TP 05 – Représentations Graphiques

L'objectif de ce TP est d'apprendre à **représenter** graphiquement des **fonctions** et des **suites** à l'aide de `Python`. La bibliothèque nécessaire est `matplotlib.pyplot` que l'on importera de la manière suivante :

Entrée [1]:
```
import matplotlib.pyplot as plt
```

Quand on souhaite représenter graphiquement une fonction (ou une suite), il nous faut deux *tableaux* de valeurs.

- Un tableau d'*abscisses* contenant un certain nombre de valeurs de x (ou de n)

  `abscisses=[x1, x2, ..., xp]` (ou `abscisses=[n1, n2, ..., np]`)

- Un tableau d'*ordonnées* contenant les valeurs correspondantes de $f(x)$ (ou de u_n)

  `ordonnees=[f(x1), f(x2), ..., f(xp)]` (ou `ordonnees=[un1, un2, ..., unp]`)

Alors, grâce aux commandes suivantes, `Python` affiche un graphe sur lequel il a placé dans le plan tous les points `(xi,f(xi))` pour `i=1,...,p` qu'il relie (par défaut) après entre eux. Ainsi, plus le nombre de points est important, plus la courbe sera précise.

Entrée [2]:
```
plt.plot(abscisses, ordonnees)
plt.show()
```

Pour les suites, on devra forcer `Python` à ne pas relier les points entre eux (une suite n'est définie que sur des valeurs entières) grâce à l'ajout d'une option dans la première commande comme suit.

Entrée [3]:
```
plt.plot(abscisses, ordonnees, '+')
plt.show()
```

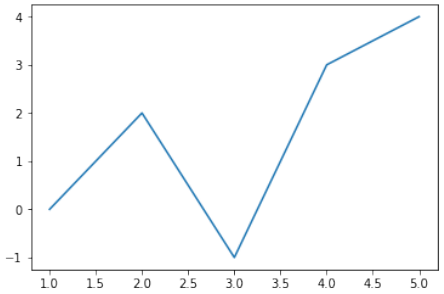
Les commandes qui suivent peuvent être rajouter pour améliorer l'esthétisme du graphe. Elles ne sont cependant pas au programme.

Commande	Effet
`plt.grid()`	Fait apparaître une grille sur le fond du repère
`plt.axis([a,b,c,d])`	restreint le repère entre les abscisses a et b et les ordonnées c et d
`plt.title("nom")`	Permet de donner un nom au graphique
`plt.plot(abscisses, ordonnes, label='nom')`	Donne un nom à la courbe (dans la légende)
`plt.legend()`	Affiche la légende

I Relier les points

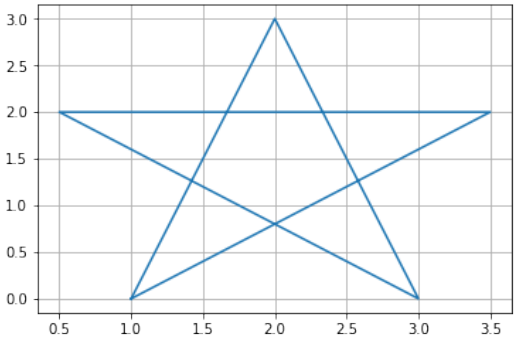
Exercice 1 Quelle courbe Python affiche-t-il à la suite de ces instructions ?

Entrée [4]:
```
abscisses = [1, 2, 3, 4, 5]
ordonnées = [0, 2, -1, 3, 4]
plt.plot(abscisses, ordonnées)
plt.show()
```



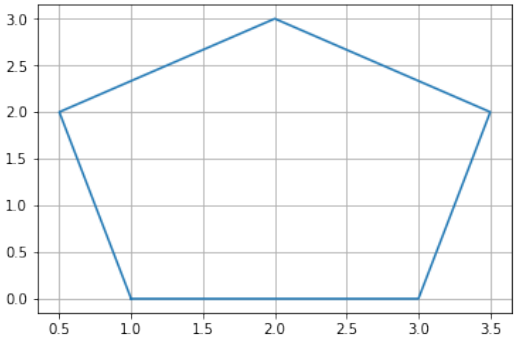
Exercice 2

1. Réaliser cette figure grâce à Python (sans la grille).




```
abscisses = [1,3.5,0.5,3,2,1]
ordonnées = [0,2,2,0,3,0]
plt.plot(abscisses, ordonnées)
plt.show()
```

