

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	6
1.1. Calcul de rang	6
1.1.1. EDHEC 2025	6
1.1.2. EDHEC 2022	6
1.2. Calcul d'une somme de matrices	7
1.2.1. ECRICOME 2026	7
1.3. Puissance d'une matrice	8
1.3.1. ECRICOME 2026	8
1.3.2. EML 2026	8
1.3.3. EDHEC 2025	9
1.3.4. EDHEC 2023	9
1.3.5. ESSEC II 2023	10
1.3.6. EDHEC 2017	10
1.4. Récupération des valeurs propres d'une matrice	11
1.4.1. ESSEC II 2024	11
1.4.2. HEC 2017	12
1.5. Modification d'une matrice	13
1.5.1. HEC/ESSEC I 2025	13
1.5.2. HEC 2019	15
1.6. Création d'une matrice dont les coefficients sont donnés explicitement	16
1.6.1. EDHEC 2025	16
1.6.2. ESSEC II 2024	16
1.7. Création d'une matrice dont les coefficients sont reliés par des relations de récurrence	17
1.7.1. HEC 2018	17
1.8. Calcul du n^{e} terme d'une suite de vecteurs colonnes	19
1.8.1. ECRICOME 2018	19
1.9. Calcul de normes	20
1.9.1. ESSEC I 2026	20
2. Analyse	21
2.1. Algorithme de dichotomie	21
2.1.1. ESSEC II 2025	21
2.1.2. ECRICOME 2023	21
2.1.3. EML 2021	22
2.1.4. EML 2020	22
2.1.5. ECRICOME 2019	23
2.2. Calcul d'une somme	24

2.2.1.	ESSEC II 2025	24
2.2.2.	EDHEC 2025	24
2.2.3.	ECRICOME 2023	25
2.2.4.	HEC/ESSEC I 2021	26
2.2.5.	ESSEC II 2019	27
2.3.	Calcul d'un produit	28
2.3.1.	HEC/ESSEC I 2022	28
2.3.2.	EDHEC 2019	28
2.4.	Calcul du n^{e} terme d'une suite récurrente	29
2.4.1.	EDHEC 2026	29
2.4.2.	EDHEC 2024	29
2.4.3.	EDHEC 2023	30
2.4.4.	EML 2023	30
2.4.5.	EML 2019	30
2.4.6.	EML 2018	31
2.4.7.	ECRICOME 2018	31
2.5.	Calcul d'une valeur approchée de la limite d'une suite	32
2.5.1.	ECRICOME 2026	32
2.5.2.	EDHEC 2023	32
2.5.3.	EDHEC 2022	32
2.5.4.	EML 2019	33
2.5.5.	EML 2018	33
2.5.6.	ECRICOME 2018	34
2.6.	Calcul et stockage des n premiers termes d'une suite récurrente	35
2.6.1.	ECRICOME 2022	35
2.7.	Suites récurrentes linéaires couplées	36
2.7.1.	EDHEC 2021	36
2.8.	Suites récurrentes linéaires doubles	37
2.8.1.	EML 2026	37
2.8.2.	EDHEC 2020	37
2.9.	Calcul du premier entier n vérifiant une propriété donnée	39
2.9.1.	ESSEC II 2025	39
2.9.2.	EML 2025	39
2.9.3.	EML 2023	39
2.9.4.	HEC/ESSEC I 2020	40
2.9.5.	EML 2017	40
2.10.	Codage d'une fonction d'une variable ou de deux variables	41
2.10.1.	EDHEC 2026	41
2.10.2.	ECRICOME 2024	41
2.10.3.	HEC/ESSEC I 2023	41
2.10.4.	ECRICOME 2019	42
2.10.5.	EDHEC 2017	42
2.11.	Tracé du graphe d'une fonction	43
2.11.1.	EML 2026	43
2.11.2.	EDHEC 2026	44
2.11.3.	HEC/ESSEC I 2023	44
2.11.4.	ESSEC II 2023	45
2.12.	Tracé d'une suite	46
2.12.1.	EDHEC 2024	46
2.12.2.	ECRICOME 2019	47
2.13.	Tracé d'une trajectoire de système différentiel	48

2.13.1. EML 2024	48
3. Graphes	49
3.1. Algorithme de coloration des graphes	49
3.1.1. EML 2025	49
3.2. Algorithme de recherche du plus court chemin	51
3.2.1. ECRICOME 2023	51
3.3. Création de la liste d'adjacence d'un graphe	53
3.3.1. ECRICOME 2023	53
3.4. Comptage du nombre de triangles dans un graphe aléatoire	54
3.4.1. HEC/ESSEC I 2024	54
3.5. Degrés des sommets d'un graphe	55
3.5.1. ESSEC II 2024	55
4. Probabilités	56
4.1. Simulation de variables aléatoires discrètes à l'aide de la bibliothèque pandas	56
4.1.1. HEC/ESSEC I 2023	56
4.2. Simulation de variables aléatoires discrètes suivant une loi usuelle	57
4.2.1. ESSEC II 2026	57
4.3. Simulation de variables aléatoires discrètes liées à une expérience concrète	58
4.3.1. EDHEC 2025	58
4.3.2. EML 2024	59
4.3.3. ECRICOME 2022	59
4.3.4. EDHEC 2022	60
4.3.5. EML 2022	61
4.3.6. ECRICOME 2021	62
4.3.7. EDHEC 2021	63
4.3.8. EML 2021	64
4.3.9. ESSEC II 2020	64
4.3.10. EDHEC 2019	65
4.3.11. EDHEC 2018	66
4.3.12. EML 2018	67
4.3.13. ECRICOME 2017	67
4.3.14. EML 2017	67
4.4. Simulation de variables aléatoires discrètes via la méthode d'inversion	69
4.4.1. ESSEC II 2022	69
4.4.2. ECRICOME 2019	69
4.5. Simulation de variables aléatoires discrètes définies comme le minimum d'un ensemble aléatoire	70
4.5.1. ESSEC II 2026	70
4.5.2. ESSEC II 2025	72
4.6. Simulation de sommes de variables aléatoires discrètes	73
4.6.1. ECRICOME 2025	73
4.6.2. ESSEC II 2021	73
4.7. Simulation d'un couple de variables aléatoires discrètes liées à une expérience concrète	74
4.7.1. EML 2026	74
4.7.2. ECRICOME 2023	75
4.8. Simulation d'un couple de variables aléatoires discrètes via des lois usuelles lors d'une expérience aléatoire en deux étapes	77
4.8.1. ECRICOME 2026	77
4.8.2. EDHEC 2020	77
4.9. Simulation d'un couple de variables aléatoires discrètes via sa loi de couple	78

4.9.1. HEC 2018	78
4.10. Simulation d'une chaîne de Markov	79
4.10.1. HEC/ESSEC I 2025	79
4.10.2. ESSEC II 2023	80
4.10.3. EDHEC 2017	82
4.11. Simulation de variables aléatoires à densité via la méthode d'inversion	84
4.11.1. ECRICOME 2025	84
4.11.2. ECRICOME 2020	84
4.11.3. EML 2020	85
4.11.4. HEC 2017	85
4.12. Simulation de variables aléatoires à densité à partir de variables aléatoires suivant une loi exponentielle	86
4.12.1. EDHEC 2024	86
4.12.2. EDHEC 2023	86
4.12.3. ECRICOME 2021	86
4.12.4. EDHEC 2019	87
4.12.5. EDHEC 2018	87
4.12.6. EDHEC 2017	87
4.12.7. ESSEC I 2017	88
4.13. Simulation de sommes de variables aléatoires à densité	89
4.13.1. ECRICOME 2020	89
4.13.2. EDHEC 2020	89
4.13.3. HEC/ESSEC I 2020	90
4.14. Simulation d'une variable aléatoire à densité lors d'une expérience en deux étapes	92
4.14.1. ECRICOME 2025	92
4.15. Simulation d'un couple de variables aléatoires à densité	93
4.15.1. ESSEC II 2025	93
4.16. Simulation d'une matrice aléatoire dont les coefficients suivent des lois à densité	94
4.16.1. ESSEC I 2026	94
4.17. Simulation de variables aléatoires suivant une « loi composée »	95
4.17.1. EML 2021	95
4.17.2. HEC/ESSEC I 2021	95
4.18. Simulation du maximum/minimum de plusieurs variables aléatoires à densité	97
4.18.1. EDHEC 2026	97
4.18.2. EDHEC 2025	97
4.18.3. EDHEC 2024	97
4.18.4. ESSEC I 2018	98
4.19. Tracé d'un diagramme en bâtons	99
4.19.1. ECRICOME 2025	99
4.20. Utilisation de la loi faible des grands nombres	100
4.20.1. EML 2026	100
4.20.2. EDHEC 2026	101
4.20.3. ESSEC II 2025	101
4.20.4. ECRICOME 2025	102
4.20.5. ECRICOME 2024	102
4.20.6. EML 2024	103
4.20.7. HEC/ESSEC I 2024	103
4.20.8. ECRICOME 2023	104
4.20.9. EDHEC 2023	104
4.20.10. ECRICOME 2022	105
4.20.11. ECRICOME 2021	105

4.20.12.EML 2021	106
4.20.13.ECRICOME 2020	107
4.20.14.EML 2020	107
4.20.15.ECRICOME 2019	108
4.20.16.HEC 2019	109
4.20.17.EDHEC 2018	110
4.20.18.EML 2018	110
4.20.19.ESSEC I 2018	111
4.20.20.ECRICOME 2017	112
4.20.21.EDHEC 2017	113
4.20.22.EML 2017	114
4.20.23.HEC 2017	115
4.21. Statistiques bivariées	116
4.21.1. ECRICOME 2026	116

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 :

```
1 r1=al.matrix_rank(matA()-5*np.eye(3,3))
2 r2=al.matrix_rank(matA()+4*np.eye(3,3))
3 print('r1=',r1)
4 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 $\mathcal{M}_2(\mathbb{R})$ 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 :

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

Python a renvoyé :

```
1 r1=3
2 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. Calcul d'une somme de matrices

1.2.1. ECRICOME 2026

Contexte

Soit n un entier naturel non nul.

Une matrice de $\mathcal{M}_n(\mathbb{R})$ est dite **nilpotente** quand il existe un entier naturel k tel que $M^k = 0_n$, où 0_n désigne la matrice nulle de $\mathcal{M}_n(\mathbb{R})$.

On admet que, pour toute matrice M **nilpotente** de $\mathcal{M}_n(\mathbb{R})$, il existe un entier naturel p non nul et inférieur ou égal à n tel que $M^{p-1} \neq 0_n$ et $\forall k \geq p, M^k = 0_n$. p est appelé **l'indice de nilpotence** de M .

On dispose d'une fonction `Nil`, qui prend en entrée une matrice M et qui renvoie p lorsque la matrice M est nilpotente d'indice de nilpotence p et 0 sinon.

On considère N une matrice nilpotente non nulle de $\mathcal{M}_n(\mathbb{R})$. On note p l'indice de nilpotence de N .

Pour tout réel t , on définit la matrice $B(t)$ de $\mathcal{M}_n(\mathbb{R})$ par

$$B(t) = \sum_{k=0}^{p-1} \frac{t^k}{k!} N^k.$$

Recopier et compléter, en langage **Python**, la fonction suivante qui prend en entrée la matrice `N` et un réel `t` et qui renvoie $B(t)$.

```

1  def B(N,t):
2      if Nil(N)==...:
3          return('N n'est pas nilpotente')
4      else:
5          T=np.eye(len(N))
6          S=T
7          for k in range(...):
8              T=.../...*np.dot(T,...)
9              S=S+T
10         return S

```

1.3. Puissance d'une matrice

1.3.1. ECRICOME 2026

Contexte

Soit n un entier naturel non nul.

Une matrice de $\mathcal{M}_n(\mathbb{R})$ est dite **nilpotente** quand il existe un entier naturel k tel que $M^k = 0_n$, où 0_n désigne la matrice nulle de $\mathcal{M}_n(\mathbb{R})$.

