TP1 : Calcul du $m^{\text{ème}}$ terme, des m premiers termes d'une suite

(Révisions sur la structure itérative `for`*)*

Objectifs du TP : manipuler la boucle for, calculer les termes d'une suite, faire le tour des questions **Python** classiques aux concours sur les suites.

On considère la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par : $\begin{cases} \forall n \in \mathbb{N}^*,\ u_{n+1} = 2\,u_n + n + 1 \\ u_1 = 1 \end{cases}$

- Calculer à la main les termes u_2, u_3 et u_4.

> $u_1 = 1$ donc $u_2 = 4$ donc $u_3 = 11$ donc $u_4 = 26$.

Vous vous servirez de ces valeurs pour vérifier vos programmes.

I. Calcul du $m^{\text{ème}}$ terme d'une suite

Dans cette partie, on souhaite uniquement calculer le $m^{\text{ème}}$ terme de la suite (u_n), sans garder en mémoire les termes précédents.

I.1. Révision des scripts avec dialogue utilisateur

- Compléter le script **Python** suivant, pour qu'il renvoie la valeur de u_m (*i.e.* le $m^{\text{ème}}$ terme de la suite (u_n)) lorsque l'utilisateur rentre m dans la console :

```
1 m = int(input('Rentrez la valeur de m :'))
2 u = 1
3 for i in __range(1,m)__:
4     u = __2 * u + i + 1__
5 print(u)
```

- Pourquoi a-t-on utiliser `int` à la première ligne ?

> `input` renvoie toujours une chaîne de caractère. Pour pouvoir utiliser la valeur de m dans la boucle `for`, il faut au préalable convertir cette chaîne de caractère en nombre entier.

- Vérifier que les résultats renvoyés sont cohérents avec les premiers termes de la suite calculés au début du TP.
- Calculer u_{12} et u_{20} à l'aide du script précédent.

> On obtient $u_{12} = 8178$ et $u_{20} = 2097130$.

I.2. Révision des fonctions

- Ecrire une fonction en **Python**, nommée `emeSuiteU` qui (en reprennant le fonctionnement du script précédent) :
  - × prend en paramètre un entier `m`,
  - × retourne $u_{\texttt{m}}$.

  Sauvegarder la fonction dans un fichier **Python** sous le nom `emeSuiteU.py`.

```
1  def emeSuiteU(m):
2      u = 1
3      for i in range(1,m):
4          u = 2 * u + i + 1
5      return u
```

- Calculer u_7 et u_{15} à l'aide de la fonction précédente.

On obtient $u_7 = 247$ et $u_{15} = 65519$.

- Selon vous, quels sont les avantages de la représentation sous forme de script avec dialogue utilisateur ? Sous forme de fonction ?

- D'un point de vue utilisateur, la version avec dialogue utilisateur, plus ludique, peut être plus appréciée. Elle peut être utile sur un site, par exemple lorsque l'on souhaite renseigner ses notes d'écrits pour prévoir ses admissibilités.
- D'un point de vue algorithmique, il faut privilégier la version sous forme de fonction. L'avantage est que le calcul réalisé par `emeSuiteU` peut facilement être utilisé ailleurs (notamment dans une autre fonction) : il suffit pour ce faire d'écrire l'appel `emeSuiteU(m)` (avec `m` choisi correctement). De plus, il n'y a pas besoin de recompiler le code à chaque utilisation de la fonction.

II. Calcul des m premiers termes d'une suite

Dans cette partie, on souhaite calculer les termes $u_1, u_2, \ldots, u_m$ de la suite (u_n).

II.1. Révision sur les tableaux

- Rappeler à quoi sert la commande `import numpy as np`.

`numpy` est une bibliothèque permettant de manipuler des tableaux (plus généralement des matrices) et de faire des calculs numériques sur de grands ensembles de données.

- Ecrire une fonction `premSuiteUTab` qui :
  - × prend en paramètre une variable `m`,
  - × retourne en sortie un tableau contenant les `m` premiers termes de la suite (u_n).

  On pourra créer un tableau (*i.e.* un vecteur) `T` de taille `m` initialement rempli de zéros, puis le transformer à l'aide d'une boucle `for`. Sauvegarder ce programme sous le nom `premSuiteU_tableau.py`.

```
import numpy as np
def premSuiteUTab(m):
    T = np.zeros(m)
    T[0] = 1
    for i in range(1,m):
        T[i] = 2 * T[i-1] + i + 1
    return T
```

- Calculer les 5 premiers termes de la suite à l'aide de la fonction précédente.

