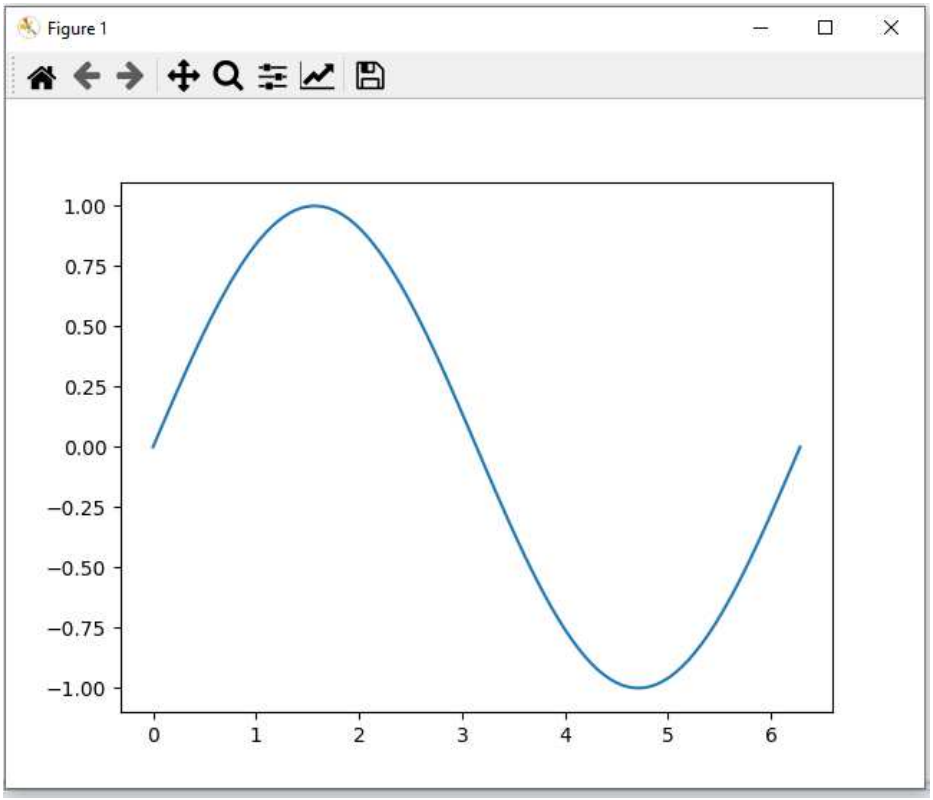
Les courbes

Pour tracer une courbe simple, il faut deux listes, tableaux de même dimension. Elles correspondent aux données des axes x et y.

Exemple minimal

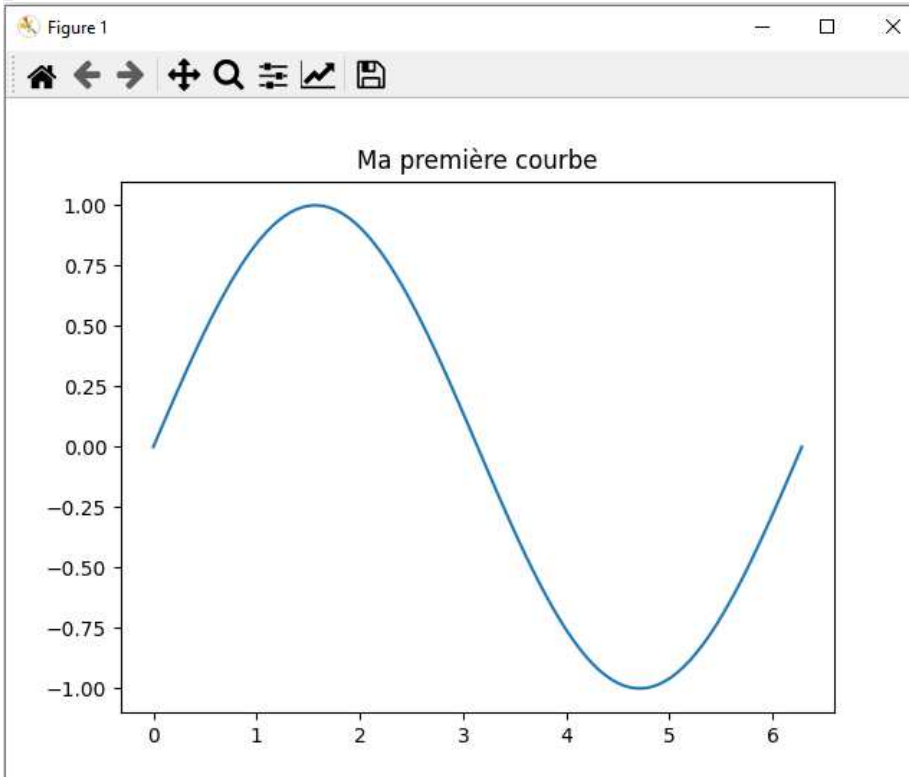
```
# Créé par Claude, le 08/10/2024 en Python 3.7

import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

plt.plot(x, y)  # Ajout du d'une courbe
plt.show()  # Affichage d'une courbe
```



Ajout d'un titre au graphique

Pour ajouter un titre au graphique, on utilise la fonction `plt.title("...")`.

```python
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

plt.plot(x, y)  # Ajout du d'une courbe

plt.title("Ma première courbe")  # Titre du graphique

plt.show()  # Affichage d'une courbe
```



Ajout des titres des axes

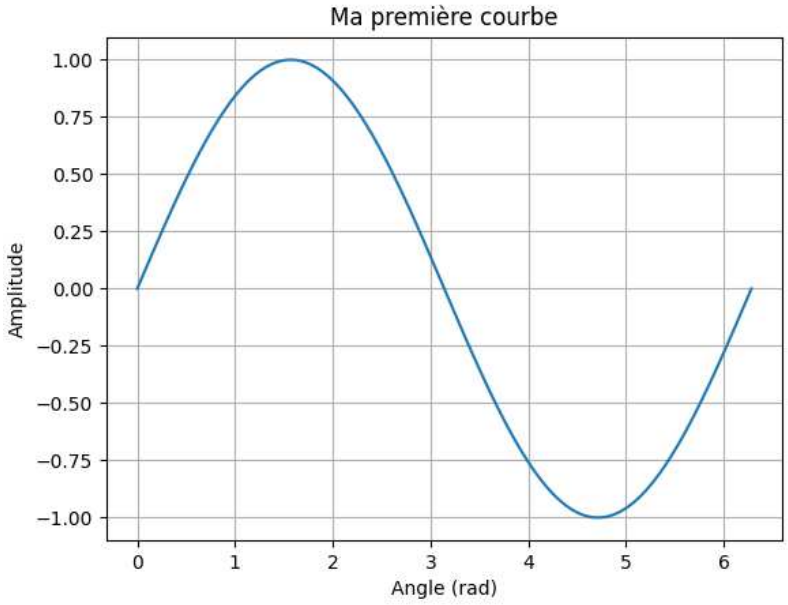
Pour ajouter un titre sur les axes, on utilise les fonctions `plt.xlabel("...")` et `plt.ylabel("...")`.

