**Nom :**

# Mathématiques approfondies
# Cours ECG 2

## Python - partie II

**Thèmes**

1. Algèbre bilinéaire
2. Compléments sur les variables aléatoires à densité
3. Convergence des variables aléatoires
4. Estimations
5. Compléments sur les fonctions de deux variables
6. Exercices


**Lycée Saint Louis 2024/2025**

# Python - semestre II

`NotesCB=rd.randint(0,20,48)`

C. FISZKA , *10 février 2025*

## 1 Algèbre bilinéaire

**Exercice 1.** ✦✦ **Méthode de la puissance itérée**

Dans la suite, A désigne une matrice de $\mathcal{M}_n(\mathbb{R})$ non nulle et diagonalisable. On pose $\lambda$, la plus grande valeur propre de A en valeur absolue.

Le but de cet exercice est de déterminer une approximation d'un vecteur propre de A associé à la valeur propre $\lambda$. Pour cela, on définit la suite de vecteurs $(X_k)_{k\in\mathbb{N}}$ par

$$X_0 \in \mathcal{M}_{n,1}(\mathbb{R}) \quad \text{et} \quad \forall\, k \in \mathbb{N}, \quad X_{k+1} = \frac{AX_k}{\|AX_k\|}$$

où $\|\cdot\|$ désigne la norme associée au produit scalaire canonique sur $\mathcal{M}_{n,1}(\mathbb{R})$. On admet que la suite $(X_k)_{k\in\mathbb{N}}$ converge vers un vecteur propre associé à $\lambda$, c'est-à-dire que, pour tout indice $i \in [\![1;n]\!]$, la suite dont le terme général est la $i$-ème coordonnée de $X_k$ converge.

1. Écrire une fonction `norme` qui prend en argument une matrice colonne X et renvoie sa norme $\|X\|$.
2. Écrire une fonction `puissance(A, X0, k)` qui renvoie le terme $X_k$.
3. *Test.* On pose
$$A = \begin{bmatrix} 0.5 & 0.5 \\ 0.2 & 0.8 \end{bmatrix} \quad \text{et} \quad X_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$
Calculer (sans Python) les valeurs propres de A et donner un vecteur propre de norme 1 associé à la plus grande valeur propre.
💻 Comparer les résultats obtenus avec les résultats obtenus avec la fonction `puissance`.
4. 🔑 Modifier le programme précédent pour construire une nouvelle fonction qui prend en arguments A, $X_0$, $\lambda$ et affiche la plus petite valeur $k$ telle que $\|AX_k - \lambda X_k\| \leqslant 10^{-6}$.
5. 🔑 On suppose maintenant la matrice A inversible. Comment obtenir une approximation d'un vecteur propre d'une matrice associé à $\mu$, la plus petite valeur propre de A en valeur absolue ?

**Exercice 2.** ✦✦ **Construction d'une famille orthonormée avec deux vecteurs**

1. Soient $u, v \in \mathbb{R}^n$ non colinéaires. Comment construire deux vecteurs orthogonaux $e_1, e_2 \in \mathbb{R}^n$ tels que
$$\|e_1\| = 1, \quad \|e_2\| = 1 \quad \text{et} \quad \text{Vect}(u,v) = \text{Vect}(e_1, e_2) \text{ ?}$$
2. **a)** Écrire une fonction `Scal` qui prend en arguments deux vecteurs $u$, $v$ de $\mathbb{R}^n$ et renvoie $\langle u, v\rangle$ où $\langle\cdot,\cdot\rangle$ correspond au produit scalaire canonique de $\mathbb{R}^n$.
*On identifiera un vecteur de $\mathbb{R}^n$ avec une matrice de $\mathcal{M}_{n,1}(\mathbb{R})$.*
**b)** En déduire un programme qui prend en arguments deux vecteurs de $\mathbb{R}^n$ puis teste si les deux vecteurs sont non colinéaires et, dans le cas où ils ne sont pas colinéaires, renvoie une base orthonormée du sous-espace vectoriel engendré par ces deux vecteurs.

**Exercice 3.** ✦✦✦ **Mouvement brownien sur une sphère**

Dans la suite, on identifie tout vecteur de $\mathbb{R}^3$ avec sa matrice de $\mathcal{M}_{3,1}(\mathbb{R})$. On munit $\mathbb{R}^3$ de son produit scalaire canonique :

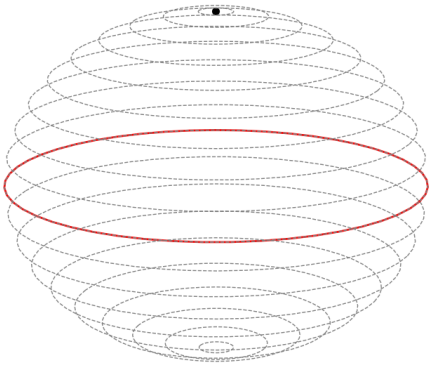
$$\forall\, x, y \in \mathbb{R}^3, \qquad \langle x, y\rangle = {}^t x y.$$

On note $\|\cdot\|$, la norme associée.

La sphère unité de $\mathbb{R}^3$ est définie par

$$\mathscr{S} = \{x \in \mathbb{R}^3 \mid \|x\| = 1\}.$$

Le pôle nord de la sphère est alors le point de coordonnées $x_P = (0;0;1)$.
Le code ci-contre permet de représenter la sphère unité et le pôle nord.



```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

axes = plt.axes(projection="3d")
# Le pôle nord
axes.plot([0],[0],[1],'.',color='black',
    linewidth=5)
# L'équateur
t = np.linspace(0,2*np.pi, 50)
axes.plot(np.cos(t),np.sin(t),np.zeros(50)
    ,color='red',linewidth=1)
# Les latitudes
r=np.linspace(0,2*np.pi,20)
for i in range(20):
    x = np.cos(r[i])*np.cos(t)
    y = np.cos(r[i])*np.sin(t)
    z= np.sin(r[i])*np.ones(50)
    axes.plot(x, y, z, '--',color='grey',
        linewidth=0.5)
plt.axis('off')
plt.show()
```

**1.** Les trois fonctions suivantes permettent de construire 3 matrices, lesquelles ? On les notera $R_1(\theta)$, $R_2(\theta)$ et $R_3(\theta)$.

```
def Rotation1(theta):
    M=np.eye(3)
    M[1,1]=np.cos(theta)
    M[2,2]=M[1,1]
    M[2,1]=np.sin(theta)
    M[1,2]=-M[2,1]
    return M
```

```
def Rotation2(theta):
    M=np.eye(3)
    M[0,0]=np.cos(theta)
    M[1,1]=M[0,0]
    M[1,0]=np.sin(theta)
    M[0,1]=-M[1,0]
    return M
```

```
def Rotation3(theta):
    M=np.eye(3)
    M[0,0]=np.cos(theta)
    M[2,2]=M[0,0]
    M[2,0]=np.sin(theta)
    M[0,2]=-M[2,0]
    return M
```

**2.** Vérifier que les trois matrices obtenues sont orthogonales. En déduire que pour tout $x \in \mathbb{R}$, $\|R_i x\| = \|x\|$.
*Les matrices* $R_1(\theta)$, $R_2(\theta)$ *et* $R_3(\theta)$ *correspondent à des matrices de rotation d'angle* $\theta$ *respectivement dans les plans* $(Oyz)$, $(Oxy)$ *et* $(Oxz)$.

Soit $h \in \mathbb{R}_*^+$. Soient les suites de variables aléatoires $(\Theta_i)_{i\in\mathbb{N}}$ de loi uniforme $\mathscr{U}([0;h])$ et $(N_i)_{i\in\mathbb{N}}$ de loi uniforme discrète sur $\{1;2;3\}$ définies sur un même espace probabilisé $(\Omega, \mathscr{A}, \mathbf{P})$. On suppose les variables $(\Theta_i, N_i)_{i\in\mathbb{N}}$ mutuellement indépendantes. On définit alors :

- La suite de matrice aléatoire $(M_i)_{i\in\mathbb{N}}$ par :

$$\forall\, i \in \mathbb{N}, \qquad M_i = R_{N_i}(\Theta_i).$$

- La suite aléatoire $(X_i)_{i\in\mathbb{N}}$ de vecteurs de $\mathbb{R}^3$ par

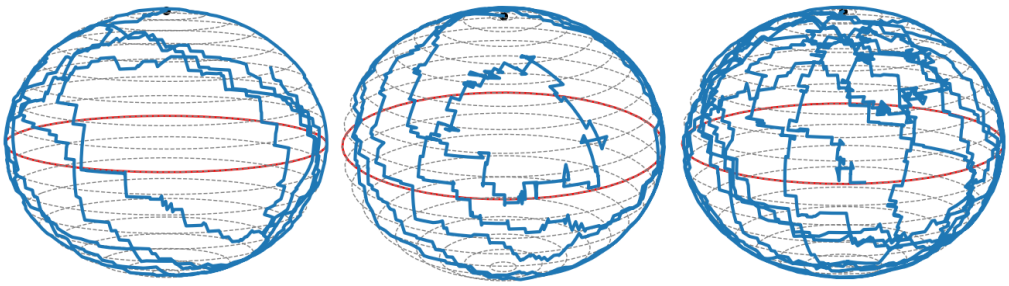
$$X_0 = x_P \quad \text{et} \quad \forall\, i \in \mathbb{N}, \qquad X_{i+1} = M_i X_i.$$

**3.** Justifier que pour tout $i \in \mathbb{N}$, tout $\omega \in \Omega$, $X_i(\omega) \in \mathscr{S}$.

**4.** Écrire une fonction python qui prend en argument $h \in \mathbb{R}_*^+$ et renvoie une réalisation de $M_1$.

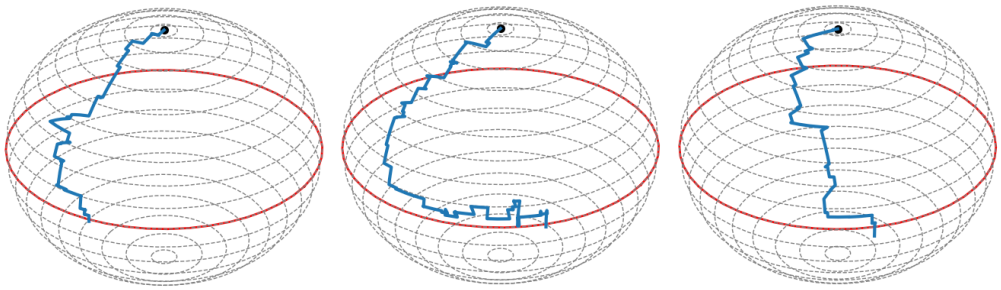
**5.** En déduire un programme qui prend en argument $n$ et renvoie une réalisation de $X_n$.
*On pourra choisir* $h = 0.1$.

**6.** Compléter le programme précédent pour afficher les $n+1$ premières réalisations de $X_0, X_1, \ldots, X_n$ pour obtenir une "trajectoire".
*Si* `x`, `y`, `z` *représentent 3 matrices lignes de même taille correspondent à des abscisses, des ordonnées et des hauteurs, alors la commande* `axes.plot(x, y, z)` *permet de représenter la trajectoire.*
*Ci-après, quelques exemples :*

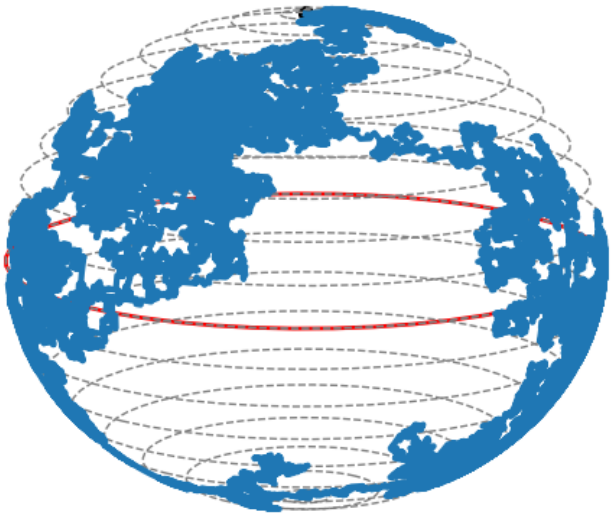


*Si* `x`, `y`, `z` *représentent 3 matrices lignes de même taille correspondent à des abscisses, des ordonnées et des hauteurs, alors la commande* `axes.plot(x, y, z)` *permet de représenter la trajectoire.*

**7.** Soit T, la variable aléatoire égale au rang pour lequel on traverse l'équateur pour la première fois. Simuler T et donner une estimation de son espérance.




**8.** Reprendre les questions précédentes où $U_i$ suit une loi uniforme sur $[-h;h]$. *Ci-dessous un exemple de trajectoire.*



# 2 Compléments sur les variables aléatoires à densité

## 2.1 Nouvelles méthodes de simulation

**Exercice 4.** ✦ **Simulation de la loi du $\chi^2$**

Soient $n \in \mathbb{N}^*$ et $(X_i)_{i\in[\![1;n]\!]}$ un vecteur aléatoire de variables aléatoires mutuellement indépendantes et de même loi $\mathcal{N}(0;1)$. On dit que la somme $S_n = {X_1}^2 + \ldots + {X_n}^2$ suit une loi du $\chi^2$ à $n$ degrés de liberté.

**1.** Écrire un programme qui prend en argument $n$ et simule la variable $S_n$.
*On pourra utiliser dans la bibliothèque* `numpy`, *la commande* `np.random.normal(mu,v,m)` *pour simuler m réalisation d'une variable aléatoire suivant une loi normale* $\mathcal{N}(\mu, v)$.

2. En déduire un second programme qui prend en argument $n$ et $m \in \mathbb{N}^*$ et renvoie un échantillon de taille $m$ de $S_n$. Comment tracer les histogrammes ?
3. 💻 Parmi ces trois densités, une seule correspond à la loi du $\chi^2$ de degré 4, laquelle ?

$$f : x \in \mathbb{R} \mapsto \begin{cases} 0 & \text{si } x < 0 \\ \frac{1}{4} x e^{-\frac{x}{2}} & \text{si } x \geqslant 0, \end{cases} \qquad g : x \in \mathbb{R} \mapsto \frac{e^x}{(e^x+1)^2} \quad \text{et} \quad h : x \in \mathbb{R} \mapsto \begin{cases} 0 & \text{si } x < 0 \\ 4x e^{-2x} & \text{si } x \geqslant 0. \end{cases}$$

**Exercice 5.** ✦ **Simulation de la loi $\gamma(n)$**

1. Soit X une variable aléatoire suivant une loi uniforme sur $]0;1[$. Donner la loi de $Y = -\ln(X)$.
2. Soit $n \in \mathbb{N}^*$. En déduire un programme Python pour simuler une variable aléatoire de loi $\gamma(n)$.
3. Tester votre programme en affichant les histogrammes et en comparant avec une densité de $\gamma(n)$.

**Exercice 6.** ✦ **Simulation de la loi de Laplace**

Une variable aléatoire à densité X suit la loi de Laplace si une densité $f$ est définie sur $\mathbb{R}$ par

$$f(x) = \frac{1}{2} e^{-|x|}.$$

1. Soit $U \hookrightarrow \mathscr{U}(]0;1])$. Donner la loi de $X = -\ln(U)$.
2. Justifier que si X et Y sont deux variables aléatoires définies sur un même espace probabilisé qui suivent une loi exponentielle alors la différence $X - Y$ suit une loi de Laplace.
3. En déduire un programme Python qui simule une loi de Laplace.
4. Tester votre programme en comparant les histogrammes obtenus avec la densité.

**Exercice 7.** ✦✦✦ **Simulation des lois normales par la méthode de Box-Müller**

- *La théorie*

  Soient X et Y deux variables aléatoires définies par

$$\begin{cases} X = \sqrt{W}\cos(\Theta) \\ Y = \sqrt{W}\sin(\Theta) \end{cases} \quad \text{où} \quad W \hookrightarrow \mathscr{E}(1/2) \quad \text{et} \quad \Theta \hookrightarrow \mathscr{U}([0;2\pi])$$

  avec W et $\Theta$ indépendantes. On admet que X et Y sont indépendantes et de même loi $\mathscr{N}(0;1)$.

1. Soient U et V sont deux variables aléatoires indépendantes suivant la loi uniforme sur $]0;1]$. Justifier que les deux variables définies par

$$\sqrt{-2\ln U}\cos(2\pi V) \quad \text{et} \quad \sqrt{-2\ln U}\sin(2\pi V)$$

   sont indépendantes et suivent une loi normale centrée réduite.

- *La pratique*

2. En déduire une fonction Python `NormaleCR` qui simule une loi normale centrée réduite.
3. Comment, à partir de `NormaleCR`, obtenir une simulation de la loi $\mathscr{N}(\mu;\sigma^2)$ ?
4. Conclure en donnant une fonction Python qui prend en argument $m \in \mathbb{N}^*$ et renvoie un échantillon de taille $2m$ de la loi normale centrée réduite avec l'histogramme de l'échantillon.

**Exercice 8.** ✦✦✦ **Simulation d'une loi de Poisson par des lois uniformes**

Soient $\lambda \in \mathbb{R}_*^+$ et $X_1, X_2, \ldots, X_n$ des variables aléatoires indépendantes, toutes de même loi exponentielle de paramètre 1. Soit N le plus grand entier $n$ tel que $X_1 + \cdots + X_n < \lambda$, en convenant que $N = 0$ si $X_1 > \lambda$. Autrement dit, pour $n \in \mathbb{N}$,

$$[N = n] = [X_1 + \cdots + X_n < \lambda] \cap [X_1 + \cdots + X_n + X_{n+1} \geqslant \lambda].$$

- *La théorie*

1. Vérifier que pour tout $n \in \mathbb{N}$, $\mathbf{P}([N \geqslant n]) = \mathbf{P}([N \geqslant n+1]) + \frac{\lambda^n}{n!} e^{-\lambda}$.
2. En déduire que N suit une loi de Poisson de paramètre $\lambda$.
3. On considère maintenant $(U_n)_{n\in\mathbb{N}^*}$, une suite de variables aléatoires mutuellement indépendantes toutes de loi uniforme sur $]0;1]$. On définit la variable aléatoire X par

$$\forall \omega \in \Omega, \quad X(\omega) = \begin{cases} 0 \text{ si } U_1(\omega) < e^{-\lambda} \\ \min\left\{ n \in \mathbb{N}^* \;\middle|\; \prod_{i=1}^{n+1} U_i(\omega) \leqslant e^{-\lambda} \right\} \text{ sinon.} \end{cases}$$

   Démontrer que X suit une loi de Poisson de paramètre $\lambda$.
   *Pour rappel, si* $U \hookrightarrow \mathscr{U}([0;1])$, *alors* $-\ln(U) \hookrightarrow \mathscr{E}(1)$.

- *La pratique*

**4.** En déduire un programme qui simule la loi de Poisson de paramètre $\lambda \in \mathbb{R}_*^+$.

### Exercice 9. ✦✦✦ Simulations d'une loi bêta

Soient $\alpha, \beta \in \mathbb{R}_*^+$. On pose $$\mathrm{B}(\alpha,\beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}\,\mathrm{d}t.$$

On considère la fonction $f_{\alpha,\beta}$ définie sur $\mathbb{R}$ par $f_{\alpha,\beta}(x) = \dfrac{1}{\mathrm{B}(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}\,\mathbf{1}_{]0;1[}(x)$.

**1.** *La loi bêta*

**a)** Vérifier que $f_{\alpha,\beta}$ est une densité de probabilité. On parle alors d'une loi bêta de paramètre $(\alpha,\beta)$.

**b)** Écrire un programme qui prend en argument le paramètre $(\alpha,\beta)$ et renvoie la courbe de $f_{\alpha,\beta}$ sur $[0;1]$.
*On pourra utiliser la commande* `sp.beta(alpha,beta)` après avoir importé la bibliothèque `scipy.special` *via* `sp`.

