Introduction à l'utilisation de numpy et matplotlib

Séna APEKE

UL/BERIIA

23/03/2022

1. **Numpy**
   - Création d'arrays numpy

2. **Opérations sur les arrays**
   - Algèbre linéaire avec np.linalg
   - Le format .npy

3. **Matplotlib**
   - Introduction
   - Le module Pyplot
   - Le module Image

Introduction

NumPy (Numerical Python) est une bibliothèque Python open source utilisée dans presque tous les domaines de la science et de l'ingénierie. C'est la norme universelle pour travailler avec des données numériques en Python, et c'est au cœur des écosystèmes scientifiques Python et PyData. Les utilisateurs de NumPy incluent tout le monde, des codeurs débutants aux chercheurs expérimentés effectuant des recherches et développements scientifiques et industriels de pointe. L'API NumPy est largement utilisée dans Pandas, SciPy, Matplotlib, scikit-learn, scikit-image et la plupart des autres packages scientifiques et de science des données.

Introduction

NumPy (Numerical Python) est une bibliothèque Python open source utilisée dans presque tous les domaines de la science et de l'ingénierie. C'est la norme universelle pour travailler avec des données numériques en Python, et c'est au cœur des écosystèmes scientifiques Python et PyData. Les utilisateurs de NumPy incluent tout le monde, des codeurs débutants aux chercheurs expérimentés effectuant des recherches et développements scientifiques et industriels de pointe. L'API NumPy est largement utilisée dans Pandas, SciPy, Matplotlib, scikit-learn, scikit-image et la plupart des autres packages scientifiques et de science des données.

La bibliothèque NumPy contient des tableaux multidimensionnels et des structures de données matricielles. Il fournit ndarray, un objet de tableau homogène à n dimensions, avec des méthodes pour fonctionner efficacement dessus. NumPy peut être utilisé pour effectuer une grande variété d'opérations mathématiques sur des tableaux.

Importer la biblio numpy

```
import numpy as np
```

Quelle est la différence entre une liste python et un tableau numpy

NumPy vous offre une vaste gamme de moyens rapides et efficaces de créer des tableaux et de manipuler des données numériques à l'intérieur de ceux-ci. Alors qu'une liste Python peut contenir différents types de données dans une seule liste, tous les éléments d'un tableau NumPy doivent être homogènes. Les opérations mathématiques censées être effectuées sur des tableaux seraient extrêmement inefficaces si les tableaux n'étaient pas homogènes.

Pourquoi utiliser numpy

Les tableaux NumPy sont plus rapides et plus compacts que les listes Python. Un tableau consomme moins de mémoire et est pratique à utiliser. NumPy utilise beaucoup moins de mémoire pour stocker les données et fournit un mécanisme de spécification des types de données. Cela permet d'optimiser encore plus le code.

C'est quoi un array ?

Un array est une structure de données centrale de la bibliothèque NumPy. Un array est une grille de valeurs et il contient des informations sur les données brutes, comment localiser un élément et comment interpréter un élément. Il dispose d'une grille d'éléments qui peuvent être indexés de différentes manières. Les éléments sont tous du même type, appelé array dtype.

Un array peut être indexé par un tuple d'entiers non négatifs, par des booléens, par un autre array ou par des entiers. Le rang du array est le nombre de dimensions. La forme du array est un tuple d'entiers donnant la taille du array le long de chaque dimension.
Une façon d'initialiser les array NumPy est à partir de listes Python, en utilisant des listes imbriquées pour les données à deux dimensions ou plus.

Exemples

$\ggg a = np.array([1, 2, 3, 4, 5, 6])$
ou
$\ggg a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])$

Différentes façon de créer un array

Utilisation de : np.array(), np.zeros(), np.ones(), np.empty(), np.arange(), np.linspace(), dtype.
import numpy as np

$\ggg a = np.array([1, 2, 3])$

$\ggg a = np.zeros(2); b = np.zeros((2, 3))$

a = np.ones(2) ; b = np.ones((2,2))

$\ggg np.empty(2)$
initialise avec des valeurs array([3.14, 42.]) may vary

$\ggg np.arange(4)$
array([0, 1, 2, 3])

Vous pouvez également utiliser np.linspace() pour créer un array avec des valeurs espacées linéairement dans un intervalle spécifié :

$\ggg np.linspace(0, 10, num = 5)$

array([0. , 2.5, 5. , 7.5, 10.])

Bien que le type de données par défaut soit à virgule flottante (np.float64), vous pouvez spécifier explicitement le type de données souhaité à l'aide du mot-clé dtype.

$\ggg x = np.ones(2, dtype = np.int64)$

Adding and sorting elements

Trier un élément est simple avec np.sort(). Vous pouvez spécifier l'axe, le type et l'ordre lorsque vous appelez la fonction.

```
>>> arr = np.array([2, 1, 5, 3, 7, 4, 6, 8])
>>> np.sort(arr)
array([1, 2, 3, 4, 5, 6, 7, 8])
```

En plus de sort, qui renvoie une copie triée d'un tableau, vous pouvez utiliser :

- **argsort**, qui est un tri indirect le long d'un axe spécifié,
- **lexsort**, qui est un tri stable indirect sur plusieurs clés,
- **searchsorted**, qui trouvera des éléments dans un tableau trié, et
- **partition**, qui est un tri partiel.

concatenate

```
>>> a = np.array([1, 2, 3, 4])
>>> b = np.array([5, 6, 7, 8])
>>> np.concatenate((a, b))
array([1, 2, 3, 4, 5, 6, 7, 8])
```

```
>>> x = np.array([[1, 2], [3, 4]])
>>> y = np.array([[5, 6]])
>>> np.concatenate((x, y), axis = 0)
array([[1, 2],
       [3, 4],
       [5, 6]])
```

How do you know the shape and size of an array?

ndarray.ndim vous indiquera le nombre d'axes, ou dimensions, du tableau.
ndarray.size vous indiquera le nombre total d'éléments du tableau. C'est le produit des éléments de la forme du tableau.
ndarray.shape affichera un tuple d'entiers indiquant le nombre d'éléments stockés le long de chaque dimension du tableau. Si, par exemple, vous avez un tableau 2D avec 2 lignes et 3 colonnes, la forme de votre tableau est (2, 3).

```
>>> a = np.array([[[0, 1, 2, 3],
                   [4, 5, 6, 7]],
                  [[0, 1, 2, 3],
                   [4, 5, 6, 7]],
                  [[0 ,1 ,2, 3],
                   [4, 5, 6, 7]]])
```

