Annales questions Python classées par thème

On supposera toujours que les bibliothèques au programme sont importées sous leurs alias habituels :

- `import numpy as np`
- `import numpy.linalg as al`
- `import numpy.random as rd`
- `import matplotlib.pyplot as plt`
- `import pandas as pd`

Table des matières

1. Algèbre

1.1. Calcul de rang

1.1.1. EDHEC 2025

Contexte

On considère la matrice $A = \begin{pmatrix} 1 & 3 & 1 \\ 3 & -1 & 3 \\ 1 & 3 & 1 \end{pmatrix}$.
On admet que 5 et -4 sont des valeurs propres de A.
On admet également que $\dim(E_5(A)) = \dim(E_{-4}(A)) = 1$.
On dispose d'une fonction `matA()` qui renvoie la matrice A sous forme de tableau numpy.

On considère les instructions **Python** suivantes :

```
r1=al.matrix_rank(matA()-5*np.eye(3,3))
r2=al.matrix_rank(matA()+4*np.eye(3,3))
print('r1=',r1)
print('r2=',r2)
```

Donner les valeurs de r_1 et r_2 renvoyées par ce script.

1.1.2. EDHEC 2022

Contexte

On considère un endomorphisme φ de $\mathscr{M}_2(\mathbb{R})$ dont la matrice représentative dans la base canonique est notée A.

En **Python**, la commande `r=al.matrix_rank(M)` renvoie dans la variable `r` le rang de la matrice M.
On a saisi :

```
A = np.array([[0,-1,1,0],[-1,0,0,1],[1,0,0,-1],[0,1,-1,0]])
r1 = al.matrix_rank(A-2*np.eye(4))
r2 = al.matrix_rank(A+2*np.eye(4))
print('r1=',r1)
print('r2=',r2)
```

Python a renvoyé :

```
r1=3
r2=3
```

Que peut-on conjecturer quant aux valeurs propres non nulles de A et à la dimension des sous-espaces propres associés ?

1.2. Puissance d'une matrice

1.2.1. EDHEC 2025

Contexte

On pose $M(a,b) = \begin{pmatrix} a & b & a \\ b & 2a-b & b \\ a & b & a \end{pmatrix}$, où a et b sont des réels.

On pose $U = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$, $V = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$ et $W = \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix}$.

On admet que :

- U est un vecteur propre de $M(a,b)$ associé à la valeur propre 0,
- V est un vecteur propre de $M(a,b)$ associé à la valeur propre $2a+b$,
- W est un vecteur propre de $M(a,b)$ associé à la valeur propre $2a-2b$.

On note P la matrice dont les colonnes sont les vecteurs U, V et W et on note D la matrice diagonale dont les coefficients diagonaux sont, dans cet ordre, 0, $2a+b$ et $2a-2b$.

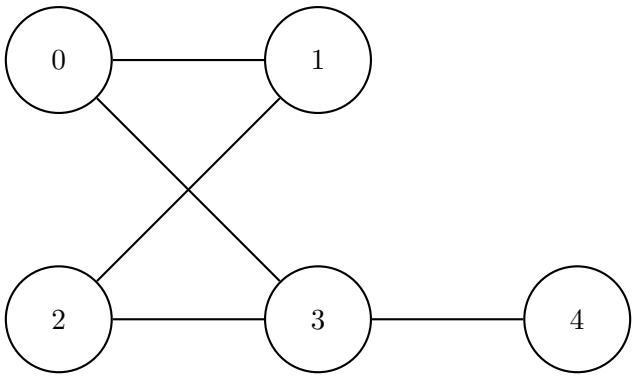
On admet que, pour tout $n \in \mathbb{N}^*$, $M(a,b)^n = PD^nP^{-1}$.

En déduire, sans la commande `al.matrix_power`, et pour $n \in \mathbb{N}^*$, une fonction **Python** d'en-tête `puissanceM(a,b,n)` renvoyant $M(a,b)^n$.

1.2.2. EDHEC 2023

Contexte

On considère le graphe G suivant et on note A la matrice d'adjacence de G.



Il y a 5 chemins de longueur 3 entre les sommets 2 et 3.

On considère la fonction **Python** suivante :

```
1  def f(M,k):
2      N=al.matrix_power(M,k)
3      return N
```

On suppose que l'on a saisi la matrice A et on considère les instructions :

```
1  B=f(A,---)
2  n=B[---]
3  print(n)
```

Compléter ces instructions pour qu'elles permettent l'affichage du nombre de chemins de longueur 3 entre les sommets 2 et 3.

1.2.3. ESSEC II 2023

Écrire une fonction **Python** `transition(t,G)` de paramètres `G` représentant une matrice carrée d'ordre n et `t` représentant un réel positif, qui renvoie la matrice $\left(I_n + \frac{t}{1000}G\right)^{1000}$.

1.2.4. EDHEC 2017

Contexte

On note E l'espace vectoriel des fonctions polynomiales de degré inférieur ou égal à 2 et on rappelle que la famille (e_0, e_1, e_2) est une base de E, les fonctions e_0, e_1 e_2 étant définies par :

$$\forall t \in \mathbb{R} \quad e_0(t) = 1, \quad e_1(t) = t, \quad e_2(t) = t^2$$

On considère l'application φ qui, à toute fonction P de E, associe la fonction, notée $\varphi(P)$, définie par :

$$\forall x \in \mathbb{R},\ (\varphi(P))(x) = \int_0^1 P(x+t)\ dt$$

On note A la matrice de φ dans la base (e_0, e_1, e_2). On admet que $A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$.

Compléter les commandes **Python** suivantes pour que soit affichée la matrice A^n pour une valeur de n entrée par l'utilisateur :

```
1 n = int(input('Rentrez la valeur de n :'))
2 A = ---
3 print(---)
```

1.3. Récupération des valeurs propres d'une matrice

1.3.1. ESSEC II 2024

Aide-mémoire

- Si `T` est un tableau `numpy`, `np.shape(T)` renvoie le nombre de lignes et le nombre de colonnes de T, dans cet ordre, sous la forme d'un couple.
- `np.zeros([p,q])` crée un tableau `numpy` à p lignes et q colonnes ne contenant que des 0.
- `np.ones([p,q])` crée un tableau `numpy` à p lignes et q colonnes ne contenant que des 1.
- `np.eye(p)` crée un tableau `numpy` à p lignes et p colonnes ne contenant que des 1 sur sa diagonale et des 0 ailleurs.
- La fonction `al.eigvals`, appliquée à un tableau `numpy` représentant une matrice symétrique A, renvoie le tableau des coefficients d'une matrice diagonale semblable à A.

Contexte

Dans tout le problème n est un entier supérieur ou égal à 2.
Soit A une matrice carrée symétrique appartenant à $\mathscr{M}_n(\mathbb{R})$.
On admet qu'il existe une matrice carrée inversible P appartenant à $\mathscr{M}_n(\mathbb{R})$ telle que $P^{-1}AP$ soit une matrice diagonale. On admet également que les éléments diagonaux de $P^{-1}AP$ sont alors les valeurs propres de la matrice A (apparaissant chacune autant de fois que la dimension du sous-espace propre associé).

En notant $\lambda_1, \ldots, \lambda_n$ les éléments diagonaux de $P^{-1}AP$, on pose alors $\mathcal{E}(A) = \sum_{k=1}^{n} |\lambda_k|$, on nomme ce réel positif l'énergie de A.
On admet que cette somme ne dépend pas du choix de P.

Ecrire une fonction **Python** `energie(A)` qui renvoie l'énergie de la matrice symétrique représentée par le tableau `numpy A`.

Contexte

Si $A = (a_{i,j})_{1 \leqslant i,j \leqslant n}$ est une matrice carrée appartenant à $\mathscr{M}_n(\mathbb{R})$, on définit la trace de A, notée $\mathrm{tr}(A)$ *par :*

$$\mathrm{tr}(A) = \sum_{i=1}^{n} a_{i,i}$$

On admet que :
- Si A et B sont deux matrices semblables de $\mathscr{M}_n(\mathbb{R})$, alors $\mathrm{tr}(A) = \mathrm{tr}(B)$.
- $\mathrm{tr}(A^2) = \sum_{k=1}^{n} \lambda_k^2$.

Dans la console **Python**, on obtient :

```
> > > energie(3*np.eye(3)-np.ones([3,3]))
6.0
```

a) Déterminer quelle est la matrice A associée au tableau `3*np.eye(3)-np.ones([3,3])`.

b) En calculant $\mathrm{tr}(A)$, $\mathrm{tr}(A^2)$ et en déterminant une valeur propre évidente de A, expliciter son spectre et retrouver son énergie.

1.3.2. HEC 2017

Contexte

Soit B la matrice de $\mathscr{M}_3(\mathbb{R})$ définie par : $B = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

On considère les instructions et la sortie (`ans`) **Python** suivantes :

```
1 B = np.array([[0,1,0], [1,0,0], [0,0,1]])
2 P = np.array([[1,1,0], [1,-1,0], [0,0,1]])
3 C = np.dot(al.inv(P), B)
4 print(np.dot(C,P))
```

```
ans  =
    1.    0.    0.
    0.  - 1.    0.
    0.    0.    1.
```

Déduire les valeurs propres de B de la séquence **Python** précédente.

1.4. Modification d'une matrice

1.4.1. HEC/ESSEC I 2025

Contexte

Dans cet énoncé r est un entier naturel, $r \geqslant 2$. On note $\mathcal{ST}_r$ l'ensemble des matrices $M = (m_{i,j})_{1\leqslant i,j\leqslant r}$ à coefficients positifs telles que pour tout $i \in [\![1, r]\!]$, $\sum_{j=1}^{r} m_{i,j} = 1$.

On considère $\mu = (\mu_1 \ldots\ldots \mu_r)$ une matrice ligne appartenant à $\mathscr{M}_{1,r}(\mathbb{R})$ dont les coefficients sont strictement positifs et tels que $\sum_{k=1}^{r} \mu_k = 1$.

On dit que la matrice $M = (m_{i,j})_{1\leqslant i,j\leqslant r}$ est μ-réversible si M appartient à $\mathcal{ST}_r$ et vérifie :

$$\forall (i,j) \in [\![1, r]\!]^2, \mu_i m_{i,j} = \mu_j m_{j,i}$$

On note, pour tout $n \in \mathbb{N}^*$, $m_{i,j}^{(n)}$ les coefficients de la matrice M^n.

S'il existe $\sigma \in \mathbb{N}^*$ tel que, pour tout $(i,j) \in [\![1, r]\!]^2$, $m_{i,j}^{(\sigma)} > 0$, on dit que M est une matrice ergodique.

On définit aussi pour tout $n \in \mathbb{N}^*$ et $(i_0, \ldots, i_n) \in [\![1, r]\!]^{n+1}$:

$$\theta_M(i_0, \ldots, i_n) = m_{i_0,i_1} \times \cdots \times m_{i_{n-1},i_n} = \prod_{k=1}^{n} m_{i_{k-1},i_k}$$

On dit que M vérifie la propriété (K) si, pour tout $n \in \mathbb{N}^*$ et $i_0, \ldots, i_n$ éléments de $[\![1, r]\!]$:

$$\theta_M(i_0, i_1, \ldots, i_n, i_0) = \theta_M(i_0, i_n, \ldots, i_1, i_0) \qquad (K)$$

On admet que, si M est ergodique, il existe μ telle que M est μ-réversible si et seulement si M vérifie la propriété (K).

Soit $\alpha \in]0, 1]$. On définit deux opérations élémentaires sur les matrices $M = (m_{i,j})_{1\leqslant i,j\leqslant r}$ appartenant à $\mathcal{ST}_r$ comme suit :

- On modifie la ligne k de M, où $k \in [\![1, r]\!]$, en remplaçant pour tout $j \neq k$, $m_{k,j}$ par $m'_{k,j} = \alpha m_{k,j}$ et $m_{k,k}$ par $m'_{k,k} = 1 - \alpha + \alpha m_{k,k}$.
- On modifie la colonne k de M, où $k \in [\![1, r]\!]$, et sa diagonale en remplaçant pour tout $i \neq k$, $m_{i,k}$ par $m'_{i,k} = \alpha m_{i,k}$ et $m_{i,i}$ par $m'_{i,i} = m_{i,i} + (1 - \alpha) m_{i,k}$.
- Par exemple si $M = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{2}{3} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{pmatrix}$, $k = 1$ et $\alpha = \frac{1}{2}$, la première opération donne $M' = \begin{pmatrix} \frac{4}{6} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{3} & 0 & \frac{2}{3} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{pmatrix}$ et la deuxième $M' = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \\ \frac{1}{12} & \frac{2}{3} & \frac{1}{4} \end{pmatrix}$.

  Dans les deux cas on obtient ainsi à partir de $M \in \mathcal{ST}_r$, $M = (m_{i,j})_{1\leqslant i,j\leqslant r}$, une matrice que l'on notera dans la suite $M' = (m'_{i,j})_{1\leqslant i,j\leqslant r}$ qui appartient à $\mathcal{ST}_r$.

  On remarquera que l'on ne modifie ainsi tout au plus, pour ce qui est des éléments non diagonaux de M, que ceux de la ligne k, pour la première opération, et ceux de la colonne k pour la deuxième.

  On admet que M vérifie la propriété (K) si et seulement si M' aussi et que si M est ergodique, alors M' l'est aussi.

Contexte

Soit I un sous-ensemble non vide de $[\![1,r]\!]$ et $M \in \mathcal{ST}_r$, $M = (m_{i,j})_{1\leqslant i,j\leqslant r}$.
On définit le graphe non orienté $G_M(I)$ dont l'ensemble des sommets est I et tel que $\{i,j\}$ est une arête si $i \neq j$, $m_{i,j} \neq 0$ et $m_{j,i} = m_{i,j}$.
Donc si I est réduit à un seul élément le graphe ne comporte pas d'arête.
Par exemple si $M = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{2}{3} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{pmatrix}$, avec $I = \{1,2,3\}$ les arêtes sont $\{1,2\}$ et $\{2,3\}$, et avec $I = \{1,3\}$ il n'y a aucune arête.

On suppose que $M \in \mathcal{ST}_r$, $M = (m_{i,j})_{1\leqslant i,j\leqslant r}$ est ergodique et vérifie (K) dans la suite.
On pose $I = \{1\}$. Le graphe $G_M(I)$ est alors connexe. On admet que l'algorithme ci-dessous, composé de $r-1$ opérations élémentaires à partir de M et $I = \{1\}$, est valide et transforme M en une matrice M^* ergodique, vérifiant (K), symétrique et telle que $G_{M^*}([\![1,r]\!])$ est connexe.

$I = \{1\}$
`Tant que` $I \neq [\![1,r]\!]$`:`
    `On détermine un couple` $(\ell,k) \in I \times ([\![1,r]\!] \setminus I)$ `tel que` $m_{\ell,k} \neq 0$ `et` $m_{k,\ell} \neq 0$
    `Si` $m_{\ell,k} \leqslant m_{k,\ell}$`:`
        `On pose` $\alpha = \dfrac{m_{\ell,k}}{m_{k,\ell}}$ `et on fait l'opération élémentaire sur la ligne` k
        `avec ce coefficient`
    `Sinon:`
        `on pose` $\alpha = \dfrac{m_{k,\ell}}{m_{\ell,k}}$ `et on fait l'opération élémentaire sur la colonne` k
        `avec ce coefficient.`
    `On ajoute` k `à` I

On admet que si, à partir de $M \in \mathcal{ST}_r$, une suite d'opérations élémentaires la transforme en une matrice symétrique M^*, alors M vérifie la propriété (K).

1. Écrire une fonction **Python** `opLigne(M,k,alph)` qui réalise l'opération élémentaire sur la k-ième ligne de M représentée par la matrice numpy `M` et α représenté par `alph`.
On définit de même une fonction **Python** `opCol(M,k,alph)` qui réalise l'opération élémentaire sur la k-ième colonne et la diagonale de M (cette fonction n'est pas demandée).

2. Écrire une fonction **Python** `NonNul(M,I,J)` qui, étant donnés une matrice $M = (m_{i,j})_{1\leqslant i,j\leqslant r}$ représentée par la matrice numpy `M` et deux ensembles non vides d'indices I et J, représentés par les listes `I` et `J`, renvoie un couple (i,j) tel que $i \in I$, $j \in J$ et $m_{i,j} \neq 0$, c'est-à-dire `M[i-1][j-1]` est non nul, s'il existe un tel couple et le couple $(0,0)$ sinon.

3. Compléter la fonction suivante pour qu'elle réalise l'implémentation de l'algorithme de la question ***19.b)*** et renvoie `True` si M ergodique vérifie (K) et `False` sinon.

```
def estRev(M):
    r=np.shape(M)[0]
    I=[1]; J=[k for k in range(2,r+1)]
    while len(I)< ... :
        ell,k=NonNul(M,I,J)
        if (ell==0) or M[k-1][ell-1]*M[ell-1][k-1]==0:
            return ...
        else:
            if M[ell-1][k-1] <= M[k-1][ell-1]:
                OpLigne(M,k,M[ell-1][k-1]/M[k-1][ell-1])
            else:
                OpCol(M,k,M[k-1][ell-1]/M[ell-1][k-1])
            I.append(...)
            J.remove(...)
    return (np.transpose(M)==M).all()
    #  Teste l'égalité de deux matrices numpy
```

1.4.2. HEC 2019

Contexte

La fonction **Python** suivante permet de multiplier la $i^{\text{ème}}$ ligne L_i d'une matrice A par un réel sans modifier ses autres lignes, c'est-à-dire de lui appliquer l'opération élémentaire $L_i \leftarrow a\,L_i$ (où $a \neq 0$).

```
def multlig(a, i, A):
    n,p = np.shape(A)
    B = np.copy(A)
    for j in range(p):
        B[i-1, j] = a * B[i-1, j]
    return B
```

1. Donner le code **Python** de deux fonctions `adlig` (d'arguments `b`, `i`, `j`, `A`) et `echlig` (d'arguments `i`, `j`, `A`) permettant d'effectuer respectivement les autres opérations sur les lignes d'une matrice :

$$Li \leftarrow L_i + b\,L_j \ (i \neq j) \quad \text{et} \quad L_i \leftrightarrow L_j \ (i \neq j)$$

2. Expliquer pourquoi la fonction `multligmat` suivante retourne le même résultat `B` que la fonction `multlig`.

```
def multligmat(a, i, A):
    n,p = np.shape(A)
    D = np.eye(n)
    D[i-1, i-1] = a
    B = np.dot(D, A)
return B
```

1.5. Création d'une matrice dont les coefficients sont donnés explicitement

1.5.1. EDHEC 2025

Contexte

On considère la matrice $A = \begin{pmatrix} 1 & 3 & 1 \\ 3 & -1 & 3 \\ 1 & 3 & 1 \end{pmatrix}$.

Écrire une fonction **Python** d'en tête `matA()` retournant la matrice A.

1.5.2. ESSEC II 2024

Contexte

- Soit $U = \begin{pmatrix} u & v \\ v & w \end{pmatrix}$ une matrice carrée appartenant à $\mathscr{M}_2(\mathbb{R})$ symétrique et A une matrice carrée appartenant à $\mathscr{M}_n(\mathbb{R})$ symétrique.
  On définit $U * A$ (produit de Kronecker) la matrice carrée appartenant à $\mathscr{M}_{2n}(\mathbb{R})$ que l'on peut naturellement représenter ainsi $\begin{pmatrix} uA & vA \\ vA & wA \end{pmatrix}$ (écriture par blocs).
- Par exemple, si $U = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}$ et $A = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$, alors $U * A = \begin{pmatrix} 2A & A \\ A & 0_3 \end{pmatrix}$ par blocs,
  d'où $U * A = \begin{pmatrix} 0 & 2 & 2 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$ (0_3 désigne la matrice nulle de $\mathscr{M}_3(\mathbb{R})$).
- Si X est une matrice colonne appartenant à $\mathscr{M}_{2n,1}(\mathbb{R})$, $X = \begin{pmatrix} x_1 \\ \vdots \\ x_{2n} \end{pmatrix}$, on écrira $X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$ avec
  $X_1 = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ et $X_2 = \begin{pmatrix} x_{n+1} \\ \vdots \\ x_{2n} \end{pmatrix}$.
  On admet qu'alors la matrice colonne $(U * A)X$ de $\mathscr{M}_{2n,1}(\mathbb{R})$ est égale à $\begin{pmatrix} uAX_1 + vAX_2 \\ vAX_1 + wAX_2 \end{pmatrix}$.

1. Écrire une fonction **Python** `prod2K(u,v,w,A)` qui étant donné $U = \begin{pmatrix} u & v \\ v & w \end{pmatrix}$ et A, représentée par un tableau `numpy`, renvoie $U * A$ sous la forme d'un tableau `numpy`.

2. Compléter le code suivant s'affichant dans la console **Python** :

```
>>> prod2K(...,-1,...,...)
array([[-2., 1., 2., -1.],
       [1., -2., -1., 2.],
       [2., -1., -4., 2.],
       [-1., 2., 2., -4.]])
```

1.6. Création d'une matrice dont les coefficients sont reliés par des relations de récurrence

1.6.1. HEC 2018

Contexte

On rappelle que le cardinal d'un ensemble fini H, noté $\text{Card}(H)$, est le nombre de ses éléments.
Pour $k \in \mathbb{N}^*$, on note $P(k)$ l'ensemble des k-uplets $(x_1, x_2, \ldots, x_k)$ d'entiers naturels tels que :

$$\sum_{i=1}^{k} i\, x_i = k$$

c'est à dire : $P(k) = \{(x_1, x_2, \ldots, x_k) \in \mathbb{N}^k \mid x_1 + 2x_2 + \ldots + kx_k = k\}$. On pose $p(k) = \text{Card}\big(P(k)\big)$.
Pour tout couple $(\ell, k) \in (\mathbb{N}^*)^2$, on pose : $Q(\ell, k) = \{(x_1, x_2, \ldots, x_k) \in P(k) \mid x_1 + x_2 + \ldots + x_k \leqslant \ell\}$ et $q(\ell, k) = \text{Card}(Q(\ell, k))$.
On admet que :

- Pour tout entier $k \in \mathbb{N}^*$: $Q(1, k) = \{(0, \ldots, 0, 1)\}$.
- Pour tout entier $\ell \geqslant k$: $Q(\ell, k) = P(k)$.
- Pour tout entier ℓ supérieur ou égal à 2 et pour tout entier $k > \ell$: $q(\ell, k) = q(\ell - 1, k) + q(\ell, k - \ell)$.
- Pour tout entier ℓ supérieur ou égal à 2 : $q(\ell, \ell) - q(\ell - 1, \ell) = 1$.

La fonction **Python** suivante dont le script est incomplet (lignes 6 et 8), calcule une matrice `qmatrix(n)` telle que pour chaque couple $(\ell, k) \in [\![1, n]\!]^2$, le coefficient situé à l'intersection de la ligne ℓ et de la colonne k est égal à $q(\ell, k)$.

```
1  def qmatrix(n):
2      q = np.ones([n,n])
3      for L in range(1,n):
4          for K in range(1,n):
5              if K < L:
6                  q[L,K] = ................
7              elif K==L:
8                  q[L,K] = ..............
9              else:
10                 q[L,K] = q[L-1,K] + q[L,K-(L+1)]
11     return q
```

L'application de la fonction `qmatrix` à l'entier $n = 9$ fournit la sortie suivante :

```
--> qmatrix(9)
    1.     1.     1.     1.     1.     1.     1.     1.     1.
    1.     2.     2.     3.     3.     4.     4.     5.     5.
    1.     2.     3.     4.     5.     7.     8.    10.    12.
    1.     2.     3.     5.     6.     9.    11.    15.    18.
    1.     2.     3.     5.     7.    10.    13.    18.    23.
    1.     2.     3.     5.     7.    11.    14.    20.    26.
    1.     2.     3.     5.     7.    11.    15.    21.    28.
    1.     2.     3.     5.     7.    11.    15.    22.    29.
    1.     2.     3.     5.     7.    11.    15.    22.    30.
```

1. Compléter les lignes 6 et 8 du script de la fonction `qmatrix`.

2. Donner un script **Python** permettant de calculer $p(n)$ à partir d'une valeur de n entrée au clavier.

3. Conjecturer une formule générale pour $q(2,k)$ applicable à tout entier $k \geqslant 1$, puis la démontrer.

1.7. Calcul du n^{e} terme d'une suite de vecteurs colonnes

1.7.1. ECRICOME 2018

Contexte

Soit A la matrice de $\mathscr{M}_3(\mathbb{R})$ donnée par : $A = \begin{pmatrix} 2 & 1 & -2 \\ 0 & 3 & 0 \\ 1 & -1 & 5 \end{pmatrix}$.

