

L'objectif de ce cours est de représenter graphiquement des fonctions et des suites sur Python. La bibliothèque à importer est `matplotlib.pyplot` que l'on importera ainsi :

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

Quand on souhaite représenter graphiquement une fonction ou une suite à la main, il nous faut un tableau de valeurs; autrement dit, les valeurs de $f(x)$ (ou u_n) pour un certain nombre de valeurs de x (ou de n). Commençons déjà par cela!

I OPÉRATIONS SUR LES TABLEAUX DE VALEURS...

La fonction `array`, disponible dans la bibliothèque `numpy`, permet de transformer une liste en un tableau sur lequel les opérations seront possibles...

Saisir les lignes suivantes dans la console :

```
>>> L=[1,2,3]
>>> type(L)
>>> L+L
>>> import numpy as np
>>> T=np.array(L)
>>> type(T)
>>> T+T
```

EN GROS...

`List` est une liste d'objets mathématiques... alors que `numpy.array` est une liste de nombres.

RAPPEL...

Si `L1` et `L2` sont deux listes, alors `L1+L2` est

Une fois une liste ainsi convertie en tableau, on peut effectuer un certain nombre d'opérations dont voici les résultats (`L1` est `array([a0, a1, ..., an])` et `L2` est `array([b0, b1, ..., bn])`).

Syntaxe Python	Résultat
<code>L1+L2</code>	
<code>L1+k</code>	
<code>k*L1</code>	
<code>L1*L2</code>	
<code>L1/L2</code>	
<code>L1**k</code>	
<code>L1**L2</code>	
<code>k**L1</code>	
<code>f(L1)</code>	

ATTENTION !

Pour effectuer des opérations entre `L1` et `L2`, elles doivent faire la même taille !

PETITE REMARQUE

Où `f` désigne une fonction existante de `numpy`, ou une fonction créée.

Exécuter les lignes suivantes et préciser ce que contient `y` :

Python 1 – Opérations

```
1 import numpy as np
2
3 x=np.array([-2+k*0.5 for k in range(0,9)])
4 y=x**2-np.exp(x)
```

COMMENTAIRES :

II REPRÉSENTATIONS GRAPHIQUES DE FONCTIONS

Pour commencer, il paraît insensé de devoir saisir à la main la liste des abscisses, surtout si l'on souhaite un graphique précis, et donc une liste relativement grande...

On retiendra donc les deux commandes suivantes :

- pour $a, b \in \mathbb{R}$ et $n \in \mathbb{N}$, la commande `np.linspace(a, b, n)` crée un **tableau de n valeurs équiréparties de a à b inclus**.
- pour $a, b, p \in \mathbb{R}$, la commande `np.arange(a, b, p)` crée un **tableau de valeurs de a inclus à b exclu avec un pas de p** .

Voici la structure ainsi que la syntaxe pour obtenir la courbe d'une fonction :

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

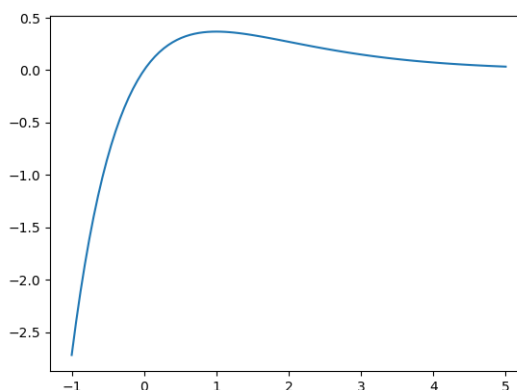
POUR INFO...

Les listes des abscisses et des ordonnées peuvent être soit du type `numpy.array`, soit du type `list`...