On obtient : $[1, 4, 11, 26, 57]$.

II.2. Révisions sur les listes

- Etant donné une liste `L`, quelle commande permet de rajouter à la fin de la liste `L` un élément `a` ?

`L.append(a)`
Attention, il ne faut pas écrire `L = L.append(a)` car cela conduirait à une erreur.

- Ecrire une fonction `premSuiteUListe` qui :
  - × prend en paramètre une variable `m`,
  - × retourne en sortie une liste contenant les `m` premiers termes de la suite (u_n).

  On pourra créer une liste `L` contenant initialement u_1, puis la transformer à l'aide d'une boucle `for`. Sauvegarder ce programme sous le nom `premSuiteU_liste.py`.

```
def premSuiteUListe(m):
    L = [1]
    for i in range(1,m):
        L.append(2 * L[i-1] + i + 1)
    return L
```

- Calculer les 5 premiers termes de la suite à l'aide de la fonction précédente.

On obtient : $[1, 4, 11, 26, 57]$.

- Rappeler à quoi sert la commande `import matplotlib.pyplot as plt`.

> `matplotlib.pyplot` est une bibliothèque permettant de faire des représentations graphiques.

- Compléter le script **Python** suivant pour qu'il trace les 30 premiers termes de la suite (u_n) sous la forme de ronds rouges :

```
1 import matplotlib.pyplot as plt
2 m = __30__
3 X = [n for n in range(1,m+1)]
4 plt.plot__(X,premSuiteUListe(m),'ro')__
5 plt.title(f'Tracé des {m} premiers termes de la suite')
```

- On rajoute une ligne au script précédent :

```
1 import matplotlib.pyplot as plt
2 m = __30__
3 X = [n for n in range(1,m+1)]
4 plt.plot__(X,premSuiteUListe(m),'ro')__
5 plt.plot(premSuiteUListe(m),'bx')
6 plt.title(f'Tracé des {m} premiers termes de la suite')
```

Que voit-on sur le nouveau tracé ? Pourquoi était-il important de construire la liste X pour effectuer le tracé ? Quelle fonctionnalité de Python est mise en valeur ici ?

> On voit un décalage dans les indices, le tracé en bleu commence à 0.
> La liste X était utile pour indiquer les abscisses du tracé. Le tracé doit commencer à 1 et pas à 0.
> **Python** numérote à partir de 0 par défaut.

- Quelle conjecture peut-on émettre sur la limite de la suite (u_n) ?

> Au vu du graphique obtenu, on peut conjecturer que la suite (u_n) tend vers $+\infty$.

- (à la maison) Démontrer par récurrence que : pour tout $n \in \mathbb{N}$, $u_n \geqslant 2^{n-1}$.

III. Suites $u_{n+1} = f(u_n)$ aux concours

III.1. ECRICOME - 2015

L'épreuve **ECRICOME - 2015** commençait par l'étude d'une suite récurrente de type $u_{n+1} = F(u_n)$. La fonction F est définie comme suit :

$$F(x) = \begin{cases} 0 & \text{si } x < 0, \\ 1 - \mathrm{e}^{-x} & \text{si } x \geqslant 0 \end{cases}$$

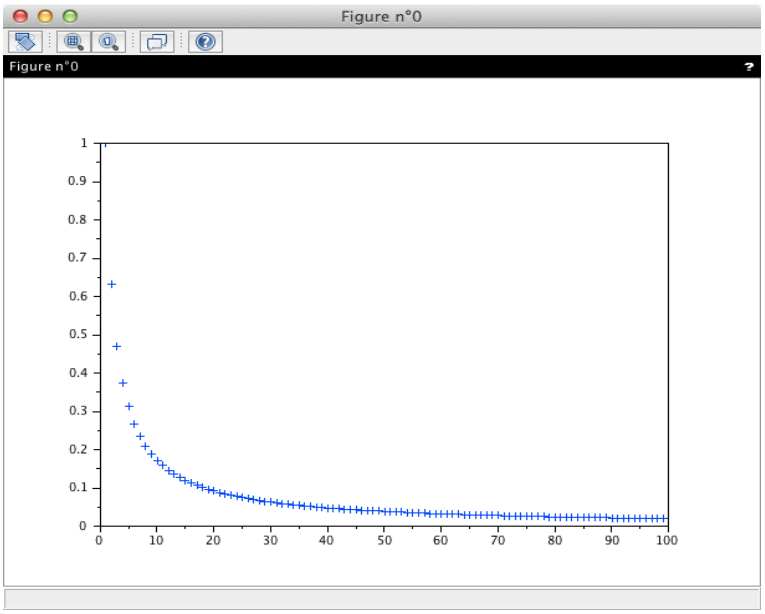
On considère la suite $(u_n)_{n\geqslant 1}$ définie par $u_1 = 1$ et pour tout $n \in \mathbb{N}^*$ par : $u_{n+1} = F(u_n)$.
On admet que : $\forall n \geqslant 1,\ u_n > 0$.

