# Metaheuristics for optimization
# Rapport du TP0 - Stochastic processes

Dans le cadre de ce TP, le code source est en Python. Cependant, ce rapport ne comporte que du pseudo-code. Ainsi, le lecteur est invité à consulter le fichier TP0.py pour plus de détails concernant le code source. Pour chaque exercice, le plan suivant est appliqué :

**Titre de l'exercice**
(1) Description de l'exercice
(2) La méthodologie/le pseudo-code adopté pour le résoudre
(3) Les résultats obtenus

***Nota bene*** *: Si nécessaire, les sources seront citées dans une dernière section intitulée* ***Références****.*

**Simulation of a balanced dice**

(1) Description de l'exercice

Il s'agit d'une expérience d'un lancer de dé équilibré à N faces. La probabilité que l'évènement [redacted] L'objectif est d'élaborer des algorithmes qui permettraient d'appliquer et d'illustrer le comportement de ce phénomène aléatoire simple.

(2) La méthodologie/le pseudo-code adopté pour le résoudre

Dans un premier temps, il est intéressant de définir une fonction qui retourne un nombre $i \in [1, N]$ généré aléatoirement.
Si nous souhaitons obtenir un nombre $r \in [0, 1]$ à partir de ce même nombre i, il suffit d'appliquer la formule $\mathrm{r} = \frac{i}{N}$, puisque cette formule est équivalente à celle de $\mathrm{i} = \lfloor rN \rfloor$. Étant donné que l'implémentation du code se fait en Python, la fonction *uniform(a, b)*[1] est utilisée afin de générer un nombre aléatoire $i \in [a, b[$. Voici le pseudo-code de cette fonction, qu'on nommera *RollingBalancedDice* et

[1] C'est une fonction du package ***random***.

qui prend en paramètre un entier N représentant le nombre de faces d'un dé :

[redacted]

On remarque que cette fonction est de complexité $O(1)$. De plus, il est important de noté que la fonction *uniform* renvoie une valeur réelle. Donc la valeur i qui est retournée par le pseudo-code ci-dessus doit être *"castée"* afin d'obtenir la partie entière. Cependant, cela fonctionne seulement s'il s'agit de valeurs positifs, ce qui est notre cas ici.

Désormais, nous pouvons procéder à la conception d'une fonction qui prend en paramètre un nombre de lancés d'un dé (l'effectif total) à N faces, N étant le second paramètre de la fonction. [redacted]Une telle fonction peut se décomposer en trois parties, dont voici le pseudo-code :

***pour i allant de 0 à N - 1 faire***
[redacted]
***pour i allant de 0 à (Nombre de lancés) - 1 faire***
[redacted]
***pour i allant de 0 à N - 1 faire***
[redacted]

Les trois parties sont donc : l'initialisation, qui permet de remplir notre tableau avec N cases de valeurs nulles, le calcul de l'effectif de chaque case (donc de chaque face du dé), et enfin, le calcul de la fréquence de chaque case.

(3) Les résultats obtenus

| Face | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Fréquence | 0.15 | 0.19 | 0.25 | 0.15 | 0.11 | 0.15 |

Table 1: *Avec 100 lancés.*

| Face | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Fréquence | 0.161 | 0.17 | 0.165 | 0.172 | 0.167 | 0.165 |

Table 2: *Avec 1000 lancés.*

| Face | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Fréquence | 0.168 | 0.167 | 0.169 | 0.167 | 0.161 | 0.165 |

Table 3: *Avec 10000 lancés.*

D'après ces résultats, nous remarquons que plus le nombre de lancés est grand, plus la fréquence de chaque face converge vers 0.16, soit la valeur approximative de $\frac{1}{6}$.

**Simulation of a biased coin toss**

(1) Description de l'exercice

Il s'agit d'une simulation d'un lancé de pièce. Au cours de l'expérience, deux événements peuvent avoir lieu : "Pile" ou "Face". Dans le cas où la pièce n'est pas truquée, les évènement "Pile" ou "Face" ont chacun la même probabilité $p$ d'avoir lieu, c'est-à-dire 0.5. Dans le cas contraire, il s'agit d'une pièce truquée. L'objectif de cet exercice est de comparer les résultats d'une telle expérience avec une pièce non biaisée et biaisée.

(2) La méthodologie/le pseudo-code adopté pour le résoudre

Comme cela est expliqué dans l'ouvrage [1] à la page 57, afin d'exécuter l'évènement "Face" avec probabilité $p$, il suffit que la fonction *uniform* retourne un nombre réel $r$ tel que $r < p$. Dans le cas contraire, l'évènement "Face" est rejeté et c'est l'événement "Pile" qui a lieu avec la probabilité $1 - p$. Le pseudo-code suivant permet d'illustrer cela :

***r = unifor***[redacted]
***Si***[redacted] ***alors***
[redacted]
***Sinon***
[redacted]

Dans le cadre de ce TP, cet algorithme est implémenté par une fonction inti[redacted]
À partir d'une telle fonction, nous pouvons déterminer si c'est l'évènement "Face" qui a lieu, ou si c'est celui de "Pile". [redacted] Cependant, ce raisonnement est adapté à un lancé d'une pièce non biaisée, donc lorsque $p$ vaut toujours 0.5. Afin de vérifier que la probabilité $p$ a été définie de façon à ce que le lancé de pièce ne soit pas biaisé[3], nous utiliseront dans la partie (3) la variable $x = \lambda a + (1 - \lambda)b$, avec $a$ pour "Face" et $b$ pour "Pile".

(3) Les résultats obtenus

[redacted]

[2]Les valeurs 0 et 1 sont attribuées respectivement à "Pile" et "Face" de façon arbitraire.

