

TP1 : Calcul du $n^{\text{ème}}$ terme et
tracé des n premiers termes d'une suite
(Révisions sur la structure itérative *for*)

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

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

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

I. Calcul du $n^{\text{ème}}$ terme d'une suite récurrente (où $n \in \mathbb{N}^*$)

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

I.1. Relation de récurrence de la forme $u_{n+1} = f(u_n)$

I.1.a) Un exemple où le premier terme est u_1

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

- Renvoyer le $n^{\text{ème}}$ terme de la suite (u_n) , c'est renvoyer u_k pour quelle valeur de k ?

Puisque la suite (u_n) est initialisée par la donnée de u_1 , u_n est le $n^{\text{ème}}$ terme de la suite (u_n) .

- Compléter le script **Python** suivant pour qu'il affiche le $n^{\text{ème}}$ terme de la suite (u_n) lorsque l'utilisateur rentre n dans la console :

```
1 n = int(input('Rentrez la valeur de n :'))
2 u = 1
3 for i in range(n-1):
4     u = u + np.exp(-u)
5 print(u)
```

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

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

- En reprenant le fonctionnement du script précédent, écrire une fonction en **Python**, nommée `suiteU1(n)`, qui prend en argument l'entier n et renvoie le $n^{\text{ème}}$ terme de la suite (u_n) .

```

1  def suiteU1(n):
2      u = 1
3      for i in range(n-1):
4          u = u + np.exp(-u)
5      return u

```

- 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 la fonction `suiteU1` peut facilement être utilisé ailleurs (notamment dans une autre fonction) : il suffit pour ce faire d'écrire l'appel `suiteU1(n)` (avec n choisi correctement). De plus, il n'y a pas besoin de recompiler le code à chaque utilisation de la fonction.

I.1.b) Un exemple où le premier terme est u_0

On considère la suite $(u_n)_{n \in \mathbb{N}}$ définie par : $\begin{cases} u_0 = 2 \\ \forall n \in \mathbb{N}, u_{n+1} = u_n^2 + \ln(1 + u_n) \end{cases}$.

- Renvoyer le $n^{\text{ème}}$ terme de la suite (u_n) , c'est renvoyer u_k pour quelle valeur de k ?

Puisque la suite (u_n) est initialisée par la donnée de u_0 , u_{n-1} est le $n^{\text{ème}}$ terme de la suite (u_n) .

- Compléter la fonction **Python** `suiteU0(n)`, qui prend en argument l'entier n et renvoie le $n^{\text{ème}}$ terme de la suite (u_n) .

```

1  def suiteU0(n):
2      u = 2
3      for i in range(n-1):
4          u = u**2 + np.log(1+u)
5      return u

```



Dans la plupart des énoncés de concours, il n'est pas demandé de renvoyer « le $n^{\text{ème}}$ terme » mais plutôt u_n , qui peut être le $n^{\text{ème}}$ terme (si la suite commence par u_1) ou le $(n+1)^{\text{ème}}$ terme (si la suite commence par u_0).

- Compléter la fonction **Python** `suiteU0bis(n)`, qui prend en argument l'entier n et renvoie u_n .

```

1 def suiteU0bis(n):
2     u = 2
3     for i in range(n):
4         u = u**2 + np.log(1+u)
5     return u

```

I.2. Relation de récurrence de la forme $u_{n+1} = f_n(u_n)$

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

- 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.

- Quelle taille doit faire la boucle `for` pour calculer u_n à partir de u_1 ?

La boucle `for` doit être de taille $n - 1$ car la suite (u_n) est initialisée par la donnée de u_1 .

- Compléter les fonctions **Python** suivantes, qui prennent en argument l'entier n et renvoient u_n .

```

1 def suiteU(n):
2     u = 1
3     for i in range(n-1):
4         u = 2*u + (i+1) + 1
5     return u

```

```

1 def suiteUbis(n):
2     u = 1
3     for i in range(1,n):
4         u = 2*u + i + 1
5     return u

```



On remarque que n'importe quelle boucle `for` de taille $n - 1$ peut convenir, à condition d'écrire une formule de récurrence adaptée.

