7
Simulation d'une chaîne de Markov

I

Modélisation d'une marche aléatoire

1. Une marche aléatoire
Un pion se déplace aléatoirement sur trois points distincts A, B et C.
1.1 Initialement, on suppose que ce pion se trouve sur le point A.
1.2 Entre l'instant n et l'instant $(n+1)$, on suppose que le pion a une chance sur deux de rester sur place. Dans le cas contraire, il se déplace de manière équiprobable vers l'un des deux autres points.
1.3 On suppose en outre que le phénomène vérifie la propriété de Markov : la manière dont le pion se déplace entre l'instant n et l'instant suivant $(n+1)$ de dépend que de la position occupée à l'instant n (et pas des positions successivement occupées depuis le début du mouvement).

2. On traduit ce phénomène aléatoire par une suite $(X_n)_{n\in\mathbb{N}}$ de variables aléatoires définies sur un espace probabilisé $(\Omega, \mathcal{A}, \mathsf{P})$, où chaque variable X_n décrit la position occupée à l'instant n :

$$X_n \,:\, \Omega \to E = \{A, B, C\}.$$

2.1 La condition initiale se traduit alors par le fait que la variable aléatoire X_0 suit une loi de Dirac :

$$\mathsf{P}(X_0 = A) = 1.$$

2.2 La loi d'évolution entre l'instant n et l'instant suivant s'exprime au moyen de probabilités conditionnelles : sachant où le pion se trouve à l'instant n, on sait avec quelles probabilités on peut le trouver aux différentes positions à l'instant $(n+1)$.

$$\forall\, i, j \in E, \quad \mathsf{P}(X_{n+1} = j \mid X_n = i) = \left| \begin{array}{ll} 1/2 & \text{si } i = j, \\ 1/4 & \text{si } i \neq j. \end{array} \right.$$

2.3 Enfin, la propriété de Markov se traduit par l'identité suivante : quel que soit $k \in \mathbb{N}^*$, quels que soit x_0, x_1, …, x_k dans E,

$$\mathsf{P}(X_k = x_k \mid X_0 = x_0, X_1 = x_1, \ldots, X_{k-1} = x_{k-1}) = \mathsf{P}(X_k = x_k \mid X_{k-1} = x_{k-1}).$$

2.4 On se gardera de croire que l'identité précédente est toujours vraie.
C'est une règle communément utilisée pour modéliser des processus aléatoires simples (les **processus de Markov**), où l'on considère que le passé du processus n'a pas plus d'influence que le présent sur le futur : on peut imaginer que de tels processus n'ont qu'une *mémoire immédiate*.

3. Loi du processus

La propriété de Markov, telle qu'on vient de la traduire mathématiquement permet de calculer la loi du processus $(X_n)_{n\in\mathbb{N}}$ au moyen de la formule des probabilités composées : quels que soient $n \in \mathbb{N}^*$, quels que soient x_0, …, x_n dans E,

$$\begin{aligned}\mathbf{P}(X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1}, X_n = x_n) \\ = \mathbf{P}(X_0 = x_0)\,\mathbf{P}(X_1 = x_1 \mid X_0 = x_0) \cdots \mathbf{P}(X_{n-1} = x_{n-1} \mid X_{n-2} = x_{n-2}) \\ \times \mathbf{P}(X_n = x_n \mid X_{n-1} = x_{n-1}).\end{aligned}$$

4. Réalisation du processus

Selon la théorie de Kolmogorov, on peut supposer qu'il existe, sur l'espace probabilisé $(\Omega, \mathcal{A}, \mathbf{P})$, une suite $(U_n)_{n\in\mathbb{N}}$ de variables aléatoires indépendantes qui suivent toutes la loi uniforme sur le segment $[0, 1]$.

