

TP Informatique n° 6/7

Chaînes de Markov et exercices sur les matrices

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 journée, il mange, et à chaque heure, il change d'activité selon les critères suivants.

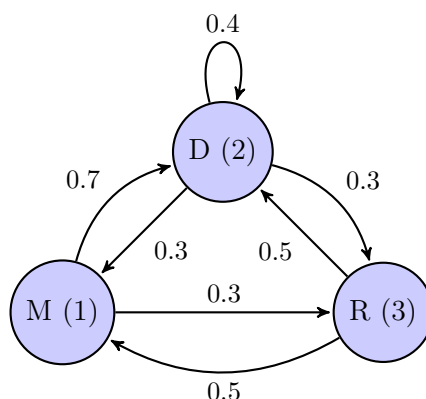
- 1) Si, à l'heure n , il est en train de manger, alors il va dormir l'heure suivante avec probabilité 0.7 et faire de l'exercice avec probabilité 0.3.
- 2) Si, à l'heure n , il est en train de dormir, alors il continue à dormir l'heure $n+1$ avec probabilité 0.4, il va manger avec probabilité 0.3 et 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 :

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

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

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 as np
```

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

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

- ▶ Soit $(i_0, \dots, i_n, i_{n+1}) \in \llbracket 1, 3 \rrbracket^{n+2}$. Que vaut $\mathbb{P}_{(X_0=i_0, \dots, X_n=i_n)}(X_{n+1} = i_{n+1})$?

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 le vecteur U_n qui définit la loi de X_n :

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

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

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

- ▶ Que réalise l'opération $A * A$ si A est de type `array`?
Calculer A^5 , A^{10} et A^{20} à l'aide de la fonction `np.linalg.matrix_power`. 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 $n \rightarrow +\infty$.

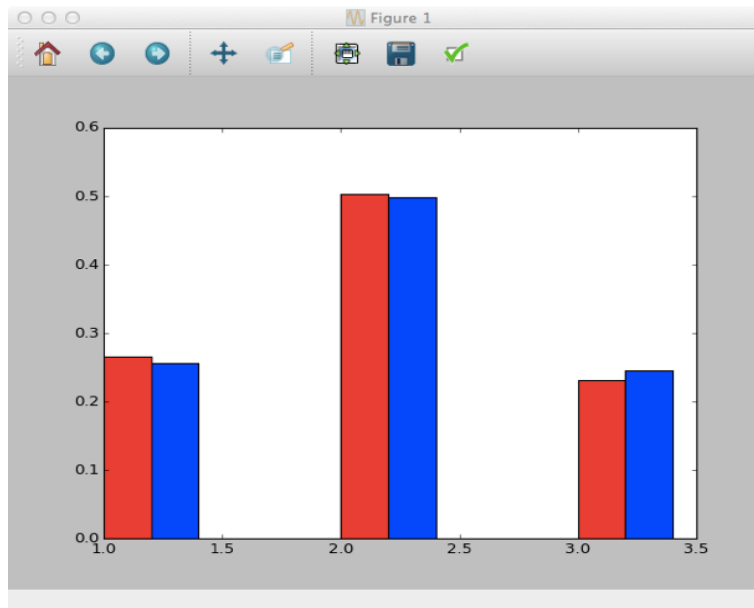
Pour ce faire, on compare :

- × 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 x_0 , par quel appel obtient-on U_n ?

- ▶ Compléter le programme suivant.

```
1 import matplotlib.pyplot as plt
2 # Valeur des paramètres
3 N = 1000
4 n = 50
5 x0 = 2
6
7 # Distribution théorique
8 P =
9
10 # Valeurs observées
11 Obs = [                for k in range(N)]
12
13 # Tableau des effectifs observés
14 cl = np.linspace(1,3,3)
15 effectif = calcEffectif(cl, Obs)
16
17 # Tracés des diagrammes
18 absc = np.linspace(1,3,3)
19 # Diagramme de la distribution théorique
20 plt.bar(absc, P, color = 'r', width = 0.2)
21 # Diagramme des fréquences observées
22 plt.bar(absc+0.2, effectif/N, color = 'b', width = 0.2)
23 plt.show()
```

On obtient le diagramme suivant :



- ▶ Quel diagramme obtient-on si l'on part initialement avec un autre état x_0 ?

- ▶ 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 colonne de la matrice A vaut 1 : $\forall j \in \llbracket 1, 3 \rrbracket, \sum_{i=1}^3 a_{i,j} = 1$.

- ▶ 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 `somColonne` qui :
 - × prend en paramètre une matrice **P** et un numéro de colonne **j** (arguments d'entrée),
 - × renvoie la somme des coefficients de la colonne **j** de la matrice **P**,

- ▶ Tester la fonction `somColonne` sur la matrice de transition **A**.
- ▶ Écrire une fonction `verifUn` permettant de vérifier que la somme des coefficients de chaque colonne 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 $\{y_j \mid j \in J\}$ deux parties de \mathbb{R} et $(p_{i,j})_{(i,j) \in I \times J}$ une famille de réels.

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

- 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 :

