Corrigé du TP n°7 - Tracés

Dans ce T.P., nous allons apprendre à visualiser des graphes de fonctions, des courbes, des surfaces, etc... grâce à Python.
Pour ce faire, nous allons utiliser deux modules :

- le module `numpy` que nous connaissons déjà et qui nous permettra principalement d'utiliser nos fonctions mathématiques usuelles et la fonction `linspace` (on y revient dans la suite) ;
- le module `matplotlib.pyplot` qui va nous apporter tous les outils dont nous avons besoin pour réaliser de jolis graphes et courbes sur notre écran !

Sans plus tarder, importons-les (avec des petits surnoms simples - et usuels) :

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

1. Tracés de graphes de fonctions

a. La fonction `plot` de `matplotlib.pyplot`

Comment trace-t-on simplement, à la main, le graphe d'une fonction f ? Et bien :

- on choisit quelques abscisses x ;
- on calcule les ordonnées $y = f(x)$ correspondantes ;
- on place les points (x, y) ainsi formés dans le plan ;
- et enfin, on relie les points - pour faire simple, par des segments.

Avec Python, on imagine facilement qu'il est possible de mettre en œuvre un tel procédé : il suffit de définir la fonction dans Python ; créer une liste contenant les abscisses ; créer une liste contenant les ordonnées calculées grâce à la fonction définie précédemment, puis... Aïe !! Comment "placer" et "relier" les points ??
C'est là qu'entre en jeu la fonction `plot()` du module `matplotlib.pyplot` que l'on a importé au début : cette fonction bien pratique, place dans une "figure" les points qu'on lui a fournit et les relie entre eux par des segments.

Détaillons son fonctionnement :

Les bases de la fonction `plot`

La fonction `plot(X,Y)` prend deux arguments obligatoires `X` et `Y`
où `X=[x0,x1,...,xn]`,`Y=[y0,x1,...,yn]` sont deux listes **de même taille** contenant des nombres puis relie les points (`xi`,`yi`) (dans l'ordre fourni) par des segments.

Pour utiliser cette super fonction, on a besoin d'utiliser quelques outils du module `matplotlib.pyplot` :

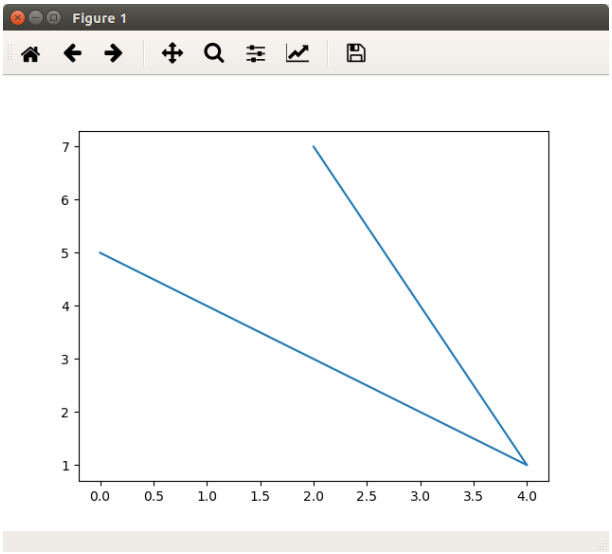
- avant d'utiliser `plt`, on emploie la fonction `figure` qui permet créer une "figure" qui contiendra notre tracé ;

— puis après avoir utilisé `plt`, on emploie la fonction `show` qui permet de visualiser notre tracé dans une fenêtre.

Testons tout ceci : on cherche à relier les points $(0,5)$, $(4,1)$ et $(2,7)$ (dans cet ordre) :

```
plt.figure() #on crée une nouvelle figure

X=[0,4,2] #on crée la liste des abscisses de nos points
Y=[5,1,7] #on crée la liste des ordonnées de nos points

plt.plot(X,Y) #plot nous relie ces points

plt.show() #show nous montre le résultat !
```

Et on obtient alors une nouvelle fenêtre à l'écran :



Remarque 1.

