

# Algorithmes et Pensée Computationnelle

## Spatial Algorithms

Le but de cette séance est de comprendre les algorithmes probabilistes. Ceux-ci permettent de résoudre des problèmes complexes de en relativement peu de temps. La contrepartie est que le résultat obtenu est généralement une solution approximée du problème initial. Ils demeurent néanmoins très utile pour beaucoup d'application.

## 1 Monte-Carlo

### Question 1: (🕒 10 minutes) Un jeu de hasard : Python

Supposez que vous lanciez une pièce de monnaie  $l$  fois et que vous voulez calculez la probabilité d'avoir un certains nombre de pile. Vous devez programmer un algorithme probabiliste, permettant de calculer cette probabilité. Pour ce faire, vous devez compléter la fonction `proba(n,l,iter)` contenue dans le fichier `Piece.py`. La fonction `Piece(l)` permet de créer une liste contenant des 0 et des 1 aléatoirement avec une probabilité  $\frac{1}{2}$ . Considérez un chiffre 1 comme une réussite (pile) et 0 comme un échec (face).

#### 💡 Conseil

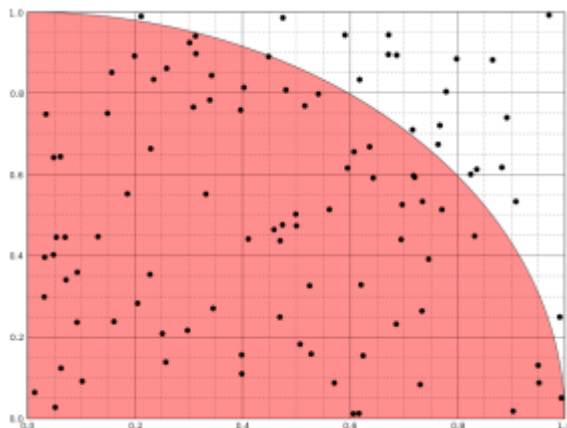
Pour estimer empiriquement la probabilité d'un événement, comptez le nombre de fois que l'événement en question se produit en effectuant un nombre d'essai. Puis divisez le nombre d'occurrence de l'événement par le nombre total d'essai. Par exemple, si vous voulez estimer la probabilité d'obtenir un 2 avec un dé. Lancez le dé 1000 fois, comptez le nombre de fois que vous obtenez 2, et divisez le résultat par 1000.

#### >\_ Solution

```
1
2 import numpy as np
3
4 #La fonction Piece retourne une liste contenant des 0 et des 1, considérez un 1 comme un succès, i.e. une fois ou la
   pièce tombe sur pile, et 0 comme un échec
5 def Piece(l):
6     return np.random.randint(0,2,l)
7
8
9
10 def proba(n,l,iter):
11     #Codez , n correspond au nombre de succès et l au nombre d'essaie. Iter correspond au nombre d'expérience
       que vous allez réaliser pour obtenir la réponse. Cela devrait être grand mais pas trop (sinon le programme
       prendra trop de temp.)
12     #10000 est un bon nombre d'itération.
13     proba = 0
14     for i in range(iter):
15         temp = Piece(l)#On simule une expérience de l lancé.
16         count = temp.sum()#On compte le nombre de fois que l'on obtient pile
17         if count == n:#Si le nombre de pile obtenue correspond à la probabilité que l'on veut estimer
18             proba +=1#On ajoute 1 à notre estimateur de probabilité
19
20
21
22     return proba/iter#Divise notre estimateur de probabilité par le nombre total d'expérience réalisée.
23
24
25 n = 5
26 l = 10
27 print("La probabilité d'avoir {} pile en {} lancés de pièce est approximativement égale à
       {}".format(n,l,proba(n,l,10000)))
```

### Question 2: (🕒 20 minutes) Une approximation de $\pi$ : Python

L'objectif de cet exercice est programmer un algorithme probabiliste permettant d'approximer le chiffre  $\pi$ . Imaginez un plan en sur lequel  $0 < x < 1$  et  $0 < y < 1$ . Sur ce dernier, nous allons dessiner un quart de cercle centré en (0,0) et avec un rayon de 1. Par conséquent, un point dans cette espace se trouve à l'intérieur du cercle si  $x^2 + y^2 < 1$ . Vous trouverez ci-dessous un schéma de la situation :