Soit B la matrice de $\mathscr{M}_3(\mathbb{R})$ donnée par : $B = \begin{pmatrix} 1 & -1 & -1 \\ -3 & 3 & -3 \\ -1 & 1 & 1 \end{pmatrix}$.

On pose $X_0 = \begin{pmatrix} 3 \\ 0 \\ -1 \end{pmatrix}$, $X_1 = \begin{pmatrix} 3 \\ 0 \\ -2 \end{pmatrix}$, et pour tout entier naturel n :

$$X_{n+2} = \frac{1}{6}\, A\, X_{n+1} + \frac{1}{6}\, B\, X_n$$

Compléter la fonction ci-dessous qui prend en argument un entier n supérieur ou égal à 2 et qui renvoie la matrice X_n :

```
1  def CalcVecteur(n):
2      A = np.array([[2,1,-2], [0,3,0], [1,-1,5]])
3      B = np.array([[1,-1,-1], [-3,3,-3], [-1,1,1]])
4      Xold = [3,0,-1]
5      Xnew = [3,0,-2]
6      for i in range(2,n+1):
7          Aux = ________
8          Xold = ________
9          Xnew = ____________________________________________________________
10     return ________
```

2. Analyse

2.1. Algorithme de dichotomie

2.1.1. ESSEC II 2025

Contexte

On définit la fonction f sur $[0,1]$ par $f(t) = (1+t)\mathrm{e}^{-t} - t$.
On admet qu'il existe un unique $\alpha \in]0,1[$ tel que $f(\alpha) = 0$ et que pour tout $t \in]0,1[$, $f(t) > 0 \iff t < \alpha$.

Écrire un programme **Python** qui renvoie une valeur approchée de α à 10^{-3} près.

2.1.2. ECRICOME 2023

Contexte

On considère la fonction f définie sur $]0,+\infty[$ par :

$$\forall x \in]0,+\infty[, \qquad f(x) = \frac{e^{\frac{x}{2}}}{\sqrt{x}}$$

On admet que f est strictement décroissante sur $]0,1[$ et que, pour tout entier n supérieur ou égal à 2, l'équation $f(x) = n$, d'inconnue x dans $]0,1[$, possède exactement une solution u_n.
Soient n un entier supérieur ou égal à 2 et ε un réel strictement positif.
On cherche à déterminer une valeur approchée de u_n avec une marge d'erreur inférieure ou égale à ε. On rappelle pour cela le principe de l'algorithme de dichotomie.

- On initialise deux variables a et b en leur affectant respectivement les valeurs 0 et 1.
- Tant que $b - a > \varepsilon$, on répète les opérations suivantes.
  On considère le milieu c du segment $[a,b]$. Par monotonie de f sur $]0,1]$, en distinguant les cas $f(c) \leqslant n$ et $f(c) > n$, on peut déterminer si u_n appartient à l'intervalle $[a,c]$ ou à l'intervalle $[c,b]$. Selon le cas, on met alors à jour la valeur de a ou de b pour se restreindre au sous-intervalle approprié.
- On renvoie finalement la valeur $\frac{a+b}{2}$, qui constitue une valeur approchée de u_n à ε près.

Recopier et compléter la fonction en langage **Python** suivante, prenant en entrée un entier `n` supérieur ou égal à 2 et un réel strictement positif `eps`, et renvoyant une valeur approchée de u_n à `eps` près en appliquant l'algorithme décrit ci-dessus.

```
1  import numpy as np
2
3  def approx_u(n, eps):
4      a = 0
5      b = 1
6      while ..... :
7          c = (a+b)/2
8          if np.exp(c/2)/np.sqrt(c) < n:
9              .....
10         else:
11             .....
12     return (a+b)/2
```

2.1.3. EML 2021

Contexte

Pour tout (a, b, c) de $\mathbb{R}^3$, on définit la matrice $M(a, b, c)$ par :

$$M(a,b,c) = \begin{pmatrix} 1+a & 1 & 1 \\ 1 & 1+b & 1 \\ 1 & 1 & 1+c \end{pmatrix}$$

On se place dans le cas où $a < b < c$ et on note g la fonction définie sur l'ensemble $D = \mathbb{R} \setminus \{a, b, c\}$ par :

$$\forall x \in D, \quad g(x) = \frac{1}{x-a} + \frac{1}{x-b} + \frac{1}{x-c}$$

On admet que la matrice $M(a, b, c)$ admet 3 valeurs propres distinctes λ_1, λ_2 et λ_3 vérifiant :

$$a < \lambda_1 < b < \lambda_2 < c < \lambda_3$$

On admet également que les valeurs propres de $M(a, b, c)$ sont exactement les solutions de l'équation $g(x) = 1$, d'inconnue $x \in D$.

On pose : $A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \end{pmatrix}$ et on note α la plus grande valeur propre de A. On admet : $4 < \alpha < 5$.

Recopier et compléter les lignes incomplètes de la fonction **Python** ci-dessous afin qu'elle renvoie une valeur approchée de α à 10^{-3} près à l'aide de la méthode de dichotomie.

```
1  def valeur_approchee():
2      x = 4
3      y = 5
4      while ____________________:
5          m = (x + y) / 2
6          if 1/m + 1/(m-1) + 1/(m-2) __________:
7              _________
8          else:
9              _________
10     return (x+y)/2
```

2.1.4. EML 2020

Contexte

On note, pour tout n de $\mathbb{N}^*$, (E_n) l'équation : $x^n + x - 1 = 0$. On admet que, pour tout n de $\mathbb{N}^*$:

- la fonction $x \mapsto x^n + x - 1$ est strictement croissante sur $\mathbb{R}^+$,
- l'équation (E_n) admet une unique solution sur $\mathbb{R}_+$ que l'on note u_n,
- u_n appartient à l'intervalle $]0, 1[$.

Recopier et compléter la fonction **Python** suivante afin que, prenant en argument un entier n de $\mathbb{N}^*$, elle renvoie une valeur approchée de u_n à 10^{-3} près, obtenue à l'aide de la méthode par dichotomie.

```
def valeur_approchee(n):
    a = 0
    b = 1
    while ...
        c = (a+b)/2
        if c**n + c - 1 > 0:
            ...
        else:
            ...
    return ...
```

2.1.5. ECRICOME 2019

Contexte

Pour tout entier n non nul, on note h_n la fonction définie sur $\mathbb{R}_+^*$ par :

$$\forall x > 0, \quad h_n(x) \;=\; x^n + 1 + \frac{1}{x^n}$$

On admet que, pour tout entier naturel n non nul :

- la fonction h_n est strictement décroissante sur $]0, 1[$ et strictement croissante sur $[1, +\infty[$,
- l'équation $h_n(x) = 4$ admet exactement deux solutions, notées u_n et v_n et vérifiant : $0 < u_n < 1 < v_n$.

On dispose d'une fonction **Python** d'en-tête `def h(n,x):` qui renvoie la valeur de $h_n(x)$ lorsqu'on lui fournit un entier naturel n non nul et un réel $x \in \mathbb{R}_+^*$ en entrée.

Compléter la fonction suivante pour qu'elle renvoie une valeur approchée à 10^{-5} près de v_n par la méthode de dichotomie lorsqu'on lui fournit un entier $n \geqslant 1$ en entrée :

```
def approxv(n):
  a = 1
  b = 3
  while (b-a) > 10**(-5):
    c = (a+b)/2
    if h(n,c) < 4 :
      ........
    else:
      ........
  ..........
```

2.2. Calcul d'une somme

2.2.1. ESSEC II 2025

Contexte

Pour tout $x \in [0, +\infty[$, on pose $h(x) = \begin{cases} \dfrac{\mathrm{e}^x - 1}{x} & \text{si } x > 0 \\ 1 & \text{si } x = 0 \end{cases}$.

On définit alors la fonction g sur $\mathbb{R}^+$ par $g(x) = \mathrm{e}^{-x} \displaystyle\int_0^x h(t)\mathrm{d}t$.

On admet que pour tout $n \in \mathbb{N}^*$ et $x \in [0, n]$,

$$\mathrm{e}^{-x}\left(\sum_{k=1}^{n} \frac{x^k}{k!k}\right) \leqslant g(x) \leqslant \mathrm{e}^{-x}\left(\sum_{k=1}^{n} \frac{x^k}{k!k}\right) + \mathrm{e}^{-x}\frac{x^{n+1}}{(n+1)!}$$

Compléter le programme suivant pour qu'il trace la partie de la courbe de g comprise entre les abscisses $\ln(3)$ et 2, les valeurs de g étant calculées à 10^{-4} près :

```
1  X = np.linspace(np.log(3),2,100)
2  Y = []
3  for x in X:
4      n = 2; s = x + x**2/4; d = x**3/6
5      while d * np.exp(-x) > 0.0001:
6          n = n + 1
7          s = s + ...
8          d = d*x/...
9      Y.append(s*...)
10 plt.plot(X,Y)
11 plt.grid()
12 plt.show()
```

2.2.2. EDHEC 2025

Contexte

Soit $n \in \mathbb{N}^*$.

On considère une variable aléatoire X_n admettant une espérance $\mathbb{E}(X_n)$ donnée par :

$$\mathbb{E}(X_n) = \frac{n}{n+1}\sum_{p=1}^{n} \frac{1}{p}$$

Compléter le script suivant afin qu'il permette de calculer et d'afficher $\mathbb{E}(X_n)$:

```
1  n=int(input('entrez la valeur de n :'))
2  v=np.arange(1,n+1)
3  E=------
4  print(E)
```

2.2.3. ECRICOME 2023

Contexte

On considère la fonction f définie sur $]0,+\infty[$ par :

$$\forall x \in]0,+\infty[, \qquad f(x) = \frac{e^{\frac{x}{2}}}{\sqrt{x}}$$

On admet que f est strictement décroissante sur $]0,1[$ et que, pour tout entier n supérieur ou égal à 2, l'équation $f(x) = n$, d'inconnue x dans $]0,1[$, possède exactement une solution u_n.
On dispose d'une fonction **Python** `approx_u(n, eps)` qui renvoie une valeur approchée de u_n à `eps` près.

Écrire une fonction en langage Python, nommée `sp`, prenant en entrée un entier `N` supérieur ou égal à 2 et un réel strictement positif `eps` et renvoyant une valeur approchée de la somme $\sum_{n=2}^{N} u_n$ à `eps` près.

2.2.4. HEC/ESSEC I 2021

Contexte

On considère une population d'effectif infini dans laquelle un individu donné est infecté le jour 0 par un virus contagieux.
Soit $d \in \mathbb{N}^*$. On suppose que :

- tout individu infecté par le virus est immédiatement contagieux et sa contagiosité ne dure que $(d+1)$ jours, du jour n où il est infecté jusqu'au jour $(n+d)$ $(n \in \mathbb{N})$;
- une fois infectés, les individus présentent un même profil de contagiosité donné par un $(d+1)$-uplet $(\alpha_0, \alpha_1, \ldots, \alpha_d)$ qui dépend généralement de facteurs biologiques.

Pour tout $k \in [\![0, d]\!]$, on dit que α_k est la contagiosité de tout individu ayant éte infecté k jours plus tôt.
Autrement dit, on peut considérer que α_k, lié à la nature du virus, détermine la proportion d'individus contaminés par un individu infecté, parmi tous ceux avec lesquels il est en contact k jours après sa contamination.
Finalement, les réels α_0, α_1, …, α_d sont tels que, pour tout $k \in [\![0, d]\!]$, $\alpha_k \in]0, 1[$ et on note $\alpha = \sum_{k=0}^{d} \alpha_k$, ce qui signifie que α est la contagiosité globale d'un individu infecté sur toute la période où il est infecté.
On utilise les notations et définitions de la partie 1 avec $J = \mathbb{R}^+$.
On suppose que les variables aléatoires qui interviennent par la suite sont définies sur l'espace $(\Omega, \mathscr{A}, \mathbb{P})$.

- Pour tout $n \in \mathbb{N}$, on note R_n la variable aléatoire qui désigne le nombre moyen de contacts réalisés le jour n par un individu contagieux ce jour-là.
  On suppose, pour tout $n \in \mathbb{N}$, l'existence de $\mathbb{E}(R_n)$ et on pose $r_n = \mathbb{E}(R_n)$.
- Pour tout $n \in \mathbb{N}$, on note Z_n la variable aléatoire égale au nombre total d'individus qui sont infectés et donc deviennent contagieux le n-ième jour. Par exemple, $Z_0 = 1$.
- Pour tout $n \in \mathbb{N}$, on note I_n la variable aléatoire égale à la contagiosité globale de la population le n-ième jour, définie par :

$$I_n = \sum_{k=0}^{\min(n,d)} \alpha_k \, Z_{n-k} \qquad (*)$$

- On suppose enfin que, pour tout $n \in \mathbb{N}$, I_n et R_n sont indépendantes et que si l'on pose $Y_n = R_n \, I_n$, on a :

$$Z_{n+1} \text{ suit la loi } \mathcal{P}(Y_n)$$

  où $\mathcal{P}$ désigne la loi de Poisson. Ainsi la loi de Z_{n+1} ne dépend que des lois de R_n et de I_n.

On admet que, pour tout $n \in \mathbb{N}$, $z_n = \mathbb{E}(Z_n)$ existe et vérifie la relation de récurrence :

$$z_{n+1} = r_n \sum_{k=0}^{\min(n,d)} \alpha_k \, z_{n-k}$$

*Programmation de z_n avec **Python**.*

On suppose que la suite $(r_n)_{n\in\mathbb{N}}$ vérifie, pour tout $n \in \mathbb{N}$, $r_n = \dfrac{n+2}{n+1}$.
On note Δ la matrice ligne $(\alpha_0 \cdots \cdots \alpha_d)$.
Écrire une fonction **Python** d'entête `def z(Delta,n) :` qui calcule z_n si `Delta` représente la matrice ligne Δ. Si nécessaire, on pourra utiliser l'instruction `len(Delta)` qui donne le nombre d'éléments de `Delta`.

2.2.5. ESSEC II 2019

Contexte

Soit A un ensemble fini non vide. On dit que X est une variable aléatoire dont la loi est à support A, si X est à valeurs dans A et si pour tout $x \in A$: $\mathbb{P}([X = x]) > 0$.
Soit X une variable aléatoire de loi à support $\{0, 1, 2, \ldots, n\}$ où n est un entier naturel. On appelle **entropie** de X le réel :

$$H(X) \;=\; -\sum_{k=0}^{n} \mathbb{P}([X = k]) \log_2\big(\mathbb{P}([X = k])\big)$$

On souhaite écrire une fonction en **Python** pour calculer l'entropie d'une variable aléatoire X dont le support de la loi est de la forme $A = \{0, 1, \ldots, n\}$ où n est un entier naturel. On suppose que le vecteur `P` de **Python** est tel que pour tout k de A, `P[k]` $= \mathbb{P}([X = k])$. Compléter la fonction ci-dessous d'argument `P` qui renvoie l'entropie de X, c'est-à-dire $-\sum_{k=0}^{n} \mathbb{P}([X = k]) \log_2\big(\mathbb{P}([X = k])\big)$.

```
1 import numpy as np
2 def Entropie(P):
3 ...
```

Si nécessaire, on pourra utiliser l'instruction `len(P)` qui donne le nombre d'éléments de `P`.

2.3. Calcul d'un produit

2.3.1. HEC/ESSEC I 2022

Contexte

On considère une variable aléatoire à densité T et un entier $a \geqslant 2$ tels que :

$$\forall t \in [0, a[,\ F_T(t) = 1 - \exp\Big(- \gamma_{\lfloor t \rfloor + 1}\big(t - \lfloor t \rfloor\big)\Big)\ \exp\left(- \sum_{k=1}^{\lfloor t \rfloor} \gamma_k \right) \quad (2)$$

où les γ_k sont des constantes fixées.

On suppose que le vecteur **Python** `gammaTab` contient les valeurs $\gamma_1, \ldots, \gamma_a$.
Compléter la fonction suivante pour qu'elle renvoie la valeur de $F_T(t)$ obtenue dans l'égalité (2) si l'on suppose que `t` contient une valeur de l'intervalle $[0, a[$:

```
1  def F(t, gammaTab):
2      produit = ...
3      i = ...
4      for k in range(i):
5          produit = produit * np.exp(-gammaTab[k])
6      return 1 - np.exp(-gammaTab[i] * (...)) * produit
```

2.3.2. EDHEC 2019

Contexte

Pour tout entier naturel n, on pose $u_n = \displaystyle\int_0^1 (1-t^2)^n\, dt$. On a donc, en particulier : $u_0 = 1$.
On admet que :

$$\forall n \in \mathbb{N},\ u_n = \frac{4^n\,(n!)^2}{(2n+1)!}$$

On admet que, si `t` est un vecteur, la commande `np.prod(t)` renvoie le produit des éléments de `t`.
Compléter le script **Python** suivant afin qu'il permette de calculer et d'afficher la valeur de u_n pour une valeur de n entrée par l'utilisateur.

```
1  n = int(input('entrez une valeur pour n :'))
2  x = np.arange(1, n+1)
3  m = 2*n + 1
4  y = np.arange(1, m+1)
5  v = _______________
6  w = _______________
7  u = ________ * v**2 / w
8  print(u)
```

2.4. Calcul du n^{e} terme d'une suite récurrente

2.4.1. EDHEC 2024

Contexte

On pose, pour tout $n \in \mathbb{N}$, $u_n = \displaystyle\int_0^1 \frac{x^n}{4-x^2}\,dx$ et on a en particulier $u_0 = \displaystyle\int_0^1 \frac{1}{4-x^2}\,dx$.
On admet que :

- $u_0 = \dfrac{1}{4}\ln(3)$ et $u_1 = \ln\left(\dfrac{2}{\sqrt{3}}\right)$;
- pour tout $n \in \mathbb{N}$, $4u_n - u_{n+2} = \dfrac{1}{n+1}$.

Compléter la fonction **Python** ci-dessous afin qu'elle renvoie la valeur de u_n à l'appel de `suite(n)`.

```
def suite(n):
    if (-1)**n==1:
        u=np.log(3)/4
        for k in range(2,n+1,2):
            u=4*u-...
    else:
        u=np.log(2/np.sqrt(3))
        for k in range(3,n+1,2):
            u=4*u-...
    return u
```

2.4.2. EDHEC 2023

Contexte

On considère la fonction f définie par :

$$\forall t \in \mathbb{R},\ f(t) = \frac{1}{1+\mathrm{e}^t}$$

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par la donnée de $u_0 = 0$ et par la relation de récurrence $u_{n+1} = f(u_n)$, valable pour tout entier naturel n.

Compléter la fonction **Python** ci-dessous afin qu'elle renvoie, pour une valeur donnée de n, la valeur de u_n à l'appel de `suite(n)` :

```
def suite(n):
    u = ------
    for k in range(1, n+1):
        u = -----
    return u
```

2.4.3. EML 2023

Contexte

Pour $x \in]0,+\infty[$ on pose : $f(x) = \dfrac{\mathrm{e}^{-x}}{x}$.
On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 1$ et par la relation de récurrence $u_{n+1} = f(u_n)$, valable pour tout entier naturel n. On admet que chaque terme de la suite $(u_n)_{n\in\mathbb{N}}$ est correctement défini et strictement positif.

Écrire une fonction **Python** qui a pour argument un entier n et qui renvoie la valeur de u_n.

2.4.4. EML 2019

Contexte

On considère la fonction f définie sur $]0,+\infty[$ par : $\forall t \in]0,+\infty[,\ f(t) = t + \dfrac{1}{t}$.
On introduit la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par :

$$u_1 = 1 \quad \text{et} \quad \forall n \in \mathbb{N}^*,\ u_{n+1} = u_n + \frac{1}{n^2\, u_n} = \frac{1}{n}\, f(n\, u_n)$$

On admet que chaque terme de la suite $(u_n)_{n\in\mathbb{N}}$ est correctement défini et strictement positif.

Recopier et compléter les lignes 3 et 4 de la fonction **Python** suivante afin que, prenant en argument un entier n de $\mathbb{N}^*$, elle renvoie la valeur de u_n.

```
1  def suite(n):
2      u = 1
3      for k in _____________:
4          u = ___________________
5      return u
```

2.4.5. EML 2018

Contexte

On pose : $u_0 = 4 \quad \text{et} \quad \forall n \in \mathbb{N},\ u_{n+1} = \ln(u_n) + 2$.
On admet que chaque terme de la suite $(u_n)_{n\in\mathbb{N}}$ est correctement défini et strictement positif.

Écrire une fonction **Python** d'en-tête `def suite(n):` qui, prenant en argument un entier n de $\mathbb{N}$, renvoie la valeur de u_n.

2.4.6. ECRICOME 2018

Contexte

Pour tout entier naturel n non nul, on pose : $u_n = \displaystyle\sum_{k=1}^{n} \frac{1}{k} - \ln(n)$.

Écrire une fonction **Python** d'en-tête `def u(n):` qui prend en argument un entier naturel n non nul et qui renvoie la valeur de u_n.

2.5. Calcul d'une valeur approchée de la limite d'une suite

2.5.1. EDHEC 2023

Contexte

On considère la fonction f définie par :

$$\forall t \in \mathbb{R},\ f(t) = \frac{1}{1+\mathrm{e}^t}$$

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par la donnée de $u_0 = 0$ et par la relation de récurrence $u_{n+1} = f(u_n)$, valable pour tout entier naturel n.
On dispose d'une fonction **Python** `suite(n)` qui renvoie le terme u_n.
On admet que (u_n) converge vers un réel a et que u_n est une valeur approchée de a à moins de 10^{-3} près dès que n vérifie $4^n \geqslant 2000/3$.

Écrire un programme **Python**, utilisant la fonction `suite(n)`, qui calcule et affiche une valeur approchée de a à moins de 10^{-3} près.

2.5.2. EDHEC 2022

Contexte

Pour tout entier naturel n supérieur ou égal à 1, on pose :

$$u_n = \int_0^1 \frac{x}{n(x+n)}\,dx, \qquad S_n = \sum_{k=1}^{n} u_k, \qquad T_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n)$$

On admet que :
- Pour tout entier naturel n non nul :

$$S_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n+1)$$

- Les suites (S_n) et (T_n) sont adjacentes ((S_n) est croissante, (T_n) est décroissante). On note γ leur limite commune.

1. Préciser ce que représente S_n pour γ lorsque $T_n - S_n$ est inférieur ou égal à 10^{-3}.

2. Déterminer $T_n - S_n$, puis compléter le script **Python** suivant afin qu'il affiche une valeur approchée de γ à 10^{-3} près.

```
1 n = 1
2 s = 1 - np.log(2)
3 while ----:
4     n = ----
5     s = s + ----
6 print(----)
```

2.5.3. EML 2019

Contexte

On considère la fonction f définie sur $]0,+\infty[$ par : $\forall t \in]0,+\infty[,\ f(t) = t + \dfrac{1}{t}$.
On introduit la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par :

$$u_1 = 1 \quad \text{et} \quad \forall n \in \mathbb{N}^*,\ u_{n+1} = u_n + \frac{1}{n^2\, u_n} = \frac{1}{n}\, f(n\, u_n)$$

On dispose d'une fonction **Python** `suite(n)`, prenant en argument un entier n de $\mathbb{N}^*$, et renvoyant la valeur de u_n.
On admet que la suite (u_n) est convergente et on note ℓ sa limite.
On admet que, pour tout entier p supérieur ou égal à 2 :

$$0 \ \leqslant\ \ell - u_p \ \leqslant\ \frac{1}{p-1}$$

En déduire une fonction **Python** qui renvoie une valeur approchée de ℓ à 10^{-4} près.

2.5.4. EML 2018

Contexte

On pose : $u_0 = 4 \quad \text{et} \quad \forall n \in \mathbb{N},\ u_{n+1} = \ln(u_n) + 2$.
On admet que chaque terme de la suite $(u_n)_{n\in\mathbb{N}}$ est correctement défini et strictement positif.
On dispose d'une fonction **Python** `suite(n)` qui prend en entrée un entier naturel n et renvoie u_n.
On admet que : $\forall n \in \mathbb{N},\ 0 \leqslant u_n - b \leqslant \dfrac{1}{2^{n-1}}$.

