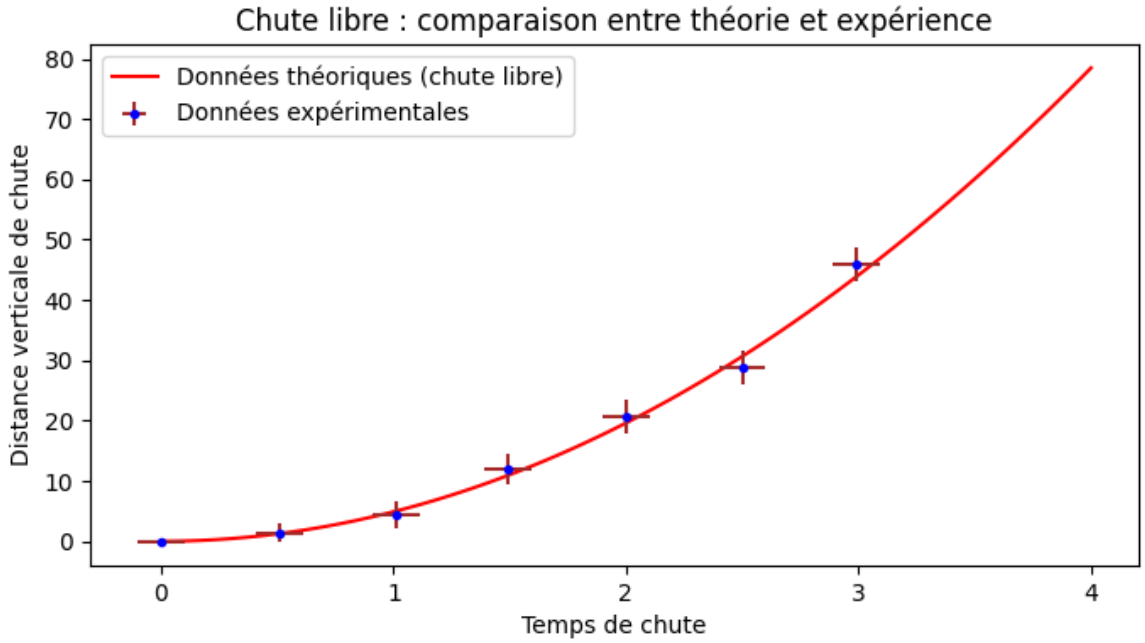
Chapitre 2

Représentation de courbes en deux dimensions et manipulation d'images (avec Matplotlib)

2.1 Introduction

La représentation graphique des données est souvent essentielle pour comprendre ces données qu'elles soient issues de mesures expérimentales ou d'un modèle théorique.



Le coeur de Python ne fournit pas d'outil dédié aux représentations graphiques et un package externe couramment utilisé dans cette optique est la **bibliothèque graphique Matplotlib** (http://matplotlib.org/). Il s'agit en fait d'une collection de bibliothèques graphiques permettant de générer des figures en 2D et en 3D. Matplotlib autorise le contrôle de tous les aspects d'une figure (titre, légendes, couleurs, inclusion de textes LaTeX, graphes multiples,

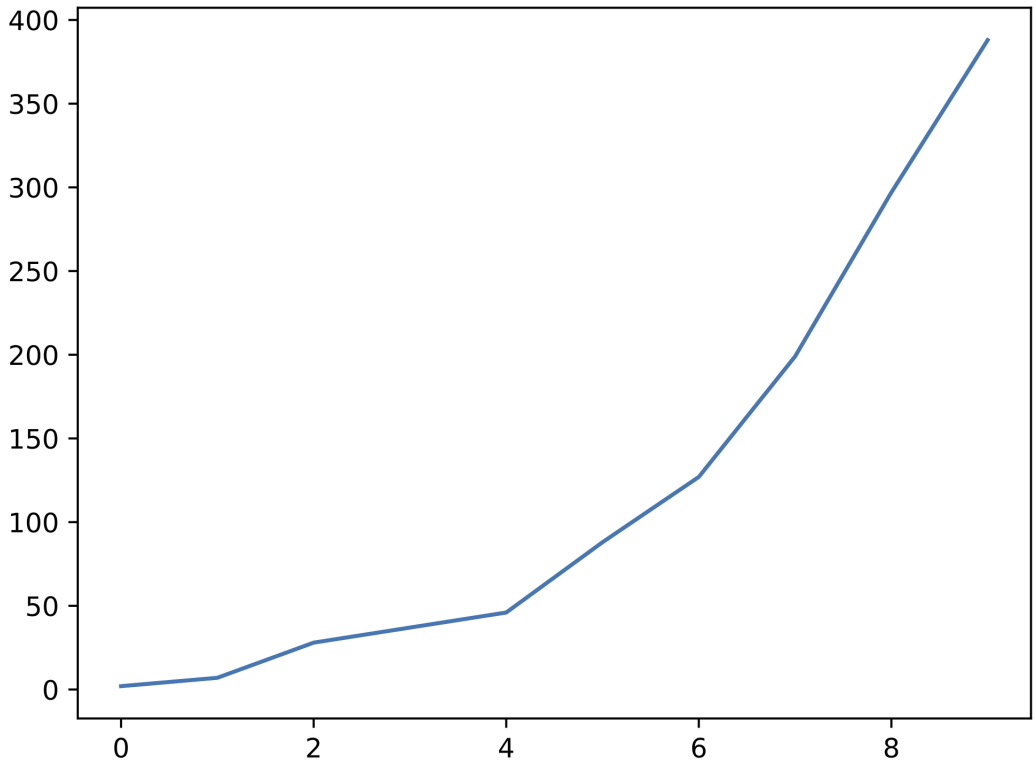
etc.) et permet une sortie de haute qualité dans différents formats (eps, pdf, jpeg, pgf, png, ps, raw, svg, tiff, etc.).
La bibliothèque Matplotlib est une bibliothèque externe qui doit en général être importée. Elle est souvent utilisée en association avec la bibliothèque Numpy. Un programme destiné à une représentation graphique en 2D contiendra donc souvent les deux lignes suivantes :