La fonction `figure` prend un argument facultatif nommé `figsize` et qui permet de régler la taille de la figure (mesurée en... inches !). Par exemple, l'instruction :

```
plt.figure(figsize=(5,5))
```

crée une figure de 5 inches sur 5.

1 inch $\simeq 2,54$ cm

Maintenant, traçons notre premier graphe de fonction !

Essayons de tracer la fonction $f : x \mapsto x^2 - 2$ sur $[-3,3]$. Rien de plus simple :

```
def f(x): #on définit notre fonction f
    return x**2-2
```

```
3
4 plt.figure() #on crée une nouvelle figure
5
6 X=[-3,-1,0,1,3] #on choisit quelques abscisses
7 Y=[f(-3),f(-1),f(0),f(1),f(3)] #on crée la liste des ordonnées correspondantes
8
9 plt.plot(X,Y) #plot nous relie les points du graphe
10
11 plt.show() #show nous montre de nouveau le résultat !
```

Et là... catastrophe ! Le résultat n'est clairement pas satisfaisant... et la raison est très simple ; on a pris bien trop peu de point. Notre procédé ne trace pas vraiment le graphe de la fonction mais l'approxime seulement par des segments de droites...
La solution est donc d'augmenter le nombre d'abscisses choisies au départ ! Disons que l'on veut prendre 100 valeurs régulièrement espacées entre -3 et 3 (donc une *subdivision régulière* de $[-3,3]$ de pas 99). Comment fait-on ? Et bien le module `numpy` nous fournit la fonction `linspace` qui sert à créer ce type de subdivisions.

b. La fonction `linspace` de `numpy`

Fonctionnement de la fonction `linspace`

La fonction `linspace(a,b,n)` prend 3 arguments obligatoires où `a`, `b` sont des valeurs numériques (float ou int) et `n` est un entier (int) et renvoie la liste $[a_0, a_1, ..., a_{n-1}]$ des éléments de la subdivision régulière de l'intervalle `[a,b]` en `n` points c'est-à-dire la liste des valeurs :

$$\texttt{a}= a_0 < a_1 < a_2 < \ldots < a_{n-2} < a_{n-1} = \texttt{b} \text{ où le pas } |a_{k+1} - a_k| \text{ est égal à } \frac{\texttt{b}-\texttt{a}}{\texttt{n}-1}.$$

Exemple : l'instruction `np.linspace(-3,3,10)` renvoie un objet de type array (nous verrons ce type plus tard) que nous pouvons voir comme un objet de type list :

```
>>> linspace(-3,3,10)#subdivision de [-3,3] en 10 points
array([-3.        , -2.33333333, -1.66666667, -1.        , -0.33333333,
        0.33333333,  1.        ,  1.66666667,  2.33333333,  3.        ])
```

Exercice 1. *Programmer simplement la fonction linspace*

Écrire une fonction `subdivision(a,b,n)` où `a`, `b` sont des valeurs numériques (float ou int) et `n` est un entier (int) et qui renvoie une liste contenant la subdivision régulière de l'intervalle `[a,b]` en `n` points *(comme la fonction `linspace` en fait).*
Question bonus : quelle est la complexité de votre fonction (en terme d'opérations arithmétiques) ?

Correction.

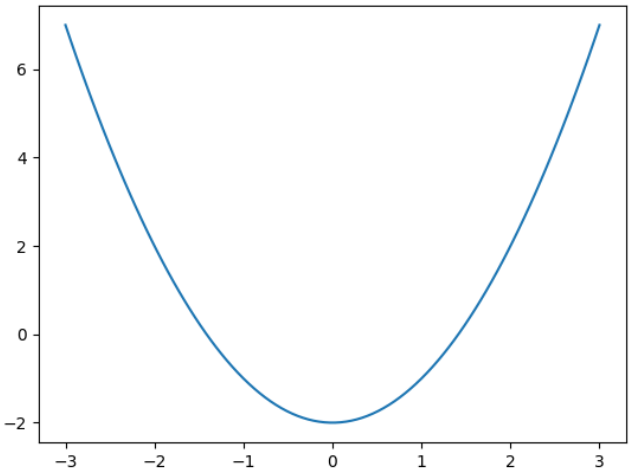
```python
def subdivision(a,b,n):
    """ a<b et n entier """
    L=[] #liste qui contiendra la subdivision
    pas=(b-a)/(n-1)
    for k in range(n):
        L.append(a+k*pas)
    return L

# ou plus concis

def subdivision(a,b,n):
    """ a<b et n entier """
    pas=(b-a)/(n-1)
    return [a+k*pas for k in range(n)]
```

