TP9 : Chaînes de Markov sur un ensemble d'états fini

I. Simulation d'une chaîne de Markov

I.1. Illustration sur un exemple

Doudou le hamster passe son temps entre ses trois activités favorites : dormir, manger et faire du sport dans sa roue. Au début de la jourée, il mange, et à chaque heure, il change d'activité selon les critères suivants.

1) Si, à l'heure n, il est en train de manger, alors il va dormir l'heure suivante avec probabilité 0.7 et faire de l'exercice avec probabilité 0.3.

2) Si, à l'heure n, il est en train de dormir, alors il continue à dormir l'heure $n+1$ avec probabilité 0.4, il va manger avec probabilité 0.3 et il va faire de l'exercice avec probabilité 0.3.

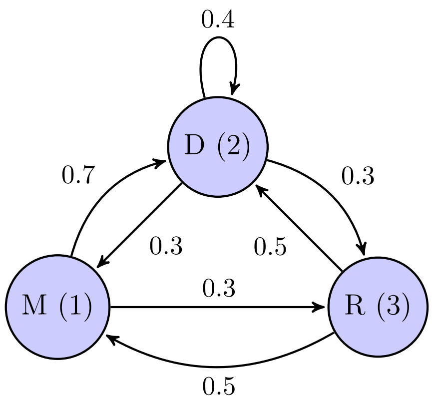
3) Si, à l'heure n, il est en train de faire de la roue, il va manger l'heure suivante avec probabilité 0.5 et il va dormir avec probabilité 0.5.

On s'intéresse ici à l'évolution du comportement de Doudou. On souhaite notamment déterminer si l'une de ses activités prendra, à terme, le dessus sur les autres.

I.2. Modélisation mathématique

On modélise ce problème comme suit.

- On commence par numéroter les activités par un entier entre 1 et 3.
- On note X_n la v.a.r. égale à l'état du hamster à l'heure n.
  Ainsi, la suite de v.a.r. (X_n) représente l'évolution des activités du hamster.
- Cette évolution peut être modélisée par le graphe suivant.



- On définit enfin la `matrice de transition` A associée au problème. Il s'agit de la matrice :

$$A = (a_{i,j}) \quad \text{où} \quad a_{i,j} = \mathbb{P}_{[X_n=i]}\big([X_{n+1}=j]\big)$$

($a_{i,j}$ représente la probabilité de passage de l'état i à l'état j)

I.3. Étude de la matrice de transition

Avant d'entamer l'étude en **Python** de ce problème, on importe les modules nécessaires.

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

- Déterminer la matrice de transition du problème précédent.
  Écrire l'appel permettant de la stocker dans une variable `A`.

- Déterminer $\mathbb{P}_{[X_0=i]}\big([X_1 = j]\big)$. À quel coefficient de la matrice A cela correspond-il ?

- Soit $(i_0, \dots, i_n, i_{n+1}) \in [\![1, 3]\!]^{n+2}$. Que vaut $\mathbb{P}_{[X_0=i_0]\,\cap\,\dots\,\cap\,[X_n=i_n]}\big([X_{n+1} = i_{n+1}]\big)$?

Ces deux propriétés font de (X_n) une chaîne de Markov homogène.

I.4. Étude de l'évolution du comportement du hamster

Dans la suite, on considère la matrice ligne U_n qui définit la loi de X_n :

$$U_n = \begin{pmatrix} \mathbb{P}\big([X_n = 1]\big) & \mathbb{P}\big([X_n = 2]\big) & \mathbb{P}\big([X_n = 3]\big) \end{pmatrix}$$

- Déterminer la probabilité $\mathbb{P}\big([X_{n+1} = 1]\big)$ en fonction de $\mathbb{P}\big([X_n = 1]\big)$, $\mathbb{P}\big([X_n = 2]\big)$ et $\mathbb{P}\big([X_n = 3]\big)$ et des coefficients de la matrice A.

- Déterminer de même $\mathbb{P}\big([X_{n+1} = 2]\big)$ et $\mathbb{P}\big([X_{n+1} = 3]\big)$.
  En déduire que pour tout $n \in \mathbb{N}$, $U_{n+1} = U_n \times A$.
  Exprimer enfin U_n en fonction de A et de U_0.

- Que réalise l'opération `A*A` ? Et l'opération `A**2` ?
  Calculer alors `A`^{5}, `A`^{10} et `A`^{20}. Que remarque-t-on ?

- Écrire une fonction `simuMarkovEtape(x0,A)` qui prend en paramètre l'état initial x_0 et la matrice de transition A et renvoie une simulation de la v.a.r. X_1.

- Évaluer `simuMarkovEtape(2,A)` une dizaine de fois de suite.
  Le résultat obtenu paraît-il cohérent ?

- Écrire une fonction `simuMarkov(X0,A,n)` qui prend en paramètre l'état initial x_0 et la matrice de transition A et renvoie une simulation de la v.a.r. X_n.

I.5. Comportement asymptotique du hamster

On souhaite conjecturer le comportement de la loi de X_n lorsque $n \to +\infty$.
Pour ce faire, on compare :

× la valeur théorique de U_n (loi de X_n) pour n grand,

× une valeur approchée de U_n obtenue en recueillant les effectifs de chaque état à la suite de N simulations de trajectoires de taille n.