2. Réaliser cette figure (sans la grille).




```
abscisses = [1,3,3.5,2,0.5,1]
ordonnées = [0,0,2,3,2,0]
plt.plot(abscisses, ordonnées)
plt.show()
```

II Opérations sur les tableaux de valeurs

De manière générale, les listes `abscisses` et `ordonnees` contiennent beaucoup de valeurs. Il faut donc apprendre à les définir et les manipules de manière efficace (sans rentrer les valeurs une par une "à la main"). Plutôt que de travailler avec des listes (comme précédemment), il est plus judicieux de travailler avec *tableaux* sur lesquels plus d'opérations sont possibles. Le module `array` de `numpy` permet de définir des tableaux.

Entrée [5]:
```
import numpy as np
L1 = np.array([1,2,3]) #definition explicite
L2 = np.array([k for k in range(4,7)]) #definition en comprehension
```

Entrée [6]:
```
print(L2)
```

Out [6]:
```
[4,5,6]
```

Commande	Effet	Résultat
`L1+L2`	Addition terme à terme	`array([5,7,9])`
`L1+2`	Addition de la constante à chaque terme	`array([3,4,5])`
`3*L1`	Multiplication par la constante de chaque terme	`array([3,6,9])`
`L1*L2`	Multiplication terme à terme	`array([4,10,18])`
`L1/L2`	Division terme à terme	`array([0.25,0.4,0.5])`
`L1**2`	Puissance terme à terme	`array([1,4,9])`
`f(L1)`	Application de la fonction f à chacun des termes	

- Pour effectuer les opérations entre `L1` et `L2`, les deux listes doivent être de la même taille.
- Dans la dernière ligne du tableau, la fonction f peut désigner une fonction déjà existante de numpy (comme `np.exp`, `np.sqrt`,...) ou une fonction créé (à l'aide de l'environnement `def`).

Exercice 3 Donner l'affichage du programme suivant.

Entrée [7]:
```
def f(x):
    return(x+1)

x = np.array( [-2+k*0.5 for k in range(0,9)])
print(x)
y = 2*x  -f(x)
print(y)
```

Out [7]:
```
[-2.  -1.5 -1.  -0.5  0.   0.5  1.   1.5  2. ]
[-3.  -2.5 -2.  -1.5 -1.  -0.5  0.   0.5  1. ]
```

III Représentation graphique d'une fonction

Pour représenter une fonction, il nous faut deux tableaux de valeurs : un contenant les abscisses souhaitées et l'autre l'image de ces abscisses par l'application qu'on souhaite représenter.

- Pour avoir un graphique précis, la *liste des abscisses* doit être grande. On ne rentre donc pas les valeurs à la main mais on peut utiliser les deux commandes suivantes.
  - La commande `np.linspace(a,b,n)` qui crée un tableau de n valeurs réparties de manière uniforme entre `a` et `b` (inclu).

Entrée [8]:
```
np.linspace(0,1,3)
```

Out [8]:
```
array([0. , 0.5, 1. ])
```

  - La commande `np.arange(a,b,p)` qui crée un tableau de de valeurs de `a` à `b` (exclu) avec un pas de `p`.

Entrée [9]:
```
np.arange(0,1,0.1)
```

Out [9]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

- La *liste des ordonnées* est obtenue par manipulation sur les listes (expliquée à la Section 2).