Cette fonction est de complexité linéaire (en fonction de la variable `n`).

c. Tracé d'un graphe de fonction

Reprenons notre exemple précédent : la fonction $f : x \mapsto x^2 - 2$ sur $[-3, 3]$.
Cette fois-ci, avec la fonction `linspace`, on choisit pour abscisses une subdivision de $[-3, 3]$ de 100 points régulièrement espacés.

```python
def f(x):
    return x**2-2

plt.figure() #nouvelle figure

X=np.linspace(-3,3,100) #on crée une liste de 100 points entre -3 et 3 pour abscisses
Y=[f(x) for x in X] #on crée la liste des ordonnées correspondantes

plt.plot(X,Y) #on relie les couple (x,y)

plt.show() #et on admire !
```



Exercice 2.

Tracer les fonctions suivantes sur l'intervalle indiqué :

1. $x \mapsto 3x - 2$ sur $[-10, 10]$;
2. $x \mapsto \exp(-x^2)$ sur $[-5, 5]$;
3. $x \mapsto \arctan(x)$ sur $[-100, 100]$;

N'hésitez pas à augmenter le nombre de points de la subdivision si l'intervalle considéré est "grand".

Correction.

1.
```python
def f(x):
    return 3*x-2

plt.figure()

X=np.linspace(-10,10,100)
Y=[f(x) for x in X]

plt.plot(X,Y)

plt.show()
```

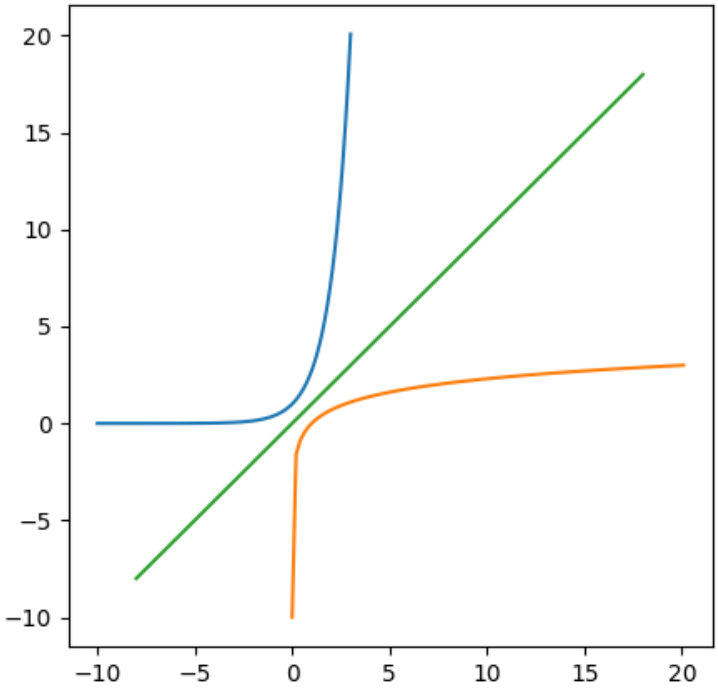
2.
```python
def f(x):
    return np.exp(-x**2)

plt.figure()

X=np.linspace(-5,5,100)
Y=[f(x) for x in X]

plt.plot(X,Y)

plt.show()
```