On admet que, pour toute matrice M **nilpotente** de $\mathcal{M}_n(\mathbb{R})$, il existe un entier naturel p non nul et inférieur ou égal à n tel que $M^{p-1} \neq 0_n$ et $\forall k \geq p, M^k = 0_n$. p est appelé **l'indice de nilpotence** de M .

Écrire une fonction, en langage **Python**, nommée `Nil`, qui prend en entrée une matrice M et qui renvoie p lorsque la matrice M est nilpotente d'indice de nilpotence p et 0 sinon.

Aide mémoire. Si M et N sont deux tableaux Numpy la commande `(M==N).all()` renvoie `True` si $M = N$ et `False` sinon.

1.3.2. EML 2026

Contexte

On note $(P_n)_{n \geq 0}$ la suite de polynômes de Tchebychev définie par ses deux premiers termes

$$\forall x \in \mathbb{R}, \quad P_0(x) = 0 \quad \text{et} \quad P_1(x) = 1$$

et la relation de récurrence, valable pour tout entier $n \geq 1$,

$$\forall x \in \mathbb{R}, \quad P_{n+1}(x) = xP_n(x) - P_{n-1}(x).$$

On dispose d'une fonction `P(n,x)` qui prend en argument un entier naturel n et un réel x , et qui renvoie le nombre $P_n(x)$.

Soit M une matrice de $\mathcal{M}_2(\mathbb{R})$ de déterminant 1, c'est-à-dire avec $\det(M) = 1$.

On définit une suite de nombres réels $(a_n)_{n \geq 0}$ en posant, pour tout $n \in \mathbb{N}$,

$$a_n = P_n(\text{tr}(M))$$

On admet que, pour tout $n \geq 1$, on a :

$$M^n = -a_{n-1}I + a_n M \quad (*)$$

En utilisant la fonction « `P(n,x)` » ainsi que l'identité $(*)$ ci-dessus, rédiger une fonction **Python** « `Puissance(n,M)` » qui prend en argument un entier $n \geq 1$ et une matrice M de déterminant 1, et qui renvoie la matrice M^n . On suppose la matrice M construite à l'aide de la commande « `np.array` ».

Rappel Python : La commande « `np.eye(2)` » renvoie la matrice identité d'ordre 2.

1.3.3. 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 $2a - 2b$.

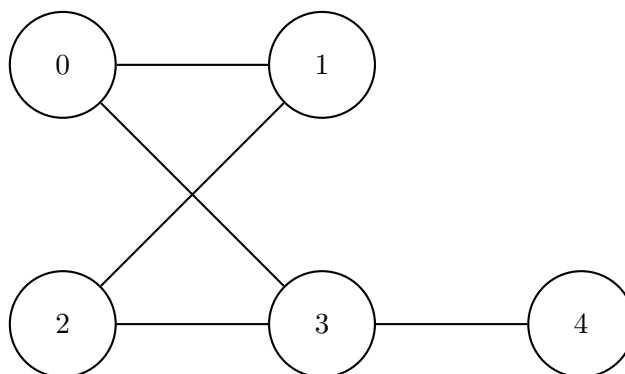
On admet que, pour tout $n \in \mathbb{N}^*$, $M(a, b)^n = PD^nP^{-1}$.

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 $M(a, b)^n$.

1.3.4. 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.

1.3.5. 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 $(I_n + \frac{t}{1000}G)^{1000}$.

1.3.6. 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 :

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

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(---)
```

1.4. Récupération des valeurs propres d'une matrice

1.4.1. 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 $\mathcal{M}_n(\mathbb{R})$.

On admet qu'il existe une matrice carrée inversible P appartenant à $\mathcal{M}_n(\mathbb{R})$ 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 $\lambda_1, \dots, \lambda_n$ 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 $A = (a_{i,j})_{1 \leq i,j \leq n}$ est une matrice carrée appartenant à $\mathcal{M}_n(\mathbb{R})$, on définit la trace de A , notée $\text{tr}(A)$ par :

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

On admet que :

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

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

1.4.2. HEC 2017

Contexte

Soit B la matrice de $\mathcal{M}_3(\mathbb{R})$ 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.

1.5. Modification d'une matrice

1.5.1. HEC/ESSEC I 2025

Contexte

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

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

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

$$\forall (i, j) \in \llbracket 1, r \rrbracket^2, \mu_i m_{i,j} = \mu_j m_{j,i}$$

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 $(i, j) \in \llbracket 1, r \rrbracket^2$, $m_{i,j}^{(\sigma)} > 0$, on dit que M est une matrice ergodique. On définit aussi pour tout $n \in \mathbb{N}^*$ et $(i_0, \dots, i_n) \in \llbracket 1, r \rrbracket^{n+1}$:

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

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

$$\theta_M(i_0, i_1, \dots, i_n, i_0) = \theta_M(i_0, i_n, \dots, i_1, i_0) \quad (K)$$

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 $M = (m_{i,j})_{1 \leq i,j \leq r}$ appartenant à $\mathcal{ST}_r$ comme suit :

- On modifie la ligne k de M , où $k \in \llbracket 1, r \rrbracket$, 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 $k \in \llbracket 1, r \rrbracket$, 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 $M \in \mathcal{ST}_r$, $M = (m_{i,j})_{1 \leq i,j \leq r}$, une matrice que l'on notera dans la suite $M' = (m'_{i,j})_{1 \leq i,j \leq r}$ 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 $\llbracket 1, r \rrbracket$ et $M \in \mathcal{ST}_r$, $M = (m_{i,j})_{1 \leq i,j \leq r}$.

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 $M \in \mathcal{ST}_r$, $M = (m_{i,j})_{1 \leq i,j \leq r}$ 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 $G_{M^*}(\llbracket 1, r \rrbracket)$ est connexe.

```

I = {1}
Tant que I ≠ ⌊1, r⌋:
    On détermine un couple (ℓ, k) ∈ I × (⌊1, r⌋ \ I) tel que mℓ,k ≠ 0 et mk,ℓ ≠ 0
    Si mℓ,k ≤ mk,ℓ:
        On pose α = mℓ,k / mk,ℓ et on fait l'opération élémentaire sur la ligne k
        avec ce coefficient
    Sinon:
        on pose α = mk,ℓ / mℓ,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 $M = (m_{i,j})_{1 \leq i,j \leq r}$ 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.

```

1  def estRev(M):
2      r=np.shape(M)[0]
3      I=[1]; J=[k for k in range(2,r+1)]
4      while len(I)< ... :
5          ell,k=NonNul(M,I,J)
6          if (ell==0) or M[k-1][ell-1]*M[ell-1][k-1]==0:
7              return ...
8          else:
9              if M[ell-1][k-1] <= M[k-1][ell-1]:
10                 OpLigne(M,k,M[ell-1][k-1]/M[k-1][ell-1])
11             else:
12                 OpCol(M,k,M[k-1][ell-1]/M[ell-1][k-1])
13             I.append(...)
14             J.remove(...)
15     return (np.transpose(M)==M).all()
16     # Teste l'égalité de deux matrices numpy

```

1.5.2. 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 $a \neq 0$).

```

1  def multlig(a, i, A):
2      n,p = np.shape(A)
3      B = np.copy(A)
4      for j in range(p):
5          B[i-1, j] = a * B[i-1, j]
6      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 :

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

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

```

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

```

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

1.6.1. 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 .

1.6.2. ESSEC II 2024

Contexte

- Soit $U = \begin{pmatrix} u & v \\ v & w \end{pmatrix}$ une matrice carrée appartenant à $\mathcal{M}_2(\mathbb{R})$ symétrique et A une matrice carrée appartenant à $\mathcal{M}_n(\mathbb{R})$ symétrique.

On définit $U * A$ (produit de Kronecker) la matrice carrée appartenant à $\mathcal{M}_{2n}(\mathbb{R})$ 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,

$$\text{d'où } U * A = \begin{pmatrix} 0 & 2 & 2 & 0 & 1 & 1 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (0_3 \text{ désigne la matrice nulle de } \mathcal{M}_3(\mathbb{R})).$$

- Si X est une matrice colonne appartenant à $\mathcal{M}_{2n,1}(\mathbb{R})$, $X = \begin{pmatrix} x_1 \\ \vdots \\ x_{2n} \end{pmatrix}$, on écrira $X = \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$ avec

$$X_1 = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \text{ et } X_2 = \begin{pmatrix} x_{n+1} \\ \vdots \\ x_{2n} \end{pmatrix}.$$

On admet qu'alors la matrice colonne $(U * A)X$ de $\mathcal{M}_{2n,1}(\mathbb{R})$ 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.]])
```

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

1.7.1. 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 $(x_1, x_2, \dots, x_k)$ d'entiers naturels tels que :

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

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

On admet que :

- Pour tout entier $k \in \mathbb{N}^*$: $Q(1, k) = \{(0, \dots, 0, 1)\}$.
- Pour tout entier $\ell \geq k$: $Q(\ell, k) = P(k)$.
- 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 $(\ell, k) \in \llbracket 1, n \rrbracket^2$, 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 $k \geq 1$, puis la démontrer.

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

1.8.1. ECRICOME 2018

Contexte

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

Soit B la matrice de $\mathcal{M}_3(\mathbb{R})$ 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 _____

```

1.9. Calcul de normes

1.9.1. ESSEC I 2026

Contexte

Pour tout d entier naturel non nul et $X \in \mathcal{M}_{d,1}(\mathbb{R})$, de coefficients $x_1, \dots, x_d$ on pose $\|X\| = \sqrt{\sum_{k=1}^d x_k^2}$.

On admet que l'on a défini une norme sur $\mathcal{M}_{d,1}(\mathbb{R})$, ce qui sous-entend en particulier que :

- Pour tout $X \in \mathcal{M}_{d,1}(\mathbb{R})$, si $\|X\| = 0$ alors $X = 0$.
- Pour tous $X \in \mathcal{M}_{d,1}(\mathbb{R})$, $Y \in \mathcal{M}_{d,1}(\mathbb{R})$, $\|X\| - \|Y\| \leq \|X + Y\| \leq \|X\| + \|Y\|$.
- Pour tous $X \in \mathcal{M}_{d,1}(\mathbb{R})$, $\lambda \in \mathbb{R}$, $\|\lambda X\| = |\lambda| \|X\|$.

Si $\mathcal{X}$ est un ensemble d'éléments de $\mathcal{M}_{n,1}(\mathbb{R})$, $\varepsilon \in]0, 1[$ et $A \in \mathcal{M}_{m,n}(\mathbb{R})$, on dit que A est une $(\varepsilon, \mathcal{X})$ -isométrie si pour tout $X \in \mathcal{X}$:

$$(1 - \varepsilon)\|X\|^2 \leq \|AX\|^2 \leq (1 + \varepsilon)\|X\|^2$$

- a) Ecrire une fonction `Norme(X)` qui renvoie $\|X\|$ si le vecteur numpy `X` représente le vecteur colonne X de $\mathcal{M}_{n,1}(\mathbb{R})$.
- b) On exécute les instructions suivantes :

```
1 X=np.array([1,0,1,1,1])
2 print(Norme(X));print(Norme((1/Norme(X))*X))
```

Quel affichage obtient-on dans la console ?

- On exécute le script suivant :

```
1 A = np.array([[2,1],[0,1],[1,0]]); X = np.array([1,-1])
2 eps = 0.6
3 print(Norme(np.dot(A,X))**2)
4 print(1-eps <= Norme(np.dot(A,X))**2/Norme(X)**2 <= 1+eps)
```

et on obtient l'affichage :

3.0

True

Expliquer ces résultats.

- On suppose que $\mathcal{X}$, un ensemble d'éléments de $\mathcal{M}_{n,1}(\mathbb{R})$, est représenté par la liste finie `LX` de vecteurs numpy, la matrice A par le tableau numpy `A` et ε par `eps`. Ecrire une fonction `EstIsom(A,LX,eps)` qui renvoie `True` si A est une $(\varepsilon, \mathcal{X})$ -isométrie et `False` sinon.

En exécutant le script,

```
1 LX=[np.array([-1,3]),np.array([1,-2])];
2 print(EstIsom(np.array([[2,1],[0,1],[1,0]]),LX,0.2))
```

quel affichage obtient-on dans la console ? On justifiera sa réponse.

2. Analyse

2.1. Algorithme de dichotomie

2.1.1. ESSEC II 2025

Contexte

On définit la fonction f sur $[0, 1]$ par $f(t) = (1 + t)e^{-t} - t$.

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 :

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

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 $f(c) \leq n$ 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.

