TP3 : Calcul de sommes finies en **Python**

I. Avant propos

On considère dans ce TP les suites $(u_n)_{n\in\mathbb{N}}$ et (v_n) suivantes :

$$\forall n \in \mathbb{N},\ u_n \ = \ \frac{n^2}{3^n} \qquad \text{et} \qquad \begin{cases} v_0 = 1 \\ \forall n \in \mathbb{N},\ v_{n+1} \ = \ \dfrac{2\, v_n}{\mathrm{e}^{v_n} + \mathrm{e}^{-v_n}} \end{cases}$$

Objectif du TP : il s'agit d'explorer les différentes méthodes permettant le calcul des sommes partielles d'ordre n : $S_n = \sum\limits_{k=0}^{n} u_k$ et $T_n = \sum\limits_{k=0}^{n} v_k$.

II. Calcul des sommes partielles d'ordre n

II.1. Calcul de S_n

Il s'agit ici d'illustrer le cas où la suite (u_n) est donnée sous forme explicite.

II.1.a) Méthode itérative

▶ Écrire une fonction `calculSn` qui :
- × prend en paramètre un entier `n`,
- × renvoie la valeur de S_n.

On pourra créer une variable `S`, initialement égale à 0, et, à l'aide d'une structure itérative, mettre à jour la variable `S` pour calculer la valeur de S_n.

```python
def calculSn(n):
    S = 0
    for i in range(n+1):
        S = S + i**2 / 3**i
    return S
```

▶ Que vaut S_0 ? S_5 ? S_{10} ? S_{100} ? S_{1000} ?

Pour ne pas avoir à appeler plusieurs fois la fonction, on peut écrire le petit script suivant :

```python
A = [0,5,10,100,1000]
for k in A:
    print(calculSn(k))
```

On trouve : $S_0 = 0$, $S_5 \simeq 1.4115226337448559$, $S_{10} \simeq 1.4988738166607394$, $S_{100} \simeq 1.5000000000000004$ et $S_{1000} \simeq 1.5000000000000004$.

II.1.b) Utilisation des fonctionnalités de la bibliothèque numpy

- Écrire une fonction `premSuiteU` qui :
  - × prend en paramètre un entier `n`,
  - × renvoie un tableau contenant les $\texttt{n}+1$ premiers termes de la suite (u_n).

  On pourra créer un tableau `U` initialement rempli de $n+1$ zéros et le remplir à l'aide d'une structure itérative.

```python
import numpy as np
def premSuiteU(n)
    U = np.zeros(n+1)
    for i in range(n+1):
        U[i] = i**2 / 3**1
    return U
```

- Que réalise l'appel `np.sum(np.arange(5))` ? Détailler le rôle de la fonction `np.sum`.

La fonction `np.sum` prend en paramètre un tableau et renvoie la somme de tous ses cœfficients. Ici, on considère le tableau `np.arange(5)` $= [0, 1, 2, 3, 4]$ dont la somme des cœfficients vaut 10.

- En déduire un appel permettant de calculer S_{10}.

`np.sum(premSuiteU(10))`. On retrouve bien $S_{10} \simeq 1.4988738166607394$.

On souhaite maintenant utiliser les opérations algébriques sur les tableaux pour construire le tableau contenant les $n+1$ premiers termes de la suite (u_n), sans passer par une structure itérative.

- Que réalisent les appels `np.arange(11)`, `np.arange(11)**2` et `3**np.arange(11)` ?

- L'appel `np.arange(11)` renvoie le tableau contenant les entiers compris entre 0 et 10 (les 11 premiers entiers).
- L'appel `np.arange(11)**2` renvoie le tableau contenant les carrés des entiers compris entre 0 et 10.
- L'appel `3**np.arange(11)` renvoie le tableau contenant les 11 premières puissances de 3.

- En déduire un appel renvoyant un tableau contenant $u_0, \dots, u_{10}$.

`np.arange(11)**2 / 3**np.arange(11)`

- A l'aide de ce qui précède, écrire une nouvelle fonction `premSuiteUopal` qui :
  - × prend en paramètre un entier `n`,
  - × renvoie un tableau contenant les $\texttt{n}+1$ premiers termes de la suite (u_n).

```python
import numpy as np
def premSuiteUopal(n)
    U = np.arange(n+1)
    return U**2 / 3**U
```

II.2. Calcul de T_n

Il s'agit ici d'illustrer le cas où la suite (v_n) est donnée sous forme récurrente.

