Problème de Mathématiques

Référence pp2127 — Version du 31 décembre 2025

Une puce se déplace sur $\mathbb{N}$. Initialement, elle est posée sur 0. Par la suite, elle se déplace selon la règle suivante : si elle se trouve sur le point d'abscisse $(k-1) \in \mathbb{N}$, alors

— elle saute vers le point d'abscisse k avec la probabilité $\frac{k}{k+1}$;

— ou elle saute vers 0 avec la probabilité $\frac{1}{k+1}$.

Le mouvement aléatoire de cette puce est modélisé par une suite de variables aléatoires $(A_n)_{n\in\mathbb{N}}$ à valeurs dans $\mathbb{N}$ définies sur un espace probabilisé $(\Omega, \mathcal{A}, \mathbf{P})$.

On note U, l'instant aléatoire (supérieur à 1) auquel la puce revient pour la première fois en 0. Si la puce ne revient jamais en 0, alors on convient que $U = 0$.

1. Démontrer que U est une variable aléatoire discrète à valeurs dans $\mathbb{N}$.

On suppose que la suite $(A_n)_{n\in\mathbb{N}}$ est une chaîne de Markov homogène au sens où il existe une application

$$f : \mathbb{N} \times [0,1] \to \mathbb{N}$$

et une suite de variables aléatoires indépendantes $(X_n)_{n\in\mathbb{N}}$ définies sur $(\Omega, \mathcal{A}, \mathbf{P})$ qui suivent toutes la loi uniforme sur $[0,1]$:

$$\forall\, 0 \leqslant a < b \leqslant 1, \quad \mathbf{P}(a \leqslant X \leqslant b) = b - a$$

telles que

$$\forall\, n \in \mathbb{N}, \quad A_{n+1} = f(A_n, X_n).$$

Dans ces conditions, on peut démontrer que

$$\forall\, k \in \mathbb{N}^*, \quad \mathbf{P}(U = k) = \frac{1}{k(k+1)}.$$

2. Donner une expression pour f qui soit cohérente avec la description du mouvement de la puce.

3. Calculer $\mathbf{P}(U = 0)$ et $\mathbf{P}(U > n)$ pour tout $n \in \mathbb{N}$.

4. On cherche à simuler informatiquement ce processus aléatoire.

```
import numpy  as np
import random as rd
```

4. a. Expliquer le comportement de la fonction `suivant`.

```
def suivant(An):
  p = 1/(An + 2)
  if (rd.random()<p):
    return 0
  else:
    return An + 1
```

4. b. Expliquer le comportement de la fonction `trajectoire`.

```
def trajectoire(lgr):
  T = [ 0 ] # départ de l'origine
  for i in range(lgr):
    Ai = T[i]
    T.append(suivant(Ai))
  return T
```

4. c. Expliquer le comportement de la fonction `echantillon_T`.

```
def echantillon_T(N, lgr):
  ech = [ trajectoire(lgr) for k in range(N) ]
  return ech
```

4. d. Expliquer le comportement de la fonction `echantillon_U`.

```
def U_tronquee(T):
  lgr, U = len(T), 1
  while (U<lgr) and (T[U]>0):
    U += 1
  return U

def echantillon_U(N, lgr):
  ech_T = echantillon_T(N, lgr)
  ech_U = [ U_tronquee(T) for T in ech_T ]
  return ech_U
```

4. e. Expliquer le comportement des fonctions suivantes.

```
def sans_retour(T):
  return T[-1]==(len(T)-1)

def estimation_sans_retour(N, lgr):
  ech_T = echantillon_T(N, lgr)
  nb_echecs = 0
  for T in ech_T:
    nb_echecs += sans_retour(T)
  return nb_echecs/N

def erreur_estimation_sans_retour(N, lgr):
  pr_th  = 1/(lgr+1)
  pr_emp = estimation_sans_retour(N, lgr)
  return 100*abs(pr_th-pr_emp)/pr_th
```

5. Proposer un code python pour vérifier que la simulation de U fournie par `echantillon_U` est conforme à la loi de U indiquée plus haut.