3.
```python
plt.figure()

X=np.linspace(-100,100,1500)
Y=[np.arctan(x) for x in X]

plt.plot(X,Y)

plt.show()
```

d. Plusieurs tracé sur une même figure

On peut bien-sûr tracer plusieurs fonctions sur le même graphe : il suffit d'effectuer les plusieurs appels de `plot` avant la fonction `show`. Par exemple, traçons les graphes des fonctions exponentielle, logarithme et identité :

```python
plt.figure(figsize=(5,5))

Xexp=np.linspace(-10,3,100) #abscisses pour la fonction exp
Yexp=[np.exp(x) for x in Xexp] #ordonnées de exp

Xln=np.linspace(np.exp(-10),np.exp(3),100) #abscisses pour la fonction ln
Yln=[np.log(x) for x in Xln] #ordonnées de exp

Xid=np.linspace(-8,18,2) #abscisses et ordonnées en même temps pour la fonction id :x ->
    x


plt.plot(Xexp,Yexp)
plt.plot(Xln,Yln)
plt.plot(Xid,Xid)

plt.show() #et on admire !
```

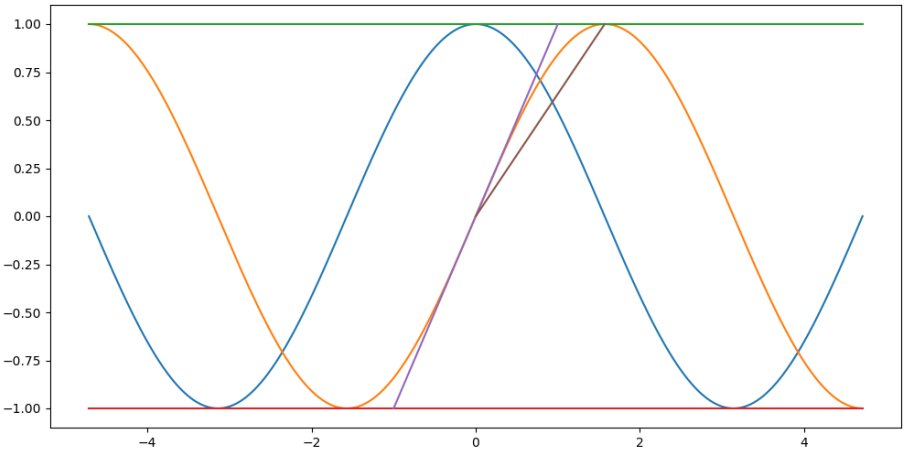


Exercice 3. *Cosinus et sinus*

1. Tracer sur une même figure les fonction cosinus et sinus sur l'intervalle $[-4\pi, 4\pi]$.
2. Ajouter à ce graphe les droites suivantes :
   — les tangentes à cosinus en $x = 0$ et $x = \pi$;
   — la tangente à sinus en $x = 0$ (on veillera à ce que le tracé de cette tangente ne dépasse pas les deux tangentes précédentes) ;
   — la corde reliant les points d'abscisses $x = 0$ et $x = \pi/2$ du graphe de sinus.

Correction.

```
plt.figure(figsize=(12,6))

X=np.linspace(-3*np.pi/2,3*np.pi/2,500) #intervalle [-4pi,4pi] en 100 points
Ycos=[np.cos(x) for x in X] #cos(X)
Ysin=[np.sin(x) for x in X] #sin(X)

plt.plot(X,Ycos)
plt.plot(X,Ysin)

#tangente à cos en 0 (c'est une droite, deux points suffisent)
XT_0_cos=[-3*np.pi/2,3*np.pi/2]
YT_0_cos=[1,1]

plt.plot(XT_0_cos,YT_0_cos)

#tangente à cos en pi
XT_pi_cos=[-3*np.pi/2,3*np.pi/2]
YT_pi_cos=[-1,-1]

plt.plot(XT_pi_cos,YT_pi_cos)

#tangente à sin en 0 (qui ne dépasse pas les deux tangentes précédentes !)
XT_0_sin=[-1,1]
YT_0_sin=[-1,1]

plt.plot(XT_0_sin,YT_0_sin)

#corde entre (0,sin(0)) et (pi/2, sin(pi/2)
Xcorde_sin=[0,np.pi/2]
Ycorde_sin=[0,1]

plt.plot(Xcorde_sin,Ycorde_sin)

#On regarde tout ça !

plt.show()
```



2. ”Décoration” des tracés

Maintenant que nous savons tracer nos fonctions, il va s'intéresser à la ”déco” de notre figure comme par exemple :

— choisir le couleur et le tracé de nos courbes ;
— placer des points sur la figure ;
— modifier le repère
— placer une légende
— etc, etc...