- Partant de l'état initial `x0`, par quel appel obtient-on U_n ?

- Compléter le programme suivant.

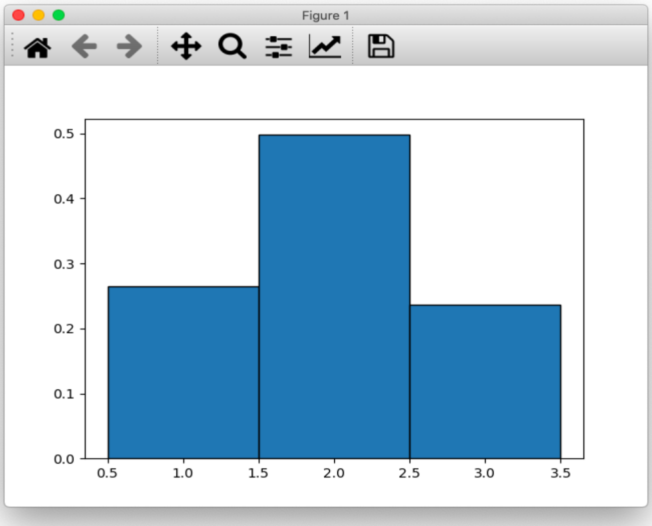
```
# Valeur des paramètres
N = 1000
n = 50
x0 = 2

# Distribution théorique
P =

# Valeurs observées
Obs = [                    for k in range(N)]

# Tracés des histogrammes
cl = [0.5, 1.5, 2.5, 3.5]
# Diagramme des fréquences observées
plt.hist(Obs, normed = True, bins = cl, edgecolor = ‘‘black‘‘,
        label = ‘‘Distribution approchée‘‘)
```

On obtient le diagramme suivant :



- Quel diagramme obtient-on si l'on part intialement avec un autre état `x0` ?

- Que peut-on en conclure sur le comportement à terme du hamster ?

- Si l'on est capable de démontrer que la suite (U_n) converge vers une limite notée U_∞, quelle relation peut-on établir entre U_∞ et A ?

II. Exercices sur les matrices

II.1. Démontrer qu'une matrice est une matrice de transition

La somme des coefficients de chaque ligne de la matrice A vaut 1 : $\forall i \in [\![1,3]\!], \ \sum\limits_{j=1}^{3} a_{i,j} \ = \ 1.$

- Démontrer cette égalité.

- Comment récupère-t-on en **Python** la **taille** d'une matrice `A` ?
  Comment récupère-t-on le nombre de lignes de `A` ? Et le nombre de colonnes ?

- Écrire une fonction `somLigne` qui :
  - prend en paramètre une matrice `P` et un numéro de ligne `j` (arguments d'entrée),
  - renvoie la somme des coefficients de la ligne `j` de la matrice `P`,

- Tester la fonction `somLigne` sur la matrice de transition `A`.
- Écrire une fonction `verifUn` permettant de vérifier que la somme des coefficients de chaque ligne d'une matrice `P` vaut 1.

- Vaut-il mieux utiliser une boucle while ou une boucle for pour cette fonction ?

- La fonction `verifUn` est-elle suffisante pour démontrer que `A` est une matrice de transition ?

- Écrire une fonction `tousPositifs` qui prend en paramètre une matrice `P`, et teste si tous les coefficients de la matrice `P` sont positifs.

- Tester la fonction `tousPositifs` sur la matrice `A`.
- Quelle est la complexité de cette fonction ?

- Écrire alors une fonction `estMTransition` qui prend en paramètre une matrice `P` et teste si `P` est une matrice de transition.

- Tester la fonction `estMTransition` sur la matrice `A`.

II.2. Démontrer qu'une matrice définit la loi d'un couple de v.a.r. discrètes

Le but de cette section est d'écrire une fonction permettant de vérifier qu'une matrice donnée en paramètre définit la loi d'un couple de v.a.r. discrètes.

Commençons par rappeler la caractérisation de la loi d'un couple de v.a.r. discrètes.

Théorème 1.

Soient I et J deux parties de $\mathbb{N}$.

Notons $\{x_i \mid i \in I\}$ et $\{y_j \mid i \in J\}$ deux parties de $\mathbb{R}$ et $(p_{i,j})_{(i,j)\in I\times J}$ une famille de réels.

L'application $(x_i, y_j) \mapsto p_{i,j}$ est la loi d'un couple de v.a.r. discrètes $\Leftrightarrow \begin{cases} \textit{1)}\ \forall (i,j) \in I \times J,\ p_{i,j} \geqslant 0 \\ \textit{2)}\ \sum\limits_{i\in I} \sum\limits_{j\in J} p_{i,j} = 1 \end{cases}$

- Écrire une fonction `sommeCoeff` qui prend en paramètre une matrice `P`, et somme l'ensemble de ses coefficients.

On souhaite maintenant écrire une fonction `estLoiCouple` qui prend en paramètre une matrice `P`, et teste si cette matrice définit la loi d'un couple de v.a.r. discrètes.

On propose les deux versions suivantes :

```python
def estLoiCouple1(P):
    (sommeCoeff(P) == 1) & tousPositifs(P)
```

```python
def estLoiCouple2(P):
    tousPositifs(P) & (sommeCoeff(P) == 1)
```

- Comparer la complexité de ces deux fonctions.