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

2.2 Les fonctions `plt.plot` et `plt.show`

Trois lignes de code suffisent en général pour obtenir une première visualisation d'un ensemble de données :

```
1 import matplotlib.pyplot as plt
2 plt.plot([2,7,28,37,46,88,127,199,297,388])
3 plt.show()
```



Par défaut, la fonction `plt.plot([x],y)` construit des **segments de droites** entre des points $P_0(x_0, y_0)$, $P_1(x_1, y_1)$, ..., $P_n(x_n, y_n)$. Elle admet deux listes (ou deux tableaux) comme arguments :

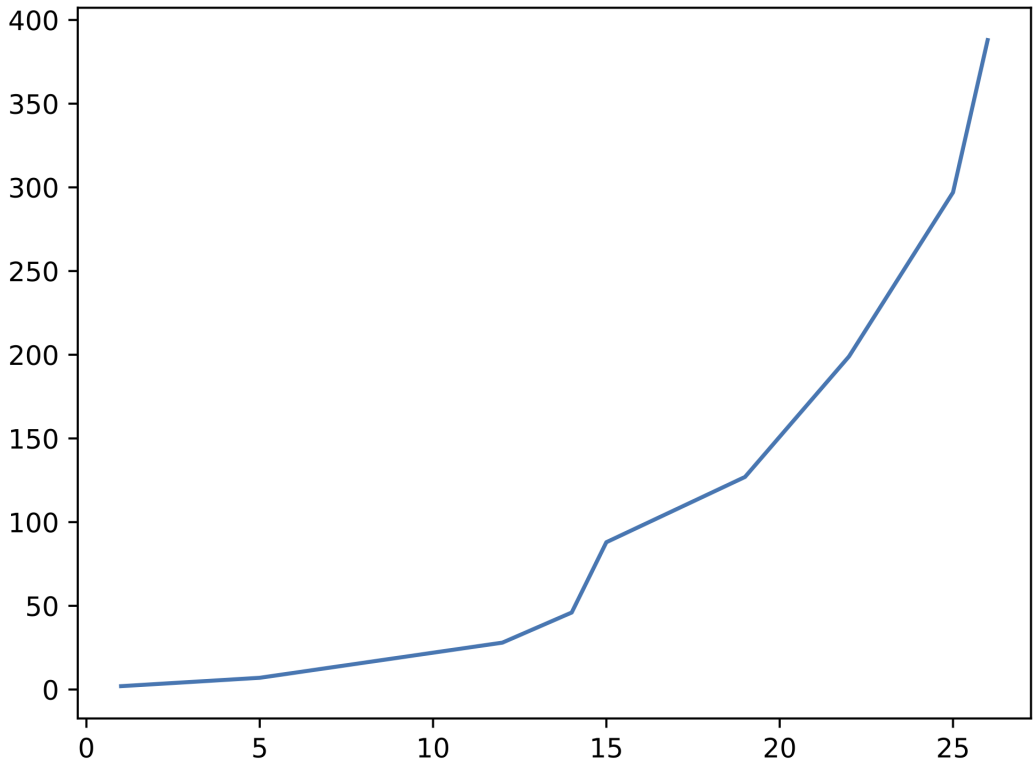
- une liste (ou un tableau) $X = [x_0, x_1, \ldots, x_n]$ renfermant les abscisses x_i ;
- une liste (ou un tableau) $Y = [y_0, y_1, \ldots, y_n]$ contenant les ordonnées y_i.

