TP11 : Loi faible des grands nombres et méthode de Monte Carlo

Commencer par importer les bibliothèques suivantes dans chaque fichier **Python** utilisé :

```
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
```

I. Loi faible des grands nombres

Théorème 1 (Loi faible des grands nombres)**.**

Soit $(X_k)_{k\in\mathbb{N}^*}$ *une suite de variables aléatoires.*
On suppose que les v.a.r. X_k *:*

× *sont indépendantes,*
× *admettent toutes la même espérance* m*,*
× *admettent toutes la même variance* σ^2*.*

On pose, pour tout $n \in \mathbb{N}^*$, $\overline{X_n} = \dfrac{1}{n}\sum\limits_{k=1}^{n} X_k$ *(moyenne empirique). On a alors,*

$$\boxed{\forall \varepsilon > 0, \ \lim_{n\to+\infty} \mathbb{P}\left(\left[\ \left|\overline{X_n} - m\right| \geqslant \varepsilon\ \right]\right) = 0}$$

Remarque

- Il est fréquent de considérer (notamment pour les simulations informatiques) une suite $(X_k)_{k\in\mathbb{N}^*}$ de v.a.r. :
  × indépendantes,
  × **de même loi**,
  × admettant une espérance et une variance.

  Ces hypothèses sont plus strictes que celles énoncés par la LfGN.
  On peut donc bien évidemment utiliser la LfGN dans ce cadre.
- La loi faible des grands nombres se comprend de la manière suivante : lorsque n est grand, il est peu probable que $\overline{X_n}$ prenne une valeur éloignée de m. Autrement dit, lorsque n est grand, il est très probable que $\overline{X_n}$ prenne une valeur proche de m.
  A retenir :

  $$\boxed{\text{Une simulation de } \overline{X_n} \text{ pour } n \text{ grand donne une approximation de } m = \mathbb{E}(X_1)}$$
- Ce résultat est plus général qu'il n'y parait, car tout calcul de probabilité peut s'interpréter comme le calcul de l'espérance d'une variable aléatoire bien choisie. En effet, si A est un événement, alors la variable aléatoire $X = \mathbb{1}_A$ suit une loi de Bernoulli de paramètre $p = \mathbb{P}(A)$ et donc $\mathbb{E}(X) = \mathbb{P}(A)$. Cette remarque est très importante et est au centre de beaucoup de questions d'informatique aux concours.
- On appelle méthode de Monte Carlo toute méthode de calcul approché basée sur la loi faible des grands nombres.

II. Structure classique de la partie informatique aux concours

Dans les énoncés de concours, la LfGN apparaît souvent dans les questions d'informatique. Considérons un énoncé consistant à étudier une v.a.r. X qui admet une variance. On trouvera fréquemment les questions suivantes.

1. Écrire une fonction / compléter une fonction / expliquer ce que fait une fonction d'en-tête `def simuX():` permettant de simuler la v.a.r. X. Cette fonction peut éventuellement prendre des paramètres d'entrée. Généralement, cette simulation peut être obtenue :

   × en écrivant une fonction permettant de simuler l'expérience aléatoire considérée. La valeur simulée de X est calculée lors de cette expérience (cas classique lorsqu'on travaille avec des v.a.r. discrètes).

   × ou bien parce que X s'écrit à l'aide d'autres v.a.r. qu'on peut facilement simuler (X est une transformée de v.a.r. qui suivent des lois usuelles par exemple).

2. Écrire un programme / compléter un programme / expliquer ce que fait un programme permettant d'obtenir une valeur approchée de l'espérance de la v.a.r. X.

Démonstration.

- L'idée naturelle est la suivante :

   × on simule un grand nombre N de fois la v.a.r. X.
   (généralement, on prend $N = 10^4$ ou 10^5 ou 10^6)

   × on détermine la moyenne arithmétique des résultats obtenus (*i.e.* la réalisation de la moyenne empirique $\overline{X_n}$).

```
def approxEsp(N):
    S = 0
    for i in range(N):
        S = S + simuX()
    return S / N
```

Autre possibilité : on peut créer un tableau/une liste contenant les N simulations de la v.a.r. X puis déterminer la moyenne arithmétique de ces observations à l'aide des opérations prédéfinies en **Python**.

```
def approxEsp(N):
    L = []
    for i in range(N):
        L.append(simuX())
    return np.mean(L)
```

- Mathématiquement parlant, cela consiste à considérer un N-échantillon $(X_1, \dots, X_N)$ de la v.a.r. X. Autrement dit, les v.a.r. $X_1, \dots, X_N$ sont :

   × indépendantes,

   × de même loi que X.