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

plt.plot(x, y)  # Ajout du d'une courbe
plt.title("Ma deuxième courbe")  # Titre du graphique

plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x

plt.show()  # Affichage d'une courbe
```

Affichage de la grille

Pour afficher la grille, on utilise la fonction `plt.grid(True)`.

```
plt.grid(True)  # Affichage de la grille

plt.show()  # Affichage d'une courbe
```

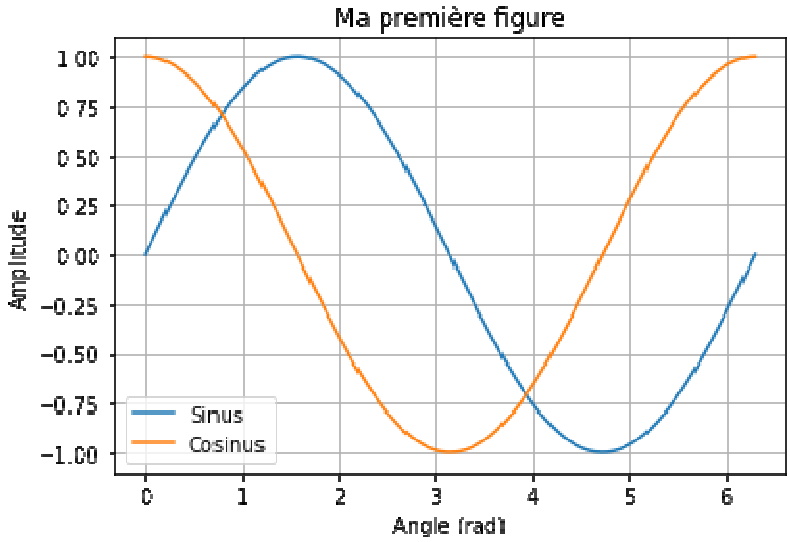


Ajout d'une deuxième courbe et d'une légende

Pour que la légende puisse s'afficher, il faut donner un label à chaque courbe avec le paramètre `label="..."` dans la fonction `plt.plot(...)`.

Par défaut, la légende se place automatique au mieux sur le graphique.

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y, label="Sinus")  # Ajout du d'une courbe avec label

plt.plot(x, y2, label='Cosinus')  # Ajout d'une deuxième courbe avec label
plt.legend()  # Affichage de la légende

plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille
plt.show()  # Affichage d'une courbe
```

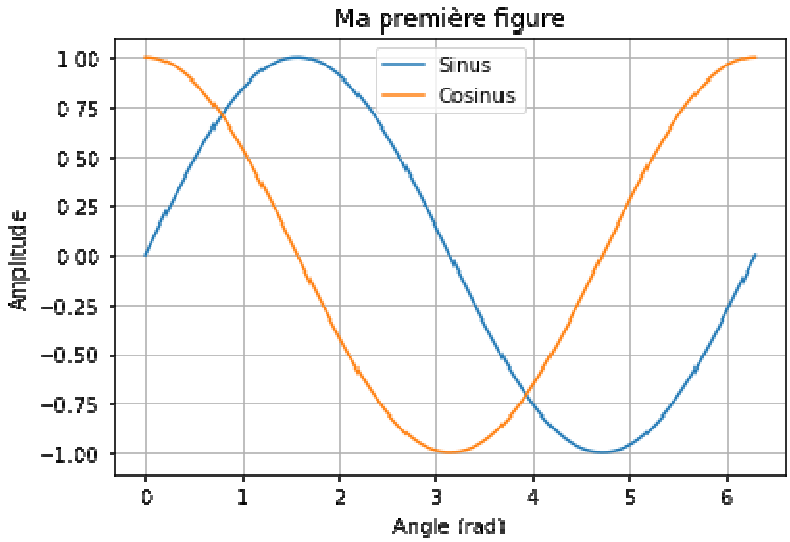


On peut modifier l'emplacement de la légende avec le paramètre `loc=x`, xétant la position voulue.
Les différentes positions sont données dans le tableau ci-dessous.

Position	**Code**	'right'	5
'best'	0	'center left'	6
'upper right'	1	'center right'	7
'upper left'	2	'lower center'	8
'lower left'	3	'upper center'	9
'lower right'	4	'center'	10

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
```

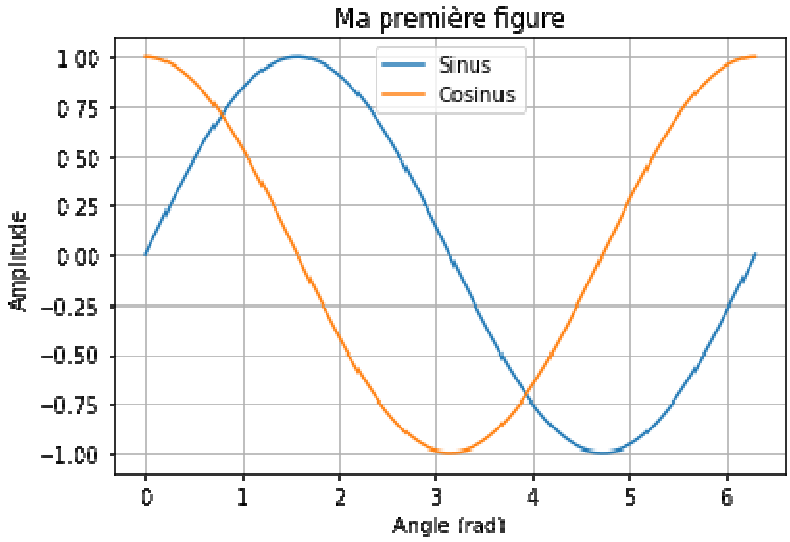
```
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y, label="Sinus")  # Ajout du d'une courbe avec label

plt.plot(x, y2, label='Cosinus')  # Ajout d'une deuxième courbe avec label
plt.legend(loc=9)  # Affichage de la légende

plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille
plt.show()  # Affichage d'une courbe
```



Sauvegarder le graphique

Il est possible de sauvegarder le graphique sous différents formats (pdf, png, jpeg ...) avec la commande `plt.savefig('Nom.pdf')`. L'extension du fichier détermine automatique son format.

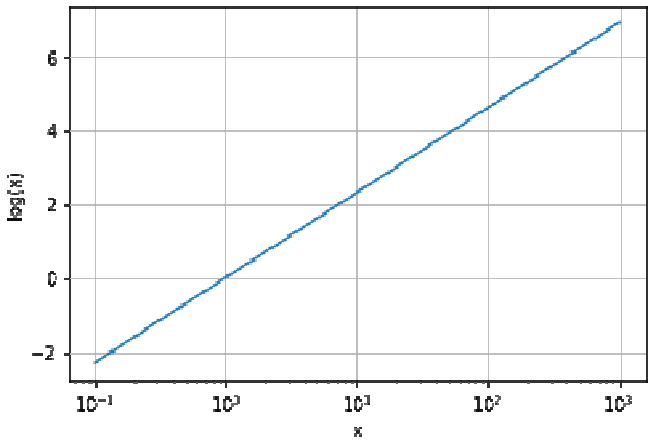
```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y, label="Sinus")  # Ajout du d'une courbe avec label
plt.plot(x, y2, label='Cosinus')  # Ajout d'une deuxième courbe avec label
plt.legend(loc=9)  # Affichage de la légende
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille

plt.savefig("Graphique.png", dpi=300)  # Sauvegarde en png avec un dpi de 300

plt.show()  # Affichage d'une courbe
```



Échelle semi-logarithmique et logarithmique

On peut tracer une courbe avec une échelle semi-logarithmique en utilisant la fonction `plt.semilogx(x,y)`

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0.1, 1000, 100)
y = np.log(x)

plt.semilogx(x, y)  # Ajout du d'une courbe
plt.ylabel('log(x)')  # Titre de l'axe y
plt.xlabel("x")  # Titre de l'axe x
plt.grid(True)  # Affichage de la grille
plt.show()  # Affichage d'une courbe
```



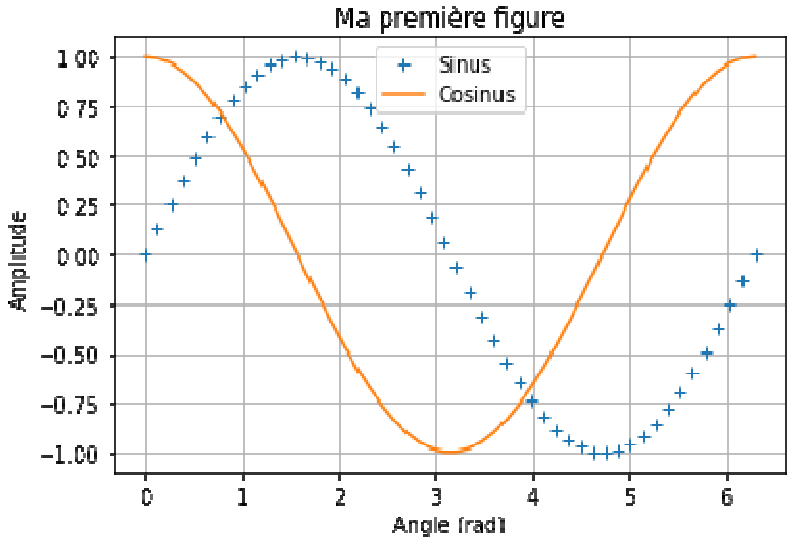
Cela marche aussi pour l'axe des ordonnées avec la fonction `plt.semilogy(x,y)`.

Pour une échelle log-log on utilise la fonction `plt.loglog(x,y)`.

Affichage des points seuls

On peut afficher uniquement les points sans les relier ensemble en rajoutant le paramètre `'o'` dans la fonction `plt.plot(...)`.

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 50)
y = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y, '+', label="Sinus")  # Ajout du des points uniquement

plt.plot(x, y2, label='Cosinus')  # Ajout d'une deuxième courbe avec label
plt.legend(loc=9)  # Affichage de la légende
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille
plt.show()  # Affichage d'une courbe
```

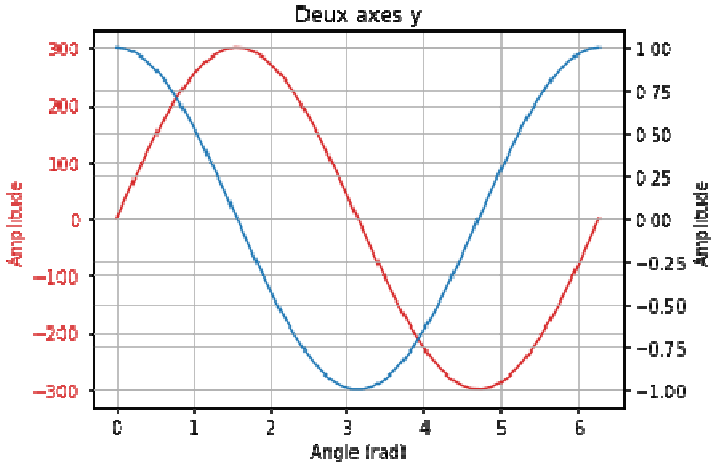


Axe secondaire

Si les données à afficher sont de dimensions différentes ou d'unités différentes, il est possible de construire un axe secondaire pour différencier ces données.

Création de deux axes y

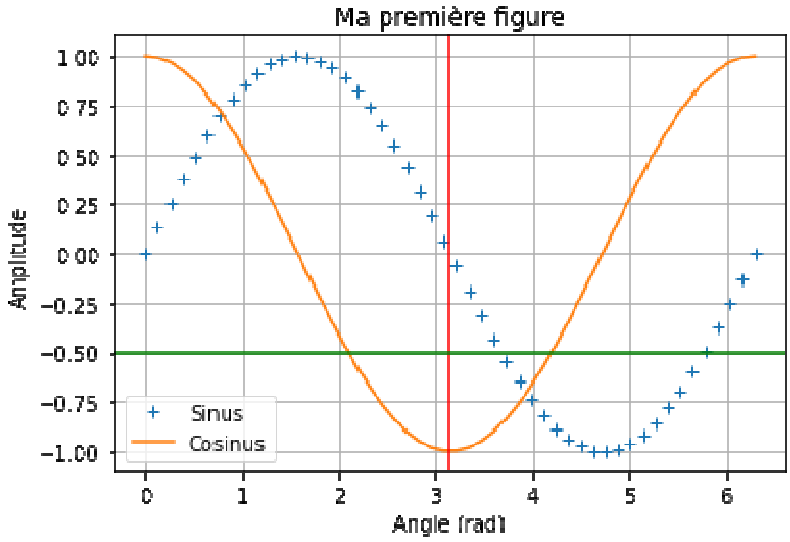
```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 50)
y = 300*np.sin(x)
y2 = np.cos(x)

# Affichage des données de l'axe principal
fig, ax1 = plt.subplots()

color = 'tab:red'
ax1.set_xlabel('Angle (rad)')
ax1.set_ylabel('Amplitude', color=color)
ax1.plot(x, y, color=color)
ax1.tick_params(axis='y', labelcolor=color)
ax1.grid(True)