**2.** *Simulation pour des paramètres entiers - statistiques d'ordre et loi uniforme*
Soient $n \in \mathbb{N}^*$ et $\mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_n$, $n$ variables aléatoires indépendantes et de loi uniforme sur $[0;1]$.
On admet l'existence de variables aléatoires à densité $\mathrm{Y}_1, \mathrm{Y}_2, \ldots, \mathrm{Y}_n$ telles que, pour tout $\omega$ de $\Omega$, les réels $\mathrm{Y}_1(\omega)$, $\mathrm{Y}_2(\omega)$, …, $\mathrm{Y}_n(\omega)$ constituent un réarrangement par ordre croissant des réels $\mathrm{X}_1(\omega)$, $\mathrm{X}_2(\omega)$, …, $\mathrm{X}_n(\omega)$ de telle sorte que, pour tout $\omega$ de $\Omega$ :

$$\mathrm{Y}_1(\omega) \leqslant \mathrm{Y}_2(\omega) \leqslant \cdots \leqslant \mathrm{Y}_n(\omega).$$

En particulier, $\mathrm{Y}_1 = \min(\mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_n)$ et $\mathrm{Y}_n = \max(\mathrm{X}_1, \mathrm{X}_2, \ldots, \mathrm{X}_n)$.

**a)** Écrire un programme qui prend en arguments $n \in \mathbb{N}^*$ et $k \in [\![1;n]\!]$ et renvoie une simulation de $\mathrm{Y}_k$.
*Indication. On pourra utiliser la commande* `np.sort(A)` *qui prend en argument une matrice ligne et renvoie une matrice mais avec les coefficients de* `A` *ordonnés par ordre croissant.*

**b)** En déduire un programme qui prend en arguments $n \in \mathbb{N}^*$ et $k \in [\![1;n]\!]$ et affiche l'histogramme associé à 5 000 réalisations de la variable $\mathrm{Y}_k$ ainsi que la courbe représentative de la fonction $f_{k,n+1-k}$. Tester et commenter.

**3.** *Simulation pour des paramètres supérieurs à 1 - méthode du rejet*
Comment simuler une loi bêta de paramètres $(\alpha;\beta)$ par la méthode de rejet ?

**4.** *Simulation - cas général avec l'algorithme de Jönk*
Soient $(\mathrm{U}_n)_{n\in\mathbb{N}^*}$ et $(\mathrm{V}_n)_{n\in\mathbb{N}^*}$ deux suites de variables aléatoires mutuellement indépendantes et suivant toutes la loi uniforme sur $]0;1]$. On note

$$\mathrm{N} = \inf\left\{ n \in \mathbb{N}^* \mid {\mathrm{U}_n}^{1/\alpha} + {\mathrm{V}_n}^{1/\beta} \leqslant 1 \right\}.$$

On admet alors que $\dfrac{{\mathrm{U}_\mathrm{N}}^{1/\alpha}}{{\mathrm{U}_\mathrm{N}}^{1/\alpha} + {\mathrm{V}_\mathrm{N}}^{1/\beta}}$ suit une loi bêta de paramètres $(\alpha,\beta)$.
En déduire un programme qui simule une loi bêta de paramètres $(\alpha,\beta)$.

**5.** *Estimations des paramètres*

**a)** Écrire un programme Python qui prend en argument $\alpha$, $\beta$ et renvoie $\overline{x}$ (resp. $v$), la moyenne arithmétique (resp. la variance empirique) de 5000 réalisations d'une loi bêta de paramètre $(\alpha;\beta)$.

**b)** *Facultatif.* Donner la valeur exacte de l'espérance et la variance d'une loi bêta de paramètre $(\alpha,\beta)$.
*On pourra admettre que pour tous $a$, $b \in \mathbb{R}_*^+$*

$$\mathrm{B}(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \quad \textit{et} \quad \Gamma(a+1) = a\Gamma(a).$$

**6.** Modifier le programme précédent en rajoutant le calcul des quantités suivantes. Tester et commenter.

$$\overline{x}\left(\frac{\overline{x}(1-\overline{x})}{v} - 1\right) \quad \text{et} \quad (1-\overline{x})\left(\frac{\overline{x}(1-\overline{x})}{v} - 1\right).$$

## 2.2 Exercices supplémentaires

**Exercice 10.** ✦ **Autour de la loi de Cauchy**

**1.** Soient X et Y deux variables aléatoires indépendantes suivant toutes les deux une loi normale centrée réduite.

**a)** En utilisant la commande `rd.normal(0,1)`, simuler la loi de Z = X/Y.

**b)** Comment obtenir un échantillon de Z contenant 10000 réalisations ? Calculer la moyenne empirique pour plusieurs échantillons. Commenter les résultats.

**c)** Tracer l'histogramme d'un échantillon issu de Z sur $[-10,10]$ avec 10000 réalisations de Z et 30 classes. Superposer la courbe représentative de la fonction $x \mapsto \frac{1}{\pi(1+x^2)}$. Commenter.

**2.** On dit qu'une variable aléatoire suit une loi de Cauchy de paramètre $\lambda \in \mathbb{R}_*^+$, noté $X \hookrightarrow \mathscr{C}(\lambda)$ si une densité est donnée sur $\mathbb{R}$ par :

$$\forall\, x \in \mathbb{R}, \qquad f(t) = \frac{1}{\pi}\left(\frac{\lambda}{x^2+\lambda^2}\right).$$

**a)** Si $X \hookrightarrow \mathscr{C}(1)$, quelle est la loi de $\alpha X$ où $\alpha \in \mathbb{R}_*^+$ ? En déduire un programme qui permet de simuler une loi de Cauchy $\mathscr{C}(\lambda)$.

**b)** Soient X et Y, deux variables aléatoires indépendantes respectivement de loi $\mathscr{C}(\lambda)$ et $\mathscr{C}(\mu)$. On souhaite tester le résultat suivant :

$$X + Y \hookrightarrow \mathscr{C}(\lambda+\mu).$$

Proposer un script Python qui permet de tester ce résultat à l'aide d'histogrammes.

**c)** Soient $(X_i)_{i\in\mathbb{N}^*}$, une suite de variables aléatoires mutuellement indépendantes et toutes de loi $\mathscr{C}(1)$. Que dire de la loi de

$$\frac{1}{n}\sum_{i=1}^{n} X_i \quad ?$$

**Exercice 11.** ✦✦ **Autour de la loi de Fréchet**

Soit $s \in \mathbb{R}_*^+$. On dit qu'une variable aléatoire X suit une loi de Fréchet de paramètre $s$, noté $\mathscr{F}(s)$, si sa fonction de répartition est donnée par

$$\forall\, x \in \mathbb{R}, \qquad F_s(x) = \begin{cases} \exp\left(-s/x\right) & \text{si } x > 0 \\ 0 & \text{sinon.} \end{cases}$$

**1.** Démontrer que : Si $U \hookrightarrow \mathscr{U}([0,1])$ (loi uniforme continue) alors $s(-\ln(U))^{-1} \hookrightarrow \mathscr{F}(s)$.

**2.** Est-ce qu'une loi de Fréchet admet une espérance ? une variance ?

**3.** Écrire un programme qui prend en argument $s$ et simule une loi de Fréchet de paramètre $s$.

**4.** *Loi d'un maximum de loi de Fréchet*
Soient $X_1, \ldots, X_n$, $n$ variables aléatoires définies sur un même espace probabilisé, indépendantes et toutes de loi de Fréchet de paramètre $s$. On pose

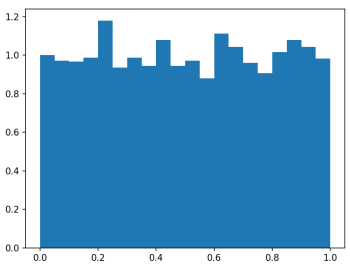
$$Y_n = \max(X_1, X_2, \ldots, X_n).$$

Écrire une fonction Python `MaxFrechet` qui prend en argument $s$, $n$ et un entier naturel $m$ et renvoie un échantillon de taille $m$ de la variable $Y_n$.

**5.** À l'aide du code suivant et des résultats numériques. Conjecturer la loi de $Y_n$.

Editeur

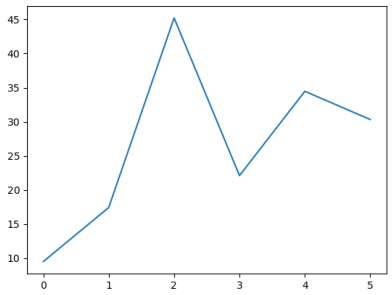
```
def Truc(s,n):
    plt.clf()
    Ech=MaxFrechet(s,n,5000)
    L=np.exp(-(Ech/(n*s))**(-1))
    plt.hist(L,20,density=True)
    plt.show()
```




**6.** Quelle propriété de la loi de Fréchet illustre le code et les résultats suivants ?

Editeur

```
L=np.zeros(6)

for p in range(2,8):
    S=0
    for k in range(1,10**p):
        S+=frechet(1)
    L[k]=(S/10**p)
plt.plot(L)
plt.show()
```




```
L=[9.52163, 17.43712, 45.20899, 22.11409,
    34.46171, 30.32867]
```

**Exercice 12.** ✦✦ **Exemple de question à l'oral d'HEC**

1. Commenter cette fonction Python.
2. Déterminer le graphe de sortie de la fonction `graphe()`.

Editeur

```
def g(x):
    X=np.random.uniform(0,0.5,10000)
    Y=np.random.uniform(0,1/3,10000)
    S=0
    for k in range(10000):
        if X[k]+Y[k]<=x:
            S=S+1
    return S/10000
```

```
def graphe():
    x=np.linspace(-0.1,1,101)
    Z=np.zeros(101)
    for k in range(101):
        Z[k]=g(x[k])
    plt.plot(x,Z)
    plt.show()
```

# 3 Convergence des variables aléatoires

## 3.1 Exemples et applications

**Exercice 13.** ✦✦ **Approche expérimentale du théorème limite central**

1. *Diagramme en bâtons et densité*
   Rappelons :
   - Un diagramme en bâtons s'obtient par les commandes `plt.bar(x,y,width=1)` où `x` représente les abscisses et `y` les ordonnées.
   - Le coefficient $\binom{n}{k}$ peut s'obtenir directement par `sp.binom(n,k)`.
   - Pour limiter le graphe aux abscisses comprises entre `a` et `b`, on peut rajouter la ligne `plt.xlim(a,b)`.

   **a)** Expliciter la loi de $S_n^* = \frac{X_n - np}{\sqrt{np(1-p)}}$ où $X_n \hookrightarrow \mathscr{B}(n;p)$.

   **b)** Écrire une fonction qui prend en arguments $n \in \mathbb{N}^*$, $p \in ]0;1[$ et affiche le diagramme en bâtons associé à la loi de $S_n^*$. C'est-à-dire que l'on place en abscisse les valeurs prises par la variable aléatoire $S_n^*$ et en ordonnée les probabilités correspondantes.
   *On multipliera les ordonnées du diagramme en bâtons par $\sqrt{np(1-p)}$ afin que la sommes des aires des rectangles soit de 1.*

   **c)** Rajouter sur le graphique précédent la courbe représentative de $x \mapsto e^{-t^2/2}/\sqrt{2\pi}$.
   *On se limitera à l'intervalle* $[-4;4]$.

   **d)** Tester par exemple pour $n \in \{10;20;30;60\}$, $p \in \{0.3;0.5\}$ et commenter.

2. *Histogramme et densité*

   **a)** Comment simuler la variable $S_n^*$ ?

**b)** Écrire une fonction qui prend en arguments $n \in \mathbb{N}^*$, $p \in ]0;1[$ et affiche sur le même graphe un histogramme de 10 000 réalisations de $S_n^*$ et la courbe représentative de $x \mapsto e^{-t^2/2}/\sqrt{2\pi}$.
*On pourra se restreindre à l'intervalle* $[-4;4]$ *et considérer 30 classes.*

**c)** Tester pour $n \in \{500;2000;5000\}$, $p \in \{0.3;0.5\}$ et commenter.

**Exercice 14.** ✦✦ **Comparaison des lois $\mathscr{B}\left(n;\frac{\lambda}{n}\right)$ et $\mathscr{P}(\lambda)$**

**1.** *Comparaison des diagrammes en bâtons*

**a)** Écrire un programme qui prend en arguments $n \in \mathbb{N}^*$, $p \in ]0;1[$ puis renvoie le diagramme en bâtons associé à la loi binomiale $\mathscr{B}(n;p)$.

**b)** Soient $\lambda \in ]0,+\infty[$ et $X \hookrightarrow \mathscr{P}(\lambda)$.

i) Écrire un programme qui prend en arguments $n$, $\lambda$ et renvoie la matrice ligne

$$\begin{bmatrix} p_0 & p_1 & \dots & p_n \end{bmatrix} \quad \text{où} \quad p_i = \frac{\lambda^i}{i!}e^{-\lambda}.$$

*On pourra remarquer que* $p_{i+1} = \lambda p_i/(i+1)$.

ii) En déduire un second programme qui prend en argument $\lambda$ et $n$ renvoie le diagramme en bâtons sur $[\![0;n_\lambda]\!]$ de la loi de Poisson $\mathscr{P}(\lambda)$.

**c)** Pour différentes valeurs de $n$ et $\lambda$, superposer les diagrammes en bâtons associés aux lois $\mathscr{B}(n;\lambda/n)$ et $\mathscr{P}(\lambda)$. Commenter.

**2.** *Comparaison des histogrammes*
Sur un même graphique, placer les histogrammes issus de 5000 réalisations des lois $\mathscr{B}(n;\lambda/n)$ et $\mathscr{P}(\lambda)$. Tester pour $\lambda = 5$ et des valeurs de $n$ de plus en plus grande. Commenter.

**Exercice 15.** ✦ **Convergence et loi de Gumbel**
On dit que X suit la loi de Gumbel si sa fonction de répartition F est définie par

$$\forall x \in \mathbb{R}, \qquad F(x) = e^{-e^{-x}}.$$

On considère aussi une suite de variables aléatoires indépendantes $(Y_n)_{n\in\mathbb{N}^*}$ suivant chacune la loi exponentielle $\mathscr{E}(1)$.

**1.** Écrire un programme qui prend en argument $n \in \mathbb{N}^*$ et simule la variable aléatoire

$$Z_n = \max(Y_1,\dots,Y_n) - \ln n.$$

**2.** 💻 Afficher sur un même graphique un histogramme associé à un échantillon de 1000 réalisations de $Z_n$ et une densité de la loi de Gumbel. On pourra se restreindre à l'intervalle $[-2;7]$. Tester pour $n = 5$, 10, 50 ... Commenter.

**Exercice 16.** ✦ 🔑 **Simulation d'une loi normale par la méthode des douze uniformes**
Le théorème limite central énonce que si $(X_n)_{n\in\mathbb{N}^*}$ est une suite de variables aléatoires mutuellement indépendantes suivant une loi uniforme sur $[0;1]$ alors

$$\overline{X}_n{}^* \xrightarrow[n\to+\infty]{\mathscr{L}} Z \quad \text{avec} \quad Z \hookrightarrow \mathscr{N}(0;1).$$

**1.** En se limitant à douze variables $X_1$, …, $X_{12}$ suivant des lois uniformes, écrire une fonction Python qui renvoie une simulation approchée de la loi normale centrée réduite.

**2.** En déduire une seconde fonction qui prend en arguments $\mu$, $\sigma$, $m$ et renvoient $m$ simulations d'une loi $\mathscr{N}(\mu,\sigma^2)$.

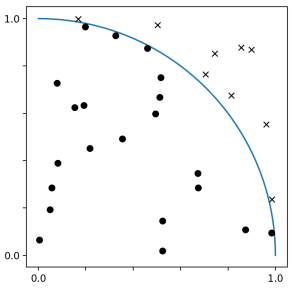
**3.** 💻 Tester votre programme en superposant sur une même graphique l'histogramme de 2000 simulations approchées de loi $\mathscr{N}(1;4)$ et une densité de cette loi.

## 3.2 Exemples de méthodes de Monte-Carlo

Les méthodes dites de Monte-Carlo sont toutes basées sur la loi (faible) des grands nombres.

### Applications aux calculs d'aire

Commençons par un cas d'école : l'approximation de $\pi$.
On tire au hasard et uniformément un point dans le carré $[0;1]\times[0;1]$. La probabilité que le point soit situé dans le quart de disque est $\pi/4$ (aire du quart du disque $\pi\times 1^2/4$ sur l'aire du carré 1). Partant de ce constat, on peut simuler un grand nombre de tirages d'un point dans le carré et approximer la probabilité de $\pi/4$ par la fréquence empirique. En multipliant par 4, on obtient une approximation de $\pi$.
Ce qui donne ici :




**Editeur**

```
def approxPI(m):
# m correspond au nombre de tirages
    Compteur=0
    for i in range(m):
        x=rd.random()
        y=rd.random()
        if x**2+y**2<1:
            Compteur+=1
    print('Approximation:',4*Compteur/m)
```

**Console**

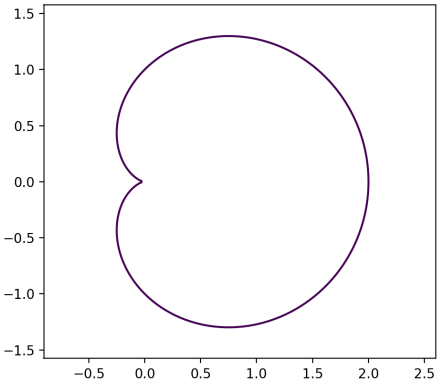
```
>>> approxPI(1000)   Approximation: 3.152
>>> approxPI(10000)  Approximation: 3.1412
>>> approxPI(50000)  Approximation: 3.14552
>>> approxPI(100000) Approximation: 3.14632

# à comparer à :
3.141592653589793 ...
```

La convergence est assez mauvaise mais il ne faut pas pour autant écarter la méthode. Elle s'avère par exemple particulièrement efficace en grande dimension.

**Exercice 17.** ✦✦ **Aire d'une cardioïde**

L'objectif de cet exercice est d'obtenir une approximation de l'aire de la partie délimitée par la courbe d'équation

$$\left(x^2+y^2-x\right)^2 = x^2+y^2.$$




1. Comment tirer un point au hasard dans le carré $[-0.5;2.5]\times[-1.5;1.5]$ en utilisant la commande **rd.random** ?
2. En déduire un programme qui tire au hasard un point dans le carré et déclare si le point est à l'intérieur de la cardioïde ou non.
3. À l'aide d'une méthode de Monte-Carlo, donner une approximation de l'aire de la cardioïde.
4. En remarquant que la courbe est la ligne de niveau $L_0$ d'une certaine fonction de deux variables, tracer la cardioïde.

### Application à l'approximation d'intégrales

- **Principe**
  Considérons :
  - $g:[0,1]\to[a;b]$ une fonction continue dont on souhaite calculer l'intégrale $\displaystyle\int_0^1 g(t)\,\mathrm{d}t$.
  - $(U_i)_{i\in\mathbb{N}^*}$ une suite de variables aléatoires indépendantes suivant la loi uniforme sur $[0,1]$.
  - Pour tout $i\in\mathbb{N}^*$, on pose $g(U_i)$. D'après les lemme d'égalité en loi et des coalitions, les variables sont indépendantes et de même loi.

Par le théorème de transfert, les variables $X_i = g(U_i)$ admettent toutes une même espérance donnée par

$$\mathbf{E}(X_i) = \int_0^1 g(t)\,\mathrm{d}t.$$

De même, ces variables admettent une variance et la loi faible des grands nombres donne

$$\frac{X_1+\cdots+X_n}{n} \xrightarrow[n\to+\infty]{\mathbf{P}} \mathbf{E}(X_1) = \int_0^1 g(t)\,\mathrm{d}t.$$

Dès lors, pour calculer une valeur approchée de l'intégrale, on peut simuler un grand nombre de fois les variables $U_i$, calculer les images $g(U_i)$ et en faire leur moyenne arithmétique.

**Exercice 18.** ✦✦

- *La théorie : estimation de la probabilité de l'erreur*
  Soit X, une variable aléatoire à valeurs dans $[a;b]$.

**1.** Pour quelle valeur de $m$, $\mathbf{E}\left((X-m)^2\right)$ atteint son minimum ? Avec $m=\frac{a+b}{2}$, déduire : $\mathbf{V}(X)\leqslant\dfrac{(b-a)^2}{4}$.

