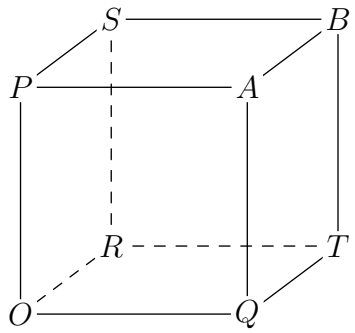
1
Calcul approché de probabilités

I
Une chaîne de Markov



1. Le problème

Une chenille se déplace sur les sommets d'un cube. De chaque sommet, elle emprunte au hasard l'une des trois arêtes qui partent de ce sommet avec la même probabilité. Mais les sommets A et B sont enduits de colle forte et la chenille ne redémarre plus...

On suppose que la chenille part du sommet O. Quelle est la probabilité pour qu'elle se trouve collée en A ?

2. Le modèle

On modélise la marche au hasard de la chenille par une suite $(X_n)_{n\in\mathbb{N}}$ de variables aléatoires discrètes définies sur un espace probabilisé $(\Omega, \mathcal{A}, \mathbf{P})$ à valeurs dans l'ensemble

$$\mathscr{C} = \{A, B, O, P, Q, R, S, T\}$$

des sommets du cube : la variable aléatoire X_0 décrit la position initiale de la chenille et, pour tout entier $n \geqslant 1$, la variable aléatoire X_n décrit la position de la chenille à l'instant n.

2.1 Il s'agit de calculer, pour tout sommet $M \in \mathscr{C}$, la probabilité conditionnelle

$$\mathbf{P}\Big(\bigcup_{n\in\mathbb{N}} [X_n = A] \mid X_0 = M\Big).$$

2.2 L'énoncé impose que

$$\forall\, n \in \mathbb{N}, \quad \mathbf{P}(X_{n+1} = A \mid X_n = A) = \mathbf{P}(X_{n+1} = B \mid X_n = B) = 1.$$

Cela signifie que le processus de déplacement évolue aléatoirement tant que la variable X_n ne prend ni la valeur A, ni la valeur B.

Ces deux **états absorbants** définissent donc la condition d'arrêt du processus aléatoire.

2.3 L'énoncé impose d'autre part, en notant $\mathcal{V}(M)$, l'ensemble des (trois) sommets voisins du sommet M, que

$$\forall\, M \in \{O, P, Q, R, S, T\},\ \forall\, N \in \mathcal{V}(M), \quad \mathbf{P}(X_{n+1} = N \mid X_n = M) = \frac{1}{3}$$

(en admettant que toutes ces probabilités conditionnelles aient un sens).

2.4 Le générateur pseudo-aléatoire permet, avec la fonction `random()` du module `random`, de simuler une famille $(U_n)_{0\leqslant n<N}$ de variables aléatoires indépendantes et de loi uniforme sur $]0, 1]$. En pratique,

$$\forall\, n \in \mathbb{N}, \quad \mathbf{P}(U_n < 1/3) = \mathbf{P}(U_n > 2/3) = \mathbf{P}(1/3 \leqslant U_n \leqslant 2/3) = \frac{1}{3}.$$

2.5 On considère la fonction g définie par morceaux de la manière suivante :

$$g(x) = \left| \begin{array}{ll} 0 & \text{si } x < 1/3, \\ 1 & \text{si } 1/3 \leqslant x \leqslant 2/3, \\ 2 & \text{si } 2/3 < x. \end{array} \right.$$

et les variables aléatoires discrètes V_n définies par

$$\forall\, n \in \mathbb{N}, \quad V_n = g(U_n).$$

Les variables aléatoires $(V_n)_{n\in\mathbb{N}}$ constituent une famille de variables aléatoires indépendantes qui suivent toutes la loi uniforme sur $\{0,1,2\}$.