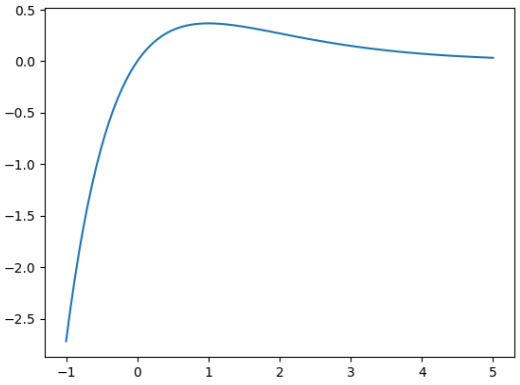
Entrée [10]:
```
#Representation de f(x)=x**2 sur [0,1]
abscisses = np.linspace(0,1,1000)
ordonnes = abscisses**2
```

Enfin, la structure complète pour obtenir la courbe d'une fonction est la suivante.

```
abscisses = liste des abscisses #obtenue avec linspace ou arange
ordonnees =  liste des abscisses #obtenue par operations
plt.plot(x,y)
plt.show()
```

Exercice 4 Représenter la fonction $x \mapsto x\exp(-x)$ sur l'intervalle $[-1, 5]$.

Entrée [11]:
```
abscisses = np.linspace(-1,5,100)
ordonnees =  abscisses*np.exp(-abscisses)
plt.plot(abscisses,ordonnees)
plt.show()
```



IV Représentation graphique d'une suite

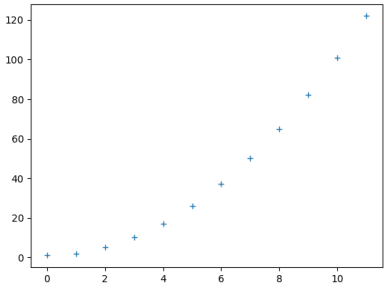
IV.1 Définie de manière explicite

Pour une suite définie de manière explicite, on procède comme pour les fonctions, en forçant Python à ne représenter que les points (et à ne pas les relier entre eux).

Exercice 5 Représenter les 12 premiers termes (de u_0 à u_{11}) de la suite $(u_n)_{n\in\mathbb{N}}$ définie par

$$\forall n \in \mathbb{N}, \qquad u_n = n^2 + 1$$

Entrée [12]:
```
abscisses = np.arange(0,12,1)
ordonnees =  abscisses**2+1
plt.plot(abscisses,ordonnees,'+')
plt.show()
```



IV.2 Définie de manière récurrente

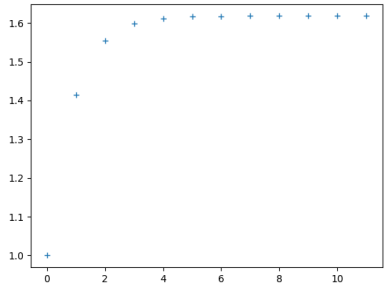
Pour représenter une suite définie de manière récurrente, on doit commencer par créer une fonction Python qui prend en argument un entier n et qui renvoie la valeur de u_n. Puis utiliser cette fonction, afin de créer la liste des ordonnées associées à une liste d'abscisses (contenant que des entiers naturels).

Exercice 6 Représenter les 12 premiers termes (de u_0 à u_{11}) de la suite $(u_n)_{n\in\mathbb{N}}$ définie par

$$u_0 = 1 \qquad \text{et} \qquad \forall n \in \mathbb{N}, \qquad u_{n+1} = \sqrt{1+u_n}$$

Entrée [13]:
```
def suite(n):
    u=1
    for k in range(1,n+1):
        u=np.sqrt(1+u)
    return(u)

abscisses = np.arange(0,12,1)
ordonnees =  [suite(n) for n in abscisses]
plt.plot(abscisses,ordonnees,'+')
plt.show()
```



V Exercices

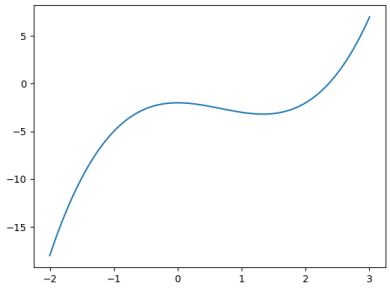
V.1 Sur les fonctions

Exercice 7 Représenter sur la fonction polynomiale suivante

$$f : x \mapsto x^3 + 2x^2 - x - 2$$

sur l'intervalle $[-2, 3]$.