Au contraire, la formule de récurrence ne dépend pas du choix des bornes du `range` dans le cas où la fonction f ne dépend pas de n .

- Calculer u_4 , u_7 et u_{15} à l'aide des fonctions précédentes.

On retrouve $u_4 = 26$. On obtient $u_7 = 247$ et $u_{15} = 65519$.

II. Calcul des n premiers termes d'une suite récurrente

Dans cette partie, nous allons réviser l'utilisation des tableaux **Numpy** ainsi que des listes, pour stocker les n premiers termes d'une suite récurrente.

Pour éviter beaucoup de calculs inutiles, il ne faut pas utiliser les fonctions précédemment écrites, mais plutôt utiliser conjointement la structure de données avec une boucle `for`.

II.1. Révision sur les tableaux

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

- 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.

- Représenter schématiquement un tableau `Numpy` contenant les n premiers termes de la suite (u_n) . On fera également apparaître sous le tableau les indices des cases du tableau.

tableau <code>Numpy</code> :	u_1	u_2	u_3	$\dots$	u_n
indices du tableau <code>Numpy</code> :	0	1	2	$\dots$	$n - 1$

- Compléter la fonction **Python** `premSuiteU1(n)`, qui prend en argument l'entier n et renvoie un tableau `Numpy` contenant les n premiers termes de la suite (u_n) .

```

1 def premSuiteU1(n):
2     T = np.zeros(n)
3     T[0] = 1
4     for i in range(n-1):
5         T[i+1] = T[i] + np.exp(-T[i])
6     return T

```

II.2. Révisions sur les listes

On considère à nouveau la suite $(u_n)_{n \in \mathbb{N}}$ définie par : $\begin{cases} u_0 = 2 \\ \forall n \in \mathbb{N}, u_{n+1} = u_n^2 + \ln(1 + u_n) \end{cases}$.

- 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 s'agit d'une « méthode » et pas d'une « fonction ».
 Il ne faut pas écrire `L = L.append(a)` car cela conduirait à une erreur.

- Représenter schématiquement une liste contenant les n premiers termes de la suite (u_n) . On fera également apparaître sous la liste les indices des éléments de la liste.

liste :	u_0	u_1	u_2	$\dots$	u_{n-1}
indices Python :	0	1	2	$\dots$	$n - 1$

- Compléter la fonction **Python** `premSuiteU0(n)`, qui prend en argument l'entier n et renvoie une liste contenant les n premiers termes de la suite (u_n) . Cette fonction utilise une variable auxiliaire.

```
1 def premSuiteU0(n):
2     u = 2
3     L = [u]
4     for i in range(n-1):
5         u = u**2 + np.log(1+u)
6         L.append(u)
7     return L
```

- Compléter la fonction **Python** `premSuiteU0bis(n)`, qui prend en argument l'entier n et renvoie une liste contenant les n premiers termes de la suite (u_n) . Cette fonction n'utilise pas de variable auxiliaire.

```
1 def premSuiteU0bis(n):
2     L = [2]
3     for i in range(n-1):
4         L.append(L[i]**2 + np.log(1+L[i]))
5     return L
```

- Selon vous, quels sont les avantages de l'utilisation de tableaux par rapport aux listes et inversement ?

- L'avantage des tableaux est qu'ils utilisent moins de mémoire que les listes. Un deuxième avantage est qu'on peut faire des calculs directement sur les tableaux grâce aux fonctionnalités Numpy.
- L'avantage des listes est qu'elles ont une taille dynamique, que l'on peut changer en cours d'algorithme. C'est très pratique lorsqu'on ne connaît pas à l'avance le nombre de données que l'on veut stocker (par exemple si on veut stocker les résultats d'une succession de lancers d'une pièce de monnaie avant l'apparition du premier double pile).

