

Devoir maison n°6
Rendu libre, courant mars
À faire pas loin de son ordi !

Simulation « générale » d'une chaîne de Markov

Dans tout ce qui suit on aura importé les packages usuels de Python par :

```
import numpy as np
import numpy.random as rd
import numpy.linalg as al
```

Le but de ce problème est de coder une fonction qui prendra en entrée la matrice de transition d'une chaîne de Markov, et permettra de simuler des trajectoires sur le graphe associé.
On utilisera ensuite cette fonction pour effectuer des statistiques et observer des phénomènes possiblement intéressants.

1 Simulation d'une variable à valeurs dans $\llbracket 1, n \rrbracket$ de loi quelconque

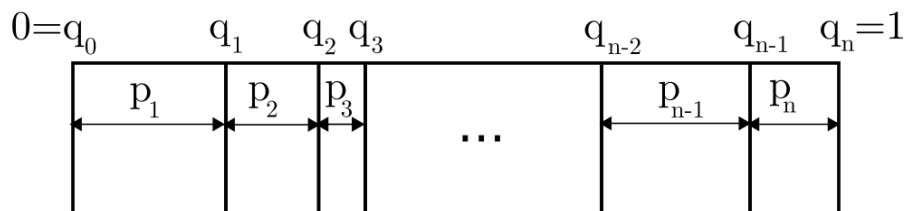
On considère ici un entier $n \in \mathbb{N}^*$, et une variable X à valeurs dans $\llbracket 1, n \rrbracket$, telle que $P(X = x_i) = p_i$. Les p_i vérifient donc : $\forall i \in \llbracket 1, n \rrbracket, p_i \geq 0$; et $\sum_{i=1}^n p_i = 1$. On cherche à simuler des tirages de X .

On procède de la manière suivante :

- On pose $q_0 = 0$, $q_k = \sum_{i=0}^k p_i$ jusqu'à $q_n = p_1 + \dots + p_n = 1$.
- On découpe l'intervalle $[0, 1]$ en :

$$[0, 1] = [q_0, q_1[\cup [q_1, q_2[\cup \dots \cup [q_{n-2}, q_{n-1}[\cup [q_{n-1}, 1]$$

comme sur le dessin ci-dessous :



On note $I_k = [q_{k-1}, q_k[$ pour $i \in \llbracket 1, n-1 \rrbracket$, et $I_n = [q_{n-1}, 1]$

- On effectue un tirage aléatoire $U \hookrightarrow \mathcal{U}([0, 1])$ et on regarde à quel intervalle I_k appartient la valeur tirée.

Soit k l'entier de $\llbracket 1, n \rrbracket$ tel que U appartienne à I_k .

1. Dessiner les intervalles I_1, I_2, I_3, I_4 pour $n = 4$, $p_1 = \frac{1}{8}$, $p_2 = \frac{1}{4}$, $p_3 = \frac{1}{2}$, $p_4 = \frac{1}{8}$.
2. Expliquer pourquoi k existe, et est unique.
3. Montrer que $P(U \in I_k) = p_k$ (en utilisant la fonction de répartition de U).
4. Montrer qu'à l'issue de cette expérience, la variable X égale à k vérifie la loi demandée.

On va maintenant programmer une fonction réalisant ce tirage.

5. Montrer que $U \in I_k \Leftrightarrow \left(\sum_{i=0}^{k-1} p_i < U \text{ et } \sum_{i=0}^k p_i \geq U \right)$.

lire $U \in I_k \Leftrightarrow \left(\sum_{i=0}^{k-1} p_i \leq U \text{ et } \sum_{i=0}^k p_i > U \right) \dots$ pour $k \leq n-1$ et $\Leftrightarrow \sum_{i=0}^{n-1} p_i \leq U$
pour $k = n$.

6. bof

Décrire un moyen, à l'aide d'une boucle `while`, de trouver la valeur de k étant donnée la liste $[p_1, \dots, p_n]$.

7. Compléter la fonction Python suivante pour qu'elle prenne en argument une liste $[p_1, \dots, p_n]$ de réels positifs, de somme 1 ; et renvoie un tirage de X décrite en début de partie.

```
def tirage(L):
    """L=[p1,p2,...,pn] est la loi de proba : P(X=i)=pi """
    u=rd.random()
    s = ... # s sera égal à q_k
    k = ... # numéro du segment I_k
    while ... : # si u n'est pas dans le segment k, on passe au suivant
        s = s + ...
        k = ...
    return k
```

8. reformuler pour faire parler de nombreuses simulations de X

Expliquer comment on pourrait s'assurer que cette fonction produit le résultat attendu.