[3]Informatiquement parlant, c'est-à-dire selon la valeur du paramètre p, passée à la fonction *SimulationCoinToss*.

[redacted]

[redacted] Par ailleurs, $x = 1a + (1-1)b = a$. [redacted] *SimulationCoinToss(0.5)* [redacted] "Face".

**Simulation of a double biased coin toss**

(1) Description de l'exercice

L'exercice est similaire au précédent. Cette fois, il s'agit d'un double lancé de la pièce. La probabilité d'obtenir l'évènement "Pile" au premier lancé est *p1* et *p2* pour le second lancé. La probabilité d'obtenir l'évènement "Face" au premier lancé est *1 - p1* et *1 - p2* pour le second lancé.

(2) La méthodologie/le pseudo-code adopté pour le résoudre

Nous avons quatre cas qu'on nommera A, B, C et D :

$$\{\overbrace{(\underbrace{Pile}_{p1},\underbrace{Pile}_{p2})}^{A},\overbrace{(\underbrace{Pile}_{p1},\underbrace{Face}_{1-p2})}^{B},\overbrace{(\underbrace{Face}_{1-p1},\underbrace{Face}_{1-p2})}^{C},\overbrace{(\underbrace{Face}_{1-p1},\underbrace{Pile}_{p2})}^{D}\}$$

Notons $P1$, $P2$, $P3$ et $P4$ les probabilités respectives d'avoir l'évènement A, B, C ou D. Si la pièce n'est pas biaisée, alors $P1 = P2 = P3 = P4 = \frac{1}{2}.\frac{1}{2} = \frac{1}{4}$. Étant donné qu'il y a plusieurs choix possibles, un algorithme un peu plus compliqué que celui de l'exercice précédent permet de réaliser l'expérience[1]. Nous développerons cet algorithme dans l'exercice suivant.

Le pseudo-code suivant, d'une fonction qui prend en paramètre la probabilité *p*, fonctionne très bien pour cet exercice :

```
t1 = SimulationCoinToss(p)
t2 = SimulationCoinToss(p)
Selon valeurs de t1 et t2 faire :
    Cas (t1=1) et (t2=1): l'évènement A a lieu
    Cas (t1=1) et (t2=0): l'évènement B a lieu
    Cas (t1=0) et (t2=0): l'évènement C a lieu
    Cas (t1=0) et (t2=1): l'évènement D a lieu
```

(3) Les résultats obtenus

**Cas d'une pièce équilibrée** : *SimulationDoubleCoinToss(0.5) = L'évènement B : {Pile, Face} a lieu.* Les autres évènements pouvaient se produire également.

**Cas d'une pièce pièce biaisée en faveur de "Pile"** : SimulationDoubleCoinToss(1) = *L'évènement A : {Pile, Pile} a lieu.* Les autres évènements ne se produiront jamais.

**Cas d'une pièce pièce biaisée en faveur de "Face"** : SimulationDoubleCoinToss(0) = *L'évènement C :* {*Face, Face*} *a lieu.* Les autres évènements ne se produiront jamais.

**Roulette method**

(1) Description de l'exercice

Nous avons $N$ évènements, chacun de probabilité $P_i, i \in \{0, .., N-1\}$. L'objectif de cet exercice est de pouvoir, à l'aide d'un algorithme optimisé, obtenir un évènement $i$ de probabilité $P_i$, avec $r$, un nombre tiré uniformément entre 0 et 1. Il faut vérifier ensuite que pour un grand nombre d'événements générés, la fréquence de chaque événement correspond à la probabilité qui lui est associée.

(2) La méthodologie/le pseudo-code adopté pour le résoudre

*Dans ce qui suit, les explications sont reprises de l'ouvrage [1], page 57 à 58. Cependant, dans le cadre de ce TP, le premier pseudo-code proposé est issu du site web [2] et a pour but de générer un événement en O(log(n)) (en terme de temps de calcul); tandis que celui proposé dans l'ouvrage [1], est de complexité O(n) puisqu'il effectue une recherche linéaire.*

Supposons que nous avons $N$ événements, chacun avec une probabilité différente. Soit $P_i, i \in \{0, .., N-1\}$ la probabilité de l'évènement $i$ telle que $\sum_i P_i = 1$. Une fois qu'un nombre $r$ est tiré uniformément entre 0 et 1, il faut choisir le plus grand [redacted] Afin de procéder à une recherche dichotomique, il faut d'abord créer un tableau intermédiaire [redacted] Le pseudo-code suivant illustre une telle démarche [2] :

***création du tableau InitTab de taille N***
[redacted]
***pour i allant de*** [redacted] ***faire***
[redacted]

Nous pouvons ensuite procéder à la génération d'un évènement :

***r = uniform(0,1)***
[redacted]
***indice =*** [redacted]

[redacted]
***Si r <*** $P_i^{cumul}$
[redacted]
***retourne indice***
[redacted]

***Sinon***

***cumul***

***retourne indice***

Désormais, il est possible d'obtenir le tableau de fréquences à l'aide d'une fonction que l'on nommera

(3) Les résultats obtenus

| Événement | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Probabilité | 0.1 | 0.2 | 0.1 | 0.3 | 0.1 | 0.2 |
| Fréquence | 0.10025 | 0.20054 | 0.09968 | 0.29951 | 0.09904 | 0.20098 |

Table 4: *Avec 100000 générations aléatoires.*

**Références**

[1] Introduction aux métaheuristiques, B. Chopard et M. Tomassini, PUP, 2017.

[2] http://www.keithschwarz.com/darts-dice-coins/.