La dimension de a

```
>>> a.ndim
3
```

Le nombre d'éléments du array

```
>>> a.size
24
```

La shape de a

```
>>> a.shape
(3, 2, 4)
```

np.reshape et np.ravel

```
>>> a = np.arange(6)
>>> print(a)
1 2 3 4 5
```

```
>>> b = a.reshape(3, 2)
>>> print(b)
[[01]
[23]
[45]]
```

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> np.ravel(x)
array([1, 2, 3, 4, 5, 6])
>>> x.reshape(-1)
array([1, 2, 3, 4, 5, 6])
```

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]])
>>> np.reshape(a, (3, -1))
array([[1, 2],
       [3, 4],
       [5, 6]])
```

PS : pou plus d'info
https://numpy.org/doc/stable/reference/generated/numpy.reshape.html

Comment créer un tableau à partir de données existantes

```
>>> a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> arr1 = a[3 : 8]
>>> arr1
array([4, 5, 6, 7, 8])
```

```
>>> a1 = np.array([[1, 1], [2, 2]])
>>> a2 = np.array([[3, 3], [4, 4]])
>>> np.vstack((a1, a2))
>>> array([[1, 1],
           [2, 2],
           [3, 3],
           [4, 4]])
```

```
>>> np.hstack((a1, a2))
>>> array([[1, 1, 3, 3],
           [2, 2, 4, 4]])
```

```
>>> x = np.arange(1, 25).reshape(2, 12)
>>> x
>>> array([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
           [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]])
```

```
>>> np.hsplit(x, 3)
>>> [array([[1, 2, 3, 4],
           [13, 14, 15, 16]]), array([[ 5, 6, 7, 8],
           [17, 18, 19, 20]]), array([[ 9, 10, 11, 12],
           [21, 22, 23, 24]])]
```

```
>>> np.hsplit(x, (3, 4))
>>> [array([[1, 2, 3],
           [13, 14, 15]]), array([[ 4],
           [16]]), array([[ 5, 6, 7, 8, 9, 10, 11, 12],
```

Résoudre des systèmes linéaires avec np.solve

$$x0 + 2 * x1 + x2 = 4$$
$$x1 + x2 = 3$$
$$x0 + x2 = 5$$

Nous pouvons exprimer ce système comme une équation matricielle A * x = b avec:

```
A = np.array([[1, 2, 1],
              [0, 1, 1],
              [1, 0, 1]])
b = np.array([4, 3, 5])
```

Ensuite, utilisez np.linalg.solve pour résoudre x :

```
x = np.linalg.solve(A, b)
# Out: x = array([ 1.5, -0.5, 3.5])
```

A doit être un carré et une matrice complète: toutes ses lignes doivent être linéairement indépendantes. A doit être inversible / non singulier (son déterminant n'est pas zéro). Par exemple, si une ligne de A est un multiple d'un autre, l'appel de linalg.solve soulèvera LinAlgError: Singular matrix :

```
A = np.array([[1, 2, 1],
              [2, 4, 2],
              [1, 0, 1]])
b = np.array([4,8,5])
```

De tels systèmes peuvent être résolus avec np.linalg.lstsq

NPY

Le format **.npy** est le format standart de fichier binaire des tableau numpy pour les persistence de toutes ses infos sur le disque.
Le format stocke toutes les informations de forme et de type nécessaires pour reconstruire correctement le tableau, même sur une autre machine avec une architecture différente. Le format est conçu pour être aussi simple que possible tout en atteignant ses objectifs limités.

Le format **.npz** est le format standard pour conserver plusieurs tableaux NumPy sur le disque. Un fichier .npz est un fichier zip contenant plusieurs fichiers .npy, un pour chaque tableau.

Les fichiers npy et npz conservent des informations telles que le type de données (dtype) et la forme.

Par exemple, les tableaux multidimensionnels de trois dimensions ou plus ne peuvent pas être directement enregistrés dans des fichiers CSV sans conversion de forme. Cependant, les fichiers npy et npz enregistrent ces tableaux tels quels, préservant à la fois leur structure et leur précision sans arrondir les décimales.

Bien que les formats npy et npz soient publiquement documentés, leur utilisation est principalement limitée à NumPy.

Contrairement aux fichiers CSV, ces fichiers ne peuvent pas être ouverts et modifiés dans d'autres applications pour une révision rapide du contenu.

Enregistrez un tableau dans un fichier npy : np.save()

np.save() enregistre un seul tableau dans un fichier npy.

```
a = np.arange(6, dtype=np.int8).reshape(1, 2, 3)
print(a)
# [[[0 1 2]
# [3 4 5]]]

print(a.shape)
# (1, 2, 3)

print(a.dtype)
# int8
```

Spécifiez le chemin du fichier, soit sous forme de chaîne, soit sous forme d'objet pathlib.Path, comme premier argument et le ndarray à enregistrer comme second.

```
np.save('data/temp/np_save', a)
```

Le fichier est enregistré sous le chemin spécifié avec une extension .npy. Si le chemin se termine déjà par .npy, il reste inchangé.

np.load()

Le chargement d'un fichier npy avec np.load() renvoie le tableau enregistré sous forme de ndarray, en préservant son type et sa forme de données d'origine.

```
a_load = np.load('data/temp/np_save.npy')
print(a_load)
# [[[0 1 2]
# [3 4 5]]]

print(a_load.shape)
# (1, 2, 3)

print(a_load.dtype)
# int8
```

Enregistrez plusieurs tableaux dans un fichier npz : np.savez()

np.savez() enregistre plusieurs tableaux dans un seul fichier npz, en préservant le type et la forme des données, similaire à npy.
Considérez les deux tableaux suivants à titre d'exemple.

$a_1 = np.arange(5)$
print(a1)
# [0 1 2 3 4]

$a_2 = np.arange(5, 10)$
print(a2)
# [5 6 7 8 9]

Spécifiez le chemin du fichier sous forme de chaîne ou d'objet pathlib.Path, suivi de tableaux à enregistrer, séparés par des virgules. Bien que cet exemple montre l'enregistrement de deux tableaux, vous pouvez en spécifier trois ou plus.

np.savez('data/temp/np_savez', a_1, a_2)

Le fichier est enregistré sous le chemin spécifié avec une extension .npz. Si le chemin se termine déjà par .npz, il reste inchangé.