```

1  def valeur_approchee(n):
2      a = 0
3      b = 1
4      while ...
5          c = (a+b)/2
6          if c**n + c - 1 > 0:
7              ...
8          else:
9              ...
10     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 $n \geq 1$ en entrée :

```

1  def approxv(n):
2      a = 1
3      b = 3
4      while (b-a) > 10**(-5):
5          c = (a+b)/2
6          if h(n,c) < 4 :
7              .....
8          else:
9              .....
10     .....

```

2.2. Calcul d'une somme

2.2.1. ESSEC II 2025

Contexte

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

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

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

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

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 :

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

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 $(\alpha_0, \alpha_1, \dots, \alpha_d)$ qui dépend généralement de facteurs biologiques.

Pour tout $k \in \llbracket 0, d \rrbracket$, on dit que α_k est la contagiosité de tout individu ayant été 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 $\alpha_0, \alpha_1, \dots, \alpha_d$ sont tels que, pour tout $k \in \llbracket 0, d \rrbracket$, $\alpha_k \in]0, 1[$ 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 $(\Omega, \mathcal{A}, \mathbb{P})$.

- 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 :

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

- 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 $n \in \mathbb{N}$, $r_n = \frac{n+2}{n+1}$.

On note Δ la matrice ligne $(\alpha_0 \dots \alpha_d)$.

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 $\{0, 1, 2, \dots, n\}$ où n est un entier naturel. On appelle **entropie** de X le réel :

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

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 $A = \{0, 1, \dots, n\}$ où n est un entier naturel. On suppose que le vecteur $\mathbf{P}$ de **Python** est tel que pour tout k de A , $\mathbf{P}[k] = \mathbb{P}([X = k])$. Compléter la fonction ci-dessous d'argument $\mathbf{P}$ qui renvoie l'entropie de X , c'est-à-dire $-\sum_{k=0}^n \mathbb{P}([X = k]) \log_2 (\mathbb{P}([X = k]))$.

```
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 $\mathbf{P}$.

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 $a \geq 2$ tels que :

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

où les γ_k sont des constantes fixées.

On suppose que le vecteur **Python** `gammaTab` contient les valeurs $\gamma_1, \dots, \gamma_a$.

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 $u_n = \int_0^1 (1-t^2)^n dt$. 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 2026

Contexte

On considère un nombre réel a strictement supérieur à 1, ainsi que la suite $(u_n)_{n \in \mathbb{N}}$ définie par la donnée de $u_0 = a$ et par la relation de récurrence, valable pour tout entier naturel n :

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

Compléter la fonction **Python** suivante afin qu'elle retourne la valeur de u_n pour des valeurs données de n et de a :

```

1 def suite_u(a,n):
2     u=-----
3     for k in range(1,n+1):
4         u=-----
5     return u

```

2.4.2. EDHEC 2024

Contexte

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

On admet que :

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

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

```

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

```

2.4.3. EDHEC 2023

Contexte

On considère la fonction f définie par :

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

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)` :

```

1  def suite(n):
2      u = -----
3      for k in range(1, n+1):
4          u = -----
5      return u

```

2.4.4. EML 2023

Contexte

Pour $x \in]0, +\infty[$ on pose : $f(x) = \frac{e^{-x}}{x}$.

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 .

2.4.5. EML 2019

Contexte

On considère la fonction f définie sur $]0, +\infty[$ par : $\forall t \in]0, +\infty[, f(t) = t + \frac{1}{t}$.

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

```

2.4.6. EML 2018**Contexte**

On pose : $u_0 = 4$ et $\forall n \in \mathbb{N}, u_{n+1} = \ln(u_n) + 2$.

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 .

2.4.7. ECRICOME 2018**Contexte**

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

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. ECRICOME 2026

Contexte

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

$$\left| \ln(2) - \sum_{k=1}^n \frac{1}{k2^k} \right| \leq \frac{1}{2^n}.$$

Écrire une fonction en langage **Python**, nommée **Val**, prenant en argument **eps** désignant le réel ε strictement positif, qui renvoie une valeur approchée de $\ln(2)$ à ε près.

2.5.2. EDHEC 2023

Contexte

On considère la fonction f définie par :

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

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 $4^n \geq 2000/3$.

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

2.5.3. EDHEC 2022

Contexte

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

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

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(----)

```

2.5.4. EML 2019

Contexte

On considère la fonction f définie sur $]0, +\infty[$ par : $\forall t \in]0, +\infty[, f(t) = t + \frac{1}{t}$.

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 :

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

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

2.5.5. EML 2018

Contexte

On pose : $u_0 = 4$ et $\forall n \in \mathbb{N}, u_{n+1} = \ln(u_n) + 2$.

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 : $\forall n \in \mathbb{N}, 0 \leq u_n - b \leq \frac{1}{2^{n-1}}$.

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)

```

2.5.6. ECRICOME 2018**Contexte**

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

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 :

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

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. EML 2026

Contexte

On note $(P_n)_{n \geq 0}$ la suite de polynômes de Tchebychev définie par ses deux premiers termes

$$\forall x \in \mathbb{R}, \quad P_0(x) = 0 \quad \text{et} \quad P_1(x) = 1$$

et la relation de récurrence, valable pour tout entier $n \geq 1$,

$$\forall x \in \mathbb{R}, \quad P_{n+1}(x) = xP_n(x) - P_{n-1}(x)$$

Compléter la fonction **Python** « P(n,x) » ci-dessous qui prend en argument un entier naturel n et un réel x , et qui renvoie le nombre $P_n(x)$. Vous pouvez utiliser autant de lignes que vous le souhaitez dans la boucle « for ».

```

1  def P(n,x):
2      P0 = 0
3      P1 = 1
4      for k in range(1,n+1):
5          .....
6          .....
7          .....
8      return(P0)
```

2.8.2. 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 :

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

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 :

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

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

$$\left\{ k \in \mathbb{N} \mid x \leq \sum_{i=0}^k \frac{p^i}{i!} e^{-p} \right\} \text{ lorsque } x \in [0, 1[\text{ et } p \in]0, 1[.$$

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

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

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 $u_n \geq 10^6$.

```

1 import numpy as np
2 u = 1
3 n = 0
4 while ... :
5     u = ...
6     n = ...
7 print(...)
```

2.9.3. EML 2023

Contexte

Pour $x \in]0, +\infty[$ on pose : $f(x) = \frac{e^{-x}}{x}$.

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 $u_n > a$.

```

1 def fonc_1(a):
2     from numpy import exp
3     u=1
4     n=0
5     while ..... :
6         u = exp(-u)/u
7         n=....
8     return n
```

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

```

1 def fonc_2(a):
2     from numpy import exp
3     u=1
4     n=0
5     while u>a :
6         u = exp(-u)/u
7         n=n+1
8     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 $p < \frac{1}{2}$:

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

où $q = 1 - p$.

1. Après avoir établi la formule $\binom{n}{k} = \frac{n}{k} \binom{n-1}{k-1}$ lorsque $k \in \llbracket 1, n \rrbracket$, 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 $\mathbb{P}(G_n) \leq \varepsilon$ pour $p < \frac{1}{2}$ et $\varepsilon > 0$ saisis au clavier par l'utilisateur.

2.9.5. EML 2017

Contexte

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

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

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 $u_N \geq A$.

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

2.10.1. EDHEC 2026

Contexte

On s'intéresse aux fonctions f et g , définies et dérivables sur $\mathbb{R}$, telles que $f(0) = 1$, $g(0) = 1$, et qui sont solutions du système différentiel :

$$(S) : \forall x \in \mathbb{R}, \begin{cases} f'(x) = f(x) - 2g(x) \\ g'(x) = 2f(x) + 5g(x) \end{cases}$$

On admet qu'il existe une unique solution (f, g) à ce problème, donnée par :

$$\forall x \in \mathbb{R}, \begin{cases} f(x) = (1 - 4x)e^{3x} \\ g(x) = (1 + 4x)e^{3x} \end{cases}$$

Écrire des fonctions **Python** d'en-têtes `def f(x)` et `def g(x)` renvoyant respectivement $f(x)$ et $g(x)$.

2.10.2. ECRICOME 2024

Contexte

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

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

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 $\mathbb{P}([X \geq x])$.

On admet que, pour tout réel x , $I_a(x) = \sqrt{\pi}e^{2ax+a^2}\mathbb{P}([X \geq x])$.

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

2.10.3. 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.

2.10.4. 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.

2.10.5. 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. EML 2026

Contexte

On note $\mathcal{D}$ le domaine du plan :

$$\mathcal{D} = \{(x, y) \in \mathbb{R}^2; 1 + xy > 0\};$$

et on considère la fonction de deux variables réelles f définie sur l'ouvert $\mathcal{D}$ de $\mathbb{R}^2$ par :

$$\forall (x, y) \in \mathcal{D}, \quad f(x, y) = x + \ln(1 + xy).$$

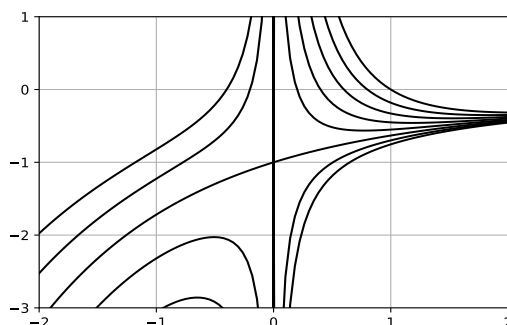
Soit $k \in \mathbb{R}$, on appelle *ligne de niveau k* de la fonction f le sous-ensemble du plan

$$\{(x, y) \in \mathcal{D}; f(x, y) = k\}.$$

Soit $k \in \mathbb{R}$ et soit $(x, y) \in \mathcal{D}$ avec $x \neq 0$. On admet que :

$$f(x, y) = k \iff y = \frac{e^{k-x} - 1}{x}.$$

Le graphique ci-dessous, obtenu à l'aide d'un programme **Python**, représente certaines lignes de niveau de la fonction f .



Recopier et compléter la ligne 5 du programme ci-dessous afin qu'il produise le graphique ci-dessus. À la lecture du programme, préciser pour quelles valeurs de k la ligne de niveau k apparaît sur le graphique.