9. Généraliser cette fonction pour générer des tirages d'une variable aléatoire quelconque d'univers-image fini, ie une variable de loi :

| | | | | |
|----------|-------|-------|---------|-------|
| x | x_1 | x_2 | \dots | x_n |
| $P(X=x)$ | p_1 | p_2 | \dots | p_n |

où les x_i sont des réels quelconques, et les p_i sont positifs de somme 1. La fonction recevra comme arguments les listes $X = [x_1, \dots, x_n]$ et $P = [p_1, \dots, p_n]$.

NB : cette généralisation ne servira pas dans la suite du problème.

2 Un exemple illustratif

Pour tester les codes des sections suivantes, on pourra utiliser le graphe probabiliste dont la matrice de transition est

$$M_0 = \begin{pmatrix} 1/3 & 1/6 & 1/2 \\ 1/2 & 1/2 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

On n'oubliera pas de manipuler un `np.array` :

```
M0=np.array([[1/3,1/6,1/2],[1/2,1/2,0],[1,0,0]])
```

10. Dessiner le graphe probabiliste associé à la matrice M_0 .

11. On exécute le code suivant :

```
D,P=al.eig(np.transpose(M0))
print(D)
print(P)
```

et on obtient la sortie :

```
[-0.59023021  1.          0.42356354]

[[ 0.75788151 -0.85714286  0.37403387]
 [-0.11585955 -0.28571429 -0.81556606]
 [-0.64202195 -0.42857143  0.44153218]]
```

Expliquer cette sortie. Comment **+ donner les commandes Python associées** déterminer les états stables de cette chaîne de Markov ?
On trouve qu'une valeur approchée de l'état stable (unique) de cette chaîne est :

(0.5455 0.1818 0.2727)

12. On rappelle que la commande `np.dot(A,B)` permet de faire le produit matriciel de deux matrices A et B de format convenable.

Que fait le programme suivant ?

```
V=[1/3,1/3,1/3]
for k in range(20):
    V=np.dot(V,M0)
    print(V)
```

Quelle semble être la limite de V pour un grand nombre d'itérations ? Ce résultat **semble-t-il être** est-il modifié si on prend une valeur initiale de V différente ? (tout en gardant une distribution de probabilité bien sûr... 3 composantes positives et de somme 1).

3 Balade(s) sur le graphe probabiliste

On considère un graphe probabiliste de matrice d'adjacence $M \in \mathcal{M}_n(\mathbb{R})$. Les sommets de ce graphe sont numérotés $1, 2, \dots, n$. On rappelle que l'indexation en Python commence à 0 : ainsi, pour tous $(i, j) \in [0, n-1]^2$, $M[i, j]$ est donc la probabilité de passer de l'état $i+1$ à l'état $j+1$.

On prendra soin de tester les fonctions suivantes sur l'exemple de la section 2.

13. Compléter la fonction `trajectoire(M, etat_initial, nb_etapes)` qui prend en arguments :

- la matrice de transition $M \in \mathcal{M}_n(\mathbb{R})$ de la chaîne ;
- un état initial qui est un entier de $[1, n]$;
- le nombre d'étapes de temps `nb_etapes` à simuler

et simule un parcours sur le graphe probabiliste, en partant de l'état initial spécifié.

La fonction devra renvoyer une liste de `nb_etapes+1` nombres : le premier est le numéro de l'état initial, et les suivants les numéros des états visités lors du parcours.

On notera que si on est dans l'état i , les probabilités de transition vers les autres états de la chaîne se lisent sur la ligne $i-1$ de la matrice de transition.

```
def trajectoire(M, etat_initial, nb_etapes):
    traj = ...
    for k ... :
        traj.append(...)
    return np.array(traj) # manipuler des np.array pour pouvoir
                        # extraire des lignes/colonnes facilement
```

14. À l'aide de la fonction `trajectoire`, programmer une fonction `trajectoires(M, etats, nb_etapes)` qui prend en arguments :

- la matrice de transition $M \in \mathcal{M}_n(\mathbb{R})$ de la chaîne ;
- la liste des états initiaux des individus, rangée dans un `np.array` ;
- le nombre d'étapes de temps `nb_etapes` à simuler

et simule les trajectoires des individus.