# Affichage de l'axe secondaire
ax2 = ax1.twinx()  # Création d'un deuxième axe qui partage le même axe x

color = 'tab:blue'
ax2.set_ylabel('Amplitude')#, color=color)  # Labelde l'axe x
ax2.plot(x, y2, color=color)
#ax2.tick_params(axis='y', labelcolor=color)
ax2.grid(True)


plt.title("Deux axes y")
fig.tight_layout()  # Amélioration de l'affichage
plt.show()
```



Barres verticales et horizontales

Il est parfois nécessaire d'afficher une barre verticale ou horizontale. Pour cela on utilise les fonctions `plt.axvline(x=0, ymin=0, ymax=1)` et `plt.axhline(y=0, xmin=0, xmax=1)`.

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 50)
y = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y, '+', label="Sinus")  # Ajout du des points uniquement

plt.plot(x, y2, label='Cosinus')  # Ajout d'une deuxième courbe avec label
plt.legend()  # Affichage de la légende
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille

plt.axvline(x=np.pi, ymin=0, ymax=1, color="red")  # Barre verticale
plt.axhline(y=-0.5, xmin=0, xmax=1, color="green")  # Barre horizontale

plt.show()  # Affichage d'une courbe
```

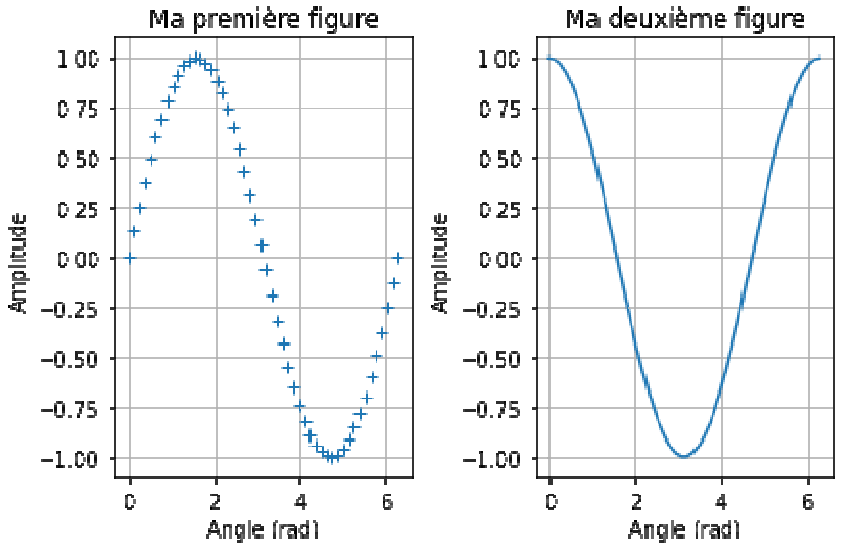


Sous figure ou afficher plusieurs graphiques sur la même figure

Il est possible d'afficher plusieurs graphiques côte à côte ou superposés sur la même figure grâce à la fonction `plt.subplot(lcp)`.
Avec comme paramètre 3 chiffres compris entre 1 et 9 pour définir une matrice de graphique avec `l` le nombre de lignes, `c` le nombre de colonnes et `p` l'emplacement où placer le graphique.

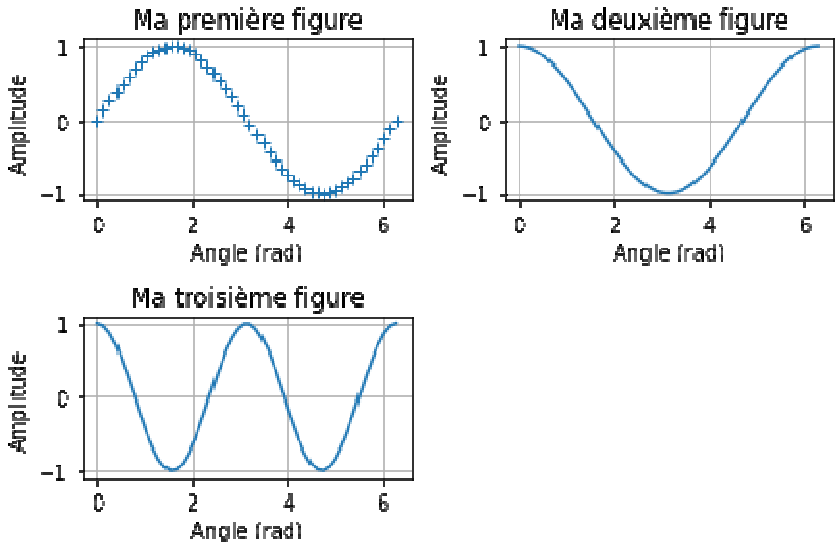
Pour éviter les chevauchements des graphiques, on utilise en plus la fonction `plt.tight_layout()`.

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 50)
y = np.sin(x)
y2 = np.cos(x)

plt.subplot(121) # Graphique côte à côte
plt.plot(x, y, '+', label="Sinus")  # Ajout du des points uniquement
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille

plt.subplot(122) # Graphique superposé
plt.plot(x, y2, label='Cosinus')  # Ajout d'une deuxième courbe avec label
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma deuxième figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille

plt.tight_layout() # Adaptation de l'affichage
plt.show()  # Affichage d'une courbe
```




```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 50)
y = np.sin(x)
y2 = np.cos(x)
y3 = np.cos(2*x)
```

```
plt.subplot(221) # Graphique matrice 2x2
plt.plot(x, y, '+', label="Sinus")  # Ajout du des points uniquement
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille

plt.subplot(222) # Graphique matrice 2x2
plt.plot(x, y2, label='Cosinus')  # Ajout d'une deuxième courbe avec label
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma deuxième figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille

plt.subplot(223) # Graphique matrice 2x2
plt.plot(x, y3, label='Cosinus')  # Ajout d'une troisème courbe avec label
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma troisième figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille

plt.tight_layout() # Adaptation de l'affichage
plt.show()  # Affichage d'une courbe
```



Personnalisation des courbes et autres options

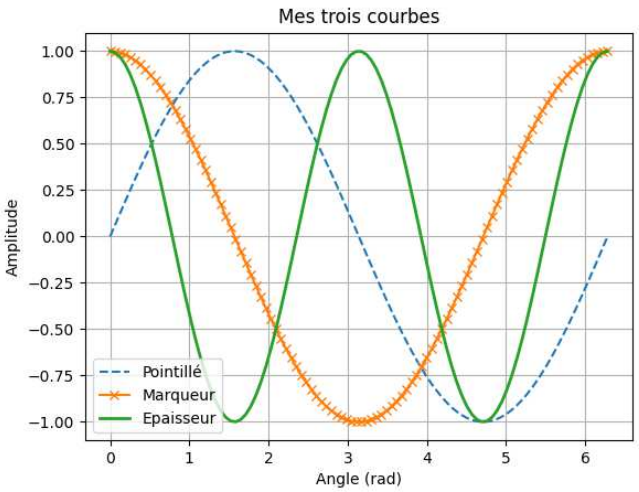
Options d'affichage des lignes

Il est possible de modifier l'affichage des courbes en jouant sur les paramètres suivants (à inclure dans `plt.plot(...)`:

- `lw=1` : épaisseur de ligne (par défaut : 1) ;
- `ls='-'` : style de ligne -> ':', '-.', '--' (par défaut : '-' ligne continue) ;
- `marker='.'` : type de marqueur pour les points -> '.', 'o', 'x', '+' ... (par défaut aucun), une liste des marqueurs possibles.

  https://fr.matplotlib.net/stable/gallery/lines_bars_and_markers/marker_reference.html

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
y2 = np.cos(x)
y3 = np.cos(2*x)

plt.plot(x, y, ls='--', label="Pointillé")  # Ajout du d'une en pointillé
# Ajout d'une courbe avec un 'x' comme marqueur
plt.plot(x, y2, marker='x', label='Marqueur')
plt.plot(x, y3, lw='2', label='Epaisseur')  # Ajout d'une courbe d'épaisseur 2

plt.legend()  # Affichage de la légende
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Mes trois courbes")  # Titre du graphique
plt.grid(True)  # Affichage de la grille

plt.savefig("Graphique.png", dpi=300)  # Sauvegarde en png avec un dpi de 300

plt.show()  # Affichage d'une courbe
```



Couleur des lignes

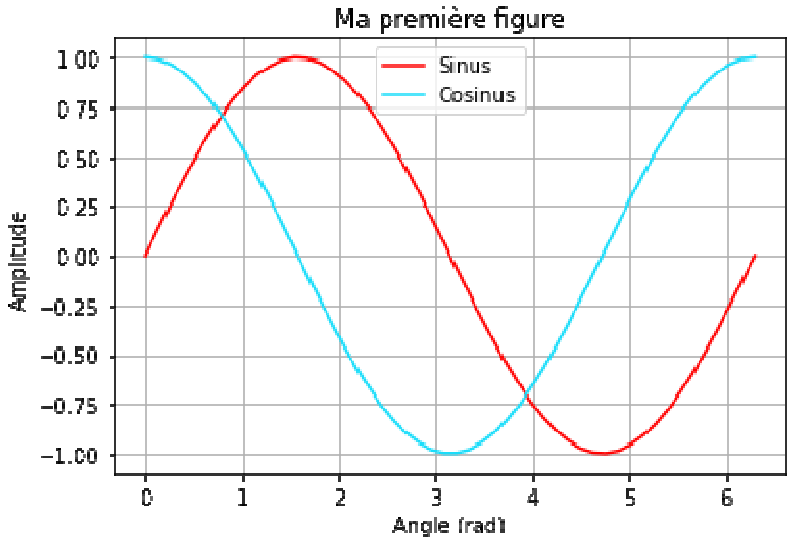
On peut changer la couleur des lignes à l'aide du paramètre `color='red'`.

Liste de couleurs disponibles :

Alias	Couleur
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Il est aussi possible de donner le code hexadécimal de la couleur `color='#18DDFA'`.

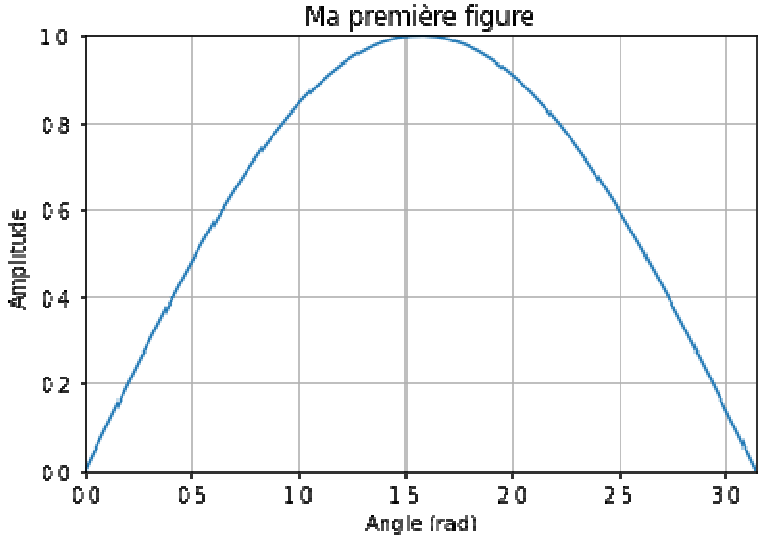
```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 50)
y = np.sin(x)
y2 = np.cos(x)

# Définition de la couleur par le nm
plt.plot(x, y, label="Sinus", color='red')
# Définition de la couleur par le code hexa
plt.plot(x, y2, label='Cosinus', color='#18DDFA')

plt.legend(loc=9)  # Affichage de la légende
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille
plt.show()  # Affichage d'une courbe
```



Bornes de l'affichage

Il est possible de réduire les bornes des axes abscisses et des ordonnées avec les fonctions `plt.xlim(min,max)` et `plt.ylim(min,max)`.

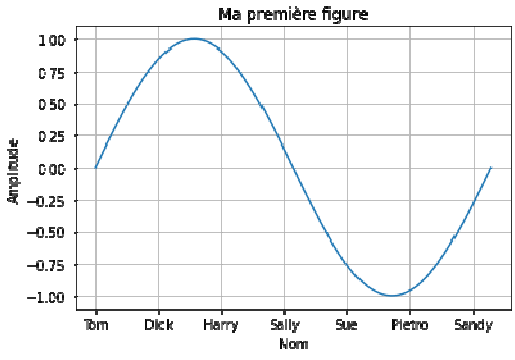
```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

plt.plot(x, y)  # Ajout du d'une courbe
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille

plt.ylim(0, 1)  # On limites l'axe des ordonnées
plt.xlim(0, np.pi)  # on limite l'axe des abscisses

plt.show()  # Affichage d'une courbe
```



Personnalisation des étiquettes des axes

On peut personnaliser les étiquettes des axes en donnant une liste avec les fonctions `plt.xticks(...)`et `plt.ytixks(...)`.