III. Tracé des n premiers termes d'une suite récurrente

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

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

III.1. Un exemple où le premier terme est u_1

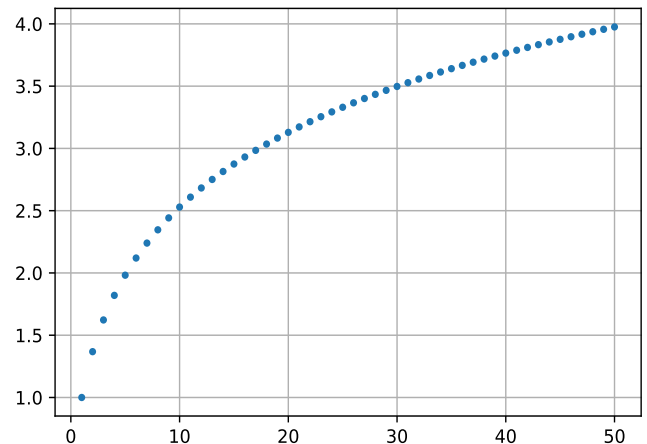
On considère à nouveau la suite $(u_n)_{n \in \mathbb{N}^*}$ définie par : $\begin{cases} u_1 = 1 \\ \forall n \in \mathbb{N}^*, u_{n+1} = u_n + e^{-u_n} \end{cases}$.

- En utilisant la fonction `premSuiteU1(n)`, compléter le script **Python** suivant pour qu'il trace les 50 premiers termes de la suite (u_n) . On affiche le tracé obtenu à sa droite.

```

1  n = 50
2  Xabs = range(1,n+1)
3  Yord = premSuiteU1(n)
4  plt.plot(Xabs, Yord, '.')
5  plt.grid()
6  plt.show()

```



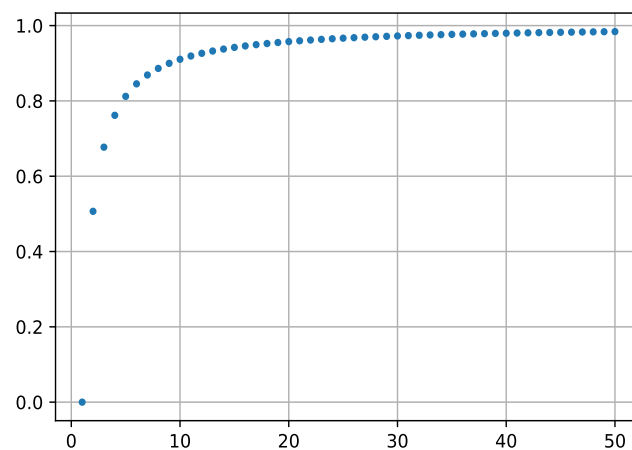
- On change la ligne 3 du script précédent et on la remplace par la ligne suivante :

```

4  Yord = np.log(np.arange(1,n+1)) / premSuiteU1(n)

```

On observe alors ce graphique :



Laquelle des quatre conjectures suivantes êtes vous amené à faire ?

❶ $u_n \xrightarrow{n \rightarrow +\infty} \ln(n)$

❷ $u_n \underset{n \rightarrow +\infty}{\sim} 1$

❸ $u_n \underset{n \rightarrow +\infty}{\sim} \ln(n)$

❹ $u_n \underset{n \rightarrow +\infty}{\sim} e^n$

III.2. Un exemple où le premier terme est u_0

On considère la suite $(u_n)_{n \in \mathbb{N}}$ définie par :
$$\begin{cases} u_0 = 1 \\ \forall n \in \mathbb{N}, u_{n+1} = \frac{1}{u_n^2} + \sqrt{u_n} \end{cases}$$