> **Remarque :**
> Il y a beaucoup trop d'options pour qu'on puisse en faire une liste exhaustive dans ce T.P. ; pour obtenir tout ce qu'il est possible de faire, n'hésitez pas à :
> - taper `help(plt.plot)` pour accéder à la documentation via Python ;
> - vous rendre sur la page de la fonction `plot` sur le site officiel https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html (accessible également en tapant ”plot matplotlib” dans votre moteur de recherche préféré !).

Dans la suite, nous omettrons les fonctions autour du ”`plot`” comme `figure` ou `show` bien qu'elles soient nécessaires pour le tracé, mais ainsi, on gagnera en place et en clarté !

a. La couleur et le style du tracé :

Voici les trois arguments optionnels principaux permettant de modifier la couleur et le style du tracé :

— `color` qui gère la couleur du tracé et qui doit valoir le nom de la couleur voulue, en anglais, sous forme de chaîne de caractères ; par exemple `blue`, `red`, `yellow`,...
— `linestyle` qui gère le style du tracé et qui doit valoir une des chaînes suivantes :
  * ’-’ ou ’solid’ (tracé normal)
  * ’–’ or ’dashed’ (tracé en pointillés)
  * ’ :’ or ’dotted’ (tracé en points)
  * ’-.’ or ’dashdot’ (tracé pointillés-points)
  * ” ou ’None’ (aucun tracé)

— `linewidth` qui gère l'épaisseur du tracé et qui doit valoir une valeur numérique.

Exemple :

Figure 1 : `plt.plot(X,Y, color="green", linestyle="dashed", linewidth=2)`

Figure 2 : `plt.plot(X,Y, color="black", linestyle="dotted", linewidth=3.5)`

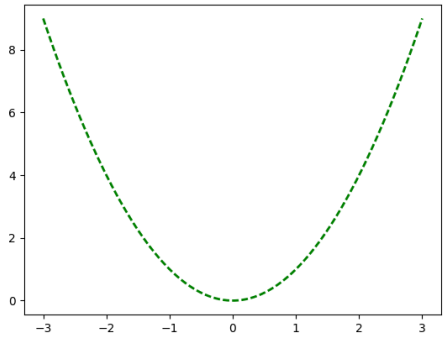


Figure 1

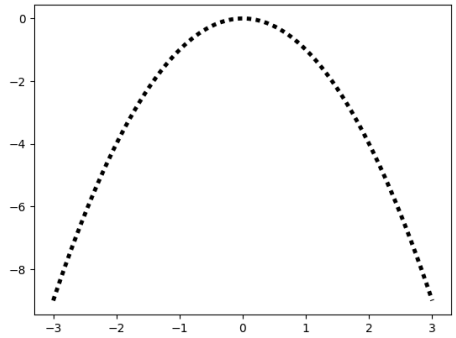


Figure 2

b. Ajout de points sur la figure

Pour placer des points sur la figure, il suffit de "dire" à `plot` qu'on veut placer les points en plus de les relier grâce l'argument `marker` :

L'argument optionnel `marker` propose plusieurs façon de placer les points avec, par exemple les valeurs :

* '+'
* 'x'
* 'X'
* '.'
* '*'
* etc... voir la documentation pour plus d'options (il y en a beaucoup !)

Exemple :

Figure 1 : `plt.plot([-1,0,1],[0,2,0], color="blue", marker="X", linewidth=1)`

Figure 2 : `plt.plot([-4,-2,-1,0,1,3,4],[-4,-2,-0.5,0,1.5,2,4], color="red", marker="*", linestyle='')#linestyle='' permet d'empêcher que les points ne soient reliés`