**2.** En reprenant les notations du début, justifier que

$$\forall\varepsilon\in\mathbb{R}_*^+,\qquad \mathbf{P}\left(\left|\int_0^1 g(t)\mathrm{d}t-\frac{1}{n}\sum_{k=1}^{n}g(U_k)\right|>\varepsilon\right)\leqslant\frac{(b-a)^2}{4n\varepsilon^2}\quad(\bullet)$$

- *La pratique*

**3.** Calculer $I=\displaystyle\int_0^1\frac{1}{1+t^2}\mathrm{d}t$. En déduire, une fonction avec en argument $n$ et qui permet d'approcher $\pi$.

**4.** Déterminer $n$ afin d'obtenir une valeur à $10^{-3}$ près de $\pi$ avec une probabilité d'au moins 95%. Commenter.

# 4 Estimations

**Exercice 19.** ✦ Soit $(X_1,\dots,X_n)$ un échantillon suivant la loi de Poisson $\mathscr{P}(\lambda)$. L'objectif de cet exercice est de comparer deux estimateurs du paramètre inconnu $\lambda$ :

- La moyenne empirique $\overline{X_n}=\frac{1}{n}\sum_{i=1}^{n}X_i$ ;
- La variance empirique $\sigma_n^2=\frac{1}{n-1}\sum_{k=1}^{n}\left(X_k-\overline{X_n}\right)^2$.

On souhaite comparer ces deux estimateurs.

**1.** Écrire un programme qui prend en argument $n$ et $\lambda$ et simule $\overline{X_n}$, puis $\sigma_n^2$.

**2.** 💻 Afficher des histogrammes et comparer les estimateurs.
*On pourra prendre par exemple* $\lambda=5$ *et* $n=200$.

**Exercice 20.** ✦ Soient $n\in\mathbb{N}^*$ et $(X_1,X_2,\dots,X_n)$ un échantillon d'une loi de Poisson $\mathscr{P}(\theta)$. On cherche à estimer $\exp(-\theta)$. Pour cela, on pose :

$$A_n=\left(1-\frac{1}{n}\right)^{\sum_{i=1}^{n}X_i}\quad\text{et}\quad B_n=\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}_{[X_i=0]}.$$

**1.** Écrire deux programmes qui prend en arguments $n$, $\theta$ et simulent respectivement $A_n$ et $B_n$.

**2.** On suppose dans la suite que $\theta=1$.
En déduire un programme qui prend en argument $n$ et renvoie une approximation de l'espérance de $A_n$ et $B_n$.
Que peut-on conjecturer sur le biais de chacun de ces estimateurs ?

**3.** Afficher l'histogramme de 20000 réalisations de $A_{100}$ et $B_{100}$. Quel est le meilleur estimateur de $\exp(-\theta)$ ?
*On pourra considérer les classes :* `classe=np.linspace(0.2,0.5,20)`.

**Exercice 21.** ✦✦ Soit $(X_n)_{n\in\mathbb{N}^*}$ une suite de variables indépendantes de même loi uniforme discrète sur $[\![1;N]\!]$. Pour tout $n\in\mathbb{N}^*$ et $k\in[\![1;N]\!]$, on pose :

- $Y_n$ le nombre de valeurs distinctes prises par $X_1,\dots,X_n$.
- $Z_{k,n}$ égale à 1 si au moins l'un des $X_i$ pour $i\in[\![1;n]\!]$ prend la valeur $k$ et 0 sinon.

**1.** *La pratique*

**a)** 🔑 Écrire un programme qui prend en arguments $n$, N et simule le vecteur aléatoire

$$\begin{bmatrix}Z_{1,n} & Z_{2,n} & \dots & Z_{N-1,n} & Z_{N,n}\end{bmatrix}.$$

**b)** 🔑 En déduire un programme qui prend en arguments $n$, N et simule $Y_n$.

**c)** Vérifier numériquement que $Y_n$ est un estimateur asymptotiquement sans biais de N.
*Un estimateur est asymptotiquement sans biais si son biais* $b_n$ *tend vers 0 lorsque* $n\to+\infty$.

**2.** *La théorie*

**a)** Justifier que pour tout $k\in[\![1;N]\!]$, $\mathbf{E}\left(Z_{k,n}\right)=1-\left(1-\frac{1}{N}\right)^n$.

**b)** En déduire que $Y_n$ est un estimateur asymptotiquement sans biais.

**c)** Montrer en plus que $Y_n$ est un estimateur convergent de N.

**Exercice 22.** ✦✦ **Comparaison des intervalles de confiance - loi de Bernoulli**

Soit $(X_i)_{i\in\mathbb{N}^*}$, une suite de variables aléatoires indépendantes et de même loi de Bernoulli $\mathscr{B}(p)$. Pour les tests, on choisit $p = 1/3$, $n = 1000$, $\alpha = 0.05$.

**1.** Écrire un programme `ech` qui prend en arguments $n$, $p$ et renvoie une réalisation de $(X_1,\dots,X_n)$.

- *Estimation à partir de l'inégalité de Bienaymé-Tchebychev*
  On rappelle que l'intervalle de confiance de $p$ à un niveau de confiance $1-\alpha$ est donné par :
$$\left[\overline{X}_n - \frac{1}{2\sqrt{n\alpha}}\,;\,\overline{X}_n + \frac{1}{2\sqrt{n\alpha}}\right].$$

**2.** **a)** Écrire un programme Python qui prend en arguments $n$, $p$, $\alpha$ puis :
- → Crée un échantillon de taille $n$ ;
- → Calcule l'intervalle de confiance associé à l'échantillon ;
- → Renvoie 1 si le paramètre $p$ est bien dans l'intervalle, 0 sinon.

**b)** Afficher ensuite une approximation de la probabilité que le paramètre $p$ soit bien dans l'intervalle de confiance calculé ainsi que la longueur de l'intervalle de confiance.

- *Intervalle de confiance asymptotique du paramètre d'une loi de Bernoulli*
  On a démontré qu'un intervalle de confiance asymptotique de $p$ au niveau de confiance $1-\alpha$ est
$$\left[\overline{X_n} - \frac{t_\alpha}{2\sqrt{n}}\,;\,\overline{X_n} + \frac{t_\alpha}{2\sqrt{n}}\right]$$
  où $t_\alpha$ est la solution de $\Phi(t_\alpha) = 1-\frac{\alpha}{2}$ avec $\Phi$ la fonction de répartition de la loi normale centrée réduite. Par exemple, $t_{0.05} \simeq 1.96$.

**3.** Reprendre la question 2 avec ce nouvel intervalle.

**4.** Comparer et commenter les résultats obtenus.

**Exercice 23.** ✦ **Estimation par intervalle de confiance de la moyenne d'une loi normale d'écart type connu**

Dans une population donnée, une étude statistique faite sur un groupe de 100 personnes donne lieu à la série statistique suivante.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Poids | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |
| Effectif | 3 | 5 | 2 | 6 | 6 | 10 | 12 | 10 |
| Poids | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| Effectif | 9 | 8 | 8 | 6 | 5 | 4 | 3 | 3 |

On suppose que le poids d'un individu du groupe est une variable aléatoire X qui suit une loi normale d'écart-type $\sigma = 3,5$. Dans chaque groupe de 100 personnes étudié, on désigne par $X_i$ la variable aléatoire égale au poids du $i$-ème individu, pour tout $i \in [\![1;100]\!]$.

**1.** Calculer l'intervalle de confiance obtenu par l'inégalité de Bienaymé-Tchebychev.

**2.** En utilisant les propriétés de stabilité des lois normales, on montre qu'un intervalle de confiance de niveau $1-\alpha$ est donné par
$$\left[\overline{X}_n - t_\alpha\frac{\sigma}{\sqrt{n}}\,;\,\overline{X}_n + t_\alpha\frac{\sigma}{\sqrt{n}}\right].$$
Expliciter cet intervalle avec les données obtenues.

**3.** Comparer les deux méthodes.

# 5 Compléments sur les fonctions de deux variables

**Exercice 24.** ✧ La fonction définie sur $\left(\mathbb{R}_+^*\right)^2$ par $f(x,y) = x^y - y^x$ possède (e, e) comme point critique. Retrouver graphiquement sa nature.

**Exercice 25.** ✦ **Points fixes dégénérés**

On définit les fonctions $f$, $g$ et $h$ sur $\mathbb{R}^2$ par :
$$f(x,y) = x^2 + y^3 \quad \text{et} \quad g(x,y) = x^3 + xy^2 - x^2y - y^3, \quad h(x,y) = x^4 + y^3 - 3y - 2.$$

**1.** Vérifier que dans les deux cas $(0;0)$ est un point critique.

**2.** Tracer les surfaces représentatives et conjecturer la nature du point critique $(0;0)$.

# 6 Exercices supplémentaires et sujets

**Exercice 26.** ✦ Soit $\mathscr{F} = (f_1, f_2, \dots, f_n)$ une famille de $n$ vecteurs de $\mathbb{R}^n$. On note P la matrice de la famille $\mathscr{F}$ dans la base canonique. Proposer un programme qui prend en argument la matrice P et indique si $\mathscr{F}$ est une base orthonormée ou non.

**Exercice 27.** ✦✦ Soit $p \in \mathbb{N}^*$.

1. Quelle est la loi suivie par la variable N ?
2. Préciser $X(\Omega)$. Calculer pour tout couple d'entiers $(i,k)$ la probabilité $\mathbf{P}_{(N=k)}(X=i)$. En déduire la loi de X (on ne cherchera pas à simplifier la somme obtenue).
3. Calculer $\mathbf{E}(X)$

Editeur

```
import numpy.random as rd
def X(p):
    N=1+np.floor(p*rd.rand())
    # floor(.) renvoie la partie entière
    x=0
    i=0
    while i <N:
        x=x+(rd.random()<0.5)
        i+=1
    return x
```

**Exercice 28.** ✦ **Suites de Fibonacci aléatoires**

- *Le cas déterministe*
  On considère la suite $(f_n)_{n\in\mathbb{N}}$ définie par : $f_0 = 0$, $f_1 = 1$ et $\forall n \in \mathbb{N}$, $f_{n+2} = f_{n+1} + f_n$.

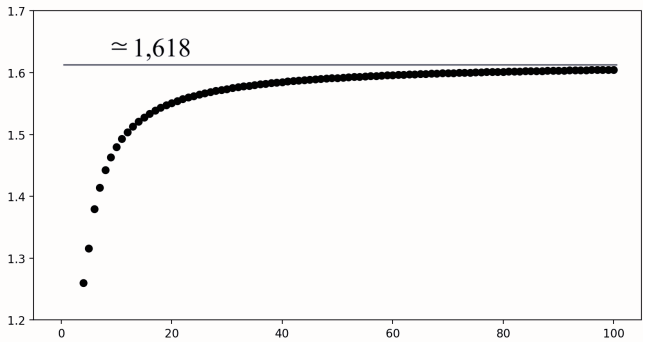
1. Écrire un programme qui prend en argument $n$ et renvoie la matrice ligne

$$L_n = \begin{bmatrix} f_0 & f_1 & f_2 & \dots & f_n \end{bmatrix}.$$

2. Que permet de conjecturer le code et résultat suivants ?

Editeur

```
n=100; L=Fibo(n); N=np.zeros(n)
for i in range(1,n):
    N[i]=L[i]**(1/i)
NbreOr=(1+np.sqrt(5))/2
plt.ylim(1.2,1.7)
 # pour restreindre l'affichage en
     ordonnée
plt.plot([0,n],[NbreOr,NbreOr])
plt.plot(np.linspace(1,n,n),N,'ko')
plt.show()
```




- *Le cas aléatoire*
  Soient $p \in [0;1]$ et $(U_n)_{n\in\mathbb{N}}$, $(V_n)_{n\in\mathbb{N}}$, deux suites de variables aléatoires discrètes définies sur le même espace probabilisé, indépendantes et de même loi que la variable U

$$\mathbf{P}(U=-1) = 1-p \quad \text{et} \quad \mathbf{P}(U=1) = p.$$

  On définit alors la suite de variables aléatoires $(F_n)_{n\in\mathbb{N}}$ par la récurrence

$$F_0 = 0, \quad F_1 = 1 \quad \text{et} \quad \forall n \in \mathbb{N}, \qquad F_{n+2} = U_n \cdot F_{n+1} + V_n \cdot F_n.$$

  On pose de plus les matrices aléatoires :

$$\forall n \in \mathbb{N}, \qquad A_n = \begin{bmatrix} 0 & V_n \\ 1 & U_n \end{bmatrix} \quad \text{et} \quad L_n = \begin{bmatrix} F_n & F_{n+1} \end{bmatrix}.$$

3. Donner une relation simple entre $A_n$, $L_n$ et $L_{n+1}$. En déduire l'existence d'une matrice $B_n$, que l'on exprimera sous forme de produit telle que $L_n = L_0 B_n$.
4. Écrire un programme `SimuU` qui prend en argument $p$ puis simule la variable U ? En déduire un second programme qui simule la matrice $A_n$.
5. Comment simuler la matrice $B_n$ ? La variable $F_n$ ? Comment tester ce dernier programme avec $p = 1$ ?
   *On rappelle que le produit matriciel en python peut s'obtenir par la commande* `np.dot( , )`.

6. Écrire un programme qui prend en argument une matrice carrée A de taille $n$ et renvoie sa norme définie par $\|A\| = \sqrt{\sum_{i,j=1}^{n} {a_{i,j}}^2}$.

7. Un théorème des mathématiciens Furstenberg et Kesten affirme que la norme matricielle de $B_n$ croît approximativement en $\lambda^n$ pour un certain $\lambda$. Proposer une méthode pour vérifier numériquement cette affirmation.

**Exercice 29.** ✦ Quelle loi est simulée par le code suivant ?

Editeur

```
import numpy as np

def Mystere():
    u=np.random.rand(4)
    return -np.log(np.prod(u))
```

**Exercice 30.** ✦✦ **Retour sur la méthode de Monte-Carlo**
Soient $(U_i)_{i\in\mathbb{N}^*}$ des variables aléatoires réelles indépendantes de loi uniforme sur $[0;1]$ et N une variable aléatoire de loi géométrique de paramètre $p$ indépendante de la suite $(U_i)_{i\in\mathbb{N}^*}$. On pose $q = 1-p$ et

$$X = \max_{i\in[\![1;N]\!]} U_i.$$

1. Déterminer la fonction de répartition de la variable aléatoire X. Comment tracer son graphe avec python ?
2. Justifier que $\mathbf{E}(X)$ admet une espérance avec $\mathbf{E}(X) = 1 + pq^{-2}\big(\ln(p) + q\big)$.
   *On pourra utiliser l'égalité* $\ln(1-x) = -\sum_{n=1}^{+\infty} x^n/n$ *pour* $x \in [-1;1[$.
3. Simuler la variable et vérifier votre résultat.

**Exercice 31.** ✦✦ **Exemple de marche aléatoire asymétrique**
Soit $(X_n)_{n\in\mathbb{N}^*}$ une suite de variables aléatoires mutuellement indépendantes telles que pour tout $n \in \mathbb{N}^*$,

$$\mathbf{P}(X_n = 1) = p \quad \text{et} \quad \mathbf{P}(X_n = -1) = 1-p.$$

1. Écrire une fonction prenant en argument $p$ et simule la variable aléatoire $X_1$.
2. On pose $S_0 = 0$ et pour tout $n \in \mathbb{N}^*$, $S_n = X_1 + X_2 + \cdots + X_n$.
   Écrire une fonction d'arguments $p$ et $n$ qui renvoie la liste $[S_0, S_1, \cdots, S_n]$.
3. Conjecturer la limite de la suite $(S_n)$ en fonction de $p$.
4. Soit $n \in \mathbb{N}^*$. On note $T_n = \min\{k \in \mathbb{N} : |S_k| = n\}$. Écrire une fonction d'arguments $p$, $n$ qui renvoie une simulation de la variable aléatoire $T_n$.

**Exercice 32.** ✦✦ On coupe un morceau de bois de longueur 1 en trois morceaux.

1. Donner un exemple dans lequel il n'est pas possible de former un triangle.
2. *Dans la suite, on admet que pour obtenir un triangle (non plat), les inégalités suivantes doivent être vérifiées simultanément :*

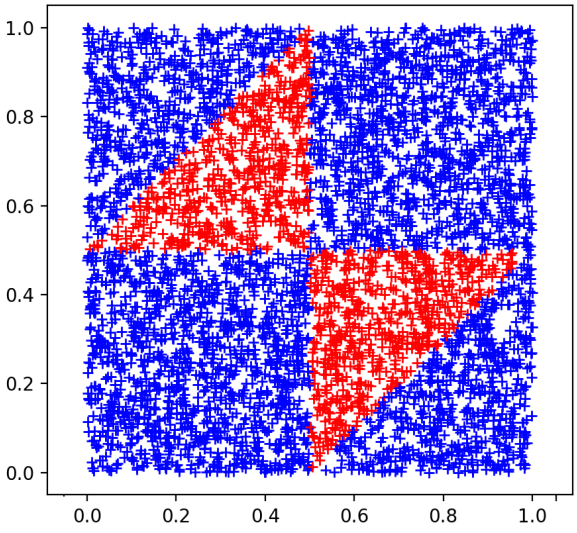
$$a < b+c, \quad b < a+c, \quad c < a+b$$

*où $a$, $b$ et $c$ sont les longueurs des trois morceaux obtenus.*

On écrit le script suivant en Python et on affiche le résultat obtenu. Que peut-on conjecturer sur la probabilité d'avoir un triangle à partir des trois morceaux coupés aléatoirement ?

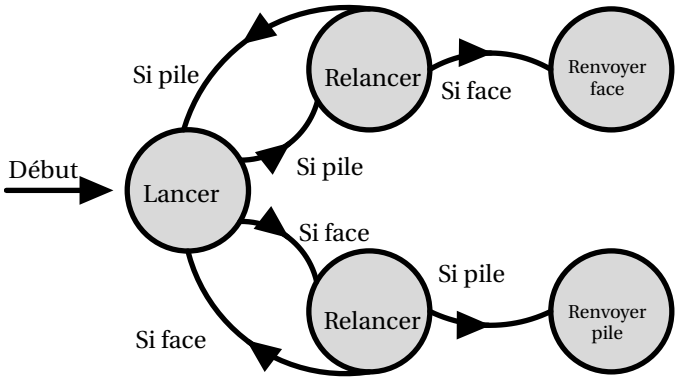
Editeur

```
m=5000
for i in range(m):
    x=np.random.rand()
    y=np.random.rand()
    L1=min(x,y)
    L2=max(x,y)-min(x,y)
    L3=1-max(x,y)
    if L1<L2+L3 and L2<L1+L3 and L3<L1+L2:
        plt.plot(x,y,'r+')
    # permet d'afficher une croix rouge
    else:
        plt.plot(x,y,'b+')
    # permet d'afficher une croix bleue
plt.axis('equal')

plt.show()
```




**Exercice 33.** ✦ **Pile - Face**

Le but de cet exercice est d'étudier un algorithme permettant de générer un pile ou face équilibré à partir d'une pièce truquée. On suppose que la pièce (truquée) renvoie pile avec probabilité $p \in ]0,1[$ avec $p \neq 1/2$. Voici une description schématique de l'algorithme. On se place dans le cas où les lancers successifs de la pièce sont mutuellement indépendants.




1. Écrire une fonction `Bernoulli` qui prend en argument $p \in [0;1]$ et simule une variable aléatoire de loi $\mathscr{B}(p)$.
2. Donner une fonction Python `algo` qui renvoie pile (1) ou face (0) en suivant l'algorithme décrit ci-dessus.
3. Effectuer 5000 réalisations et afficher les fréquences d'obtention de pile et face. Commenter.
4. Modifier la fonction `algo` pour afficher en plus le nombre de lancers nécessaires pour que l'algorithme se termine.
5. On note T la variable aléatoire donnant le nombre de lancers nécessaires pour que l'algorithme se termine. Estimer l'espérance **E**(T).

**Exercice 34.** ✦✦ **Le problème du collectionneur** *d'après HEC 2022*

Le but de ce problème est de mettre en évidence quelques résultats asymptotiques liés au modèle du collectionneur de vignettes. Dans chaque paquet de céréales se trouve une vignette et il y a en tout des vignettes de $n$ types différents, où $n$ est un entier supérieur ou égal à 1. Chacun des $n$ types de vignettes se retrouve avec la même fréquence dans les paquets de céréales. Une collection est alors complète lorsqu'elle comporte $n$ vignettes de types différents.