Comme les fichiers npy, les fichiers npz sont également chargés à l'aide de np.load(), mais ils renvoient un objet NpzFile.

```
npz = np.load('data/temp/np_savez.npz')
print(type(npz))
# <class 'numpy.lib.npyio.NpzFile'>
```

Accédez aux tableaux stockés en spécifiant leurs noms entre []. Les noms de chaque tableau peuvent être vérifiés à l'aide de l'attribut files.

```
print(npz.files)
# ['arr_0', 'arr_1']

print(npz['arr_0'])
# [0 1 2 3 4]

print(npz['arr_1'])
# [5 6 7 8 9]
```

L'utilisation d'arguments de mots-clés avec np.savez() permet d'attribuer des noms personnalisés aux tableaux.

```
np.savez('data/temp/np_savez_kw', x=a1, y = a2)

npz_kw = np.load('data/temp/np_savez_kw.npz')
print(npz_kw.files)
# ['x','y']

print(npz_kw['x'])
# [01234]

print(npz_kw['y'])
# [56789]
```

Bien que cela puisse être moins courant, il est également possible de nommer uniquement certains tableaux à l'aide d'arguments de mots clés (keyword arguments).

```
np.savez('data/temp/np_savez_kw2', a_1, y = a_2)

npz_kw2 = np.load('data/temp/np_savez_kw2.npz')
print(npz_kw2.files)
# ['y','arr_0']

print(npz_kw2['arr_0'])
# [01234]

print(npz_kw2['y'])
# [56789]
```

Enregistrez plusieurs tableaux dans un fichier npz compressé : np.savez_compressed()

np.savez_compressed() fonctionne comme np.savez() mais compresse les tableaux pour réduire la taille du fichier.
L'extension de fichier de np.savez_compressed() est .npz, identique à np.savez(), et peut être chargée de la même manière avec np.load().

```
np.savez_compressed('data/temp/np_savez_compressed', a1, a2)
npz_comp = np.load('data/temp/np_savez_compressed.npz')
print(type(npz_comp))
# <class 'numpy.lib.npyio.NpzFile'>
print(npz_comp.files)
# ['arr_0', 'arr_1']
print(npz_comp['arr_0'])
# [01234]
print(npz_comp['arr_1'])
# [56789]
```

Les arguments de mots clés sont également pris en charge.

```
np.savez_compressed('data/temp/np_savez_comp_kw', x=a1, y = a2)

npz_comp_kw = np.load('data/temp/np_savez_comp_kw.npz')
print(npz_comp_kw.files)
# ['x','y']

print(npz_comp_kw['x'])
# [01234]

print(npz_comp_kw['y'])
# [56789]
```

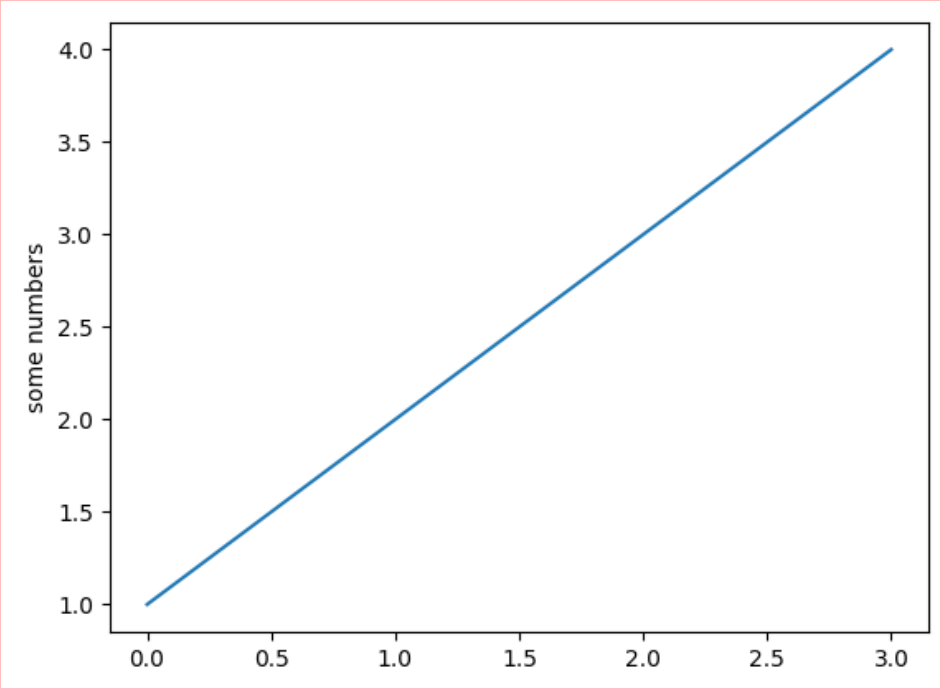
Introduction

Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous formes de graphiques5. Elle est souvent combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy.

Le module pyplot de matplotlib est l'un de ses principaux modules. Il regroupe un grand nombre de fonctions qui servent à créer des graphiques et les personnaliser (travailler sur les axes, le type de graphique, sa forme et même rajouter du texte).

Procédons par des exemples

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



Remarque

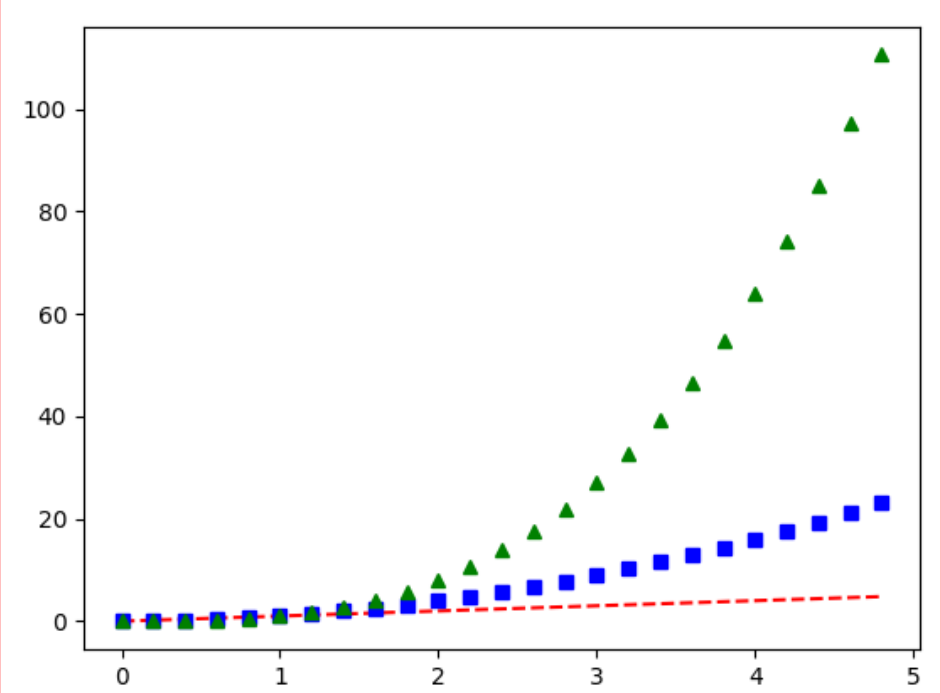
Vous vous demandez peut-être pourquoi l'axe des x va de 0 à 3 et l'axe des y de 1 à 4. Si vous fournissez une seule liste ou un seul tableau à tracer, matplotlib suppose qu'il s'agit d'une séquence de valeurs y et génère automatiquement les valeurs x pour vous. Étant donné que les plages python commencent par 0, le vecteur x par défaut a la même longueur que y mais commence par 0. Par conséquent, les données x sont [0, 1, 2, 3].