4.1 Soit U, une variable aléatoire sur $(\Omega, \mathcal{A}, \mathbf{P})$ qui suit la loi uniforme sur $[0, 1]$.
Admettons qu'il existe une application $\varphi : [0, 1] \to E$ telle que

$$\forall\, i \in E, \quad \mathbf{P}\big(\varphi(U_0) = i\big) = \mathbf{P}(X_0 = i)$$

et une application

$$f : [0, 1] \times E \to E$$

telle que

$$\forall\, i, j \in E, \quad \mathbf{P}\big(f(U, i) = j\big) = \left| \begin{array}{ll} 1/2 & \text{si } i = j, \\ 1/4 & \text{si } i \neq j \end{array} \right.$$

et posons pour tout $\omega \in \Omega$:

$$Y_0(\omega) = \varphi(U_0(\omega)) \qquad \text{et} \qquad \forall\, n \in \mathbb{N}, \quad Y_{n+1}(\omega) = f\big(U_{n+1}(\omega),\, Y_n(\omega)\big).$$

4.2 Dans ces conditions, quels que soient x_0, …, x_n dans E,

$$\begin{aligned}[Y_0 = x_0, Y_1 = x_1, \dots, Y_{n-1} = x_{n-1}, Y_n = x_n] \\ = [\varphi(U_0) = x_0, f(U_1, Y_0) = x_1, \dots, f(U_{n-1}, Y_{n-2}) = x_{n-1}, f(U_n, Y_{n-1}) = x_n] \\ = [\varphi(U_0) = x_0, f(U_1, x_0) = x_1, \dots, f(U_{n-1}, x_{n-2}) = x_{n-1}, f(U_n, x_{n-1}) = x_n].\end{aligned}$$

Comme les variables aléatoires U_0, …, U_n sont indépendantes, on déduit du lemme des coalitions que les variables aléatoires $\varphi(U_0)$, $f(U_1, x_0)$, …, $f(U_{n-1}, x_{n-2})$, $f(U_n, x_{n-1})$ sont indépendantes. De plus, on a supposé que toutes les variables aléatoires U_k suivaient la même loi que U et donc conséquent,

$$\begin{aligned}&\mathbf{P}(Y_0 = x_0, Y_1 = x_1, \dots, Y_{n-1} = x_{n-1}, Y_n = x_n) \\ &= \mathbf{P}\big(\varphi(U_0) = x_0\big)\,\mathbf{P}\big(f(U_1, x_0) = x_1\big) \cdots \mathbf{P}\big(f(U_{n-1}, x_{n-2}) = x_{n-1}\big)\,\mathbf{P}\big(f(U_n, x_{n-1}) = x_n\big) \\ &= \mathbf{P}\big(\varphi(U) = x_0\big)\,\mathbf{P}\big(f(U, x_0) = x_1\big) \cdots \mathbf{P}\big(f(U, x_{n-2}) = x_{n-1}\big)\,\mathbf{P}\big(f(U, x_{n-1}) = x_n\big) \\ &= \mathbf{P}(X_0 = x_0)\,\mathbf{P}(X_1 = x_1 \mid X_0 = x_0) \cdots \mathbf{P}(X_{n-1} = x_{n-1} \mid X_{n-2} = x_{n-2})\,\mathbf{P}(X_n = x_n \mid X_{n-1} = x_{n-1}) \\ &= \mathbf{P}(X_0 = x_0, X_1 = x_1, \dots, X_{n-1} = x_{n-1}, X_n = x_n).\end{aligned}$$

4.3 Le processus $(Y_n)_{n\in\mathbb{N}}$ qu'on a ainsi construit au moyen du processus standard $(U_n)_{n\in\mathbb{N}}$ et des fonctions f et φ a donc exactement la même loi (en tant que processus aléatoire) que le processus $(X_n)_{n\in\mathbb{N}}$.
Non seulement cette construction valide notre modélisation (en prouvant la cohérence de nos hypothèses), mais elle nous donne aussi un moyen concret de simuler le processus aléatoire au moyen du générateur pseudo-aléatoire de Python !