La première étape de cette exercice consiste à créer une fonction permettant de déterminer si un point est à l'intérieur (zone rouge) ou à l'extérieur du cercle. Puis, générez 10000 points dans cette espace (x et y devrait appartenir à [0,1]). Pour ce faire, vous pouvez utiliser la fonction `random.random()` après avoir importé le module **random**. Vous pouvez obtenir l'approximation de  $\pi$  à partir de la formule suivante :  $\pi \approx \left[ \frac{\text{Nombre de point dans le cercle}}{\text{Nombre de point total}} \right] \cdot 4$ . Votre réponse devrait être assez proche du vrai chiffre  $\pi$ .

#### 💡 Conseil

La fonction `random.random()` génère aléatoirement un chiffre compris entre 0 et 1. Etant donné que vous devez simuler des points en 2 dimension, vous devrez utiliser 2 fois cette fonction.

#### >\_ Solution

```
1 import random
2
3 def inside(point):#Point définit sous la forme d'un tuple
4
5     if (point[0]**2+point[1]**2) < 1:
6         return 1
7
8     else:
9         return 0
10
11 def app():
12     count = 0 #On initialise le nombre de point dans le cercle
13     for i in range(10000):
14         temp1 = random.random()#Génère la première coordonnée
15         temp2 = random.random()#Génère la deuxième coordonnée
16         temp = [temp1,temp2]#Crée le point
17
18         count += inside(temp)#On appelle la fonction. Si le point est dans le cercle, elle retourne 1, par conséquent on
19             ajoute 1 au compteur. Sinon elle retourne 0, on ajoute donc rien.
20
21     return count/10000*4#Retourn selon la formule.
22
23 print("L'approximation du chiffre pi est : {}".format(app()))
```

### Question 3: (🕒 15 minutes) Un exemple simple de la chaîne de Markov

Les slides nous ont donné un exemple de la chaîne de Markov, mais on peut aussi formaliser le concept avec les notations suivantes.

Un processus est une chaîne de Markov si

$$P(X_{n+1} = j | X_0 = i_0, X_1 = i_1, \dots, X_{n-1} = i_{n-1}, X_n = i) = P(X_{n+1} = j | X_n = i). \quad (1)$$

Le nombre  $P(X_{n+1} = j | X_n = i)$  est appelé *probabilité de transition* de l'état  $i$  à l'état  $j$  (en un pas), et on écrit :

$$p_{ij} = P(X_{n+1} = j | X_n = i) \quad (2)$$

La matrice  $\mathbf{P}$  dont l'élément à l'indice  $(i, j)$  est  $p_{ij}$  est appelée *matrice de transition*. Si les chaînes sont homogènes,  $\mathbf{P}$  a deux propriétés importantes : i,  $p_{ij} \geq 0$  et ii,  $\sum_i p_{ij} = 1$ .

Soit  $\mu_n = (\mu_n(1), \dots, \mu_n(N))$  un vecteur-ligne, avec  $\mu_n(i) = P(X_n = i)$ . Soit  $\mu_0$  la loi initiale (la loi de  $X_0$ ). En général, on a qu'à connaître  $\mu_0$  et  $\mathbf{P}$  pour simuler une chaîne de Markov.

Les probabilités marginales sont :

$$\mu_n = \mu_0 \mathbf{P}^n$$

Et on a ci-dessous un résumé de la terminologie :

1. Matrice de transition  $\mathbf{P}(i, j) = P(X_{n+1} = j | X_n = i) = p_{ij}$ .
2. Matrice de transition en  $n$  pas :  $\mathbf{P}_n(i, j) = P(X_{n+m} = j | X_m = i)$ .
3.  $\mathbf{P}_n = \mathbf{P}^n$ .
4. Probabilité marginale :  $\mu_n(i) = P(X_n = i)$ .
5.  $\mu_n = \mu_0 \mathbf{P}^n$

Etant donné que  $X_0, X_1, \dots$  est une chaîne de Markov avec 3 états  $\{0, 1, 2\}$  et la matrice de transition :

$$\mathbf{P} = \begin{bmatrix} 0.1 & 0.2 & 0.7 \\ 0.9 & 0.1 & 0.0 \\ 0.1 & 0.8 & 0.1 \end{bmatrix}$$

Supposons que  $\mu_0 = (0.3, 0.4, 0.3)$ . Trouvez  $P(X_0 = 0, X_1 = 1, X_2 = 2)$  et  $P(X_0 = 0, X_1 = 1, X_2 = 1)$ .