Formater le style de votre tracé

Pour chaque paire d'arguments x, y, il existe un troisième argument facultatif qui est la chaîne de format qui indique la couleur et le type de ligne du tracé.

Formater le style de votre tracé

```
import numpy as np
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r–', t, t**2, 'bs', t, t**3, 'g')
plt.show()
```

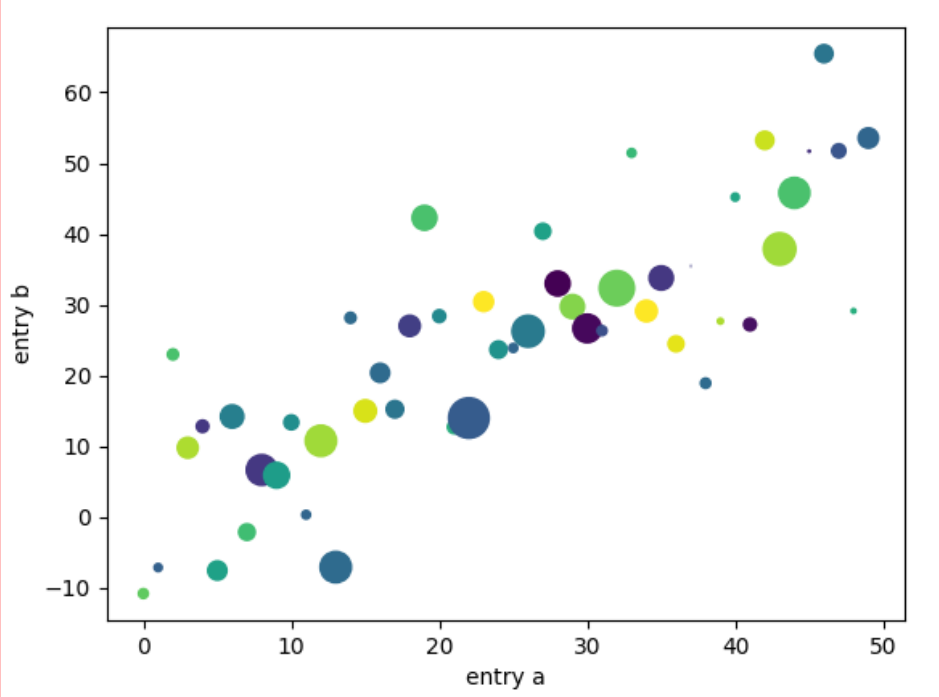


Tracé avec des chaînes de mots-clés

Tracé avec des chaînes de mots-clés

Dans certains cas, vous avez des données dans un format qui vous permet d'accéder à des variables particulières avec des chaînes. Par exemple, avec numpy.recarray ou pandas.DataFrame.
Matplotlib vous permet de fournir un tel objet avec l'argument du mot-clé data. Si elles sont fournies, vous pouvez générer des graphiques avec les chaînes correspondant à ces variables.

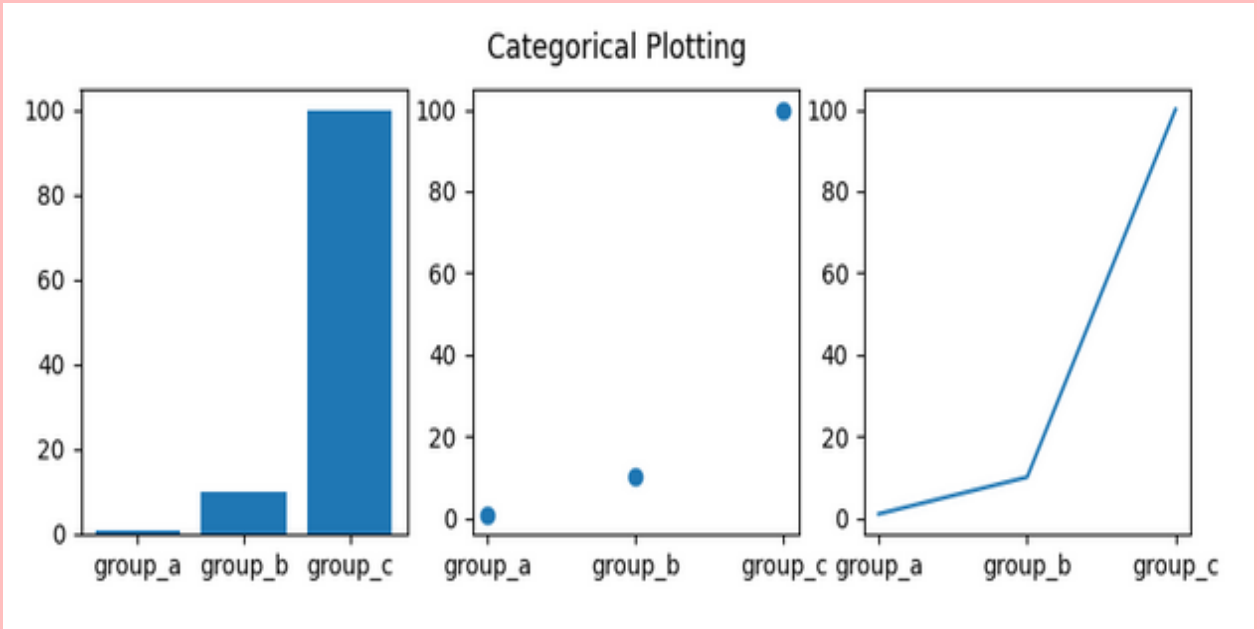
```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
```


entry b
entry a
60
50
40
30
20
10
0
−10
0
10
20
30
40
50

Traçage avec des variables catégorielles

Il est également possible de créer un tracé à l'aide de variables catégorielles. Matplotlib vous permet de transmettre des variables catégorielles directement à de nombreuses fonctions de traçage. Par exemple :