II

Simulation informatique

Q 1. *Pour simplifier, on remplace $E = \{A, B, C\}$ par $E = \{0, 1, 2\}$.*
Q 1.a *Proposer un code pour la fonction φ définie au* **[4.1]**.
Q 1.b *Proposer un code pour la fonction f définie au* **[4.1]**.

5. Un générateur de nombres pseudo-aléatoires comme la fonction `numpy.random.random` est censé fournir des listes de nombres dont le comportement imite celui d'une famille de variables aléatoires indépendantes qui suivent la loi uniforme sur $[0, 1]$.
En effectuant s appels à `numpy.random.random(n+1)`, on obtient des réels qu'on peut interpréter comme s débuts de trajectoire du processus aléatoire $(U_n)_{n\in\mathbb{N}}$.

$$\begin{array}{l}\big(U_0(\omega_0), U_1(\omega_0), \ldots, U_n(\omega_0)\big)\\ \big(U_0(\omega_1), U_1(\omega_1), \ldots, U_n(\omega_1)\big)\ \ \ldots\\ \big(U_0(\omega_{s-1}), U_1(\omega_{s-1}), \ldots, U_n(\omega_{s-1})\big)\end{array}$$

Q 2. *On considère le code suivant.*

```
import numpy as np

def U():
    return np.random.random()
```

Utiliser les fonctions `phi`, `f` *et* `U()` *pour définir une fonction* `trajectoire(n)` *dont l'exécution retourne une liste*

$$(x_0, x_1, \ldots, x_n)$$

qu'on puisse interpréter comme une réalisation des n premiers pas de la marche aléatoire :

$$\exists\, \omega \in \Omega, \quad (x_0, x_1, \ldots, x_n) = \big(Y_0(\omega), Y_1(\omega), \ldots, Y_n(\omega)\big).$$

Q 3. Nombre de passages
On considère une liste `traj` *qui contient des positions occupées successivement par le pion au cours de sa marche aléatoire (cette liste a pu être produite par un appel à la fonction* `trajectoire`*).*
On souhaite que l'exécution de

```
nb_passages(traj, pos)
```

compte le nombre d'occurrences de la position `pos` *dans la liste* `traj`*. Proposer un code pour la fonction* `nb_passages`.

Q 4. Temps de retour

On veut maintenant simuler le temps de retour en un point $i \in E$. Le début de trajectoire

$$\big(Y_0(\omega), Y_1(\omega), \dots, Y_n(\omega)\big)$$

étant connu (par exemple au moyen de la fonction `trajectoire`*), on pose $T(\omega) = 0$ si aucun des $Y_k(\omega)$ n'est égal à i et $T(\omega) = k_0$ si k_0 est la plus petite valeur de $k \geqslant 1$ pour laquelle $Y_k(\omega) = 1$.*

Q 4.a *Proposer un code pour* `premier_passage(traj, i)` *qui reproduise le fonctionnement de T.*

Q 4.b *Proposer un code qui n'utilise qu'une seule instruction* `return`*.*

Q 5. Distribution asymptotique

On souhaite étudier le comportement asymptotique de la marche aléatoire au moyen de notre simulation. Pour cela, on choisit un entier n_0 assez grand et on cherche à évaluer la loi de X_{n_0}, c'est-à-dire à estimer les probabilités

$$\mathsf{P}(X_{n_0} = 0), \qquad \mathsf{P}(X_{n_0} = 1), \qquad \mathsf{P}(X_{n_0} = 2).$$

Comment procéder ?

