TP 0 - Processus stochastiques
Métaheuristiques pour l'optimisation

Romain Mencattini

20 septembre 2016

Table des matières

1 Introduction

Dans ce rapport, nous allons détailler chacun des problèmes, ainsi que les solutions imaginées. Puis nous parlerons des implémentations au cas par cas. Nous regarderons aussi les résultats obtenus lors des simulations. Nous avons choisi Python 3.5 pour l'implémentation.

2 Simulation d'un dé équilibré

On suppose avoir un dé à N faces. On suppose également que ce dé est équilibré, et donc que la probabilité d'une face vaut $\frac{1}{N}$. La méthode consiste à :

1. Générer un nombre aléatoire $r \in [0, 1[$
2. Calculer $i = \lfloor rN \rfloor$

Le i obtenu est donc le résultat du lancé de dé. La seule difficulté d'implémentation se trouve dans la génération du r. Pour ce faire, nous utilisons la méthode *random* de la librairie *random* de Python. Le comportement de la méthode est définie dans cette documentation : `https://docs.python.org/2/library/random.html#random.random`. Le résultat de $random.random \in [0, 1[$. Le reste du code suit la méthode présentée ci-dessus. Voici les résultats obtenus :

```
Dé équilibré, 6 faces, 100000 lancés
La face :  1  est apparue :  16715
La face :  2  est apparue :  16732
La face :  3  est apparue :  16670
La face :  4  est apparue :  16553
La face :  5  est apparue :  16874
La face :  6  est apparue :  16456
```

On obtient donc des résultats équilibrés.

3 Simulation d'un dé pipé à l'aide d'un dé équilibré

Dans ce cas, le problème consiste en un dé à N faces mais avec une probabilité d'apparition de P_i pour chaque face. Voici la méthode proposée dans la série d'exercice :

1. On divise le segment en morceau de taille $ppcm(P_i, P_j) \forall i, jinP$ ou P est notre tableau de probabilités.
2. On fait du pré-processing pour se "souvenir" à quel nombre est associé une case.
3. On lance le dé équilibré avec $N = length(Tableau)$, ou $Tableau$ est le résultat du pré-processing de l'étape 2

Tout d'abord,nous avons implémenté le ppcm entre deux nombres, puis nous l'avons appliqué à tous les éléments du tableau des P_i. Pour ce faire, nous avons utilisé la fonction *map* en Python, qui applique une fonction à tous ses éléments. Sachant que :

$ppcm(P_i, P_j) = l$, $ppcm(P_k, l) = res$
$ppcm(ppcm(P_i, P_j), P_k) = res$ L'utilisation de *map* est donc justifiée.
Ayant obtenu le ppcm, nous divisons le tableau en morceau de taille res. Puis par une double boucle, nous créons un tableau de sauvegarde, qui va permettre de faire correspondre une case à une chiffre.

Il ne reste plus qu'à lancer le dé avec un N valant la taille du nouveau tableau pour obtenir les résultats. Voici ce que donne ce programme :

```
Dé déséquilibré, 6 faces, 100000 lancés, tableau = [1/4,1/4,1/4,1/8,1/16,1/16]
La face :  1  est apparue :  24918
La face :  2  est apparue :  25141
La face :  3  est apparue :  24968
La face :  4  est apparue :  12506
La face :  5  est apparue :  6288
La face :  6  est apparue :  6179
```

On a plus ou moins la répartition $\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{16}$
Ce qui semble correct.

4 Simulation d'un dé pipé avec une pièce pipée

Pour fabriquer ce dé pipé à N faces, il nous faut une pièce biaisée.
Ces probabilités sont : p pour obtenir un pile, $1-p$ pour face. Soit r notre nombre aléatoire $\in [0, 1[$. On a que si $r < p$ alors c'est l'évènement pile. Face sinon. Ci-dessous le pseudo-code de l'énoncé :

1. On lance une pièce avec une probabilité $p = P_0$ ou P_0 est la probabilité de la première case du tableau.
2. Si c'est pile, on a comme résultat le premier chiffre
3. Sinon, on normalise la masse de probabilité et on recommence avec le reste du tableau.

Pour normaliser la masse, on fait à chaque étape :
— $masse = 1 - P_i$, où P_i est la probabilité utilisée lors du lancer
— On divise chaque élément du tableau de probabilité par la masse. On utilise *map* pour se faire.
— On rappelle la fonction jusqu'à ce qu'elle se termine.

Nous avons fait le choix d'utiliser une fonction récursive pour la beauté et la compréhensibilité du code. Une version itérative aurait été bien plus efficace. Voici les résultats obtenus :

```
Dé déséquilibré, 6 faces, 10000 lancés, tableau = [1/4,1/4,1/4,1/8,1/16,1/16]
La face :  1  est apparue :  2534
La face :  2  est apparue :  2438
La face :  3  est apparue :  2530
La face :  4  est apparue :  1261
La face :  5  est apparue :  613
La face :  6  est apparue :  624
```

Les résultats correspondent au tableau de probabilités.

5 Méthode de la roulette

La méthode de la roulette consiste à :

1. calculer le tableau de probabilités cumulées
2. tire un nombre aléatoire $r \in [0, 1[$
3. choisir la case du tableau tel que : $r > P_i^{cumul}$ pour le plus grand i possible

Nous avons parcouru le tableau de manière linéaire pour trouver le résultat. Voici ce que donnent cinquante milles lancés :

```
Roulette, 50000 lancés, tableau = [1/4,1/4,1/4,1/8,1/16,1/16]
La face :  1  est apparue :  12649
La face :  2  est apparue :  12493
La face :  3  est apparue :  12400
La face :  4  est apparue :  6269
La face :  5  est apparue :  3123
La face :  6  est apparue :  3066
```

On a encore une fois des résultats cohérent avec le tableau de probabilités initial.

6 Conclusion

Pour conclure, on peut remarquer que toutes les méthodes donnent des résultats cohérents. Leurs seules différences se trouvent dans leurs durées d'exécutions ainsi que dans leurs complexités.