- Compléter le programme **Python** suivant qui permet de représenter les cent premiers termes de la suite $(u_n)_{n\geqslant 1}$:

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

```
U[n+1] = 1 - np.exp(-U[n])
```
(lorsqu'il n'y a qu'une ligne à compléter, il n'y a pas lieu de recopier tout le programme)

- *Le programme précédent complété permet d'obtenir la représentation graphique suivante.*



Quelle conjecture pouvez-vous émettre sur la monotonie et la limite de la suite $(u_n)_{n\geqslant 1}$?

D'après la représentation graphique, la suite (u_n) semble être convergente, de limite nulle.

III.2. ESSEC II - 2016

L'épreuve **ESSEC II - 2016** comportait une unique question d'informatique [1] qui consistait au codage d'une suite récurrente définie comme suit :

$$\begin{cases} u_0 = 1 \\ u_n = u_{n-1}\, p_1 + \ldots + u_0\, p_n \end{cases}$$

▶ *En **Python**, soit $P = [p_1, p_2, \ldots, p_n]$ la liste telle que $P(j) = p_{j+1}$ pour j dans $[\![0, n-1]\!]$. Écrire une fonction en **Python**, nommée* `calcSuiteU`*, qui calcule u_n à partir de P.*

Remarque

Cette question est bien plus compliquée que ce que laisse entrevoir son énoncé. Détaillons pourquoi.

1) En terme de structures de données :

La suite (u_n) est une suite récurrente dont le $n^{\text{ème}}$ terme dépend de `TOUS` les précédents.
Ce type de suite est bien plus délicate à traiter que les suites récurrentes d'ordre 1 (dont la relation de récurrence s'écrit $u_{n+1} = f(u_n)$) ou d'ordre 2 ($u_{n+1} = g(u_n, u_{n-1})$).

Pour calculer le terme d'indice n, il faut avoir accès aux termes d'indice $0, \ldots, n-1$ de la suite.
↪ il faut donc se servir d'un vecteur (ou d'une liste) pour stocker au fur et à mesure ces valeurs.

2) En terme d'indices :

La suite (u_n) est définie à partir du rang 0 et la numérotation des listes (ou des vecteurs) commence à 0 en **Python**. Il n'y a donc pas de danger ici. Par contre, il n'en est pas de même pour le vecteur P. De plus, la formule de récurrence mélange ordre croissant pour les termes de (u_n) et décroissant pour ceux de (p_n).

3) En terme de paramètre :

La fonction consiste à calculer le $n^{\text{ème}}$ terme de la suite (u_n) mais n n'est pas annoncé comme paramètre de la fonction. En effet, le seul paramètre annoncé pour la fonction est le vecteur P. C'est la taille de ce vecteur qui nous fournit la valeur de n.

Pour ce type de questions, il est conseillé de commencer par écrire au brouillon les premières étapes de l'algorithme. Autrement dit, d'effectuer le calcul de u_0, u_1, u_2, u_3, afin de comprendre le mécanisme de calcul.

```python
import numpy as np
def calcSuiteU(P):
    n = len(P) # Récupère la taille de la liste P
    U = np.zeros(n+1) # Vecteur de taille n+1 rempli de zéros
    U[0] = 1 # Initialisation
    # Boucle qui remplit le vecteur U avec les termes de la suite
    for k in range(n):
        S = 0
        for i in range(k+1): # Boucle qui calcule une somme
            S = S + U[k-i] * P[i]
        U[k+1] = S
    return U[n]
```

En fin de programme, la variable `U[n]` contient u_n. Comme l'exige l'énoncé, c'est cette valeur qui est renvoyée et pas le vecteur `U` des $n+1$ premiers termes de la suite (u_n).