```

1  K = [-0.4 + i/5 for i in range(8)]
2  X = np.linspace(-2,2,100)
3
4  for k in K:
5      M = [..... for x in X]
6      plt.plot(X,M,color = 'black')
7  plt.xlim(-2,2)
8  plt.ylim(-3,1)
9  plt.show()
```

2.11.2. EDHEC 2026

Contexte

On s'intéresse aux fonctions f et g , définies et dérivables sur $\mathbb{R}$, telles que $f(0) = 1$, $g(0) = 1$, et qui sont solutions du système différentiel :

$$(S) : \forall x \in \mathbb{R}, \begin{cases} f'(x) = f(x) - 2g(x) \\ g'(x) = 2f(x) + 5g(x) \end{cases}$$

On admet qu'il existe une unique solution (f, g) à ce problème.

On dispose de fonctions **Python** d'en-têtes `def f(x)` et `def g(x)` renvoyant respectivement $f(x)$ et $g(x)$.

Écrire un script **Python** utilisant `np.arange` ou `np.linspace` et permettant le tracé des courbes de f et g sur $\left[-\frac{1}{2}, \frac{1}{2}\right]$.

2.11.3. 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.

2.11.4. ESSEC II 2023

Contexte

Soit $n \in \mathbb{N}^*$, $n \geq 2$. 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 :

(H₁) Pour tout $t \geq 0$, $X_t(\Omega) = \{1, \dots, n\}$.

(H₂) Pour tout $r \in \mathbb{N}^*$ et $t_1 < t_2 < \dots < t_r$ des réels positifs, $i_1, \dots, i_{r+1}$ des éléments de $\{1, \dots, n\}$ et s un réel positif, si $\mathbb{P}([X_{t_1} = i_1] \cap \dots \cap [X_{t_r} = i_r]) \neq 0$,

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

(H₃) 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 $f_i(t) \neq 0$.

(H₄) Pour tout $(i, j) \in \{1, \dots, n\}^2$, $i \neq j$ et $h \geq 0$, 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 $h \in \mathbb{R}^+$,

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

(H₅) Pour tout $i \in \{1, \dots, n\}$ et $h \geq 0$, 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 $h \in \mathbb{R}^+$,

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

On note :

- L_t la matrice ligne d'ordre n , $(\mathbb{P}([X_t = 1]) \dots \mathbb{P}([X_t = n])) = (f_1(t) \dots f_n(t))$
- 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 $(i, j) \in \{1, \dots, n\}^2$, $s \geq 0$ et $t \in S_i$. D'après les hypothèses (H₄) et (H₅), $m_{i,j}(s)$ est indépendant de t .

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

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 $M(t) = (I_n + \frac{t}{1000}G)^{1000}$.

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 $(I_n + \frac{t}{1000}G)^{1000}$.

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 L_0 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

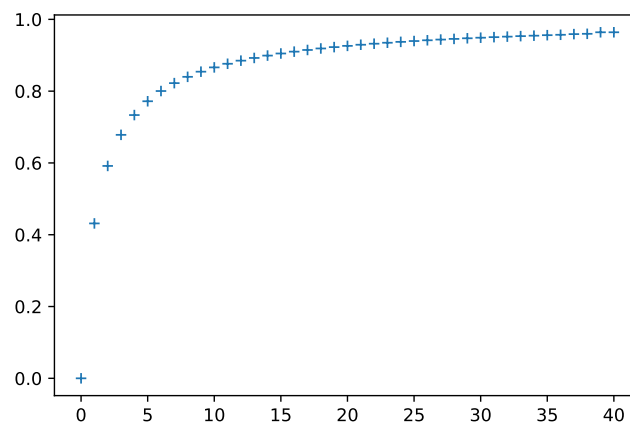
Contexte

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

```
1 x=np.arange(0,41)
2 u=[] # liste vide
3 for n in range(41):
4     u.append(3*n*suite(n))
5 plt.plot(x,u,'+')
6 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 $u_n \underset{n \rightarrow +\infty}{\sim} 3n.$ ❷ $\lim_{n \rightarrow +\infty} u_n = 1.$ ❸ $u_n \underset{n \rightarrow +\infty}{\sim} \frac{1}{3n}.$ ❹ $u_n \underset{n \rightarrow +\infty}{\sim} \frac{1}{n}.$

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 $n \geq 1$ en entrée.

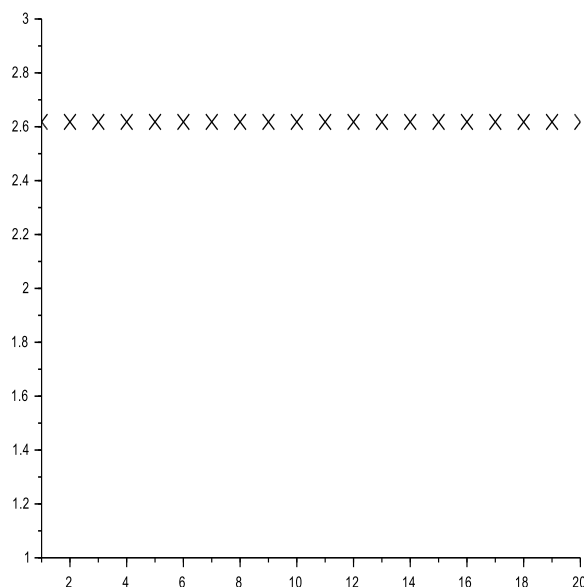
À la suite de la fonction `approxv`, on écrit le code suivant :

```

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 $\mathbb{R}$.

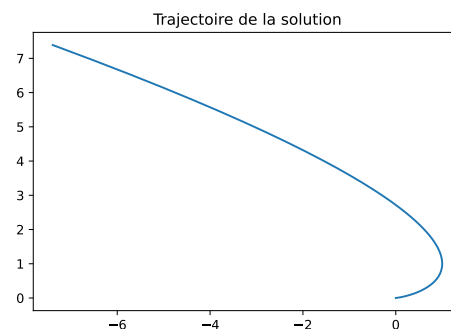
On admet qu'il existe une unique solution (x, y) de (S) telle que $x(0) = 1$ et $y(0) = 1$, donnée par :

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

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 $n \geq 1$ un entier, on considère un graphe non orienté G donné par sa matrice d'adjacence $A \in \mathcal{M}_n(\mathbb{R})$. On note $\mathcal{S} = \{s_0, \dots, s_{n-1}\}$ 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 $c : \mathcal{S} \rightarrow \mathbb{N}$ 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 $i \in \llbracket 0, n-1 \rrbracket$, 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 : $c(s_0) = 0, c(s_1) = 1, c(s_2) = 0, c(s_3) = 1, c(s_4) = 2$.

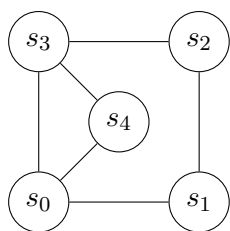


Figure 1 : Un graphe d'ordre cinq

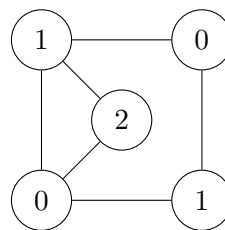


Figure 2 : Le graphe colorié avec trois « couleurs » (0, 1 et 2)

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 $i \in \llbracket 0, n-1 \rrbracket$, 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 $A \in \mathcal{M}_n(\mathbb{R})$ d'un graphe G , et renvoyant une coloration de G sous la forme d'une liste d'entiers $C = [C_0, \dots, C_{n-1}]$, où C_i désigne la « couleur » du sommet s_i pour tout $i \in \llbracket 0, n-1 \rrbracket$.

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 $[0, 1, \dots, n-1]$.
 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.
- Donner la liste obtenue en exécutant la commande « `coloration(A)` ».
 - Le graphe G admet-il une coloration à trois couleurs ? Si oui, exhiber une telle coloration.

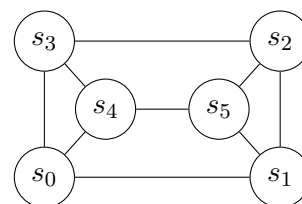


Figure 3 : Le graphe G

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 $s_0, s_1, \dots, s_{p-1}$ 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 $\llbracket 0, p-1 \rrbracket$, 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 $s_0, s_1, \dots, s_{p-1}$ 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 $\llbracket 0, p-1 \rrbracket$, 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 :

- $S = \llbracket 0, n - 1 \rrbracket$, 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

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

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 $0, 1, \dots, n - 1$, modifie L afin qu'elle devienne la liste des listes d'adjacence du graphe G' , dont les sommets sont $0, 1, \dots, n - 2$, 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 $m \geq 1$, $n \geq p \geq 2$.

On suppose que $A = (a_{i,j})_{1 \leq i,j \leq n}$ est la matrice d'adjacence d'un graphe $G(A)$, non orienté sans boucle, à n sommets $1, \dots, n$, 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 $\theta e^{-\theta a}$.

On dispose pour tout $\theta \in \Theta$, d'une suite de variables aléatoires $(X_i)_{i \geq 1}$ indépendantes de même loi que X .

On choisit une suite $(h_n)_{n \geq 1}$ de réels strictement positifs tels que :

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

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

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

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

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 $(X_1(\omega), \dots, X_n(\omega))$ de la loi de X est représentée en **Python** par `échantillon` et, pour tout $n \in \mathbb{N}^*$, $h_n = \frac{1}{\sqrt{n}}$.

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

```

1  a = float(input('a='))
2  n = échantillon.count()
3  h = 1 / np.sqrt(n)
4  C = 0
5  for i in range(n):
6      if ... and ...:
7          ... += 1
8  print(C / ...)
```

4.2. Simulation de variables aléatoires discrètes suivant une loi usuelle

4.2.1. ESSEC II 2026

Ecrire une fonction **Python** `Bernoulli(p)` qui renvoie la valeur d'une simulation d'une variable aléatoire suivant la loi de Bernoulli de paramètre p .

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

4.3.1. 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 $\llbracket 1, n+1 \rrbracket$, 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 $\llbracket 1, n + 1 \rrbracket$, 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 $\llbracket a, b \rrbracket$.

`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 $\llbracket 2, n + 1 \rrbracket$, on code les $j - 1$ boules noires de l'urne numérotée j par les entiers de $\llbracket 1, j - 1 \rrbracket$. 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 $\llbracket 1, n \rrbracket$, la loi de X_n , conditionnellement à l'événement U_k , est la loi géométrique de paramètre $\frac{n - k + 1}{n}$.

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)

```

4.3.2. 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 $i \in \llbracket 1, N \rrbracket$, 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 $i \in \llbracket 1, N \rrbracket$. 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(...)

```

4.3.3. 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 $\llbracket a, b - 1 \rrbracket$.

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 .

```

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

4.3.4. 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, ..., 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 $\llbracket 1; n - 1 \rrbracket$, 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 $(\Omega, \mathcal{A}, \mathbb{P})$ 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 .

```

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

```

4.3.5. 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 :

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

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 $X_1, \dots, X_k$, toutes indépendantes et de même loi que la variable aléatoire X , et reverse au joueur $(X_1 + \dots + X_k)$ 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 .

```

1 def simulX(p):
2     Y = .....
3     while .....:
4         Y = Y + 1
5     X = Y - 1
6     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`.

```

1  def simulZ(n, p):
2      Z = 1
3      for i in range(n):
4          s = 0
5          for j in range(Z):
6              .....
7          Z = .....
8      return Z

```

4.3.6. 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 $(\Omega, \mathcal{A}, \mathbb{P})$. 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

```

4.3.7. 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.

```

1  p = float(input('entrez une valeur pour p :'))
2  c = 1
3  X = rd.geometric(p)
4  Y = rd.geometric(p)
5  while X == Y :
6      X = _____
7      Y = _____
8      c = _____
9  if X < Y :
10     _____
11 else :
12     _____
13 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.

```

11 if _____ :
12     print('A gagne le deuxième jeu')
13 else :
14     _____
```

4.3.8. 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 .

```

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

4.3.9. ESSEC II 2020

Contexte

Soit m un entier fixé tel que $1 \leq m \leq n$. 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 $A \in \mathcal{P}_m$: $\mathbb{P}(R = A) = \frac{1}{\binom{n}{m}}$.

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 : $\mathbb{P}(R = A) = \frac{m!(n-m)!}{n!}$.

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 $X = \lfloor nU \rfloor$ 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 $m \leq n$, 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

```

4.3.10. 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 $\llbracket a, b \rrbracket$.

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)

```

4.3.11. 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.

4.3.12. 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 .