Exemple :

Python 2 – Courbe d'une fonction

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x=np.linspace(-1,5,100)
5 y=x*np.exp(-x)
6 plt.plot(x,y)
7 plt.show()
```



PETITE REMARQUE

Il est également possible de prédéfinir la fonction $x \mapsto xe^{-x}$ dans une fonction Python, puis de l'utiliser dans la ligne 5...

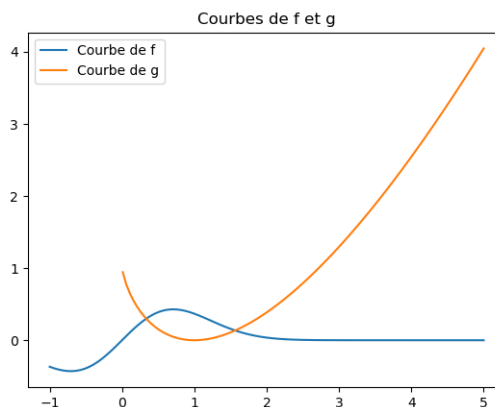
Les commandes qui suivent ne sont pas exigibles, mais elles peuvent parfois servir :

- `plt.grid()` : fait apparaître une grille sur le fond du repère
- `plt.axis('equal')` : rend le repère orthonormé
- `plt.axis([a,b,c,d])` : restreint le repère entre les abscisses a et b et les ordonnées c et d
- `plt.plot(x,y,label="nom de la courbe")`
- `plt.plot(x,y,'couleur')`, où `couleur` désigne la couleur voulue (ou son initiale)
- `plt.legend()` : affiche la légende

Voici comment représenter sur un même graphique les courbes des fonctions $f : x \mapsto xe^{-x^2}$ et $g : x \mapsto x \ln(x) - x + 1$:

Python 3 – Courbe de deux fonctions

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x=np.linspace(-1,5,100)
5 y=x*np.exp(-x**2)
6 plt.plot(x,y, label="Courbe de f")
7
8 x=np.linspace(0.01,5,100)
9 y=x*np.log(x)-x+1
10 plt.plot(x,y, label="Courbe de g")
11
12 plt.title("Courbes de f et g")
13 plt.legend()
14 plt.show()
```



III REPRÉSENTATIONS GRAPHIQUES DE SUITES

C'est sensiblement la même chose, à la différence majeure près que les abscisses sont des entiers naturels ; et que l'on représentera les points par des symboles + ou • par exemple.

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

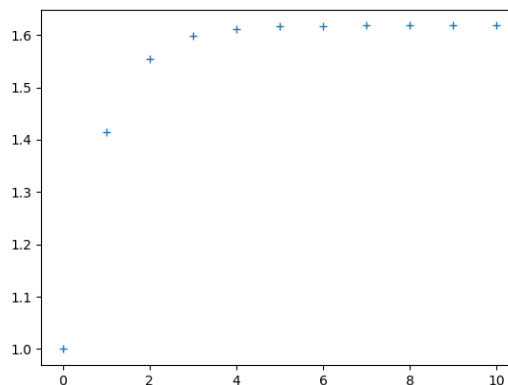
Pour des suites définies explicitement, on procède comme pour les fonctions... Pour les suites récurrentes, voici un exemple :

On souhaite représenter les termes de la suite (u_n) définie par :
$$\begin{cases} u_0 = 1 \\ \forall n \in \mathbb{N}, u_{n+1} = \sqrt{1 + u_n} \end{cases}$$

On peut commencer par créer une fonction permettant les calculs des termes de u_n ... puis créer une liste d'abscisses et une liste d'ordonnées :

Python 4 – Suite

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def u(n):
5     U=1
6     for k in range(1,n+1):
7         U=np.sqrt(1+U)
8     return U
9
10 x=range(0,11)
11 y=[u(n) for n in range(0,11)]
12 plt.plot(x,y,'+')
13 plt.show()
```



INCONVÉNIENT

Cette méthode fonctionne, mais

On pourrait également procéder avec les deux autres méthodes ci-dessous :

Python 5 – On place un point après l'autre

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 n=0
5 u=1
6 plt.plot(n,u,'b+')
7 for k in range(1,11):
8     u=np.sqrt(1+u)
9     plt.plot(k,u,'b+')
10
11 plt.show()
```

PETITE REMARQUE

On précise 'b+' pour que tous les points soient représentés avec le même style (ici des croix bleues), sinon, ils seraient tous avec un style différent.