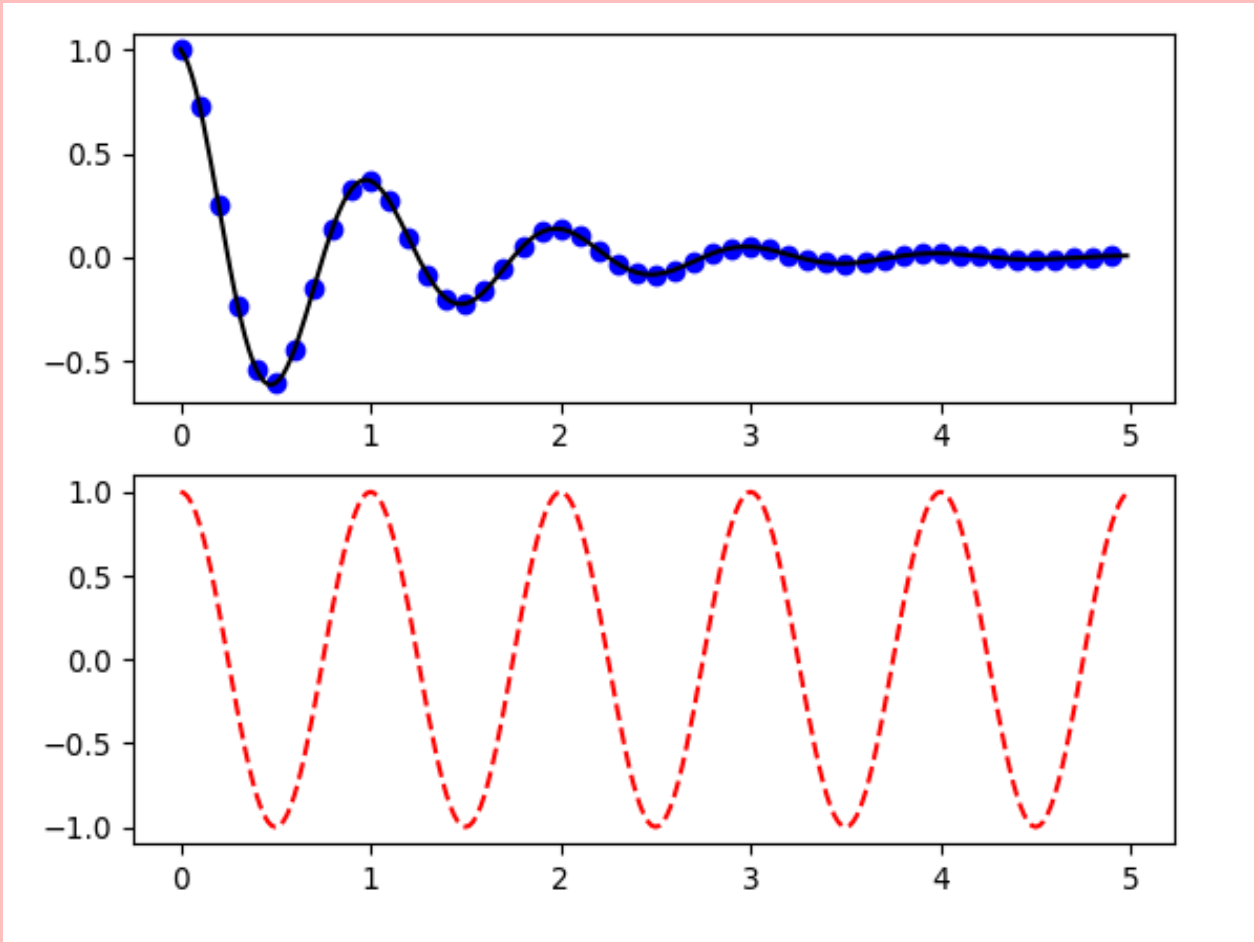
```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```


Categorical Plotting
100
80
60
40
20
0
group_a
group_b
group_c

Travailler avec plusieurs figures et axes

MATLAB et pyplot ont le concept de la figure actuelle et des axes actuels. Toutes les fonctions de traçage s'appliquent aux axes actuels. La fonction gca renvoie les axes actuels (une instance matplotlib.axes.Axes) et gcf renvoie la figure actuelle (une instance matplotlib.figure.Figure). Normalement, vous n'avez pas à vous inquiéter à ce sujet, car tout est réglé en coulisses. Vous trouverez ci-dessous un script pour créer deux sous-intrigues.

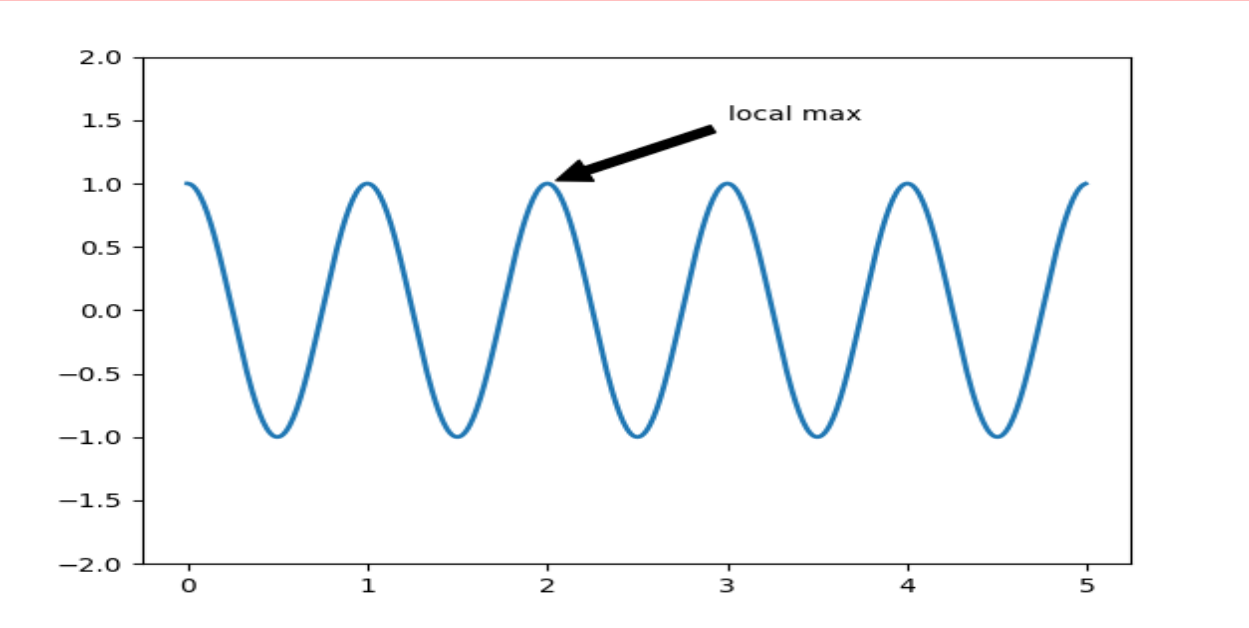
```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')
plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r–')
plt.show()
```


1.0
0.5
0.0
−0.5
0
1
2
3
4
5
1.0
0.5
0.0
−0.5
−1.0
0
1
2
3
4
5

Annoter du texte

Les utilisations de la fonction de texte de base ci-dessus placent le texte à une position arbitraire sur les axes. Une utilisation courante du texte consiste à annoter certaines caractéristiques du tracé, et la méthode annotate fournit une fonctionnalité d'assistance pour faciliter les annotations. Dans une annotation, il y a deux points à considérer : l'emplacement à annoter représenté par l'argument xy et l'emplacement du texte xytext. Ces deux arguments sont des tuples (x, y).