Cette fonction renverra un `np.array` avec autant de lignes que d'individus, et `nb_etapes+1` colonnes (la première colonne donne les états initiaux et les `nb_etapes` colonnes suivantes les états aux instants 1, 2, ..., `nb_etapes`).

NB : une liste d'états initiaux doit être constituée d'entiers, sinon Python refusera de les considérer comme des numéros d'état. Si vous voulez passer un `np.ones()` comme liste d'états initiaux, il faut que ce soient des entiers. Utiliser `np.ones(k, dtype=int)`.

À ce stade nous sommes capables de simuler l'évolution de plusieurs individus ; donc de faire des statistiques sur l'évolution d'un individu sur ce graphe.

4 Diverses statistiques

On reprend l'exemple à 3 états décrit précédemment ; on suppose que X_0 (position initiale) est la variable constante égale à 1.

Nous avons vu que l'état stable de cette chaîne de Markov est la matrice-ligne approximativement égale à :

$$(0.5455 \quad 0.1818 \quad 0.2727)$$

Nous allons observer de manière empirique la convergence de la chaîne de Markov vers son état stable.

Si $n \in \mathbb{N}^*$ et k est un état de la chaîne, on sait¹ qu'on obtient une bonne approximation de $P(X_n = k)$ en faisant évoluer un grand nombre d'individus initialement dans l'état 1, pendant n étapes de temps, et en mesurant la fraction de ces individus dans l'état k à l'issue de ces n étapes.

15. Écrire une fonction `frequencies` qui prend en argument un `np.array` de nombres, et renvoie sa fraction de composantes égales à 1 (resp à 2, à 3).

Par exemple, `L=[1, 1, 2, 3, 1, 2]` a la moitié de ses composantes égales à 1 ; un tiers de ses composantes égales à 2, et un sixième égales à 3 : `frequencies(L)` doit renvoyer $\left[\frac{1}{2}, \frac{1}{3}, \frac{1}{6}\right]$.

16. À l'aide de la fonction `trajectoires`, construire une matrice `evol` encodant l'évolution sur 20 étapes de temps d'un million (10^6) d'individus initialement dans l'état 1.
17. Isoler la dernière colonne (donc la colonne d'indice 20) de cette matrice. On nomme cette colonne `C` ; `frequencies(C)` renvoie alors :

```
[0.54385, 0.18314, 0.27301]
```

Commenter.

18. On exécute la ligne suivante

```
[frequencies(evol[:,k]) for k in range(21)]
```

qui renvoie :

```
[[1.0, 0.0, 0.0], [0.33246, 0.16656, 0.50098],  
[0.69408, 0.14022, 0.1657], [0.46869, 0.18491, 0.3464],  
[0.59338, 0.17066, 0.23596], [0.51994, 0.18214, 0.29792],  
[0.56513, 0.17491, 0.25996], [0.53653, 0.18099, 0.28248],  
[0.55034, 0.1793, 0.27036], [0.5439, 0.182, 0.2741],  
[0.54731, 0.1812, 0.27149], [0.5452, 0.18177, 0.27303],  
[0.54506, 0.18164, 0.2733], [0.54734, 0.18119, 0.27147],  
[0.54426, 0.18187, 0.27387], [0.54642, 0.18256, 0.27102],  
[0.54546, 0.18155, 0.27299], [0.54456, 0.18317, 0.27227],  
[0.54474, 0.18342, 0.27184], [0.54791, 0.18252, 0.26957],  
[0.54385, 0.18314, 0.27301]]
```

Commenter ce résultat.

19. Le comportement limite observe à la question précédente dépend-il des positions des individus sur le graphe à l'instant initial ? (on répondra en générant des matrices `evol` avec des positions initiales différentes (tout le monde sur 2 ; ou des positions initiales aléatoires obtenues avec un `randint` simulant la loi $\mathcal{U}([1,3])$; ou toute autre recette de votre choix) et en leur appliquant le traitement détaillé ci-dessus.

¹C'est la loi faible des grands nombres ! On reprendra cet argument à la rentrée.

On s'intéresse enfin à une autre observable : regarder, sur un temps d'évolution long, la fraction de temps passée par un individu dans chaque état.

20. Générer la trajectoire d'un individu pendant un grand nombre d'étapes de temps ; calculer la fraction du temps passé par l'individu dans l'état 1 (resp. l'état 2, l'état 3). Que valent ces fractions ?

On parle de *propriété ergodique*.