4.3.13. 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 $\llbracket 1, n \rrbracket$. Compléter la fonction suivante pour qu'elle simule la v.a.r. T_n :

```

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

```

4.3.14. 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.

```
1  def EML(n):
2      b = 1 # b désigne le nombre de boules bleues présentes dans l'urne
3      r = 2 # r désigne le nombre de boules rouges présentes dans l'urne
4      s = 0 # s désigne le nombre de boules rouges obtenues lors des n tirages
5      for k in range(n):
6          x = rd.random()
7          if _____:
8              _____
9              _____
10         else:
11             _____
12     return s
```

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

4.4.1. 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 :

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

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

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

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      ...

```

4.4.2. 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 $n \geq 1$ en entrée, et renvoie une matrice ligne contenant n réalisations de la variable aléatoire D .

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

4.5.1. ESSEC II 2026

Contexte

L'apprentissage par renforcement est un sujet largement utilisé dans le domaine de l'intelligence artificielle. Cela consiste schématiquement à apprendre, à partir d'expériences, quelles actions sont à réaliser pour optimiser une récompense quantitative au cours du temps.

On modélise le problème de la manière suivante :

n et r sont des entiers naturels plus grands que 2 et $r < n$.

Les actions sont identifiées aux entiers de 1 à r et n actions sont réalisées l'une après l'autre et numérotées de 1 à n .

$p_1, \dots, p_r$ sont des réels appartenant à $]0, 1[$ non tous égaux et on note p^* leur maximum.

On définit les variables aléatoires suivantes :

- **Le choix des actions :** $(I_j)_{j \in \llbracket 1, n \rrbracket}$ à valeurs dans $\llbracket 1, r \rrbracket$, I_j est égale à la j -ème action réalisée.
- **Les récompenses aléatoires proposées :** $(X_{i,j})_{i \in \llbracket 1, r \rrbracket, j \in \llbracket 1, n \rrbracket}$ sont des variables de Bernoulli indépendantes telles que pour tout $i \in \llbracket 1, r \rrbracket, j \in \llbracket 1, n \rrbracket$, $X_{i,j}$ est de paramètre p_i .
 $X_{i,j}$ est la récompense aléatoire pour l'action i lorsque celle-ci est effectuée pour la j -ième fois.
 On suppose que les $(X_{i,j})_{i \in \llbracket 1, r \rrbracket, j \in \llbracket 1, n \rrbracket}$ sont indépendantes des $(I_j)_{j \in \llbracket 1, n \rrbracket}$.
- Y_j est la variable aléatoire égale à la récompense pour la j -ème action réalisée, $j \in \llbracket 1, n \rrbracket$.
- Par exemple, si pour un certain $\omega \in \Omega$, les quatre premières actions réalisées sont 2, 4, 3 et 2, alors on a :
 $I_1(\omega) = 2, I_2(\omega) = 4, I_3(\omega) = 3, I_4(\omega) = 2, Y_1(\omega) = X_{2,1}(\omega), Y_2(\omega) = X_{4,1}(\omega), Y_3(\omega) = X_{3,1}(\omega), Y_4(\omega) = X_{2,2}(\omega).$
- Pour tous $i \in \llbracket 1, r \rrbracket, j \in \llbracket 1, n \rrbracket$ et $\omega \in \Omega$, $N_{i,j}(\omega)$ est égal au nombre d'indices $k \in \llbracket 1, j \rrbracket$ tels que $I_k(\omega) = i$.
- Pour $i \in \llbracket 1, r \rrbracket, j \in \llbracket 1, n \rrbracket$ et $\omega \in \Omega$, $\bar{X}_{i,j}(\omega) = \frac{1}{j} \sum_{k=1}^j X_{i,k}(\omega)$ et

$$\hat{X}_{i,j}(\omega) = \begin{cases} \frac{1}{N_{i,j}(\omega)} \sum_{k=1}^{N_{i,j}(\omega)} X_{i,k}(\omega) & \text{si } N_{i,j}(\omega) \neq 0 \\ 0 & \text{sinon.} \end{cases}$$

L'objectif est de maximiser, en moyenne, la récompense totale en définissant les variables I_j pour ce faire. Ces définitions constituent la définition d'une stratégie.

Contexte

- **Stratégie « ETC »** - On suppose que m est un entier plus grand que 2 tel que $rm < n$.

On définit une nouvelle variable aléatoire Z_m par, pour tout $\omega \in \Omega$:

$Z_m(\omega)$ est égal au plus petit indice $k \in \llbracket 1, r \rrbracket$ pour lequel on a $\hat{X}_{k,mr}(\omega) = \max_{1 \leq i \leq r} \hat{X}_{i,mr}(\omega)$.

La stratégie ETC consiste à définir les I_j comme suit pour tout $j \in \llbracket 1, n \rrbracket$ et pour tout $\omega \in \Omega$:

- si $j \in \llbracket 1, m \rrbracket$ alors $I_j(\omega) = 1$, si $j \in \llbracket m+1, 2m \rrbracket$ alors $I_j(\omega) = 2, \dots$, si $j \in \llbracket (k-1)m+1, km \rrbracket$ alors $I_j(\omega) = k, \dots$, si $j \in \llbracket (r-1)m+1, rm \rrbracket$ alors $I_j(\omega) = r$,
- si $mr < j \leq n$, $I_j(\omega) = Z_m(\omega)$.

Écrire une fonction **Python** `Z(m,r,Rec)` qui renvoie la valeur de Z_m obtenue lorsque la variable `Rec` est le vecteur des récompenses obtenues pour les rm premières actions suivant la stratégie ETC.

Contexte

► La stratégie UCB consiste à définir les I_j comme suit pour tout $j \in \llbracket 1, n \rrbracket$ et pour tout $\omega \in \Omega$:

- si $j \in \llbracket 1, r \rrbracket$ alors $I_j(\omega) = j$;

- si $j \in \llbracket r, n-1 \rrbracket$, posons pour tout $i \in \llbracket 1, r \rrbracket$, $U_{i,j}(\omega) = \hat{X}_{i,j}(\omega) + \sqrt{\frac{\ln(n)}{N_{i,j}(\omega)}}$.

$I_{j+1}(\omega)$ est égal au plus petit indice $i \in \llbracket 1, r \rrbracket$ pour lequel on a $U_{i,j}(\omega) = \max_{1 \leq k \leq r} U_{k,j}(\omega)$.

- On note aussi pour $i \in \llbracket 1, r \rrbracket$ et $j \in \llbracket 1, n-1 \rrbracket$, $V_{i,j}(\omega) = \bar{X}_{i,j}(\omega) + \sqrt{\frac{\ln(n)}{j}}$.

On suppose dans la suite que l'on applique la stratégie UCB.

On dispose d'une fonction **Python** `Bernoulli(p)` qui renvoie la valeur d'une simulation d'une variable aléatoire suivant la loi de Bernoulli de paramètre p .

1. En supposant que `hatX` contient $\hat{X}_{i,j}$, `N` contient $N_{i,j}$ et le vecteur `P` contient les valeurs de $p_1, \dots, p_r$, écrire une fonction **Python** `maj(hatX, N, i, P)` qui simule la récompense de l'action i , celle-ci étant la $(j+1)$ -ème action réalisée et renvoie la valeur de $\hat{X}_{i,j+1}$ dans ces conditions.
2. Compléter l'écriture de la fonction **Python** `Actions(P, n)` ci-dessous pour qu'elle renvoie la simulation d'un vecteur des n actions réalisées par l'algorithme UCB si le vecteur `P` contient les valeurs de $p_1, \dots, p_r$.

```

1  def Actions(P,n):
2      r=np.shape(P)[0]; I=np.zeros(n)
3      I[0:r]=[k+1 for k in range(r)];N=np.ones(r)
4      hatX=np.array([Bernoulli(P[i]) for i in range(r)])
5      for j in range(...):
6          Max=hatX[0]+np.sqrt(np.log(n)/N[0])
7          I[j]=1
8          for k in range(1,r):
9              val=hatX[k]+np.sqrt(np.log(n)/N[k])
10             if val>Max:
11                 Max=...
12                 I[j]=...
13             hatX[I[j]-1]=maj(hatX[I[j]-1],N[I[j]-1],I[j],P)
14             N[I[j]-1]=...
15     return I

```

4.5.2. ESSEC II 2025

Contexte

On définit la fonction f sur $[0, 1]$ par $f(t) = (1 + t)e^{-t} - t$.

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 :

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

On dispose d'une fonction **Python**, `minimum(x,p)` qui renvoie le minimum de l'ensemble

$$\left\{ k \in \mathbb{N} \mid x \leq \sum_{i=0}^k \frac{p^i}{i!} e^{-p} \right\} \text{ lorsque } x \in [0, 1[\text{ et } p \in]0, 1[.$$

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

4.6. Simulation de sommes de variables aléatoires discrètes

4.6.1. 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 .

4.6.2. ESSEC II 2021

Contexte

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

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

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

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

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

```

4.7. Simulation d'un couple de variables aléatoires discrètes liées à une expérience concrète

4.7.1. EML 2026

Contexte

On dispose d'une pièce qui tombe sur pile avec une probabilité $p \in]0, 1[$. Lorsqu'on effectue une succession de $n \geq 2$ lancers avec cette pièce on note, pour tout $k \in \llbracket 1, n \rrbracket$,

- P_k l'événement : « on obtient pile au k -ième lancer »,
- F_k l'événement : « on obtient face au k -ième lancer ».

Par abus de notation, on omettra le symbole d'intersection entre ces événements, ainsi on écrira $P_1 F_2 F_3$ au lieu de $P_1 \cap F_2 \cap F_3$. Enfin, on appelle :

- *double pile* tout événement de la forme $P_{k-1} P_k$ avec $k \in \llbracket 2, n \rrbracket$,
- *pile isolé* tout événement de la forme $P_1 F_2$ ou $F_{k-1} P_k F_{k+1}$ avec $k \in \llbracket 2, n-1 \rrbracket$

On considère la *première expérience aléatoire* suivante : on lance une pièce jusqu'à obtenir un premier pile, puis on relance la pièce une seule fois. On note :

- N la variable aléatoire égale au nombre de lancers effectués,
- S l'événement : « le dernier lancer donne pile ».

Exemple : si la succession de lancers donne $F_1 P_2 F_3$ alors on a $N = 3$ et l'événement $\bar{S}$ est réalisé, si la succession de lancers donne $F_1 F_2 F_3 P_4 P_5$ alors on a $N = 5$ et l'événement S est réalisé.

On s'intéresse maintenant à l'expérience aléatoire consistant à lancer la pièce jusqu'à obtenir un premier double pile. On admet que cela revient à répéter de manière indépendante la première expérience aléatoire jusqu'à la réalisation de l'événement S . On note alors :

- R la variable aléatoire égale au nombre de répétitions de la première expérience aléatoire,
- T la variable aléatoire égale au nombre total de lancers effectués,

et, pour tout entier $i \geq 1$,

- N_i la variable aléatoire égale au nombre de lancers réalisés lors de la i -ième répétition de la première expérience aléatoire si $R \geq i$, et égale à 0 si $R < i$.

On admet que toutes ces variables aléatoires sont bien définies presque sûrement.

Exemple : la série de lancers $F_1 P_2 F_3 F_4 F_5 F_6 P_7 P_8$ se décompose en deux répétitions de la première expérience aléatoire, on a $F_1 P_2 F_3$ puis $F_4 F_5 F_6 P_7 P_8$, ce sont les deux exemples vus précédemment. Dans ce cas, on a $R = 2$, $T = 8$, $N_1 = 3$, $N_2 = 5$ et $N_i = 0$ pour tout $i \geq 3$.

Par définition des variables aléatoires N_i on a $T = \sum_{i=1}^{+\infty} N_i$, c'est-à-dire :

$$\forall \omega \in \Omega, \quad T(\omega) = \sum_{i=1}^{+\infty} N_i(\omega)$$

Notez que cette somme est (presque sûrement) bien définie puisque $N_i(\omega) = 0$ pour tout $i > R(\omega)$.

Rappel Python : la commande « `rd.random()` » renvoie un nombre dans $[0, 1[$ selon la loi uniforme sur $[0, 1[$.