Réponses aux questions

II Simulation informatique

R 1.a D'après **[2.1]**, la variable aléatoire X_0 est p.s. constante, égale à 0. Le code suivant convient donc !

```
def phi(u):
    return 0
```

R 1.b Comme U suit la loi uniforme sur $[0,1]$, on a

$$\mathsf{P}(U < 1/4) = 1/4, \qquad \mathsf{P}(1/4 \leqslant U \leqslant 3/4) = 1/2, \quad \mathsf{P}(3/4 < U) = 1/4.$$

On peut donc coder f de la manière suivante.

```
def f(u, i):
    if (u<0.25):
        j = (i-1)%3  # une chance sur quatre d'aller à gauche
    elif (u>0.75):
        j = (i+1)%3  # une chance sur quatre d'aller à droite
    else:
        j = i        # et donc une chance sur deux de ne pas bouger
    return j
```

R 2. On initialise la trajectoire avec la fonction `phi` et on itère n fois de suite avec la fonction `f`.

```
def trajectoire(durée):
    Y = phi(U())
    traj = [ Y ]
    for i in range(durée):
        Y = f(U(), Y)
        traj.append(Y)
    return traj
```

R 3. Nombre de passages

Une première solution, simple et directe : on parcourt la trajectoire et on compte...

```
def nb_passages(traj, pos):
    nb = 0
    for x in traj:
        if x==pos:
            nb += 1
    return nb
```

Une seconde solution, plus pythonnienne : on convertit la liste en tableau `numpy` ; on compare chaque élément de ce tableau à l'argument `pos` pour obtenir un tableau de booléens ; on calcule la somme de ce tableau, c'est-à-dire le nombre de booléens qui prennent la valeur `True`.

```
def nb_passages(traj, pos):
    return np.sum(np.array(traj)==pos)
```

Temps de retour

R 4.a Si on fait au plus simple, on parcourt la liste et on retourne l'indice k_0 dès qu'on rencontre la position voulue. Si on atteint la fin de la liste sans avoir rencontrée cette valeur, c'est qu'il faut retourner la valeur 0.

```
def premier_passage(traj, pos):
    for i, x in enumerate(traj):
        if (x==pos):
            return i
    return 0
```

R 4.b On parcourt la liste jusqu'à ce qu'on trouve la position cherchée ou qu'on atteigne la fin de la liste. La dernière valeur de l'indice permet de savoir si on a rencontré, ou non, la position cherchée.

```
def premier_passage_bis(traj, pos):
    lgr, i = len(traj), 0
    while ((i<lgr) and (traj[i]!=pos)):
        i += 1
    if (i<lgr):
        T = i
    else:
        T = 0
    return T
```

R 5. Distribution asymptotique

On réalise un très grand nombre N_s de simulations et, pour chacune d'elles, on note la valeur de X_{n_0}. D'après le fonctionnement du générateur de nombres pseudo-aléatoires, cela revient à définir un échantillon

$$(Z_0, Z_1, \ldots, Z_{N_s-1})$$

de N_s variables aléatoires indépendantes et de même loi que X_{n_0}.
On peut alors appliquer la Loi des grands nombres aux variables aléatoires

$$(\mathbb{1}_{[Z_0=i]}, \mathbb{1}_{[Z_1=i]}, \ldots, \mathbb{1}_{[Z_{N_s-1}=i]})$$

qui sont des variables aléatoires indépendantes (lemmes des coalitions !) suivant la loi de Bernoulli de paramètre $p_i = \mathsf{P}(X_{n_0} = i)$. Dans ce cas, la Loi des grands nombres dit que

$$\frac{1}{N_s} \sum_{0 \leqslant k < N_s} \mathbb{1}_{[Z_k=i]} \approx \mathsf{E}(\mathbb{1}_{[Z_0=i]}) = \mathsf{P}(X_{n_0} = i).$$

Cela est tout simple à coder.

```
n0, Nb_sim = 100, 10000
echantillon_Z = [ trajectoire(n0)[-1] for k in range(Nb_sim) ]
loi_Xn0 = [ nb_passages(echantillon_Z, i)/Nb_sim for i in range(3) ]
```

La théorie dit que, si l'indice n_0 est assez grand, la variable aléatoire X_{n_0} suit à peu près la loi uniforme sur $E = \{0, 1, 2\}$. C'est plutôt bien confirmé par notre simulation.