On modélise le nombre total de paquets de céréales qu'il est nécessaire d'acheter pour obtenir la collection complète de $n$ vignettes de types différents par la variable aléatoire notée $C_n$.

On pose par convention $C_0 = 0$ et pour tout entier $i \in [\![1;n]\!]$, on note $C_i$ le nombre d'achats de paquets de céréales nécessaires pour obtenir $i$ vignettes de types différents.

De même, pour tout $i \in [\![1;n]\!]$, on pose $X_i = C_i - C_{i-1}$, qui représente le nombre d'achats supplémentaires de paquets de céréales qu'il est nécessaire d'effectuer pour obtenir une nouvelle vignette d'un type différent des $(i-1)$ vignettes de types différents déjà obtenues. Par convention, on pose $X_1 = C_1 = 1$.

On suppose que les variables aléatoires $X_1, \ldots, X_n$ sont mutuellement indépendantes. Enfin, on pose :

$$V_n = \frac{C_n}{n} - \ln(n)$$

On admet que $C_n = X_1 + \cdots + X_n = \sum_{i=1}^{n} X_i$ et pour tout $i \in [\![1;n]\!]$, la variable aléatoire $X_i$ suit la loi géométrique $\mathscr{G}\left(\frac{n-i+1}{n}\right)$.

1. Écrire une fonction python qui prend en argument $p \in ]0;1[$ et simule un loi géométrique de paramètre $p$. On rappelle que la commande `np.random.rand()<p` permet de simuler une loi de Bernoulli de paramètre $p$.
2. Écrire une fonction d'en-tête `def simulV(n)` qui pour un entier $n$ fourni en entrée, renvoie une simulation de la variable aléatoire $V_n$ définie en introduction de ce problème.

3. À la suite de la fonction `simulV`, écrire une fonction `simulVech` qui construit un vecteur-ligne V contenant 1000 simulations indépendantes de la variable aléatoire $V_n$ pour un certain entier $n$ entré par l'utilisateur.

4. On complète ce programme par le code suivant. Tester le pour $n = 5$, $n = 10$, $n = 50$ et $n = 100$.

Editeur

```
import matplotlib.pyplot as plt

n=100

def f(x):
    return np.exp(-x-np.exp(-x))

absc=np.linspace(-2,10,100)
y=f(absc)

plt.clf()
plt.plot(absc,y)
plt.hist(simulVech(n),20,density=True)
plt.show()
```

5. Que peut-on observer sur ces figures ? Quelle conjecture peut-on en déduire pour la suite $(V_n)_{n\in\mathbb{N}^*}$ ?

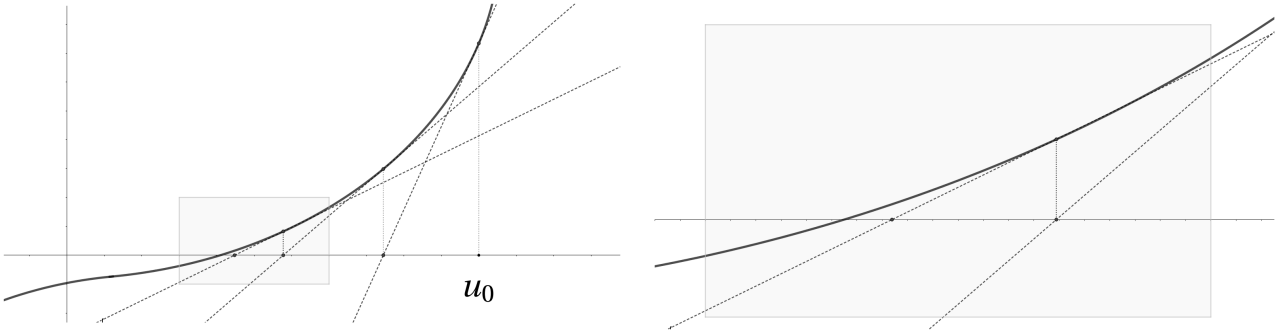
**Exercice 35.** ✦✦ **La méthode de Newton**

• *Partie A - Étude théorique*

Soit $f : [a;b] \to \mathbb{R}$ de classe $\mathscr{C}^1$. On suppose que $f'$ ne s'annule pas et que $f$ s'annule une unique fois en un point $c$. On construit la suite $u$ de la façon suivante :

- On part de $u_0 = a$.
- Pour tout $n \in \mathbb{N}$, $u_{n+1}$ est l'abscisse du point d'intersection de l'axe des abscisses et de la tangente à la courbe représentative de $f$ au point d'abscisse $u_n$.

1. Placer sur le graphe des courbes de $f$ et des tangentes, les réels $u_1$, $u_2$ et $u_3$.
Conjecturer le comportement limite de la suite $(u_n)_{n\in\mathbb{N}}$.




2. Vérifier que pour tout $n \in \mathbb{N}$, $\quad u_{n+1} = u_n - \dfrac{f(u_n)}{f'(u_n)}$.

3. Si la suite $u$ converge vers une limite finie, justifier que $u$ converge vers $c$.

4. Dans cette question, on prend l'exemple de $f : x \in [1;2] \mapsto x^2 - 2$.

**a)** Vérifier que pour tout $n \in \mathbb{N}$, $\quad u_{n+1} - \sqrt{2} = \dfrac{(u_n - \sqrt{2})^2}{2u_n}$.

Puis $\quad |u_{n+1} - \sqrt{2}| \leqslant |u_n - \sqrt{2}|^2 \quad$ et $\quad |u_n - \sqrt{2}| \leqslant |u_0 - \sqrt{2}|^{2^n}$.

**b)** Démontrer la convergence de la suite $u$ vers $\sqrt{2}$.

• *Partie B - Python*

5. Écrire un programme qui prend en entrée un entier $n$, $u_0$, $f$ et $f'$ et renvoie $u_n$.

**6.** Déduire de la question 4. une fonction Python qui prend en argument une précision $p$ et renvoie une approximation de $\sqrt{2}$ à $p$-près.

• *Partie C - En dimension n*

Soit F une fonction de $\mathbb{R}^n$ à valeurs dans $\mathbb{R}^n$. Une telle fonction est définie par ses $m$ fonctions composantes à valeurs réelles

$$\mathrm{F} : \boldsymbol{x} \in \mathbb{R}^n \mapsto \big( \; f_1(\boldsymbol{x}), \quad f_2(\boldsymbol{x}), \quad \ldots, \quad f_n(\boldsymbol{x}) \; \big) \in \mathbb{R}^n.$$

On suppose dans la suite que toutes les fonctions composantes $f_i$ sont de classe $\mathscr{C}^1$ sur $\mathbb{R}^n$ et on définit la matrice jacobienne de F par :

$$\mathrm{J}(\boldsymbol{x}) = \begin{bmatrix} \partial_1 f_1(\boldsymbol{x}) & \partial_2 f_1(\boldsymbol{x}) & \cdots & \partial_n f_1(\boldsymbol{x}) \\ \partial_1 f_2(\boldsymbol{x}) & \partial_2 f_2(\boldsymbol{x}) & \cdots & \partial_n f_2(\boldsymbol{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \partial_1 f_n(\boldsymbol{x}) & \partial_2 f_n(\boldsymbol{x}) & \cdots & \partial_n f_n(\boldsymbol{x}) \end{bmatrix}.$$

La méthode de Newton se généralise à plusieurs dimensions pour résoudre l'équation

$$\mathrm{F}(\boldsymbol{x}) = 0_{\mathbb{R}^n} \quad \text{d'inconnue } \boldsymbol{x} \in \mathbb{R}^n \qquad (\bullet)$$

Lorsque la matrice Jacobienne est inversible, on définit la suite $(\boldsymbol{x}_i)_{i\in\mathbb{N}}$ par la donnée de $\boldsymbol{x}_0$ et

$$\forall\, i \in \mathbb{N}, \qquad \boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \mathrm{J}(\boldsymbol{x}_i)^{-1}\mathrm{F}(\boldsymbol{x}_i)$$

où on identifie $\mathscr{M}_{n,1}(\mathbb{R})$ avec $\mathbb{R}^n$. Sous certaines conditions, on montre que la suite $(\boldsymbol{x}_i)_{i\in\mathbb{N}}$ converge vers une solution de l'équation $(\bullet)$.

*Exemple.* Considérons le système

$$\begin{cases} \cos(x) &= \sin(y) \\ \mathrm{e}^{-x} &= \cos(y) \end{cases} \quad \text{d'inconnue } (x,y) \in \mathbb{R}^2 \qquad (\star)$$

**7.** Déterminer une fonction $\mathrm{F} : \mathbb{R}^2 \to \mathbb{R}^2$ de sorte que l'équation $(\star)$ soit équivalente à $\mathrm{F}(x,y) = (0,0)$. Préciser la matrice jacobienne de F. En déduire deux programmes, un pour le calcul de $\mathrm{F}(x,y)$, un autre pour $\mathrm{J}(x,y)$.

**8.** En déduire un programme et une approximation d'une solution de $(\star)$.
*On rappelle que la commande* `np.linalg.inv(A)` *donne l'inverse de la matrice A.*

*La méthode de Newton est assez efficace et permet une convergence rapide vers une solution. Cette méthode a de nombreuses variantes et extensions. Citons par exemple, son application dans les « théorèmes K.A.M » dont l'étude de la stabilité du système solaire.*


# Indications et solutions

*Dans toute la suite, on suppose importé :*

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

### Indication de l'exercice 1 p. 3

**4.** Faire une boucle while.

**5.** Appliquer la méthode de la puissance à la matrice inverse $A^{-1}$ au lieu de A.

### Indication de l'exercice 11 p. 8

**5.** L'histogramme est proche de la courbe représentative d'une densité de la loi uniforme continue sur $[0;1]$. Pourtant $Y_n$ suit une loi de Fréchet, expliquez.

### Indication de l'exercice 12 p. 9

**1.** Reconnaître une approximation d'une fonction de répartition.

### Indication de l'exercice 17 p. 11

**3.** Un point $(x,y)$ est à l'intérieur de la cardioïde si

$$\left(x^2+y^2-x\right)^2 \leqslant x^2+y^2.$$

De plus, la probabilité d'un point pris au hasard d'être dans la cardioïde est égale à l'aide de la cardioïde divisée par 9.

### Indication de l'exercice 21 p. 12

**1.a)** Compléter le code :

```
def SimuZ(n,N):
    M=np.zeros(N)
    X=rd.randint(1,N+1,n)
    for i in range(n):
        M[ ... ]= ...
    return M
```

**1.b)** Exprimer $Y_n$ à l'aide des variables $Z_{k,n}$.

### Indication de l'exercice 23 p. 13

**1.** Compléter :

```
def Fibo(n):
    L= ...
    L[1]=  ...
    for i in range( ...  ):
        ...
    return L
```

### Indication de l'exercice 29 p. 15

Si $U \hookrightarrow \mathscr{U}(]0;1))$ alors $-\ln(U) \hookrightarrow \mathscr{E}(1)$.

### Indication de l'exercice 30 p. 15

**1.** Pour calculer $\mathbf{P}(X \leqslant x)$, appliquer la formule des probabilités totales avec le système complet d'événements $([N=n])_{n\in\mathbb{N}}$.

---

### Exercice 1 p. 3

Avant de commencer, on importe :

```
import numpy as np
```

**1.** Si les coefficients de X sont donnés par $x_i$, on sait que

$$\|X\| = \sqrt{\sum_{i=1}^{n} {x_i}^2}.$$

On en déduit le programme :

```
def norme(X):
    n=len(X)
    s=0
    for i in range(n):
        s+=X[i]**2
    return s**(1/2)

# Un petit test
X=np.array([3,0,4])
print(norme(X))
5.0
```

**2.** Le produit matriciel s'obtient par la commande `np.dot(.,.)`.

```
def puissance(X0,A,k):
    X=X0
    for i in range(k):
        X=np.dot(A,X)
        X=X/norme(X)
    return X
```

**3.** Dans un premier temps, on vérifie que la matrice A est diagonalisable avec deux valeurs propres 1 et 0.3. De plus,

$$E_1(A) = \text{Vect}\left(\begin{bmatrix}1\\1\end{bmatrix}\right) \quad \text{et} \quad E_{0.3}(A) = \text{Vect}\left(\begin{bmatrix}5\\-2\end{bmatrix}\right).$$

En particulier, un vecteur propre de norme associé à la plus grande valeur propre est

$$X_1 = \frac{\sqrt{2}}{2}\begin{bmatrix}1\\1\end{bmatrix} \quad \text{avec} \quad \frac{\sqrt{2}}{2} \simeq 0.7.$$

On teste :

```
X0=np.array([1,0])
A=np.array([[1/2,1/2],[1/5,4/5]])

print(puissance(X0,A,10))
[0.70711409 0.70709947]
```

**4.**

```
def puissance2(X0,A,lbda):
    X=X0
    Y=np.dot(A,X)
```

```
    c=0
    while norme(Y-lbda*X)>10**(-6):
        Y=np.dot(A,X)
        X=Y/norme(Y)
        c+=1
    return c
```

**5.** Si A est inversible alors 0 n'est pas valeur propre et l'équation $AX = \lambda X$ est équivalente à $\lambda^{-1}X = A^{-1}X$. On en déduit que la plus petite valeur propre de A en valeur absolue est donc la plus grande valeur propre en valeur absolue de $A^{-1}$ et surtout les espaces propres restent identiques. On peut donc appliquer la méthode de la puissance à $A^{-1}$. Ce qui donne

```
Ainv=np.linalg.inv(A)
puissance(X0,Ainv,k)
```

Optimisons un peu le code en rajoutant le cas où la matrice n'est pas inversible. Dans ce cas, on sait que 0 est la plus petite valeur propre en valeur absolue.

```
def puissance3(X0,A,k):
    [n,p]=np.shape(A)
    if np.linalg.matrix_rank(A)!=n or n!=p:
        return 0
    else :
        Ainv=np.linalg.inv(A)
        return puissance(X0,Ainv,k)
```

Et le test :

```
>>> puissance3(X0,A,10))
[0.92847669 -0.37139068]

>> X=np.array([5,-2])
>>> X/norme(X)
array([ 0.92847669, -0.37139068])
```

## Exercice 2 p. 3

**1.** En reprenant le procédé d'orthonormalisation de Schmidt, on peut poser

$$e_1 = \frac{u}{\|u\|}, \quad w = v - \frac{\langle u, v\rangle}{\|u\|^2}u, \quad e_2 = \frac{w}{\|w\|}.$$

Comme la famille $(u, v)$ est libre, $w$ ne peut être nul.

**2.a)**

```
def scal(u,v):
    n=len(u)
    s=0
    for i in range(n):
        s+=u[i]*v[i]
    return s
```

**2.b)** Précisons que les vecteurs $u$ et $v$ sont liés si et seulement si $u = 0_E$ ou $w = 0_E$. On en déduit le code :

```
def orth(u,v):

    if scal(u,u)==0:
        return 'u est nul'
    e1=u/scal(u,u)**(1/2)
    w=v-scal(e1,v)*e1
    if scal(w,w)==0:
        return  'vecteurs liés'
    else : return e1, w/scal(w,w)**(1/2)


# et le test
u=np.array([3,0,4])
v=np.array([0,4,3])
[e1,e2]=orth(u,v)
```

```
>>> e1
array([0.6, 0. , 0.8])
>>> e2
array([-0.32829174,  0.91192151,
    0.24621881])
>>> scal(e1,e2)
-8.326672684688674e-17
>>> scal(e1,e1)
1.0
>>> scal(e2,e2)
1.0000000000000002
```

## Exercice 3 p. 3

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d  #
    Fonction pour la 3D
plt.clf()
axes = plt.axes(projection="3d")

# Le pôle nord
axes.plot([0],[0],[1],'.',color='black',
    linewidth=5)
# L'équateur
t = np.linspace(0,2*np.pi, 50)
axes.plot(np.cos(t),np.sin(t),np.zeros(50),
    color='red',linewidth=1)
# Les latitudes
r=np.linspace(0,2*np.pi,20)
for i in range(20):
    x = np.cos(r[i])*np.cos(t)
    y = np.cos(r[i])*np.sin(t)
    z= np.sin(r[i])*np.ones(50)
    axes.plot(x, y, z, '--',color='grey',
        linewidth=0.5)
plt.axis('off')
#plt.show()
```

```
import numpy.random as rd
def Rotation1(theta):
    M=np.eye(3)
    M[1,1]=np.cos(theta)
    M[2,2]=M[1,1]
    M[2,1]=np.sin(theta)
    M[1,2]=-M[2,1]
    return M

def Rotation2(theta):
    M=np.eye(3)
    M[0,0]=np.cos(theta)
    M[1,1]=M[0,0]
    M[1,0]=np.sin(theta)
    M[0,1]=-M[1,0]
    return M

def Rotation3(theta):
    M=np.eye(3)
    M[0,0]=np.cos(theta)
    M[2,2]=M[0,0]
    M[2,0]=np.sin(theta)
    M[0,2]=-M[2,0]
    return M

def rotation(h):
    p=rd.random()
    theta=rd.random()*h
    if p<(1/3):
        return Rotation1(theta)
    if p>(2/3):
        return Rotation3(theta)
```

```
    return Rotation2(theta)

h=0.1
n=2000
X=np.array([[0],[0],[1]])
x=np.zeros(n+1)
y=np.zeros(n+1)
z=np.zeros(n+1)
z[0]=1
for i in range(1,n+1):
    X=np.dot(rotation(h),X)
    x[i]=X[0,0]
    y[i]=X[1,0]
    z[i]=X[2,0]
axes.plot(x, y, z)
plt.show()

h=0.05
X=np.array([[0],[0],[1]])
x=[0]
y=[0]
z=[1]
while z[-1]>0:
    X=np.dot(rotation(h),X)
    x.append(X[0,0])
    y.append(X[1,0])
    z.append(X[2,0])



theta=(2*rd.random()-1)*h
```

## Exercice 4 p. 5

**1.**

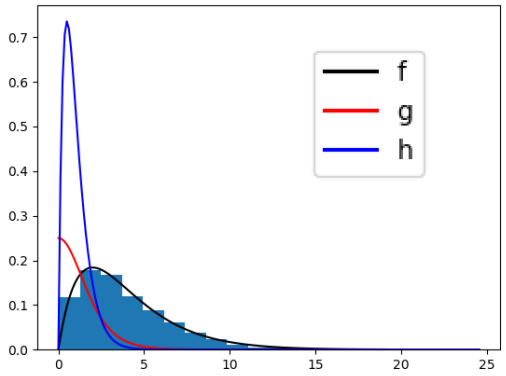
```
def chi2(n):
    S=0
    for i in range(n):
        S+=np.random.normal(0,1,1)**2
    return S
```

**2.**

```
def chi2Ech(n,m):
    Ech=0
    for i in range(n):
        Ech+=np.random.normal(0,1,m)**2
    return Ech

 # On continue avec le tracé de l'
     histogramme n=4, m=2000

plt.clf()
plt.hist(chi2Ech(4,2000),20,density=True)
plt.show()
```

**3.** Superposons à l'histogramme les différentes densités proposées. Plus la taille de l'échantillon est importante plus l'histogramme épouse la forme de la bonne densité.

```
Ech=chi2Ech(4,2000)
x=np.linspace(0,max(Ech),200)
f=x*np.exp(-x/2)/4
g=np.exp(x)/((np.exp(x)+1)**2)
h=4*x*np.exp(-2*x)
plt.clf()
plt.hist(Ech,20,density=True)
plt.plot(x,f,label='f')
plt.plot(x,g,label='g')
plt.plot(x,h,label='h')
plt.legend()
plt.show()
```




La bonne densité est $f$.

## Exercice 5 p. 6

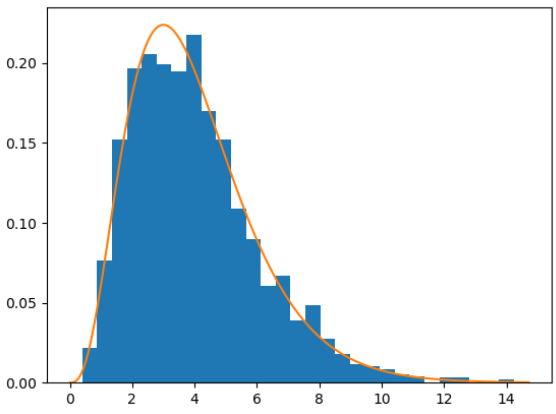
**1.** À l'aide de la fonction de répartition, on trouve une loi exponentielle de paramètre 1.

**2.** On sait que si $(X_i)_i$ sont des variables mutuellement indépendantes et de loi exponentielle de paramètre 1, alors la somme $\sum_{i=1}^{n} X_i$ suit une loi gamma de paramètre $n$.