1. Recopier et compléter la fonction « **Exp_1(p)** » ci-dessous qui simule la première expérience aléatoire. Cette fonction prend en argument le paramètre $p \in]0, 1[$ et renvoie un couple d'entiers (N, S) , où N représente le nombre de lancers effectués, et $S \in \{0, 1\}$ prend la valeur 1 si le dernier lancer a donné pile et 0 sinon.

```

1  def Exp_1(p):
2      N = 1
3      while ..... :
4          N = N + 1
5
6      if ..... :
7          S = .....
8      else:
9          S = .....
10     return N+1, S

```

2. Rédiger une fonction « **Exp_2(p)** » qui simule la deuxième expérience aléatoire consistant à répéter la première expérience aléatoire jusqu'à la réalisation de l'événement S . Cette fonction prend en argument le paramètre p et renvoie le couple d'entiers (R, T) où R est le nombre de répétitions de la première expérience aléatoire, et T est le nombre total de lancers effectués.

On vous demande de suivre cet algorithme :

On affecte aux variables R , S et T la valeur 0.
 Tant que $S = 0$.
 | On augmente la variable R de 1
 | On affecte à la variable E le résultat de **Exp_1(p)**
 | On affecte à la variable S la deuxième composante de E
 | On ajoute à la variable T la première composante de E
 On renvoie le couple (R, T) .

4.7.2. 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 $j \in \llbracket 1, k \rrbracket$, 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.

- 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 $[1, 2, 2, 3, 3, 3, 4, 4, 4, 4]$.
- 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
```

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

4.8.1. ECRICOME 2026

Contexte

Soit n un entier naturel supérieur ou égal à 2.

Un particulier souhaite assurer un nouveau bien. Il consulte pour cela un site internet qui diffuse un classement de n compagnies d'assurances, de la mieux classée par les utilisateurs (compagnie n° 1) à la moins bien classée (compagnie n° n). Ce classement ne varie pas d'année en année.

La première année, le particulier choisit une compagnie d'assurances au hasard, avec équiprobabilité, parmi les n compagnies proposées.

On note X la variable aléatoire égale au numéro, dans le classement, de la compagnie choisie.

L'année suivante, le particulier choisit au hasard de manière équiprobable une compagnie parmi toutes celles qui sont au moins aussi bien classées que celle qu'il avait choisie l'année précédente (il peut éventuellement être amené à conserver la même compagnie). On note alors Y la variable aléatoire égale au numéro, dans le classement, de la compagnie choisie pour cette seconde année.

Écrire une fonction en langage **Python** nommée `simulXY`, prenant en argument d'entrée l'entier n , et renvoyant une réalisation du couple de variables aléatoires (X, Y) .

4.8.2. 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 :

- $X \hookrightarrow \mathcal{U}([1, n])$
- pour tout $k \in [1, n]$, 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 $[a, b]$,
- `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 = -----

```

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

4.9.1. HEC 2018

Contexte

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

Pour tout réel z , soit $((z)^{[m]})_{m \in \mathbb{N}}$ 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 $0 \leq k \leq \ell$:

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

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

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

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

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

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

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 .

4.10. Simulation d'une chaîne de Markov

4.10.1. 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 $P = (p_{i,j})_{1 \leq i,j \leq r}$ 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 $L[i-1]$.

4.10.2. ESSEC II 2023

Contexte

Soit $n \in \mathbb{N}^*$, $n \geq 2$. 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 :

(H₁) Pour tout $t \geq 0$, $X_t(\Omega) = \{1, \dots, n\}$.

(H₂) Pour tout $r \in \mathbb{N}^*$ et $t_1 < t_2 < \dots < t_r$ des réels positifs, $i_1, \dots, i_{r+1}$ des éléments de $\{1, \dots, n\}$ et s un réel positif, si $\mathbb{P}([X_{t_1} = i_1] \cap \dots \cap [X_{t_r} = i_r]) \neq 0$,

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

(H₃) 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 $f_i(t) \neq 0$.

(H₄) Pour tout $(i, j) \in \{1, \dots, n\}^2$, $i \neq j$ et $h \geq 0$, 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 $h \in \mathbb{R}^+$,

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

(H₅) Pour tout $i \in \{1, \dots, n\}$ et $h \geq 0$, 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 $h \in \mathbb{R}^+$,

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

On note :

- L_t la matrice ligne d'ordre n , $(\mathbb{P}([X_t = 1]) \dots \mathbb{P}([X_t = n])) = (f_1(t) \dots f_n(t))$
- 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 $(i, j) \in \{1, \dots, n\}^2$, $s \geq 0$ et $t \in S_i$. D'après les hypothèses (H₄) et (H₅), $m_{i,j}(s)$ est indépendant de t .

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

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 $M(t) = (I_n + \frac{t}{1000}G)^{1000}$.

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 $(I_n + \frac{t}{1000}G)^{1000}$.

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 $X_0, X_t, \dots, X_{kt}$, 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 :

```
1  def simulX(t,k,L0,G):  
2      listeDesT=[] ; listeDesX=[]  
3      Mt=transition(t,G) ; Lt = L0  
4      for i in range(k+1):  
5          listeDesT.append(i*t)  
6          p=rd.random()  
7          s=...  
8          j=0  
9          while p>...:  
10             j+=1  
11             s+=Lt[j]  
12             Lt=...  
13             listeDesX.append(j+1)  
14     plt.plot(listeDesT,listeDesX) ; plt.show()
```

4.10.3. 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.

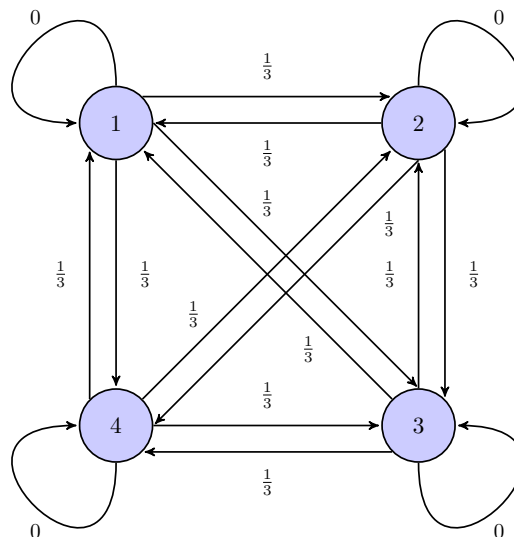
Un mobile se déplace aléatoirement sur les sommets de ce carré selon le protocole suivant :

- 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 $n \in \mathbb{N}$, $\mathbb{P}([X_n = 1]) = \frac{1}{4} + \frac{3}{4} \left(-\frac{1}{3}\right)^n$

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 $\llbracket a, b-1 \rrbracket$.

```

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 ?

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

4.11.1. ECRICOME 2025

Contexte

Soit i un entier naturel non nul.

On considère une variable aléatoire X_i admettant f_i pour densité, où

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

Soit U une variable aléatoire suivant la loi uniforme sur $]0, 1[$. On pose : $V_i = \frac{1}{U^{1/i}}$.

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 .

4.11.2. ECRICOME 2020

Contexte

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

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

Soit U une variable aléatoire suivant la loi uniforme sur $]0, 1[$. On pose : $Y = \frac{a}{U^{1/3}}$.

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.

4.11.3. EML 2020

Contexte

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

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

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 $bU^{-\frac{1}{a}}$ 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 .

4.11.4. 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 : $f_{a,b}(x) = \begin{cases} (a + bx) \exp\left(-ax - \frac{b}{2}x^2\right) & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$.

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 : $X = \frac{-a + \sqrt{a^2 + 2bY}}{b}$. 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.

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

4.12.1. EDHEC 2024

Contexte

Soit f la fonction qui à tout réel x associe

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

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 $\frac{1}{2}$.

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

4.12.2. 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 .

4.12.3. ECRICOME 2021

Contexte

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

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

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

```

4.12.4. 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 : $f : x \mapsto \begin{cases} \frac{1}{\theta x^{1+\frac{1}{\theta}}} & \text{si } x \geq 1 \\ 0 & \text{si } x < 1 \end{cases}$

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 .

4.12.5. EDHEC 2018

Contexte

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

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

On considère la variable aléatoire Y définie par : $Y = \frac{X^2}{2a}$. 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 .

4.12.6. EDHEC 2017

Contexte

On désigne par n un entier naturel non nul et par $X_1, \dots, X_n$ 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 $Y_n = \max(X_1, X_2, \dots, X_n)$, c'est à dire que pour tout ω de Ω , on a : $Y_n(\omega) = \max(X_1(\omega), X_2(\omega), \dots, X_n(\omega))$. 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 _____
```

4.12.7. 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 $D = \llbracket 0, d \rrbracket$ 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 $a = (a_1, \dots, a_n)$, 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 $a = (a_1, \dots, a_n)$ 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 $\beta = \frac{d}{\varepsilon}$.

On pose :

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

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 $\frac{|Z_a - q(a)|}{q(a)}$ (on considèrera que $q(a)$ est toujours non nul).

4.13. Simulation de sommes de variables aléatoires à densité

4.13.1. ECRICOME 2020

Contexte

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

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

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 $X_1, \dots, X_n$ n variables aléatoires indépendantes et suivant toutes la même loi que X . On pose : $V_n = \frac{2}{3n} \sum_{k=1}^n X_k$.

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
```

4.13.2. EDHEC 2020

Contexte

On considère une variable aléatoire X suivant la loi normale $\mathcal{N}(0, \sigma^2)$, 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 $(Y_1, Y_2, \dots, Y_n)$ composé de variables aléatoires, mutuellement indépendantes, et ayant toutes la même loi que Y .

On note S_n la variable aléatoire définie par : $S_n = \frac{1}{n} \sum_{k=1}^n Y_k$. On note enfin $T_n = \sqrt{\frac{\pi}{2}} S_n$.

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.

```

1  n = int(input('entrez la valeur de n :'))
2  sigma = float(input('entrez la valeur de sigma :'))
3  X = ----- # simulations de X1, ..., Xn
4  Y = ----- # simulations de Y1, ..., Yn
5  S = -----
6  T = -----

```

4.13.3. 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 $(P_k)_{k \geq 1}$ 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 $(Q_k)_{k \geq 1}$; 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 $(\Omega, \mathcal{A}, \mathbb{P})$ sur lequel sont définies les suites de variables aléatoires $(X_k)_{k \geq 1}$ et $(Y_k)_{k \geq 1}$, 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 $X_1, \dots, X_k, Y_1, \dots, Y_k$ 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 $n \geq 0$, 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 $U_1, U_2, \dots, U_n$:

```
1  p = float(input('p = '))
2  n = int(input('n = '))
3  q = 1 - p
4  U = np.zeros(n)
5  sommeX = rd.exponential(1/p)
6  sommeY = rd.exponential(1/q)
7  mini = min(sommeX, sommeY)
8  for k in range(n):
9      if sommeX == ...:
10         U[k] = ...
11         sommeX = sommeX + rd.exponential(1/p)
12     else:
13         sommeY = ...
14         mini = min(sommeX, sommeY)
15     ...
```

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

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

4.14.1. 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 :

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

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 socio-professionnelle 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 .

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

4.15.1. 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 $\omega \in \Omega$:

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

et $K_{n,s}(\omega)$ est égal au nombre d'indices $k \in \llbracket 1, n \rrbracket$ 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 $k \in \llbracket 1, n \rrbracket$, $p_k = \mathbb{P}(X_k > s)$ 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 $(Z_n, Y_{n,s})$.

4.16. Simulation d'une matrice aléatoire dont les coefficients suivent des lois à densité

4.16.1. ESSEC I 2026

Contexte

Soit n, m des entiers tels que $n > m \geq 1$, m bien plus petit que n .

Pour tout d entier naturel non nul et $X \in \mathcal{M}_{d,1}(\mathbb{R})$, de coefficients $x_1, \dots, x_d$ on pose $\|X\| = \sqrt{\sum_{k=1}^d x_k^2}$.

