

Pour les graphiques la bibliothèque à importer est `matplotlib.pyplot` que l'on importera ainsi :

```
import matplotlib.pyplot as plt
```

Pour représenter une liste `y` en fonction d'une liste `x` :

```
x = liste des abscisses (ou tableau)
y = liste des ordonnées (ou tableau)
plt.plot(x,y)
plt.show()
```

**Rappel...**

Par défaut, les points ainsi représentés sont reliés. Pour obtenir des points isolés, on ajoute un style en option, exemples :

- `plt.plot(x,y,'+')` : marque les points avec des +
- `plt.plot(x,y,'o')` : marque les points avec des •

Dans cette fiche, nous allons reprendre des questions classiques sur des suites et sommes : calculs de termes, représentations graphiques, calculs de sommes, création de listes de termes...

## I PETIT MÉLANGE...

On considère les suites  $(s_n)_{n \in \mathbb{N}}$ ,  $(t_n)_{n \in \mathbb{N}}$ ,  $(u_n)_{n \in \mathbb{N}}$ ,  $(v_n)_{n \in \mathbb{N}}$  et  $(w_n)_{n \in \mathbb{N}}$  définies par :

$$\forall n \in \mathbb{N}, s_n = \frac{n^2}{2^n}; \quad \begin{cases} t_0 = 1 \\ \forall n \in \mathbb{N}, t_{n+1} = e^{-t_n} \end{cases}; \quad \begin{cases} u_1 = 1 \\ \forall n \in \mathbb{N}^*, u_{n+1} = u_n + \frac{1}{n^2 u_n} \end{cases}; \quad \begin{cases} v_0 = v_1 = 1 \\ \forall n \in \mathbb{N}, v_{n+2} = v_{n+1} + 3v_n \end{cases}; \quad \forall n \in \mathbb{N}^*, w_n = \frac{(2n)!}{n^n}$$

- 1.a. Écrire une fonction `liste_suite_s` prenant en argument d'entrée un entier naturel `n` et renvoyant la liste composée des valeurs  $s_0$  à  $s_n$ .
- 1.b. Représenter alors les termes  $s_0$  à  $s_{10}$  sur un graphique.

```
1 import matplotlib.pyplot as plt
2
3 def liste_suite_s(n):
4     L=[k**2/(2**k) for k in range(0,n+1)]
5     return L
6
7 plt.plot(range(0,11),liste_suite_s(10),"+")
8 plt.show()
```

- 2.a. Écrire une fonction `suite_t` prenant en argument d'entrée un entier naturel `n` et renvoyant la valeur de  $t_n$ .
- 2.b. En utilisant la fonction `suite_t`, écrire une fonction `somme_t` prenant en argument d'entrée un entier naturel `n` et renvoyant la valeur de  $\sum_{k=0}^n t_k$ . Quel est l'inconvénient de cette méthode ? En proposer une autre.
- 2.c. Sans utiliser la fonction `suite_t`, écrire une fonction `liste_suite_t` prenant en argument d'entrée un entier naturel `n` et renvoyant la liste composée des valeurs  $t_0$  à  $t_n$ .

```
1 import numpy as np
2
3 def suite_t(n):
4     t=1
5     for k in range(1,n+1):
6         t=np.exp(-t)
7     return t
8
9 def somme_t(n):
10    L=[suite_t(k) for k in range(0,n+1)]
11    return sum(L)
12
13 def somme_t_bis(n):
14    t,S=1,1
15    for k in range(1,n+1):
16        t=np.exp(-t)
17        S=S+t
18    return S
19
```

```

20 def liste_suite_t(n):
21     t,T=1,[1]
22     for k in range(1,n+1):
23         t=np.exp(-t)
24         T.append(t)
25     return T
26
27 def liste_suite_t_bis(n):
28     T=[1]
29     for k in range(1,n+1):
30         t=np.exp(-T[k-1])
31         T.append(t)
32     return T

```

3. Écrire une fonction **suite\_u** prenant en argument d'entrée un entier naturel non nul  $n$  et renvoyant la valeur de  $u_n$ .

```

1 def suite_u(n):
2     u=1
3     for k in range(2,n+1):
4         u=u+1/((k-1)**2*u) #attention : k = rg du terme calculé
5     return u

```

4. Écrire une fonction **suite\_v** prenant en argument d'entrée un entier naturel  $n$  et renvoyant la valeur de  $v_n$ .

```

1 def suite_v(n):
2     if n==0 or n==1:
3         return 1
4     else:
5         u,v=1,1
6         for k in range(2,n+1): #k=rg du terme calculé
7             u,v=v,v+3*u
8     return v

```

5. A l'aide d'une liste définie en compréhension, écrire une fonction **suite\_w** prenant en argument d'entrée un entier naturel non nul  $n$  et renvoyant la valeur de  $w_n$ .

```

1 import numpy as np
2
3 def suite_w(n):
4     L=[k for k in range(1,2*n+1)]
5     N=np.prod(L)
6     return N/(n**n)

```

#### Aide

La commande `np.prod(L)` renvoie le produit des nombres de la liste `L`.

6. Représenter la fonction  $x \mapsto e^{-x}$ , la première bissectrice ainsi que les termes de la suite  $(t_n)_{n \in \mathbb{N}}$  sur un même graphique, en faisant apparaître les termes de  $(t_n)_{n \in \mathbb{N}}$  en une spirale.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x=np.linspace(0,1,100)
5 plt.plot(x,x)
6 plt.plot(x,np.exp(-x))
7
8 t=0
9 for k in range(1,11):
10     plt.plot([t,t],[t,np.exp(-t)], 'r')
11     plt.plot([t,np.exp(-t)],[np.exp(-t),np.exp(-t)], 'r')
12     t=np.exp(-t)
13
14 plt.show()

```

## II SÉRIES HARMONIQUES

7. Recopier et exécuter le programme suivant :

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 L=[1/k for k in range(1,101)]
5 S=np.cumsum(L)
6 x=np.linspace(1,100,10000)
7 plt.plot(range(1,101),S, 'r+')
8 plt.plot(x,np.log(x))
9 plt.show()
```

Que permet la commande `np.cumsum(L)` ? Que met en évidence le graphique obtenu ?

- La commande `np.cumsum(L)` prend en argument d'entrée une liste `L` et renvoie en sortie une liste `M` telle que, pour tout  $k \in [0, \text{len}(L)]$ , l'élément  $k$  de `M` est la somme des éléments 0 à  $k$  de `L`. C'est donc une commande très pratique pour créer une liste de termes d'une suite définie par une somme.

- Ce graphique met en évidence l'équivalent classique :  $\sum_{k=1}^n \frac{1}{k} \underset{n \rightarrow +\infty}{\sim} \ln(n)$ .

8. Adapter le programme précédent afin qu'il affiche les 100 premiers termes de la suite des sommes partielles de la série  $\sum_{n \geq 1} \frac{(-1)^n}{n}$ . Que peut-on conjecturer ? Comment pourrait-on démontrer cette conjecture ?

```
1 import matplotlib.pyplot as plt
2
3 L=[(-1)**k/k for k in range(1,101)]
4 S=np.cumsum(L)
5 plt.plot(range(1,101),S, 'r+')
6 plt.show()
```