Recopier et compléter la ligne 3 de la fonction **Python** suivante afin que, prenant en argument un réel `epsilon` strictement positif, elle renvoie une valeur approchée de b à `epsilon` près.

```
1  def valeur_approchee(epsilon):
2      n = 0
3      while ..........
4          n = n + 1
5      return suite(n)
```

2.5.5. ECRICOME 2018

Contexte

Pour tout entier naturel n non nul, on pose : $u_n = \displaystyle\sum_{k=1}^{n} \frac{1}{k} - \ln(n)$.
On dispose d'une fonction **Python** `u(n)` qui prend en argument un entier naturel n non nul et qui renvoie la valeur de u_n.
On admet que la suite (u_n) admet une limite, notée γ, et que :

$$\forall n \in \mathbb{N}^*,\ \left|u_n - \gamma\right| \leqslant \frac{1}{n}$$

On rappelle que l'instruction `np.floor(x)` renvoie la partie entière d'un réel x. Expliquer l'intérêt et le fonctionnement du script ci-dessous :

```
1 eps = float(input('Entrer un réel strictement positif : '))
2 n = int(np.floor(1/eps)) + 1
3 print(u(n))
```

2.6. Calcul et stockage des n premiers termes d'une suite récurrente

2.6.1. ECRICOME 2022

Contexte

Pour tout réel $x > 0$, on pose :

$$g(x) \;=\; \exp\left(\left(2 - \frac{1}{x}\right)\ln(x)\right)$$

Soit $(u_n)_{n\in\mathbb{N}}$ la suite définie par son premier terme $u_0 > 0$ et la relation de récurrence :

$$\forall n \in \mathbb{N}, \quad u_{n+1} \;=\; g(u_n)$$

On admet que, pour tout entier naturel n, u_n existe et : $u_n > 0$.

Écrire une fonction **Python** qui prend en argument un réel `u0` et un entier `n` et renvoie sous forme de vecteur ligne la liste des $n+1$ premières valeurs de la suite $(u_n)_{n\in\mathbb{N}}$ de premier terme $u_0 =$ `u0`.

2.7. Suites récurrentes linéaires couplées

2.7.1. EDHEC 2021

Contexte

On considère un nombre réel a élément de $]0,1[$ et la matrice $M_a = \begin{pmatrix} 1 & 0 & 0 \\ 1-a & a & 0 \\ 0 & 1-a & a \end{pmatrix}$.

On admet que, pour tout entier naturel n, il existe un unique triplet de réels (u_n, v_n, w_n) tel que :

$$\forall n \in \mathbb{N}, \quad M_a^n = u_n M_a^2 + v_n M_a + w_n I$$

Plus précisément :

$$\begin{cases} u_0 = 0 \\ v_0 = 0 \\ w_0 = 1 \end{cases} \quad \text{et, pour tout } n \in \mathbb{N}, \quad \begin{cases} u_{n+1} = (2a+1)u_n + v_n \\ v_{n+1} = w_n - a(a+2)u_n \\ w_{n+1} = a^2 u_n \end{cases}$$

1. En utilisant les relations précédentes, expliquer pourquoi le script **Python** qui suit ne permet pas de calculer et d'afficher les valeurs de u_n, v_n et w_n lorsque n et a sont entrés par l'utilisateur. On pourra examiner attentivement la boucle « for ».

```
1  n = input('entrez une valeur pour n : ')
2  a = input('entrez une valeur pour a : ')
3  u = 0
4  v = 0
5  w = 1
6  for k in range(n):
7      u = (2 * a + 1) * u + v
8      v = -a * (a + 2) * u + w
9      w = a * a * u
10 print(w, v, u)
```

2. Modifier la boucle de ce script en conséquence.

2.8. Suites récurrentes linéaires doubles

2.8.1. EDHEC 2020

Contexte

Pour tout n de $\mathbb{N}$, on note :

$$I_n = \int_0^1 \frac{x^n}{(1+x)^2}\,dx$$

On admet que :

- $\forall n \in \mathbb{N},\ I_{n+2} + 2\,I_{n+1} + I_n = \dfrac{1}{n+1}$
- $I_0 = \dfrac{1}{2}$
- $I_1 = \ln(2) - \dfrac{1}{2}$

Compléter le script **Python** suivant pour qu'il permette le calcul de I_n (dans la variable `b`) et son affichage pour une valeur de n entrée par l'utilisateur.

```
1  n = int(input('donnez une valeur pour n : '))
2  a = 1/2
3  b = np.log(2) - 1/2
4  for k in range(2,n+1):
5      aux = a
6      a = ------
7      b = ------
8  print(b)
```

Contexte

Pour tout n de $\mathbb{N}$, on note :

$$J_n = \int_0^1 \frac{x^n}{1+x}\,dx$$

On admet que :

- $\forall n \in \mathbb{N},\ J_n + J_{n+1} = \dfrac{1}{n+1}$
- $J_0 = \ln(2)$
- $\forall n \in \mathbb{N}^*,\ I_n = n\,J_{n-1} - \dfrac{1}{2}$

Compléter le script **Python** suivant afin qu'il permette le calcul et l'affichage de I_n pour une valeur de n entrée par l'utilisateur.

```
1  n = int(input('donnez une valeur pour n : '))
2  J = np.log(2)
3  for k in range(1,n):
4      J = ------
5  print(--------)
```

2.9. Calcul du premier entier n vérifiant une propriété donnée

2.9.1. ESSEC II 2025

Écrire une fonction **Python**, `minimum(x,p)` qui renvoie le minimum de l'ensemble $\left\{k \in \mathbb{N} \mid x \leqslant \sum_{i=0}^{k} \frac{p^i}{i!}\mathrm{e}^{-p}\right\}$ lorsque $x \in [0,1[$ et $p \in]0,1[$.

2.9.2. EML 2025

Contexte

On s'intéresse à la suite récurrente $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 1$ et

$$\forall n \in \mathbb{N}, \quad u_{n+1} = u_n \mathrm{e}^{1/u_n}.$$

On admet que (u_n) est croissante et diverge vers $+\infty$.

Recopier et compléter le programme **Python** ci-dessous de sorte qu'il affiche le premier entier $n \in \mathbb{N}$ tel que $u_n \geqslant 10^6$.

```
import numpy as np
u = 1
n = 0
while ... :
    u = ...
    n = ...
print(...)
```

2.9.3. EML 2023

Contexte

Pour $x \in]0,+\infty[$ on pose : $f(x) = \dfrac{\mathrm{e}^{-x}}{x}$.
On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 = 1$ et par la relation de récurrence $u_{n+1} = f(u_n)$, valable pour tout entier naturel n. On admet que chaque terme de la suite $(u_n)_{n\in\mathbb{N}}$ est correctement défini et strictement positif.

1. Recopier et compléter la fonction **Python** suivante afin que l'appel `fonc_1(a)` renvoie le plus petit entier n tel que $u_n > a$.

```
def fonc_1(a):
    from numpy import exp
    u=1
    n=0
    while ..... :
        u = exp(-u)/u
        n=....
    return n
```

2. On considère maintenant la fonction **Python** :

```
def fonc_2(a):
    from numpy import exp
    u=1
    n=0
    while u>a :
        u = exp(-u)/u
        n=n+1
    return n
```

Les appels `fonc_1(10**6)` et `fonc_2(10**(-6))` donnent respectivement 6 et 5.
Qu'en déduire pour u_5 et u_6 ?
Commenter ce résultat en une ligne.

2.9.4. HEC/ESSEC I 2020

Contexte

On considère un événement G_n qui dépend de deux paramètres : $n \in \mathbb{N}^*$ et $p \in]0,1[$.
On admet que, pour tout $n \in \mathbb{N}^*$ et pour $p < \dfrac{1}{2}$:

$$\mathbb{P}(G_n) \;=\; \frac{p}{q} - \left(1 - \frac{p}{q}\right) \sum_{k=1}^{n} \binom{2k-1}{k} (pq)^k$$

où $q = 1 - p$.

1. Après avoir établi la formule $\binom{n}{k} = \dfrac{n}{k}\binom{n-1}{k-1}$ lorsque $k \in [\![1, n]\!]$, écrire une fonction **Python** qui calcule les coefficients binomiaux.

2. Écrire un script **Python** qui détermine n_p, le plus petit entier n tel que $\mathbb{P}(G_n) \leqslant \varepsilon$ pour $p < \dfrac{1}{2}$ et $\varepsilon > 0$ saisis au clavier par l'utilisateur.

2.9.5. EML 2017

Contexte

On considère la fonction $f :]0, +\infty[\to \mathbb{R}$ définie, pour tout x de $]0, +\infty[$, par :

$$f(x) = \mathrm{e}^x - \mathrm{e}\ln(x).$$

On considère la suite réelle $(u_n)_{n\in\mathbb{N}}$ définie par :

$$\begin{cases} u_0 = 2 \\ \forall n \in \mathbb{N},\ u_{n+1} = f(u_n) \end{cases}$$

On admet que la suite $(u_n)_{n\in\mathbb{N}}$ admet $+\infty$ pour limite.

Écrire un programme en **Python** qui, étant donné un réel A, renvoie un entier naturel N tel que $u_N \geqslant A$.

2.10. Codage d'une fonction d'une variable ou de deux variables

2.10.1. ECRICOME 2024

Contexte

Soit a un réel. On considère la fonction I_a définie par :

$$I_a(x) = \int_x^{+\infty} \mathrm{e}^{2a(x-t)-t^2}\, dt.$$

On considère une variable aléatoire X de loi normale d'espérance $-a$ et de variance $\frac{1}{2}$.
On dispose d'une fonction **Python** `estim_proba(a, x)` qui, prenant en arguments d'entrée les réels a et x, renvoie une estimation de la probabilité $\mathbb{P}([X \geqslant x])$.
On admet que, pour tout réel x, $I_a(x) = \sqrt{\pi}\mathrm{e}^{2ax+a^2}\mathbb{P}([X \geqslant x])$.

Écrire une fonction **Python**, nommée `approx_I`, prenant en arguments d'entrée les réels a et x et renvoyant une valeur approchée de $I_a(x)$.

2.10.2. HEC/ESSEC I 2023

Contexte

On définit la fonction γ sur $\mathbb{R}$ par :

$$\gamma(t) = \begin{cases} 1 & \text{si } t < 0 \\ 0 & \text{si } t > 1 \\ 1 - 3t^2 + 2t^3 & \text{si } t \in [0,1] \end{cases}$$

Écrire une fonction **Python** `gamma(t)` qui calcule et renvoie la valeur de $\gamma(t)$, t étant donné.

2.10.3. ECRICOME 2019

Contexte

Pour tout entier n non nul, on note h_n la fonction définie sur $\mathbb{R}_+^*$ par :

$$\forall x > 0, \quad h_n(x) \;=\; x^n + 1 + \frac{1}{x^n}$$

Écrire une fonction **Python** d'en-tête `def h(n,x):` qui renvoie la valeur de $h_n(x)$ lorsqu'on lui fournit un entier naturel n non nul et un réel $x \in \mathbb{R}_+^*$ en entrée.

2.10.4. EDHEC 2017

Contexte

On considère la fonction f qui à tout couple (x, y) de $\mathbb{R}^2$ associe le réel :

$$f(x,y) = x^4 + y^4 - 2\,(x-y)^2$$

Compléter la deuxième ligne du programme suivant afin de définir la fonction f en **Python**.

```
1  def f(x,y):
2      return ..............
```

2.11. Tracé du graphe d'une fonction

2.11.1. HEC/ESSEC I 2023

Contexte

On définit la fonction γ sur $\mathbb{R}$ par :

$$\gamma(t) = \begin{cases} 1 & \text{si } t < 0 \\ 0 & \text{si } t > 1 \\ 1 - 3t^2 + 2t^3 & \text{si } t \in [0,1] \end{cases}$$

On dispose d'une fonction **Python** `gamma(t)` qui calcule et renvoie la valeur de $\gamma(t)$, t étant donné.

Écrire un script qui affiche le graphe de γ sur le segment $[-1, 2]$ dans un repère.

2.11.2. ESSEC II 2023

Contexte

Soit $n \in \mathbb{N}^*$, $n \geqslant 2$. On considère une famille de variables aléatoires X_t, pour $t \in \mathbb{R}^+$, sur un espace probabilisé $(\Omega, \mathcal{A}, \mathbb{P})$, vérifiant les propriétés suivantes :

(H_1) Pour tout $t \geqslant 0$, $X_t(\Omega) = \{1, \dots, n\}$.

(H_2) Pour tout $r \in \mathbb{N}^*$ et $t_1 < t_2 < \cdots < t_r$ des réels positifs, $i_1, \dots, i_{r+1}$ des éléments de $\{1, \dots, n\}$ et s un réel positif, si $\mathbb{P}([X_{t_1} = i_1] \cap \cdots \cap [X_{t_r} = i_r]) \neq 0$,

$$\mathbb{P}_{[X_{t_1}=i_1]\cap\cdots\cap[X_{t_r}=i_r]}([X_{t_r+s} = i_{r+1}]) = \mathbb{P}_{[X_{t_r}=i_r]}([X_{t_r+s} = i_{r+1}])$$

(H_3) Pour tout $i \in \{1, \dots, n\}$, la fonction $f_i : t \mapsto \mathbb{P}([X_t = i])$ est définie, dérivable sur $\mathbb{R}^+$ et n'est pas la fonction nulle. On note S_i l'ensemble des réels positifs t tels que $f_i(t) \neq 0$.

(H_4) Pour tout $(i, j) \in \{1, \dots, n\}^2$, $i \neq j$ et $h \geqslant 0$, la fonction $t \mapsto \mathbb{P}_{[X_t=i]}([X_{t+h} = j])$ est constante sur son ensemble de définition S_i et il existe un réel positif que l'on note $\alpha_{i,j}$, tel que, si $t \in S_i$ et $h \in \mathbb{R}^+$,

$$\mathbb{P}_{[X_t=i]}([X_{t+h} = j]) = \alpha_{i,j}h + \underset{h\to 0}{o}(h)$$

(H_5) Pour tout $i \in \{1, \dots, n\}$ et $h \geqslant 0$, la fonction $t \mapsto \mathbb{P}_{[X_t=i]}([X_{t+h} = i])$ est constante sur son ensemble de définition S_i et il existe un réel négatif que l'on note $\alpha_{i,i}$, tel que, si $t \in S_i$ et $h \in \mathbb{R}^+$,

$$\mathbb{P}_{[X_t=i]}([X_{t+h} = i]) = 1 + \alpha_{i,i}h + \underset{h\to 0}{o}(h)$$

On note :

- L_t la matrice ligne d'ordre n, $\big(\mathbb{P}([X_t = 1]) \dots\dots \mathbb{P}([X_t = n])\big) = \big(f_1(t) \dots\dots f_n(t)\big)$
- G la matrice carrée d'ordre n dont les coefficients sont les $\alpha_{i,j}$, appelée **matrice génératrice du processus**
- $M(s)$ la matrice (appelée **matrice de transition**) d'élément générique

$$m_{i,j}(s) = \mathbb{P}_{[X_t=i]}([X_{t+s} = j])$$

pour $(i, j) \in \{1, \dots, n\}^2$, $s \geqslant 0$ et $t \in S_i$. D'après les hypothèses (H_4) et (H_5), $m_{i,j}(s)$ est indépendant de t.

On admet que pour tout $s \geqslant 0$, $L_s = L_0 M(s)$.
On veut simuler le processus à partir de la donnée de la matrice G et de L_0. On admet que pour $t \in [0, 100]$, on peut considérer que $M(t) = \left(I_n + \frac{t}{1000}G\right)^{1000}$.
On dispose d'une fonction **Python** `transition(t,G)` de paramètres `G` représentant la matrice génératrice carrée d'ordre n et `t`, qui renvoie la matrice $\left(I_n + \frac{t}{1000}G\right)^{1000}$.
On rappelle que si `M` est une matrice, représentée par un tableau `numpy`, `M[:,j]` désigne le vecteur des coefficients de la `j`-ème colonne de `M`, de même pour `M[i,:]` et la `i`-ème ligne de `M`.

Utiliser la fonction `transition(t,G)` pour écrire une fonction `traceLoi2Xt(G,L0,tmax)` qui trace, sur un même graphique, les graphes des fonctions $t \mapsto \mathbb{P}([X_t = i])$ sur le segment $[0, t_{\max}]$ pour i variant de 1 à n, `G` et `L0` représentant, respectivement, la matrice génératrice du processus et la ligne L_0.
On utilisera 1000 points pour les graphes.

2.12. Tracé d'une suite

2.12.1. EDHEC 2024

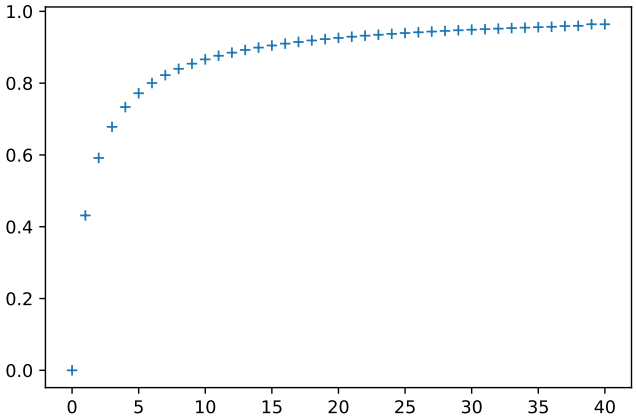
Contexte

On pose, pour tout $n \in \mathbb{N}$, $u_n = \displaystyle\int_0^1 \frac{x^n}{4-x^2}\,dx$ et on a en particulier $u_0 = \displaystyle\int_0^1 \frac{1}{4-x^2}\,dx$.
On dispose d'une fonction `suite(n)` qui calcule et renvoie u_n.

On considère le script suivant qui utilise la fonction `suite(n)` déclarée plus haut.

```
x=np.arange(0,41)
u=[] # liste vide
for n in range(41):
    u.append(3*n*suite(n))
plt.plot(x,u,'+')
plt.show()
```

Ce script renvoie le graphique suivant :



Laquelle des quatre conjectures suivantes peut-on émettre quant au comportement de la suite $(u_n)_{n\in\mathbb{N}}$ au voisinage de $+\infty$?

❶ $u_n \underset{n\to+\infty}{\sim} 3n$.

❷ $\lim\limits_{n\to+\infty} u_n = 1$.

❸ $u_n \underset{n\to+\infty}{\sim} \dfrac{1}{3n}$.

❹ $u_n \underset{n\to+\infty}{\sim} \dfrac{1}{n}$.

2.12.2. ECRICOME 2019

Contexte

Pour tout entier n non nul, on note h_n la fonction définie sur $\mathbb{R}_+^*$ par :

$$\forall x > 0, \quad h_n(x) \;=\; x^n + 1 + \frac{1}{x^n}$$

On admet que, pour tout entier naturel n non nul :

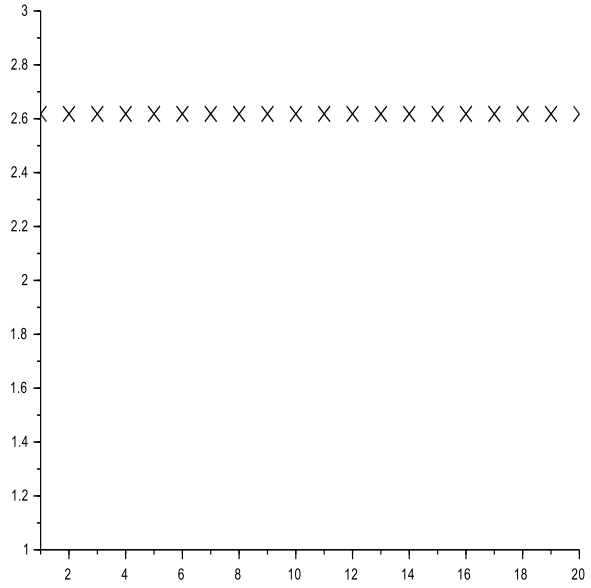
- la fonction h_n est strictement décroissante sur $]0,1[$ et strictement croissante sur $[1,+\infty[$,
- l'équation $h_n(x) = 4$ admet exactement deux solutions, notées u_n et v_n et vérifiant : $0 < u_n < 1 < v_n$.

On dispose d'une fonction **Python** nommée `approxv(n)` qui renvoie une valeur approchée à 10^{-5} près de v_n par la méthode de dichotomie lorsqu'on lui fournit un entier $n \geqslant 1$ en entrée.

À la suite de la fonction `approxv`, on écrit le code suivant :

```
1  X = np.arange(1,21)
2  Y = np.zeros(20)
3  for k in range(20):
4      Y[k] = approxv(k+1)**(k+1)
5  plt.plot(X,Y,'x')
6  axes = plt.gca()
7  axes.set_ylim([1, 3])
```

À l'exécution du programme, on obtient la sortie graphique suivante :



Expliquer ce qui est affiché sur le graphique ci-dessus.
Que peut-on conjecturer ?

2.13. Tracé d'une trajectoire de système différentiel

2.13.1. EML 2024

Contexte

On s'intéresse au système différentiel :

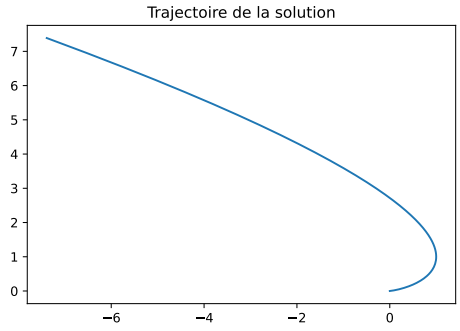
$$(S) : \begin{cases} x'(t) &= -x(t) + y(t) \\ y'(t) &= -y(t) \end{cases}$$

où x et y désignent des fonctions définies et dérivables sur $\mathbb{R}$ à valeurs dans $\mathbb{R}$.
On admet qu'il existe une unique solution (x, y) de (S) telle que $x(0) = 1$ et $y(0) = 1$, donnée par :

$$\forall t \in \mathbb{R}, \begin{cases} x(t) &= \mathrm{e}^{-t} + t\mathrm{e}^{-t} \\ y(t) &= \mathrm{e}^{-t} \end{cases}$$

Recopier et compléter le programme en langage **Python** ci-dessous de manière à ce qu'il produise le graphique sur la droite représentant la trajectoire $t \mapsto (x(t), y(t))$ pour $t \in [-2, 10]$.
On rappelle que la commande `np.linspace(-2,10,200)` crée une liste de 200 valeurs régulièrement espacées allant de -2 à 10.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  T = np.linspace(-2,10,200)
5  x = [... for t in T]
6  y = [... for t in T]
7
8  plt.title('Trajectoire de la solution')
9  plt.plot(...)
10 plt.show()
```



3. Graphes

3.1. Algorithme de coloration des graphes

3.1.1. EML 2025

Contexte

Soit $n \geqslant 1$ un entier, on considère un graphe non orienté G donné par sa matrice d'adjacence $A \in \mathscr{M}_n(\mathbb{R})$. On note $\mathscr{S} = \{s_0, \ldots, s_{n-1}\}$ l'ensemble des sommets de G, dans les programmes informatiques on confondra un sommet s_i avec son numéro i. On dit que deux sommets sont voisins s'ils sont distincts et reliés par une arête.
Une coloration de G est une application $c : \mathscr{S} \to \mathbb{N}$ telle que $c(s_i) \neq c(s_j)$ si les sommets s_i et s_j sont voisins. Dans cette définition, $\mathbb{N}$ représente l'ensemble des « couleurs » disponibles, la coloration c attribue à chaque sommet une « couleur » de sorte que deux sommets voisins soient de « couleurs » différentes.
Le graphe G admet la coloration triviale donnée par $c(s_i) = i$ pour tout $i \in [\![0, n-1]\!]$, il peut cependant admettre une coloration nécessitant moins de n « couleurs ». Ainsi, le graphe à cinq sommets ci-dessous admet la coloration à trois « couleurs » définie par : $c(s_0) = 0, c(s_1) = 1, c(s_2) = 0, c(s_3) = 1, c(s_4) = 2$.

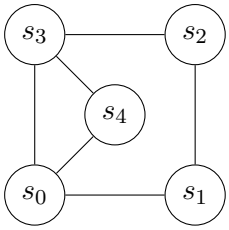


Figure 1 : Un graphe d'ordre cinq

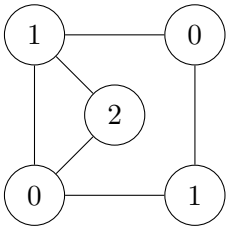


Figure 2 : Le graphe colorié avec trois « couleurs » (0, 1 et 2)

Les questions suivantes ont pour but de réaliser un programme **Python** qui renvoie une coloration d'un graphe G quelconque, en essayant de minimiser le nombre de couleurs utilisées. On commence par rédiger deux fonctions auxiliaires, « `voisins` » et « `min_ext` », qui serviront pour la fonction finale « `coloration` ». On suppose que la matrice d'adjacence A de G est définie à l'aide de la commande « `np. array` ».

1. Recopier et compléter le programme **Python** ci-dessous de manière à ce qu'il définisse une fonction « `voisins` », prenant en arguments la matrice d'adjacence A et un entier $i \in [\![0, n-1]\!]$, et renvoyant la liste des sommets voisins de s_i.

