y_k) = \sum_{k=1}^{n} 1 = n.$$

Dès lors, $C_n/\sqrt{n}$ est centrée et réduite. On sait alors que

$$\frac{C_n}{\sqrt{n}} \underset{n\to+\infty}{\xrightarrow{\mathscr{L}}} Z \quad \text{où} \quad Z \hookrightarrow \mathcal{N}(0;1).$$

Pour n grand, on s'attend avec une grande probabilité que l'histogramme d'un échantillon construit à partir de $C_n/\sqrt{n}$ épouse la courbe d'une densité de la loi normale centrée réduite. Par exemple, celle donnée par le code :

$$t \in \mathbb{R} \mapsto \frac{1}{\sqrt{2\pi}} \exp\left(-x^2/2\right).$$

7.

```
def simuD(n):
    p=1/2
    Point=simuS(p,n)
    return np.sqrt(Point[0]**2+Point
        [1]**2)
```

8. On a ici une convergence en loi de $(D_n/\sqrt{n})$ vers Z, qui suit une loi de Rayleigh de paramètre $1/\sqrt{2}$.