```

1 def estLoiCouple1(P):
2     (sommeCoeff(P) == 1) & tousPositifs(P)

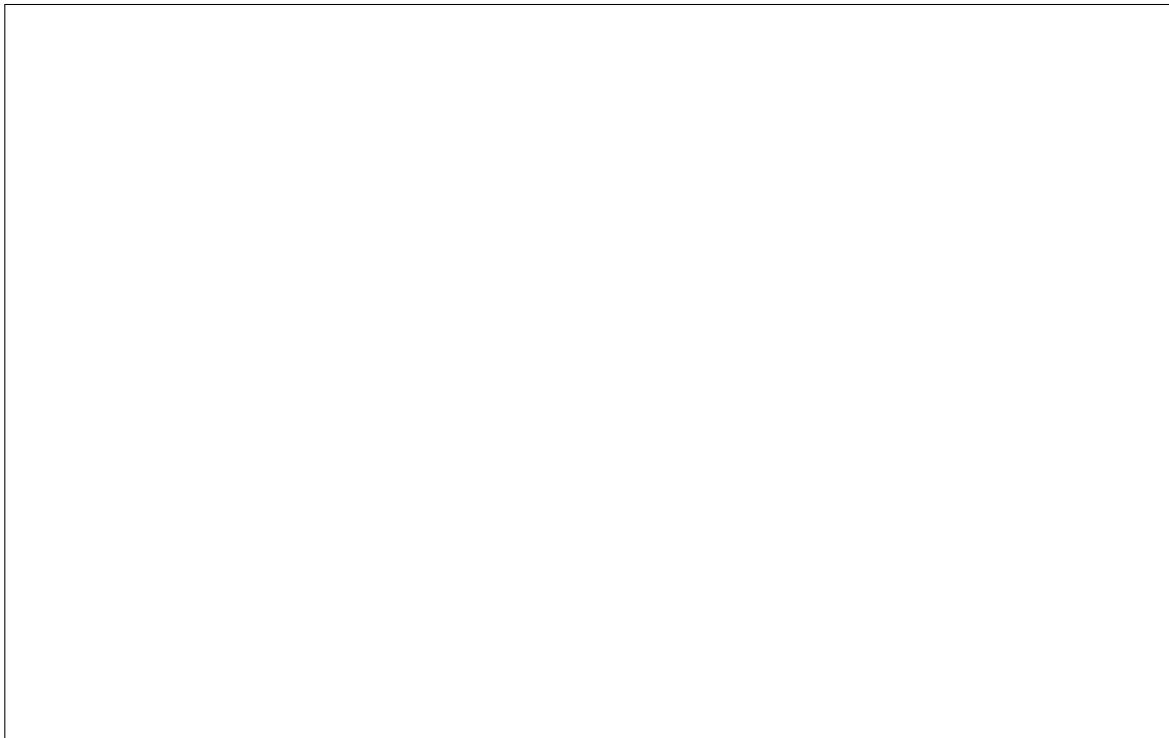
```

```

1 def estLoiCouple2(P):
2     tousPositifs(P) & (sommeCoeff(P) == 1)

```


- Comparer la complexité de ces deux fonctions.



II.3. Exercices de l'épreuve Modélisation Mathématique et Informatique (Agro-Véto 2015)

On souhaite dans ce problème écrire un programme permettant de calculer, sous certaines conditions, un vecteur propre d'une matrice.

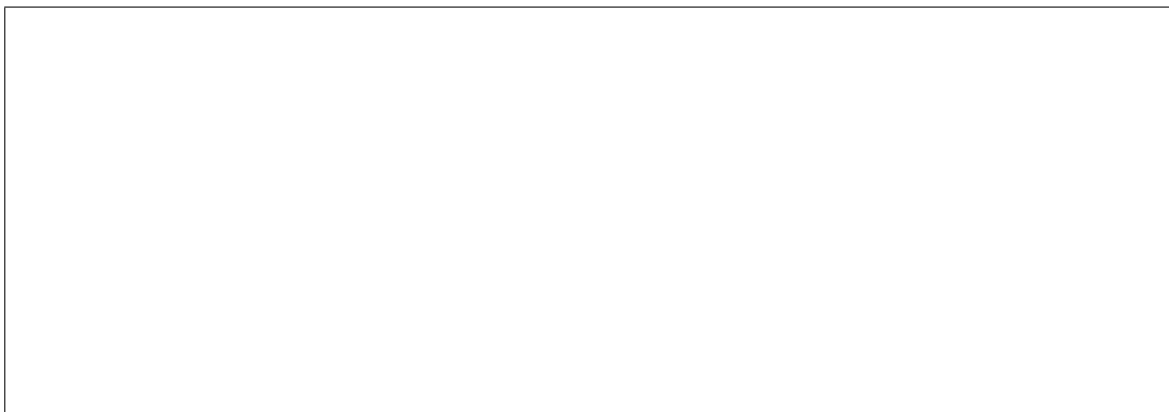
On utilisera les notations suivantes :

- $\mathcal{M}_{m,n}(\mathbb{R})$: ensemble des matrices de taille $m \times n$ à coefficients réels.
- $\mathcal{M}_p(\mathbb{R}) = \mathcal{M}_{p,p}(\mathbb{R})$: ensemble des matrices carrées.
- À chaque matrice $M = (m_{ij}) \in \mathcal{M}_{m,n}(\mathbb{R})$ on associe un nombre réel positif $\|M\|$, appelé **norme** de M , défini par : $\|M\| = \max_{i,j} |m_{ij}|$.

Remarquons que $\|M\|$ est toujours strictement positif, sauf lorsque la matrice M est nulle.

- Écrire une fonction `Norme(M)` qui étant donnée une matrice M de taille quelconque calcule et renvoie le nombre `Norme(M)`.

On interdit le recours à une quelconque fonction max prédéfinie en Python



- Écrire une fonction `Normalise(v)` qui étant donnée une matrice colonne $v \in \mathcal{M}_{p,1}(\mathbb{R})$ non nulle renvoie une nouvelle matrice colonne \tilde{v} , de même forme, égale à : $\tilde{v} = \frac{v}{\|v\|}$.

On se donne à présent une matrice carrée $A \in \mathcal{M}_p(\mathbb{R})$. Soit v_0 un élément quelconque de $\mathcal{M}_{p,1}(\mathbb{R})$. En supposant qu'aucun des termes n'est dans le noyau de A , on peut former la suite $(u_n)_{n \geq 0}$ d'éléments de $\mathcal{M}_{p,1}(\mathbb{R})$ définie par la relation de récurrence :

$$\forall n \in \mathbb{N}, v_{n+1} = \frac{Av_n}{\|v_n\|}$$