Solution ❀ Chaîne de Markov

1. Par construction, U est une application de Ω dans $\mathbb{N}$.
Par convention,

$$U(\omega) = 0 \iff \forall\, n \geqslant 1, \quad A_n(\omega) \neq 0$$

donc

$$[U = 0] = \bigcap_{n\in\mathbb{N}^*} [A_n = 0]^c \in \mathcal{A}.$$

(Comme A_n est une variable aléatoire, alors $[A_n = 0] \in \mathcal{A}$ et, comme toutes les tribus, $\mathcal{A}$ est stable par passage au complémentaire et par intersection dénombrable.)

Pour tout entier $n \geqslant 1$,

$$[U = n] = [A_1 \neq 0, \dots, A_{n-1} \neq 0] \cap [A_n = 0] \in \mathcal{A}$$

(puisque $\mathcal{A}$ est stable par intersection finie).

On a démontré que

$$\forall\, n \in \mathbb{N}, \quad [U = n] \in \mathcal{A}$$

donc $U : \Omega \to \mathbb{N}$ est bien une variable aléatoire discrète sur $(\Omega, \mathcal{A})$.

2. Pour $k \in \mathbb{N}$ et $u \in [0,1]$, on pose $p = \frac{1}{k+2}$ et

$$f(k,u) = \left| \begin{array}{ll} 0 & \text{si } 0 \leqslant u < p, \\ k+1 & \text{si } p \leqslant u \leqslant 1. \end{array} \right.$$

❧ On rappelle que les variables aléatoires $X_1, \dots, X_n, X_{n+1}$ sont indépendantes.
Par hypothèse,

$$\begin{aligned} A_n &= f(A_{n-1}, X_{n-1}) \\ &= f\big(f(A_{n-2}, X_{n-2}), X_{n-1}\big) \\ &= f\big(f\big(f(A_{n-3}, X_{n-3}), X_{n-2}\big), X_{n-1}\big) = \cdots \\ &= g(X_1, \dots, X_{n-1}) \end{aligned}$$

et d'après le lemme des coalitions, les variables aléatoires A_n et X_n sont indépendantes.

❧ Par conséquent, en supposant que $\mathbf{P}(A_n = k-1) > 0$,

$$\begin{aligned} \mathbf{P}(A_{n+1} = k \mid A_n = k-1) &= \frac{\mathbf{P}\big([f(A_n, X_n) = k] \cap [A_n = k-1]\big)}{\mathbf{P}(A_n = k-1)} \\ &= \frac{\mathbf{P}\big([f(k-1, X_n) = k] \cap [A_n = k-1]\big)}{\mathbf{P}(A_n = k-1)} \\ &= \mathbf{P}\big(f(k-1, X_n) = k\big). \qquad \text{(indépendance de } A_n \text{ et de } X_n) \end{aligned}$$

Or, par construction de f,

$$\forall\, k \geqslant 1, \quad \big[f(k-1, X_n) = k\big] = \Big[\frac{1}{k+1} \leqslant X_n \leqslant 1\Big]$$

et, par hypothèse sur X_n,

$$\mathbf{P}\Big(\frac{1}{k+1} \leqslant X_n \leqslant 1\Big) = 1 - \frac{1}{k+1} = \frac{k}{k+1}.$$

❧ De même,

$$\begin{aligned} \mathbf{P}(A_{n+1} = 0 \mid A_n = k-1) &= \frac{\mathbf{P}\big([f(A_n, X_n) = 0] \cap [A_n = k-1]\big)}{\mathbf{P}(A_n = k-1)} \\ &= \frac{\mathbf{P}\big([f(k-1, X_n) = 0] \cap [A_n = k-1]\big)}{\mathbf{P}(A_n = k-1)} \\ &= \mathbf{P}\big(f(k-1, X_n) = 0\big). \qquad \text{(indépendance de } A_n \text{ et de } X_n) \end{aligned}$$

Or, par construction de f,

$$\forall\, k \geqslant 1, \quad \big[f(k-1, X_n) = 0\big] = \Big[0 \leqslant X_n < \frac{1}{k+1}\Big]$$

et, par hypothèse sur X_n,

$$\mathbf{P}\Big(0 \leqslant X_n < \frac{1}{k+1}\Big) = \frac{1}{k+1}.$$

❧ On a donc bien

$$\mathbf{P}(A_{n+1} = k \mid A_n = k-1) = \frac{k}{k+1} \quad \text{et} \quad \mathbf{P}(A_{n+1} = 0 \mid A_n = k-1) = \frac{1}{k+1},$$

conformément à l'énoncé.