II.2.a) Méthode itérative

► Écrire une fonction `calculTn` qui :

× prend en paramètre un entier `n`,

× renvoie la valeur de T_n,

On pourra utiliser une variable auxiliaire `v` afin de calculer les différents termes de la suite (v_n).

```
import numpy as np
def calculTn(n):
    T = 0
    v = 1
    T = T + v
    for i in range(n):
        v = 2 * v / (np.exp(v) + np.exp(-v))
        T = T + v
    return T
```

► Que vaut T_0 ? T_{100} ? T_{10000} ?

On trouve : $T_0 = 1$, $T_{100} \simeq 18.18$, $T_{10000} \simeq 197.85$.

II.2.b) Utilisation des fonctionnalités de la bibliothèque numpy

► Écrire une fonction `premSuiteV` qui :

× prend en paramètre un entier `n`,

× renvoie un tableau contenant les $\mathtt{n}+1$ premières valeurs de la suite (v_n).

On utilisera la bibliothèque numpy.

```
import numpy as np
def premSuiteV(n):
    V = np.zeros(n+1)
    V[0] = 1
    for i in range(n):
        V[i+1] = 2 * V[i] / (np.exp(V[i]) + np.exp(-V[i]))
    return V
```

► En déduire un appel permettant de calculer T_{10}.

```
np.sum(premSuiteV(10))
```

III. Calcul des n premières sommes partielles

III.1. Calcul des n premières sommes partielles de la série $\sum u_n$

- Soit $n \in \mathbb{N}$. Quel lien y a-t-il entre S_n et S_{n+1} ?

$$S_{n+1} = S_n + \frac{(n+1)^2}{3^{n+1}}$$

- En tirant profit de l'égalité précédente, écrire une fonction `premSn` qui :
  - × prend en paramètre une variable `n`,
  - × renvoie un tableau contenant les $\mathtt{n}+1$ premières sommes partielles de la série $\sum u_n$, *i.e.* les sommes $S_0, S_1, \dots, S_{\mathtt{n}}$.

  On ne devra pas effectuer d'appel à `calculSn` mais on pourra s'inspirer de son code.

```
import numpy as np
def premSn(n):
    tabS = np.zeros(n+1)
    tabS[0] = 0
    for i in range(n):
        tabS[i+1] = tabS[i] + (i+1)**2 / 3**(i+1)
    return tabS
```

- Comme précédemment, on aurait aussi pu tirer parti des fonctionnalités de la bibliothèque numpy. Que réalise l'appel `np.cumsum(np.arange(6))` ? Détailler le rôle de la fonction `np.cumsum`.

  - La fonction `np.cumsum` (*Cumulative Sum*) prend en paramètre un tableau `u` et renvoie un vecteur `v` de même taille dont le $i^{\text{ème}}$ cœfficient est la somme des i premiers éléments de `u`.
  - Ainsi, `np.cumsum(np.arange(6))` renvoie le vecteur $[0, 1, 3, 6, 10, 15]$.

- Définir une fonction nommée `premSnbis`, utilisant la fonction `np.cumsum` et la fonction `premSuiteU`, qui
  - × prend en paramètre une variable `n`,
  - × renvoie un tableau contenant les $\mathtt{n}+1$ premières sommes partielles de la série $\sum u_n$, *i.e.* les sommes $S_0, S_1, \dots, S_{\mathtt{n}}$.

```
import numpy as np
def premSnbis(n):
    return np.cumsum(premSuiteU(n))
```

III.2. Calcul des n premières sommes partielles de la série $\sum v_n$

- Soit $n \in \mathbb{N}$. Quel lien y a-t-il entre T_n et T_{n+1} ?

$T_{n+1} = T_n + v_{n+1}$

- En tirant profit de l'égalité précédente, écrire une fonction `premTn` qui :
  - × prend en paramètre une variable `n`,
  - × renvoie un tableau contenant les $\mathtt{n}+1$ premières sommes partielles de la série $\sum v_n$, *i.e.* les sommes $T_0, T_1, \dots, T_{\mathtt{n}}$.

  On ne devra pas effectuer d'appel à `calculTn` mais on pourra s'inspirer de son code.

```python
import numpy as np
def premTn(n):
    tabT = np.zeros(n+1)
    v = 1
    tabT[0] = 1
    for i in range(n):
        v = 2 * v / (np.exp(v) + np.exp(-v))
        tabT[i+1] = tabT[i] + v
    return tabT
```

À retenir : on a étudié deux méthodes en **Python** pour obtenir les éléments de (S_n).

1) La suite des sommes partielles étant une suite (grande nouvelle), on peut se servir des procédés vus en TP1.