Si, comme dans l'exemple ci-dessus, la liste X manque, la fonction utilise les valeurs $0, 1, \ldots, n$ en abscisse.

La fonction `plt.show()` permet d'afficher la représentation graphique. Selon l'environnement Python utilisé (terminal, Jupyter Notebook, etc.), cette fonction peut parfois être appelée par défaut.

Exemple : représentation de données expérimentales

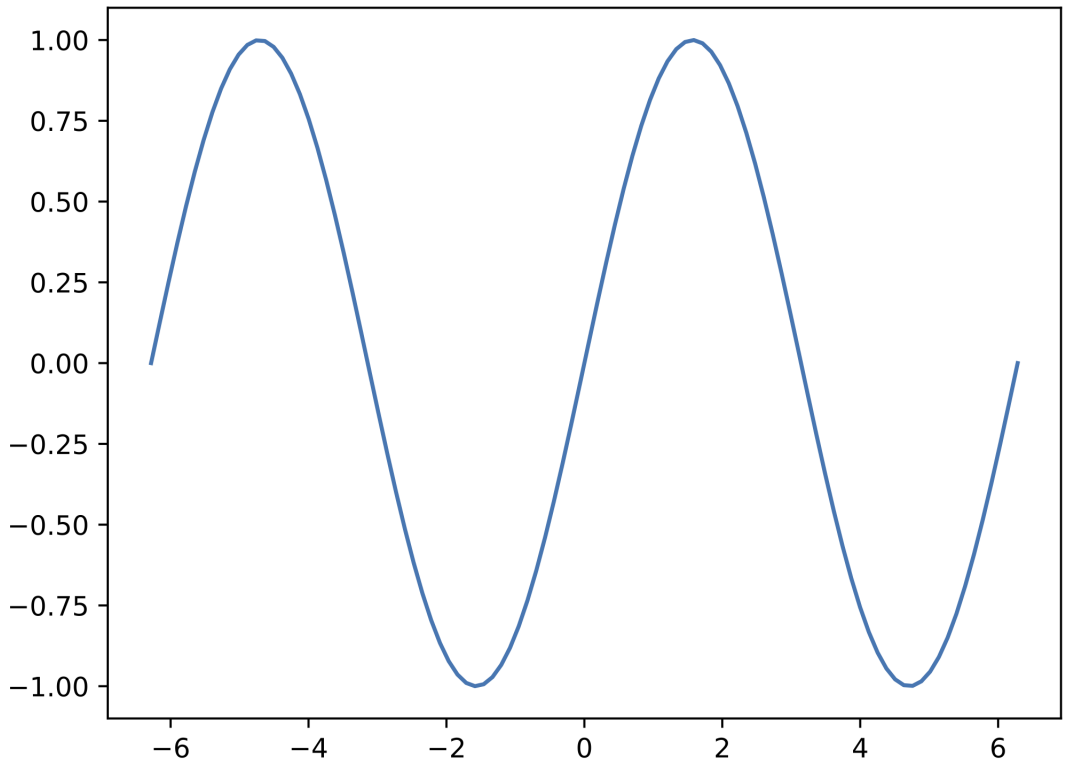
```
1 import matplotlib.pyplot as plt
2 X = [1,5,12,13,14,15,19,22,25,26]
3 Y = [2,7,28,37,46,88,127,199,297,388]
4 plt.plot(X,Y)
5 plt.show()
```



2.3 Utilisation de `np.linspace`, `np.logspace` et `np.arange` (fonctions de NumPy)

Comme le montrent les trois exemples ci-dessous, les fonctions `np.linspace`, `np.logspace` et `np.arange` de NumPy permettant de générer des tableaux unidimensionnels dont les **éléments** sont **régulièrement espacés** sont très utiles pour **représenter des fonctions mathématiques**.
Exemple 1 : représentation de la fonction $\sin(x)$ **dans l'intervalle** $[-2\pi, 2\pi]$

