DS n°5

22/03/2025

Informatique

Durée : 2h

Exercice 1 : Suites

1. Importer le package `numpy` sous son alias habituel.

2. Soit la suite (a_n) définie par :

$$\begin{cases} a_0 = 1 \\ \forall\, n \in \mathbb{N},\ a_{n+1} = a_n + \dfrac{1}{a_n} \end{cases}$$

   (a) Écrire une fonction `a(n)` qui renvoie la valeur de a_n.

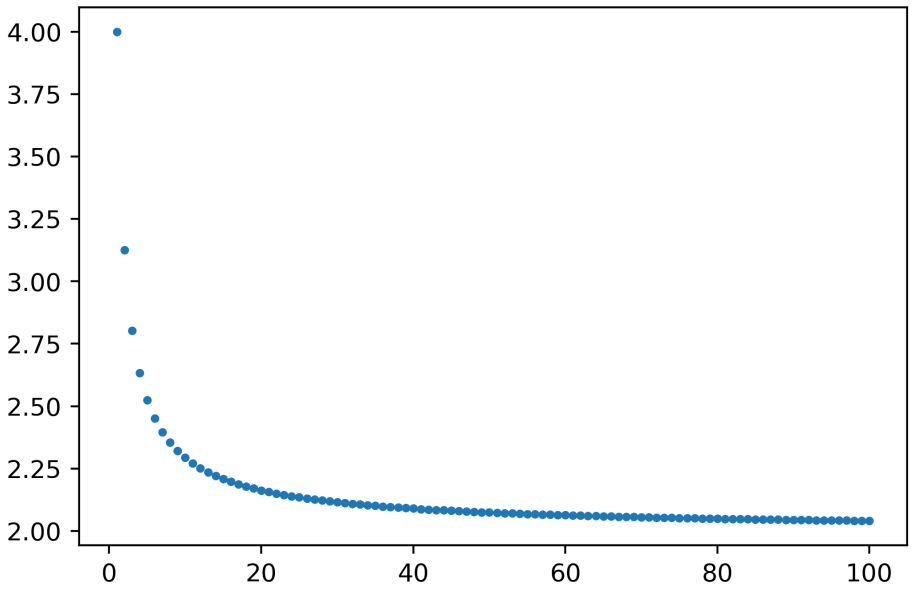
   (b) Écrire une fonction `liste_a(n)` qui renvoie la liste $[a_0, a_1, ..., a_n]$ sous forme d'un `np.array`.

   (c) Importer le package permettant de tracer des représentations graphiques, sous l'alias `plt`.

   (d) Les commandes suivantes :

```
1 L=liste_a(100)
2 M=[L[k]**2/k for k in range(1,101)]
3 plt.scatter(range(1,101),M)
4 plt.show()
```

   renvoient l'affichage



   Quel équivalent de a_n pour $n \to +\infty$ peut-on déduire de cette figure ?

3. Soit la suite (b_n) définie par :

$$\begin{cases} b_0 = 1 \\ \forall\, n \in \mathbb{N},\ b_{n+1} = \displaystyle\sum_{k=0}^{n} (k+1) b_{n-k} = b_n + 2b_{n-1} + \cdots + nb_1 + (n+1)b_0 \end{cases}$$

   On cherche à coder une fonction qui calcule le n-ième terme de cette suite.
   On remarque pour cela que pour tout $n \in \mathbb{N}$, b_{n+1} est la somme des éléments de la liste obtenue en effectuant le produit terme à terme des listes

$$[b_n, b_{n-1}, \ldots, b_2, b_1, b_0] \quad \text{et} \quad [1, 2, \ldots, n-1, n, n+1]$$

Compléter la fonction suivante qui calcule b_n, un entier $n \in \mathbb{N}$ étant passé en argument :

```
def suite_b(n):
    B = .... # initialisation
    for k in range(n):
        L = ....
        b = np.sum(L*np.array(B))
        B = .....
    return .....
```

Exercice 2 : Algèbre linéaire

1. Importer sous l'alias `al` le package python d'outils d'algèbre linéaire.

2. Écrire une ligne de code permettant de stocker dans une variable `G`, sous forme d'un `np.array`, la matrice $\begin{pmatrix} -3 & 1 & 2 \\ 1 & -2 & 1 \\ 2 & 1 & -3 \end{pmatrix}$

3. Écrire une ligne de code permettant d'afficher la matrice $G^3(G^2 - I_3)$ (I_3 étant la matrice identité d'ordre 3).