- Écrire une fonction `PuissanceIteree(A, n)` qui étant donnée une matrice carrée A et un entier naturel n , détermine la taille p de A , choisit aléatoirement une matrice colonne $v_0 \in \mathcal{M}_{p,1}(\mathbb{R})$, puis calcule et renvoie, en supposant que tous les termes de la suite ci-dessus sont bien définis, la matrice colonne v_n .

On se propose d'écrire maintenant une fonction `VecteurPropre(A, e)` qui étant donnée une matrice carrée A et un nombre $e > 0$ calcule les termes de la suite (v_n) jusqu'à ce que deux termes successifs vérifient $\|v_n - v_{n+1}\| < e$ et renvoie alors la matrice colonne v_{n+1} . Voici trois propositions de programme.

```

1 def VecteurPropre1(A, e)
2     d = A.shape
3     v = matrix(random.rand(d[0],1))
4     v = Normalise(v)
5     w = Normalise(A*v)
6     while Norme(v - w) >= e:
7         v = w
8         w = Normalise(A*v)
9     return w

```

```

1 def VecteurPropre2(A, e)
2     d = A.shape
3     v = matrix(random.rand(d[0],1))
4     v = Normalise(v)
5     w = Normalise(A*v)
6     ecart = Norme(v - w)
7     while ecart >= e:
8         v = w
9         w = Normalise(A*v)
10    return w

```

```

1 def VecteurPropre3(A, e)
2     d = A.shape
3     v = matrix(random.rand(d[0],1))
4     v = Normalise(v)
5     while Norme(v - Normalise(A*v)) >= e:
6         v = Normalise(A*v)
7     return Normalise(A*v)

```

Note sur ces différents programmes

- Il était précisé dans l'énoncé que la librairie NumPy était ouverte.
- Comme on peut le constater, les concepteurs ont fait ici le choix d'utiliser la classe `matrix`. L'intérêt est que l'opérateur `*` a été surchargé afin que $A * B$ réalise la multiplication matricielle si A et B sont des objets de cette classe. Les lignes 3 de chacun des programmes précédents permettent de transformer un objet de type `array` en objet de la classe `matrix`.
- Parmi ces trois programmes, indiquer lequel est (ou lesquels sont) correct(s). Pour chaque programme *incorrect* on indiquera succinctement ce qui ne va pas.