```
1  def voisins(A,i):
2      n = len(A[i])
3      V = []
4      for j in range(n):
5          if j!= i and ... :
6              V.append(...)
7      return(V)
```

2. Rédiger en **Python** une fonction « `min_ext` » qui prend en argument une liste d'entiers naturels L, et qui renvoie le plus petit entier naturel n'appartenant pas à L (par exemple, si $L = [1, 0, 3]$ alors la commande « `min_ext(L)` » renvoie 2). On pourra transcrire en langage **Python** l'algorithme suivant :

> On affecte à une variable m la valeur 0.
>
> Tant que m appartient à la liste L :
> |On augmente de 1 la valeur de m.
>
> On renvoie m.

3. À l'aide des fonctions introduites précédemment on rédige maintenant une fonction « `coloration` » prenant en argument la matrice d'adjacence $A \in \mathscr{M}_n(\mathbb{R})$ d'un graphe G, et renvoyant une coloration de G sous la forme d'une liste d'entiers $C = [C_0, \ldots, C_{n-1}]$, où C_i désigne la « couleur » du sommet s_i pour tout $i \in [\![0, n-1]\!]$.
On construit cette fonction selon l'algorithme « glouton » ci-dessous :

> On affecte à la variable n le nombre de sommets de G.
>
> On affecte à la variable C la liste $[0, 1, ..., n-1]$.
>
> Pour i allant de 1 à $n-1$:
> | On affecte à la variable « `C_voisins` » la liste des « couleurs » des
> | sommets voisins de s_i
> | On affecte à C_i le plus petit entier naturel qui n'est pas élément de la
> | liste « `C_voisins` ».
>
> On renvoie la liste C.

Recopier et compléter la fonction « `coloration` » ci-dessous.

```
1  def coloration(A):
2      n = len(A[0])
3      C = ...
4      for i in range(1,n):
5          C_voisins = [ ... for j in ... ]
6          C[i] = min_ext(...)
7      return(C)
```

12. On note A la matrice d'adjacence du graphe G représenté en figure 3 ci-contre.

a) Donner la liste obtenue en exécutant la commande « `coloration(A)` ».

b) Le graphe G admet-il une coloration à trois couleurs ? Si oui, exhiber une telle coloration.

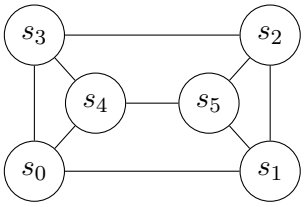


Figure 3 : Le graphe G

3.2. Algorithme de recherche du plus court chemin

3.2.1. ECRICOME 2023

Contexte

Soient p un entier naturel non nul et G un graphe non pondéré orienté à p sommets. On note s_0, $s_1, \ldots, s_{p-1}$ les sommets de G.
Soit s un sommet de G. On dit que le sommet t est un voisin de s quand $s \neq t$ et (s,t) est une arête du graphe.
Comme le graphe est orienté, si t est un voisin de s, alors s n'est pas forcément un voisin de t.
On appelle liste d'adjacence du graphe G, une liste de p sous-listes telle que, pour tout entier k de $[\![0, p-1]\!]$, la sous-liste située à la position k contient tous les numéros des sommets voisins de s_k.

On cherche à écrire une fonction en langage **Python** permettant d'obtenir la longueur du plus court chemin menant d'un sommet de départ s_i à chaque sommet du graphe G.
On souhaite pour cela appliquer un algorithme faisant intervenir les variables suivantes :

- Une liste `distances` à p éléments, où l'élément situé à la position k sera égal, à la fin de l'algorithme, à la longueur du plus court chemin menant du sommet de départ s_i au sommet s_k.
- Une liste `a_explorer` contenant tous les sommets restant à traiter.
- Une liste `marques` contenant tous les sommets déjà traités.

Nous donnons ci-dessous la description de l'algorithme :

- Initialisation des trois listes décrites ci-dessus :
  - × Initialement, chaque élément de la liste `distances` est égal à p, à l'exception du sommet s_i, auquel on affecte la distance 0.
  - × La liste `marques` ne contient initialement que le numéro du sommet de départ s_i.
  - × La liste `a_explorer` ne contient initialement que le numéro du sommet de départ s_i.
- Tant que la liste `a_explorer` n'est pas vide, on répète les opérations suivantes :
  - × Nommer `s` le premier sommet de la liste `a_explorer`, et le retirer de cette liste.
  - × Pour chaque voisin `v` du sommet `s` : si `v` n'est pas dans la liste `marques`, on l'ajoute à la fin de la liste `a_explorer`, et on lui affecte une distance égale à `distances[s]+1`.

a) On considère le graphe orienté G dont la matrice d'adjacence est la matrice $A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$.

Donner la valeur de la liste `distances` à l'issue de l'exécution de l'algorithme décrit ci-dessus, lorsqu'on l'applique au graphe G en choisissant s_1 comme sommet de départ.

b) Recopier et compléter la fonction suivante, prenant en entrée la liste d'adjacence `L` du graphe G et le numéro `i0` du sommet de départ s_i, et renvoyant la liste `distances` après exécution de l'algorithme décrit ci-dessus.

```
1  def parcours(L, i0):
2      p = len(L)
3      distances = .....
4      distances[i0] = 0
5      a_explorer = .....
6      marques = .....
7      while .....:
8          s = .....
9          .....
10         for v in .....:
11             if v not in marques :
12                 marques.append(v)
13                 .....
14                 .....
15     return distances
```

c) Modifier la fonction précédente pour qu'elle renvoie la liste de tous les sommets s pour lesquels il existe un chemin menant du sommet de départ s_i au sommet s.

3.3. Création de la liste d'adjacence d'un graphe

3.3.1. ECRICOME 2023

Contexte

Soient p un entier naturel non nul et G un graphe non pondéré orienté à p sommets. On note s_0, $s_1, \ldots, s_{p-1}$ les sommets de G.
Soit s un sommet de G. On dit que le sommet t est un voisin de s quand $s \neq t$ et (s, t) est une arête du graphe.
Comme le graphe est orienté, si t est un voisin de s, alors s n'est pas forcément un voisin de t.
On appelle liste d'adjacence du graphe G, une liste de p sous-listes telle que, pour tout entier k de $[\![0, p-1]\!]$, la sous-liste située à la position k contient tous les numéros des sommets voisins de s_k.

Écrire une fonction en langage **Python**, nommée `matrice_vers_liste`, prenant en entrée la matrice d'adjacence `A` d'un graphe G (définie sous forme de listes de listes) et renvoyant la liste d'adjacence de G.

3.4. Comptage du nombre de triangles dans un graphe aléatoire

3.4.1. HEC/ESSEC I 2024

Contexte

Soit n un entier supérieur ou égal à 3 et p un réel appartenant à $]0,1[$.
Pour générer des graphes non orientés de manière aléatoire, on se donne :

- $S = [\![0, n-1]\!]$, les sommets du graphe ;
- pour toute paire de sommets $\{u, v\}$ avec $u < v$, $T_{u,v}$ une variable aléatoire de Bernoulli de paramètre p.
  Les variables aléatoires $T_{u,v}$ pour $\{u, v\}$ décrivant les paires de sommets avec $u < v$, sont supposées indépendantes ;
- les arêtes d'un graphe G ainsi généré sont les paires $\{u, v\}$ telles que $T_{u,v} = 1$ si $u < v$ ou $T_{v,u} = 1$ si $v < u$.

On note $\mathcal{T}$ l'ensemble des parties $\{u, v, w\}$ à trois éléments de l'ensemble des sommets, r le nombre de ses éléments et on pose

$$\mathcal{T} = \{t_1, \ldots, t_r\}$$

Étant donné $t = \{u, v, w\}$, un élément de $\mathcal{T}$, on dit que t est un triangle dans un graphe G généré aléatoirement si $\{u, v\}$, $\{v, w\}$ et $\{w, u\}$ sont des arêtes de G.

On se donne un graphe G généré par le procédé décrit dans le préambule.
On définit la fonction `supprimeDer(L)` qui si `L` est la liste des listes d'adjacence du graphe G dont les sommets sont $0, 1, \ldots, n-1$, modifie `L` afin qu'elle devienne la liste des listes d'adjacence du graphe G', dont les sommets sont $0, 1, \ldots, n-2$, obtenu en supprimant dans G le sommet $n-1$ et les arêtes contenant ce sommet.

```
1  def supprimeDer(L):
2      s = len(L) - 1
3      L.pop() # supprime le dernier élément de la liste L
4      for a in L:
5          if s in a:
6              a.remove(s) # supprime s dans la liste a
```

1. Compléter la fonction suivante pour qu'elle retourne le nombre de triangles dont un des sommets est le sommet `s` dans le graphe G dont la liste des listes d'adjacence est `L` :

```
1  def triangle2s(s, L):
2      cpt = 0
3      adj = L[s]
4      for i in range(len(adj)):
5          for j in range(..., len(adj)) :
6              if ... in L[...]:
7                  cpt += 1
8      return cpt
```

2. Écrire une fonction `nbTriangles(L)`, utilisant les deux fonctions précédentes, qui retourne le nombre de triangles du graphe G dont la liste des listes d'adjacence est représentée par L.

3.5. Degrés des sommets d'un graphe

3.5.1. ESSEC II 2024

Contexte

Soit m, n et p des entiers tels que $m \geqslant 1$, $n \geqslant p \geqslant 2$.
On suppose que $A = (a_{i,j})_{1 \leqslant i,j \leqslant n}$ est la matrice d'adjacence d'un graphe $G(A)$, non orienté sans boucle, à n sommets $1, \ldots, n$, m arêtes et $n - p$ sommets isolés, c'est-à-dire de degré 0.

Écrire une fonction **Python** `degMax(A)` qui renvoie le maximum des degrés des sommets du graphe $G(A)$, celui-ci étant donné par sa matrice d'adjacence sous la forme du tableau `numpy A`.

4. Probabilités

4.1. Simulation de variables aléatoires discrètes à l'aide de la bibliothèque pandas

4.1.1. HEC/ESSEC I 2023

Contexte

On considère X une variable aléatoire à densité de fonction de répartition F et de densité de probabilité f qui dépendent d'un paramètre inconnu θ, où $\theta \in \Theta$, Θ un intervalle de $\mathbb{R}$.
Soit a un point de continuité de f, fixé. On souhaite estimer $f(a)$.
Par exemple, si X suit la loi exponentielle de paramètre θ et $a > 0$, on souhaite estimer $\theta \mathrm{e}^{-\theta a}$.
On dispose pour tout $\theta \in \Theta$, d'une suite de variables aléatoires $(X_i)_{i\geqslant 1}$ indépendantes de même loi que X.
On choisit une suite $(h_n)_{n\geqslant 1}$ de réels strictement positifs tels que :

$$\lim_{n\to+\infty} h_n = 0 \text{ et } \lim_{n\to+\infty} nh_n = +\infty$$

Pour tout $n \in \mathbb{N}^*$, et $\omega \in \Omega$, on définit :

$C_n(\omega)$ comme le nombre d'indices $i \in [\![1, n]\!]$ tels que $X_i(\omega) \in]a - h_n, a + h_n]$

et $f_n(\omega) = \dfrac{1}{2nh_n} C_n(\omega)$.

1. Après avoir exécuté `import pandas as pd`, quelle(s) instruction(s) permet(tent) de lire dans le fichier `stats.csv` les valeurs de la colonne `salaire` et d'affecter cette série `pandas` obtenue à une variable `échantillon` ?

On supposera que le fichier `stats.csv` se trouve dans le répertoire de travail.

2. On souhaite calculer et afficher $f_n(\omega)$ pour a donné, lorsque la réalisation d'un échantillon $(X_1(\omega), \ldots, X_n(\omega))$ de la loi de X est représentée en **Python** par `échantillon` et, pour tout $n \in \mathbb{N}^*$, $h_n = \dfrac{1}{\sqrt{n}}$.

Compléter le script suivant pour qu'il réalise cette tâche.

```
a = float(input('a='))
n = échantillon.count()
h = 1 / np.sqrt(n)
C = 0
for i in range(n):
    if ... and ...:
        ... += 1
print(C / ...)
```

4.2. Simulation de variables aléatoires discrètes liées à une expérience concrète

4.2.1. EDHEC 2025

Contexte

La lettre n désigne un entier naturel non nul.
On dispose de $n+1$ urnes, numérotées de 1 à $n+1$, et contenant chacune n boules.
Pour tout k de $[\![1, n+1]\!]$, l'urne numéro k contient $k-1$ boules noires, les autres boules étant blanches (ainsi, l'urne numérotée 1 ne contient que des boules blanches et l'urne numérotée $n+1$ ne contient que des boules noires).
L'épreuve consiste à choisir une urne au hasard et à y effectuer indéfiniment des tirages au hasard d'une boule, avec remise de la boule tirée dans l'urne dont elle provient après chaque tirage.
Pour tout k de $[\![1, n+1]\!]$, on note U_k l'événement : « On a choisi l'urne numérotée k ».
On appelle X_n la variable aléatoire qui prend la valeur 0 si l'on n'obtient aucune boule blanche au cours de l'épreuve et qui prend la valeur j ($j \in \mathbb{N}^*$) si la première boule blanche apparaît au $j^{\text{ième}}$ tirage.
On rappelle les commandes **Python** suivantes qui permettent de simuler certaines variables discrètes usuelles :
`rd.randint(a,b+1)` simule une variable aléatoire suivant la loi uniforme sur $[\![a, b]\!]$.
`rd.binomial(n,p)` simule une variable aléatoire suivant la loi binomiale de paramètres n et p.
`rd.geometric(p)` simule une variable aléatoire suivant la loi géométrique de paramètre p.

1. Pour tout j de $[\![2, n+1]\!]$, on code les $j-1$ boules noires de l'urne numérotée j par les entiers de $[\![1, j-1]\!]$. Compléter alors la fonction **Python** suivante pour qu'elle renvoie la valeur prise par X_n lors de l'épreuve aléatoire décrite ci-dessus :

```
1  def varX(n):
2      k=------ # choix de l'urne
3      if k==n+1:
4          X=------
5      elif k==1:
6          X=------
7      else:
8          X=1
9          while rd.randint(1,n+1) <= ------:
10             X=------
11     return(X)
```

2. On admet que, pour tout k de $[\![1, n]\!]$, la loi de X_n, conditionnellement à l'événement U_k, est la loi géométrique de paramètre $\dfrac{n-k+1}{n}$.
En conservant, sans les écrire de nouveau, les 6 premières lignes de la fonction **Python** précédente, compléter les 3 lignes suivantes afin d'obtenir une nouvelle simulation de X_n :

```
7  else:
8      X=------
9  return(X)
```

4.2.2. EML 2024

Contexte

Soit N un entier naturel supérieur ou égal à 1. On dispose d'une urne contenant N boules numérotées de 1 à N, et on effectue une succession illimitée de tirages d'une boule avec remise dans l'urne. Pour tout $k \in \mathbb{N}^*$, on note X_k la variable aléatoire indiquant le numéro de la boule obtenue au $k-$ième tirage.
Pour tout entier $i \in [\![1, N]\!]$, on note T_i la variable aléatoire égale au nombre de tirages nécessaires pour obtenir i numéros distincts, ainsi $T_i = k$ si on a obtenu i numéros distincts lors des k premiers tirages, mais seulement $i-1$ numéros distincts lors des $k-1$ premiers tirages.
Exemple : on suppose $N = 4$, si les huit premiers tirages donnent

i	1	2	3	4	5	6	7	8
X_i	2	3	3	3	1	2	1	4

alors $T_1 = 1$, $T_2 = 2$, $T_3 = 5$ et $T_4 = 8$.

1. Soit $k \in \mathbb{N}^*$. Reconnaître la loi de X_k.

2. Le programme en langage **Python** ci-dessous définit une fonction « `ajout` » qui prend en argument une liste `L` et un entier `x`.

```
1  def ajout(L,x):
2      if (x in L) == False :
3          L.append(x)
```

Expliquez succinctement comment et à quelle condition l'exécution de la commande `ajout(L,x)` modifie la liste `L`.

3. Recopier et compléter la fonction **Python** « `Simul_T` » ci-dessous.
Cette fonction prend en argument deux entiers $N \in \mathbb{N}^*$ et $i \in [\![1, N]\!]$. Elle a pour but de simuler la variable aléatoire T_i. Dans le script nous notons :

- `L` la liste sans répétition des numéros sortis lors des tirages effectués ;
- `k` le rang du tirage en cours ;
- `x` le résultat du tirage en cours.

```
1  import numpy.random as rd
2
3  def Simul_T(N,i):
4      L = []
5      k = 0
6      while ... :
7          x = rd.randint(1,N+1)
8          ajout(L,x)
9          k = ...
10     return(...)
```

4.2.3. ECRICOME 2023

Contexte

Soit n un entier naturel non nul.
Une urne contient n boules indiscernables au toucher et numérotées de 1 à n. On tire une boule au hasard dans l'urne. Si cette boule tirée porte le numéro k, on place alors dans une seconde urne toutes les boules suivantes : une boule numérotée 1, deux boules numérotées 2, et plus généralement pour tout $j \in [\![1, k]\!]$, j boules numérotées j, jusqu'à k boules numérotées k. Les boules de cette deuxième urne sont aussi indiscernables au toucher. On effectue alors un tirage au hasard d'une boule dans cette seconde urne.
On note X la variable aléatoire égale au numéro de la première boule tirée et on note Y la variable aléatoire égale au numéro de la deuxième boule tirée.

1. Écrire une fonction en langage **Python**, nommée `seconde_urne`, prenant en entrée un entier naturel `k` non nul, et renvoyant une liste contenant 1 élément valant 1, 2 éléments valant 2, ..., j éléments valant j, ... , jusqu'à k éléments valant k.

Par exemple, l'appel de `seconde_urne(4)` renverra `[1,2,2,3,3,3,4,4,4,4]`.

2. Recopier et compléter la fonction en langage **Python** suivante pour qu'elle prenne en entrée un entier naturel n non nul, et qu'elle renvoie une réalisation du couple de variables aléatoires (X, Y).

```
1  import numpy.random as rd
2
3  def simul_XY(n):
4      X = .....
5      urne2 = seconde_urne(.....)
6      nb = len(urne2)
7      i = rd.randint(0, nb)
8      Y = .....
9      return X,Y
```

4.2.4. ECRICOME 2022

Contexte

On dispose de trois urnes U_1, U_2 et U_3, et d'une infinité de jetons numérotés 1, 2, 3, 4, ...
On répartit un par un les jetons dans les urnes : pour chaque jeton, on choisit au hasard et avec équiprobabilité une des trois urnes dans laquelle on place le jeton. Le placement de chaque jeton est indépendant de tous les autres jetons, et la capacité des urnes en nombre de jetons n'est pas limitée.
Pour tout entier naturel n non nul, on note X_n (respectivement Y_n, Z_n) le nombre de jetons présents dans l'urne 1 (respectivement l'urne 2, l'urne 3) après avoir réparti les n premiers jetons.
Soit T la variable aléatoire égale au nombre de jetons nécessaires pour que, pour la première fois, chaque urne contienne au moins un jeton.

On rappelle qu'en **Python** la commande `rd.randint(a,b)` renvoie une réalisation d'une variable aléatoire suivant une loi uniforme sur l'intervalle $[\![a, b-1]\!]$.
Compléter la fonction **Python** ci-dessous pour qu'elle simule le placement des jetons jusqu'au moment où chaque urne contient au moins un jeton, et pour qu'elle renvoie la valeur prise par la variable aléatoire T.

```
import numpy.random as rd
def simuT():
    X = 0
    Y = 0
    Z = 0
    n = 0
    liste = [X, Y, Z]
    while ....................
        i = rd.randint(0,3) # choix d'un entier entre 0 et 2
        liste[i] = ...................
        n = n+1
    return ....................
```

4.2.5. EDHEC 2022

Contexte

On désigne par n un entier naturel non nul, par p un réel de $]0,1[$ et on pose $q = 1-p$.
Dans la suite, on s'intéresse à un jeu vidéo au cours duquel le joueur doit essayer, pour gagner, de réussir, dans l'ordre, n niveaux numérotés $1, 2, \ldots, n$, ce joueur ne pouvant accéder à un niveau que s'il a réussi le niveau précédent. Le jeu s'arrête lorsque le joueur échoue à un niveau ou bien lorsqu'il a réussi les n niveaux du jeu.
Pour tout entier k de $[\![1; n-1]\!]$, on dit que le joueur a le niveau k si, et seulement si, il a réussi le niveau k et échoué au niveau $k+1$. On dit que le joueur a le niveau n si, et seulement si, il a réussi le niveau n et on dit que le joueur a le niveau 0 s'il a échoué au niveau 1.
On admet que la probabilité de passer d'un niveau à un autre est constante et égale à p, la probabilité d'accéder au niveau 1 étant, elle aussi, égale à p.
On note X_n le niveau du joueur et on admet que X_n est une variable aléatoire définie sur un espace probabilisé $(\Omega, \mathscr{A}, \mathbb{P})$ que l'on ne cherchera pas à déterminer.

Compléter le script **Python** suivant afin qu'il permette de simuler ce jeu et d'afficher la valeur prise par X_n dès que l'utilisateur saisit une valeur pour p.

```
p = float(input('Entrez la valeur de p dans ]0;1[ :'))
n = int(input('Entrez la valeur de n :'))
X = _____
while _________ and rd.random() < p:
    X = _________
print('Le niveau du joueur est :', X)
```

4.2.6. EML 2022

Contexte

Dans tout l'exercice, p désigne un réel de $]0,1[$ et on pose : $q = 1-p$.
On considère une variable aléatoire X à valeurs dans $\mathbb{N}$, dont la loi est donnée par :

$$\forall k \in \mathbb{N}, \quad \mathbb{P}\big([X=k]\big) \;=\; q^k\, p \;=\; (1-p)^k\, p$$

On admet que la variable aléatoire $Y = X+1$ suit une loi géométrique de paramètre p.

Un casino a conçu une nouvelle machine à sous dont le fonctionnement est le suivant :

- le joueur introduit un nombre k de jetons de son choix ($k \in \mathbb{N}$), puis il appuie sur un bouton pour activer la machine ;
- si k est égal à 0, alors la machine ne reverse aucun jeton au joueur ;
- si k est un entier supérieur ou égal à 1, alors la machine définit k variables aléatoires X_1, …, X_k, toutes indépendantes et de même loi que la variable aléatoire X, et reverse au joueur $(X_1+\cdots+X_k)$ jetons ;
- les fonctionnements de la machine à chaque activation sont indépendants les uns des autres et ne dépendent que du nombre de jetons introduits.

Le casino s'interroge sur la valeur à donner à p pour que la machine soit attractive pour le joueur, tout en étant rentable.
Le casino imagine alors le cas d'un joueur invétéré qui, avant chaque activation, place l'intégralité de ses jetons dans la machine, et continue de jouer encore et encore.

On note, pour tout n de $\mathbb{N}$, Z_n la variable aléatoire égale au nombre de jetons dont dispose le joueur après n activations de la machine.
On suppose que le joueur commence avec un seul jeton ; ainsi : $Z_0 = 1$.
On remarque en particulier que Z_1 suit la même loi que X.

1. Compléter la fonction **Python** suivante afin que, prenant en entrée le réel p, elle renvoie une simulation de la variable aléatoire X.

```
def simulX(p):
    Y = .....
    while .....:
        Y = Y + 1
    X = Y - 1
    return X
```

2. Compléter la fonction **Python** suivante afin que, prenant en entrée un entier n de $\mathbb{N}$ et le réel p, elle simule l'expérience aléatoire et renvoie la valeur de Z_n.
Cette fonction devra utiliser la fonction `simulX`.