Python 6 – On crée une liste

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 u=1
5 U=[u]
6 for k in range(1,11):
7     u=np.sqrt(1+u)
8     U.append(u)
9
10 plt.plot(range(0,11),U,'+')
11 plt.show()
```

●●● EXERCICE 1 - REPRÉSENTATIONS GRAPHIQUES DE FONCTIONS ET DE SUITES

1. Sur un même graphique, tracer les courbes des fonctions $f : x \mapsto \frac{e^x + e^{-x}}{2}$ et $g : x \mapsto \frac{e^x - e^{-x}}{2}$ sur la fenêtre $[-3;3] \times [-4;4]$. Ajouter un titre et une légende.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x=np.linspace(-6,6,100)
5 y1=(np.exp(x)+np.exp(-x))/2
6 y2=(np.exp(x)-np.exp(-x))/2
7
8 plt.plot(x,y1, label='courbe de f')
9 plt.plot(x,y2,label='courbe de g')
10
11 plt.axis([-6,6,-8,8])
12 plt.legend()
13 plt.title('Courbes de f et g')
14 plt.show()

```

2. Représenter les premiers termes de la suite $(u_n)_{n \in \mathbb{N}^*}$ définie par : $\forall n \in \mathbb{N}^*, u_n = \frac{n!}{n^n}$.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def facto(n):
5     P=1
6     for k in range(1,n+1):
7         P=P*k
8     return P
9
10 x=range(0,11)
11 u=[facto(n)/n**n for n in x]
12 plt.plot(x,u, '+' )
13
14 plt.show()

```

PETITE REMARQUE

Si besoin, créer une fonction permettant le calcul de $n!$...

3. Représenter les premiers termes de la suite (u_n) définie par : $\begin{cases} u_0 = 0 \\ \forall n \in \mathbb{N}, u_{n+1} = e^{-u_n} \end{cases}$.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def u(n):
5     U=0
6     for k in range(1,n+1):
7         U=np.exp(-U)
8     return U
9
10 x=range(0,11)
11 U=[u(n) for n in x]
12 plt.plot(x,U, '+' )
13
14 plt.show()

```

4. Représenter les premiers termes de la suite (u_n) définie par : $\begin{cases} u_0 = 0, u_1 = 1 \\ \forall n \in \mathbb{N}, u_{n+2} = u_{n+1} + 2u_n \end{cases}$.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 U=[0,1]
5 for k in range(2,11):
6     u=U[k-1]+2*U[k-2]
7     U.append(u)
8
9 x=range(0,11)
10 plt.plot(x,U, '+' )
11
12 plt.show()

```

PETITE REMARQUE

Ne pas hésiter à remettre le nez dans le TP1...

5. Représenter les premiers termes des suites (u_n) et (v_n) définies par $u_0 = 0, v_0 = 1$ et :

$$\forall n \in \mathbb{N}, \begin{cases} u_{n+1} = 0,2u_n + 0,7v_n \\ v_{n+1} = 0,8u_n + 0,3v_n \end{cases}$$

PETITE REMARQUE

Puisque l'objectif est de créer une liste contenant les valeurs des termes de (u_n) , j'ai choisi de la créer au fur et à mesure du calcul des termes.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 u=0
6 U=[u]
7 v=1
8 V=[v]
9 for k in range(1,21):
10     u,v=0.2*u+0.7*v,0.8*u+0.3*v
11     U.append(u)
12     V.append(v)
13
14 plt.plot(range(0,21),U,'b+')
15 plt.plot(range(0,21),V,'ro')
16
17 plt.show()
18
19 mat=np.array([U,V])
20 plt.plot(range(0,21),np.transpose(mat),'b+')
21 plt.show()