3. Comme U est une variable aléatoire à valeurs dans $\mathbb{N}$,

$$[U = 0] = [U \geqslant 1]^c = \bigsqcup_{k \geqslant 1} [U = k]$$

et par σ-additivité de $\mathbf{P}$,

$$\mathbf{P}(U = 0) = 1 - \sum_{k=1}^{+\infty} \mathbf{P}(U = k) = 1 - \sum_{k=1}^{+\infty} \Big(\frac{1}{k} - \frac{1}{k+1}\Big) = 0$$

(par télescopage archi-classique).

❧ De manière analogue,

$$[U > n] = \bigsqcup_{k > n} [U = k]$$

et donc, toujours par σ-additivité et télescopage,

$$\mathbf{P}(U > n) = \sum_{k=n+1}^{+\infty} \frac{1}{k(k+1)} = \frac{1}{n+1}.$$

REMARQUE.— On retrouve ainsi que $\mathbf{P}(U > 0) = 1$ et donc que $\mathbf{P}(U = 0) = 0$.

4. a. L'argument `An` représente la position occupée à l'instant n. Soit $(k-1) \in \mathbb{N}$, la valeur de cet entier. On pose alors $p = \frac{1}{(k-1)+2} = \frac{1}{k+1}$.

L'exécution de `random()` revient à simuler une variable aléatoire X suivant la loi uniforme sur $[0,1]$:

— Si $X(\omega) < p$, ce qui a lieu avec probabilité $\mathbf{P}(0 \leqslant X < p) = p$, alors la fonction renvoie $0 = f\big(k-1, X(\omega)\big)$;

— Sinon, on a $p \leqslant X(\omega) \leqslant 1$, ce qui a lieu avec probabilité $\mathbf{P}(p \leqslant X \leqslant 1) = 1-p$, alors la fonction renvoie $(k-1)+1 = k = f\big(k-1, X(\omega)\big)$.

En admettant que le générateur aléatoire fonctionne correctement, les appels successifs à la fonction `random()` simulent fidèlement un échantillon $(X_n)_{n \geqslant 0}$ de variables aléatoires indépendantes et de loi uniforme sur $[0,1]$.

Dans ces conditions, les appels successifs à la fonction `suivant(Ai)` simulent fidèlement un échantillon $(A_n)_{n \geqslant 0}$ de la chaîne de Markov étudiée.

4. b. On effectue ici `lgr` appels successifs à la fonction `suivant`. La liste renvoyée sera donc

$$T = (T_0 = 0, T_1, \ldots, T_{lgr}).$$

Au début de la i-ième itération, la liste T ne contient que $(T_0, \ldots, T_i)$ et on considère que T_i est la valeur de A_i. On simule alors A_{i+1} en exécutant `suivant(Ai)` qui devient alors $T_{i=1}$.

On peut donc considérer que la fonction `trajectoire` renvoie une liste

$$\big(0, A_1(\omega), A_2(\omega), \ldots, A_{lgr}(\omega)\big) \in \mathbb{N}^{lgr+1}$$

qui représente le début d'une trajectoire de la marche aléatoire $(A_n)_{n \in \mathbb{N}}$.

4. c. La fonction `echantillon_T(N, lgr)` calcule une liste de N trajectoires de la marche aléatoire $(A_n)_{n \in \mathbb{N}}$ (plus précisément : N début de trajectoires, entre l'instant initial 0 et l'instant `lgr` inclus).

4. d. La fonction `echantillon_U` calcule une liste `ech_T` de N débuts de trajectoires.

Sur chacune de ces trajectoires, elle appelle la fonction `U_tronquee`.

Cette fonction `U_tronquee` parcourt un début de trajectoire, en commençant à l'instant 1 (c'est-à-dire en ignorant la position initiale, qui est toujours égale à 0). Tant que la trajectoire ne revient pas à l'origine (soit : tant que $T_U > 0$), on continue.