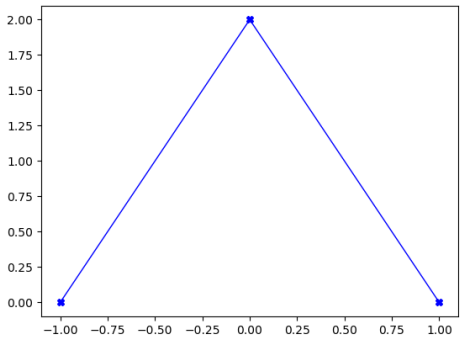


Figure 1

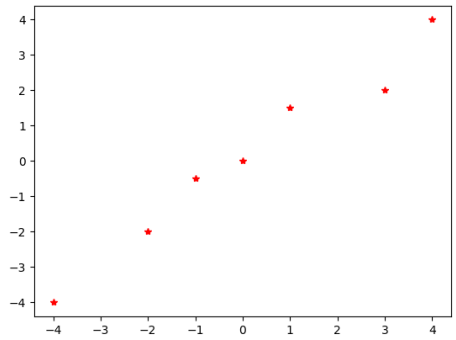


Figure 2

> **Attention :** `marker` et `linspace` font souvent très mauvais ménage ! En effet, si on choisit de marquer les 100 points définis par une subdivision en 100 points, la figure risque d'être rapidement illisible !
> En pratique, on fera deux appels de plot : l'un pour tracer la fonction, l'autre pour placer les points particuliers en utilisant l'option `linewidth=''` (pour empêcher que les points ne soient reliés).

c. Ajout d'une légende

L'ajout d'une légende est très simple, il suffit d'utiliser l'argument `label`, de lui donner comme valeur le nom de la légende souhaitée en chaîne de caractères puis de placer la légende sur la figure grâce à la fonction `plt.legend()` qui prend pour argument facultatif `loc` qui prend pour valeur une chaîne de caractère parmi :

> 'best', 'upper right', 'upper left', 'lower left', 'lower right', 'right', 'center left', 'center right', 'lower center', 'upper center', ou 'center'

qui indiquent, en anglais, le placement de la légende sur la figure

Exemple :

Figure 1 :

```
1 plt.plot(X,Y, label="la fonction carré")
2 plt.legend()
```

Figure 2 :

```
1 plt.plot(X,Y, color="purple", label="encore la fonction carré")
2 plt.legend(loc='center')
```

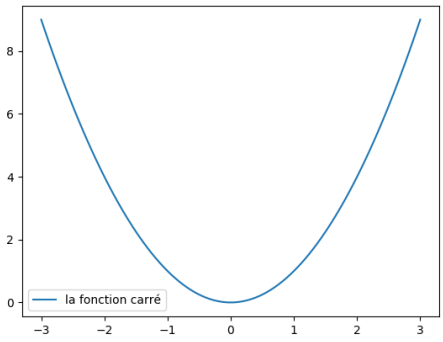


Figure 1

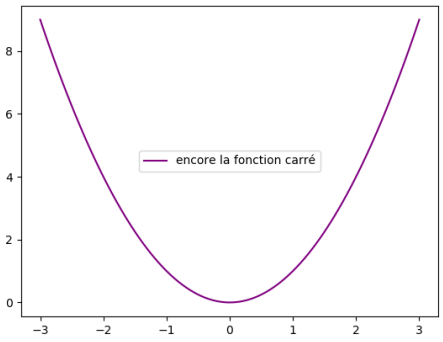
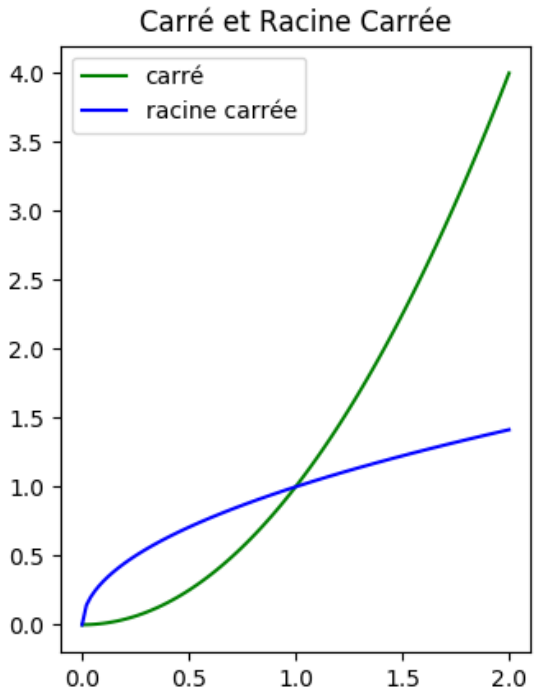