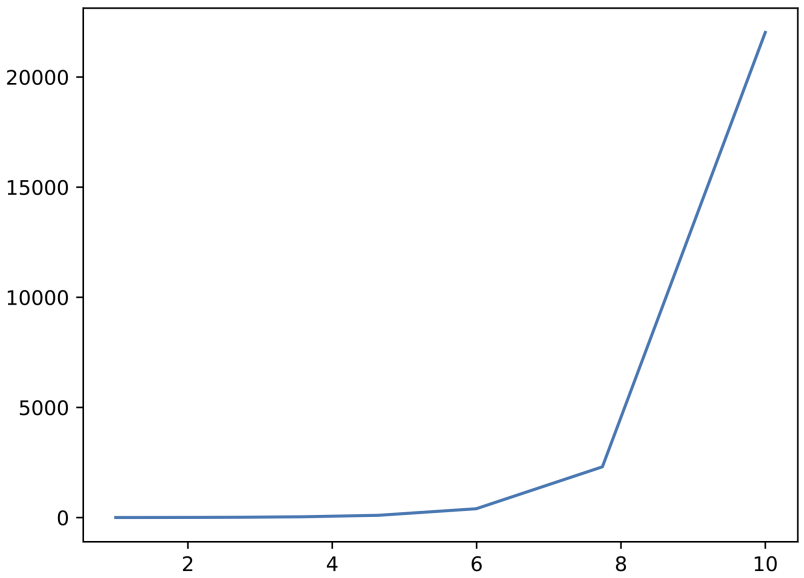
```
import matplotlib.pyplot as plt
import numpy as np
X = np.linspace(-2*np.pi,2*np.pi,100)
Y = np.sin(X)
plt.plot(X,Y)
plt.show()
```

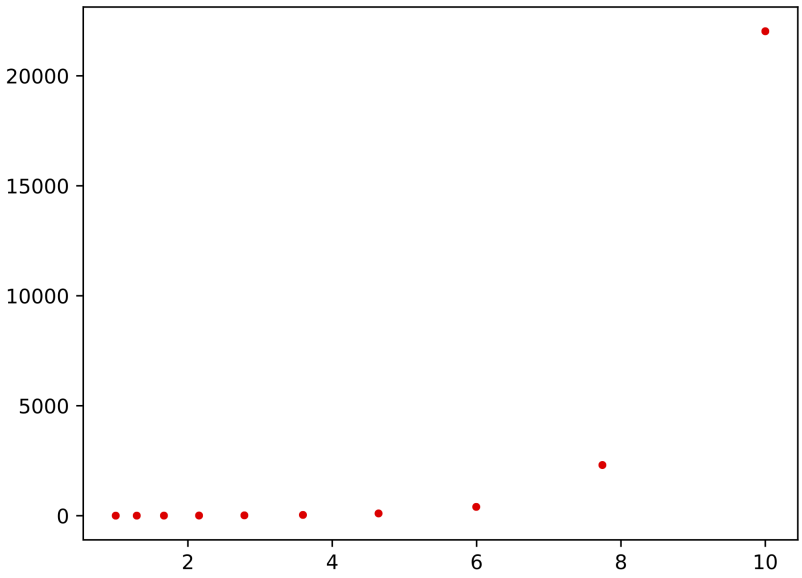


Pour représenter la fonction $sin(x)$ de manière "lisse", nous avons créé deux tableaux correspondant à un nombre suffisant de points $P(x, y)$. Ici, nous avons estimé que 100 points étaient suffisants.
Les bords de l'intervalle, -2π et $+2\pi$, sont inclus dans le tableau.

Exemple 2 : représentation de la fonction $\exp(x)$ dans l'intervalle $[0, 10]$ sur une échelle logarithmique

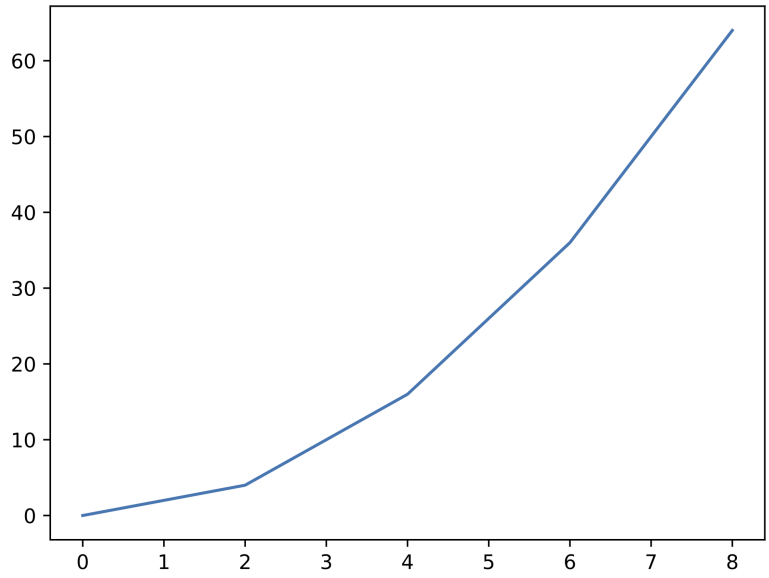
```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 X = np.logspace(0,1,10)
4 Y = np.exp(X)
5 plt.plot(X,Y)
6 #plt.plot(X,Y,'r.')
7 plt.show()
```