```
def simulZ(n, p):
    Z = 1
    for i in range(n):
        s = 0
        for j in range(Z):
            ......
        Z = ......
    return Z
```

4.2.7. ECRICOME 2021

Contexte

On lance indéfiniment une pièce équilibrée.
On s'intéresse au rang du lancer auquel on obtient pour la première fois deux « Pile » consécutifs.
On modélise cette expérience aléatoire par un espace probabilisé $(\Omega, \mathscr{A}, \mathbb{P})$. On note alors X la variable aléatoire égale au rang du lancer où, pour la première fois, on obtient deux « Pile » consécutifs. Si on n'obtient jamais deux « Pile » consécutifs, on conviendra que X vaut -1.
Par exemple, si on obtient dans cet ordre : Pile, Face, Face, Pile, Pile, Pile, Face,... alors X prend la valeur 5.

Recopier et compléter la fonction **Python** ci-dessous afin qu'elle simule les lancers de la pièce jusqu'à l'obtention de deux « Pile » consécutifs, et qu'elle renvoie le nombre de lancers effectués.

```
1  def simulX():
2      tirs = 0
3      pile = 0
4      while pile _______:
5          if rd.random() < 1/2:
6              pile = pile + 1
7          else:
8              pile = _____
9          tirs = _____________
10     return tirs
```

4.2.8. EDHEC 2021

Contexte

On dispose de deux pièces identiques donnant pile avec la probabilité p, élément de $]0,1[$, et face avec la probabilité $q = 1 - p$.

Premier jeu. Deux joueurs A et B s'affrontent lors de lancers de ces pièces de la façon suivante, les lancers de chaque pièce étant supposés indépendants :
Pour la première manche, A et B lancent chacun leur pièce simultanément jusqu'à ce qu'ils obtiennent pile, le gagnant du jeu étant celui qui a obtenu pile le premier. En cas d'égalité et en cas d'égalité seulement, les joueurs participent à une deuxième manche dans les mêmes conditions et avec la même règle, et ainsi de suite jusqu'à la victoire de l'un d'entre eux.
Pour tout k de $\mathbb{N}^*$, on note X_k (resp. Y_k) la variable aléatoire égale au rang d'obtention du 1^{er} pile par A (resp. par B) lors de la $k^{\text{ème}}$ manche.

Deuxième jeu. En parallèle du jeu précédent, A parie sur le fait que la manche gagnée par le vainqueur le sera par un lancer d'écart et B parie le contraire.

On rappelle que la commande `rd.geometric(p)` permet à **Python** de simuler une variable aléatoire suivant la loi géométrique de paramètre p.

6. Compléter le script **Python** suivant pour qu'il simule l'expérience décrite dans la partie 1 et affiche le nom du vainqueur du premier jeu ainsi que le numéro de la manche à laquelle il a gagné.

```
p = float(input('entrez une valeur pour p :'))
c = 1
X = rd.geometric(p)
Y = rd.geometric(p)
while X == Y :
    X = ____________________
    Y = ____________________
    c = _______
if X < Y :
    ____________________________________
else :
    ____________________________________
print(c)
```

7. Compléter la commande suivante afin qu'une fois ajoutée au script précédent elle permette de simuler le deuxième jeu et d'en donner le nom du vainqueur.

```
if ____________________________ :
    print('A gagne le deuxième jeu')
else :
    ____________________________________________
```

4.2.9. EML 2021

Contexte

On considère une urne contenant initialement une boule bleue et une boule rouge. On procède à des tirages successifs d'une boule au hasard selon le protocole suivant :

- si on obtient une boule bleue, on la remet dans l'urne et on ajoute une boule bleue supplémentaire ;
- si on obtient une boule rouge, on la remet dans l'urne et on arrête l'expérience.

On suppose que toutes les boules sont indiscernables au toucher et on admet que l'expérience s'arrête avec une probabilité égale à 1. On note N la variable aléatoire égale au nombre de boules présentes dans l'urne à la fin de l'expérience.

Recopier et compléter les lignes incomplètes de la fonction **Python** suivante de façon à ce qu'elle renvoie une simulation de la variable aléatoire N.

```
def simuleN():
    b = 1 # b désigne le nombre de boules bleues dans l'urne
    while rd.random() < .........
        b = b+1
    return .........
```

4.2.10. ESSEC II 2020

Contexte

Soit m un entier fixé tel que $1 \leqslant m \leqslant n$. On note $\mathcal{P}_m$ l'ensemble des parties $A \subset \{1, \dots, n\}$ de cardinal m. On considère une variable aléatoire R, à valeurs dans $\mathcal{P}_m$ et de loi uniforme, c'est-à-dire telle que pour toute partie $A \in \mathcal{P}_m$: $\mathbb{P}(\,[R = A]\,) = \dfrac{1}{\binom{n}{m}}$.
On souhaite écrire un programme pour choisir l'ensemble R au hasard.

On considère la procédure suivante : on prend un premier élément s_1 uniformément dans $\{1, \dots, n\}$, puis un deuxième élément s_2 uniformément dans $\{1, \dots, n\} \setminus \{s_1\}$, etc. puis un $m^{\text{ème}}$ élément s_m uniformément dans $\{1, \dots, n\} \setminus \{s_1, \dots, s_{m-1}\}$. On note $S = (s_1, \dots, s_m)$, qui est un m-uplet aléatoire.
On note $R = \{s_1, \dots, s_m\}$ l'*ensemble* des entiers tirés lors de la procédure décrite ci-dessus (l'ordre dans lequel ils ont été tirés n'importe plus). On admet que pour tout ensemble $A = \{a_1, \dots, a_m\} \subset \{1, \dots, n\}$ de cardinal m, on a : $\mathbb{P}(\,[R = A]\,) = \dfrac{m!\,(n-m)!}{n!}$.
Ainsi, l'ensemble R a été choisi uniformément dans $\mathcal{P}_m$.

Pour un réel x, on note $\lfloor x \rfloor$ sa partie entière, c'est-à-dire le plus grand entier naturel inférieur ou égal à x. On admet que si U suit la loi uniforme sur $[0, 1[$, alors $X = \lfloor n\,U \rfloor$ suit la loi uniforme sur $\{0, \dots, n-1\}$.

Commentaire

Tout se passe comme si l'on souhaitait modéliser le tirage simultané de m boules dans une urne contenant n boules numérotées de 1 à n.

1. On rappelle que la fonction `rd.random()` renvoie un nombre aléatoire de loi uniforme sur $[0, 1[$, et que `np.floor(x)` renvoie la partie entière de `x`. Écrire une fonction `uniforme` en **Python** qui prend en argument un entier n, et renvoie un nombre (aléatoire), uniforme sur $\{0, \dots, n-1\}$.

```
1  def uniforme(n):
2      ...
```

2. Écrire une fonction `selection`, qui prend en argument une liste `V` et renvoie un élément `x` de `V` pris de manière aléatoire parmi tous les éléments de `V`, ainsi que la liste `V` à laquelle on a enlevé l'élément `x`. L'instruction `len(V)` renvoie le nombre d'éléments de la liste `V`.

```
1  def selection(V):
2      n = len(V)
3      ...
4      return x, V
```

3. Compléter le programme suivant, qui prend en argument deux entiers n et m avec $m \leqslant n$, et renvoie une liste `R` de m entiers distincts, pris uniformément dans $\{1, \dots, n\}$:

```
1  def choix(m, n):
2      V = [k for k in range(1, n+1)]
3      R = []
4      for i in range(m):
5          ...
6      return R
```

4.2.11. EDHEC 2019

Contexte

Soit n un entier naturel supérieur ou égal à 3.
Une urne contient une boule noire non numérotée et $n-1$ boules blanches dont $n-2$ portent le numéro 0 et une porte le numéro 1. On extrait ces boules au hasard, une à une, sans remise, jusqu'à l'apparition de la boule noire.
On note X la variable aléatoire égale au rang d'apparition de la boule noire.
On note Y la variable aléatoire qui vaut 1 si la boule numérotée 1 a été piochée lors de l'expérience précédente et qui vaut 0 sinon.

On rappelle qu'en **Python**, la commande `rd.randint(a, b+1)` simule une variable aléatoire suivant la loi uniforme $[\![a, b]\!]$.

1. Compléter le script **Python** suivant afin qu'il simule l'expérience aléatoire décrite dans cet exercice et affiche la valeur prise par la variable aléatoire X.
On admettra que la boule noire est codée tout au long de ce script par le nombre `nB + 1`, où `nB` désigne le nombre de boules blanches.

```
1  n = int(input('Entrez une valeur pour n :'))
2  nB = n - 1
3  X = 1
4  u = rd.randint(1, nB + 2)
5  while u < nB + 1:
6      nB = ________
7      u = rd.randint(1, __________)
8      X = _________
9  print('La boule noire est apparue au tirage numéro', X)
```

2. Compléter les lignes 4 et 9 ajoutées au script précédent afin que le script qui suit renvoie et affiche, en plus de celle prise par X, la valeur prise par Y.

```
1  n = int(input('Entrez une valeur pour n :'))
2  nB = n - 1
3  X = 1
4  Y = _____
5  u = rd.randint(1, nB + 2)
6  while u < nB + 1:
7      nB = __________
8      if u == 1:
9          Y = _____
10     u = rd.randint(1, __________)
11     X = _________
12 print('La boule noire est apparue au tirage numéro', X)
13 print('La valeur de Y est', Y)
```

4.2.12. EDHEC 2018

Contexte

On dispose de trois pièces : une pièce numérotée 0, pour laquelle la probabilité d'obtenir Pile vaut $\frac{1}{2}$ et celle d'obtenir Face vaut également $\frac{1}{2}$, une pièce numérotée 1, donnant Face à coup sûr et une troisième pièce numérotée 2, donnant Pile à coup sûr.
On choisit l'une de ces pièces au hasard et on la lance indéfiniment.
On considère la variable aléatoire X, égale au rang d'apparition du premier Pile et la variable aléatoire Y, égale au rang d'apparition du premier Face. On convient de donner à X la valeur 0 si l'on n'obtient jamais Pile et de donner à Y la valeur 0 si l'on n'obtient jamais Face.

On rappelle que, pour tout entier naturel non nul m, l'instruction `rd.randint(0,m)` renvoie un entier aléatoire compris entre 0 et $m-1$ (ceci de façon équiprobable).
On décide de coder Pile par 1 et Face par 0.

1. Compléter le script **Python** suivant pour qu'il permette le calcul et l'affichage de la valeur prise par la variable aléatoire X lors de l'expérience réalisée dans cet exercice.

```
1  piece = rd.randint(_____,_____)
2  x = 1
3  if piece == 0:
4      lancer = rd.randint(_____,_____)
5      while lancer == 0:
6          lancer = ____________________
7          x = _________
8  else:
9      if piece == 1:
10         x = _____
11 print(x)
```

2. Justifier que le cas où l'on joue avec la pièce numérotée 2 ne soit pas pris en compte dans le script précédent.

4.2.13. EML 2018

Contexte

Dans cette partie, p désigne un réel de $]0,1[$.

Deux individus A et B s'affrontent dans un jeu de Pile ou Face dont les règles sont les suivantes :

- le joueur A dispose d'une pièce amenant Pile avec la probabilité $\frac{2}{3}$ et lance cette pièce jusqu'à l'obtention du deuxième Pile ; on note X la v.a.r. prenant la valeur du nombre de Face alors obtenus ;
- le joueur B dispose d'une autre pièce amenant Pile avec la probabilité p et lance cette pièce jusqu'à l'obtention d'un Pile ; on note Y la v.a.r. prenant la valeur du nombre de Face alors obtenus ;
- le joueur A gagne si son nombre de Face obtenus est inférieur ou égal à celui de B ; sinon c'est le joueur B qui gagne.

Écrire une fonction **Python** d'en-tête `def simule_X():` qui simule la v.a.r. X.

4.2.14. ECRICOME 2017

Contexte

Soit n un entier naturel non nul.
On effectue une série illimitée de tirages d'une boule avec remise dans une urne contenant n boules numérotées de 1 à n. Pour tout entier naturel k non nul, on note X_k la variable aléatoire égale au numéro de la boule obtenue au $k^{\text{ème}}$ tirage.
Pour tout entier naturel k non nul, on note S_k la somme des numéros des boules obtenues lors des k premiers tirages :

$$S_k = \sum_{i=1}^{k} X_i$$

On considère enfin la variable aléatoire T_n égale au nombre de tirages nécessaires pour que, pour la première fois, la somme des numéros des boules obtenues soit supérieure ou égale à n.
Exemple : avec $n = 10$, si les numéros obtenus aux cinq premiers tirages sont dans cet ordre 2, 4, 1, 5 et 9, alors on obtient : $S_1 = 2$, $S_2 = 6$, $S_3 = 7$, $S_4 = 12$, $S_5 = 21$ et $T_{10} = 4$.

On rappelle que la commande `rd.randint(1,n+1)` simule une v.a.r. qui suit la loi uniforme sur $[\![1, n]\!]$.
Compléter la fonction suivante pour qu'elle simule la v.a.r. T_n :

```
1  def simulT(n):
2      S = _____
3      k = _____
4      while _________ :
5          tirage = rd.randint(1,n+1)
6          S = S + tirage
7          k = _______
8      return k
```

4.2.15. EML 2017

Contexte

On considère une urne contenant initialement une boule bleue et deux boules rouges.
On effectue, dans cette urne, des tirages successifs de la façon suivante : on pioche une boule au hasard et on note sa couleur, puis on la replace dans l'urne en ajoutant une boule de la même couleur que celle qui vient d'être obtenue.

Recopier et compléter la fonction suivante afin qu'elle simule l'expérience étudiée et renvoie le nombre de boules rouges obtenues lors des n premiers tirages, l'entier n étant entré en argument.

```
def EML(n):
    b = 1 # b désigne le nombre de boules bleues présentes dans l'urne
    r = 2 # r désigne le nombre de boules rouges présentes dans l'urne
    s = 0 # s désigne le nombre de boules rouges obtenues lors des n tirages
    for k in range(n):
        x = rd.random()
        if ________________:
            ___________
            ___________
        else:
            ___________
    return s
```

4.3. Simulation de variables aléatoires discrètes via la méthode d'inversion

4.3.1. ESSEC II 2022

Contexte

Considérons un joueur de fléchettes. Au $i^{\text{ème}}$ lancer de fléchette, le score est une variable aléatoire X_i qui prend ses valeurs dans $\{0, 2, 5, 10\}$. On suppose que les X_i sont indépendantes et de même loi donnée par :

$$\mathbb{P}\big([X_i = 0]\big) = \frac{1}{5}, \quad \mathbb{P}\big([X_i = 2]\big) = \frac{1}{2}, \quad \mathbb{P}\big([X_i = 5]\big) = \frac{1}{5}, \quad \mathbb{P}\big([X_i = 10]\big) = \frac{1}{10}$$

Soit $f : [0, 1] \to \mathbb{R}$ la fonction définie de la manière suivante :

$$f(x) = 0 \text{ si } x \in [0, \tfrac{1}{5}[, \quad f(x) = 2 \text{ si } x \in [\tfrac{1}{5}, \tfrac{7}{10}[, \quad f(x) = 5 \text{ si } x \in [\tfrac{7}{10}, \tfrac{9}{10}[, \quad f(x) = 10 \text{ si } x \in [\tfrac{9}{10}, 1]$$

On admet que si U est une variable aléatoire de loi uniforme sur $[0, 1]$, alors $f(U)$ a même loi que X_i.

Compléter le programme **Python** suivant, qui permet de générer un nombre aléatoire de même loi que X_i. On rappelle que la fonction `rd.random()` simule une variable aléatoire de loi uniforme sur $[0, 1]$.

```
1  def X():
2      U = rd.random()
3      ...
```

4.3.2. ECRICOME 2019

Contexte

Soit D une variable aléatoire prenant les valeurs -1 et 1 avec équiprobabilité.

Écrire une fonction en langage **Python**, d'en-tête `def D(n):`, qui prend un entier $n \geqslant 1$ en entrée, et renvoie une matrice ligne contenant n réalisations de la variable aléatoire D.

4.4. Simulation de variables aléatoires discrètes définies comme le minimum d'un ensemble aléatoire

4.4.1. ESSEC II 2025

Contexte

On définit la fonction f sur $[0,1]$ par $f(t) = (1+t)\mathrm{e}^{-t} - t$.
On admet qu'il existe un unique $\alpha \in]0,1[$ tel que $f(\alpha) = 0$ et que pour tout $t \in]0,1[$, $f(t) > 0 \iff t < \alpha$.
Soit U une variable aléatoire à densité, à valeurs dans $[0,1[$, qui suit la loi uniforme et $p \in]0,\alpha[$. On pose $\beta = f(p)$.
On définit les variables aléatoires X et Y par :

$$X = \mathbb{1}_{[\beta < U \leqslant \beta + p]} \text{ et pour tout } \omega \in \Omega,\ Y(\omega) = \min\left\{k \in \mathbb{N} \mid U(\omega) \leqslant \sum_{i=0}^{k} \frac{p^i}{i!}\mathrm{e}^{-p}\right\}$$

On dispose d'une fonction **Python**, `minimum(x,p)` qui renvoie le minimum de l'ensemble $\left\{k \in \mathbb{N} \mid x \leqslant \sum_{i=0}^{k} \frac{p^i}{i!}\mathrm{e}^{-p}\right\}$ lorsque $x \in [0,1[$ et $p \in]0,1[$.

En déduire une fonction `simulY(p)` qui réalise une simulation de Y.

4.5. Simulation de sommes de variables aléatoires discrètes

4.5.1. ECRICOME 2025

Contexte

Soit n un entier naturel non nul. Soit p un réel de $]0,1[$. Soit Y une variable aléatoire telle que la variable aléatoire $Y-1$ suit la loi binomiale $\mathcal{B}(n-1,p)$.

En utilisant uniquement la fonction `random` du module `numpy.random`, écrire une fonction en langage **Python** nommée `simulY`, prenant en arguments d'entrée les paramètres n et p, et renvoyant une simulation de la variable aléatoire Y.

4.5.2. ESSEC II 2021

Contexte

Pour tout le problème, on se donne une suite de variables aléatoires réelles $(X_n)_{n\geqslant 1}$ positives, indépendantes et de même loi.

On pose $S_0 = 0$ et, pour tout entier $n \geqslant 1$: $S_n = \sum_{i=1}^{n} X_i$.

On admet qu'avec probabilité 1, la suite $(S_n(\omega))_{n\geqslant 1}$ tend vers l'infini. On peut donc définir, pour tout réel $t \geqslant 0$, la variable aléatoire :

$$N_t \;=\; \max\{k \in \mathbb{N} \mid S_k \leqslant t\}$$

On souhaite écrire une fonction **Python** qui simule informatiquement la variable N_t. On suppose que la fonction `X` renvoie une réalisation de la variable aléatoire X. Compléter la fonction suivante, qui prend en argument un nombre réel t, et renvoie une réalisation de N_t :

```
1  def renouvellement(t):
2      N = 0
3      S = 0
4      while ...:
5          ...
6      return N - 1
```

4.6. Simulation d'un couple de variables aléatoires discrètes via des lois usuelles lors d'une expérience aléatoire en deux étapes

4.6.1. EDHEC 2020

Contexte

Soit n un entier naturel non nul et p un réel de $]0,1[$. On pose $q = 1 - p$.
On dispose de deux urnes, l'urne U qui contient n boules numérotées de 1 à n et l'urne V qui contient des boules blanches en proportion p.
On pioche une boule au hasard dans U et on note X la variable aléatoire égale au numéro de la boule tirée.
Si X prend la valeur k, on pioche k boules dans V, une par une, avec remise à chaque fois de la boule tirée, et on appelle Y la variable aléatoire égale au nombre de boules blanches obtenues.
On admet que :

- $X \hookrightarrow \mathcal{U}(\llbracket 1, n \rrbracket)$
- pour tout $k \in \llbracket 1, n \rrbracket$, la loi conditionnelle de Y sachant $[X = k]$ est la loi binomiale $\mathcal{B}(k, p)$.

On rappelle les commandes **Python** suivantes qui permettent de simuler des variables usuelles discrètes :

- `rd.randint(a, b+1)` simule une variable aléatoire suivant la loi uniforme sur $\llbracket a, b \rrbracket$,
- `rd.binomial(n, p)` simule une variable aléatoire suivant la loi binomiale de paramètres n, p,
- `rd.geometric(p)` simule une variable aléatoire suivant la loi géométrique de paramètre p,
- `rd.poisson(a)` simule une variable aléatoire suivant la loi de Poisson de paramètre a.

Compléter le script **Python** suivant afin qu'il permette de simuler les variables X et Y.

```
1 n = int(input('entrez la valeur de n :'))
2 p = float(input('entrez la valeur de p :'))
3 X = ------
4 Y = ------
```

4.7. Simulation d'un couple de variables aléatoires discrètes via sa loi de couple

4.7.1. HEC 2018

Contexte

On pose : $\forall (x,y) \in (\mathbb{R}_+^*)^2,\ B(x,y) = \displaystyle\int_0^1 t^{x-1}\,(1-t)^{y-1}\,dt$.

Pour tout réel z, soit $\left((z)^{[m]}\right)_{m\in\mathbb{N}}$ la suite définie par :

$$(z)^{[0]} = 1 \quad \text{et} \quad \forall m \in \mathbb{N},\ (z)^{[m+1]} = (z+m) \times (z)^{[m]}$$

(par exemple, pour tout $m \in \mathbb{N}$, on a $(1)^{[m]} = m!$)

On admet que, pour tout $(x,y) \in (\mathbb{R}_+^*)^2$ et pour tout couple (k,ℓ) d'entiers tels que $0 \leqslant k \leqslant \ell$:

$$B(x+k, y+\ell-k) \ = \ \frac{(x)^{[k]} \times (y)^{[\ell-k]}}{(x+y)^{[\ell]}} \times B(x,y)$$

Soit a et b des réels strictement positifs et X_1 et X_2 deux variables aléatoires à valeurs dans $\{0,1\}$ telles que :

$$\forall (x_1,x_2) \in \{0,1\}^2,\ \mathbb{P}\left([X_1 = x_1] \cap [X_2 = x_2]\right) \ = \ \frac{B(a+x_1+x_2, b+2-x_1-x_2)}{B(a,b)}$$

On admet que les deux variables X_1 et X_2 suivent la même loi de Bernoulli $\mathcal{B}\left(\dfrac{a}{a+b}\right)$.

On admet que : $\mathbb{P}_{[X_1=1]}([X_2=1]) = \dfrac{a+1}{a+b+1}$.

On admet que le coefficient de corrélation linéaire de X_1 et X_2 est : $\rho(X_1,X_2) = \dfrac{1}{a+b+1}$.

La fonction **Python** suivante dont le script est incomplet (lignes 7 et 10), effectue une simulation des deux variables X_1 et X_2 qu'elle place dans un vecteur ligne à deux composantes.

