

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 $n \in \mathbb{N}^*$, $\overline{X}_n = \frac{1}{n} \sum_{k=1}^n X_k$ (moyenne empirique). On a alors,

$$\forall \varepsilon > 0, \lim_{n \rightarrow +\infty} \mathbb{P} ([|\overline{X}_n - m| \geq \varepsilon]) = 0$$

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 :

Une simulation de \overline{X}_n pour n grand donne une approximation de $m = \mathbb{E}(X_1)$

- Ce résultat est plus général qu'il n'y paraît, 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 (généralement on demande plutôt de compléter une fonction ou d'expliquer ce que fait une fonction) d'entête `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 (généralement on demande plutôt de compléter un programme ou d'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 $\overline{X_n}$).

```

1 def approxEsp(N):
2     S = 0
3     for i in range(N):
4         S = S + simuX()
5     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**.

```

1 def approxEsp(N):
2     L = []
3     for i in range(N):
4         L.append(simuX())
5     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'observation (x_1, \dots, x_N) (c'est ce que contient la liste L dans le programme précédent). La LfGN permet d'affirmer :

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

□

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 .

```
1 def simuX():
2     r = _____
3     if _____:
4         return 0
5     elif _____:
6         return 1
7     else:
8         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 $N = 10^6$.

```
1 def mystere(N):
2     L = []
3     for k in range(N):
4         L.append(simuX())
5     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 .

```

1 def simuX(n):
2     k = _____
3     return _____

```

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

```

1 def mystere(N, n):
2     E = np.zeros(N)
3     for i in range(N):
4         E[i] = simuX(n)
5     return _____

```

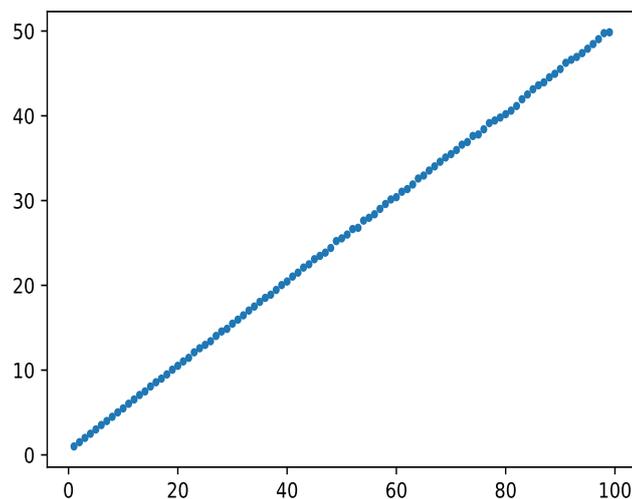
- Le programme **Python**

```

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

```

renvoie



Que peut-on conjecturer ?

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

III.3. Une expérience aléatoire sur la suite de Syracuse

Le problème « $3x+1$ », également appelé « conjecture de Syracuse », date d'une centaine d'année et porte sur des suites d'entiers définies récursivement.

Considérons un entier naturel non nul N fixé. On définit la suite $(u_n(N))_{n \in \mathbb{N}^*}$ par récurrence en posant :

$$u_1 = N \quad \text{et} \quad \forall n \in \mathbb{N}^*, \quad u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$$

et on appelle cette suite la suite de Syracuse associée à l'entier N . Par exemple :

$$\begin{aligned} (u_n(1))_{n \in \mathbb{N}^*} &= (1, 4, 2, 1, 4, 2, 1, 4, 2, 1, \dots) \\ (u_n(2))_{n \in \mathbb{N}^*} &= (2, 1, 4, 2, 1, 4, 2, 1, 4, 2, 1, \dots) \\ (u_n(3))_{n \in \mathbb{N}^*} &= (3, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \dots) \\ (u_n(4))_{n \in \mathbb{N}^*} &= (4, 2, 1, 4, 2, 1, 4, 2, 1, 4, 2, 1, \dots) \\ (u_n(5))_{n \in \mathbb{N}^*} &= (5, 16, 8, 4, 2, 1, 4, 2, 1, \dots) \end{aligned}$$

On remarque que si la suite de Syracuse prend la valeur 1, alors elle devient périodique et le cycle 1, 4, 2 se répète éternellement.