```
ax = plt.subplot()
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)
plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
            arrowprops=dict(facecolor='black', shrink=0.05),
            )
plt.ylim(-2, 2)
plt.show()
```



Dans cet exemple de base, les emplacements xy (pointe de la flèche) et xytext (emplacement du texte) sont tous deux en coordonnées de données. Il existe une variété d'autres systèmes de coordonnées que vous pouvez choisir – voir Annotation de base et Annotation avancée pour plus de détails. D'autres exemples peuvent être trouvés dans Annotation de tracés.

Axes logarithmiques et autres axes non linéaires

Matplotlib.pyplot prend en charge non seulement les échelles d'axes linéaires, mais également les échelles logarithmiques et logit. Ceci est couramment utilisé si les données couvrent plusieurs ordres de grandeur. Changer l'échelle d'un axe est simple :

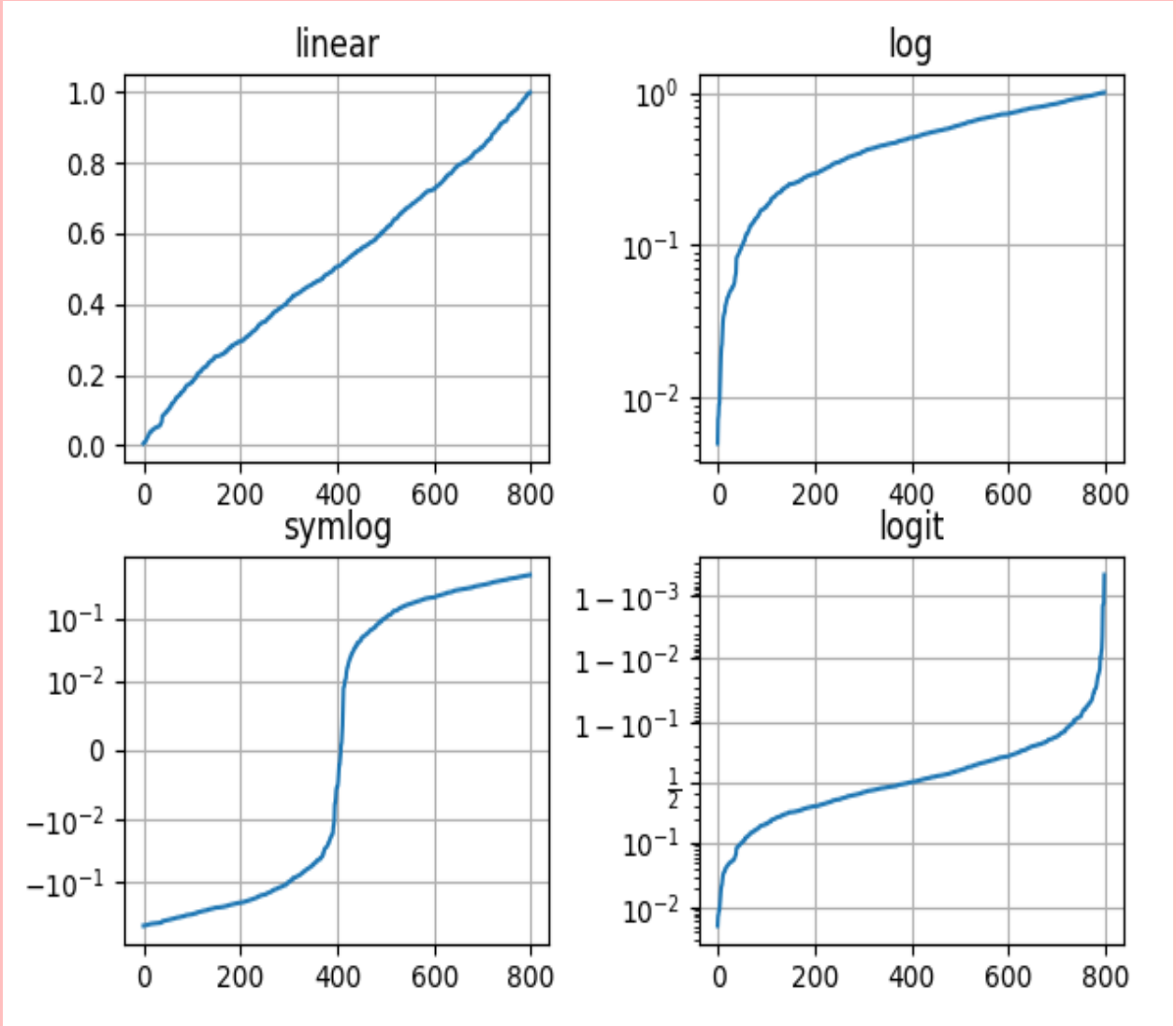
```
plt.xscale('log')
```

Un exemple de quatre tracés avec les mêmes données et des échelles différentes pour l'axe des y est présenté ci-dessous.