2.6 En supposant connue une fonction $f : \mathscr{C} \times \{0,1,2\} \to \mathscr{C}$ telle que, pour tout $M \in \mathscr{C} \setminus \{A,B\}$, la variable aléatoire $f(M,V_n)$ suive la loi uniforme sur l'ensemble $\mathcal{V}(M)$, on peut considérer que les variables $(X_n)_{n\in\mathbb{N}}$ sont liées par la relation

$$\forall\, n \in \mathbb{N}, \qquad X_{n+1} = f(X_n, V_n).$$

3. Toute chaîne de Markov sur un ensemble fini d'états peut être simulée de manière analogue au moyen du générateur pseudo-aléatoire.

II

La méthode de Monte-Carlo

4. La **méthode de Monte-Carlo** consiste à effectuer un grand nombre de simulations (plusieurs milliers au moins) et relever à chaque fois le sommet finalement atteint (succès si on atteint A, échec si on atteint B). D'après la Loi des grands nombres, la proportion de succès est proche de la probabilité cherchée — d'autant plus proche que le nombre de simulations est grand.

5. Utilisation d'un dictionnaire

5.1 Un dictionnaire est un ensemble d'**entrées**. Chaque entrée est constituée d'un mot (la **clé**) et d'une définition de ce mot (la **valeur**).

On peut donc considérer un dictionnaire comme une sorte de liste dont les valeurs seraient repérées par des *clés* et non pas par des *indices*.

5.2 Le dictionnaire suivant tient ses promesses : à chaque sommet $M \in \mathscr{C}$ (la clé), il associe la liste des sommets accessibles depuis M (la valeur).

```
adjacences = {
  # deux états absorbants
  'A':['A'], 'B':['B'],
  # pour les autres sommets, les trois sommets voisins sont accessibles
  'O':['P', 'Q', 'R'], 'P':['A', 'O', 'S'], 'Q':['A', 'O', 'T'],
  'R':['O', 'S', 'T'], 'S':['B', 'P', 'R'], 'T':['B', 'Q', 'R'] }
```

Ainsi, on obtient par exemple la liste `['O', 'S', 'T']` des sommets accessibles depuis le sommet `'R'` avec la commande `adjacences['R']`.

5.3 On peut alors traduire la relation $X_{n+1} = f(X_n, V_n)$ par le code suivant.

```
def suivant(X):                  # L'argument X est égal à la valeur M de X_n
  sommets_possibles = adjacences[X]   # liste donnant V(M)
  if (len(sommets_possibles)==1):     # une seule possibilité si M ∈ {A,B}
    return sommets_possibles[0]
  else:                               # trois possibilités pour M ∉ {A,B}
    U = random.random()   # valeur de U_n
    if (U<1/3):           # une chance sur trois
      return sommets_possibles[0]
    elif (U>2/3):         # une chance sur trois
      return sommets_possibles[2]
    else:                 # une chance sur trois
      return sommets_possibles[1]
```

Q 1. *Pour chaque sommet $M \in \mathscr{C}$, on souhaite calculer une valeur approchée de la probabilité pour que la chenille atteigne le sommet A en partant du sommet M en effectuant N_s simulations.*

Q 1.a *Proposer une simplification de la fonction* `suivant(X)` *définie plus haut.*

Q 1.b *Définir une fonction* `point_d_arrivee(pt_de_depart)` *dont l'argument est un point $M \in \mathscr{C}$ et qui renvoie le point final (A ou B) atteint par la chenille au terme de son déplacement.*
On notera `X` *la variable initialement égale à l'argument* `pt_de_depart` *et dont les valeurs successives seront calculées à l'aide de la fonction* `suivant`.

Q 1.c *Définir une fonction* `estimation_proba(pt_de_depart, nb_simulations)` *dont le premier argument est un point $M \in \mathscr{C}$ et le second argument est un "grand" entier N_s.*
Cette fonction effectuera N_s appels à la fonction `point_d_arrivee` *et renverra la proportion des appels pour lesquels la valeur retournée est égale à* `'A'` *(et donc pas à* `'B'`*).*
On fera varier N_s entre 100 *et* 10 000.