```
def gammaSimu(n):
    X=0
    for i in range(n) :
        U=np.random.rand()
        E=-np.log(U)
        X+=E
    return X
```

**3.**

```
# Création d'un échantillon
import scipy.special as sp

def histogamma(n,m):
    Ech=np.zeros(m)
    for i in range(m):
        Ech[i]=gammaSimu(n)
    plt.hist(Ech,30,density=True)
    x=np.linspace(0,max(Ech),200)
    Gamma=sp.gamma(n)
    y=x**(n-1)*np.exp(-x)/Gamma
    plt.plot(x,y)
    plt.show()

# On teste
histogamma(4,2000)
```



*Précisons que la valeur* $\Gamma(n)$ *peut s'obtenir directement par* $\Gamma(n) = (n-1)!$.

## Exercice 6 p. 6

## Exercice 7 p. 6

**1.** On a déjà vu que $-2\ln U \hookrightarrow \mathscr{E}\left(\frac{1}{2}\right)$. Par les transformations linéaires, $2\pi V \hookrightarrow \mathscr{U}([0;2\pi])$. De plus, par le lemme des coalitions, $-2\ln U$ et $2\pi V$ sont indépendantes. D'après le résultat admis en préambule,

$$X = \sqrt{-2\ln U}\cos(2\pi V) \quad \text{et} \quad Y = \sqrt{-2\ln U}\sin(2\pi V)$$

sont indépendantes et de loi $\mathscr{N}(0;1)$.

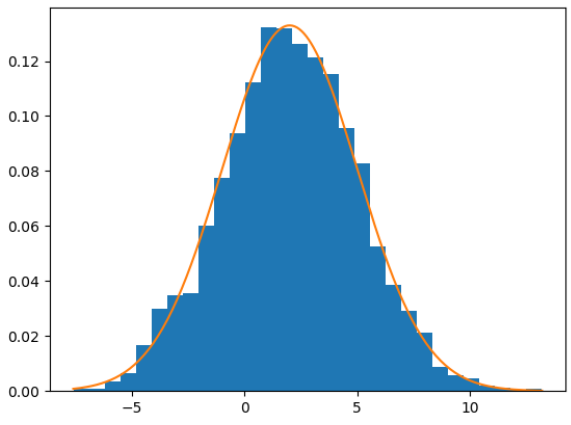
**2.**

```
    u=rd.random()
    v=rd.random()
    n=(-2*np.log(u))**(1/2)*np.cos(2*np.pi*
        v)
    return n
```

**3.** Par transformation affine, si $X \hookrightarrow \mathscr{N}(0;1)$ alors

$$\sigma X + \mu \hookrightarrow \mathscr{N}(\mu;\sigma^2).$$

```
def normaleEchGeneral(m,mu,sigma):
    return sigma*normaleEch(m)+mu

# Testons
m=5000; mu=2; sigma=3

Echa=normaleEchGeneral(2000,2,3)
x=np.linspace(min(Echa),max(Echa),200)
densite=1/(sigma * np.sqrt(2*np.pi))*np.exp
    (-0.5*((x-mu)/sigma)**2)
plt.hist(Echa,30,density=True)
plt.plot(x,densite)
plt.show()
```




**4.** Utilisons l'indépendance de X et Y pour avoir facilement le double de valeurs simulées.

```
def normaleEch(m):
    u=np.random.rand(m)
    v=np.random.rand(m)
    ech1=(-2*np.log(u))**(1/2)*np.cos(2*np.
        pi*v)
    ech2=(-2*np.log(u))**(1/2)*np.sin(2*np.
        pi*v)
    return np.concatenate([ech1,ech2])
```

## Exercice 8 p. 6

**1.** Soit $n \in \mathbb{N}^*$. Comme les variables $X_i$ sont mutuellement indépendantes et de loi $\mathscr{E}(1)$, on sait que

$$X_1 + X_2 + \ldots + X_n \hookrightarrow \gamma(n)$$
$$X_1 + X_2 + \ldots + X_n + X_{n+1} \hookrightarrow \gamma(n+1).$$

Une densité est donnée respectivement par

$$\forall t \in \mathbb{R}^+, \quad f_n(t) = \frac{t^{n-1}}{(n-1)!}e^{-t} \quad \text{et} \quad f_{n+1}(t) = \frac{t^n e^{-t}}{n!}$$

et 0 sur $\mathbb{R}^-$ (car $\Gamma(n) = (n-1)!$ et $\Gamma(n+1) = n!$). Ensuite

$$\begin{aligned}\mathbf{P}(N \geqslant n) &= \mathbf{P}(X_1 + X_2 + \ldots + X_n < \lambda)\\ &= \int_{-\infty}^{\lambda} f_n(t)\,dt\end{aligned}$$
$$\mathbf{P}(N \geqslant n) = \int_0^{\lambda} f_n(t)\,dt.$$

De même $$\mathbf{P}(N \geqslant n+1) = \int_0^{\lambda} f_{n+1}(t)\,dt.$$

De plus, notons que pour tout $t \in \mathbb{R}^+$

$$f'_{n+1}(t) = f_n(t) - f_{n+1}(t).$$

On en déduit

$$\begin{aligned}\int_0^{\lambda} f_n(t)\,dt &= \int_0^{\lambda} f'_{n+1}(t) + f_{n+1}(t)\,dt\\ &= \int_0^{\lambda} f'_{n+1}(t)\,dt + \int_0^{\lambda} f_{n+1}(t)\,dt\\ &= \left[f_{n+1}(t)\right]_0^{\lambda} + \int_0^{\lambda} f_{n+1}(t)\,dt\\ &= \frac{\lambda^n}{n!}e^{-\lambda} + \int_0^{\lambda} f_{n+1}(t)\,dt.\end{aligned}$$

D'où $$\mathbf{P}(N \geqslant \lambda) = \frac{\lambda^n}{n!}e^{-\lambda} + \mathbf{P}(N \geqslant n+1).$$

**2.** La variable N est à valeurs dans $\mathbb{N}$. D'après ce qui précède, pour tout $n \in \mathbb{N}$,

$$\mathbf{P}(N = n) = \mathbf{P}(N \geqslant n) - \mathbf{P}(N \geqslant n+1) = \frac{\lambda^n}{n!}e^{-\lambda}.$$

Par définition $N \hookrightarrow \mathscr{P}(\lambda)$.

**3.** Rappelons que si U suit une loi uniforme sur $]0;1]$ alors on a vu que

$$-\lambda\ln(U) \hookrightarrow \mathscr{E}(\lambda).$$

Ainsi pour tout $i \in \mathbb{N}^*$, $X_i = -\lambda\ln(U_i) \hookrightarrow \mathscr{E}(\lambda)$. Les variables $X_i$ restent indépendantes par le lemme des coalitions. La condition $[U_1 < e^{-\lambda}]$ devient $[X_1 > \lambda]$ et pour $n \in \mathbb{N}^*$

$$\prod_{i=1}^{n+1} U_i \leqslant e^{-\lambda}.$$

correspond à $$\sum_{i=1}^{n+1} X_i \geqslant \lambda.$$

On retrouve ainsi pour tout $n \in \mathbb{N}$

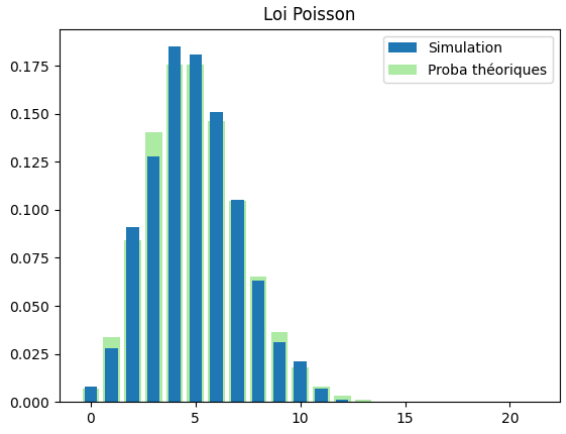
$$\begin{aligned}&[X = n]\\ &= [X_1 + \cdots + X_n < \lambda] \cap [X_1 + \cdots + X_n + X_{n+1} \geqslant \lambda].\end{aligned}$$

On sait alors que X suit une loi de Poisson de paramètre $\lambda$.

**4.**

```
def simulPoisson(lbda):
    n=0
    u=rd.random()
    S=-np.log(u)
    while S<lbda:
        u=rd.random()
        S+=-np.log(u)
        n+=1
    return n
```

*Pour être sûr du programme, on crée l'histogramme d'un échantillon que l'on compare avec le diagramme en bâtons des probabilités théoriques. Par exemple pour $\lambda = 5$ avec un échantillon de $m = 1000$, on a obtenu :*




## Exercice 9

p. 7

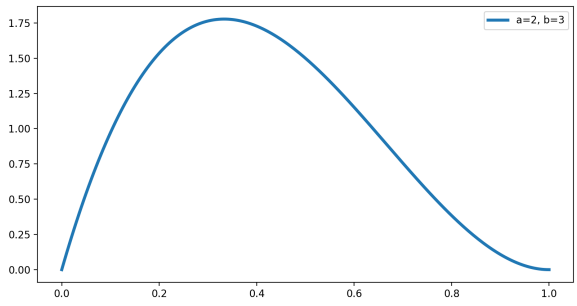
**1.a)** Précisons que par comparaison à des intégrale de Riemann, $B(\alpha,\beta)$ est bien définie.
On vérifie que $f_{\alpha,\beta}$ est continue sur $\mathbb{R}\setminus\{0;1\}$, positive et par construction

$$\int_{-\infty}^{+\infty} f_{\alpha,\beta}(t)\,\mathrm{d}t = 1.$$

**1.b)**

```
import scipy.special as sp

def BetaCourbe(a,b):
    x=np.linspace(0,1,200)
    y=x**(a-1)*(1-x)**(b-1)/sp.beta(a, b)
    plt.plot(x,y,linewidth=3,label='a='+str
        (a)+", "+'b='+str(b))
    plt.legend()
    plt.show()
```

Test avec $a = 2$, $b = 3$ :




**2.a)**

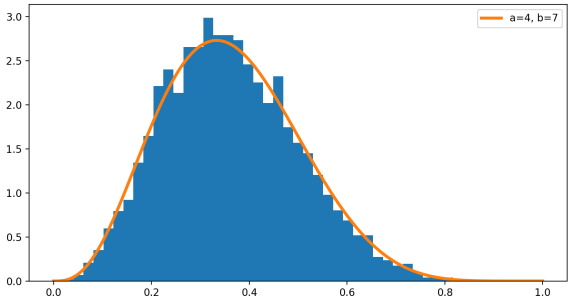
```
def SimuYk(n,k):
    X=rd.random(n)
    X=np.sort(X)
    return X[k-1]
    # attention au décalage d'indice
```

**2.b)**

```
def testBeta(n,k):
    m=5000
    for i in range(m):
        Ech[i]=SimuYk(n,k)
    inter=np.linspace(0,1,50)
    plt.hist(Ech, bins=inter, density=True)
    BetaCourbe(k,n-k+1)
```

Après plusieurs tests, on constate l'adéquation en l'histogramme et la densité $f_{k,n+1-k}$. Par exemple :

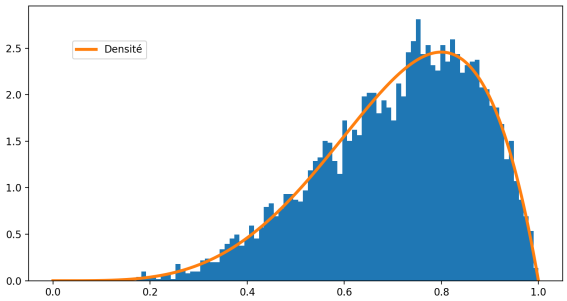
```
>>> testBeta(10,4)
```




Ce qui permet de conjecturer que $Y_k$ suit une loi bêta de paramètre $(k, n+1-k)$.

**3.**

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

def f(a,b,t):
    return t**(a-1)*(1-t)**(b-1)
    # la constante B(a,b) n'est pas utile
        ici
    # car elle apparaît des deux cotés dans
    # l'inégalité y>f(x)

def simuBeta(a,b):

    x=rd.random()
    y=rd.random()

    while y>f(a,b,x):
        x=rd.random()
        y=rd.random()

    return x

import scipy.special as sp
a=5
b=2
plt.clf()
m=5000 # taille échantillon
Ech=np.zeros(m)
for i in range(m):
    Ech[i]=simuBeta(a,b)
inter=np.linspace(0,1,100)
plt.hist(Ech, bins=inter, density=True)
# Création de l'histogramme
x=np.linspace(0,1,200)
y=x**(a-1)*(1-x)**(b-1)/sp.beta(a, b)
plt.plot(x,y,linewidth=3,label="Densité")
plt.legend()
plt.show()
```




**4.**

```
# loi beta

import numpy as np
import numpy.random as rd

def simu(a,b):
    u=rd.random()
    v=rd.random()
    z=u**(1/a)+v**(1/b)
    while z>1:
        u=rd.random()
        v=rd.random()
        z=u**(1/a)+v**(1/b)
    return u**(1/a)/z

N=5000
Ech=np.zeros(N)
for i in range(N):
    Ech[i]=simu(2,2)

plt.hist(Ech,30,density=True)
plt.show()
```

**4.a)**

```
def MoyVarEmpiriques(a,b):
    N=5000
    s=0
    v=0
    for i in range(N):
        X=simuBeta(a,b)
        s+=X
        v+=X**2
    moy=s/N
    var=v/N-moy**2
    return moy,var
```

**4.b)** Notons que pour tout $t \in \mathbb{R}$

$$\begin{aligned} t f_{\alpha,\beta}(t) &= \frac{1}{B(\alpha,\beta)} t^{\alpha}(1-t)^{\beta-1} \\ &= \frac{B(\alpha+1,\beta)}{B(\alpha,\beta)} f_{\alpha+1,\beta}(t). \end{aligned}$$

On en déduit la convergence (absolue) de $\int_{-\infty}^{+\infty} t f_{\alpha,\beta}(t)\,dt$ avec

$$\begin{aligned} \int_{-\infty}^{+\infty} t f_{\alpha,\beta}(t)\,dt &= \frac{B(\alpha+1,\beta)}{B(\alpha,\beta)} \int_{-\infty}^{+\infty} f_{\alpha+1,\beta}(t)\,dt \\ &= \frac{B(\alpha+1,\beta)}{B(\alpha,\beta)}. \end{aligned}$$

Ensuite, on a

$$\begin{aligned} B(\alpha+1,\beta) &= \frac{\Gamma(\alpha+1)\Gamma(\beta)}{\Gamma(\alpha+\beta+1)} = \frac{\alpha}{\alpha+\beta}\frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \\ &= \frac{\alpha}{\alpha+\beta} B(\alpha,\beta). \end{aligned}$$

On en déduit l'existence de l'espérance avec

$$\mathbf{E}(X) = \frac{\alpha}{\alpha+\beta}$$

ou X suit une loi bêta de paramètre $(\alpha,\beta)$. En appliquant la formule de Koenig-Huygens et, on montre

$$\mathbf{V}(X) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}.$$

*Remarque. Pour $a = b = 1$, on trouve $\mathbf{E}(X) = 1/2$ et $\mathbf{V}(X) = 1/12 \simeq 0.833$. On peut donc tester le programme précédent*

```
>>> MoyVarEmpiriques(1,1)
(0.4989861288681462, 0.08066192251361898)
```

**5.**

```
def test(a,b):
    x,v=MoyVarEmpiriques(a,b)
    return x*(x*(1-x)/v-1),(1-x)*(x*(1-x)/v
        -1)
```

Quelques tests :

```
>>> test(2,3)
(2.009373415933195, 2.9809109796171507)

>>> test(4,1)
(3.976530467568522, 1.0050926883236162)

>>> test(5,3)
(4.963890366699264, 2.950227335140227)
```

À partir d'un échantillon, on peut obtenir une approximation des paramètres $\alpha$, $\beta$.

## Exercice 10 p. 8

**1.a)**

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

def simuZ():
    x=rd.normal(0,1)
    y=rd.normal(0,1)
    return x/y

# deuxième solution
def simuZ():
    X=rd.normal(0,1,2)
    return X[0]/X[1]
```

**1.b)**

```
def echZ():
    m=10000
    Ech=np.zeros(m)
    for i in range(m):
        Ech[i]=simuZ()
    return Ech
```

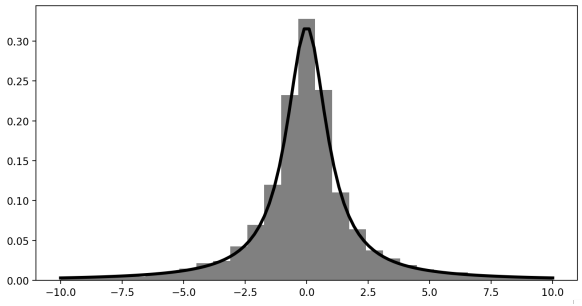
On teste :

```
>>> np.mean(echZ())
1.5453371520266093

>>> np.mean(echZ())
0.7474043707779702

>>> np.mean(echZ())
-5.419777150119312

>>> np.mean(echZ())
-0.1389511116794706
```

On constate qu'il ne semble pas avoir de valeur fixe. Cela suggère que Z n'a pas d'espérance.

**1.c)**

```
absc=np.linspace(-10,10,100)
ordo=1/(1+absc**2)/np.pi
plt.plot(absc,ordo,linewidth=3,color='k')


plt.hist(echZ(),np.linspace(-10,10,30),
    density=True,color='grey')
plt.show()
```



L'histogramme semble épouser la forme de la courbe. Cela suggère que Z est une variable aléatoire à densité et une densité est donnée par la fonction $f$. En anticipant sur la suite

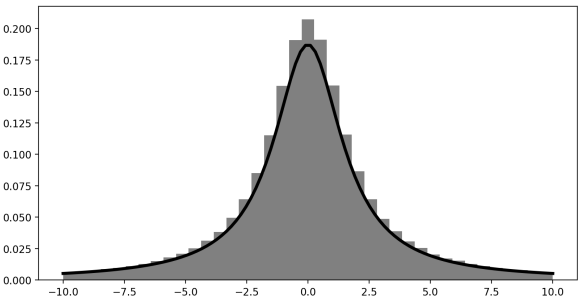
$$Z \hookrightarrow \mathscr{C}(1).$$

**2.a)** Si $X \hookrightarrow \mathscr{C}(1)$, on vérifie que $\lambda X \hookrightarrow \mathscr{C}(\lambda)$.