Exemple 3 : représentation de la fonction x^2 dans l'intervalle $[0, 8]$

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 X = np.arange(0,10,2)
4 Y = pow(X,2)
5 plt.plot(X,Y)
6 plt.show()
```



2.4 Exemple détaillé

Nous allons étudier quelques-unes des possibilités offertes par Matplotlib à travers les données d'une **expérience de chute libre** (fichier texte `ChuteLibreData.txt` que nous avons déjà utilisé) :

```
Données collectées (masse m en chute libre)
Date : 20 février 2025

No de la mesure  Temps[s]  Distance parcourue[m]  Incertitude[m]
0 0 0 0
1 0.51 1.41 1.5
2 1.01 4.39 2.3
3 1.49 11.9 2.5
4 2. 20.7 2.9
5 2.5 28.8 2.7
6 2.99 45.9 2.9
```

Nous allons lire le fichier de données, représenter les données en les comparant avec le modèle théorique (équation horaire de la chute libre) et sauvegarder la figure obtenue :

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  # lecture des données depuis un fichier texte
4  tdata, ydata = np.loadtxt('ChuteLibreData.txt', usecols = (1,2),
   ↪  skiprows=4, unpack=True)
5  # création des tableaux abscisses et ordonnées pour la courbe
   ↪  théorique
6  t = np.linspace(0, 4, 100)
7  y = 0.5*9.81*t**2
8  # création de la représentation
9  plt.figure('Ma fenêtre de représentation', figsize = (8,4) )
10 plt.plot(t, y, 'r-', label='Données théoriques (chute libre)')
11 plt.plot(tdata, ydata, 'bo', label="Données expérimentales")
12 plt.xlabel('Temps de chute')
13 plt.ylabel('Distance verticale de chute')
14 plt.legend(loc='upper left')
15 plt.title('Chute libre : comparaison entre théorie et expérience')
16 # sauvegarder la représentation dans un fichier png
17 plt.savefig('ChuteLibre.png')
18 # afficher la représentation
19 plt.show()
```

Les fonctions des lignes 9 à 15 permettent de construire la représentation :

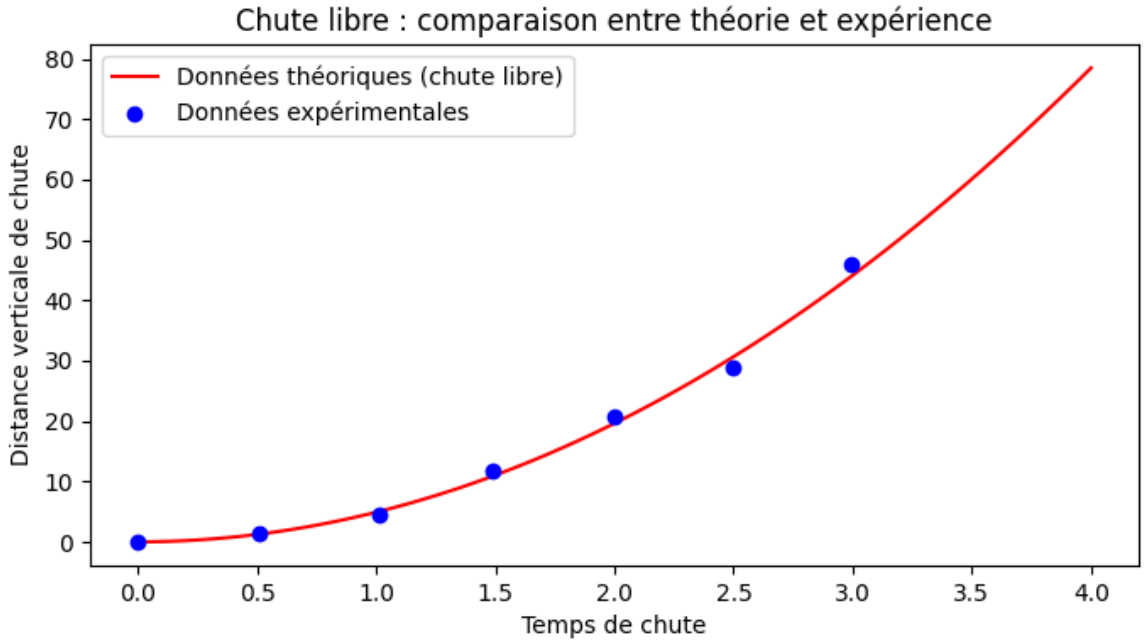
- `plt.figure()` crée une fenêtre destinée à contenir la figure.