   Comme X admet une variance, on se trouve dans un cadre (plus strict) permettant d'appliquer la LfGN. Simuler ce N-échantillon, c'est en obtenir un N-uplet d'observations $(x_1, \dots, x_N)$ (c'est ce que contient la liste `L` dans le programme précédent). La LfGN permet d'affirmer que :

$$\boxed{\frac{1}{N} \sum_{i=1}^{N} x_i \ \simeq \ \mathbb{E}(X)}$$

□

III. Exemples d'approximation d'une espérance

III.1. Un exemple simple et vérifiable

Soit X la v.a.r. dont la loi est donnée par :

- $X(\Omega) = \{0, 1, 2\}$
- $\mathbb{P}([X = 0]) = \frac{7}{12}$, $\mathbb{P}([X = 1]) = \frac{1}{6}$, $\mathbb{P}([X = 2]) = \frac{1}{4}$

► Vérifier que cela définit bien une loi de probabilité.

► Calculer $\mathbb{E}(X)$.

► Compléter la fonction suivante pour qu'elle renvoie une simulation de la v.a.r. X.

```
def simuX():
    r = ________________
    if _____________:
        return 0
    elif _____________:
        return 1
    else:
        return 2
```

► Expliquer ce que renvoie le programme suivant et le tester plusieurs fois avec $N = 10^4$ puis avec $N = 10^5$ et enfin avec $N = 10^6$.

```
def mystere(N):
    L = []
    for k in range(N):
        L.append(simuX())
    return np.mean(L)
```

III.2. Un autre exemple vérifiable mais où le calcul est moins simple

Soit $n \in \mathbb{N}^*$. On considère une urne contenant n boules, numérotées de 1 à n. On effectue un unique tirage dans cette urne et on note le numéro de la boule obtenue. Si k désigne le numéro de la boule tirée, alors on retire de l'urne toutes les boules portant un numéro strictement supérieur à k. On effectue ensuite une seconde série de tirages successifs et avec remise dans l'urne, jusqu'à-ce qu'on obtienne la boule numéro 1. On note X_n la variable aléatoire égale au nombre de tirages effectués lors de cette seconde série de tirages.

- Compléter la fonction **Python** suivante pour qu'elle renvoie une simulation de la v.a.r. X_n.

```
def simuX(n):
    k = ______________________
    return ________________________
```

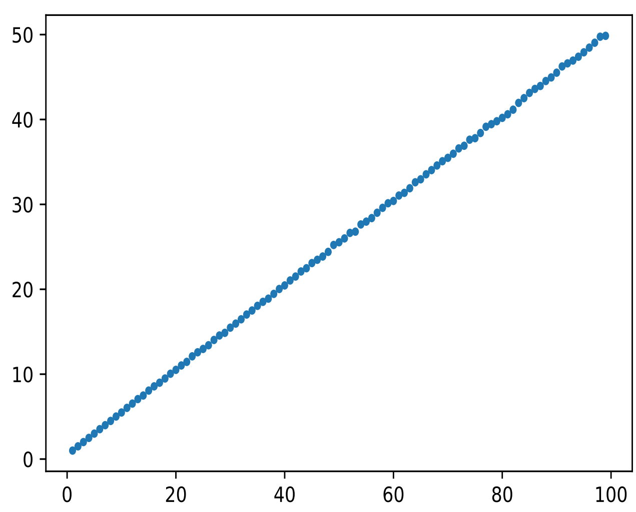
- Compléter la fonction **Python** suivante pour qu'elle renvoie une approximation de $\mathbb{E}(X_n)$.

```
def approxEsp(N, n):
    E = np.zeros(N)
    for i in range(N):
        E[i] = simuX(n)
    return ______________
```

- Le programme **Python**

```
N = 10**5
abscisse = [n for n in range(1,100)]
ordonnee = [approxEsp(N, n) for n in abscisse]
plt.plot(abscisse, ordonnee, '.')
```

renvoie



Que peut-on conjecturer ?

- A la maison : montrer que, pour tout $n \in \mathbb{N}^*$, $\mathbb{E}(X_n) = \dfrac{n+1}{2}$.

IV. Exemples d'approximation d'une probabilité

IV.1. Un cas d'école pour illustrer la méthode

On considère un dé à 6 faces et on souhaite savoir si le dé est truqué ou non. Nous allons proposer un protocole d'expérience qui nous permette de répondre à cette question.
Notons, pour tout $i \in [\![1, 6]\!]$, p_i la probabilité que le dé tombe sur la face numéro i lorsqu'on le lance une fois. Le dé est équilibré si et seulement si, pour tout $i \in [\![1, 6]\!]$, $p_i = \frac{1}{6}$. Il s'agit donc de proposer un protocole qui permette de calculer une approximation de p_i. Si toutes les approximations sont proches de $\frac{1}{6}$, on pourra conclure que le dé est équilibré, sinon on pourra conclure qu'il est truqué.
Nous allons calculer une approximation de p_6 et le protocole sera analogue pour les autres valeurs.
On lance une infinité de fois le dé. Pour tout $k \in \mathbb{N}^*$, on note A_k l'événement : « le dé tombe sur 6 lors du $k^{\text{ème}}$ lancer » et on pose

$$Z_k = \mathbb{1}_{A_k}$$

Autrement dit,

$$Z_k = \begin{cases} 1 & \text{si le dé tombe sur 6 lors du } k^{\text{ème}} \text{ lancer} \\ 0 & \text{sinon} \end{cases}$$