```
# Fixing random state for reproducibility
np.random.seed(19680801)

# make up some data in the open interval (0, 1)
y = np.random.normal(loc = 0.5, scale = 0.4, size = 1000)
y = y[(y > 0)(y < 1)] Fixing random state for reproducibility
np.random.seed(19680801)
make up some data in the open interval (0, 1) y =
np.random.normal(loc=0.5, scale=0.4, size=1000) y = y[(y ¿ 0)  (y ¡ 1)]
y.sort() x = np.arange(len(y))
plot with various axes scales plt.figure()
y.sort()
x = np.arange(len(y))

# plot with various axes scales
plt.figure()
```

```
# linear
plt.subplot(221)
plt.plot(x, y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)

# log
plt.subplot(222)
plt.plot(x, y)
plt.yscale('log')
plt.title('log')
plt.grid(True)
```

```
# symmetric log
plt.subplot(223)
plt.plot(x, y - y.mean())
plt.yscale('symlog', linthresh=0.01)
plt.title('symlog')
plt.grid(True)

# logit
plt.subplot(224)
plt.plot(x, y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)

# Adjust the subplot layout, because the logit one may take more space
# than usual, due to y-tick labels like "1 - 10^{-3}"

plt.subplots_adjust(top=0.92, bottom=0.08, left=0.10, right=0.95, hspace=0.25,
                    wspace=0.35)
plt.show()
```


linear
log
symlog
logit
1.0
0.8
0.6
0.4
0.2
0.0
10^0
10^{-1}
10^{-2}
-10^{-1}
-10^{-2}
0
$1-10^{-1}$
$1-10^{-2}$
$1-10^{-3}$
$\frac{1}{2}$
0
200
400
600
800

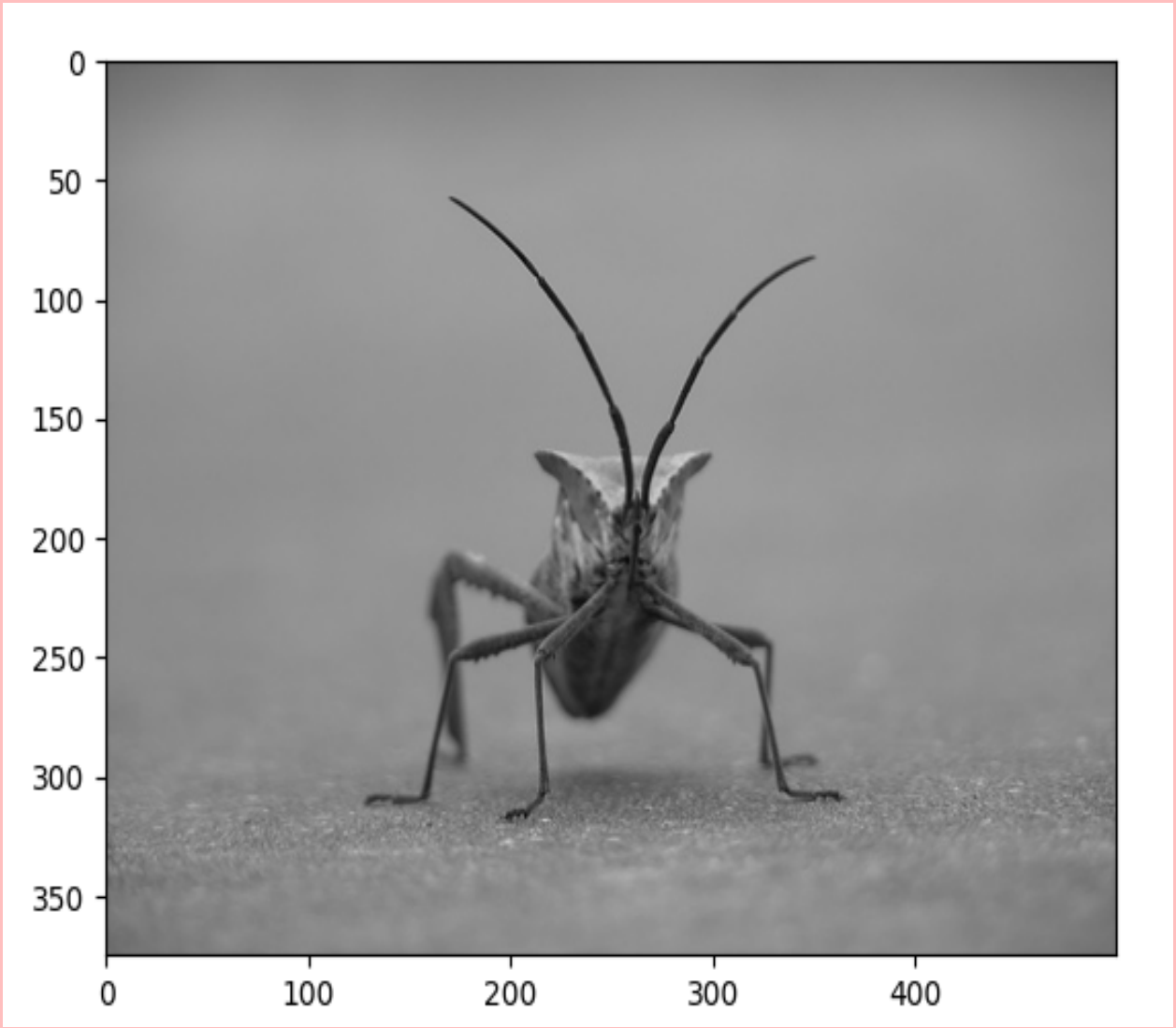
Importer des données d'image dans des tableaux Numpy

Matplotlib s'appuie sur la bibliothèque Pillow pour charger les données d'image.
Voici l'image avec laquelle nous allons jouer :

```
from PIL import Image

img = np.asarray(Image.open('/home/apeke/Documents/Cours_Kara - 2024/stinkbug.png'))
print(repr(img))
```

```
array([[[104, 104, 104],
        [104, 104, 104],
        [104, 104, 104],
        ...,
        [109, 109, 109],
        [109, 109, 109],
        [109, 109, 109]],

       [[105, 105, 105],
        [105, 105, 105],
        [105, 105, 105],
        ...,
        [109, 109, 109],
        [109, 109, 109],
        [109, 109, 109]],