Il est possible de donner un nom à cette figure et de préciser ses dimensions par l'intermédiaire d'un argument nommé supplémentaire (*kwarg* `figsize`). Par défaut, la fenêtre fait 6.4 inches de large et 4.8 inches de hauteur (1 inch $\equiv$ 1 pouce = 2.54 cm).

- `plt.plot(t, y, arguments optionnels)` représente les données t-y qui se trouvent dans les tableaux t et y. Le troisième argument est une chaîne de caractères qui spécifie la couleur et le type de ligne ou de symbole qui doivent être utilisés pour représenter les données.
  Le *kwarg* `label` est une chaîne de caractères utilisée ensuite par la fonction `legend`.

- `plt.xlabel(xlabel)` utilise la chaîne de caractères donnée en argument pour spécifier la légende de l'axe t (abscisses).
- `plt.ylabel(ylabel)` utilise la chaîne de caractères donnée en argument pour spécifier la légende de l'axe y (ordonnées).
- `plt.legend()` crée la légende de la représentation graphique. Le *kwarg* `loc` permet de préciser la localisation de la légende.
- `plt.title(label)` ajoute un titre à la représentation graphique.

Les lignes 17 et 19 permettent de sauvegarder (`plt.savefig(fname)`) et d'afficher (`plt.show()`) la représentation.
Figure sauvegardée :

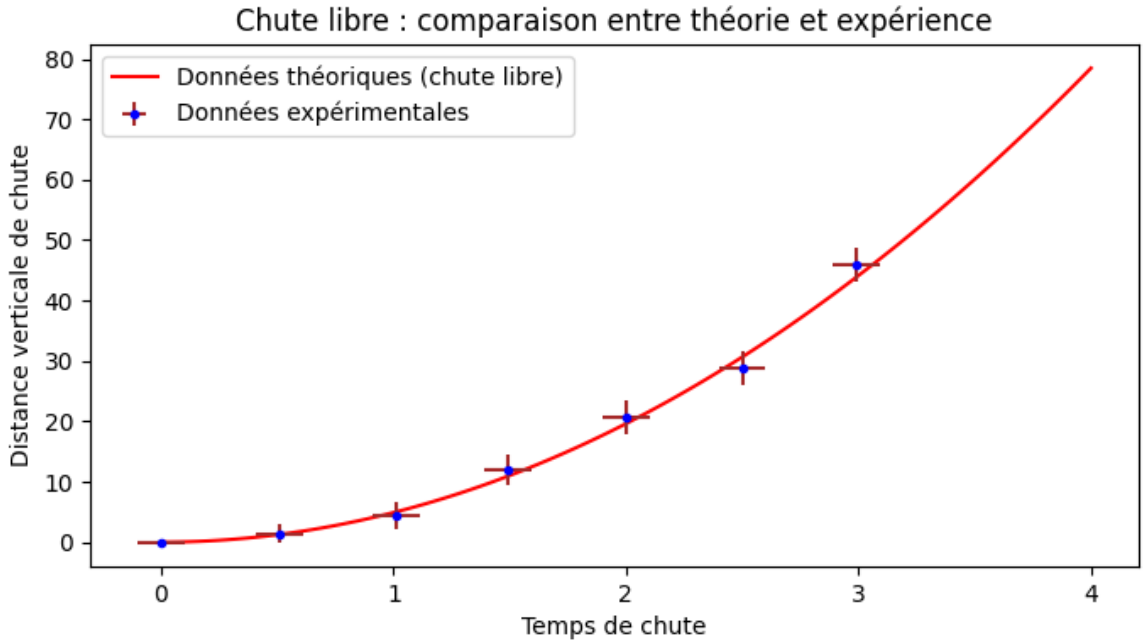


Il est possible de compléter la représentation en ajoutant des barres d'erreur à l'aide de la fonction `plt.errorbar(x,y)` :

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 # lecture des données depuis un fichier texte
4 tdata, ydata, erreur = np.loadtxt('ChuteLibreData.txt', usecols =
  ↪  (1,2,3), skiprows=4, unpack=True)
```