[1] Ce qui représente une nette amélioration par rapport à l'épreuve 2015.

III.3. ECRICOME - 2018

On considère les matrices : $A = \begin{pmatrix} 2 & 1 & -2 \\ 0 & 3 & 0 \\ 1 & -1 & 5 \end{pmatrix}$ et $B = \begin{pmatrix} 1 & -1 & -1 \\ -3 & 3 & -3 \\ -1 & 1 & 1 \end{pmatrix}$.

On pose $X_0 = \begin{pmatrix} 3 \\ 0 \\ -1 \end{pmatrix}$, $X_1 = \begin{pmatrix} 3 \\ 0 \\ -2 \end{pmatrix}$, et pour tout entier naturel n : $X_{n+2} = \frac{1}{6}\, A\, X_{n+1} + \frac{1}{6}\, B\, X_n$.

a) Compléter la fonction ci-dessous qui prend en argument un entier n supérieur ou égal à 2 et qui renvoie la matrice X_n :

```
1  import numpy as np
2  def CalcVecteur(n):
3      A = np.array([[2,1,-2], [0,3,0], [1,-1,5]])
4      B = np.array([[1,-1,-1], [-3,3,-3], [-1,1,1]])
5      Xold = [3,0,-1]
6      Xnew = [3,0,-2]
7      for i in range(2,n+1):
8          Aux = __Xold__
9          Xold = __Xnew__
10         Xnew = ___(1/6)*(A @ Xold) + (1/6)*(B @ Aux)___
11     return __Xnew__
```

Détaillons les différents éléments présents dans cette fonction.

- En ligne 3 et 4, on stocke les matrices A et B dans les variables `A` et `B`.
- En ligne 5 et 6, on définit les variables `Xold` et `Xnew`.
  Ces deux variables sont initialement affectées aux valeurs X_0 et X_1.
- De la ligne 7 à la ligne 10, on met à jour les variables `Xold` et `Xnew` de sorte à ce qu'elles contiennent les valeurs successives de la suite matricielle (X_n).

```
7      for i in range(2,n+1):
8          Aux = Xold
9          Xold = Xnew
10         Xnew = (1/6)*(A @ Xold) + (1/6)*(B @ Aux)
```

- A la fin de la boucle, la variable `Xnew` contient le vecteur X_n, il n'y a plus qu'à retourner `Xnew`.

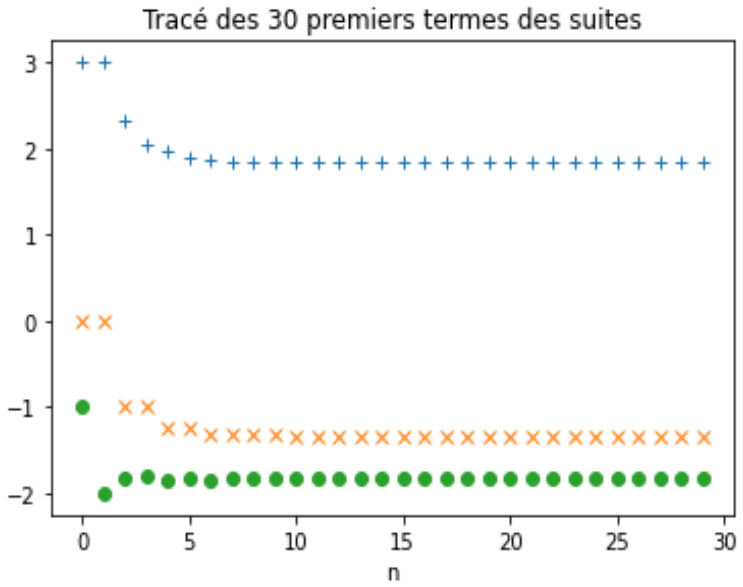
```
11     return Xnew
```

- Dans l'exercice, il était noté $X_n = \begin{pmatrix} \alpha_n \\ \beta_n \\ \gamma_n \end{pmatrix}$ et on démontrait, pour tout $n \in \mathbb{N}$:

$$\alpha_n = \frac{11}{6} + \frac{2}{3}\left(-\frac{1}{2}\right)^n + \left(\frac{1}{2}\right)^{n-1} - \frac{3}{2}\left(-\frac{1}{3}\right)^n$$

$$\beta_n = \left(\frac{1}{2}\right)^{n-1} - \frac{2}{3}\left(-\frac{1}{2}\right)^n - \frac{4}{3} \quad \text{et} \quad \gamma_n = -\frac{11}{6} - \frac{2}{3}\left(-\frac{1}{2}\right)^n + \frac{3}{2}\left(-\frac{1}{3}\right)^n$$

b) La fonction précédente a été utilisée dans un script permettant d'obtenir graphiquement les valeurs de α_n, β_n et γ_n en fonction de n.