Figure 2

On peut également ajouter un titre à la figure grâce à la fonction `plt.title`. Par exemple :

```
X=np.linspace(0,3,100)
Yc=[x**2 for x in X]
Yr=[np.sqrt(x) for x in X]
plt.plot(X,Yc, color="green", label="carré")
plt.plot(X,Yr, color="blue", label="racine carrée")
plt.legend(loc='upper left')

plt.title('Carré et Racine Carrée') #création d'un titre en haut de la figure
```

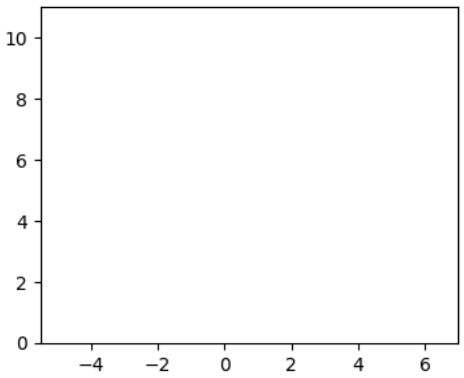


d. Modification du repère

Pour agir sur les axes du repère, on utilise plusieurs fonctions :

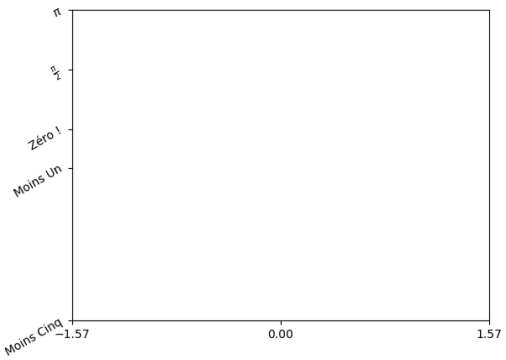
- `plt.xlim` et `plt.ylim` qui permettent de choisir les extrémités des axes :

```
plt.figure()
plt.xlim(-5.5,7)
plt.ylim(0,11)
plt.show()
```



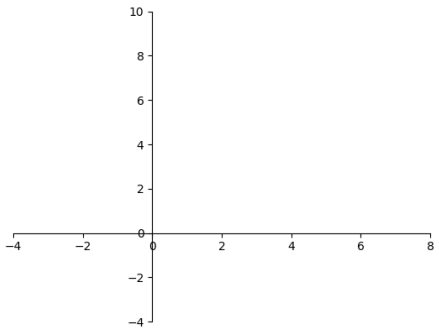
- `plt.xticks plt.yticks` qui permettent de gérer les graduations des axes (horizontal et vertical respectivement).

```
plt.figure()

#liste des graduations de l'axe (Ox)
plt.xticks([-np.pi/2,0,np.pi/2])

#liste des graduations de l'axe (Oy)
#ainsi que leur texte de remplacement (on peut utiliser du latex!)
#et on peut faire pivoter les textes associés
plt.yticks([-5,-1, 0, np.pi/2,np.pi],['Moins Cinq', 'Moins Un','Zéro !', r'$\frac
    {\pi}{2}$',r'$\pi$'], rotation=30)

plt.show()
```



- `plt.gca()` permet de ”récupérer” les axes (GCA=Get the Current Axes) et de les modifier ; par exemple, on peut changer leurs couleurs ou les déplacer :

```
plt.figure()

plt.xlim(-4,8)
plt.ylim(-4,10)

axes = plt.gca() #on récupère les 4 axes dans la variables axes (haut, droite, bas, gauche)
axes.spines['right'].set_color('none') #on rend l'axe de droite transparent (sans couleur)
axes.spines['top'].set_color('none') #de même pour l'axe du haut

## On place le "0" des axes du bas et de gauche en correspondance avec le 0 des graduations
axes.xaxis.set_ticks_position('bottom')
axes.spines['bottom'].set_position(('data',0))
axes.yaxis.set_ticks_position('left')
axes.spines['left'].set_position(('data',0))

plt.show()
```



- De nombreuses autres options de modifications des axes sont disponibles (par exemple `grid`, etc...). N’hésitez pas à fouiner dans la documentation !

e. A vous de jouer

Exercice 4.