#### Conseil

Cet exercice vous demande de trouver deux probabilités **jointes**. Peut-être vous rappelez-vous qu'en général,

$$P(X = x, Y = y) = P(X = x)P(Y = y | X = x).$$

Pouvez-vous le reformuler avec 3 variables (notez bien que  $X_0, X_1, \dots$  est une chaîne de Markov) et utiliser les définitions ci-dessus pour trouver la réponse ?

#### Solution

$$P(X_0 = 0, X_1 = 1, X_2 = 2) = P(X_0 = 0) \times p_{01} \times p_{12} = 0.3 \times 0.1 \times 0.7 = 0.021 \quad (3)$$

$$P(X_0 = 0, X_1 = 1, X_2 = 1) = P(X_0 = 0) \times p_{01} \times p_{11} = 0.3 \times 0.1 \times 0.1 = 0.003 \quad (4)$$

#### Question 4: 7 minutes) Probabilités marginales

En utilisant la loi initiale  $\mu_0$  et la matrice  $\mathbf{P}$  de Question 3, trouvez  $\mu_1, \mu_2$  et  $\mu_3$ .

#### Conseil

Relisez et familiarisez-vous avec les notations et la terminologie ci-dessus !

### >\_ Solution

$$\mu_1 = (0.42 \quad 0.34 \quad 0.24) \quad (5)$$

$$\mu_2 = (0.37 \quad 0.31 \quad 0.32) \quad (6)$$

$$\mu_3 = (0.35 \quad 0.36 \quad 0.29) \quad (7)$$

#### Question 5: (🕒 20 minutes) Simuler une chaîne de Markov

Comme déjà mentionné plus haut, la simulation d'une chaîne de Markov  $(X_0, X_1, \dots)$  exige seulement deux éléments : la loi initiale  $\mu_0$  et la matrice de transition  $\mathbf{P}$ . Spécifiquement, l'algorithme est :

1. Supposer que  $X_0 \sim \mu_0$ . Donc,  $P(X_0 = i) = \mu_0(i)$ .
2. Le résultat de l'étape 1 est donc  $i$ ; obtenir  $X_1 \sim \mathbf{P}$  i.e  $P(X_1 = j | X_0 = i) = p_{ij}$ .
3. Le résultat de l'étape 2 est  $j$ ; obtenir  $X_2 \sim \mathbf{P}$  i.e  $P(X_2 = k | X_1 = j) = p_{jk}$ .
4. Répéter jusqu'à la fin (nombre d'itérations est arbitraire).

Ecrivez une fonction simple afin d'implémenter l'algorithme ci-dessus. Nommez-la `sim_mc()`, fonction qui prend comme paramètres `P`, `mu_0` et `n_iters`.

```
def sim_mc(mu_0, P, n_iters=500):  
    # mu_0 (n,1) # probabilités initiales - n états  
    # P (n, n) # matrice de transition  
  
    states = range(len(mu_0)) # e.g si la longueur de mu_0 est 2, on aura 2 états 0, 1  
  
    ...  
P = [[0.1, 0.9],  
      [0.7, 0.3]]  
mu_0 = [0.3, 0.7]
```

### 💡 Conseil

La méthode `random.choices()` s'avéra utile !

### >\_ Solution

```
1 import random  
2  
3 def sim_mc(mu_0, P, n_iters=500):  
4     # mu_0 (n,1) # probabilités initiales - n états  
5     # P (n, n) # matrice de transition  
6  
7     states = range(len(mu_0)) # e.g si la longueur de mu_0 est 2, on aura 2 états 0, 1  
8     X0 = random.choices(states, mu_0)[0]  
9  
10    future_states = []  
11    future_states.append(X0)  
12    for i in range(n_iters):  
13        next_state = random.choices(states, P[future_states[i]])[0]  
14        future_states.append(next_state)  
15  
16    return future_states  
17  
18 P = [[0.1, 0.9],  
19      [0.7, 0.3]]  
20  
21 mu_0 = [0.3, 0.7]  
22  
23 print(sim_mc(mu_0, P))
```