```
def Cauchy(lbda):
    return lbda*simuZ()
```

**2.b)**

```
def echC(lbda,mu):
    m=10000
    Ech=np.zeros(m)
    for i in range(m):
        Ech[i]=Cauchy(lbda)+Cauchy(mu)
    return Ech

plt.clf()
lbda=0.8
mu=0.5
absc=np.linspace(-10,10,100)
lm=lbda+mu
ordo=lm/(lm**2+absc**2)/np.pi
plt.plot(absc,ordo,linewidth=3,color='k')


plt.hist(echC(lbda,mu),np.linspace
    (-10,10,30),density=True,color='grey')
plt.show()
```



## Exercice 11 p. 8

**1.** La variable U est presque sûrement à valeurs dans $]0;1[$, $s(-\ln U)^{-1}$ est à valeurs dans $\mathbb{R}_*^+$. Pour $x \in \mathbb{R}^-$

$$\mathbf{P}\left(-s(\ln U)^{-1} \leqslant x\right) = 0.$$

Pour $x \in \mathbb{R}_*^+$

$$\begin{aligned}\mathbf{P}\left(-s(\ln U)^{-1} \leqslant x\right) &= \mathbf{P}\left(-\ln(U)^{-1} \leqslant \frac{x}{s}\right)\\ &= \mathbf{P}(U \leqslant e^{-s/x}) \quad \text{par croissance stricte}\\ &= F_U\left(e^{-s/x}\right).\end{aligned}$$

Comme $e^{-s/x} \in [0;1]$, il vient

$$\mathbf{P}\left(-s(\ln U)^{-1} \leqslant x\right) = e^{-s/x}.$$

On reconnaît la fonction de répartition d'une loi de Fréchet de paramètre $s$. Comme la fonction de répartition caractérise la loi

$$-s(\ln U)^{-1} \hookrightarrow \mathscr{F}(s).$$

**2.** Une densité de $\mathscr{F}(s)$ est donnée par $f(x) = 0$ sur $\mathbb{R}^-$ et

$$\forall x \in \mathbb{R}_*^+, \qquad f(x) = \frac{s}{x^2} e^{-s/x}.$$

Or $$xf(x) = \frac{1}{x} e^{-s/x} \underset{x\to+\infty}{\sim} \frac{1}{x}.$$

Par le critère d'équivalence $\displaystyle\int_1^{+\infty} xf(x)\,dx$ diverge au voisinage de $+\infty$, et il n'y a pas d'espérance et donc pas de variance.

**3.**

```
def frechet(s):
    u=np.random.rand()
    return -s*np.log(u)**(-1)
```

**4.**

```
def MaxFrechet(s,n,m):
    Ech=np.zeros(m)
    for i in range(m):
        u=np.random.rand(n)
        y=-s*np.log(u)**(-1)
        Ech[i]=max(y)
    return Ech
```

**5.** Posons $Z_n = e^{-(Y_n/(sn))^{-1}} = F_{sn}(Y_n)$. Le programme renvoie l'histogramme d'un échantillon de taille 5000 associé à la variable $Z_n$. On conjecture que

$$Z_n \hookrightarrow \mathscr{U}[0;1].$$

car l'histogramme est très proche d'une densité de cette loi. Si tel est bien est le cas, en utilisant la question 1

$$Y_n = sn(-\ln(Z_n))^{-1} \hookrightarrow \mathscr{F}(sn).$$

*Résultat que l'on prouve en passant par les fonctions de répartition.*

**6.** Ce code illustre le fait que la loi de Fréchet n'a pas d'espérance car les moyennes empiriques ne semblent pas se stabiliser/converger avec l'augmentation de a taille de l'échantillon.

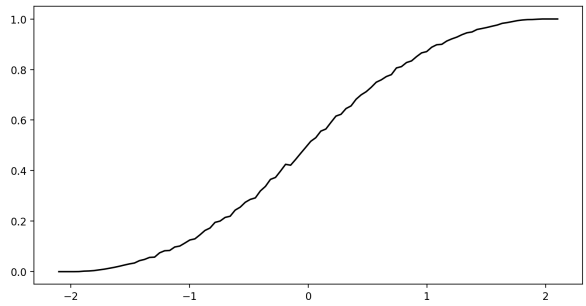
## Exercice 12 p. 9

**1.** La fonction $g$ prend en argument un réel $x$ et renvoie une approximation de la probabilité

$$\mathbf{P}(X+Y \leqslant x)$$

où X et Y sont deux variables aléatoires de lois uniformes sur $[0;1]$. Dit autrement, $g(x)$ donne une approximation de $F(x)$ où F est la fonction de répartition de $X+Y$.

**2.** La fonction `graphe()` renvoie une approximation de la courbe de la fonction de répartition F.



En reprenant l'exemple du cours, on vérifie qu'une densité est donnée par un produit de convolution :

$$f(x)=\begin{cases}0 & \text{si} \quad x\notin[-2;2]\\ (2+x)/4 & \text{si} \quad x\in[-2;0]\\ (2-x)/4 & \text{si} \quad x\in[0;2].\end{cases}$$

Et la fonction de répartition s'obtient par intégration

$$\mathrm{F}(x)=\begin{cases}0 & \text{si} \quad x\leqslant -2\\ (x+2)^2/8 & \text{si} \quad x\in[0;1]\\ 1-(1-x)^2/8 & \text{si} \quad x\in[1;2]\\ 1 & \text{si} \quad x\geqslant 1.\end{cases}$$

## Exercice 13 p. 9

**1.a)**

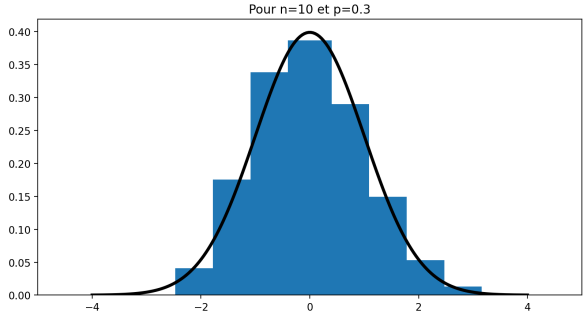
**1.b)**

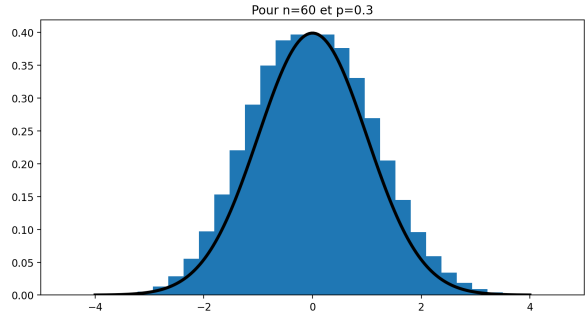
```
def batons(n,p):
    val=(np.linspace(0,n,n+1)-n*p)/np.sqrt(
        n*p*(1-p))

    proba=np.zeros(n+1)
    for k in range(n+1):
        proba[k]=sp.binom(n,k)*p**k*(1-p)
            **(n-k)

    plt.bar(val,np.sqrt(n*p*(1-p))*proba)
    plt.show()
```

**1.c)** On rajoute au code précédent :

```
# limites en abscisses
plt.xlim(-5,5)

#tracé de la courbe de la densité
x=np.linspace(-4,4,200)
y=np.exp(-x**2/2)/(2*np.pi)**(1/2)
plt.plot(x,y,color='k',linewidth=3)
```

**1.d)** On constate que plus $n$ est grand plus le diagramme en bâtons épouse la courbe de la densité. C'est bien en accord avec le théorème de la limite centrée.



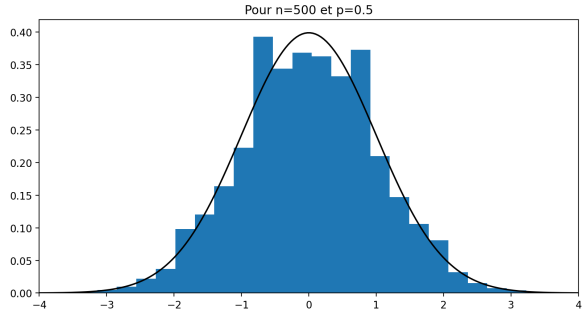

**2.a)**

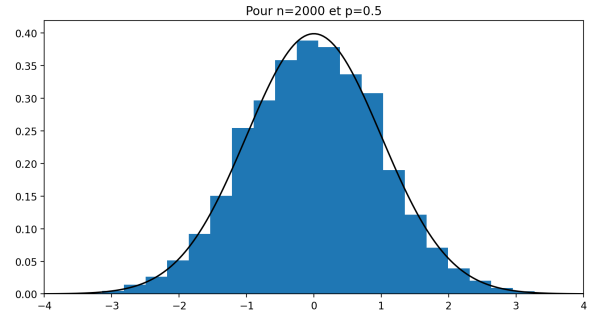
```
def simuSn(n,p):
    X=rd.binomial(n,p)
    S=(X-n*p)/np.sqrt(n*p*(1-p))
    return S
```

**2.b)**

```
def histo(n,p):

    # Création d'un échantillon
    m=100000
    Ech=np.zeros(m)
    for i in range(m):
        Ech[i]=simuSn(n,p)

    # Tracé de l'histogramme
    plt.hist(Ech,30,density=True)

    # Tracé de la courbe de la densité
    plt.xlim(-4,4)
    x=np.linspace(-4,4,200)
    y=np.exp(-x**2/2)/(2*np.pi)**(1/2)

    plt.show()
```

**2.c)**




## Exercice 14 p. 10

**1.a)**

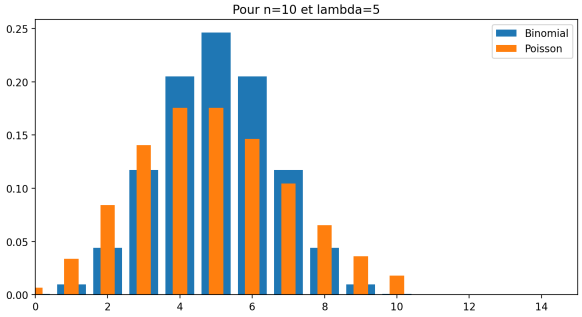
```
def BatonBin(n,p):
    val=np.linspace(0,n,n+1)
    proba=np.zeros(n+1)
```

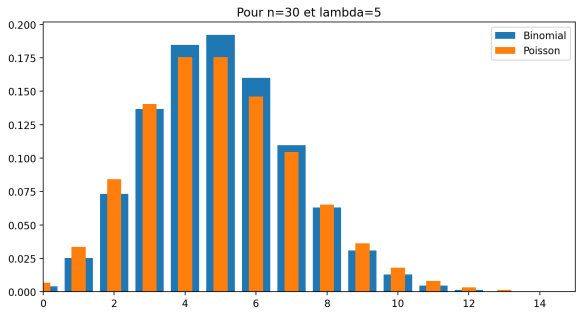
```
    for k in range(n+1):
        proba[k]=sp.binom(n,k)*p**k*(1-p)
            **(n-k)
    plt.bar(val,proba,width=0.8,label='
        Binomial')
    plt.show()
```

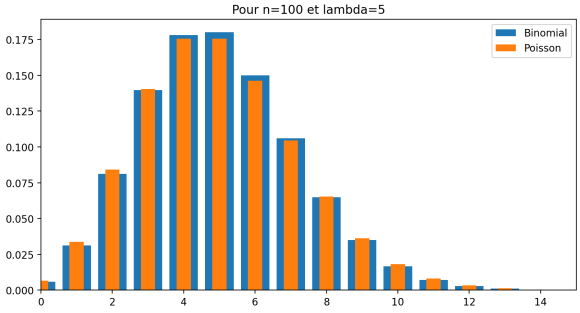
**1.b)**

```
def BatonPoi(lbda,n) :
    proba=np.zeros(n+1)
    proba[0]=np.exp(-lbda)

    for i in range(1,n+1) :
        proba[i]=proba[i-1]*lbda/(i)

    ind=np.linspace(0,n,n+1)
    plt.bar(ind,proba)
```

**1.c)**



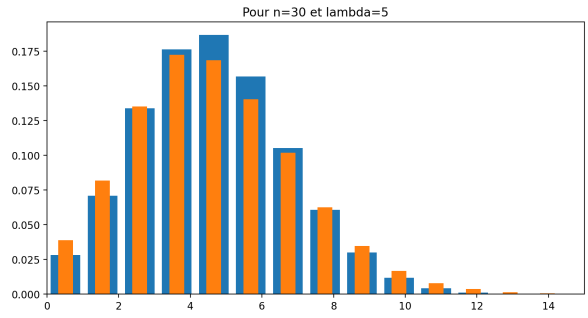

**2.**

- *Version courte*

```
n=30
lbda=5
m=50000
EchBinomiale=rd.binomial(n,lbda/n,m)
EchPoisson=rd.poisson(lbda,m)
classe=np.linspace(0,n,30)
plt.hist(EchBinomiale,classe,density=True)
plt.hist(EchPoisson,classe,density=True)

plt.show()
```

*Version longue en soignant l'affichage*

```
plt.clf()
n=30
lbda=5
plt.xlim(0,15)
m=50000
EchBinomiale=rd.binomial(n,lbda/n,m)
EchPoisson=rd.poisson(lbda,m)
classe=np.linspace(0,n,30)
plt.hist(EchBinomiale,classe,rwidth=0.8,
    density=True)
plt.hist(EchPoisson,classe,rwidth=0.4,
    density=True)
titre='Pour n='+str(n)+' et lambda='+str(
    lbda)
plt.title(titre)
plt.show()
```

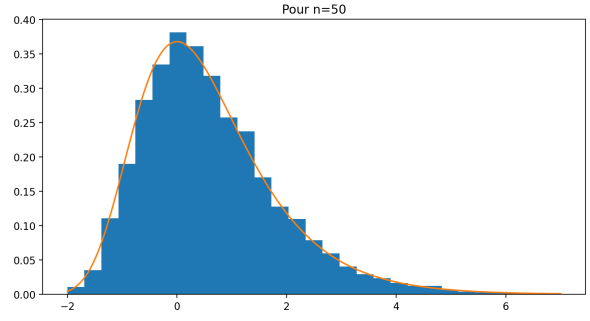



## Exercice 15 p. 10

**1,2.**

```
def simuZn(n):
    return max(rd.exponential(1,n)-np.log(n
        ))

def test(n):
    m=10000
    Ech=np.zeros(m)
    for i in range(m):
        Ech[i]=simuZn(n)


    a=-2;   b=7
    classe=np.linspace(a,b,30)
    plt.hist(Ech,classe,density=True)

    x=np.linspace(a,b,200)
    y=np.exp(-x-np.exp(-x))
    plt.plot(x,y)
    plt.show()
```

Plus $n$ est grand, plus l'histogramme s'approche de la densité. On conjecture que $(Z_n)_n$ converge en loi vers une variable X qui suit une loi de Gumbel.



## Exercice 16 p. 10

**1.** Pour $n = 12$, on a une petite simplification avec l'écart-type :

$$X_{12}^* = \sqrt{12}(\overline{X_{12}} - 1/2)/(1/\sqrt{12}) = 12(\overline{X_{12}} - 1/2).$$

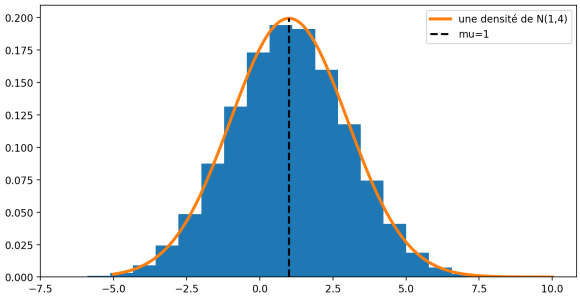
```
def douze():
    X=rd.random(12)
    e=1/2
    s=np.sqrt(1/12)
    return 12*(np.mean(X)-e)
```

**2.** Par les règles de transformation affine des lois normales. Si $X \hookrightarrow \mathcal{N}(0;1)$ alors $\sigma X + \mu \hookrightarrow \mathcal{N}(\mu;\sigma^2)$.

```
def Normal(mu,sigma,m):
    Ech=np.zeros(m)
    for i in range(m):
        X=douze()
        Ech[i]=sigma*X+mu
    return Ech
```

**3.**

```
mu=1; sigma=2; m=100000
plt.hist(Normal(mu,sigma,m),20,density=True
    )
x=np.linspace(-5,10,200)
y=np.exp(-(x-mu)**2/(2*sigma**2))/(np.sqrt
    (2*np.pi)*sigma)
plt.plot(x,y)

## On peut rajouter la moyenne
plt.plot([mu,mu],[0,1/np.sqrt(2*np.pi)*
    sigma],'k--')
plt.show()
```




## Exercice 17 p. 11

**1.** Si $U \hookrightarrow \mathcal{U}([0;1])$ alors on sait que

$$3U - \frac{1}{2} \hookrightarrow \mathcal{U}([-0.5;2.5])$$

et $$3U - \frac{3}{2} \hookrightarrow \mathcal{U}([-1.5;1.5])$$

```
def point():
    x=3*rd.random()-0.5
    y=3*rd.random()-1.5
    return x,y
```

**2.**

```
def dedans():
    x,y=point()
    if (x**2+y**2-x)**2<(x**2+y**2):
        print('In !')
    else : print('Out !')
```

**3.** Un point $(x, y)$ est à l'intérieur de la cardioïde si

$$\left(x^2+y^2-x\right)^2 \leqslant x^2+y^2.$$

On simule donc un grand nombre de tirages d'un point dans le carré et on compte le nombre de points à l'intérieur de la cardioïde. La fréquence donne alors une approximation de la probabilité. Or la probabilité vaut

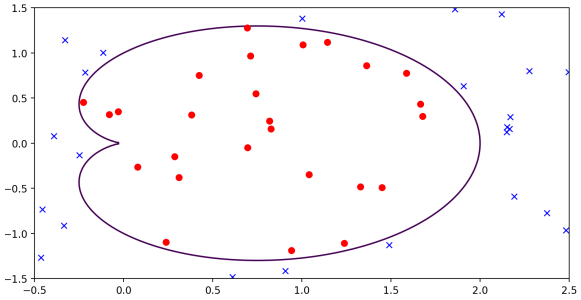
$$p = \frac{\text{Aire cardioïde}}{\text{Aire carré}} = \frac{\text{Aire cardioïde}}{9}.$$

Ainsi, 9 fois la fréquence empirique donne une approximation de l'aire.

```
def dedans01():
    x,y=point()
    if (x**2+y**2-x)**2<(x**2+y**2):
        return 1
    else : return 0

s=0
m=50000
for i in range(m):
    s+=dedans01()
print(9*s/m)
4.69782

print(3*np.pi/2)
4.71238898038469
# valeur théorique
```

**4.**

```
def f(x,y):
    return (x**2+y**2-x)**2-(x**2+y**2)

x=np.linspace(-0.5,2.5,200)
y=np.linspace(-1.5,1.5,200)
X , Y = np.meshgrid(x,y)

graphe = plt.contour(X,Y,f(X,Y),[0])

# On affiche aussi quelques tirages de
    points.
for iter in range(50):
    a,b=point()
    if (a**2+b**2-a)**2<(a**2+b**2):
        plt.plot(a,b,'o')
    else : plt.plot(a,b,'x')
plt.show()
```



## Exercice 18 p. 12

**1.** Soit $m \in \mathbb{R}$. Par linéarité de l'espérance

$$\begin{aligned}\mathbf{E}\left((X-m)^2\right) &= \mathbf{E}\left(X^2 - 2mX + m^2\right)\\ &= \mathbf{E}\left(X^2\right) - 2m\mathbf{E}(X) + m^2\\ &= \left(\mathbf{E}\left(X^2\right) - \mathbf{E}(X)^2\right) + \mathbf{E}(X)^2 - 2m\mathbf{E}(X) + m^2\\ &= \mathbf{V}(X) + \left(\mathbf{E}(X) - m\right)^2.\end{aligned}$$

$$\mathbf{E}\left((X-m)^2\right) \geqslant \mathbf{V}(X).$$

Le minimum est $\mathbf{V}(X)$ et atteint uniquement pour

$$m = \mathbf{E}(X).$$

- Posons $\tilde{m} = \dfrac{a+b}{2}$. Comme $a \leqslant X \leqslant b$, on a

$$-\frac{b-a}{2} \leqslant X - \tilde{m} \leqslant \frac{b-a}{2}.$$

Puis $$\left(X-\tilde{m}\right)^2 \leqslant \frac{(b-a)^2}{4}.$$

Par croissance de l'espérance

$$\mathbf{E}\left((X-\tilde{m})^2\right) \leqslant \frac{(b-a)^2}{4}.$$

Mais, la variance est le minimum. Nécessairement

$$\mathbf{V}(X) \leqslant \frac{(b-a)^2}{4}.$$

**2.** C'est une conséquence de l'inégalité de Bienaymé-Tchebychev appliquée à

$$Z = \frac{1}{n}\sum_{k=1}^{n} g\left(U_k\right).$$

En effet

$$\begin{aligned}\mathbf{E}(Z) &= \frac{1}{n}\sum_{k=1}^{n} \mathbf{E}\left(g\left(U_k\right)\right) = \mathbf{E}\left(g\left(U_1\right)\right) \text{ (même loi)}\\ &= \int_0^1 g(t)\,\mathrm{d}t.\end{aligned}$$

et

$$\begin{aligned}\mathbf{V}(Z) &= \frac{1}{n^2}\mathbf{V}\left(\sum_{k=1}^{n} g\left(U_k\right)\right)\\ &= \frac{1}{n^2}\sum_{k=1}^{n} \mathbf{V}\left(g\left(U_k\right)\right) \quad \text{(indépendance)}\\ \mathbf{V}(Z) &= \frac{1}{n}\mathbf{V}\left(g\left(U_1\right)\right) \quad \text{(même loi).}\end{aligned}$$

Or $g(U_1)$ est à valeurs dans $[a,b]$, la question 1 donne

$$\mathbf{V}\left(g\left(U_1\right)\right) \leqslant \frac{(b-a)^2}{4}.$$

D'où le résultat.

**3.** On vérifie que $I = \left[4\arctan(t)\right]_0^1 = \pi$.