```
1   def randbetabin(a, b):
2       x = np.zeros(2)
3       u = (a+b)*rd.random()
4       v = (a+b+1)*rd.random()
5       if u < a:
6           x[0] = 1
7           if __________:
8               x[1] = 1
9       else:
10          if _________:
11              x[1] = 1
12      return x
```

1. Préciser la loi simulée par la variable `u` de la ligne 3.

2. Compléter les lignes 7 et 10.

3. Soit (p,r) un couple de réels vérifiant $0 < p < 1$ et $0 < r < 1$.
Expliquer comment utiliser la fonction `randbetabin` pour simuler deux variables aléatoires suivant une même loi de Bernoulli de paramètre p et dont le coefficient de corrélation linéaire est égal à r.

4.8. Simulation d'une chaîne de Markov

4.8.1. HEC/ESSEC I 2025

Contexte

Soit G un graphe à r sommets $1, \dots, r$, non orienté, connexe, simple (deux sommets ne peuvent être reliés que par une seule arête).
On considère l'expérience aléatoire qui consiste à se déplacer d'un sommet à l'autre de la manière suivante :

- on choisit un sommet au hasard ce qui définit la valeur de X_0 ;
- si on se trouve au sommet k après n déplacements, on a alors $X_n = k$, on se déplace vers un sommet adjacent au sommet k, ce qui définit X_{n+1}. Tous les sommets en question peuvent être choisis de manière équiprobable.

On admet que l'on définit ainsi une chaîne de Markov et on note encore $P = (p_{i,j})_{1 \leqslant i,j \leqslant r}$ sa matrice de transition.

Écrire une fonction **Python** `Trajectoire(L,n)` qui étant donnée la liste des listes d'adjacence `L` du graphe G et un entier n, simule n déplacements sur le graphe et renvoie la liste des sommets visités.
On notera que les sommets reliés au sommet i par une arête se trouvent dans la liste `L[i-1]`.

4.8.2. ESSEC II 2023

Contexte

Soit $n \in \mathbb{N}^*$, $n \geqslant 2$. On considère une famille de variables aléatoires X_t, pour $t \in \mathbb{R}^+$, sur un espace probabilisé $(\Omega, \mathcal{A}, \mathbb{P})$, vérifiant les propriétés suivantes :

(H_1) Pour tout $t \geqslant 0$, $X_t(\Omega) = \{1, \dots, n\}$.

(H_2) Pour tout $r \in \mathbb{N}^*$ et $t_1 < t_2 < \cdots < t_r$ des réels positifs, $i_1, \dots, i_{r+1}$ des éléments de $\{1, \dots, n\}$ et s un réel positif, si $\mathbb{P}([X_{t_1} = i_1] \cap \cdots \cap [X_{t_r} = i_r]) \neq 0$,

$$\mathbb{P}_{[X_{t_1}=i_1]\cap\cdots\cap[X_{t_r}=i_r]}([X_{t_r+s} = i_{r+1}]) = \mathbb{P}_{[X_{t_r}=i_r]}([X_{t_r+s} = i_{r+1}])$$

(H_3) Pour tout $i \in \{1, \dots, n\}$, la fonction $f_i : t \mapsto \mathbb{P}([X_t = i])$ est définie, dérivable sur $\mathbb{R}^+$ et n'est pas la fonction nulle. On note S_i l'ensemble des réels positifs t tels que $f_i(t) \neq 0$.

(H_4) Pour tout $(i, j) \in \{1, \dots, n\}^2$, $i \neq j$ et $h \geqslant 0$, la fonction $t \mapsto \mathbb{P}_{[X_t=i]}([X_{t+h} = j])$ est constante sur son ensemble de définition S_i et il existe un réel positif que l'on note $\alpha_{i,j}$, tel que, si $t \in S_i$ et $h \in \mathbb{R}^+$,

$$\mathbb{P}_{[X_t=i]}([X_{t+h} = j]) = \alpha_{i,j}h + \underset{h\to 0}{o}(h)$$

(H_5) Pour tout $i \in \{1, \dots, n\}$ et $h \geqslant 0$, la fonction $t \mapsto \mathbb{P}_{[X_t=i]}([X_{t+h} = i])$ est constante sur son ensemble de définition S_i et il existe un réel négatif que l'on note $\alpha_{i,i}$, tel que, si $t \in S_i$ et $h \in \mathbb{R}^+$,

$$\mathbb{P}_{[X_t=i]}([X_{t+h} = i]) = 1 + \alpha_{i,i}h + \underset{h\to 0}{o}(h)$$

On note :

- L_t la matrice ligne d'ordre n, $\big(\mathbb{P}([X_t = 1]) \dots\dots \mathbb{P}([X_t = n])\big) = \big(f_1(t) \dots\dots f_n(t)\big)$
- G la matrice carrée d'ordre n dont les coefficients sont les $\alpha_{i,j}$, appelée **matrice génératrice du processus**
- $M(s)$ la matrice (appelée **matrice de transition**) d'élément générique

$$m_{i,j}(s) = \mathbb{P}_{[X_t=i]}([X_{t+s} = j])$$

pour $(i, j) \in \{1, \dots, n\}^2$, $s \geqslant 0$ et $t \in S_i$. D'après les hypothèses (H_4) et (H_5), $m_{i,j}(s)$ est indépendant de t.

On admet que pour tout $s \geqslant 0$, $L_s = L_0 M(s)$.

On veut simuler le processus à partir de la donnée de la matrice G et de L_0. On admet que pour $t \in [0, 100]$, on peut considérer que $M(t) = \left(I_n + \frac{t}{1000}G\right)^{1000}$.

On dispose d'une fonction **Python** `transition(t,G)` de paramètres `G` représentant la matrice génératrice carrée d'ordre n et `t`, qui renvoie la matrice $\left(I_n + \frac{t}{1000}G\right)^{1000}$.

On rappelle que si `M` est une matrice, représentée par un tableau `numpy`, `M[:,j]` désigne le vecteur des coefficients de la `j`-ème colonne de `M`, de même pour `M[i,:]` et la `i`-ème ligne de `M`.

On veut simuler et représenter, sur un même graphique, les valeurs de $X_0, X_t, \ldots, X_{kt}$, pour $t > 0$ et $k \in \mathbb{N}^*$, à partir de la loi de X_0 donnée dans une ligne `L0`. Compléter la fonction suivante pour qu'elle réalise cette tâche :

```
def simulX(t,k,L0,G):
    listeDesT=[] ; listeDesX=[]
    Mt=transition(t,G) ; Lt = L0
    for i in range(k+1):
        listeDesT.append(i*t)
        p=rd.random()
        s=...
        j=0
        while p>...:
            j+=1
            s+=Lt[j]
        Lt=...
        listeDesX.append(j+1)
    plt.plot(listeDesT,listeDesX) ; plt.show()
```

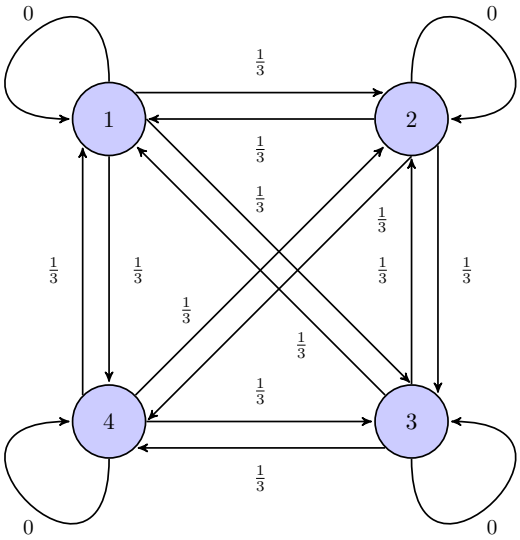
4.8.3. EDHEC 2017

Contexte

Les sommets d'un carré sont numérotés 1, 2, 3, et 4 de telle façon que les côtés du carré relient le sommet 1 au sommet 2, le sommet 2 au sommet 3, le sommet 3 au sommet 4 et le sommet 4 au sommet 1.
Un mobile se déplace aléatoirement sur les sommets de ce carré selon le protocole suivant :

- Au départ, c'est à dire à l'instant 0, le mobile est sur le sommet 1.
- Lorsque le mobile est à un instant donné sur un sommet, il se déplace à l'instant suivant sur l'un quelconque des trois autres sommets, et ceci de façon équiprobable.

Pour tout $n \in \mathbb{N}$, on note X_n la variable aléatoire égale au numéro du sommet sur lequel se situe le mobile à l'instant n. D'après le premier des deux points précédents, on a donc $X_0 = 1$.
On admet que, pour tout $n \in \mathbb{N}$, $\mathbb{P}([X_n = 1]) = \dfrac{1}{4} + \dfrac{3}{4}\left(-\dfrac{1}{3}\right)^n$
On donne ci-dessous le graphe probabiliste associé à cette chaîne de Markov.



1. Compléter la fonction **Python** suivante pour qu'elle :

- prenne en argument la position `x` à un instant n donné
- renvoie la position du mobile à l'instant suivant

On rappelle que `rd.randint(a,b)` simule un tirage uniforme dans $[\![a, b-1]\!]$.

```
1 def EtapeMarkov(x):
2 L = [1,2,3,4]
3 L.pop(x-1) # supprime le sommet x de la liste
4 i = ____________________
5 return L[i]
```

2. Compléter le script **Python** suivant pour qu'il affiche les 100 premières positions autres que celle d'origine du mobile dont le voyage est étudié dans ce problème, ainsi que le nombre n de fois où il est revenu sur le sommet numéroté 1 au cours de ses 100 premiers déplacements.

```
1  chaine = []
2  x = _____ # point de départ
3  for k in range(100):
4      x = ____________________
5      chaine.append(x)
6  print(chaine)
7  cpt = 0
8  for k in range(100):
9      if ____________________:
10         cpt = _________
11 print(cpt)
```

3. Après avoir exécuté cinq fois ce script, les réponses concernant le nombre de fois où le mobile est revenu sur le sommet 1 sont : $n = 23$, $n = 28$, $n = 23$, $n = 25$, $n = 26$.
En quoi est-ce normal ?

4.9. Simulation de variables aléatoires à densité via la méthode d'inversion

4.9.1. ECRICOME 2025

Contexte

Soit i un entier naturel non nul.
On considère une variable aléatoire X_i admettant f_i pour densité, où

$$\forall x \in \mathbb{R},\ f_i(x) = \begin{cases} \dfrac{i}{x^{i+1}} & \text{si } x \geqslant 1, \\ 0 & \text{si } x < 1. \end{cases}$$

Soit U une variable aléatoire suivant la loi uniforme sur $]0,1[$. On pose : $V_i = \dfrac{1}{U^{1/i}}$.
On admet que X_i et V_i suivent la même loi.

Écrire une fonction en langage **Python** nommée `simulX`, prenant en argument d'entrée l'entier i, et renvoyant une simulation de la variable aléatoire X_i.

4.9.2. ECRICOME 2020

Contexte

On considère une variable aléatoire X admettant f pour densité, où

$$f : t \mapsto \begin{cases} 0 & \text{si } t < a \\ \dfrac{3\,a^3}{t^4} & \text{si } t \geqslant a \end{cases}$$

Soit U une variable aléatoire suivant la loi uniforme sur $]0,1]$. On pose : $Y = \dfrac{a}{U^{\frac{1}{3}}}$.
On admet que X et Y suivent la même loi.

Écrire une fonction **Python** nommée `simulX(a, m, n)` prenant en argument un réel a strictement positif et deux entiers naturels m et n non nuls, qui renvoie une matrice à m lignes et n colonnes dont chaque coefficient est un réel choisi de façon aléatoire en suivant la loi de X. Ces réels seront choisis de façon indépendante. On rappelle que si m et n sont des entiers naturels non nuls, l'instruction `rd.random([m,n])` renvoie une matrice à m lignes et n colonnes dont chaque coefficient suit la loi uniforme sur $]0,1]$, ces coefficients étant choisis de façon indépendantes.

4.9.3. EML 2020

Contexte

Soient a et b deux réels strictement positifs. On définit la fonction f sur $\mathbb{R}$ par :

$$f : x \mapsto \begin{cases} 0 & \text{si } x > b \\ a\,\dfrac{b^a}{x^{a+1}} & \text{si } x \geqslant b \end{cases}$$

On admet que f est une densité de probabilité.
On dit qu'une variable aléatoire suit la loi de Pareto de paramètres a et b lorsqu'elle admet pour densité la fonction f.
On considère une variable aléatoire X suivant la loi de Pareto de paramètres a et b.
Soit U une variable aléatoire suivant la loi uniforme sur $[0, 1[$.
On admet que la variable aléatoire $b\,U^{-\frac{1}{a}}$ suit la loi de Pareto de paramètres a et b.

En déduire une fonction **Python** d'en-tête `def pareto(a,b):` qui prend en arguments deux réels a et b strictement positifs et qui renvoie une simulation de la variable aléatoire X.

4.9.4. HEC 2017

Contexte

Dans tout le problème, on note :
- a et b deux réels strictement positifs ;
- $f_{a,b}$ la fonction définie sur $\mathbb{R}$ par : $f_{a,b}(x) = \begin{cases} (a+bx)\ \exp\left(-ax - \dfrac{b}{2}x^2\right) & \text{si } x \geqslant 0 \\ 0 & \text{si } x < 0 \end{cases}$.

On dit qu'une variable aléatoire suit la loi exponentielle linéaire de paramètres a et b, notée $\mathcal{E}_\ell(a,b)$, si elle admet $f_{a,b}$ pour densité.
Soit Y une variable aléatoire suivant la loi exponentielle de paramètre 1.
On pose : $X = \dfrac{-a + \sqrt{a^2 + 2bY}}{b}$. On admet que X suit la loi $\mathcal{E}_\ell(a,b)$.

La fonction **Python** suivante génère des simulations de la loi exponentielle linéaire.

```
1  def simulLinExp(a,b,n):
2      U = rd.random(n)
3      Y = _________________
4      X = (-a + np.sqrt(a**2 + 2*b*Y)) / b
5      return X
```

1. Quelle est la signification de la ligne de code 2 ?

2. Compléter la ligne de code 3 pour que la fonction `simulLinExp` génère les simulations désirées.

4.10. Simulation de variables aléatoires à densité à partir de variables aléatoires suivant une loi exponentielle

4.10.1. EDHEC 2024

Contexte

Soit f la fonction qui à tout réel x associe

$$f(x) = \begin{cases} x\mathrm{e}^{-x^2/2} & \text{si } x \geqslant 0 \\ 0 & \text{sinon} \end{cases}$$

On admet que f est une densité d'une certaine variable aléatoire que l'on note X.
On pose $Z = X^2$ et on admet que Z suit la loi exponentielle de paramètre $\dfrac{1}{2}$.

Écrire une fonction **Python** d'en-tête `def simulX()` qui renvoie une simulation de X.

4.10.2. EDHEC 2023

Contexte

On considère X une variable aléatoire qui suit la loi de Pareto de paramètre c.
On pose $Z = \ln(X)$ et on admet que Z suit la loi exponentielle de paramètre c.

Écrire une fonction **Python** d'en-tête `def simulX(c)` et permettant de simuler X.

4.10.3. ECRICOME 2021

Contexte

Soit X une variable aléatoire définie sur un espace probabilisé $(\Omega, \mathscr{A}, \mathbb{P})$, suivant la loi exponentielle de paramètre 1.
Pour tout entier n supérieur ou égal à 2, on pose : $Y_n = \dfrac{-X}{1 + \mathrm{e}^{-n\,X}}$.

On rappelle qu'en langage **Python**, l'instruction `rd.exponential(1)` renvoie une réalisation d'une variable aléatoire suivant la loi exponentielle de paramètre 1.
Recopier et compléter la fonction ci-dessous qui prend en argument deux entiers n et m, et qui renvoie une matrice à une ligne et m colonnes dont chaque coefficient est une simulation de la réalisation de Y_n.

```
1  def simulY(n, m):
2      Y = np.zeros(_____)
3      for i in _____________:
4          X = rd.exponential(1)
5          Y[i] = ____________________________
6      return Y
```

4.10.4. EDHEC 2019

Contexte

Dans cet exercice, θ (theta) désigne un réel élément de $\left]0, \frac{1}{2}\right[$.

On considère la fonction f définie par : $f : x \mapsto \begin{cases} \dfrac{1}{\theta\, x^{1+\frac{1}{\theta}}} & \text{si } x \geqslant 1 \\ 0 & \text{si } x < 1 \end{cases}$

On admet que f est une densité et on considère une variable aléatoire X qui admet f pour densité.

On pose $Y = \ln(X)$ et on admet que $Y \hookrightarrow \mathcal{E}\left(\frac{1}{\theta}\right)$.

On rappelle qu'en **Python**, la commande `rd.exponential(1/a)` simule une variable aléatoire suivant la loi exponentielle de paramètre a. Écrire des commandes **Python** utilisant `rd.exponential` et permettant de simuler X.

4.10.5. EDHEC 2018

Contexte

Soit a un réel strictement positif et f la fonction définie par : $f(x) = \begin{cases} \frac{x}{a}\, \mathrm{e}^{-\frac{x^2}{2a}} & \text{si } x \geqslant 0 \\ 0 & \text{si } x < 0 \end{cases}$.

On admet que f est une densité. On considère une variable aléatoire X de densité f.

On considère la variable aléatoire Y définie par : $Y = \dfrac{X^2}{2a}$. On admet que Y suit la loi exponentielle de paramètre 1.

On rappelle qu'en **Python** la commande `rd.exponential(c)` simule une variable aléatoire suivant la loi exponentielle de paramètre $\frac{1}{c}$. Écrire un script **Python** demandant la valeur de a à l'utilisateur et permettant de simuler la variable aléatoire X.

4.10.6. EDHEC 2017

Contexte

On désigne par n un entier naturel non nul et par $X_1, \ldots, X_n$ des variables aléatoires définies sur le même espace probabilisé, indépendantes et suivant la même loi que V, c'est à dire la loi $\mathcal{E}(1)$.

On considère la variable aléatoire Y_n définie par $Y_n = \max(X_1, X_2, \ldots, X_n)$, c'est à dire que pour tout ω de Ω, on a : $Y_n(\omega) = \max(X_1(\omega), X_2(\omega), \ldots, X_n(\omega))$. On admet que Y_n est une variable aléatoire à densité.

On pose $Z_n = Y_n - \ln(n)$.

On rappelle que `rd.exponential(1,n)` simule n variables aléatoires indépendantes et suivant toutes la loi exponentielle de paramètre 1. Compléter la déclaration de fonction **Python** suivante afin qu'elle simule la variable aléatoire Z_n.

```
1 def simulZ(n):
2     x = rd.exponential(1,n)
3     return ____________________
```

4.10.7. ESSEC I 2017

Contexte (Partie 1)

Soit $\alpha \in \mathbb{R}$ et $\beta > 0$. On dit qu'une variable aléatoire réelle à densité suit une loi de Laplace de paramètre (α, β), notée $\mathcal{L}(\alpha, \beta)$, si elle admet comme densité la fonction f donnée par :

$$\forall t \in \mathbb{R},\ f(t) = \frac{1}{2\beta}\ \exp\left(-\frac{|t-\alpha|}{\beta}\right)$$

On admet que si X suit la loi $\mathcal{L}(0,1)$, alors $\beta X + \alpha$ suit la loi $\mathcal{L}(\alpha, \beta)$.
Soit U une variable aléatoire qui suit la loi exponentielle de paramètre 1 et V une variable aléatoire qui suit la loi de Bernoulli de paramètre $\frac{1}{2}$ et indépendante de U.
On admet que $X = (2V-1)U$ suit la loi $\mathcal{L}(0,1)$.

1. Compléter la fonction **Python** ci-dessous pour que la fonction ainsi définie réalise la simulation d'une variable aléatoire qui suit la loi $\mathcal{L}(\alpha, \beta)$:

```
1  def Laplace(alpha, beta):
2      if _________________ < 1/2:
3          V = 1
4      else:
5          V = 0
6      X = (2*V-1) * rd.exponential(1)
7      return ____________________
```

Contexte (Partie 3)

- Soit $d \in \mathbb{N}^*$. On considère $D = [\![0, d]\!]$ et n un entier naturel plus grand que 2.
- On considère q une application de D^n dans $\mathbb{R}$.

Concrètement, un élément de D^n représente une table d'une base de données et q une requête sur cette base. Étant donné $a = (a_1, \ldots, a_n)$, on s'intéresse au problème de la confidentialité de certains des a_i lorsque les autres a_i sont connus, ainsi que D, q et $q(a)$.

Soit $\varepsilon > 0$.
Pour tout $a = (a_1, \ldots, a_n)$ appartenant à D^n, $q(a) = \sum_{k=1}^{n} a_k$.

Pour tout $a \in D^n$, on pose $X_a = q(a) + Y$ où Y suit la loi de Laplace de paramètre $(0, \beta)$ avec $\beta = \dfrac{d}{\varepsilon}$.
On pose :

$$Z_a = \begin{cases} 0 & \text{si } X_a < \frac{1}{2} \\ \left\lfloor X_a + \frac{1}{2} \right\rfloor & \text{si } X_a \in [\frac{1}{2}, nd - \frac{1}{2}[\\ nd & \text{si } X_a \geqslant nd - \frac{1}{2} \end{cases}$$

2. Écrire un script qui pour d, n et ε saisis par l'utilisateur, génère une valeur aléatoire de $a \in D^n$ puis affiche $q(a)$ et Z_a.

3. Pour $n = 1000$, $d = 4$ et ε choisi par l'utilisateur, écrire un script qui estime la valeur moyenne de $\dfrac{|Z_a - q(a)|}{q(a)}$ (on considèrera que $q(a)$ est toujours non nul).

4.11. Simulation de sommes de variables aléatoires à densité

4.11.1. ECRICOME 2020

Contexte

On considère une variable aléatoire X admettant f pour densité, où

$$f : t \mapsto \begin{cases} 0 & \text{si } t < a \\ \dfrac{3\,a^3}{t^4} & \text{si } t \geqslant a \end{cases}$$

On dispose d'une fonction **Python** nommée `simulX(a, m, n)` prenant en argument un réel a strictement positif et deux entiers naturels m et n non nuls, qui renvoie une matrice à m lignes et n colonnes dont chaque coefficient est un réel choisi de façon aléatoire et indépendant des autres en suivant la loi de X.
Soit n un entier naturel non nul, et X_1, ..., X_n n variables aléatoires indépendantes et suivant toutes la même loi que X. On pose : $V_n = \dfrac{2}{3n} \sum_{k=1}^{n} X_k$.

On rappelle que si `A` est un tableau (ou un vecteur ligne) **Python**, l'instruction `sum(A)` renvoie la somme des coefficients du tableau `A`.
Compléter la fonction ci-dessous afin qu'elle réalise m simulations de la variable aléatoire V_n et renvoie les résultats obtenus sous forme d'un tableau à m éléments :

```
1  def simulV(a,m,n):
2      X = simulX(a,m,n)
3      V = np.zeros(m)
4      for k in _____________:
5          V[k] = ____________________________
6      return V
```

4.11.2. EDHEC 2020

Contexte

On considère une variable aléatoire X suivant la loi normale $\mathcal{N}\left(0, \sigma^2\right)$, où σ est strictement positif.
On pose $Y = |X|$ et on admet que Y est une variable aléatoire.
Soit n un entier naturel supérieur ou égal à 1. On considère un échantillon $(Y_1, Y_2, \ldots, Y_n)$ composé de variables aléatoires, mutuellement indépendantes, et ayant toutes la même loi que Y.
On note S_n la variable aléatoire définie par : $S_n = \dfrac{1}{n} \sum_{k=1}^{n} Y_i$. On note enfin $T_n = \sqrt{\dfrac{\pi}{2}}\, S_n$.

On rappelle qu'en **Python**, si i désigne un entier naturel non nul, la commande `rd.normal(m,s,i)` simule, dans un tableau à `i` colonnes, `i` variables aléatoires mutuellement indépendantes et suivant toutes la loi normale d'espérance `m` et de variance `s`^{2}.
Compléter le script **Python** suivant afin qu'il permette de simuler les variables aléatoires S_n et T_n pour des valeurs de n et σ entrées par l'utilisateur.