```

6. Représenter sur un même graphique la courbe de la fonction \ln ainsi que les 30 premiers termes de la somme partielle de la série harmonique.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x=np.linspace(0.01,30,1000)
5 plt.plot(x,np.log(x))
6
7 S=0
8 for k in range(1,31):
9     S=S+1/k
10     plt.plot(k,S,'r+')
11
12 plt.show()

```

●●● EXERCICE 2 - COURBES DES FONCTIONS PUISSANCES

Sur un même graphique, tracer les courbes des fonctions $f_n : x \mapsto x^n$, pour $n \in \{-2, -1, 0.5, 0.7, 1, 1.5, 2, 3\}$.

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 x=np.linspace(0.001,5,100)
5
6 for n in [-2,-1,0.5,0.7,1,1.5,2,3]:
7     y=x**n
8     plt.plot(x,y)
9
10 plt.axis([0,5,0,5])
11 plt.title("Courbes des puissances")
12 plt.legend()
13 plt.show()

```

●●● EXERCICE 3 - PETIT MÉLANGE...

Pour $n \in \mathbb{N}^*$, on considère la fonction $f_n : x \mapsto x^n + x - 1$. On a vu, dans un précédent exercice, que pour tout $n \in \mathbb{N}^*$, l'équation $f_n(x) = 0$ possède une unique solution dans $[0; 1]$, que l'on notera α_n .

1. Écrire un programme Python qui demande une valeur de $n \in \mathbb{N}^*$ à l'utilisateur et affiche un graphique représentant la courbe de f_n , en précisant avec une croix rouge le point de coordonnées $(\alpha_n, 0)$.
2. Écrire maintenant un programme qui demande une valeur de $n \in \mathbb{N}^*$ à l'utilisateur et affiche un graphique représentant toutes les courbes des fonctions f_k , en précisant avec une croix rouge le point de coordonnées $(\alpha_k, 0)$, pour $k \in \llbracket 1; n \rrbracket$.

Réponse à la question 2 :

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def f(n,x):
5     y= x**n+x-1
6     return y
7
8 def dichot(n):
9     a=0
10    b=1
11    while b-a>10**(-3):
12        m=(a+b)/2
13        if f(n,m)==0:
14            a,b=m,m
15        elif f(n,a)*f(n,m)<0:
16            b=m
17        elif f(n,a)*f(n,m)>0:
18            a=m
19    return (a+b)/2
20
21 n=int(input("Quelle valeur de n ?"))
22 for k in range(1,n+1):
23     x=np.linspace(0,1,100)
24     y=f(k,x)
25     plt.plot(x,y)
26     plt.plot(dichot(k),0,'r+')
27
28 plt.grid()
29 plt.show()

```

IMPORTANT !

On essaie de "compartmenter" le programme. Plutôt que de tout faire dans une boucle for, on crée déjà une fonction pour f_n , puis une fonction pour obtenir une valeur approchée de α_n ... Cela permet de vérifier à chaque étape que le code fonctionne, et aussi de le rendre plus lisible.

●●● EXERCICE 4 - SUITES RÉCURRENTES...

Écrire un programme Python permettant d'obtenir la représentation en "escalier", en "spirale" (ou autre?) de la suite (u_n) définie par :

$$\begin{cases} u_0 \in \mathbb{R} \\ \forall n \in \mathbb{N}, u_{n+1} = f(u_n) \end{cases}$$

où f est une fonction donnée.

On testera le programme dans les cas suivants :

1. $u_0 = 0$ et $\forall n \in \mathbb{N}, u_{n+1} = \sqrt{1+u_n}$
2. $u_0 = 0$ et $\forall n \in \mathbb{N}, u_{n+1} = e^{-u_n}$
3. d'autres cas au choix!

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def f(x):
5     y=np.exp(-x)
6     return y
7
8 x=np.linspace(0,1,100)
9 plt.plot(x,x)
10 plt.plot(x,f(x))
11
12 u=0
13 for k in range(1,11):
14     plt.plot([u,u],[u,f(u)],'r')
15     plt.plot([u,f(u)],[f(u),f(u)],'r')
16     u=f(u)
17
18 plt.show()

```