```
def approxPi(n):
    s=0
    for i in range(n):
        u=rd.random()
        s+=1/(1+u**2)
    return 4*s/m

>>> approxPi(10000)
3.1414778584480114
```

**4.** On cherche $n$ tel que

$$\mathbf{P}\left(\left|\pi - \frac{1}{n}\sum_{k=1}^{n} g\left(U_k\right)\right| \geqslant 10^{-3}\right) \leqslant 1 - 95\%.$$

D'après la relation (•), il suffit que

$$5\% \geqslant \frac{1}{4n\varepsilon^2} \quad \text{avec } a=1, b=0, \varepsilon = 10^{-3}.$$

D'où $$n \geqslant \frac{100}{\varepsilon^2 - 45} = \frac{5}{\varepsilon^2} = 5\cdot 10^6.$$

La méthode est assez peu efficace en comparaison avec les sommes de Riemann.

*Remarque. La méthode devient intéressante numériquement en dimension plus grande que 1.*

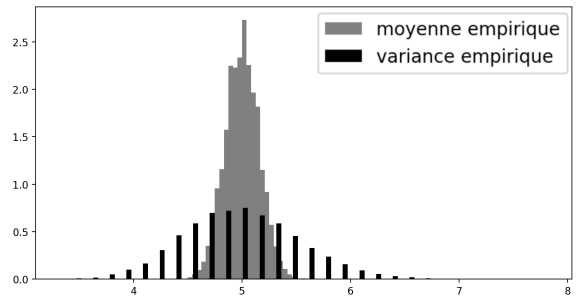
## Exercice 19 p. 12

**1.** Par exemple

```
def Estimateurs(n,lbda):
    X=rd.poisson(lbda,n)
    Xbar=np.mean(X)
    S=0
    for i in range(n):
        S+=(X[i]-Xbar)**2
    Sigma=(S/(n-1))
    return [Xbar,Sigma]

# On peut aussi la commande np.std(X) pour
    avoir l'écart-type empirique
```

**2.**

```
def comparaisonPoisson(lbda,n):
    m=20000
    Xbar=np.zeros(m)
    Sigma=np.zeros(m)
    for i in range(m):
        S=Estimateurs(n,lbda)
        Xbar[i]=float(S[0])
        Sigma[i]=float(S[1])
    plt.clf()
    plt.hist(Xbar,30,density=True,color='
        grey')
    plt.hist(Sigma,30,density=True, color='
        k',rwidth=0.3)
    plt.show()

comparaisonPoisson(5,200)
```




Après plusieurs essais, on constate que la dispersion de l'échantillon par rapport à $\lambda = 5$ est plus faible pour la moyenne empirique $\overline{X_n}$ que la variance $\sigma_n^2$. On conjecture que l'estimateur $\overline{X_n}$ serait meilleur que $\sigma_n^2$.

## Exercice 20 p. 12

**1.**

```
def SimuAB(n,theta):
    s=0
    t=0
    for i in range(n):
        x=rd.poisson(theta)
        s+=x
        if x==0:
            t+=1
    A=(1-1/n)**s
    B=t/n
    return A,B

# un test :

>>> SimuAB(10,1)
(0.3874204890000001, 0.4)
```

**2.**

```
def ApproxEsp(n):
    m=10000
    EchA=np.zeros(m)
    EchB=np.zeros(m)
    for i in range(m):
        EchA[i],EchB[i]=SimuAB(n,1)
    MoyA=np.mean(EchA)
    MoyB=np.mean(EchB)
    para=np.exp(-1)
    print('Esp A',MoyA,'Esp B',MoyB)
```

On teste :

```
>>> ApproxEsp(10)
Esp A 0.36887920980929095 Esp B 0.36823
```

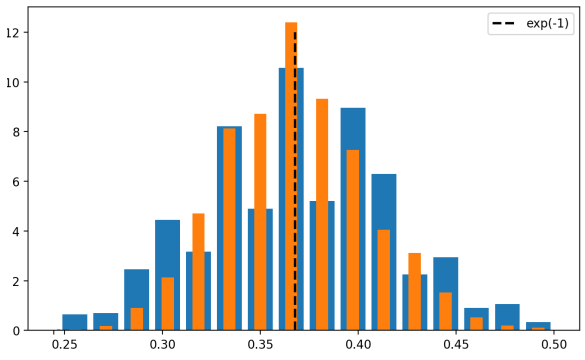
Valeurs à comparer au paramètre que l'on cherche à estimer :

```
>>> np.exp(-1)
0.36787944117144233
```

La différence est proche de 0. Il semble que les deux estimateurs soient sans biais.

**3.**

```
def Histo(n):
    m=20000
    EchA=np.zeros(m)
    EchB=np.zeros(m)
    for i in range(m):
        EchA[i],EchB[i]=SimuAB(n,1)
    classe=np.linspace(0.2,0.5,20)
    plt.hist(EchB,classe,density=True,
        rwidth=0.8)
    plt.hist(EchA,classe,density=True,
        rwidth=0.4)

    para=np.exp(-1)
    plt.plot([para,para],[0,12],'k--',
        linewidth=2,label='exp(-1)')
    plt.legend()
    plt.show()
```




## Exercice 21 

**1.a)**

```
def SimuZ(n,N):
    M=np.zeros(N)
    X=rd.randint(1,N+1,n)
# X est une matrice ligne de n réalisations
    d'une loi uniforme sur [1,N]
    for i in range(n):
        M[X[i]-1]=1
    return M
```

**1.b)** À partir de la relation $Y_n = \sum_{k=1}^{N} Z_{k,n}$, on écrit le code :

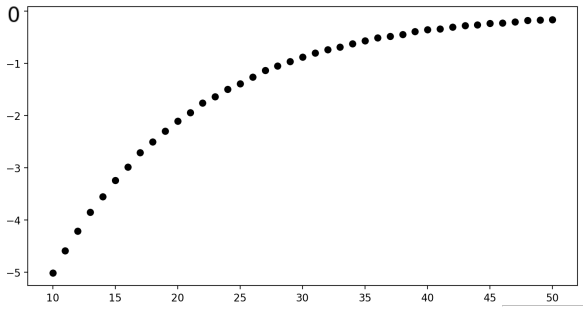
```
def SimuYn(n,N):
    return np.sum(SimuZ(n,N))
```

**1.c)**

```
def esperanceYn(n,N):
    m=5000
    Ech=np.zeros(m)
    for i in range(m):
        Ech[i]=SimuYn(n,N)
    return np.mean(Ech)
```

Pour tester, fixons une valeur pour N et calculons quelques termes de la suite $(b_n)_n$.

```
N=12
Bn=np.zeros(41)
for i in range(10,51):
    Bn[i-10]=esperanceYn(i,N)-N
x=np.linspace(10,50,41)
plt.plot(x,Bn,'ko')
plt.show()
```




On constate que la suite $(b_n)_n$ semble converger vers 0. L'estimateur serait asymptotiquement sans biais.

**2.a)** Soit $k \in [\![1;N]\!]$. La variable $Z_{k,n}$ suit une loi de Bernoulli. Son espérance est son paramètre. Donc

$$\mathbf{E}(Z_{k,n}) = \mathbf{P}(Z_{k,n}=1) = 1 - \mathbf{P}(Z_{k,n}=0).$$

De plus, par indépendance des $X_i$.

$$\begin{aligned}\mathbf{P}(Z_{k,n}=0) &= \mathbf{P}\left(\bigcap_{i=1}^{n}[X_i \neq k]\right)\\ &= \prod_{i=1}^{n}\mathbf{P}(X_i \neq k)\\ &= \prod_{i=1}^{n}\left(1-\frac{1}{N}\right) = \left(1-\frac{1}{N}\right)^n.\end{aligned}$$

Ce qui conclut.

**2.b)** Par linéarité de l'espérance :

$$\mathbf{E}(Y_n) = \sum_{k=1}^{N}\mathbf{E}(Z_{k,n}) = N\left(1-\left(1-\frac{1}{N}\right)^n\right).$$

On en déduit le biais de $Y_n$

$$b(Y_n) = \mathbf{E}(Y_n) - N = \left(1 - \frac{1}{N}\right)^n \underset{n\to\infty}{\longrightarrow} 0.$$

**2.c)** On a

$$\begin{aligned}\mathbf{V}(Y_n) &= \mathbf{V}\left(\sum_{k=1}^{N} Z_{k,n}\right)\\ &= \mathrm{Cov}\left(\sum_{k=1}^{N} Z_{k,n}, \sum_{\ell=1}^{n} Z_{\ell,n}\right)\\ \mathbf{V}(Y_n) &= \sum_{k,\ell=1}^{N} \mathrm{Cov}(Z_{k,n}, Z_{\ell,n}).\end{aligned}$$

Par les inégalités triangulaire et de Cauchy-Schwarz

$$\begin{aligned}\mathbf{V}(Y_n) &\leqslant \sum_{k,\ell=1}^{N} \sqrt{\mathbf{V}(Z_{k,n})\mathbf{V}(Z_{\ell,n})}\\ &\leqslant N^2 \mathbf{V}(Z_{1,n})\end{aligned}$$

car les variables ont même loi. Or, $Z_{1,n}$ suit une loi de Bernoulli, on connaît l'expression de la variance

$$\begin{aligned}\mathbf{V}(Z_{1,n}) &= \left(1 - \left(1 - \frac{1}{N}\right)^n\right)\left(1 - \frac{1}{N}\right)^n\\ &\underset{n\to\infty}{\longrightarrow} 0.\end{aligned}$$

On en déduit que

$$\mathbf{V}(Y_n) \underset{n\to\infty}{\longrightarrow} 0 \quad \text{et} \quad b(Y_n) \underset{n\to\infty}{\longrightarrow} 0$$

On sait alors que l'estimateur est convergent.

## Exercice 22 p. 13

**1.**

```
def ech(n,p):
    return rd.random(n)<p
```

**2.a)**

```
def MonBienayme(n,p,alpha):
    # Création de l'échantillon
    E=ech(n,p)
    e=np.mean(E)
    # Calcul des bords de l'intervalle
    u=e-1/(2*(n*alpha)**(1/2))
    v=e+1/(2*(n*alpha)**(1/2))
    # Test si p est dans l'intervalle
    if (u<=p) and (p<=v):
        return 1
    else : return 0
```

**2.b)** On réalise un grand de fois l'expérience précédente et on regarde le nombre de succès sur le nombre de tentatives.

```
def proba(n,p,alpha):
    m=5000
    s=0
    for i in range(m):
        s+=MonBienayme(n,p,alpha)
    print('Longueur :', 1/(n*alpha)**(1/2),
        'Proba :', s/m)

n=2000
p=1/3
alpha=0.05
proba(n,p,alpha)
>>> Longueur : 0.1 Proba : 1.0
```

La longueur de l'intervalle est assez grande (relativement à $p$) mais le paramètre a toujours été dans l'intervalle.

**3.**

```
talpha=1.96

def IntAsymptotique(n,p,talpha):
    E=ech(n,p)
    # Calcul des bords de l'intervalle
    u=np.mean(E)-talpha/(2*(n)**(1/2))
    v=np.mean(E)+talpha/(2*(n)**(1/2))
    if (u<=p) and (p<=v):
        return 1
    else : return 0

def proba2(n,p,talpha):
    m=5000
    s=0
    for i in range(m):
        s+=IntAsymptotique(n,p,talpha)
    print('Longueur :', talpha/(n)**(1/2),'
        Proba :', s/m)

proba2(n,p,talpha)
>>> Longueur : 0.043 Proba : 0.9628
```

**4.** Le second intervalle de confiance semble plus précis car il a une étendue (une longueur) plus petite mais cela a un coût, il peut arriver (environ 1 chance sur 20) que le paramètre soit en dehors de l'intervalle.

## Exercice 23 p. 13

## Exercice 24 p. 13

## Exercice 25 p. 13

## Exercice 26 p. 14

## Exercice 27 p. 14

## Exercice 28 p. 14

**1.**

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

**2.** Le code affiche les 100 premiers termes $(f_n^{1/n})_n$. On conjecture que

$$f_n^{1/n} \underset{n\to\infty}{\longrightarrow} \varphi = \frac{1+\sqrt{5}}{2} \simeq 1{,}618.$$

*Ce résultat est assez facile à démontrer car, à partir des résultats sur les suites récurrentes linéaires d'ordre 2, on a*

$$F_n = \frac{1}{\sqrt{5}}\left(\varphi^n - \varphi'^n\right),$$

$$\text{avec} \quad \varphi = \frac{1+\sqrt{5}}{2} \quad \text{et} \quad \varphi' = \frac{1-\sqrt{5}}{2} = -\frac{1}{\varphi}.$$

*Le nombre $\varphi$ est appelé le nombre d'or. Ce nombre a de nombreuses propriétés arithmétiques. L'Antiquité attribue à ce nombre des propriétés « esthétiques ». Un rectangle dont le rapport longueur sur largeur serait égal au nombre d'or serait le plus harmonieux. Citons plus récemment, l'utilisation faite par l'architecte Le Corbusier avec le modulor permettant de construire des habitations à taille humaine.*

**3.** Vérifier que pour tout $n \in \mathbb{N}$

$$L_n A_n = L_{n+1}.$$

*Attention à l'ordre du produit.*
Par récurrence, on montre que

$$B_n = A_0 A_1 \dots A_n = \prod_{i=0}^{n-1} A_i.$$

**4.** Plusieurs solutions possibles :

```
def simuU(p):
    x=rd.rand()
    if x<p: return 1
    else : return -1
# ou
def simuU(p):
    return (-1)**(rd.rand()>p)
```

Et pour $A_n$ :

```
def simuAn(p):
    A=np.array([[0,simuU(p)],[1,simuU(p)]])
    return A
```

**5.**

```
def simuBn(n,p):
    B=simuAn(p)
    for i in range(1,n):
        A=simuAn(p)
        B=np.dot(B,A)
    return B

def simuFn(n,p):
    B=simuBn(n,p)
    return B[1,0]
```

Pour $p = 1$, on doit retrouver le cas déterminé de la première question. Testons :

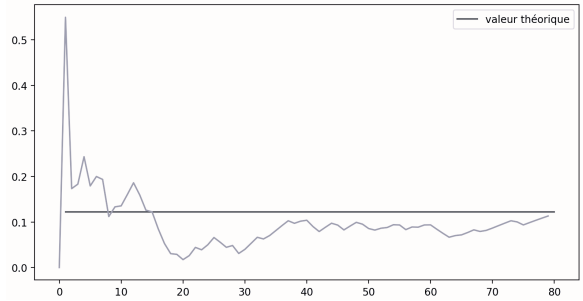
```
>>> Fibo(18)[-1]
2584.0
>>> simuFn(18,1)
2584
# Oui !
```

**6.**

```
def norme(A):
    [n,p]=np.shape(A)
    s=0
    for i in range(n):
        for j in range(p):
            s+=A[i,j]**2
    return s**(1/2)
```

**7.** On peut tester dans un premier temps si la suite de terme général $\dfrac{\ln(\|B_n\|)}{n}$ converge vers une constante.

```
def cv(n,p):
    N=np.zeros(n)
    B=simuAn(p)
    for i in range(1,n):
        A=simuAn(p)
        B=np.dot(B,A)
        N[i]=np.log(norme(B))/i
    return N


l=np.log(1.131988)
plt.plot([1,80],[l,l], label='valeur théorique')
plt.legend()
plt.plot(cv(80,1/2))
plt.show()
```




La réponse est positive, on devine une convergence.

## Exercice 29 p. 15

On montre que si $U \hookrightarrow \mathcal{U}(]0;1))$ alors $-\ln(U) \hookrightarrow \mathcal{E}(1)$. Par les propriétés du logarithme, le code simule la variable

$$-\ln\Big(\prod_{i=1}^{4} U_i\Big) = \sum_{i=1}^{4} \underbrace{-\ln(U_i)}_{\hookrightarrow \mathcal{E}(1)}.$$

D'après les propriétés par somme de variables aléatoires de loi exponentielle de paramètre 1, on peut affirmer que le programme simule une loi $\gamma(4)$.

## Exercice 30 p. 15

**1.** Soit $x \in [0;1]$. Appliquons la formule des probabilités totales avec le système complet d'événements $(N = n)_{n\in\mathbb{N}}$

$$\begin{aligned} F(x) &= \mathbf{P}(X \leqslant x) \\ &= \sum_{n=1}^{+\infty} \mathbf{P}(N = n)\mathbf{P}_{[N=n]}(X \leqslant x) \\ &= \sum_{n=1}^{+\infty} \mathbf{P}(N = n)\mathbf{P}_{[N=n]}(\max\{U_1 \dots U_n\} \leqslant x) \\ &= \sum_{n=1}^{+\infty} \mathbf{P}(N = n)\mathbf{P}(\max\{U_1, \dots, U_n\} \leqslant x). \end{aligned}$$

La dernière égalité s'obtenant par indépendance de N avec $\max\{U_1, \dots, U_n\}$. Ensuite

$$\begin{aligned} F(x) &= \sum_{n=1}^{+\infty} \mathbf{P}(N = n)\mathbf{P}\big([U_1 \leqslant x] \cap \dots \cap [U_n \leqslant x]\big) \\ &= \sum_{n=1}^{+\infty} \mathbf{P}(N = n)\mathbf{P}(U_1 \leqslant x)^n \end{aligned}$$

par indépendance de $(U_i)_{i\in\mathbb{N}^*}$ et même loi. En remplaçant par les expressions des lois :

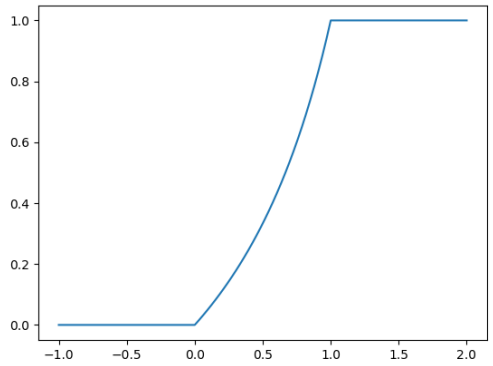
$$\begin{aligned} F(x) &= \sum_{n=1}^{+\infty} p(1-p)^{n-1} \cdot x^n \\ &= px \sum_{n=1}^{+\infty} ((1-p)x)^{n-1} = \frac{px}{1-(1-p)x}. \end{aligned}$$

Pour $x \leqslant 0$, $F(x) = 0$, et $x \geqslant 1$, $F(x) = 1$.

*Voici un code pour tracer la courbe représentative de la fonction de répartition.*

```
def FctionRep(x,p):
    if x<0 :
        return 0
    if x>1:
        return 1
    else :
        return p*x/(1-(1-p)*x)
```

```
def trace(p):
    x=np.linspace(-1,2,100)
    y=np.zeros(len(x))
    for i in range(len(x)):
        y[i]=FctionRep(x[i],p)
    plt.clf()
    plt.plot(x,y)
    plt.show()