- Écrire une fonction **Python**, nommée `premSuiteV(n)`, qui prend en argument un entier naturel non nul n et qui renvoie une liste ou un tableau contenant les n premiers termes de la suite (v_n) .

```

1 def premSuiteV(n):
2     L = [1]
3     for i in range(n-1):
4         L.append(L[i]**(-2) + np.sqrt(L[i]))
5     return L

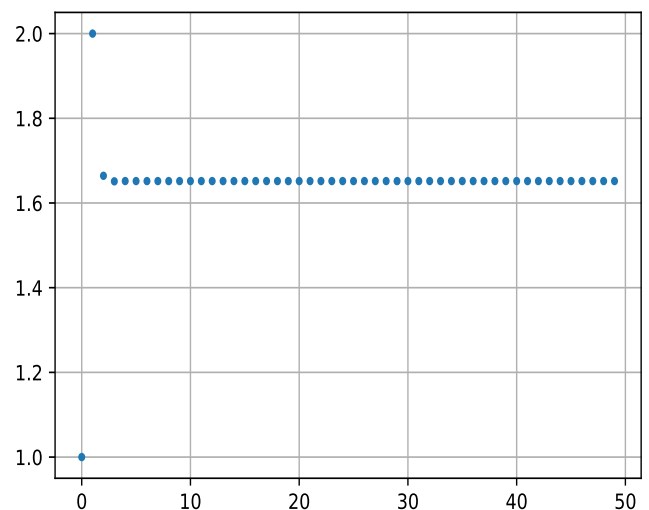
```

- En utilisant la fonction `premSuiteV(n)`, compléter le script **Python** suivant pour qu'il trace les 50 premiers termes de la suite (u_n) . On affiche le tracé obtenu à sa droite.

```

1 n = 50
2 Yord = premSuiteV(n)
3 plt.plot(Yord, '.')
4 plt.grid()
5 plt.show()

```



- Que pouvez-vous conjecturer à l'aide du tracé précédent ?

La suite (v_n) semble converger. Plus précisément, sa limite ℓ semble être comprise entre 1,6 et 1,7.

- Pourquoi était-il important de construire la liste `Xabs` pour effectuer le tracé dans la partie précédente et pas dans cette partie ? Quelle fonctionnalité de Python est mise en valeur ici ?

La liste `X` était utile pour indiquer les abscisses du tracé. Le tracé doit commencer à 1 et pas à 0 dans la partie précédente.

Python numérote à partir de 0 par défaut.

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

IV.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 :

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

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

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

```

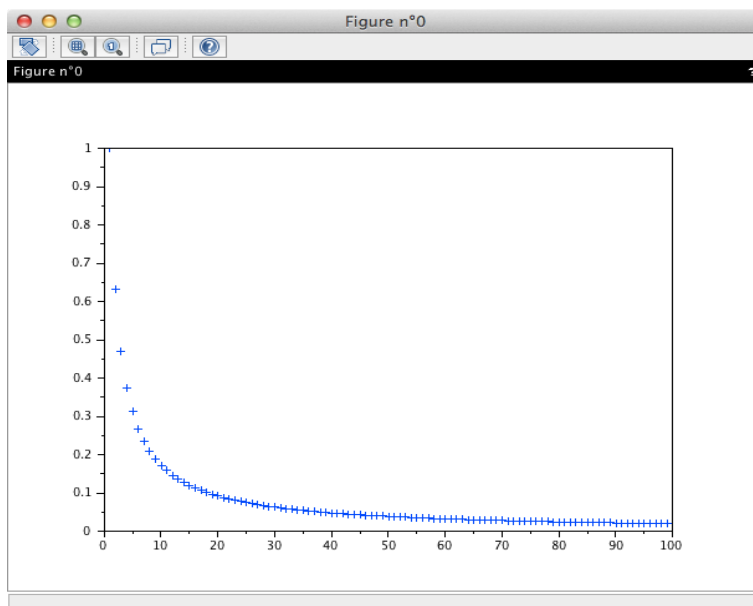
1 import numpy as np
2 import matplotlib as plt
3 U = np.zeros(100)
4 U[0] = 1
5 for n in range(99):
6     U[n+1] = 1 - np.exp(-U[n])
7 X = [n for n in range(1,101)]
8 plt.plot(X,U,'+')

```

`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, sauf indication contraire de l'énoncé)

- 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 $(u_n)_{n \geq 1}$?