En reprenant le code de l'exercice précédent avec les tracés de cos et sin, utiliser les techniques ci-dessus ainsi que celles que vous avez dénichées dans la documentation pour rendre la figure bien plus lisible !

3. Tracer un arc paramétré

Au delà des graphes de fonctions, on peut également tracés des arcs paramétrés (par exemple des portraits de phases en Physique) c'est-à-dire des courbes du plan de la forme :

$$\{(x(t), y(t)) \in \mathbb{R}^2 \mid t \in I\} \text{ où } I \text{ est un intervalle.}$$

Nous étudierons ce genre de courbes en Mathématiques en 2ème année, mais en attendant, amusons-nous à en tracer !

Le principe est simple :

- On crée une liste `T` grâce à `linspace` qui contient les valeurs du paramètres t dans I ;
- On crée des listes X,Y correspondants aux valeurs $x(t)$, $y(t)$ respectivement ;
- et on trace le tout avec `plot`

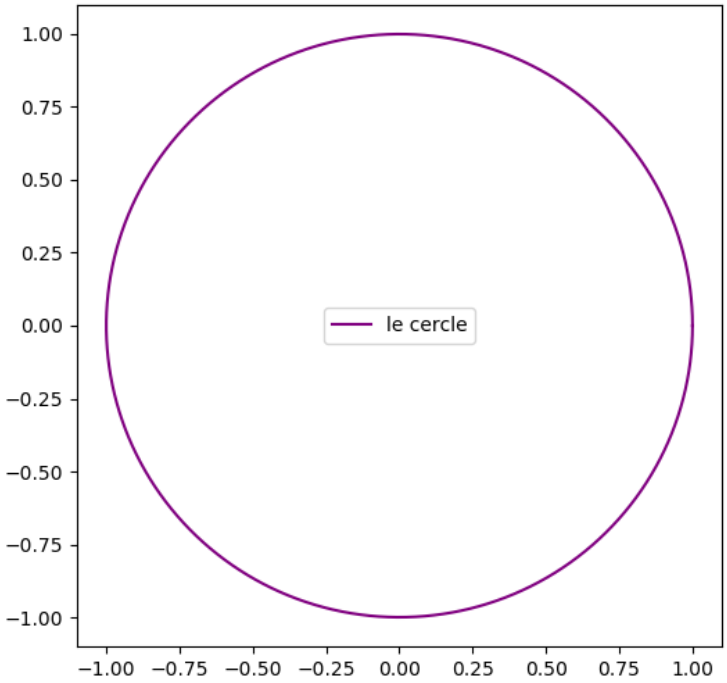
Exemple : le cercle

Le cercle est paramétré, pour $t \in [0, 2\pi]$ par :

$$\begin{cases} x(t) = \cos(t) \\ y(t) = \sin(t) \end{cases}$$

On trace ainsi le cercle :

```
plt.figure(figsize=(6,6))

T=np.linspace(0,2*np.pi,200)
X=[np.cos(t) for t in T]
Y=[np.sin(t) for t in T]

plt.plot(X,Y, color="purple", label="le cercle")
plt.legend(loc='center')

plt.show()
```



Exercice 5.

1. Tracer l'arc paramétré défini, pour $t \in [-\pi, \pi]$, par :

$$\begin{cases} x(t) = \sin(2t) \\ y(t) = \cos(3t) \end{cases}$$

2. Tracer l'arc paramétré défini, pour $t \in [-\pi, \pi]$, par :

$$\begin{cases} x(t) = (\cos(2t) - \cos(t))/2 \\ y(t) = (\sin(2t) + \sin(t))/2 \end{cases}$$

3. Essayer, en manipulant le paramétrage du cercle vu plus haut, de tracer une ellipse ;
4. En faisant quelques recherches sur internet, trouver des paramétrages puis tracer les courbes suivantes :
   - la cycloïde ;
   - la cardioïde ;
   - l'astroïde ;
   - le folium de Descartes
5. A vous d'en trouver d'autres ou même d'en inventer !