Q 1.d *Définir une fonction* `verification(liste_departs, nb_simulations, nb_repetitions=10)` *dont le premier argument est une liste de sommets du cube $\mathscr{C}$; le second argument est un "grand" entier N_s et le troisième argument est un entier R, par défaut égal à* 10.
Pour chaque sommet appartenant à `liste_departs`*, il s'agit de calculer R valeurs approchées de la probabilité étudiée en effectuant R répétitions des N_s simulations. Les R valeurs approchées devront être affichées dans le terminal.*
On cherchera pour quelles valeurs de N_s les R valeurs approchées ainsi calculées donnent des résultats relativement proches les uns des autres.

6. On a admis comme une évidence que la chenille arrivait inévitablement sur l'un des deux sommets piégés, notamment en définissant la fonction `point_d_arrivee`.
On peut démontrer que la probabilité de cet événement est effectivement égale à 1, bien qu'on puisse imaginer que la chenille ne passe jamais par un des sommets piégés...

Q 2. *Nous allons maintenant étudier la durée de la marche aléatoire, c'est-à-dire à la valeur du plus petit indice $n \in \mathbb{N}$ tel que X_n prenne la valeur* `'A'` *ou* `'B'`.
On définit une nouvelle variable aléatoire T en posant

$$T(\omega) = \min\{n \in \mathbb{N} \,:\, X_n(\omega) \in \{A, B\}\}.$$

Comme on l'a indiqué plus haut, ce minimum n'est pas défini pour tout $\omega \in \Omega$ *mais pour* presque tout $\omega \in \Omega$ *seulement.*

Q 2.a *En s'inspirant de la fonction* `point_d_arrivee`*, définir une fonction* `duree_deplacement(pt_de_depart)` *dont l'argument est un point $M \in \mathscr{C}$ et qui renvoie le couple constitué du point final (A ou B) et de la durée T.*

Q 2.b *Définir une fonction* `duree_moyenne(pt_de_depart, nb_simulations)` *dont le premier argument est un point $M \in \mathscr{C}$ et le second argument est un grand entier N_s.*
En effectuant N_s appels à la fonction `duree_deplacement`*, cette fonction devra renvoyer la valeur moyenne de T (c'est-à-dire une estimation de son espérance), ainsi que la durée moyenne de T lorsque le point final est le point A (resp. le point B).*
On expliquera les valeurs ainsi obtenues.

III

Résolution littérale

7. Pour alléger les notations, nous noterons

$$[X_\infty = A] \quad \text{et} \quad [X_\infty = B]$$

au lieu de

$$\bigcup_{n\in\mathbb{N}} [X_n = A] \quad \text{et} \quad \bigcup_{n\in\mathbb{N}} [X_n = B].$$

On peut démontrer, et nous l'admettrons, que :

$$\mathbf{P}(X_\infty = A \mid X_0 = A) = 1, \qquad \mathbf{P}(X_\infty = A \mid X_0 = B) = 0,$$
$$\mathbf{P}(X_\infty = A \mid X_0 = O) = {}^8\!/_{14} \approx 57\%, \qquad \mathbf{P}(X_\infty = A \mid X_0 = R) = {}^6\!/_{14} \approx 43\%,$$
$$\mathbf{P}(X_\infty = A \mid X_0 = P) = {}^9\!/_{14} \approx 64\%, \qquad \mathbf{P}(X_\infty = A \mid X_0 = Q) = {}^9\!/_{14},$$
$$\mathbf{P}(X_\infty = A \mid X_0 = S) = {}^5\!/_{14} \approx 36\%, \qquad \mathbf{P}(X_\infty = A \mid X_0 = T) = {}^5\!/_{14}.$$