Entrée [14]:
```
abscisses = np.linspace(-2,3,100)
ordonnees =  abscisses**3-2*abscisses**2-2
plt.plot(abscisses,ordonnees)
plt.show()
```



Exercice 8 Représenter sur le même graphique, les fonctions

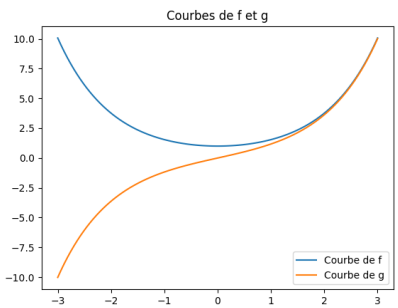
$$f : x \mapsto \frac{\mathrm{e}^x + \mathrm{e}^{-x}}{2} \qquad \text{et} \qquad g : x \mapsto \frac{\mathrm{e}^x - \mathrm{e}^{-x}}{2}$$

sur l'intervalle $[-3, 3]$.

Entrée [15]:
```
#Courbe de f
abscisses = np.linspace(-3,3,100)
ordonnees =  (1/2)*(np.exp(abscisses) +np.exp(-abscisses) )
plt.plot(abscisses,ordonnees, label='Courbe de f')


#Courbe de g
abscisses = np.linspace(-3,3,100)
ordonnees =  (1/2)*(np.exp(abscisses) -np.exp(-abscisses) )
plt.plot(abscisses,ordonnees, label='Courbe de g')

plt.title('Courbes de f et g')
plt.legend()
plt.show()
```

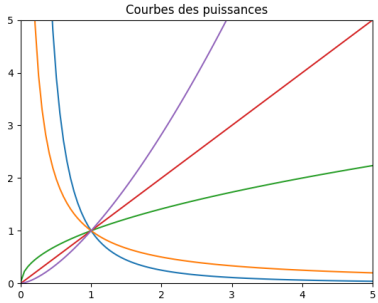


Exercice 9 Représenter sur le même graphique, les fonctions $x \mapsto x^n$ pour $n \in \{-2, -1, 0.5, 1, 1.5\}$ sur l'intervalle $]0, 5]$.

Entrée [16]:
```
abscisses = np.linspace(0.001,5,100)

for n in [-2,-1, 0.5, 1, 1.5]:
    ordonnees = abscisses**n
    plt.plot(abscisses,ordonnees)


plt.axis([0,5,0,5])
plt.title('Courbes des puissances')
plt.show()
```

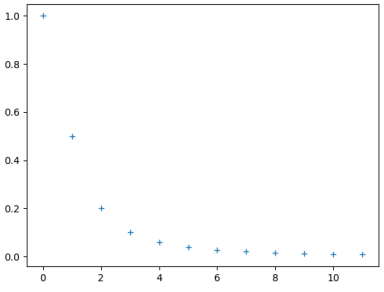


V.2 Sur les suites

Exercice 10 Représenter les premiers termes de la suite $(u_n)_{n\in\mathbb{N}}$ définie par

$$\forall n \in \mathbb{N}, \qquad u_{n+1} = \frac{1}{n^2+1}$$

Entrée [17]:
```
abscisses = np.arange(0,12,1)
ordonnees =  1/(abscisses**2+1)
plt.plot(abscisses,ordonnees,'+')
plt.show()
```

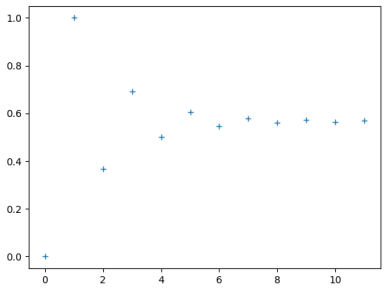


Exercice 11 Représenter les premiers termes de la suite $(u_n)_{n\in\mathbb{N}}$ définie par

$$u_0 = 0 \qquad \text{et} \qquad \forall n \in \mathbb{N}, \qquad u_{n+1} = \exp(-u_n)$$