4. La commande `al.eig(G)` renvoie l'affichage :

```
(array([ 8.8817842e-16, -5.0000000e+00, -3.0000000e+00]),

 array([[-5.77350269e-01, -7.07106781e-01,  4.08248290e-01],
        [-5.77350269e-01,  3.19350444e-16, -8.16496581e-01],
        [-5.77350269e-01,  7.07106781e-01,  4.08248290e-01]]))
```

   En déduire le spectre de G (toutes les valeurs propres sont dans $\mathbb{Z}$) et donner une base de $E_0(G)$.

Exercice 3 : Tirages aléatoires

1. Écrire la ligne permettant d'importer, sous l'alias `rd`, le package usuel de Python permettant d'effectuer des expériences aléatoires.

2. On effectue 10 lancers d'une pièce équilibrée, et on note Y le nombre de « Face » obtenus.
   Programmer une fonction Python `tirage_Y` qui renvoie un tirage aléatoire de Y.

3. On note k le nombre de Face obtenus à l'issue des 10 premiers tirages.
   On remplit alors une urne avec k boules blanches et $10-k$ boules noires ; on tire au hasard une boule dans cette urne.
   Programmer une fonction Python `tirage_2` qui simule la totalité de l'expérience aléatoire, et renvoie 1 si la boule tirée est noire et 0 si elle est blanche.
   Cette fonction devra obligatoirement appeler la fonction `tirage_Y`.

4. Proposer des commandes Python permettant d'obtenir une approximation de la probabilité de tirer une boule noire.
   Justifier votre réponse.

Exercice 4 : À la fête foraine

On considère une machine à sous constituée de 3 roues mobiles, fonctionnant de manière indépendante. Chaque roue comporte m secteurs ($m \geqslant 2$), numérotés de 1 à m, dont un seul est gagnant. Un tirage consiste à faire tourner une ou plusieurs roues, qui s'arrêtent alors sur un secteur au hasard de manière équiprobable. On gagne quand toutes les roues sont sur le secteur m.
Initialement, aucune roue n'est sur le secteur m. Un joueur joue plusieurs parties successives : à chaque partie, il fait tourner les roues qui ne sont pas sur le secteur gagnant, et laisse les autres.
Dans la suite on admettra que, presque sûrement, le joueur gagnera au bout d'un nombre fini de parties.

1. On suppose ici tous les imports nécessaires effectués.
   Quelle commande permet de renvoyer p tirages indépendants d'une variable aléatoire $X \hookrightarrow \mathscr{U}(\llbracket 1, m \rrbracket)$?

2. Compléter la fonction suivante, qui simule les parties successives d'un joueur, et renvoie le nombre de parties nécessaires pour gagner.
   Dans le programme qui suit la liste `roues` contient trois composantes ; `roues[i]` est égal au numéro du secteur sur lequel se trouve la roue $i+1$.

```
def jeu(m):
    roues = ... # premier tirage
    n = 1
    while roues != ..... :
        for k in range(3):
            if roues[k] ... :
                roues[k] = ...
        n = ...
    return n
```

3. Le joueur paie un droit d'entrée de 1 € pour jouer à ce jeu ; à chaque essai il paie 2 €. Lorsqu'il gagne il reçoit 10 €.
   Modifier le programme précédent pour qu'il renvoie le gain algébrique du joueur à l'issue du jeu.

Exercice 5

À l'aide de l'algorithme de dichotomie sur la fonction $x \mapsto x^2 - 2$, écrire un code renvoyant une valeur approchée à 0.0001 près de $\sqrt{2}$. Comme on sait que $1 < \sqrt{2} < 2$, on initialisera l'intervalle de recherche d'une solution à $[1,2]$.