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

IV.2. 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 $u_0, u_1, \dots, u_n$.

- 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 stocker tous les $n + 1$ premiers termes de la suite (u_n) dans une liste, 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
```

IV.3. EDHEC - 2023

On considère la fonction f définie par :

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

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 $u_0, u_1, \dots, u_n$.

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

```
2     u = 0
```

- On met ensuite à jour **u** n fois de manière itérative avec la ligne 4 :

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

IV.4. EDHEC - 2026

On considère un nombre réel a strictement supérieur à 1, ainsi que la suite $(u_n)_{n \in \mathbb{N}}$ définie par la donnée de $u_0 = a$ et par la relation de récurrence, valable pour tout entier naturel n :

$$u_{n+1} = u_n^2 - u_n + 1$$

- Compléter la fonction **Python** suivante afin qu'elle retourne la valeur de u_n pour des valeurs données de n et de a :

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

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

Expliquons un peu ce programme.

La variable `u` est créée pour contenir successivement les valeurs $u_0, u_1, \dots, u_n$.

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

```
2         u=a
```

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

```
4         u=u**2-u+1
```

IV.5. ECRICOME - 2018 (suite récurrence linéaire d'ordre 2 matricielle)

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)*np.dot(A, Xold) + (1/6)*np.dot(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)*np.dot(A, Xold) + (1/6)*np.dot(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

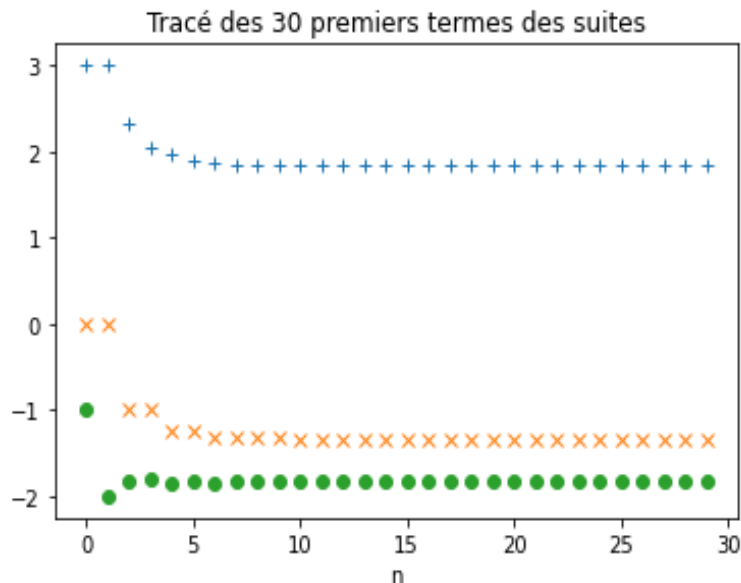
```

- 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 :

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

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

- On démontre de même :

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

- 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) .

IV.6. ESSEC II - 2016 (suite récurrente dépendant de tous les termes précédents)

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

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

- En **Python**, soit $P = [p_1, p_2, \dots, p_n]$ la liste telle que $P(j) = p_{j+1}$ pour j dans $\llbracket 0, n-1 \rrbracket$. 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 $0, \dots, n-1$ de la suite. $\hookrightarrow$ 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.

```

1  import numpy as np
2  def calcSuiteU(P):
3      n = len(P) # Récupère la taille de la liste P
4      U = np.zeros(n+1) # Vecteur de taille n+1 rempli de zéros
5      U[0] = 1 # Initialisation
6      # Boucle qui remplit le vecteur U avec les termes de la suite
7      for k in range(n):
8          S = 0
9          for i in range(k+1): # Boucle qui calcule une somme
10             S = S + U[k-i] * P[i]
11          U[k+1] = S
12      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⁾ Le nombre de questions d'informatique a beaucoup augmenté depuis la réforme de 2022.