Q 3. *On cherche à estimer l'erreur commise en approchant ces probabilités par la méthode de Monte-Carlo.*
Q 3.a *Définir une fonction* `erreur_relative(pt_de_depart, nb_simulations)` *dont les arguments sont un sommet $M \in \mathscr{C}$ et le nombre N de simulations effectuées. Cette fonction doit renvoyer l'**erreur relative** commise en approchant la probabilité conditionnelle $\mathbf{P}(X_\infty = A \mid X_0 = M)$ par la méthode de Monte-Carlo.*
On pourra utiliser le dictionnaire suivant, qui rassemble les valeurs de référence.

```
Probas_exactes = { 'O':4/7, 'P':9/14, 'Q':9/14, 'R':3/7, 'S':5/14, 'T':5/14 }
```

Q 3.b *Pour différentes valeurs du nombre N_s de simulations effectuées, calculer et afficher l'erreur relative commise. On veillera à soigner la présentation des résultats.*

Réponses aux questions

II La méthode de Monte-Carlo

R 1.a

```
def suivant(X):
  # Le dictionnaire 'adjacences' nous donne les sommets qu'il est
  # possible d'atteindre en partant de la position X.
  sommets_possibles = adjacences[X]
  nb_sommets = len(sommets_possibles)
  # Le réel U est aléatoirement et uniformément réparti sur [0,1[.
  U = random.random()
  # Astuce : L'indice i est aléatoirement et uniformément réparti
  # entre 0 et (n-1) [où n est le nombre de sommets possibles].
  i = int(nb_sommets*U)
  sommet_choisi = sommets_possibles[i]
  return sommet_choisi
```

R 1.b Tant qu'elle n'a pas atteint un état absorbant, la chenille redémarre...

```
def point_d_arrivee(pt_de_depart):
  X = pt_de_depart
  while (X!='A') and (X!='B'):
    X = suivant(X)
  return X
```

Les valeurs renvoyées par la fonction sont `'A'` ou `'B'`.

R 1.c En Python, le booléen `True` est identifié à 1 et le booléen `False` est identifié à 0. Cela permet de simplifier le calcul du nombre de succès.

```
def estimation_proba(pt_de_depart, nb_simulations):
  nb_succes = 0
  for i in range(nb_simulations):
    X = point_d_arrivee(pt_de_depart)
    nb_succes += (X=='A')
  return nb_succes/nb_simulations
```

R 1.d

```
def verification(liste_departs, nb_simulations, nb_repetitions=10):
  # On prépare l'affichage des résultats.
  print("Nombre de simulations : {:>7}".format(nb_simulations))
  en_tete = "Sommet de départ {}"
  resultat = " {:6.2f}% "
  for M in liste_departs:
    liste_resultats = []
    print(en_tete.format(M))
    for i in range(nb_repetitions):
      p = estimation_proba(M, nb_simulations)
      liste_resultats.append(resultat.format(100*p))
    print("-".join(liste_resultats))
```

Pour chaque sommet, les valeurs approchées lors des différentes répétitions sont très proches les unes des autres lorsque le nombre N_s de simulations est de l'ordre de 10^4.

	Proportions calculées lors des 10 répétitions									
Sommet de départ	1	2	3	4	5	6	7	8	9	10
A	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%	100,0%
B	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%	0,0%
O	57,2%	57,1%	56,1%	58,0%	57,1%	57,8%	57,3%	57,7%	57,1%	57,6%
P	64,9%	64,6%	63,9%	64,3%	65,1%	64,2%	63,5%	65,3%	64,9%	64,5%
Q	64,3%	64,1%	64,2%	65,3%	63,8%	64,1%	64,4%	64,7%	64,3%	63,9%
R	42,2%	43,5%	42,9%	42,4%	43,0%	43,2%	41,9%	43,0%	42,7%	42,2%
S	36,2%	35,7%	36,5%	35,8%	35,7%	35,5%	35,5%	35,5%	35,8%	36,1%
T	35,9%	36,0%	35,6%	35,5%	34,9%	36,1%	36,4%	36,7%	35,4%	35,5%

Estimations de la probabilité d'atteindre le sommet A depuis un sommet quelconque ($N_s = 10^4$)