```
5  # création des tableaux abscisses et ordonnées pour la courbe
   ↪ théorique
6  t = np.linspace(0, 4, 100)
7  y = 0.5*9.81*t**2
8  # création de la représentation
9  plt.figure('Ma fenêtre de représentation', figsize = (8,4) )
10 plt.plot(t, y, 'r-', label='Données théoriques (chute libre)')
11 plt.errorbar(tdata, ydata, fmt='b.', label="Données
   ↪ expérimentales", xerr=0.1, yerr=erreur, ecolor='brown')
12 plt.xlabel('Temps de chute')
13 plt.ylabel('Distance verticale de chute')
14 plt.legend(loc='upper left')
15 plt.title('Chute libre : comparaison entre théorie et expérience')
16 # sauvegarder la représentation dans un fichier png
17 plt.savefig('ChuteLibreAvecErreur.png')
18 # afficher la représentation
19 plt.show()
```

La ligne 11 permet d'afficher les barres d'erreur en abscisse et en ordonnée (l'erreur en abscisse est ici supposée constante et l'erreur en ordonnée est lue à la ligne 4 à partir du fichier `ChuteLibreData.txt`).



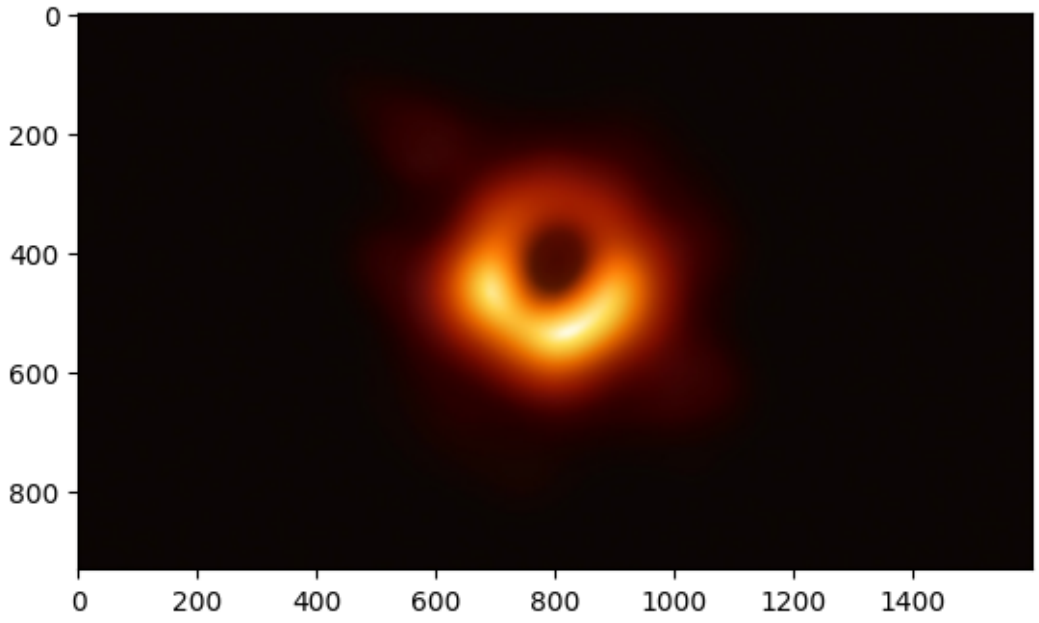
2.5 Affichage, manipulation et sauvegarde d'images

Outre les représentations graphiques de fonctions et de données, Matplotlib permet de manipuler très facilement des images. Ainsi, le code suivant permet par exemple d'afficher

(et éventuellement de manipuler) l'image du trou noir déjà évoquée en introduction. Ici, l'image `blackhole.jpg` chargée ne contient pas de canal lié à la transparence et est de la forme `(932, 1600, 3)`.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  # lecture et conversion de l'image
4  import matplotlib.image as mpimg
5  bhM87 = mpimg.imread("blackhole.jpg") # forme : (932, 1600, 3)
6  # affichage de l'image
7  plt.imshow(bhM87)
8  plt.show()
9  # traitement de l'image
10 bhM87[100:400,100:200,:] = [0,0,255]
11 # sauvegarde de l'image
12 mpimg.imsave("blackhole_modifie.png", bhM87)
```

