Méthode de Monte-Carlo - loi faible des grands nombres

Code de partage avec Capytale : d31a-9959885

On utilisera les bibliothèques suivantes :

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

Une méthode de Monte-Carlo consiste à faire un grand nombre de réalisations numériques d'un phénomène aléatoire pour approcher une espérance. L'idée est fondée sur la loi faible des grands nombres, dont la traduction pratique consiste à dire que, étant donnée une variable admettant un moment d'ordre 2, $(x_k)_{k\in\mathbf{N}}$ une suite de réalisations indépendantes de la loi de X, alors, pour n « grand » :

$$\frac{1}{n}\sum_{k=1}^{n} x_k \approx E(X)$$

et plus n augmente, meilleure est l'approximation d'après la loi faible des grands nombres

Comme nous l'avons vu, la valeur $\frac{1}{n}\sum_{k=1}^{n} x_k$ est appelée **moyenne empirique**.

Exercice 1

1. créer une fonction `Tirage1(p)` qui simule la loi $\mathscr{B}(p)$ modélisant le jeu du Pile ou Face avec une probabilité p d'obtenir Pile sur un lancer.
2. créer une fonction `Simulation(N, p)` qui simule N lancers et renvoie la fréquence d'apparition de Pile sur ces N lancers.
   On utilisera une boucle `for` et un compteur Pile qui comptera le nombre de Pile.
   On vérifiera que la valeur renvoyée se rapproche de p lorsqu'on augmente la valeur de N

Exercice 2

On considère une urne contenant n boules noires et b boules blanches ($b \geqslant 2$). On effectue des tirages sans remise d'une boule dans l'urne et on note :

- X le rang d'apparition de la première boule blanche,
- Y le rang d'apparition de la deuxième boule blanche.

1. Définir une fonction `SimX(n, b)` qui simule une réalisation de la variable X
2. Compléter la fonction suivante simulant une réalisation de Y

```
def SimY(n, b):
    # x=

    return
```

3. Vérifier que, pour tous entiers k et ℓ tels que $1 \leqslant k \leqslant \ell$:

$$\binom{\ell}{k} = \frac{\ell}{k}\binom{\ell-1}{k-1}$$

   Puis définir une fonction récursive `coefbin(k, l)` qui calcule le coefficient binomial $\binom{\ell}{k}$

4. On considère la fonction suivante :

```
def simulationX(n, b):
    L=np.zeros(n+2)
    for i in range(10000):
        x= SimX(n, b)
        L[x] +=1/10000
    return L
```

   (a) Que contient le vecteur ligne `L` ?

   (b) On montre que : $P(X=k)=\dfrac{\binom{n-k+b}{b-1}}{\binom{n+b}{b}}$ pour $k \in \{1,\dots n+1\}$

   Ecrire une fonction `LoiX(n, b)` renvoyant un vecteur ligne contenant les valeurs $P(X=0),\dots,P(X=n)$
   On utilisera la fonction `coefbin` et on remarquera que $P(X=0)=0$

   (c) Vérifier le résultat précédent par simulation à l'aide de la fonction `simulationX`

5. On veut modifier et compléter la fonction `simulationX` pour approcher la loi de Y

   (a) Compléter cette fonction.

```
def SimulationXY(n, b):
    LXY=np.zeros([n+3, n+3])
    for i in range(10000):
        x= SimX(n, b)
        y = ...
        LXY[x, y]= ...
    return LXY
```

   (b) On montre que : $P(Y=i)=(i-1)\times\dfrac{\binom{n+b-i}{b-2}}{\binom{n+b}{b}}$ pour $i \in \{2,\dots n+2\}$

   Vérifier ce résultat par simulation.

   (c) Compléter la fonction suivante pour simuler la loi du couple (X,Y)

```
def SimulationXY(n, b):
    LXY=np.zeros([n+3, n+3])
    for i in range(10000):
        x= SimX(n, b)
        y = ...
        LXY[x, y]= ...
    return LXY
```