```
n = int(input('entrez la valeur de n :'))
sigma = float(input('entrez la valeur de sigma :'))
X = ------ # simulations de X1, ..., Xn
Y = ------ # simulations de Y1, ..., Yn
S = ------
T = ------
```

4.11.3. HEC/ESSEC I 2020

Contexte

Dans toute la suite du sujet, on désigne par p un réel de l'intervalle $]0,1[$ et on pose $q = 1-p$.
On modélise une compétition entre deux groupes d'individus A et B avec les règles suivantes.

- Le groupe A doit résoudre une suite de problèmes $(P_k)_{k\geqslant 1}$ dans l'ordre des indices. Au temps $t = 0$, le groupe commence la résolution du problème P_1, ce qui lui prend un temps représenté par la variable aléatoire X_1. Une fois P_1 résolu, le groupe aborde immédiatement le problème P_2, et on note X_2 le temps consacré à la résolution de P_2 par le groupe A, et ainsi de suite.
  Pour tout $k \in \mathbb{N}^*$, on note X_k la variable aléatoire donnant le temps consacré à la résolution du problème P_k par le groupe A.
- De même, le groupe B doit résoudre dans l'ordre une suite de problèmes $(Q_k)_{k\geqslant 1}$; la résolution du premier problème Q_1 commence au temps $t = 0$ et on note, pour tout $k \in \mathbb{N}^*$, Y_k la variable aléatoire donnant le temps consacré par le groupe B à la résolution du problème Q_k.
- À ce jeu est associé un espace probabilisé $(\Omega, \mathscr{A}, \mathbb{P})$ sur lequel sont définies les suites de variables aléatoires $(X_k)_{k\geqslant 1}$ et $(Y_k)_{k\geqslant 1}$, et on fait les hypothèses suivantes :
  - pour tout $k \in \mathbb{N}^*$, X_k suit la loi exponentielle de paramètre p, notée $\mathcal{E}(p)$, et Y_k suit la loi exponentielle $\mathcal{E}(q)$;
  - pour tout $k \in \mathbb{N}^*$, les variables aléatoires $X_1, \ldots, X_k, Y_1, \ldots, Y_k$ sont indépendantes.
- On établit alors la liste de tous les problèmes résolus *dans l'ordre où ils le sont par les deux groupes.* En cas de simultanéité temporelle de la résolution par les deux groupes d'un de leurs problèmes, on placera d'abord le problème résolu par A dans la liste puis celui résolu par B.
  Pour tout $n \in \mathbb{N}^*$, on note U_n la variable aléatoire de Bernoulli associée à l'événement « le $n^{\text{ème}}$ problème placé dans la liste est un problème résolu par le groupe A ».
  Par exemple, si la liste des cinq premiers problèmes résolus est $(P_1, P_2, Q_1, P_3, Q_2)$, alors $U_1 = 1$, $U_2 = 1$, $U_3 = 0$, $U_4 = 1$ et $U_5 = 0$.
- Pour tout $n \geqslant 0$, on note aussi S_n la variable aléatoire donnant le nombre de problèmes qui ont été résolus par A présents dans la liste des n premiers problèmes résolus. En particulier, S_0 vaut toujours 0.

1. Compléter le script **Python** suivant pour qu'il simule le jeu et, pour n, p donnés, affiche la liste des valeurs $U_1, U_2, \ldots, U_n$:

```
p = float(input('p = '))
n = int(input('n = '))
q = 1 - p
U = np.zeros(n)
sommeX = rd.exponential(1/p)
sommeY = rd.exponential(1/q)
mini = min(sommeX, sommeY)
for k in range(n):
    if sommeX == ...:
        U[k] = ...
        sommeX = sommeX + rd.exponential(1/p)
    else:
        sommeY = ...
    mini = min(sommeX, sommeY)
...
```

2. Quelle(s) instruction(s) faut-il ajouter pour afficher la valeur de S_n ?

4.12. Simulation d'une variable aléatoire à densité lors d'une expérience en deux étapes

4.12.1. ECRICOME 2025

Contexte

Soit n un entier naturel non nul. Soit p un réel de $]0,1[$.
La population active d'un territoire est divisée en n catégories socioprofessionnelles, numérotées de 1 à n.
Pour tout entier i compris entre 1 et n, on note X_i la variable aléatoire égale au revenu mensuel, en milliers d'euros, d'un individu choisi au hasard avec équiprobabilité au sein de la catégorie socioprofessionnelle numéro i. On suppose que la variable aléatoire X_i admet pour densité la fonction f_i définie sur $\mathbb{R}$ par :

$$\forall x \in \mathbb{R},\ f_i(x) = \begin{cases} \dfrac{i}{x^{i+1}} & \text{si } x \geqslant 1, \\ 0 & \text{si } x < 1. \end{cases}$$

On dispose d'une fonction `simulX(i)` simulant la variable aléatoire X_i.
Un institut réalise un sondage selon le protocole suivant :

- On choisit une catégorie socioprofessionnelle de manière aléatoire (mais sans équiprobabilité), et on note Y la variable aléatoire égale au numéro de la catégorie choisie.
  On suppose que $Y - 1$ suit la loi binomiale $\mathcal{B}(n-1, p)$ et on dispose d'une fonction `simulY(n,p)` qui simule la variable aléatoire Y.
- On sélectionne alors un individu au hasard (avec équiprobabilité) dans la catégorie socioprofessionnelle choisie à l'étape précédente, et on note Z_n la variable aléatoire égale à son revenu mensuel, en milliers d'euros.

Écrire une fonction en langage **Python** nommée `sondage`, prenant en arguments d'entrée les paramètres n et p, et renvoyant une simulation de la variable aléatoire Z_n.

4.13. Simulation d'un couple de variables aléatoires à densité

4.13.1. ESSEC II 2025

Contexte

On considère $(X_k)_{k\in\mathbb{N}^*}$ une suite de variables aléatoires indépendantes à densité.
Soit $s \in \mathbb{R}$, pour tout $n \in \mathbb{N}^*$ on définit des variables aléatoires, $a_{n,s}$, $Y_{n,s}$, Z_n et $K_{n,s}$ par, pour tout $\omega \in \Omega$:

$$a_{n,s}(\omega) = \min(\{k \in [\![1,n]\!] \mid X_k(\omega) > s\} \cup \{n\}) \ , \ Y_{n,s}(\omega) = X_{a_{n,s}(\omega)}(\omega) \ , \ Z_n(\omega) = \max_{k\in[\![1,n]\!]}(X_k(\omega))$$

et $K_{n,s}(\omega)$ est égal au nombre d'indices $k \in [\![1,n]\!]$ tels que $X_k(\omega) > s$.
On cherche à choisir s pour maximiser $r_n = \mathbb{P}(Y_{n,s} = Z_n)$.
On pose pour tout $k \in [\![1,n]\!]$, $p_k = \mathbb{P}(X_k > s)$ et on suppose que $p_k \neq 1$.
On suppose que les X_k suivent la même loi uniforme sur $[0,1[$ et donc que les p_k sont tous égaux, non nuls. On note p cette valeur commune.
On admet que $s = 1 - p$.

Écrire une fonction **Python** `simulCouple(n,p)` qui renvoie une simulation du couple $(Z_n, Y_{n,s})$.

4.14. Simulation de variables aléatoires suivant une « loi composée »

4.14.1. EML 2021

Contexte

On considère N une variable aléatoire à valeurs dans $\mathbb{N} \setminus \{0, 1\}$.
On dispose d'une fonction **Python** `simuleN()` qui simule la variable aléatoire N.
On considère une suite $(X_n)_{n\in\mathbb{N}^*}$ de variables aléatoires indépendantes et de même loi uniforme sur $[0, 1]$. On suppose que, pour tout n de $\mathbb{N}^*$, les variables aléatoires $X_1, \ldots, X_n$ et N sont mutuellement indépendantes.
On définit la variable aléatoire $T = \max(X_1, \ldots, X_N)$, ce qui signifie :

$$\forall \omega \in \Omega,\ T(\omega) \ = \ \max\left(X_1(\omega), \ldots, X_{N(\omega)}(\omega)\right)$$

Ainsi par exemple, si N prend la valeur 3, alors $T = \max(X_1, X_2, X_3)$; si N prend la valeur 5, alors $T = \max(X_1, X_2, X_3, X_4, X_5)$; etc.

On rappelle que l'instruction `rd.random(d)` renvoie un tableau de taille `d` où les coefficients sont des réalisations de variables aléatoires indépendantes suivant la loi uniforme sur $[0, 1]$.

Écrire une fonction **Python** nommée `simuleT()` qui renvoie une simulation de la variable aléatoire T.

4.14.2. HEC/ESSEC I 2021

Contexte

On considère :
- un espace probabilisé $(\Omega, \mathscr{A}, \mathbb{P})$ et J un sous-ensemble non vide de $\mathbb{R}^+$;
- une variable aléatoire Y sur cet espace à valeurs dans J.
- une famille $(X_t)_{t\in J}$ de variables aléatoires sur cet espace **à valeurs dans** $\mathbb{N}$ et **indépendantes de** Y telles que pour tout $t \in J$:

$$X_t \text{ suit la loi } \mu(t)$$

$\mu(t)$ désignant une loi de probabilité de paramètre t.
On définit la variable aléatoire Z sur cet espace par :

$$\forall \omega \in \Omega, \text{ si } Y(\omega) = t \text{ alors } Z(\omega) = X_t(\omega)$$

et on dit que Z suit la loi $\mu(Y)$.
On considère dans cette partie une telle variable Z qui suit la loi $\mu(Y)$.

On considère le script **Python** suivant :

```
1  def X(t) :
2      r = 1
3      while rd.random() > ... :
4          r = ...
5      return r
6
7  Y = rd.random()
8  Z = ...
9  print(Z)
```

En considérant les notations précédentes avec $J =]0, 1[$ et en notant Y la variable aléatoire dont `Y` est une simulation, compléter le script précédent pour que Z soit une simulation d'une variable aléatoire qui suit la loi géometrique $\mathcal{G}(Y)$.

4.15. Simulation du maximum/minimum de plusieurs variables aléatoires à densité

4.15.1. EDHEC 2025

Contexte

Soit $n \in \mathbb{N}^*$. On considère une variable aléatoire X_n admettant f_n comme densité, où f_n est la fonction définie par :

$$\forall x \in \mathbb{R},\ f_n(x) = \begin{cases} \left(1 - \dfrac{x}{n}\right)^{n-1} & \text{si } 0 \leqslant x \leqslant n \\ 0 & \text{sinon} \end{cases}$$

Soit $U_1, \ldots, U_n$ des variables aléatoires mutuellement indépendantes, et suivant toutes la loi uniforme sur $[0, 1]$. On considère la variable aléatoire M_n définie par $M_n = \min(U_1, \ldots, U_n)$, ce qui signifie que, pour tout $\omega \in \Omega$, $M_n(\omega)$ est le plus petit des réels $U_1(\omega), \ldots, U_n(\omega)$.
On pose $Z_n = nM_n$.
On admet que Z_n suit la même loi que X_n.

Écrire une fonction **Python** renvoyant une réalisation de X_n.

4.15.2. EDHEC 2024

Contexte

Soit f la fonction qui à tout réel x associe

$$f(x) = \begin{cases} x\mathrm{e}^{-x^2/2} & \text{si } x \geqslant 0 \\ 0 & \text{sinon} \end{cases}$$

On admet que f est une densité d'une certaine variable aléatoire que l'on note X.
On dispose d'une fonction **Python** d'en-tête `def simulX()` qui renvoie une simulation de X.
On considère une suite $(X_n)_{n\in\mathbb{N}^*}$ de variables aléatoires mutuellement indépendantes, et suivant toutes la même loi que X. Pour tout entier naturel n non nul, on pose $M_n = \min(X_1, \ldots, X_n)$.

Compléter la fonction **Python** suivante afin qu'elle renvoie une simulation de M_n à l'appel de `simulM(n)`.

```
1  def simulM(n):
2      X=np.array([----- for k in range(n)])
3      M=-----
4      return M
```

4.15.3. ESSEC I 2018

Contexte

Dans tous le sujet :

- on désigne par n un entier naturel, au moins égal à 2,
- X est une v.a.r. à valeurs dans un intervalle $]0,\alpha[$, où α est un réel strictement positif. On suppose que X admet une densité f strictement positive et continue sur $]0,\alpha[$, et nulle en dehors de $]0,\alpha[$.
- on note F la fonction de répartition de X.
- X_1, …, X_n est une famille de v.a.r. mutuellement indépendantes et de même loi que X.

On définit deux variables aléatoires Y_n et Z_n de la façon suivante.
Pour tout $\omega \in \Omega$:

- $Y_n(\omega) = \max(X_1(\omega), \ldots, X_n(\omega))$ est le plus grand des réels $X_1(\omega)$, …, $X_n(\omega)$;
  on remarque que Y_n est définie également lorsque n vaut 1, de sorte que dans la suite du sujet on pourra considérer Y_{n-1}.
- $Z_n(\omega)$ est le « deuxième plus grand » des nombres $X_1(\omega)$, …, $X_n(\omega)$, autrement dit, une fois que ces n réels sont ordonnés dans l'ordre croissant, Z_n est l'avant-dernière valeur. On note que lorsque la plus grande valeur est présente plusieurs fois, $Z_n(\omega)$ et $Y_n(\omega)$ sont égaux.

On suppose que l'on a défini une fonction **Python** d'entête `def simulX(n):` qui retourne une simulation d'un échantillon de taille n de la loi de X sous la forme d'un vecteur de longueur n. Compléter la fonction qui suit pour qu'elle retourne le couple $(Y_n(\omega), Z_n(\omega))$ associé à l'échantillon simulé par l'instruction `X = simulX(n)` :

```
1  def deuxPlusGrands(n):
2      X = simulX(n)
3      if _________________:
4          y = X[0]; z = X[1]
5      else:
6          ______________________
7      for k in range(2,n):
8          if X[k] > y:
9              z = _____; y = ________
10         else:
11             if _____________:
12                 z = ________
13     return [y,z]
```

4.16. Tracé d'un diagramme en bâtons

4.16.1. ECRICOME 2025

Contexte

Soit n un entier naturel non nul. Soit p un réel de $]0,1[$. Soit Y une variable aléatoire telle que la variable aléatoire $Y-1$ suit la loi binomiale $\mathcal{B}(n-1,p)$.
On dispose d'une fonction `loiY(n,p)` renvoyant une liste $(p_1,\ldots,p_n)$ où pour i compris entre 1 et n, p_i est une valeur approchée de $\mathbb{P}(Y=i)$.
Extrait de l'annexe fournie en fin de sujet :

- La fonction `plt.bar` prend en arguments d'entrée deux listes `x` et `y` de même longueur ou deux tableaux Numpy `x` et `y` à une ligne et de même longueur, et produit un diagramme en bâtons, les coefficients de `x` indiquant les abscisses auxquels sont centrés les bâtons, et les coefficients de `y` déterminant la hauteur de chaque bâton en ordonnée.
- La fonction `plt.show`, employée sans argument d'entrée, permet l'affichage d'une figure préalablement tracée, par exemple avec les fonctions `plt.plot` ou `plt.bar`.

Écrire une fonction **Python**, prenant en arguments d'entrée les paramètres n et p, permettant d'afficher un diagramme en bâtons représentant approximativement la loi de Y. On représentera les valeurs de Y en abscisses et les probabilités correspondantes en ordonnées.

4.17. Utilisation de la loi faible des grands nombres

4.17.1. ESSEC II 2025

Contexte

On considère $(X_k)_{k\in\mathbb{N}^*}$ une suite de variables aléatoires indépendantes à densité.
Soit $s \in \mathbb{R}$, pour tout $n \in \mathbb{N}^*$ on définit des variables aléatoires, $a_{n,s}$, $Y_{n,s}$, Z_n et $K_{n,s}$ par, pour tout $\omega \in \Omega$:

$$a_{n,s}(\omega) = \min(\{k \in [\![1,n]\!] \mid X_k(\omega) > s\} \cup \{n\}) \ , \ Y_{n,s}(\omega) = X_{a_{n,s}(\omega)}(\omega) \ , \ Z_n(\omega) = \max_{k\in[\![1,n]\!]}(X_k(\omega))$$

et $K_{n,s}(\omega)$ est égal au nombre d'indices $k \in [\![1,n]\!]$ tels que $X_k(\omega) > s$.
On cherche à choisir s pour maximiser $r_n = \mathbb{P}(Y_{n,s} = Z_n)$.
On pose pour tout $k \in [\![1,n]\!]$, $p_k = \mathbb{P}(X_k > s)$ et on suppose que $p_k \neq 1$.
On suppose que les X_k suivent la même loi uniforme sur $[0,1[$ et donc que les p_k sont tous égaux, non nuls. On note p cette valeur commune
On dispose d'une fonction `simulCouple(n,p)` qui renvoie une simulation du couple $(Z_n, Y_{n,s})$.

Écrire un programme **Python** qui réalise et affiche une estimation de r_{10} dans ces conditions lorsque $n = 10$ et $p = 0.15$.

4.17.2. ECRICOME 2025

Contexte

Soit n un entier naturel non nul. Soit p un réel de $]0,1[$. Soit Y une variable aléatoire telle que la variable aléatoire $Y - 1$ suit la loi binomiale $\mathcal{B}(n-1,p)$.
On dispose d'une fonction `simulY(n,p)` qui simule la variable aléatoire Y.

Recopier et compléter la fonction, en langage **Python**, nommée `loiY`, prenant en arguments d'entrée les paramètres n et p, et renvoyant une liste $(p_1, \ldots, p_n)$ où pour i compris entre 1 et n, p_i est une valeur approchée de $\mathbb{P}(Y = i)$.

```
1  def loiY(n,p):
2      N = 10000
3      loi = [0] * n
4      for k in ------ :
5          y = simulY(n,p)
6          loi[--] --
7      loi
```

4.17.3. ECRICOME 2024

Contexte

On considère une variable aléatoire X de loi normale d'espérance $-a$ et de variance $\dfrac{1}{2}$.
Soit Z une variable aléatoire de loi normale centrée réduite.
On admet que, si $\alpha = \frac{1}{\sqrt{2}}$ et $\beta = -a$, alors $\alpha Z + \beta$ suit la même loi que X.

Recopier et compléter la fonction **Python** suivante, prenant en arguments d'entrée les réels a et x, pour qu'elle renvoie une estimation de la probabilité $\mathbb{P}([X \geqslant x])$.

```
import numpy as np
import numpy.random as rd
def estim_proba(a, x):
    num = 0
    for i in range(10000):
        Z = rd.normal()
        X = _____ + Z/_____
        if _____:
            num = num + 1
    return _____
```

4.17.4. EML 2024

Contexte

Soit N un entier naturel supérieur ou égal à 1. On dispose d'une urne contenant N boules numérotées de 1 à N, et on effectue une succession illimitée de tirages d'une boule avec remise dans l'urne. Pour tout $k \in \mathbb{N}^*$, on note X_k la variable aléatoire indiquant le numéro de la boule obtenue au $k-$ième tirage.
Pour tout entier $i \in [\![1, N]\!]$, on note T_i la variable aléatoire égale au nombre de tirages nécessaires pour obtenir i numéros distincts, ainsi $T_i = k$ si on a obtenu i numéros distincts lors des k premiers tirages, mais seulement $i-1$ numéros distincts lors des $k-1$ premiers tirages.
Exemple : on suppose $N = 4$, si les huit premiers tirages donnent

i	1	2	3	4	5	6	7	8
X_i	2	3	3	3	1	2	1	4

alors $T_1 = 1$, $T_2 = 2$, $T_3 = 5$ et $T_4 = 8$.
On dispose d'une fonction `ajout(L,x)` dont le code est donné ci-dessous :

```
def ajout(L,x):
    if (x in L) == False :
        L.append(x)
```

Cette fonction prend en entrée une liste `L` et un nombre réel `x`. Si le nombre `x` n'est pas déjà contenu dans la liste `L`, alors il est ajouté à la fin de la liste `L`.
On dispose également d'une fonction `Simul_T(N,i)` qui simule la variable aléatoire T_i.

On suppose $N = 3$.
Rédiger un programme **Python** qui calcule et affiche la moyenne de 100 réalisations de `Simul_T(3,2)`.
Que représente le résultat obtenu par rapport à la variable aléatoire T_2 ?

4.17.5. HEC/ESSEC I 2024

Contexte

Soit n un entier supérieur ou égal à 3 et p un réel appartenant à $]0,1[$.
Pour générer des graphes non orientés de manière aléatoire, on se donne :

- $S = [\![0, n-1]\!]$, les sommets du graphe ;
- pour toute paire de sommets $\{u, v\}$ avec $u < v$, $T_{u,v}$ une variable aléatoire de Bernoulli de paramètre p.
  Les variables aléatoires $T_{u,v}$ pour $\{u, v\}$ décrivant les paires de sommets avec $u < v$, sont supposées indépendantes ;
- les arêtes d'un graphe G ainsi généré sont les paires $\{u, v\}$ telles que $T_{u,v} = 1$ si $u < v$ ou $T_{v,u} = 1$ si $v < u$.

On note $\mathcal{T}$ l'ensemble des parties $\{u, v, w\}$ à trois éléments de l'ensemble des sommets, r le nombre de ses éléments et on pose

$$\mathcal{T} = \{t_1, \ldots, t_r\}$$

Étant donné $t = \{u, v, w\}$, un élément de $\mathcal{T}$, on dit que t est un triangle dans un graphe G généré aléatoirement si $\{u, v\}$, $\{v, w\}$ et $\{w, u\}$ sont des arêtes de G.
On note Z_n la variable aléatoire égale au nombre de triangles de G.
On dispose d'une fonction **Python** `nbTriangles(L)` qui retourne le nombre de triangles du graphe G dont la liste des listes d'adjacence est représentée par L.

On suppose que la fonction `graphe(n,p)` génère un graphe aléatoire suivant les hypothèses décrites dans le préambule.
Expliquer ce que retourne la fonction suivante :

```
1  def fonctionMystere(n):
2      cpt = 0
3      for i in range(1000):
4          L = graphe(n, 1/n)
5          if nbTriangles(L) == 0:
6              cpt += 1
7      return cpt / 1000
```

4.17.6. ECRICOME 2023

Contexte

Soit n un entier naturel non nul.
On considère (X, Y) un couple de variables aléatoires discrètes tel que $X(\Omega) = Y(\Omega) = [\![1, n]\!]$.
On dispose d'une fonction **Python** nommée `simul_XY(n)` qui renvoie une liste dont le premier élément est une réalisation de X et le second est une réalisation de Y.

On considère la fonction en langage **Python** suivante, prenant en entrée un entier naturel `n` non nul.

```
1  def fonction(n):
2      liste = [0]*n
3      for i in range(10000):
4          j = simul_XY(n)[1]
5          liste[j-1] = liste[j-1] + 1/10000
6      return liste
```

Quelles valeurs les éléments de la liste renvoyée permettent-ils d'estimer ?

4.17.7. EDHEC 2023

Contexte

On considère deux variables aléatoires, X et Y, indépendantes et suivant la même loi géométrique de paramètre $\frac{1}{2}$.

Soit $A(X,Y)$ la matrice aléatoire définie par $A(X,Y) = \begin{pmatrix} X & 1 \\ 0 & Y \end{pmatrix}$.

On admet que la probabilité p pour que $A(X,Y)$ ne soit pas diagonalisable vaut $p = \frac{1}{3}$.

On admet également que $A(X,Y)$ n'est pas diagonalisable si et seulement si l'événement $[X = Y]$ est réalisé.

On considère le script **Python** suivant :

```
m=int(input('entrez une valeur entière pour m :'))
c=0
for k in range(m):
    X=rd.geometric(1/2)
    Y=rd.geometric(1/2)
    if X==Y:
        c=c+1
i = 1-c/m
print(i)
```

Pour de grandes valeurs de l'entier naturel m, de quel réel le contenu de la variable i est-il proche ?

4.17.8. ECRICOME 2022

Contexte

On dispose d'une fonction **Python** `simuT()` qui simule une variable aléatoire discrète T.

Écrire un script **Python** qui simule 10 000 fois la variable aléatoire T et qui renvoie une valeur approchée de son espérance (en supposant que cette espérance existe).

4.17.9. ECRICOME 2021

Contexte

`(Exo 2)` Soit X une variable aléatoire définie sur un espace probabilisé $(\Omega, \mathscr{A}, \mathbb{P})$, suivant la loi exponentielle de paramètre 1.

Pour tout entier n supérieur ou égal à 2, on pose : $Y_n = \dfrac{-X}{1 + \mathrm{e}^{-n\,X}}$.

On dispose d'une fonction **Python** `simulY(n, m)` qui renvoie une matrice à une ligne et m colonnes dont chaque coefficient est une simulation de la réalisation de Y_n.

On tape dans **Python** le script suivant :

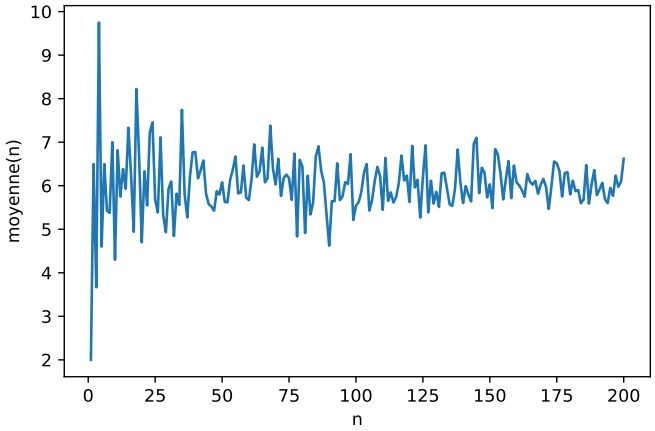
```
n = int(input('Entrer la valeur de n :'))
print(np.mean(simulY(n, 1000)))
```

Expliquer ce que fait ce script dans le contexte de l'exercice.

Contexte

`(Exo 3)` On lance indéfiniment une pièce équilibrée.
On s'intéresse au rang du lancer auquel on obtient pour la première fois deux « Pile » consécutifs.
On modélise cette expérience aléatoire par un espace probabilisé $(\Omega, \mathscr{A}, \mathbb{P})$. On note alors X la variable aléatoire égale au rang du lancer où, pour la première fois, on obtient deux « Pile » consécutifs. Si on n'obtient jamais deux « Pile » consécutifs, on conviendra que X vaut -1.
On dispose d'une fonction **Python** `simulX()` qui simule les lancers de la pièce jusqu'à l'obtention de deux « Pile » consécutifs, et qui renvoie le nombre de lancers effectués.

1. Écrire une fonction **Python** nommée `moyenne(n)` qui simule `n` fois l'expérience ci-dessus et renvoie la moyenne des résultats obtenus.
2. On calcule `moyenne(n)` pour chaque entier `n` de $[\![1, 200]\!]$, et on trace les résultats obtenus dans le graphe suivant.



Que pouvez-vous conjecturer sur la variable aléatoire X ?

4.17.10. EML 2021

Contexte

On considère N une variable aléatoire à valeurs dans $\mathbb{N} \setminus \{0, 1\}$.
On considère une suite $(X_n)_{n\in\mathbb{N}^*}$ de variables aléatoires indépendantes et de même loi uniforme sur $[0, 1]$. On suppose que, pour tout n de $\mathbb{N}^*$, les variables aléatoires $X_1, \ldots, X_n$ et N sont mutuellement indépendantes.
On définit la variable aléatoire $T = \max(X_1, \ldots, X_N)$, ce qui signifie :

$$\forall \omega \in \Omega,\ T(\omega) \ = \ \max\left(X_1(\omega), \ldots, X_{N(\omega)}(\omega)\right)$$

Ainsi par exemple, si N prend la valeur 3, alors $T = \max(X_1, X_2, X_3)$; si N prend la valeur 5, alors $T = \max(X_1, X_2, X_3, X_4, X_5)$; etc.
On dispose d'une fonction **Python** `simuleT()` qui renvoie une simulation de la variable aléatoire T.

On considère la fonction **Python** suivante :

```
def mystere():
    m = np.zeros(3)
    for k in range(3):
        s = np.zeros(1000)
        for j in range(1000):
            s[j] = simuleT()
        m[k] = np.mean(s)
    return m
```

À son appel, on obtient :

```
ans  =
    0.7474646     0.7577248     0.7470916
```

Que renvoie la fonction `mystere` ? Que peut-on conjecturer sur la variable aléatoire T ?

4.17.11. ECRICOME 2020

Contexte

On considère une variable aléatoire X admettant f pour densité, où

$$f : t \mapsto \begin{cases} 0 & \text{si } t < a \\ \dfrac{3\,a^3}{t^4} & \text{si } t \geqslant a \end{cases}$$

On dispose d'une fonction **Python** nommée `simulX(a, m, n)` prenant en argument un réel a strictement positif et deux entiers naturels m et n non nuls, qui renvoie une matrice à m lignes et n colonnes dont chaque coefficient est un réel choisi de façon aléatoire et indépendant des autres en suivant la loi de X.
On a calculé $\mathbb{P}_{[X>2a]}([X > 6a]) = \dfrac{1}{27}$ à la question précédente.

Compléter le script ci-dessous afin qu'il renvoie une valeur permettant de vérifier le résultat de la question précédente.