Associer chacune des trois représentations graphiques à chacune des suites $(\alpha_n)_{n\in\mathbb{N}}$, $(\beta_n)_{n\in\mathbb{N}}$, $(\gamma_n)_{n\in\mathbb{N}}$ en justifiant votre réponse.

Soit $n \in \mathbb{N}$.

- D'après les formules admises :

$$\beta_n = \left(\frac{1}{2}\right)^{n-1} - \frac{2}{3}\left(-\frac{1}{2}\right)^{n} - \frac{4}{3} \underset{n\to+\infty}{\longrightarrow} -\frac{4}{3} \simeq -1,33$$

  En effet, comme $\left|\frac{1}{2}\right| < 1$ et $\left|-\frac{1}{2}\right| < 1$ alors : $\lim\limits_{n\to+\infty} \left(\frac{1}{2}\right)^{n-1} = 0$ et $\lim\limits_{n\to+\infty} \left(-\frac{1}{2}\right)^{n} = 0$.

- On démontre de même :

$$\alpha_n \underset{n\to+\infty}{\longrightarrow} \frac{11}{6} \simeq 1,8 \quad \text{et} \quad \gamma_n \underset{n\to+\infty}{\longrightarrow} -\frac{11}{6} \simeq -1,8$$

- Or, d'après la figure :
  - × la suite repérée par + semble converger vers un réel valant approximativement $1,8$.

    Cette représentation graphique correspond à la suite (α_n).
  - × la suite repérée par × semble converger vers un réel valant approximativement $-1,3$.

    Cette représentation graphique correspond à la suite (β_n).
  - × la suite repérée par • semble converger vers un réel valant approximativement $-1,8$.

    Cette représentation graphique correspond à la suite (γ_n).

III.4. EML - 2018

On pose : $u_0 = 4$ et $\forall n \in \mathbb{N},\ u_{n+1} = \ln(u_n) + 2$.

► Écrire une fonction **Python** d'en-tête `def suite(n):` qui, prenant en argument un entier n de $\mathbb{N}$, renvoie la valeur de u_n.

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

Expliquons un peu ce programme.
La variable `u` est créée pour contenir successivement les valeurs u_0, u_1, …, u_n.

- On initialise donc cette variable à $u_0 = 4$ avec la ligne 3 :

```
3      u = 4
```

- On met ensuite à jour `u` de manière itérative avec la ligne 5 :

```
5          u = np.log(u) + 2
```

Commentaire

Si on avait souhaité afficher tous les $n + 1$ premiers termes de la suite (u_n), on aurait modifié la fonction de la façon suivante :

```
1  import numpy as np
2  def suite(n):
3      U = [4]
4      for k in range(n):
5          U.append(np.log(U[k]) + 2)
6      return U
```

III.5. EDHEC - 2023

On considère la fonction f définie par :

$$\forall t \in \mathbb{R},\ f(t) = \frac{1}{1+\mathrm{e}^t}$$

On considère la suite $(u_n)_{n\in\mathbb{N}}$ définie par la donnée de $u_0 = 0$ et par la relation de récurrence $u_{n+1} = f(u_n)$, valable pour tout entier naturel n.

► Compléter la fonction **Python** ci-dessous afin qu'elle renvoie, pour une valeur donnée de n, la valeur de u_n à l'appel de `suite(n)` :

```
1  def suite(n):
2      u = ------
3      for k in range(1, n+1):
4          u = -----
5      return u
```

```
1  def suite(n):
2      u = 0
3      for k in range(1, n+1):
4          u = 1 / (1 + np.exp(u))
5      return u
```

Expliquons un peu ce programme.
La variable `u` est créée pour contenir successivement les valeurs u_0, u_1, …, u_n.

- On initialise donc cette variable à $u_0 = 0$ avec la ligne 2 :

```
2      u = 0
```

- On met ensuite à jour `u` de manière itérative avec la ligne 4 :

```
4          u = 1 / (1 + np.exp(u))
```