On admet que l'on a défini une norme sur $\mathcal{M}_{d,1}(\mathbb{R})$, ce qui sous-entend en particulier que :

- Pour tout $X \in \mathcal{M}_{d,1}(\mathbb{R})$, si $\|X\| = 0$ alors $X = 0$.
- Pour tous $X \in \mathcal{M}_{d,1}(\mathbb{R})$, $Y \in \mathcal{M}_{d,1}(\mathbb{R})$, $\|X\| - \|Y\| \leq \|X + Y\| \leq \|X\| + \|Y\|$.
- Pour tous $X \in \mathcal{M}_{d,1}(\mathbb{R})$, $\lambda \in \mathbb{R}$, $\|\lambda X\| = |\lambda| \|X\|$.

Si $\mathcal{X}$ est un ensemble d'éléments de $\mathcal{M}_{n,1}(\mathbb{R})$, $\varepsilon \in]0, 1[$ et $A \in \mathcal{M}_{m,n}(\mathbb{R})$, on dit que A est une $(\varepsilon, \mathcal{X})$ -isométrie si pour tout $X \in \mathcal{X}$:

$$(1 - \varepsilon)\|X\|^2 \leq \|AX\|^2 \leq (1 + \varepsilon)\|X\|^2$$

On dispose d'une fonction `EstIsom(A, LX, eps)` qui renvoie `True` si A est une $(\varepsilon, \mathcal{X})$ -isométrie et `False` sinon, dès lors que $\mathcal{X}$, un ensemble d'éléments de $\mathcal{M}_{n,1}(\mathbb{R})$, est représenté par la liste finie `LX` de vecteurs `numpy`, la matrice A par le tableau `numpy A` et ε par `eps`.

On considère $(\Omega, \mathcal{A}, \mathbb{P})$ un espace probabilisé sur lequel sont définies les variables aléatoires $G_{i,j}$ pour $(i, j) \in \llbracket 1, m \rrbracket \times \llbracket 1, n \rrbracket$.

Ces variables sont indépendantes et toutes de loi normale $\mathcal{N}(0, 1)$. On définit les matrices aléatoires $M(\omega)$, pour tout $\omega \in \Omega$ par :

$$M(\omega) = \frac{1}{\sqrt{m}} \begin{pmatrix} G_{1,1}(\omega) & \dots & G_{1,n}(\omega) \\ G_{2,1}(\omega) & \dots & G_{2,n}(\omega) \\ \vdots & & \vdots \\ G_{m,1}(\omega) & \dots & G_{m,n}(\omega) \end{pmatrix}$$

Soit $\mathcal{X} = \{X_1, \dots, X_r\}$, un ensemble d'éléments de $\mathcal{M}_{n,1}(\mathbb{R})$ et soit $\varepsilon \in]0, 1[$.

On admet que, si $m > \frac{8 \ln(2r)}{\varepsilon^2}$, alors la probabilité de l'événement « M n'est pas une $(\varepsilon, \mathcal{X})$ -isométrie » est strictement inférieure à 1.

1. Ecrire une expression **Python** qui réalise une simulation d'une telle matrice M si m et n sont donnés et représentés par les variables `m` et `n`.
2. On suppose que $\mathcal{X}$, un ensemble d'éléments de $\mathcal{M}_{n,1}(\mathbb{R})$, est représenté par la liste finie `LX` de vecteurs colonnes `numpy`. Écrire une fonction `Isom(LX, m, eps)` qui renvoie une matrice à m lignes et n colonnes qui est une $(\varepsilon, \mathcal{X})$ -isométrie.

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

4.17.1. 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 $X_1, \dots, X_n$ et N sont mutuellement indépendantes.

On définit la variable aléatoire $T = \max(X_1, \dots, X_N)$, ce qui signifie :

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

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 .

4.17.2. HEC/ESSEC I 2021

Contexte

On considère :

- un espace probabilisé $(\Omega, \mathcal{A}, \mathbb{P})$ 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 géométrique $\mathcal{G}(Y)$.

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

4.18.1. EDHEC 2026

Contexte

On désigne par n un entier naturel supérieur ou égal à 2.

Une variable aléatoire X suit une loi uniforme sur le segment $[0, \theta]$, θ étant un réel élément de $[5, 7]$. On dispose de n variables aléatoires $X_1, X_2, \dots, X_n$, mutuellement indépendantes et de même loi que X .

On pose $Y_n = \max(X_1, X_2, \dots, X_n)$.

On rappelle que `rd.random(n)` renvoie, sous forme de vecteur, une simulation **Python** de n variables aléatoires à densité, indépendantes, et suivant la loi uniforme sur $[0, 1]$.

On admet que, si U suit la loi uniforme sur $[0, 1]$, alors θU suit la loi uniforme sur $[0, \theta]$.

Compléter la fonction **Python** suivante afin qu'elle simule la variable aléatoire Y_n .

```

1 def var_Y(theta,n):
2     X=-----
3     Y=-----
4     return Y

```

4.18.2. 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 :

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

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

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 .

4.18.3. EDHEC 2024

Contexte

Soit f la fonction qui à tout réel x associe

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

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 $M_n = \min(X_1, \dots, X_n)$.

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

```

4.18.4. 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 .
- $X_1, \dots, X_n$ 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 $\omega \in \Omega$:

- $Y_n(\omega) = \max(X_1(\omega), \dots, X_n(\omega))$ est le plus grand des réels $X_1(\omega), \dots, X_n(\omega)$;
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 $X_1(\omega), \dots, X_n(\omega)$, 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]

```

4.19. Tracé d'un diagramme en bâtons

4.19.1. 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 $(p_1, \dots, p_n)$ 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.

4.20. Utilisation de la loi faible des grands nombres

4.20.1. EML 2026

Contexte

On dispose d'une pièce qui tombe sur pile avec une probabilité $p \in]0, 1[$. Lorsqu'on effectue une succession de $n \geq 2$ lancers avec cette pièce on note, pour tout $k \in \llbracket 1, n \rrbracket$,

- P_k l'événement : « on obtient pile au k -ième lancer »,
- F_k l'événement : « on obtient face au k -ième lancer ».

Par abus de notation, on omettra le symbole d'intersection entre ces événements, ainsi on écrira $P_1 F_2 F_3$ au lieu de $P_1 \cap F_2 \cap F_3$. Enfin, on appelle :

- *double pile* tout événement de la forme $P_{k-1} P_k$ avec $k \in \llbracket 2, n \rrbracket$,
- *pile isolé* tout événement de la forme $P_1 F_2$ ou $F_{k-1} P_k F_{k+1}$ avec $k \in \llbracket 2, n-1 \rrbracket$

On considère la *première expérience aléatoire* suivante : on lance une pièce jusqu'à obtenir un premier pile, puis on relance la pièce une seule fois. On note :

- N la variable aléatoire égale au nombre de lancers effectués,
- S l'événement : « le dernier lancer donne pile ».

Exemple : si la succession de lancers donne $F_1 P_2 F_3$ alors on a $N = 3$ et l'événement $\bar{S}$ est réalisé, si la succession de lancers donne $F_1 F_2 F_3 P_4 P_5$ alors on a $N = 5$ et l'événement S est réalisé.

On s'intéresse maintenant à l'expérience aléatoire consistant à lancer la pièce jusqu'à obtenir un premier double pile. On admet que cela revient à répéter de manière indépendante la première expérience aléatoire jusqu'à la réalisation de l'événement S . On note alors :

- R la variable aléatoire égale au nombre de répétitions de la première expérience aléatoire,
- T la variable aléatoire égale au nombre total de lancers effectués,

et, pour tout entier $i \geq 1$,

- N_i la variable aléatoire égale au nombre de lancers réalisés lors de la i -ième répétition de la première expérience aléatoire si $R \geq i$, et égale à 0 si $R < i$.

On admet que toutes ces variables aléatoires sont bien définies presque sûrement.

Exemple : la série de lancers $F_1 P_2 F_3 F_4 F_5 F_6 P_7 P_8$ se décompose en deux répétitions de la première expérience aléatoire, on a $F_1 P_2 F_3$ puis $F_4 F_5 F_6 P_7 P_8$, ce sont les deux exemples vus précédemment. Dans ce cas, on a $R = 2$, $T = 8$, $N_1 = 3$, $N_2 = 5$ et $N_i = 0$ pour tout $i \geq 3$.

Par définition des variables aléatoires N_i on a $T = \sum_{i=1}^{+\infty} N_i$, c'est-à-dire :

$$\forall \omega \in \Omega, \quad T(\omega) = \sum_{i=1}^{+\infty} N_i(\omega)$$

Notez que cette somme est (presque sûrement) bien définie puisque $N_i(\omega) = 0$ pour tout $i > R(\omega)$. On dispose d'une fonction « Exp_2(p) » qui simule la deuxième expérience aléatoire consistant à répéter la première expérience aléatoire jusqu'à la réalisation de l'événement S . Cette fonction prend en argument le paramètre p et renvoie le couple d'entiers (R, T) où R est le nombre de répétitions de la première expérience aléatoire, et T est le nombre total de lancers effectués.

- a) Rédiger une fonction « Freq_T(n,p) » qui prend en argument un entier $n \geq 2$ ainsi que le paramètre p , et qui renvoie la fréquence d'apparition de n parmi 10^4 réalisations de la variable aléatoire T (on pourra simuler T par la deuxième composante de « Exp_2(p) »).
- b) En exécutant la commande « Freq_T(4, 1/2) » l'ordinateur affiche « 0.123 ». Ce résultat vous paraît-il cohérent ? Justifier votre réponse.

4.20.2. EDHEC 2026

Contexte

On désigne par n un entier naturel supérieur ou égal à 2.

Une variable aléatoire X suit une loi uniforme sur le segment $[0, \theta]$, θ étant un réel élément de $[5, 7]$, supposé **inconnu**. On dispose de n variables aléatoires $X_1, X_2, \dots, X_n$, mutuellement indépendantes et de même loi que X .

On pose $Y_n = \max(X_1, X_2, \dots, X_n)$.

On dispose d'une fonction `var_Y(theta,n)` qui simule la variable aléatoire Y_n .

On pose $Z_n = \frac{n+1}{n} Y_n$.

On admet Z_n est un estimateur de θ , que $\mathbb{E}(Z_n) = \theta$ et que Z_n admet une variance.

On considère le code **Python** suivante :

```

1 def mystere(theta):
2     c=0
3     for k in range(1000):
4         y=var_Y(theta,2000)
5         z=2001*y/2000
6         if np.abs(z-theta) <= 0.1:
7             c=c+1
8     return (c/1000)
9 print(mystere(6))

```

Expliquer le fonctionnement de ce code et préciser ce que représente le résultat affiché.

4.20.3. 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 $\omega \in \Omega$:

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

et $K_{n,s}(\omega)$ est égal au nombre d'indices $k \in \llbracket 1, n \rrbracket$ 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 $k \in \llbracket 1, n \rrbracket$, $p_k = \mathbb{P}(X_k > s)$ 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 $p = 0.15$.

4.20.4. 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 $(p_1, \dots, p_n)$ 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

```

4.20.5. ECRICOME 2024

Contexte

On considère une variable aléatoire X de loi normale d'espérance $-a$ et de variance $\frac{1}{2}$.

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 $\mathbb{P}([X \geq x])$.

```

1  import numpy as np
2  import numpy.random as rd
3  def estim_proba(a, x):
4      num = 0
5      for i in range(10000):
6          Z = rd.normal()
7          X = ----- + Z/-----
8          if -----:
9              num = num + 1
10     return -----

```

4.20.6. 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 $i \in \llbracket 1, N \rrbracket$, 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 :

```

1  def ajout(L,x):
2      if (x in L) == False :
3          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 ?