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

plt.plot(x, y)  # Ajout du d'une courbe
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Nom")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille

# Changement des étiquettes de l'abscisse par des prénoms
plt.xticks(np.arange(2*np.pi), ('Tom', 'Dick',
                                'Harry', 'Sally', 'Sue', 'Pietro', 'Sandy'))

plt.show()  # Affichage d'une courbe
```



Affichage des étiquettes en écriture scientifique

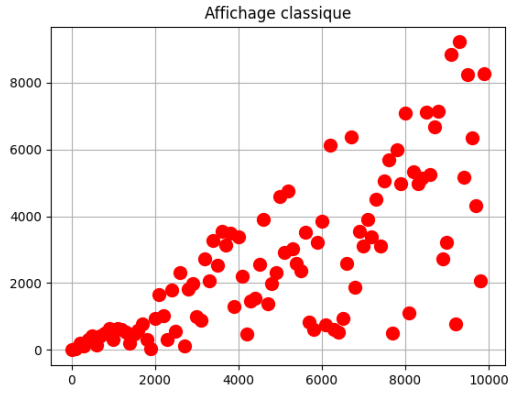
Pour de grands nombres, il est parfois nécessaire de changer le mode d'affichage des étiquettes et de le passer en écriture scientifique. Pour cela, on utilise la fonction `plt.ticklabel_format(axis='both', style='sci', scilimits=(0,0))`, avec les paramètres suivant :

- `axis='x'` : l'axe sur lequel appliquer le changement d'affichage -> 'x', 'y' ou 'both' ;
- `style='sci` : pour appliquer l'écriture scientifique ;
- `scilimiter=(m,n)` : limitation de l'écriture scientifique à l'extérieur de l'intervalle de 10^m

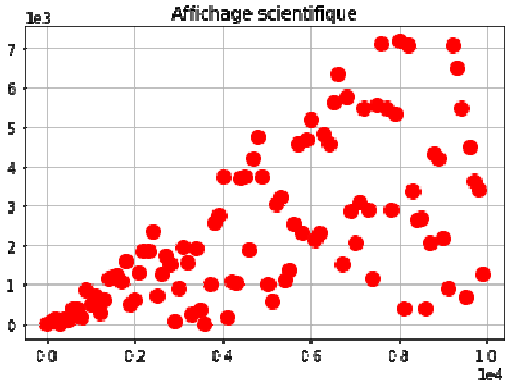
à 10^n

- , on peut mettre `(0,0)`pour l'appliquer à toutes les valeurs.

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

# Création des données à afficher
x = np.arange(start=0, stop=10000, step=100)
y = np.random.rand(len(x))
y = x*y

# Affichage
plt.plot(x, y, 'ro', markersize=10)
plt.grid(True)
plt.title("Affichage classique")
plt.show()
```




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

# Création des données à afficher
x = np.arange(start=0, stop=10000, step=100)
y = np.random.rand(len(x))
y = x*y

# Affichage
plt.plot(x, y, 'ro', markersize=10)
plt.grid(True)
plt.title("Affichage scientifique")

plt.ticklabel_format(axis='both', style='sci',
scilimits=(0, 0))

plt.show()
```

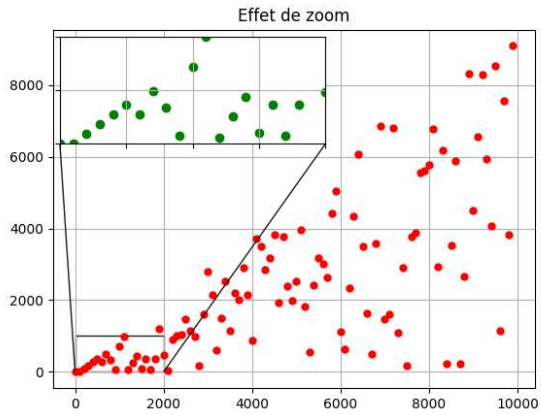


Zoom sur une partie du graphique

Il est possible d'inclure dans un graphique un effet de zoom sur une partie du graphique pour mettre en évidence un élément.
L'emplacement du zoom est défini par un numéro, voir le tableau ci-dessous :

Position	Code
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

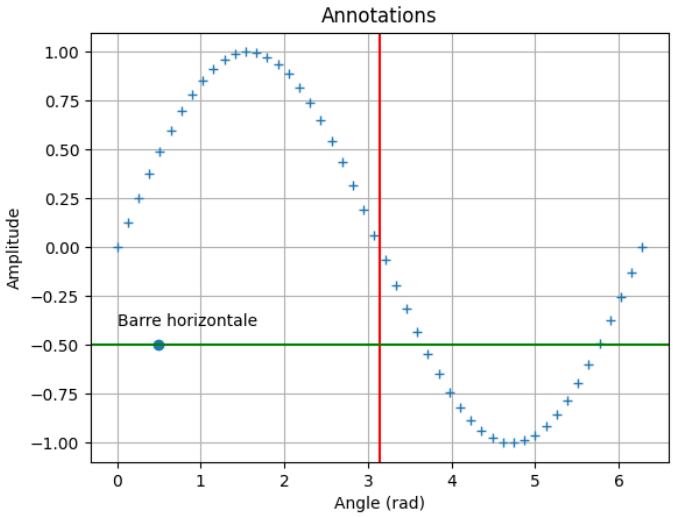
```
from mpl_toolkits.axes_grid1.inset_locator import zoomed_inset_axes, mark_inset
import matplotlib.pyplot as plt
import numpy as np

# Création des données à afficher
x = np.arange(start=0, stop=10000, step=100)
y = np.random.rand(len(x))
y = x*y

# Création des éléments d'axe et figure
fig, ax = plt.subplots()

# Affichage du graphique de base
plt.plot(x, y, 'ro', markersize=5)
plt.grid(True)
plt.title("Effet de zoom")

# Zoom
# Facteur de zoom: 3, location: en haut à gauche
axins = zoomed_inset_axes(ax, 3, loc=2)

axins.scatter(x, y, label='Zoom', color='green') # On dessine le graphique zoomé
axins.set_xlim(0, 2000)  # Limite de l'axe x
axins.grid(True)
axins.set_ylim(0, 1000)  # Limite de l'axe y
plt.xticks(visible=False) # On cache les ticks
sur x
plt.yticks(visible=False) # On cache les ticks
sur y

# Effet de cadre sur le zoom
mark_inset(ax, axins, loc1=3, loc2=4,
fc="None", ec="0.1")

plt.show()
```



Annotations sur le graphique

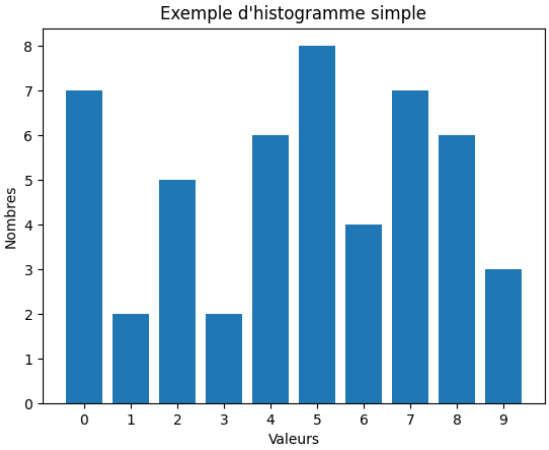
Il est possible d'ajouter des annotations sur le graphique avec la fonction `plt.annotate(s,(x,y))`.

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau pour le tracé
x = np.linspace(0, 2*np.pi, 50)
y = np.sin(x)

plt.plot(x, y, '+', label="Sinus")  # Ajout du des points uniquement
plt.ylabel('Amplitude')  # Titre de l'axe y
plt.xlabel("Angle (rad)")  # Titre de l'axe x
plt.title("Ma première figure")  # Titre du graphique
plt.grid(True)  # Affichage de la grille
plt.axvline(x=np.pi, ymin=0, ymax=1, color="red")  # Barre verticale
plt.axhline(y=-0.5, xmin=0, xmax=1, color="green")  # Barre horizontale

plt.scatter(0.5,-0.5) # Ajout d'un point
plt.annotate("Barre horizontale",(0,-0.4)) # Ajout d'une annotation


plt.show()  # Affichage d'une courbe
```



