TP libre : Loi faible des grands nombres et suite de Syracuse

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
```

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 :

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

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 $N \leqslant 10^{20}$ 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 :

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

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.

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

```
def trajectoireVolSyracuse(n):
    L = [n]
    if n == 1:
        return L
    else:
        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 se donne une suite $(X_k)_{k\in\mathbb{N}^*}$ de variables aléatoires indépendantes telle que, pour tout entier $k \in \mathbb{N}^*$, $X_k \hookrightarrow \mathcal{G}\left(\frac{1}{k}\right)$. On note 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 $0 \leqslant Y_k \leqslant X_k$, donc Y_k admet une espérance car X_k en admet une (par un théorème de domination hors-programme) et $\mathbb{E}(Y_k) \leqslant \mathbb{E}(X_k)$. 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.

```
def simuY(k):
    X = ____________________
    Y = 1
    while ____________________________________________
        Y = Y + 1
    return Y
```

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

```
def mystere(N, k):
    S = 0
    for i in range(N):
        S = S + simuY(k)
    return S / N
```

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

```
def simuZ(k):
    X = ____________________
    return ____________________________________
```

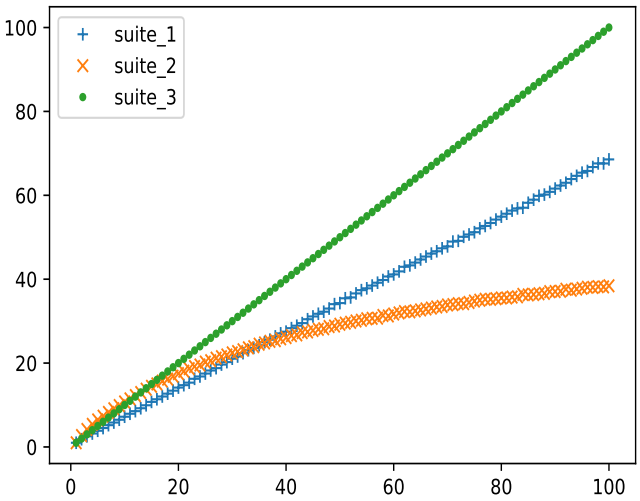
- Expliquer ce que renvoie le programme suivant lorsque N est grand.

```
def mystere(N, k):
    F = 0
    for i in range(N):
        F = F + simuY(k)/N
    return F
```

- On exécute le script suivant

```
N = 10**5
nb_points = 100
abscisse = [k for k in range(1,nb_points+1)]
ordonneeY = []
ordonneeZ = []
for k in abscisse:
    print(k)
    ordonneeY.append(approxEspY(N, k))
    ordonneeZ.append(approxEspZ(N, k))
ordonneeX = [k for k in range(1,nb_points+1)]
plt.plot(abscisse, ordonneeY, '+', label='suite_1')
plt.plot(abscisse, ordonneeZ, 'x', label='suite_2')
plt.plot(abscisse, ordonneeX, '.', label='suite_3')
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 ?