La conjecture de Syracuse s'énonce comme suit : pour tout entier naturel non nul N , il existe un rang $n \in \mathbb{N}^*$ tel que $u_n(N) = 1$. Autrement dit, pour tout entier $N > 0$, la suite de Syracuse associée à N prend la valeur 1.

Cette conjecture n'est toujours pas démontrée (n'essayez pas de le faire!). Cependant, il a été vérifié par ordinateur qu'elle est vraie pour $N \leq 10^{20}$ environ. Dans ce TP, nous ferons comme si cette conjecture était vraie.

Pour tout entier $N > 0$, on appelle temps de vol de N , et on note $T(N)$, le premier entier $n \in \mathbb{N}^*$ tel que $u_n(N) = 1$. Par exemple :

$$T(1) = 1, \quad T(2) = 2, \quad T(3) = 8, \quad T(4) = 3, \quad T(5) = 6$$

Puisque toute suite de Syracuse se répète une fois le nombre 1 atteint, toute l'information utile est contenue dans la première partie de la suite, jusqu'à ce qu'on tombe sur 1. Cette première partie de la suite a, par définition, une longueur égale à $T(N)$. On appellera trajectoire de vol associée à N cette partie de la suite.

Nous donnons maintenant deux fonctions **Python** qui permettent de calculer la trajectoire de vol de tout entier N non nul. La première permet de calculer le prochain terme d'une suite de Syracuse, tandis que la deuxième renvoie la trajectoire de vol associée à un entier non nul N donné, sous forme de liste.

```

1 def nextTermeSyracuse(u):
2     if u % 2 == 0: # Si u_n est pair
3         return u // 2
4     else: # Si u_n est impair
5         return 3*u+1

```

```

1 def trajectoireVolSyracuse(n):
2     L = [n]
3     if n == 1:
4         return L
5     else:
6         return L + trajectoireVolSyracuse(nextTermeSyracuse(n))

```

La deuxième fonction utilise un procédé récursif hors-programme (la fonction s'appelle elle-même, mais une condition d'arrêt assure que le procédé récursif ne va pas devenir une boucle infinie).

On fixe, pour tout entier $k \in \mathbb{N}^*$, $X_k \hookrightarrow \mathcal{G}(\frac{1}{k})$. On définit ensuite, pour tout $k \in \mathbb{N}^*$:

- Y_k la variable aléatoire définie par : pour tout $\omega \in \Omega$, $Y_k(\omega)$ est égal au premier entier non nul N tel que le nombre $X_k(\omega)$ appartient à la trajectoire de vol associée à N .
- Z_k la variable aléatoire définie par : pour tout $\omega \in \Omega$, $Z_k(\omega)$ est égal au temps de vol de l'entier $X_k(\omega)$. Autrement dit, $Z_k = T(X_k)$.

On remarque que $0 \leq Y_k \leq X_k$, donc Y_k admet une espérance car X_k en admet une et $\mathbb{E}(Y_k) \leq \mathbb{E}(X_k)$. De même, Y_k admet un moment d'ordre 2 et donc une variance.

Il n'est par contre pas évident que Z_k admette une variance, on supposera que c'est le cas dans la suite.