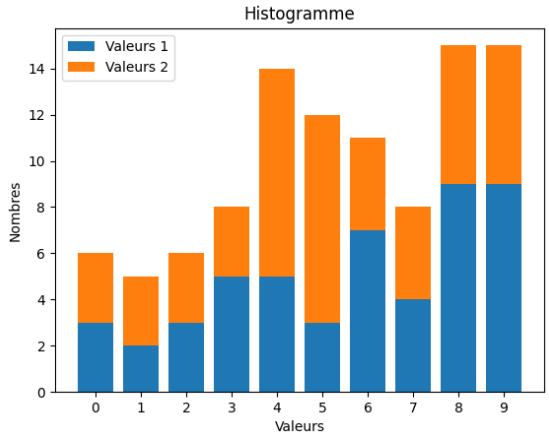
Histogramme simple

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau de 50 valeurs comprises entre 0 et 9
valeurs = np.random.randint(0, 10, 50)
print(valeurs)  # Affichage des valeurs
# Création d'un tableau avec les intervalles centrés sur la valeur entière
inter = [-0.5, 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5]

plt.hist(valeurs, bins=inter, rwidth=0.8)  #
Création de l'histogramme
plt.xlabel('Valeurs')
plt.xticks(np.arange(0, 10))
plt.ylabel('Nombres')
plt.title("Exemple d'histogramme simple")
plt.show()
```



Exemple avec 2 séries de valeurs

```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau de 50 valeurs comprises entre 0 et 9
valeurs = np.random.randint(0, 10, 50)
# Tableau de 50 valeurs comprises entre 0 et 9
valeurs2 = np.random.randint(0, 10, 50)
# Création d'un tableau avec les intervalles centrés sur la valeur entière
inter = np.linspace(-0.5, 9.5, 11)

plt.hist([valeurs, valeurs2], bins=inter, rwidth=0.8, stacked=True,
         label=['Valeurs 1', 'Valeurs 2'])  # Création de l'histogramme
plt.xlabel('Valeurs')
plt.xticks(np.arange(0, 10))
plt.ylabel('Nombres')
plt.title("Histogramme")
plt.legend()
plt.show()
```



Personnalisation des intervalles

Pour modifier les intervalles, il est possible d'utiliser plusieurs combinaisons des paramètres `range` et `bins`.

Paramètre	Détails
`bins`	Si c'est une valeur entière, défini le nombre de parts égal du paramètre `range` Si c'est une liste, un tableau ou une séquence, il définit les différents intervalles (valeur de gauche incluse et celle de droite exclue (sauf la dernière)). Les intervalles peuvent être de largeur inégale.
`range`	Valeurs min et max des intervalles

Exemples d'utilisation :

- `plt.hist(valeurs, range=(0,9), bins=9)` : les valeurs sont comprises entre 0 et 9 avec 9 intervalles de taille égale ([0,1[,[1,2[, ...,[8,9]) ;
- `plt.hist(valeurs, range=(0,9))` : identique au précédent ;
- `plt.hist(valeurs, bins=range(10))` : même résultat que précédemment, mais en indiquant explicitement les intervalles ;
- `plt.hist(valeurs, bins=np.linspace(-0.5,9.5,11))` : création des intervalles de manière explicite centrés sur la valeur entière.

Personnalisation de l'histogramme

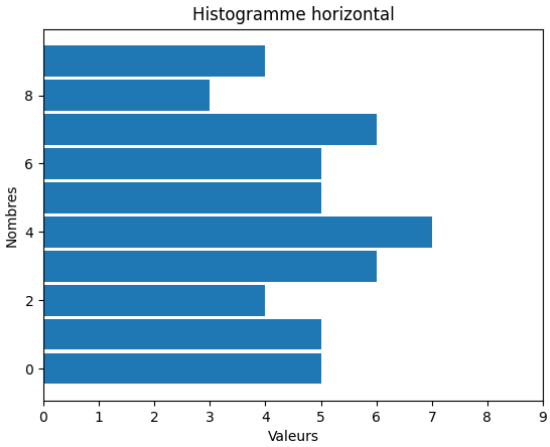
Les paramètres pour modifier l'apparence de l'histogramme sont disponibles ci-dessous :

Paramètre	Détails
`density=True`	Trace les fréquences plutôt que les nombres en ordonnée (somme vaut 1).
`weights=[1,2,1]`	Tableau de même dimension que les valeurs qui attribue un poids à chaque intervalle. S'applique uniquement si `density=True`.
`cumulative=True`	On construit l'histogramme du plus petit au plus grand est chaque barre prend ça valeur plus le cumul des valeurs plus petites (de gauche)
`histtype='bar'`	Défini le type d'histogramme à afficher : - 'bar' histogramme traditionnel avec des barres ; - 'barstacked' si on a plusieurs liste de valeurs, elles sont affichées empilées l'une sur l'autre et pas côte-à-côte ; - 'step' génère une courbe (contour) à partir de l'histogramme ; - 'stepfilled' génère une courbe (contour) et la zone entre le 0 et la valeur est coloriée.
`align='mid'`	Alignement de la barre par rapport au centre de l'intervalle : 'left', 'mid'ou 'right'.
`orientation='vertical'`	Orientation des barres : 'horizontal' ou 'vertical'.
`rwidth=0.8`	Largeur relative de la barre par rapport à l'intervalle (ici 80 %).
`log=True`	Utilise une échelle logarithmique.

Paramètre	Détails
`color='red'`	Couleur des barres.
`edgecolor='black'`	Couleur du cadre des barres.
`label='Série 1'`	Nom donné à la série. C'est le nom qui apparaît dans la légende.
`stacked=True`	Si il y a plusieurs séries de données, elles s'affichent empilées et non côte-à-côte.
`hatch='/'`	Hachurer les barres : '/', '\', '|', '-', '+', 'x', 'o', 'O', '.', '*'

Exemple barres horizontales

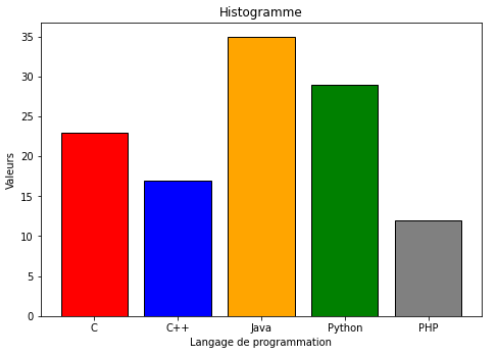
```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Tableau de 50 valeurs comprises entre 0 et 9
valeurs = np.random.randint(0, 10, 50)
# Création d'un tableau avec les intervalles centrés sur la valeur entière
inter = [-0.5, 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5]