Exercice 6

On admet que si $U \hookrightarrow \mathscr{U}([0,1[)$, alors la variable $\dfrac{1}{\sqrt{1-U}}$ suit la loi de Pareto de paramètres (1,2) (notée Par(1,2)).
À l'aide de cette propriété, coder une fonction `Pareto12(n)` qui renvoie un `np.array` de n tirages indépendants d'une variable $Y \hookrightarrow \text{Par}(1,2)$.

Exercice 7

Soit $(X_n)_{n\in\mathbb{N}}$ une suite de variables indépendantes et identiquement distribuées, de loi $\mathscr{E}(\alpha)$.
Soit également $N \hookrightarrow \mathscr{P}(10)$, indépendante des X_n.
On considère la variable Y telle que

$$\forall\, \omega \in \Omega,\ Y(\omega) = \sum_{n=0}^{N(\omega)} X_n(\omega)$$

Autrement dit, $Y = X_0 + \cdots + X_N$ où N est aléatoire et suit la loi de Poisson de paramètre λ.

1. Programmer une fonction Python `YN(alpha)` qui renvoie un tirage du couple (Y,N) ; c'est-à-dire, pour une même issue ω, le couple $(Y(\omega),N(\omega))$.

2. On tape les commandes suivantes :

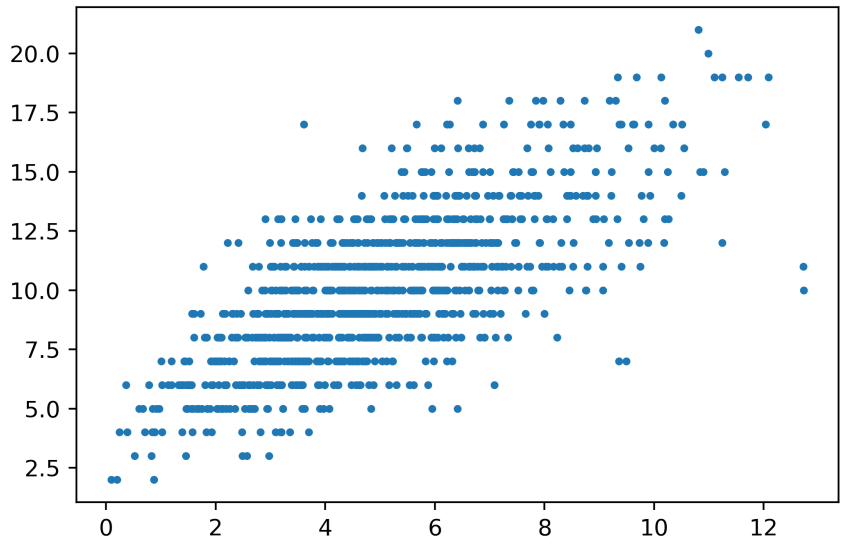
```
1 tirages = [YN(2) for k in range(1000)]
2 L_Y = np.array([t[0] for t in tirages])
3 L_N = np.array([t[1] for t in tirages])
```

   Expliquer ce que contiennent les listes `L_Y` et `L_N`.

3. Donner une commande permettant, à partir des listes créées, d'afficher une approximation de l'espérance de Y.

4. Donner des commandes permettant de tracer le nuage de points représentant les 1000 tirages du couple (Y,N) (on mettra N en ordonnées).

Avec les données issues d'une simulation, on obtient la figure suivante :



5. Donner une commande permettant de définir une variable `cov` qui contiendra la covariance empirique des listes `L_Y` et `L_N`.

6. Donner une commande permettant de définir une variable `corr` qui contiendra le coefficient de corrélation linéaire des listes `L_Y` et `L_N`.
   Rappel : l'écart-type d'une série statistique stockée dans une liste `L` s'obtient par `np.std(L)`.

7. Lors d'une expérience, `corr` prend l'une des 4 valeurs suivantes

   `0.04` `-0.801` `0.689` `0.998`

   Laquelle ? (justifier)

8. Lors d'une expérience, la commande écrite en question 3 renvoie une des valeurs suivantes

   `0.04` `5.051` `9.872`

   Laquelle ? (justifier)