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

```

1 def simuY(k):
2     X = _____
3     Y = 1
4     while _____
5         Y = Y + 1
6     return Y

```

- Expliquer ce que renvoie le programme suivant lorsque N est grand.

```

1 def mystere(N, k):
2     S = 0
3     for i in range(N):
4         S = S + simuY(k)
5     return S / N

```

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

```

1 def simuZ(k):
2     X = _____
3     return _____

```

- Expliquer ce que renvoie le programme suivant lorsque N est grand.

```

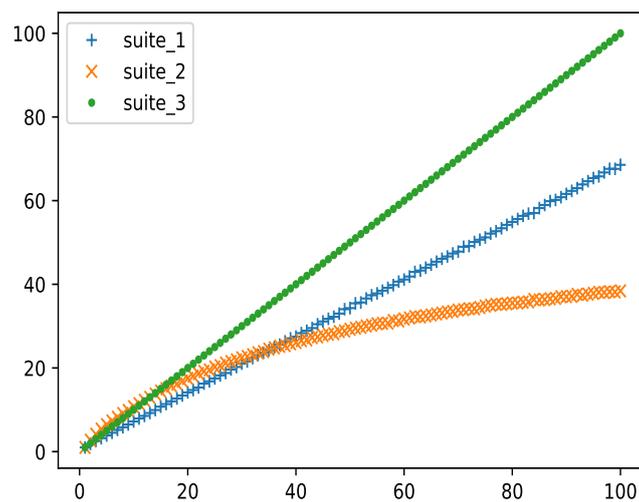
1 def mystere(N, k):
2     F = 0
3     for i in range(N):
4         F = F + simuY(k)/N
5     return F

```

► On exécute le script suivant

```
1 N = 10**5
2 nb_points = 100
3 abscisse = [k for k in range(1,nb_points+1)]
4 ordonneeY = []
5 ordonneeZ = []
6 for k in abscisse:
7     print(k)
8     ordonneeY.append(approxEspY(N, k))
9     ordonneeZ.append(approxEspZ(N, k))
10 ordonneeX = [k for k in range(1,nb_points+1)]
11 plt.plot(abscisse, ordonneeY, '+', label='suite_1')
12 plt.plot(abscisse, ordonneeZ, 'x', label='suite_2')
13 plt.plot(abscisse, ordonneeX, '.', label='suite_3')
14 plt.legend(loc='best')
```

et on reproduit ci-dessous la figure obtenue :



Préciser qui sont les trois suites représentées. Qu'y a-t-il de remarquable ?

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 $i \in \llbracket 1, 6 \rrbracket$, 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 $i \in \llbracket 1, 6 \rrbracket$, $p_i = \frac{1}{6}$. 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 $\varepsilon > 0$:

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

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** `SimuX()` qui simule le résultat d'un lancer de notre dé. Alors les fonctions :

```

1 def Approxp6(N):
2     S = 0
3     for k in range(N):
4         if SimuX() == 6:
5             S = S + 1
6     return S / N

```

```

1 def Approxp6bis(N):
2     F = 0
3     for k in range(N):
4         if SimuX() == 6:
5             F = F + 1/N
6     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 .

```

1  def simuX():
2      n = 1 # numéro du lancer
3      resultat = rd.binomial(1,1/2) # Pile est encodé par 1
4      while True:
5          if resultat == 0:
6              n = n+1
7              resultat = rd.binomial(1,1/2)
8          else:
9              n = n+1
10             resultat = rd.binomial(1,1/2)
11             if _____:
12                 return n

```

- Que calcule la fonction suivante ?

```

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

```

- Que calcule la fonction suivante ?

```

1  def mystere2(N):
2      F = 0
3      for k in range(N):
4          if simuX() != 3:
5              F = F + 1/N
6      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 qui renvoie **True** si l'événement A_n est réalisé, et **False** sinon.

```

1 def simuA(n):
2     B = ____ # Nombre de boules blanches
3     N = ____ # Nombre de boules noires
4     booleen = False
5     for k in range(n):
6         if _____ # Si on tire une boule noire
7             if booleen == True: # Si on avait déjà tiré une boule noire
8                 return _____
9             else:
10                booleen = _____
11        else: # Si on tire une boule blanche
12            booleen = _____
13        if _____
14            B = B+1
15        else:
16            N = N+1
17        return _____

```

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

```

1 def approxProbA(n):
2     S = 0
3     for k in range(10**5):
4         if _____
5             S += 1
6     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 ?