```
a = 10
N = 100000
s1 = 0
s2 = 0
X = simulX(a, 1, N)[0]
for k in range(N):
    if _______________:
        s1 = s1 + 1
        if X[k] > 6*a:
            _______________
if s1 > 0:
    print(_________)
```

4.17.12. EML 2020

Contexte

Soient a et b deux réels strictement positifs. On définit la fonction f sur $\mathbb{R}$ par :

$$f : x \mapsto \begin{cases} 0 & \text{si } x > b \\ a\,\dfrac{b^a}{x^{a+1}} & \text{si } x \geqslant b \end{cases}$$

On admet que f est une densité de probabilité.
On dit qu'une variable aléatoire suit la loi de Pareto de paramètres a et b lorsqu'elle admet pour densité la fonction f.
On considère une variable aléatoire X suivant la loi de Pareto de paramètres a et b.
On dispose d'une fonction **Python** d'en-tête `def pareto(a,b):` qui prend en arguments deux réels a et b strictement positifs et qui renvoie une simulation de la variable aléatoire X.

1. On considère la fonction **Python** ci-dessous.
Que contient la liste `L` renvoyée par la fonction `mystere` ?

```
1  def mystere(a,b):
2      L = []
3      for p in range(2,7):
4          S = 0
5          for k in range(10**p):
6              S = S + pareto(a,b)
7          L.append(S/10**p)
8      return L
```

2. On exécute la fonction précédente avec différentes valeurs de a et de b.
Comment interpréter les résultats obtenus ?

```
-->  mystere(2,1)
    ans =
        1.9306917   1.9411352   1.9840089   1.9977684   2.0012415
-->  mystere(3,2)
    ans =
        3.1050951   3.0142956   2.9849407   2.9931656   2.9991517
-->  mystere(1,4)
    ans =
        21.053151   249.58609   51.230522   137.64549   40.243918
```

4.17.13. ECRICOME 2019

Contexte

Soit f la fonction définie sur $\mathbb{R}$ par :

$$\forall t \in \mathbb{R},\ f(t) = \begin{cases} \dfrac{1}{t^3} & \text{si } t \geqslant 1 \\ 0 & \text{si } -1 < t < 1 \\ -\dfrac{1}{t^3} & \text{si } t \leqslant -1 \end{cases}$$

On admet que f est une densité et on considère une variable aléatoire X qui admet f pour densité. On pose $Y = |X|$.
Soit D une variable aléatoire prenant les valeurs -1 et 1 avec équiprobabilité, indépendante de la variable aléatoire Y.
Soit T la variable aléatoire définie par $T = DY$. On admet que T admet une espérance.
Soit U une variable aléatoire suivant la loi uniforme sur $]0, 1[$ et V la variable aléatoire définie par : $V = \dfrac{1}{\sqrt{1-U}}$. On admet que les variables V et Y suivent la même loi.
On dispose d'une fonction en langage **Python**, nommée `D(n)`, qui prend un entier $n \geqslant 1$ en entrée, et renvoie une matrice ligne contenant n réalisations de la variable aléatoire D.

On considère le script suivant :

```
1 n = int(input('entrer n'))
2 a = D(n)
3 b = rd.random(n)
4 c = a / np.sqrt(1-b)
5 print(sum(c)/n)
```

De quelle variable aléatoire les coefficients du vecteur `c` sont- ils une simulation ? Pour `n` assez grand, quelle sera la valeur affichée ? Justifier votre réponse.

4.17.14. HEC 2019

Contexte

On note S une variable aléatoire à valeurs dans $\{-1, 1\}$ dont la loi est donnée par :

$$\mathbb{P}([S = -1]) \ = \ \mathbb{P}([S = +1]) \ = \ \frac{1}{2}$$

On note X_2 une variable aléatoire qui suit la loi binomiale $\mathcal{B}\left(2, \dfrac{1}{2}\right)$.
On suppose que les variables aléatoires X_2 et S sont indépendantes et on pose $Y_2 = S\,X_2$.
On admet que la variable aléatoire $X_2 - (S + 1)$ suit la même loi que Y_2.

Le script **Python** suivant permet d'effectuer des simulations de la variable aléatoire Y_2 définie dans la question précédente.

```
1  n = 10
2  X = rd.binomial(2,0.5,[n,2])
3  B = rd.binomial(1,0.5,[n,2])
4  S = 2*B - np.ones([n,2])
5  Z1 = [S[:,0]*X[:,0], X[:,0] - S[:,0] - np.ones(n)]
6  Z2 = [S[:,0]*X[:,0], X[:,1] - S[:,1] - np.ones(n)]
```

1. Que contiennent les variables `X` et `S` après l'exécution des quatre premières instructions ?

2. Expliquer pourquoi, après l'exécution des six instructions, chacun des coefficients des matrices `Z1` et `Z2` contient une simulation de la variable aléatoire Y_2.

3. On modifie la première ligne du script précédent en affectant à `n` une valeur beaucoup plus grande que 10 (par exemple, 100000) et en lui adjoignant les deux instructions 7 et 8 suivantes :

```
7  p1 = len(np.argwhere(Z1[0] == Z1[1])) / n
8  p2 = len(np.argwhere(Z2[0] == Z2[1])) / n
```

Quelles valeurs numériques approchées la loi faible des grands nombres permet-elle de fournir pour `p1` et `p2` après l'exécution des huit lignes du nouveau script ?

Dans le langage **Python**, la fonction `len` fournit la « longueur » d'un vecteur, d'une liste ou d'une matrice carrée et la fonction `np.argwhere` calcule les positions des coefficients d'une matrice pour lesquels une propriété est vraie, comme l'illustre le script suivant :

```
--> A = np.array([1,2,0,4])
--> B = np.array([2,2,4,3])
--> len(A)
    ans = 4.
--> np.argwhere(A < B)
    = [[0]
       [2]]
    # car 1 < 2 et 0 < 4, alors que 2 ⩾ 2 et 4 ⩾ 3
```

4.17.15. EDHEC 2018

Contexte

On considère la fonction f qui à tout réel x associe : $f(x) = \displaystyle\int_0^x \ln(1+t^2)\,dt$.

On rappelle qu'en **Python**, la commande `rd.random(d)` simule un d-échantillon de la loi uniforme sur $[0,1]$. Compléter le script **Python** suivant pour qu'il calcule et affiche, à l'aide de la méthode de Monte-Carlo, une valeur approchée de $f(1)$:

```
1  import numpy as np
2  import numpy.random as rd
3  U = rd.random(100000)
4  V = np.log(1 + U**2)
5  f = _______________
6  print(f)
```

4.17.16. EML 2018

Contexte

Dans cette partie, p désigne un réel de $]0,1[$.

Deux individus A et B s'affrontent dans un jeu de Pile ou Face dont les règles sont les suivantes :

- le joueur A dispose d'une pièce amenant Pile avec la probabilité $\frac{2}{3}$ et lance cette pièce jusqu'à l'obtention du deuxième Pile ; on note X la v.a.r. prenant la valeur du nombre de Face alors obtenus ;
- le joueur B dispose d'une autre pièce amenant Pile avec la probabilité p et lance cette pièce jusqu'à l'obtention d'un Pile ; on note Y la v.a.r. prenant la valeur du nombre de Face alors obtenus ;
- le joueur A gagne si son nombre de Face obtenus est inférieur ou égal à celui de B ; sinon c'est le joueur B qui gagne.

On dispose d'une fonction **Python** `simule_X()` (resp. `simule_Y(p)`) qui simule la v.a.r. X (resp. la v.a.r. Y).

Expliquer ce que renvoie la fonction suivante :

```
def mystere(p):
    r = 0
    N = 10**4
    for k in range(N):
        x = simule_X()
        y = simule_Y(p)
        if x <= y:
            r = r + 1/N
    return r
```

4.17.17. ESSEC I 2018

Contexte

Dans tous le sujet :

- on désigne par n un entier naturel, au moins égal à 2,
- X est une v.a.r. à valeurs dans un intervalle $]0,\alpha[$, où α est un réel strictement positif. On suppose que X admet une densité f strictement positive et continue sur $]0,\alpha[$, et nulle en dehors de $]0,\alpha[$.
- on note F la fonction de répartition de X.
- X_1, …, X_n est une famille de v.a.r. mutuellement indépendantes et de même loi que X.

On définit deux variables aléatoires Y_n et Z_n de la façon suivante.
Pour tout $\omega \in \Omega$:

- $Y_n(\omega) = \max(X_1(\omega), \ldots, X_n(\omega))$ est le plus grand des réels $X_1(\omega)$, …, $X_n(\omega)$;
  on remarque que Y_n est définie également lorsque n vaut 1, de sorte que dans la suite du sujet on pourra considérer Y_{n-1}.
- $Z_n(\omega)$ est le « deuxième plus grand » des nombres $X_1(\omega)$, …, $X_n(\omega)$, autrement dit, une fois que ces n réels sont ordonnés dans l'ordre croissant, Z_n est l'avant-dernière valeur. On note que lorsque la plus grande valeur est présente plusieurs fois, $Z_n(\omega)$ et $Y_n(\omega)$ sont égaux.

On dispose d'une fonction **Python** `simulX(n)` qui retourne une simulation d'un échantillon de taille n de la loi de X sous la forme d'un vecteur de longueur n.

On considère la fonction φ_x définie sur $\mathbb{R}_+$ par : $\varphi_x(t) = \begin{cases} t & \text{si } t \leqslant x \\ 0 & \text{sinon} \end{cases}$.

On pose, pour tout $x \in]0,\alpha[$, $\sigma(x) = \dfrac{\mathbb{E}(\varphi_x(Y_{n-1}))}{\mathbb{P}([Y_{n-1} \leqslant x])}$.

Écrire une fonction **Python** `sigma(x,n)` qui retourne une valeur approchée de $\sigma(x)$ obtenue comme quotient d'une estimation de $\mathbb{E}(\varphi_x(Y_{n-1}))$ et de $\mathbb{P}([Y_{n-1} \leqslant x])$.
On utilisera la fonction `simulX` pour simuler des échantillons de la loi de X, et on rappelle que si `v` est un vecteur, `max(v)` est égal au plus grand élément de `v`.

4.17.18. ECRICOME 2017

Contexte

Soit n un entier naturel non nul.
On effectue une série illimitée de tirages d'une boule avec remise dans une urne contenant n boules numérotées de 1 à n. Pour tout entier naturel k non nul, on note X_k la variable aléatoire égale au numéro de la boule obtenue au $k^{\text{ème}}$ tirage.
Pour tout entier naturel k non nul, on note S_k la somme des numéros des boules obtenues lors des k premiers tirages :

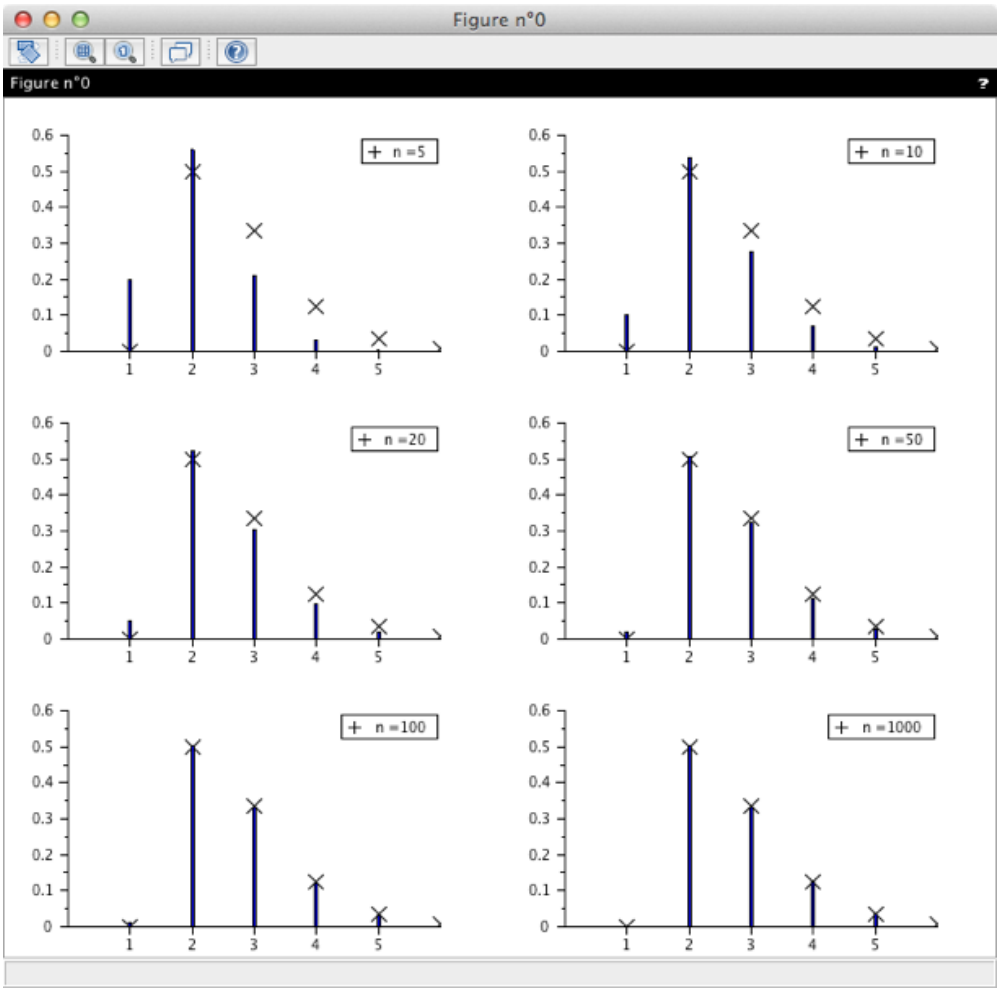
$$S_k = \sum_{i=1}^{k} X_i$$

On considère enfin la variable aléatoire T_n égale au nombre de tirages nécessaires pour que, pour la première fois, la somme des numéros des boules obtenues soit supérieure ou égale à n.
Exemple : avec $n = 10$, si les numéros obtenus aux cinq premiers tirages sont dans cet ordre 2, 4, 1, 5 et 9, alors on obtient : $S_1 = 2$, $S_2 = 6$, $S_3 = 7$, $S_4 = 12$, $S_5 = 21$ et $T_{10} = 4$.
On dispose d'une fonction **Python** `simulT(n)` qui simule la variable aléatoire T_n.
Soit Y une variable aléatoire à valeurs dans $\mathbb{N}^*$ telle que : $\forall k \in \mathbb{N}^*$, $\mathbb{P}([Y = k]) = \dfrac{k-1}{k!}$.
On admet que $(T_n)_{n\geqslant 1}$ converge en loi vers la variable aléatoire Y $(*)$.

On admet que l'appel `math.factorial(k)` renvoie le nombre $k!$.
On suppose déclarée la fonction précédente et on écrit le script ci-dessous :

```
1  def freqT(n):
2      tab = np.zeros(n)
3      for i in range(100000):
4          k = simulT(n)
5          tab[k-1] = tab[k-1] + 1
6      return tab / 100000
7
8  def loitheoY(n):
9      tab = np.zeros(n)
10     for k in range(1, n+1):
11         tab[k-1] = (k-1) / math.factorial(k)
12     return tab
13
14 n = int(input('Rentrer la valeur de n:'))
15 Labs = range(1, 6)
16 x = freqT(n)
17 debutfreqT = [x[k] for k in range(5)]
18 plt.bar(Labs, debutfreqT, width=0.1)
19 plt.plot(Labs, loitheoY(5), 'kx')
```

L'exécution de ce script pour les valeurs de n indiquées a permis d'obtenir les graphes page suivante.



1. Expliquer ce que représentent les vecteurs renvoyés par les fonctions `freqT` et `loitheoY`. Comment ces vecteurs sont-ils représentés graphiquement dans chaque graphique obtenu ?
2. Expliquer en quoi cette succession de graphiques permet d'illustrer la propriété $(*)$.

4.17.19. EDHEC 2017

Contexte

Soit V une variable aléatoire suivant la loi exponentielle de paramètre 1, dont la fonction de répartition est la fonction F_V définie par : $F_V(x) = \begin{cases} 0 & \text{si } x \leqslant 0 \\ 1 - \mathrm{e}^{-x} & \text{si } x > 0 \end{cases}$.
On pose $W = -\ln(V)$ et on admet que W est aussi une variable aléatoire dont le fonction de répartition est notée F_W. On dit que W suit une loi de Gumbel.
On désigne par n un entier naturel non nul et par $X_1, \ldots, X_n$ des variables aléatoires définies sur le même espace probabilisé, indépendantes et suivant la même loi que V, c'est à dire la loi $\mathcal{E}(1)$.
On considère la variable aléatoire Y_n définie par $Y_n = \max(X_1, X_2, \ldots, X_n)$, c'est à dire que pour tout ω de Ω, on a : $Y_n(\omega) = \max(X_1(\omega), X_2(\omega), \ldots, X_n(\omega))$. On admet que Y_n est une variable aléatoire à densité.
On pose $Z_n = Y_n - \ln(n)$.
On dispose d'une fonction **Python** `simulZ(n)` qui simule la variable aléatoire Z_n.

Voici deux scripts :

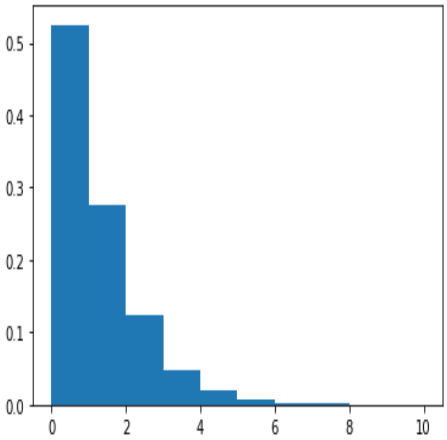
```
V = rd.exponential(1,10000)
W =-np.log(V)
s = np.linspace(0,10,11)
plt.hist(W,s,density=True)
plt.show()
```

Script (1)

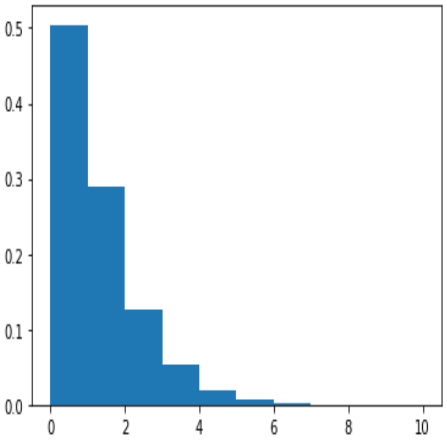
```
n=int(input('entrez la valeur de n :'))
Z = [] # La liste Z est vide
for k in range(10000):
    Z.append(simulZ(n))
s = np.linspace(0,10,11)
plt.hist(Z,s,density = True)
plt.show()
```

Script (2)

Chacun des scripts simule 10000 variables indépendantes, regroupe les valeurs renvoyées en 10 classes qui sont les intervalles $[0,1]$, $]1,2]$, $]2,3]$, …, $]9,10]$ et trace l'histogramme correspondant (la largeur de chaque rectangle est égale à 1 et leur hauteur est proportionnelle à l'effectif de chaque classe).
Le script (1) dans lequel les variables aléatoires suivent la loi de Gumbel (loi suivie par W), renvoie l'histogramme (1) ci-dessous, alors que le script (2) dans lequel les variables aléatoires suivent la même loi que Z_n, renvoie l'histogramme (2) ci-dessous, pour lequel on a choisi
$n = 1000$.



Histogramme (1)



Histogramme (2) pour $n = 1000$

Quelle conjecture peut-on émettre quant au comportement de la suite des v.a.r. (Z_n) ?

4.17.20. EML 2017

Contexte

On considère une urne contenant initialement une boule bleue et deux boules rouges.
On effectue, dans cette urne, des tirages successifs de la façon suivante : on pioche une boule au hasard et on note sa couleur, puis on la replace dans l'urne en ajoutant une boule de la même couleur que celle qui vient d'être obtenue.
On dispose d'une fonction **Python** `EML(n)` qui simule l'expérience étudiée et renvoie le nombre de boules rouges obtenues lors des n premiers tirages.

On exécute le programme suivant :

```
n = 10
m = 0
for i in range(1000):
    m = m + EML(n)
print(m/1000)
```

On obtient 6.657. Comment interpréter ce résultat ?

4.17.21. HEC 2017

Contexte

Dans tout le problème, on note :

- a et b deux réels strictement positifs ;
- $f_{a,b}$ la fonction définie sur $\mathbb{R}$ par : $f_{a,b}(x) = \begin{cases} (a+bx)\ \exp\left(-ax - \dfrac{b}{2}x^2\right) & \text{si } x \geqslant 0 \\ 0 & \text{si } x < 0 \end{cases}$.

On dit qu'une variable aléatoire suit la loi exponentielle linéaire de paramètres a et b, notée $\mathcal{E}_\ell(a,b)$, si elle admet $f_{a,b}$ pour densité.

On dispose d'une fonction **Python** `simulLinExp(a,b,n)` qui renvoie n simulations indépendantes de la loi exponentielle linéaire de paramètres a et b.

De quel nombre réel peut-on penser que les six valeurs générées par la boucle **Python** suivante fourniront des valeurs approchées de plus en plus précises et pourquoi ?

```
for k in range(1,7):
    print(np.mean(simulLinExp(0, 1, 10**k)))
```