Les v.a.r. Z_k :
× sont indépendantes
× sont de même loi (elles suivent toutes la loi $\mathcal{B}(p_6)$)
× admettent toutes une espérance (qui est p_6) et une variance (qui est $p_6(1 - p_6)$)
D'après la loi faible des grands nombres, pour tout $\varepsilon > 0$:

$$\lim_{n \to +\infty} \mathbb{P}\left(\left[\left| \overline{Z_n} - p_6 \right| \geqslant \varepsilon \right] \right) = \lim_{n \to +\infty} \mathbb{P}\left(\left[\left| \frac{Z_1 + \cdots + Z_n}{n} - p_6 \right| \geqslant \varepsilon \right] \right) = 0$$

On fixe maintenant $N \in \mathbb{N}$ un entier très grand (par exemple $N = 10^6$) et on choisit comme approximation de p_6 la valeur prise par la variable $\overline{Z_N}$ au terme des N premiers lancers. On obtient alors

$$\begin{aligned} p_6 &\simeq \frac{\text{nombre de 6 obtenus en } N \text{ tirages}}{N} \\ &\simeq \text{fréquence d'apparition observée de la face 6 en } N \text{ tirages} \end{aligned}$$

Le protocole consiste donc à lancer un nombre important de fois le dé et à compter la fréquence d'apparition du chiffre 6. Cette fréquence est une bonne approximation de la probabilité que le dé tombe sur 6 d'après la loi faible des grands nombres.
Ce résultat est cohérent avec notre intuition concernant le sens à donner au nombre $\mathbb{P}(A)$ pour A un événement : si l'on répète un grand nombre de fois l'expérience, l'événement A sera parfois réalisé, parfois pas réalisé, mais la fréquence empirique à laquelle il se réalise est proche de $\mathbb{P}(A)$.

Traduction informatique : Supposons que l'on ait une fonction **Python** `simulX()` qui simule le résultat d'un lancer de notre dé. Alors les fonctions :

```
def Approxp6(N):
    S = 0
    for k in range(N):
        if simulX() == 6:
            S = S + 1
    return S / N
```

```
def Approxp6bis(N):
    F = 0
    for k in range(N):
        if simulX() == 6:
            F = F + 1/N
    return F
```

renvoient une approximation de p_6 lorsque N est choisi suffisamment grand.

IV.2. Rang du premier double `Pile`

On lance une infinité de fois une pièce équilibrée. On note X la variable aléatoire égale au rang du premier double `Pile`, c'est-à-dire que X prend la valeur n si on obtient, pour la première fois, deux `Pile` consécutifs aux rangs n et $n-1$.

- Donner $X(\Omega)$.

- Compléter la fonction suivante pour qu'elle simule X.

```
def simuX():
    n = 1 # numéro du lancer
    resultat = rd.binomial(1,1/2) # Pile est encodé par 1
    while True:
        if resultat == 0:
            n = n+1
            resultat = rd.binomial(1,1/2)
        else:
            n = n+1
            resultat = rd.binomial(1,1/2)
            if ________________:
                return n
```

- Que calcule la fonction suivante ?

```
def mystere(N):
    S = 0
    for k in range(N):
        if simuX() <= 5:
            S = S + 1
    return S / N
```

- Que calcule la fonction suivante ?

```
def mystere2(N):
    F = 0
    for k in range(N):
        if simuX() != 3:
            F = F + 1/N
    return F
```

IV.3. Deux boules noires d'affilée

On fixe un entier naturel non nul n. On considère une urne contenant initialement 1 boule noire et 1 boule blanche. On effectue une série de n tirages successifs et avec remise dans cette urne. A chaque tirage, après avoir remis dans l'urne la boule tirée, on lance une pièce équilibrée. Si cette pièce tombe sur `Pile`, on rajoute une boule noire, sinon on rajoute une boule blanche. On note :

A_n : « On tire deux boules noires consécutives au cours de l'expérience »

- Compléter la fonction **Python** suivante pour qu'elle simule l'expérience décrite et renvoie `True` si l'événement A_n est réalisé, et `False` sinon.

```python
def simuA(n):
    B = _____ # Nombre de boules blanches
    N = _____ # Nombre de boules noires
    booleen = False
    for k in range(n):
        if ________________________________ # Si on tire une boule noire
            if booleen == True: # Si on avait déjà tiré une boule noire
                return ________
            else:
                booleen = ________
        else: # Si on tire une boule blanche
            booleen = _________
        if _______________________
            B = B+1
        else:
            N = N+1
    return _________
```

- Compléter la fonction **Python** suivante pour qu'elle renvoie une approximation de $\mathbb{P}(A_n)$.

```python
def approxProbA(n):
    S = 0
    for k in range(10**5):
        if _____________
            S += 1
    return _____________
```

- On considère un second protocole, où l'on procède cette fois-ci à une suite de tirages sans remise. On trace les 50 premiers termes de la suite $(\mathbb{P}(A_n))_{n\in\mathbb{N}^*}$ pour chacun des deux protocoles. Que remarque-t-on ?