plt.hist(valeurs, bins=inter, orientation='horizontal',
         rwidth=0.9)  # Création de l'histogramme
plt.xlabel('Valeurs')
plt.xticks(np.arange(0, 10))
plt.ylabel('Nombres')
plt.title("Histogramme horizontal")
plt.show()
```



Histogramme avec étiquettes personnalisées

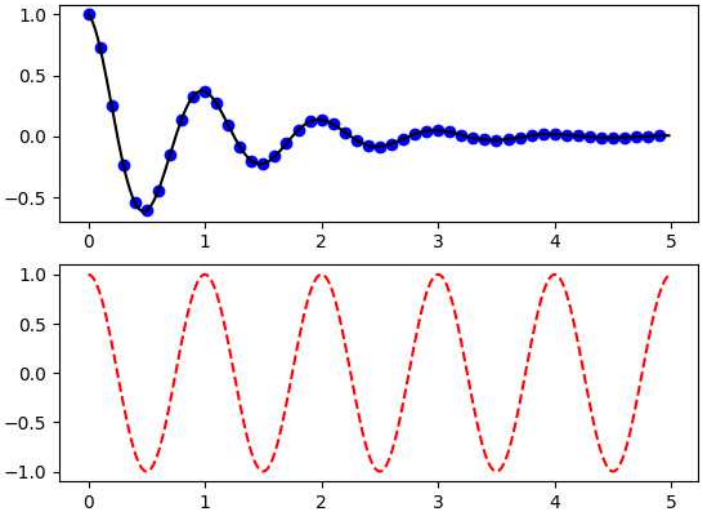
```
import matplotlib.pyplot as plt  # Module pour tracer les graphiques
import numpy as np

# Préparation de la figure
fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])

etiquettes = ['C', 'C++', 'Java', 'Python', 'PHP']
valeurs = [23, 17, 35, 29, 12]

# Affichage des données
ax.bar(etiquettes, valeurs)

plt.title("Histogramme")  # Titre du graphique
plt.ylabel('Valeurs')  # Titre de l'axe y
plt.xlabel('Langages de programmation')
plt.show()  # Affichage d'une courbe
```

Pour finir essayez de créer ce graphique :



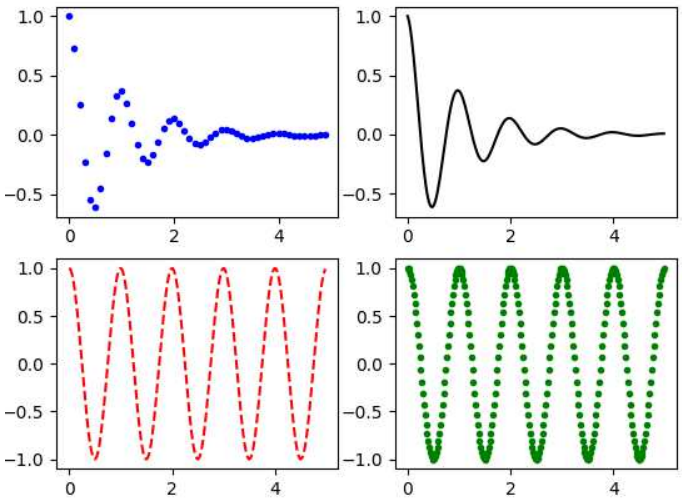
Plusieurs courbes dans la même figure avec un code orienté objet :

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

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.subplot(211)
plt.plot(t1, f(t1), "bo")
plt.plot(t2, f(t2), "k")

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), "r--
")
plt.show()
```



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

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

fig, axs = plt.subplots(2, 2)

axs[0, 0].plot(t1, f(t1), "b.")
axs[0, 1].plot(t2, f(t2), "k")
axs[1, 0].plot(t2, np.cos(2*np.pi*t2),
"r--")
axs[1, 1].plot(t2, np.cos(2*np.pi*t2),
"g.")

plt.show()
```

2 courbes par lignes sur 2 lignes



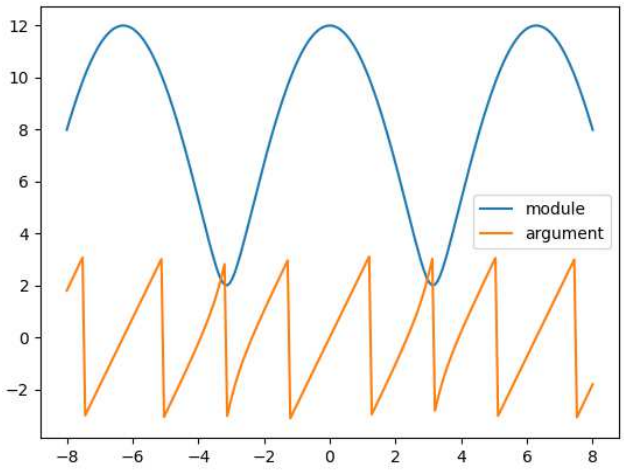
Représentation du module et de l'argument :

z= f(x)

z est complexe x est réel

$$f(x) = 5e^{i2x} + 7e^{i3x}.$$

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

x = np.linspace(-8, 8, 201)
z = 5*np.exp(2j*x) + 7*np.exp(3j*x)

plt.plot(x, np.abs(z), label="module")
plt.plot(x, np.angle(z), label="argument")
plt.legend()

plt.show()
```



Animation :

La fonction `FuncAnimation()` dispose d'un argument avec une étiquette appelée `interval`, qui est le temps en millisecondes entre deux appels de la fonction de mise à jour, ici `animate()`.

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

k = 2*np.pi
w = 2*np.pi
dt = 0.01

xmin = 0
xmax = 3
nbx = 151

x = np.linspace(xmin, xmax, nbx)

fig = plt.figure() # initialise la figure
line, = plt.plot([], [])
plt.xlim(xmin, xmax)
plt.ylim(-1, 1)

def animate(i):
    t = i * dt
    y = np.cos(k*x - w*t)
    line.set_data(x, y)
    return line,

ani = animation.FuncAnimation(fig, animate, frames=100,
interval=1, blit=True, repeat=False)
plt.show()
```