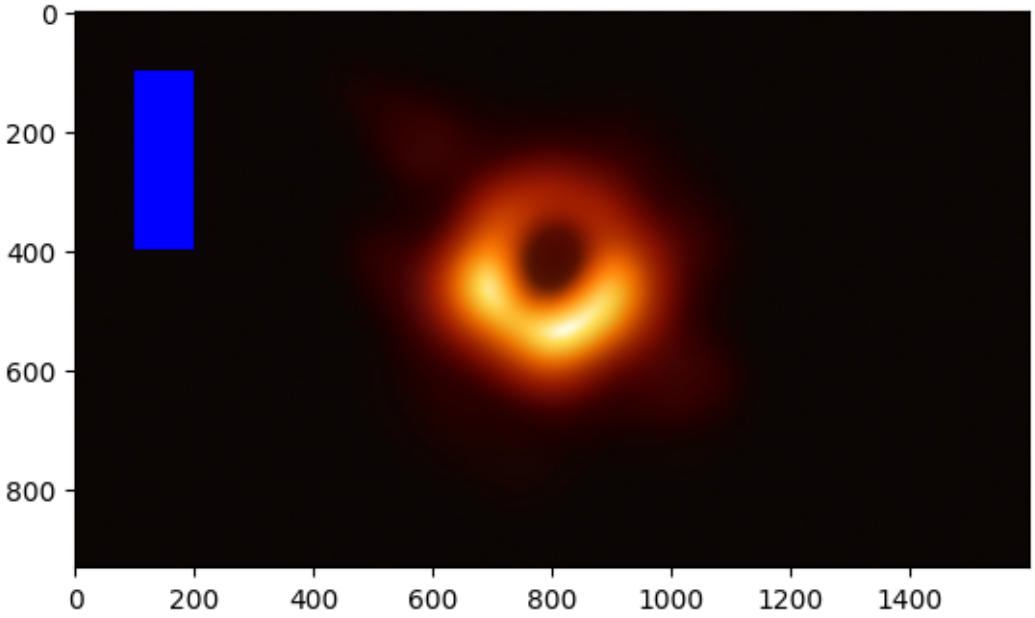
Affichage de l'image originale Black Hole M87 (*Event Horizon Telescope Collaboration*) :



Plus précisément, . . .

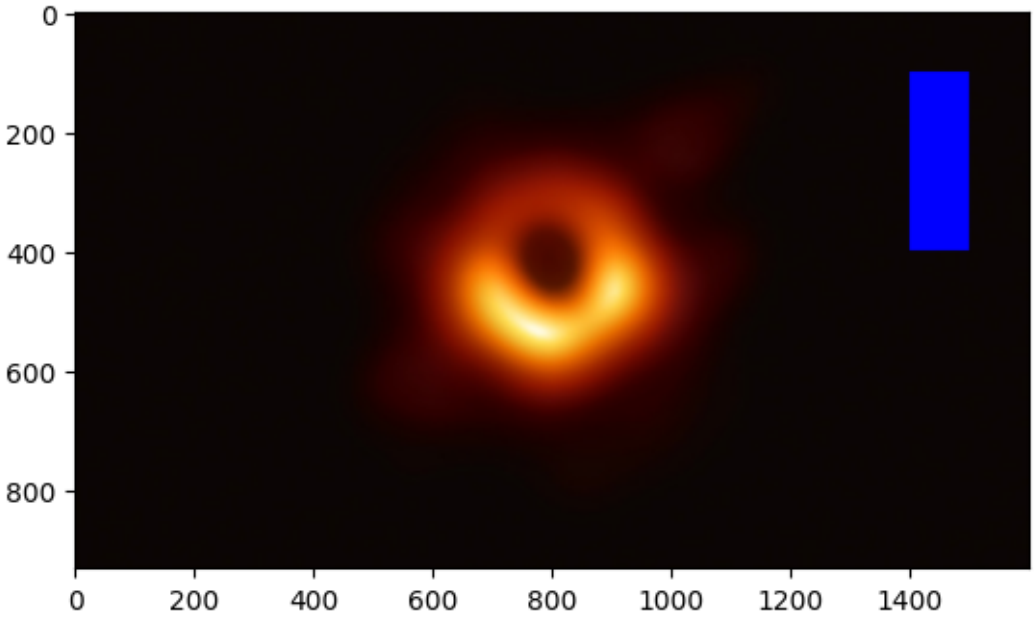
- . . . les lignes 4 et 5 permettent d'importer l'image dans un tableau `numpy` ;
- . . . les lignes 7 et 8 permettent d'afficher le tableau `numpy` comme une image ;
- . . . la ligne 10 traite l'image en modifiant certains éléments du tableau `bhM87` de manière à faire apparaître un rectangle bleu ;
- . . . la ligne 12 permet de sauvegarder le tableau manipulé sous la forme d'une image.

Image manipulée (ajout d'un rectangle bleu) :



Le fait de travailler avec des tableaux numpy permet de modifier (traiter) très facilement les images. Ainsi, il est par exemple facile de retourner horizontalement l'image obtenue ci-dessus en rajoutant une ligne de code qui inverse les colonnes du tableau :

```
1 bhM87 = bhM87[:,::-1,:]
```



Il est également très facile d'augmenter la luminosité du canal rouge en complétant le code avec les deux lignes suivantes (la seconde ligne permet de s'assurer qu'aucune valeur dans la troisième dimension du tableau ne dépasse 255) :

```
bhM87_brighter_red = bhM87 + [100,0,0]
bhM87_brighter_red[bhM87_brighter_red > 255] = 255
```