#test
trace(1/2)
```




**2.** Appliquons la formule de l'espérance totale :

- $(\mathrm{N}=n)_{n\in\mathbb{N}}$ est un système complet d'événements.
- Soit $n\in\mathbb{N}^*$. Posons $\overline{\mathrm{U}}_n=\max(\mathrm{U}_1,\dots,\mathrm{U}_n)$. En reprenant le calcul précédent. Pour $x\in[0;1]$

$$\mathrm{F}_{\overline{\mathrm{U}}_n}(x)=x^n.$$

$\overline{\mathrm{U}}_n$ est à valeurs dans $[0,1]$, et une densité est :

$$\forall x\in\mathbb{R},\quad f_n(x)=\begin{cases} nx^{n-1} & \text{si } x\in[0;1]\\ 0 & \text{sinon.}\end{cases}$$

$\overline{\mathrm{U}}_n$ est bornée, elle admet une espérance avec

$$\mathbf{E}\left(\overline{\mathrm{U}}_n\right)=\int_0^1 x\cdot nx^{n-1}\,\mathrm{d}x=\frac{n}{n+1}.$$

- Ainsi $\overline{\mathrm{U}}_n$ est positive et

$$\mathbf{E}\left(\overline{\mathrm{U}}_n\right)=\mathbf{E}\big(|\mathrm{X}|\,|[\mathrm{N}=n]\big).$$

De plus

$$\sum\mathbf{P}(\mathrm{N}=n)\mathbf{E}\big(|\mathrm{X}|\,|\,[\mathrm{N}=n]\big)=\sum pq^{n-1}\cdot\frac{n}{n+1}.$$

La série est convergente.
D'après la formule de l'espérance totale, X admet une espérance et :

$$\begin{aligned}\mathbf{E}(\mathrm{X})&=\sum_{n=1}^{+\infty}\mathbf{P}(\mathrm{N}=n)\mathbf{E}(\mathrm{X}\,|\,[\mathrm{N}=n])\\&=\sum_{n=1}^{+\infty}pq^{n-1}\cdot\frac{n}{n+1}\\&=\sum_{n=1}^{+\infty}pq^{n-1}\left(\frac{n+1-1}{n+1}\right).\\&=\sum_{n=1}^{+\infty}pq^{n-1}-pq^{-2}\sum_{n=1}^{+\infty}\frac{q^{n+1}}{n+1}.\end{aligned}$$

Or on montre que pour $x\in]-1;1]$

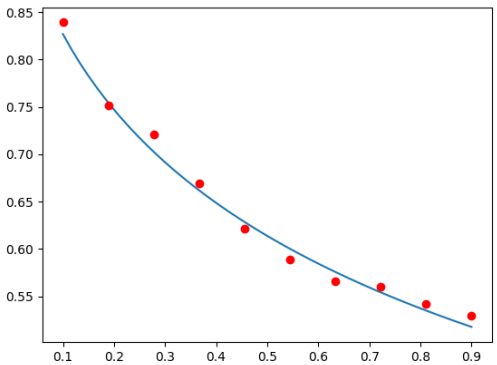
$$\ln(1-x)=-\sum_{n=1}^{+\infty}\frac{x^n}{n}.$$

D'où

$$\begin{aligned}\mathbf{E}(\mathrm{X})&=1-pq^{-2}\sum_{n=2}^{+\infty}\frac{q^n}{n^2}\\&=1-pq^{-2}\left(\sum_{n=1}^{+\infty}\frac{q^n}{n}-q\right)\\&=1+pq^{-2}(\ln(1-q)+q)\\&=1+\frac{p}{(1-p)^2}\big(\ln(p)+1-p\big).\end{aligned}$$

**3.**

```
def simuGeometrique(p):
    n=1
    while np.random.rand()>p:
        n=n+1
    return n

def simu(p):
    n=simuGeometrique(p)
    U=np.random.rand(n)
    return max(U)

def approxEsperance(p):
    s=0
    m=500
    for i in range(m):
        s+=simu(p)

    ValTheo=1+p*(1-p)**(-2)*(np.log(p)+1-p)
    #print(s/1000,'à comparer à',ValTheo)
    return s/m

def Valtheorique(p):
    return 1+p*(1-p)**(-2)*(np.log(p)+1-p)

def comparaison():
    plt.clf()
    x=np.linspace(0.1,0.9,50)
    plt.plot(x,Valtheorique(x))

    p=np.linspace(0.1,0.9,10)
    for i in range(len(p)):
        plt.plot([p[i]],[approxEsperance(p[
            i])],'ro')
    plt.show()

comparaison()
```




## Exercice 31 p. 15

**1.** On donne deux solutions :

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

```
def X(p):
    u=rd.random()<p
    return 2*u-1
```

**2.**

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

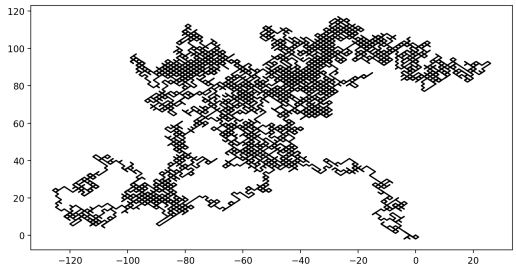
**3.**

```
    A=0
    n=0
    while abs(A)<m:
        A+=X(p)
        n+=1
    return n
```

**4.** Comme $B_n$ et $C_n$ ont même loi que $A_n$, on peut utiliser le programme de la question 2.

```
def simuPlan(n,p):
    Absc=simuA(p,n)
    Ordo=simuA(p,n)
    plt.plot(Absc,Ordo,'k-')
    plt.show()
```

Un exemple de trajectoire dans plan obtenue avec :
`>>> simuPlan(10000,1/2)`



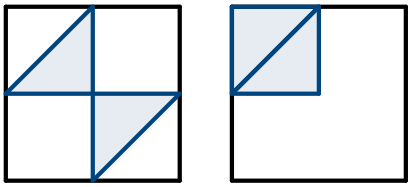

**5.** Un point $(x,y)$ est en dehors du cercle de rayon $m$ et de centre l'origine si $\|(x,y)\| > m$, soit

$$x^2+y^2 > m^2.$$

```
def TmPlan(p,m):
    A=0; B=0; n=0
    while A**2+B**2<=m**2:
        A+=X(p)
        B+=X(p)
        n+=1
    return n
```

## Exercice 32 p. 15

Le code tire au hasard deux réels $x$, $y$ dans $[0;1]$ correspondants aux points de cassure du bâton. Si les trois morceaux ainsi formés donne un triangle, le point $(x,y)$ est marqué d'un point. Sinon, il est marqué d'un carré. Il se dessine alors deux zones (dessin de gauche) dans le carré $[0;1]^2$. Si $(x,y)$ est dans la partie grisée, on obtient un triangle, au contraire de la partie blanche. La probabilité d'un triangle est alors le quotient de l'aire grisée sur l'aire totale du carré (ici 1). En déplaçant la partie en bas à droite de la figure, on constate que l'aire (et donc la probabilité) est de 1/4 (dessin de droite).



## Exercice 33 p. 16

**1.**

```
def Bernoulli(p):
    x=rd.random()
    if x<p :
        return 1
    else : return 0
```

**2.** On peut retraduire l'algorithme : on effectue deux tirages, si ces deux tirages donnent la même chose, on recommence sinon, on renvoie le résultat du deuxième tirage. On utilise une boucle `while`.

```
def algo(p):
    Tirage1=Bernoulli(p)
    Tirage2=Bernoulli(p)
    while Tirage1==Tirage2:
        Tirage1=Bernoulli(p)
        Tirage2=Bernoulli(p)
    return Tirage2
```

**3.**

```
# test
p=1/3
Compteur=0
for i in range(4000):
    Compteur+=algo(p)
    # Si Pile, on ajoute 1
    # Si Face, on ajoute 0
# Compteur donne le nombre de Pile
Freq=Compteur/4000
print('Pile :',Freq,'Face :',1-Freq)

# Résultat
Pile : 0.498575 Face : 0.501425
```

Le programme semble fonctionner car on a approximativement le même nombre de Pile que de Face, c'est-à-dire, la même chance d'avoir Pile que Face.

**4.** Noter que le nombre de tirage est pair. Car si on recommence, on relance deux fois la pièce.

```
def algo2(p):
    CompteLancer=2
    Tirage1=Bernoulli(p)
    Tirage2=Bernoulli(p)
    while Tirage1==Tirage2:
        Tirage1=Bernoulli(p)
        Tirage2=Bernoulli(p)
        CompteLancer+=2
    return Tirage2,CompteLancer
```

**5.** Par application de la loi faible des grands nombres, si on réalise un grand nombre de tirages et que l'on regarde la moyenne empirique du nombre de lancers, on obtient une approximation de $\mathbf{E}(T)$. Ici, cela donne par exemple :

```
def esperanceT(p):
    C=0
    for i in range(10000):
        C+=algo2(p)[1]
    return C/10000
```

Un petit test, on en vérifie la symétrie. On obtient la même espérance pour $p$ et pour $1-p$ car cela revient à échanger pile et face.

```
>>> esperanceT(1/4)
5.4214
>>> esperanceT(3/4)
5.3502
```

**1.**

```
def Bernoulli(p):
    x=rd.random()
    if x<p :
        return 1
    else : return 0
```

**2.** On peut retraduire l'algorithme : on effectue deux tirages, si ces deux tirages donnent la même chose, on recommence sinon, on renvoie le résultat du deuxième tirage. On utilise une boucle `while`.

```
def algo(p):
    Tirage1=Bernoulli(p)
    Tirage2=Bernoulli(p)
    while Tirage1==Tirage2:
        Tirage1=Bernoulli(p)
        Tirage2=Bernoulli(p)
    return Tirage2
```

**3.**

```
# test
p=1/3
Compteur=0
for i in range(4000):
    Compteur+=algo(p)
    # Si Pile, on ajoute 1
    # Si Face, on ajoute 0
# Compteur donne le nombre de Pile
Freq=Compteur/4000
print('Pile :',Freq,'Face :',1-Freq)

# Résultat
Pile : 0.498575 Face : 0.501425
```

Le programme semble fonctionner car on a approximativement le même nombre de Pile que de Face, c'est-à-dire, la même chance d'avoir Pile que Face.

**4.** Noter que le nombre de tirage est pair. Car si on recommence, on relance deux fois la pièce.

```
def algo2(p):
    CompteLancer=2
    Tirage1=Bernoulli(p)
    Tirage2=Bernoulli(p)
    while Tirage1==Tirage2:
        Tirage1=Bernoulli(p)
        Tirage2=Bernoulli(p)
        CompteLancer+=2
    return Tirage2,CompteLancer
```

**5.** Par application de la loi faible des grands nombres, si on réalise un grand nombre de tirages et que l'on regarde la moyenne empirique du nombre de lancers, on obtient une approximation de $\mathbf{E}(T)$. Ici, cela donne par exemple :

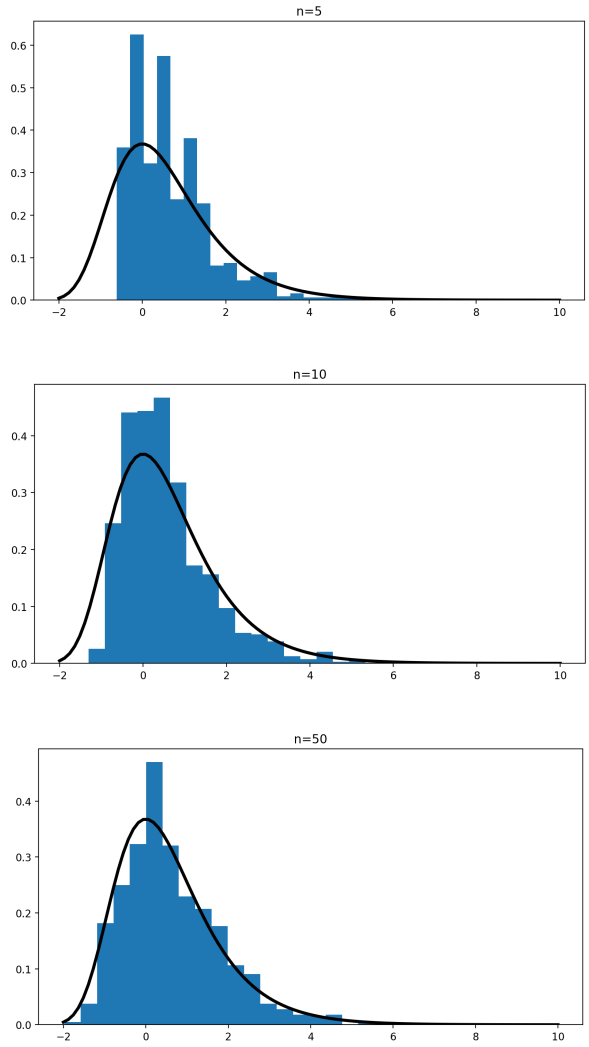
```
def esperanceT(p):
    C=0
    for i in range(10000):
        C+=algo2(p)[1]
    return C/10000
```

Un petit test, on en vérifie la symétrie. On obtient la même espérance pour $p$ et pour $1-p$ car cela revient à échanger pile et face.

```
>>> esperanceT(1/4)
5.4214
>>> esperanceT(3/4)
5.3502
```

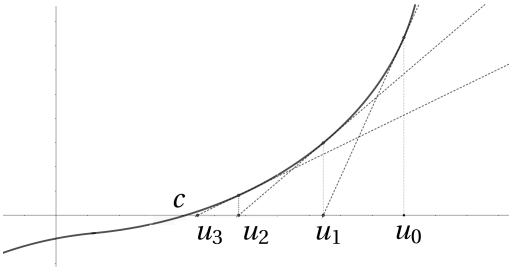
## Exercice 34 p. 16

```
def geo(p):
    n=1
    while rd.random()>p:
        n+=1
    return n

def simulV(n):
    # simulation de Cn
    c=0
    for i in range(1,n+1):
        p=(n-i+1)/n
        c+=geo(p)
    return c/n-np.log(n)

def simulVech(n):
    E=np.zeros(1000)
    for i in range(1000):
        E[i]=simulV(n)
    return E
```




## Exercice 35 p. 17

**1.** On conjecture que la suite $u$ tend vers $c$, le zéro de la fonction $f$.




**2.** L'équation au point d'abscisse $u_n$ est

$$y = f(u_n) + f'(u_n)(x - u_n).$$

On résout donc l'équation d'inconnue $x \in \mathbb{R}$,

$$0 = f(u_n) + f'(u_n)(x - u_n).$$

Comme $f'(u_n) \neq 0$, on trouve une unique solution donnée par :

$$u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}.$$

**3.** Supposons qu'il existe $\ell \in \mathbb{R}$ tel que $u_n \underset{n\to\infty}{\longrightarrow} \ell$.
Par continuité de $f$ et $f'$, on a aussi :

$$f(u_n) \underset{n\to\infty}{\longrightarrow} f(\ell) \quad \text{et} \quad f'(u_n) \underset{n\to\infty}{\longrightarrow} f'(\ell).$$

Comme $f'$ ne s'annule pas, on a par somme et quotient :

$$u_n - \frac{f(u_n)}{f'(u_n)} \underset{n\to\infty}{\longrightarrow} \ell - \frac{f(\ell)}{f'(\ell)}.$$

De plus, on a par définition de $u$,

$$\forall\, n \in \mathbb{N}, \quad u_{n+1} = u_n - \frac{f(u_n)}{f'(u_n)}.$$

Par unicité de la limite $\quad \ell = \ell - \dfrac{f(\ell)}{f'(\ell)}.$

Nécessairement $\quad f(\ell) = 0.$

Retenons que si la suite $u$ converge, alors elle converge vers l'unique zéro de $f$.

**4.a)** La fonction $f$ est bien de classe $\mathscr{C}^1$ car polynomiale. Elle ne s'annule qu'en $\sqrt{2}$ et la dérivée ne s'annule pas sur $[1;2]$. De plus, pour tout $x \in [1;2]$, $f'(x) = 2x$. On en déduit que pour tout $n \in \mathbb{N}$,

$$u_{n+1} = u_n - \frac{{u_n}^2 - 2}{2u_n}.$$

Expression que l'on simplifie sous la forme :

$$u_{n+1} = \frac{1}{2}\left(u_n + \frac{2}{u_n}\right).$$

Dans ce cas, on obtient

$$\begin{aligned} u_{n+1} - \sqrt{2} &= \frac{1}{2}\left(u_n + \frac{2}{u_n}\right) - \sqrt{2} \\ &= \frac{{u_n}^2 - 2\sqrt{2}u_n + 2}{2u_n} \\ u_{n+1} - \sqrt{2} &= \frac{(u_n - \sqrt{2})^2}{2u_n}. \end{aligned}$$

Comme $2u_n \geqslant 1$, $1/(2u_n) \leqslant 1$ et

$$\left|u_{n+1} - \sqrt{2}\right| \leqslant \frac{\left|u_n - \sqrt{2}\right|^2}{2u_n} \leqslant \left|u_n - \sqrt{2}\right|^2.$$

- Procédons par récurrence sur la propriété :

$$\mathscr{P}(n) : \qquad \left|u_n - \sqrt{2}\right| \leqslant |u_0 - \sqrt{2}|^{2^n}.$$

*Initialisation.*
On a $|u_0 - \sqrt{2}| = |u_0 - \sqrt{2}|^{2^0}$.
La propriété $\mathscr{P}(0)$ est donc vraie.

*Hérédité.* Soit $n \in \mathbb{N}$. Supposons $\mathscr{P}(n)$ vraie et démontrons que $\mathscr{P}(n+1)$ est vraie. On a d'après ce qui précède

$$\begin{aligned} \left|u_{n+1} - \sqrt{2}\right| &\leqslant |u_n - \sqrt{2}|^2 \leqslant \left||u_0 - \sqrt{2}|^{2^n}\right|^2 \\ &\leqslant |u_0 - \sqrt{2}|^{2^n \cdot 2} \leqslant |u_0 - \sqrt{2}|^{2^{n+1}}. \end{aligned}$$

$\mathscr{P}(n+1)$ est donc bien vérifiée.

*Conclusion.* Pour tout entier $n$, $\mathscr{P}(n)$ est vraie.

**4.b)** Notons que $|u_0 - \sqrt{2}| < 1$, puis

$$\left|u_0 - \sqrt{2}\right|^{2^n} \underset{n\to\infty}{\longrightarrow} 0.$$

Par encadrement $\quad u_n \underset{n\to\infty}{\longrightarrow} \sqrt{2}.$

**5.** Un programme possible est :

```
def u_n(fonct,der,u0,n) :
    u = u0
    for i in range(n) :
        u = u-fonct(u)/der(u)
    return u
```

**6.** Donnons un code à l'aide d'une boucle while.

```
def approx(p) :
    # p est la précision
    u=1
    erreur=1/2
    while erreur>p :
        erreur=erreur**2
        u=(u+2/u)/2
    # La formule de récurrence de la suite
        u
    return u
```

*La méthode de Newton est à comparer à la méthode d'approximation par dichotomie. La convergence de la suite obtenue par la méthode de Newton est bien plus rapide que celle par dichotomie. Il faut cependant supposer la fonction $f$ dérivable alors que la dichotomie ne suppose que la continuité.*

**7.** On peut considérer la fonction F définie sur $\mathbb{R}^2$ par

$$\mathrm{F}(x,y) = \big(\cos(x) - \sin(y), \mathrm{e}^{-x} - \cos(y)\big).$$

Les fonctions coordonnées sont de classe $\mathscr{C}^1$ sur $\mathbb{R}^2$ et

$$\mathrm{J}(x,y) = \begin{bmatrix} -\sin(x) & -\cos(y) \\ -\mathrm{e}^{-x} & \sin(y) \end{bmatrix}.$$

On en déduit :

```
def F(X):
    a=np.cos(X[0])-np.sin(X[1])
    b=np.exp(-X[0])-np.cos(X[1])
    return np.array([[a],[b]])

def jacobienne(X):
    a=-np.sin(X[0])
    b=-np.cos(X[1])
    c=-np.exp(-X[0])
    d=np.sin(X[1])
    return np.array([[a,b],[c,d]])
```

**8.**

```
def approximation(n):
    X=np.array([[1],[0]])
    for i in range(n):
        Jinv=np.linalg.inv(
                jacobienne(X))
        X=X-np.dot(Jinv,F(X))
    return X
```

On obtient une solution approximative avec le couple $(0.58853, 0.98226)$.