R 2.a Il suffit d'introduire un compteur, incrémenté à chaque itération, pour savoir combien de déplacements ont lieu avant d'atteindre la position finale.

```
def duree_deplacement(pt_de_depart):
  X, T = pt_de_depart, 0
  while (X!='A') and (X!='B'):
    # Tant qu'elle n'a pas atteint un état absorbant, la chenille redémarre...
    X = suivant(X)
    T += 1
  return X, T
```

R 2.b L'utilisation d'un dictionnaire pour enregistrer les durées de déplacements en fonction du point d'arrivée est très pratique.
Il est commode de définir une fonction auxiliaire pour calculer la durée moyenne et afficher le résultat pour chacun des trois cas envisagés : arrivée au point A, arrivée au point B et cas général.

```
def afficher_moyenne(etiquette, liste_durees):
  moy = sum(liste_durees)/len(liste_durees)
  s = etiquette + "{:6.4f}".format(moy)
  print(s)

def duree_moyenne(pt_de_depart, nb_simulations):
  durees = { 'A':[], 'B':[] }
  for i in range(nb_simulations):
    X, T = duree_deplacement(pt_de_depart)
    durees[X].append(T)
  print("Durées moyennes de déplacement partant de "+pt_de_depart)
  afficher_moyenne('arrivant en A : ', durees['A'])
  afficher_moyenne('arrivant en B : ', durees['B'])
  afficher_moyenne('globale       : ', durees['A']+durees['B'])
```

Évidemment, il ne faut pas utiliser la fonction `duree_moyenne` en partant du sommet A ou du sommet B (division par zéro assurée !), mais comme la durée de parcours est évidente dans ces deux cas, ce n'est pas vraiment gênant.

	Sommet de départ					
	O	P	Q	R	S	T
Arrivée en A	5,47	3,72	3,71	6,70	5,96	5,95
Arrivée en B	6,70	5,90	5,92	5,47	3,66	3,70
Total	5,99	4,49	4,51	6,00	4,48	4,51

Durée moyenne de la marche aléatoire en fonction du point de départ ($N_s = 5.10^4$)

III Résolution littérale

R 3.a L'**erreur relative** commise en approchant la valeur exacte $x_0 \neq 0$ par la valeur approchée x est par définition égale à

$$\varepsilon = \frac{|x - x_0|}{|x_0|}.$$

Cette erreur est *sans dimension*.
Le numérateur $|x - x_0|$ est l'**erreur absolue** (beaucoup moins significative).

```
Probas_exactes = { 'O':4/7, 'P':9/14, 'Q':9/14, 'R':3/7, 'S':5/14, 'T':5/14 }

def erreur_relative(pt_de_depart, nb_simulations):
  probabilite = Probas_exactes[pt_de_depart]
  estimation = estimation_proba(pt_de_depart, nb_simulations)
  erreur_absolue = abs(probabilite - estimation)
  return erreur_absolue/probabilite
```

R 3.b

```
cube = [ 'A', 'B', 'O', 'P', 'Q', 'R', 'S', 'T' ]

en_tete = "Sommet initial : {}"
resultat = "Nb de simulations : {:>6} - Erreur relative : {:6.2f}% "
for M in cube[2:]:
  print(en_tete.format(M))
  for Nb_Sim in [ 100, 1000, 5000, 10000, 50000 ]:
    erreur = erreur_relative(M, Nb_Sim)
    print(resultat.format(Nb_Sim, 100*erreur))
```

On constate qu'une approximation par la méthode de Monte-Carlo donne des résultats approchés d'une qualité correcte (suffisante en pratique), que la précision de ces résultats a tendance à s'améliorer lorsque le nombre de simulations augmente mais aussi que cette précision ne décroît pas forcément en fonction du nombre de simulations...

Nombre de simulations	Sommet de départ					
	O	P	Q	R	S	T
100	9,00%	9,78%	5,78%	12,00%	16,00%	20,40%
1000	0,25%	2,98%	0,18%	3,83%	1,16%	6,48%
10000	1,67%	0,26%	0,38%	0,92%	0,18%	1,83%

Erreur relative sur l'estimation de la probabilité d'atteindre A