2) On peut aussi préférer créer le vecteur des premiers éléments de la suite (u_n) et utiliser les fonctions `np.sum` ou `np.cumsum` suivant ce que l'on cherche à obtenir.

IV. Tracé des sommes partielles de $\sum u_n$

- Compléter le programme suivant afin qu'il permette d'effectuer le tracé des 51 premiers éléments de (S_n). On exécutera ce programme.

```python
import matplotlib.pyplot as plt
import numpy as np
N=51
U = np.arange(N)**2 / 3**np.arange(N)
tabS = ________________
plt.plot(tabS,'+')
```

- Quelle conjecture peut-on émettre sur la nature de la série $\sum u_n$?

D'après la représentation graphique, on peut émettre l'hypothèse que $\sum u_n$ est une série convergente, de somme $\frac{3}{2}$.

V. Suite des sommes partielles aux concours

V.1. EML 2015

On considère l'application $f : \mathbb{R} \to \mathbb{R},\ x \mapsto f(x) = x^3\, e^x$
et la suite réelle $(u_n)_{n\in\mathbb{N}}$ *définie par :* $u_0 = 1$ *et* $\forall n \in \mathbb{N}, u_{n+1} = f(u_n)$.
Il était demandé de démontrer que la série $\sum\limits_{n\geqslant 1} \dfrac{1}{f(n)}$ converge (de somme S) et que :

$$\forall n \in \mathbb{N}^*,\ \left| S - \sum_{k=1}^{n} \frac{1}{f(k)} \right| \leqslant \frac{1}{(\mathrm{e}-1)\,\mathrm{e}^n}$$

► *En déduire une fonction* ***Python*** *qui calcule une valeur approchée de* S *à* 10^{-4} *près.* (*cf* TP2)

- Si on sait : $\dfrac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} \leqslant 10^{-4}$, on obtient par transitivité : $|S - S_n| \leqslant 10^{-4}$.

  L'idée est donc de trouver le premier entier tel que : $\dfrac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} \leqslant 10^{-4}$.

  On peut démontrer que cet entier vaut : $\lceil 4\ \ln(10) - \ln(\mathrm{e}-1) \rceil$ $(= 9)$ et ainsi, utiliser une boucle `for` pour calculer S_9 qui fournit l'approximation souhaitée.

```python
import numpy as np
def calcApprochS():
    n = int(np.ceil(4 * np.log(10) - np.log(np.exp(1)-1)))
    S = 0
    for k in range(1,n+1):
        S = S + 1 / ((k**3) * np.exp(k))
    return S
```

- Le deuxième choix est de calculer les valeurs successives de (S_n) tant que $\dfrac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} \leqslant 10^{-4}$ **n'est pas** vérifiée *i.e.* tant que $\dfrac{1}{(\mathrm{e}-1)\,\mathrm{e}^n} > 10^{-4}$.

```python
import numpy as np
def calcApprochS():
    n = 1
    S = 1 / np.exp(1)
    while 1 / ((np.exp(1)-1) * np.exp(n)) > 10**(-4):
        n = n + 1
        S = S + 1 / ((n**3) * np.exp(n))
    return S
```

V.2. ECRICOME 2018

Pour tout entier naturel n non nul, on pose : $u_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n)$.

► *Écrire une fonction d'en-tête* `def u(n):` *qui prend en argument un entier naturel n non nul et qui renvoie la valeur de u_n.*

```
1 import numpy as np
2 def u(n):
3     S = 0
4     for k in range(1,n+1)
5         S = S + 1/k
6 return S - np.log(n)
```

Détaillons les différents éléments de ce code :

× en ligne 3, on crée la variable `S` dont le but est de contenir, en fin de programme $\sum_{k=1}^{n} \frac{1}{k}$.
Cette variable `S` est donc initialisée à 0.

× de la ligne 4 à la ligne 5, on met à jour la variable `S` à l'aide d'une boucle.
Pour ce faire, on ajoute au $k^{\text{ème}}$ tour de boucle la quantité $\frac{1}{k}$.
Ainsi, `S` contient bien $\sum_{k=1}^{n} \frac{1}{k}$ en sortie de boucle.

× en ligne 6, on renvoie la valeur $u_n = \sum_{k=1}^{n} \frac{1}{k} - \ln(n)$.