Entrée [18]:
```
def suite(n):
    u=0
    for k in range(1,n+1):
        u=np.exp(-u)
    return(u)

abscisses = np.arange(0,12,1)
ordonnees =  [suite(n) for n in abscisses]
plt.plot(abscisses,ordonnees,'+')
plt.show()
```

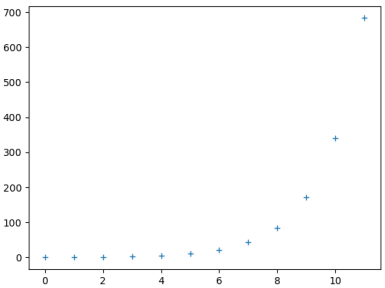


Exercice 12 Représenter les premiers termes de la suite $(u_n)_{n\in\mathbb{N}}$ définie par

$$u_0 = 0, \quad u_1 = 1 \qquad \text{et} \qquad \forall n \in \mathbb{N}, \qquad u_{n+2} = u_{n+1} + 2u_n$$

Entrée [19]:
```
def suite(n):
    u =0
    v =1
    for k in range(1,n+1):
        aux = u
        u = v
        v = v+2*aux
    return(u)

abscisses = np.arange(0,12,1)
ordonnees =  [suite(n) for n in abscisses]
plt.plot(abscisses,ordonnees,'+')
plt.show()
```

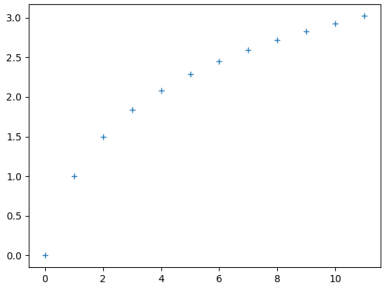


Exercice 13 Représenter les premiers termes de la suite $(S_n)_{n\in\mathbb{N}^*}$ définie par

$$\forall n \in \mathbb{N}^*, \qquad S_n = \sum_{k=1}^{n} \frac{1}{k}$$

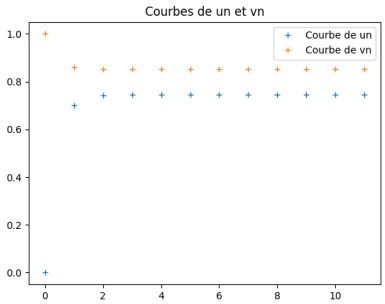
Entrée [20]:
```
def suite(n):
    S = 0
    for k in range(1,n+1):
        S = S + 1/k
    return(S)

abscisses = np.arange(0,12,1)
ordonnees =  [suite(n) for n in abscisses]
plt.plot(abscisses,ordonnees,'+')
plt.show()
```



Exercice 14 Représenter les premiers termes des suites $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ définies par

$$u_0 = 0, \quad v_0 = 1 \qquad \text{et} \qquad \forall n \in \mathbb{N}, \qquad \begin{cases} u_{n+1} &= 0,2u_n + 0,7v_n \\ v_{n+1} &= 0,8u_n + 0,3v_n \end{cases}$$



Entrée [21]:
```
#Courbe de un
def suiteu(n):
    u=0
    v=1
    for k in range(1, n+1):
        u = 0.2*u + 0.7*v
        v = 0.8*u + 0.3*v
    return(u)

abscisses = np.arange(0,12,1)
ordonnees =  [suiteu(n) for n in abscisses]
plt.plot(abscisses,ordonnees, '+', label='Courbe de un')


#Courbe de vn
def suitev(n):
    u=0
    v=1
    for k in range(1, n+1):
        u = 0.2*u + 0.7*v
        v = 0.8*u + 0.3*v
    return(v)

abscisses = np.arange(0,12,1)
ordonnees =  [suitev(n) for n in abscisses]
plt.plot(abscisses,ordonnees, '+', label='Courbe de vn')

plt.title('Courbes de un et vn')
plt.legend()
plt.show()
```