4.20.7. 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 :

- $S = \llbracket 0, n - 1 \rrbracket$, 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

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

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

```

4.20.8. ECRICOME 2023

Contexte

Soit n un entier naturel non nul.

On considère (X, Y) un couple de variables aléatoires discrètes tel que $X(\Omega) = Y(\Omega) = \llbracket 1, n \rrbracket$.

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 ?

4.20.9. 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 :

```

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

```

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

4.20.10. 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).

4.20.11. ECRICOME 2021

Contexte

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

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

On dispose d'une fonction **Python** `simuY(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 :

```

1  n = int(input('Entrer la valeur de n :'))
2  print(np.mean(simuY(n, 1000)))

```

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

Contexte

(Exo 3) On lance indéfiniment une pièce équilibrée.

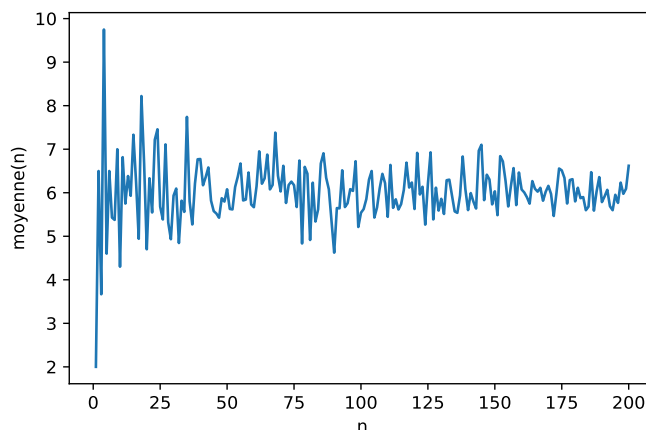
On s'intéresse au rang du lancer auquel on obtient pour la première fois deux « Pile » consécutifs.

On modélise cette expérience aléatoire par un espace probabilisé $(\Omega, \mathcal{A}, \mathbb{P})$. 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** `simuX()` 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 $\llbracket 1, 200 \rrbracket$, et on trace les résultats obtenus dans le graphe suivant.



Que pouvez-vous conjecturer sur la variable aléatoire X ?

4.20.12. 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 $X_1, \dots, X_n$ et N sont mutuellement indépendantes.

On définit la variable aléatoire $T = \max(X_1, \dots, X_N)$, ce qui signifie :

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

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 :

```

1  def mystere():
2      m = np.zeros(3)
3      for k in range(3):
4          s = np.zeros(1000)
5          for j in range(1000):
6              s[j] = simuleT()
7          m[k] = np.mean(s)
8      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 ?

4.20.13. ECRICOME 2020

Contexte

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

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

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 $\mathbb{P}_{[X > 2a]}([X > 6a]) = \frac{1}{27}$ à la question précédente.

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

```

1  a = 10
2  N = 100000
3  s1 = 0
4  s2 = 0
5  X = simulX(a, 1, N)[0]
6  for k in range(N):
7      if _____:
8          s1 = s1 + 1
9          if X[k] > 6*a:
10             _____
11  if s1 > 0:
12      print(_____)
```

4.20.14. EML 2020

Contexte

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

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

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

```

4.20.15. ECRICOME 2019

Contexte

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

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

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 :

$V = \frac{1}{\sqrt{1-U}}$. 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 $n \geq 1$ 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 $\mathbf{c}$ sont-ils une simulation ? Pour n assez grand, quelle sera la valeur affichée ? Justifier votre réponse.

4.20.16. 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 $\mathcal{B}\left(2, \frac{1}{2}\right)$.

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 $\mathbf{X}$ et $\mathbf{S}$ après l'exécution des quatre premières instructions ?
2. Expliquer pourquoi, après l'exécution des six instructions, chacun des coefficients des matrices $\mathbf{Z1}$ et $\mathbf{Z2}$ 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

```

4.20.17. EDHEC 2018

Contexte

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

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)
```

4.20.18. 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 :

```

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

4.20.19. 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 .
- $X_1, \dots, X_n$ 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 $\omega \in \Omega$:

- $Y_n(\omega) = \max(X_1(\omega), \dots, X_n(\omega))$ est le plus grand des réels $X_1(\omega), \dots, X_n(\omega)$;
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 $X_1(\omega), \dots, X_n(\omega)$, 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 : $\varphi_x(t) = \begin{cases} t & \text{si } t \leq x \\ 0 & \text{sinon} \end{cases}$.

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

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 $\mathbb{P}([Y_{n-1} \leq x])$.

On utilisera la fonction `simulX` pour simuler des échantillons de la loi de X , et on rappelle que si $\mathbf{v}$ est un vecteur, `max(v)` est égal au plus grand élément de $\mathbf{v}$.

4.20.20. 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 : $\forall k \in \mathbb{N}^*, \mathbb{P}([Y = k]) = \frac{k-1}{k!}$.

On admet que $(T_n)_{n \geq 1}$ converge en loi vers la variable aléatoire Y (*).

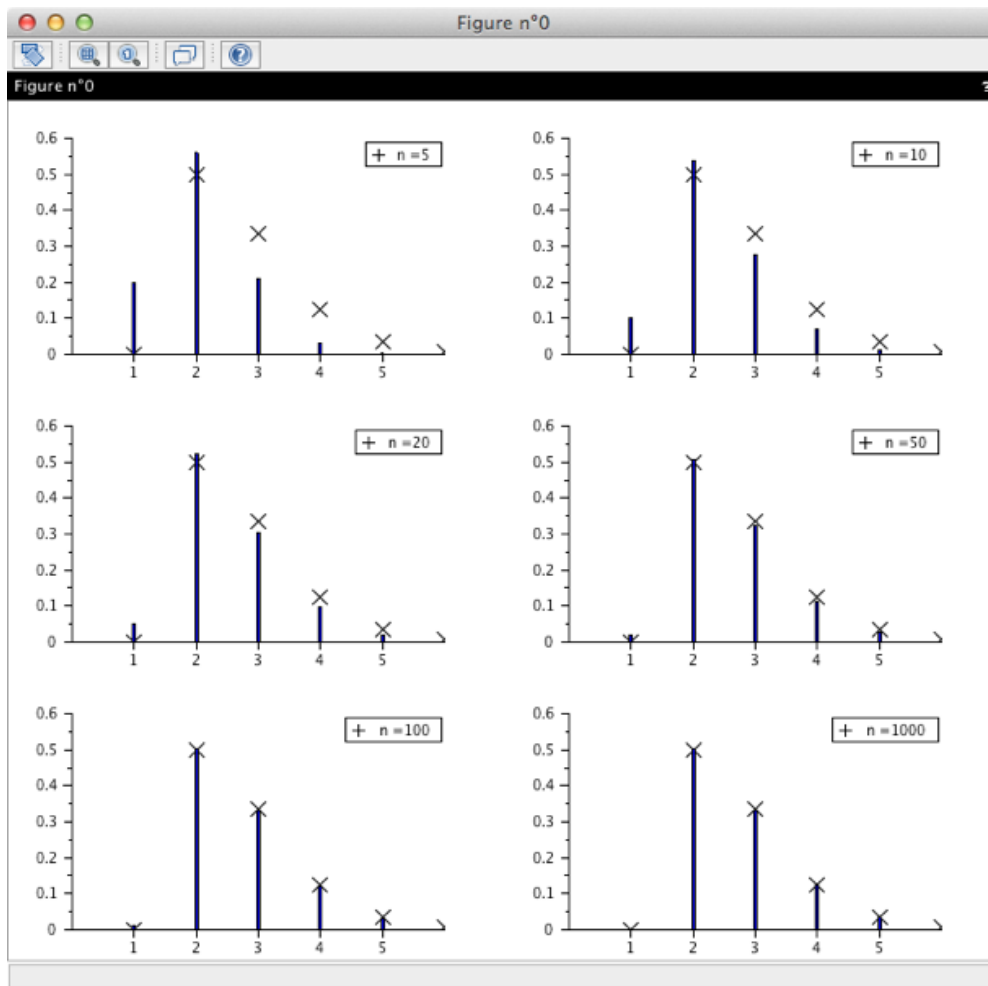
On admet que l'appel `math.factorial(k)` renvoie le nombre $k!$.

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 (*).

4.20.21. 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 : $F_V(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ 1 - e^{-x} & \text{si } x > 0 \end{cases}$.

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 $X_1, \dots, X_n$ 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 $Y_n = \max(X_1, X_2, \dots, X_n)$, c'est à dire que pour tout ω de Ω , on a : $Y_n(\omega) = \max(X_1(\omega), X_2(\omega), \dots, X_n(\omega))$. 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 :

```

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

```

Script (1)

```

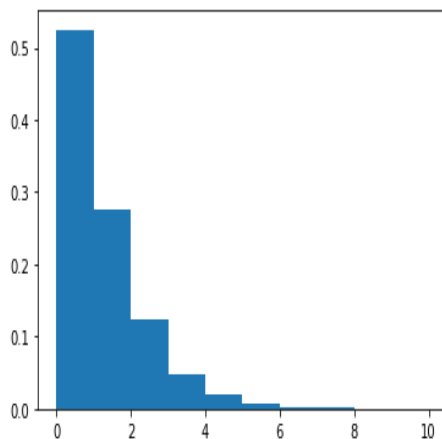
1 n=int(input('entrez la valeur de n :'))
2 Z = [] # La liste Z est vide
3 for k in range(10000):
4     Z.append(simulZ(n))
5 s = np.linspace(0,10,11)
6 plt.hist(Z,s,density = True)
7 plt.show()

```

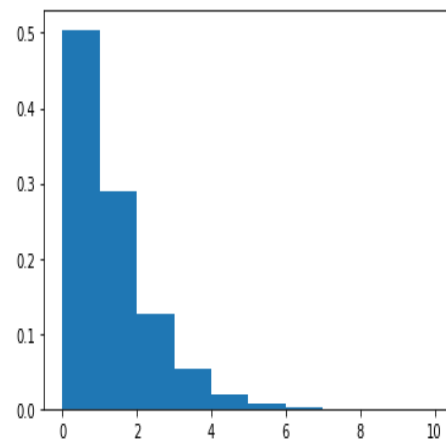
Script (2)

Chacun des scripts simule 10000 variables indépendantes, regroupe les valeurs renvoyées en 10 classes qui sont les intervalles $[0, 1]$, $[1, 2]$, $[2, 3]$, $\dots$, $[9, 10]$ 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) ?

4.20.22. 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** $\text{EML}(n)$ qui simule l'expérience étudiée et renvoie le nombre de boules rouges obtenues lors des n premiers tirages.

On exécute le programme suivant :

```

1  n = 10
2  m = 0
3  for i in range(1000):
4      m = m + EML(n)
5  print(m/1000)

```

On obtient 6.657. Comment interpréter ce résultat ?

4.20.23. 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 :
$$f_{a,b}(x) = \begin{cases} (a + bx) \exp\left(-ax - \frac{b}{2}x^2\right) & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}.$$

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 ?

```

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

```

4.21. Statistiques bivariées

4.21.1. ECRICOME 2026

On considère une série statistique à deux variables (x, y) .

Expliquer ce que renvoie la fonction `Myst` suivante qui prend en argument d'entrée la série statistique (x, y) sous forme de tableau `Numpy` à deux lignes.

```
1 import numpy as np
2
3 def Myst(Tab):
4     M=np.cov(Tab)
5     R=M[0,1]/np.sqrt(M[0,0]*M[1,1])
6     return(R)
```

Aide mémoire. La fonction `np.cov` prend en argument une série statistique à deux variables (x, y) sous forme de tableau `Numpy` à deux lignes et renvoie, sous forme de tableau `Numpy`, la matrice suivante :

$$\begin{pmatrix} v_x & s_{x,y} \\ s_{x,y} & v_y \end{pmatrix},$$

où v_x , v_y et $s_{x,y}$ désignent respectivement la variance de la série statistique x , la variance de la série statistique y et la covariance empirique de la série statistique double (x, y) .