Commentaire

Pour le calcul de la somme $\sum_{k=1}^{n} \frac{1}{k}$, on peut aussi tirer profit des fonctionnalités de la bibliothèque numpy :

```
S = np.sum(1/np.arange(1,n+1))
```

Pour bien comprendre cette instruction, rappelons que :

× l'instruction `np.arange(1,n+1)` permet de créer le tableau $\begin{pmatrix} 1 & 2 & \dots & n \end{pmatrix}$.

× l'opérateur `/` permet d'effectuer la division terme à terme. Ainsi, l'instruction `1 / np.arange(1,n+1)` permet de créer le tableau $\begin{pmatrix} \frac{1}{1} & \frac{1}{2} & \dots & \frac{1}{n} \end{pmatrix}$.

× la fonction `np.sum` permet de sommer tous les coefficients d'un tableau.

On obtient donc bien la somme à calculer par cette méthode.

V.3. ECRICOME 2015

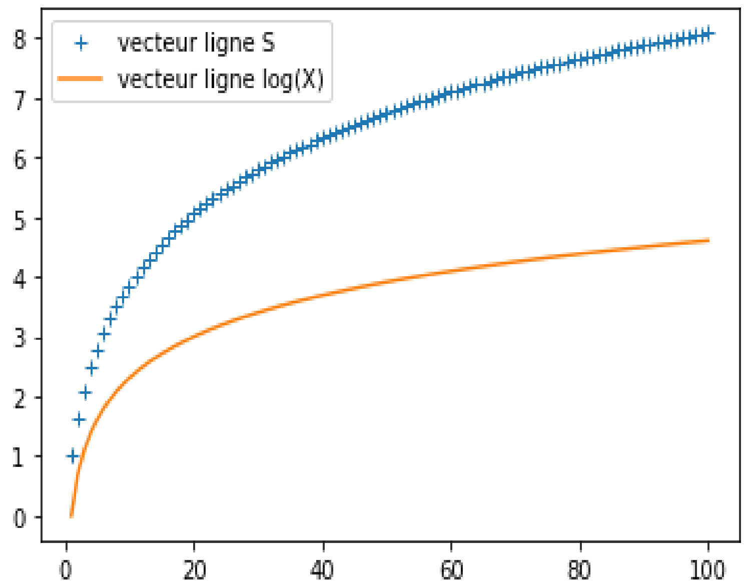
Au TP1, on a déjà introduit l'épreuve `ECRICOME 2015` qui commençait par l'étude d'une suite récurrente $(u_n)_{n\in\mathbb{N}^*}$, définie par une relation de récurrence de type $u_{n+1} = F(u_n)$.
La première question, consistait à compléter le programme permettant le calcul et le tracé des 100 premiers éléments de (u_n). On rappelle ce programme ci-dessous.

```
import numpy as np
import matplotlib as plt
U = np.zeros(100)
U[0] = 1
for n in range(99):
    U[n+1] = 1 - np.exp(-U[n])
X = [n for n in range(1,101)]
plt.plot(X,U,'+')
```

- ► *On modifie le programme précédent en remplaçant la dernière ligne par :*

```
S = np.cumsum(U)
plt.plot(X,S,'+', label='vecteur ligne S')
plt.plot(X,np.log(X), label='vecteur ligne log(X)')
plt.legend(loc='best')
```

Le programme ci-dessus permet d'obtenir la représentation graphique suivante :



- ► *Que représente le vecteur-ligne* `S` *?*
Quelle conjecture pouvez-vous émettre sur la nature de la série de terme général u_n ?

> - Le vecteur `U` contient les 100 premiers éléments de la suite (u_n). L'opérateur `cumsum` permet de calculer la somme cumulée de ce vecteur. Ainsi, `S` contient les 100 premières sommes partielles de la série $\sum u_n$.
> - D'après le tracé obtenu, on peut émettre l'hypothèse que la série $\sum u_n$ est divergente. Plus précisément, que $S_n \underset{n\to+\infty}{\longrightarrow} +\infty$.

► *À l'aide de la question **2.h)*** (consistait à démontrer : $\forall n \in \mathbb{N}^*,\ u_n \geqslant \frac{1}{n}$) *établir la nature de la série de terme général u_n.*

> On sait :
>
> × $\forall n \in \mathbb{N}$, $0 \leqslant \frac{1}{n} \leqslant u_n$.
>
> × La série $\sum\limits_{n \geqslant 1} \frac{1}{n}$ est une série de Riemann d'exposant 1 ($1 \not> 1$).
>
> Ainsi, la série $\sum\limits_{n \geqslant 1} \frac{1}{n}$ est divergente.
>
> Donc, par critère de comparaison des séries à termes positifs, la série $\sum u_n$ diverge.