Deux cas se présentent :

— Si la trajectoire repasse par l'origine, c'est la valeur du premier instant pour lequel on revient à l'origine qui est renvoyée, c'est-à-dire la valeur de U ;
— Si la trajectoire ne repasse pas par l'origine, c'est la longueur de la trajectoire qui est renvoyée. (NB : Dans cette fonction, la variable `lgr` est la longueur de la liste `T` contrairement à la fonction `trajectoire` où la longueur de la liste renvoyée était égale à `lgr`+1.)

L'échantillon renvoyé par cette fonction est donc *presque* un échantillon de valeurs de la variable aléatoire U : c'est un échantillon de la variable aléatoire $\min\{U, n\}$ calculé sur des trajectoires $T = (T_0, \dots, T_{n-1})$.

4. e. La fonction `sans_retour` renvoie un booléen. Ce booléen prend la valeur `True` si, et seulement si, le dernier élément de la liste `T` (c'est-à-dire la position finale de la trajectoire) est égal à $n-1$.

On distingue deux cas pour une trajectoire $T = (T_0, \dots, T_{n-1})$:
— Si on n'est pas encore revenu à l'origine, la valeur finale est égale à $(n-1)$ (où n est la longueur de la liste `T`) ;
— Si on est déjà revenu à l'origine, on est reparti de 0 à un instant postérieur à $i = 0$, donc la valeur finale est strictement inférieure à $(n-1)$.

Par conséquent, le booléen renvoyé par `sans_retour` est égal à `True` si, et seulement si, au cours du début de trajectoire `T` la marche aléatoire n'est pas encore repassée par l'origine, c'est-à-dire si $U \geqslant n$ (en notant n la longueur de la liste `T`).

❧ La fonction `estimation_sans_retour` calcule un nombre N de trajectoires et pour chacune de ces trajectoires elle vérifie si un retour à l'origine a eu lieu. Elle compte le *nombre* de trajectoires pour lesquelles il n'y a pas eu de retour à l'origine (avec la convention `True` vaut 1 et `False` vaut 0) avant de renvoyer la *proportion* de ces trajectoires (en divisant par le nombre total N de trajectoires étudiées).

❧ On a ainsi estimé la fréquence de réalisation de l'événement $[U \geqslant n] = [U > \text{lgr}]$ (la longueur n des listes est ici égale à $\text{lgr} + 1$).

L'étude théorique nous dit que la probabilité de cet événement est égale à $1/n$.

La simulation nous donne une fréquence empirique (calculée sur les résultats constatés).

On renvoie alors l'erreur relative entre la probabilité (théorique) et la fréquence (empirique) exprimée en pourcentage.

❧ Concrètement, pour $N = 10^4$ et $\text{lgr} = 10$, l'erreur relative est de l'ordre de 1 à 2%. Elle est un peu plus petite pour $N = 10^5$ (sans être significativement plus petite).

5. On tronque la variable U pour la traiter informatiquement (bien obligé !).

La loi de U est représentée par un tableau qui contient les probabilités théoriques.

```
def loi_retour(lgr):
  d = np.zeros(lgr)
  for k in range(1, lgr):
    d[k] = 1/(k*(k+1))
  return d
```

❧ On simule ensuite un échantillon de N valeurs de la variable U tronquée et on compte la fréquence de chacune des valeurs observées (méthode du *tri postal*).

```
def estimation_loi_retour(N, lgr):
  ech_U = echantillon_U(N, lgr)
  dist_U = np.zeros(lgr)
  for U in ech_U:
    if (U<lgr):
      dist_U[U] += 1
  return dist_U/N
```

❧ Pour comparer le tableau théorique (les probabilités) et le tableau empirique (les fréquences),

```
loi_th = loi_retour(10)
freq_emp = estimation_loi_retour(10000, 10)
```

on peut par exemple calculer l'écart absolu moyen.

```
np.abs(loi_th-freq_emp).max()
```

Le plus important est de trouver une valeur de référence qui puisse donner un sens au résultat ! On peut par exemple rapporter cet écart à la probabilité $\mathbf{P}(U \leqslant n)$.

```
loi_th.sum()
```

Valeurs observées			
N	10^3	10^4	10^5
écart	2.10^{-2}	5.10^{-3}	10^{-3}

NB : ces valeurs sont soumises à des fluctuations d'échantillonnage, il ne faut pas s'étonner de trouver des valeurs différentes (mais l'ordre de grandeur de ces valeurs devrait peu varier).