       [[107, 107, 107],
        [106, 106, 106],
        [106, 106, 106],
        ...,
        [110, 110, 110],
        [110, 110, 110],
        [110, 110, 110]],
```

imgplot = plt.imshow(img)

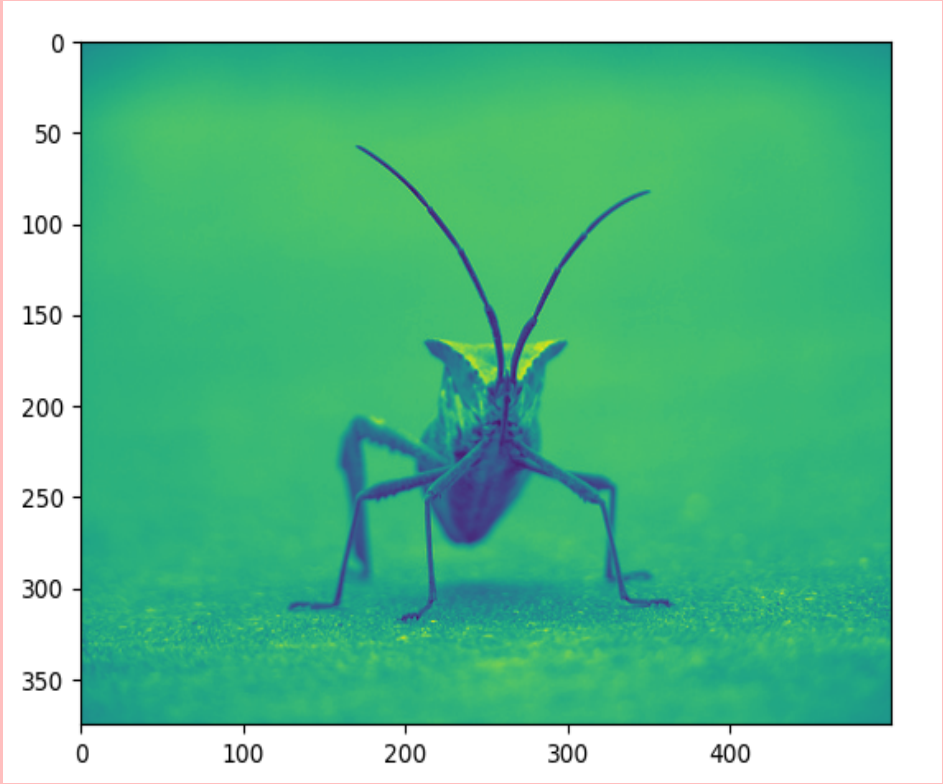


Application de schémas de pseudo-couleurs aux tracés d'images

Pseudocolor peut être un outil utile pour améliorer le contraste et visualiser plus facilement vos données. Ceci est particulièrement utile lorsque vous effectuez des présentations de vos données à l'aide de projecteurs – leur contraste est généralement assez faible.

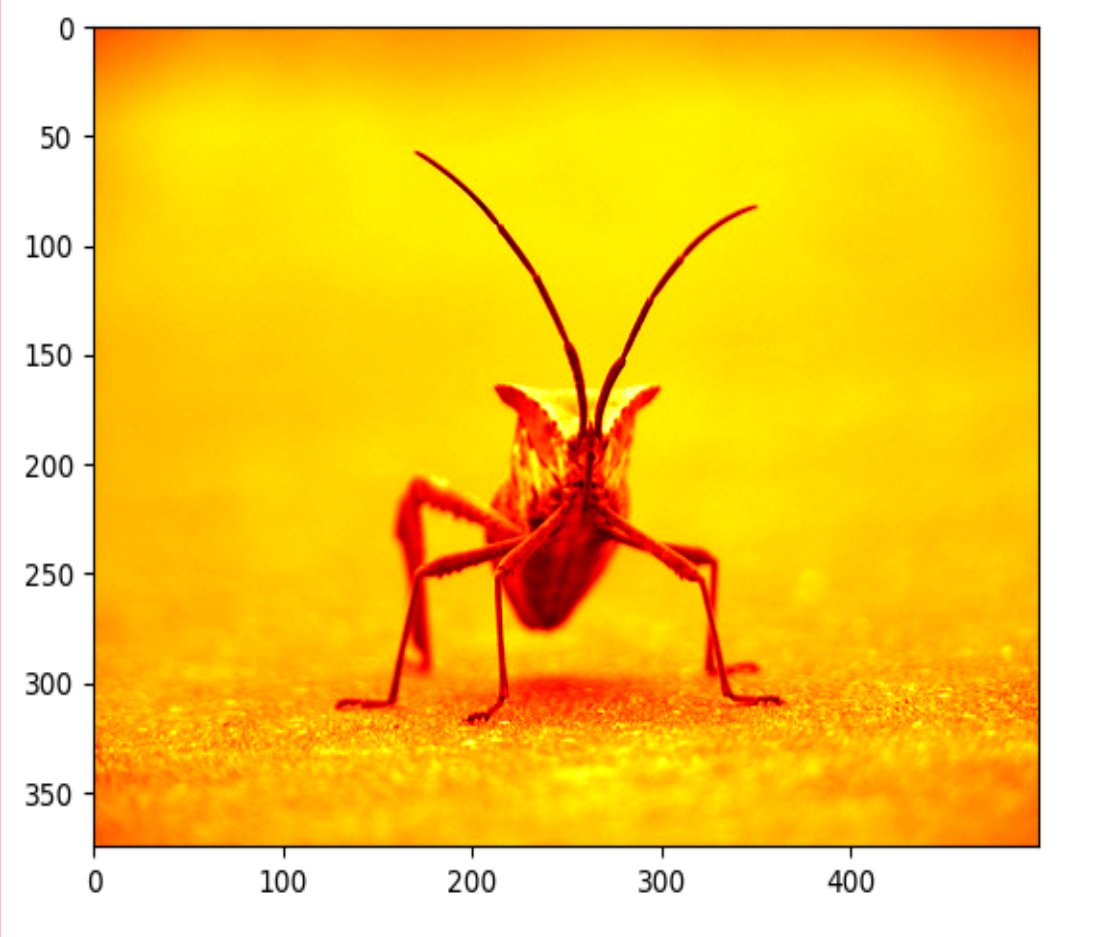
La pseudo-couleur ne concerne que les images de luminosité monocanal, en niveaux de gris. Nous avons actuellement une image RVB. Puisque R, G et B sont tous similaires (voyez par vous-même ci-dessus ou dans vos données), nous pouvons simplement choisir un canal de nos données en utilisant le découpage de tableau (vous pouvez en savoir plus dans le didacticiel Numpy) :

```
lum_img = img[:, :, 0]
plt.imshow(lum_img)
```



Désormais, avec une image de luminosité (2D, sans couleur), la palette de couleurs par défaut (alias table de recherche, LUT) est appliquée. La valeur par défaut est appelée viridis. Il y en a bien d'autres parmi lesquels choisir.

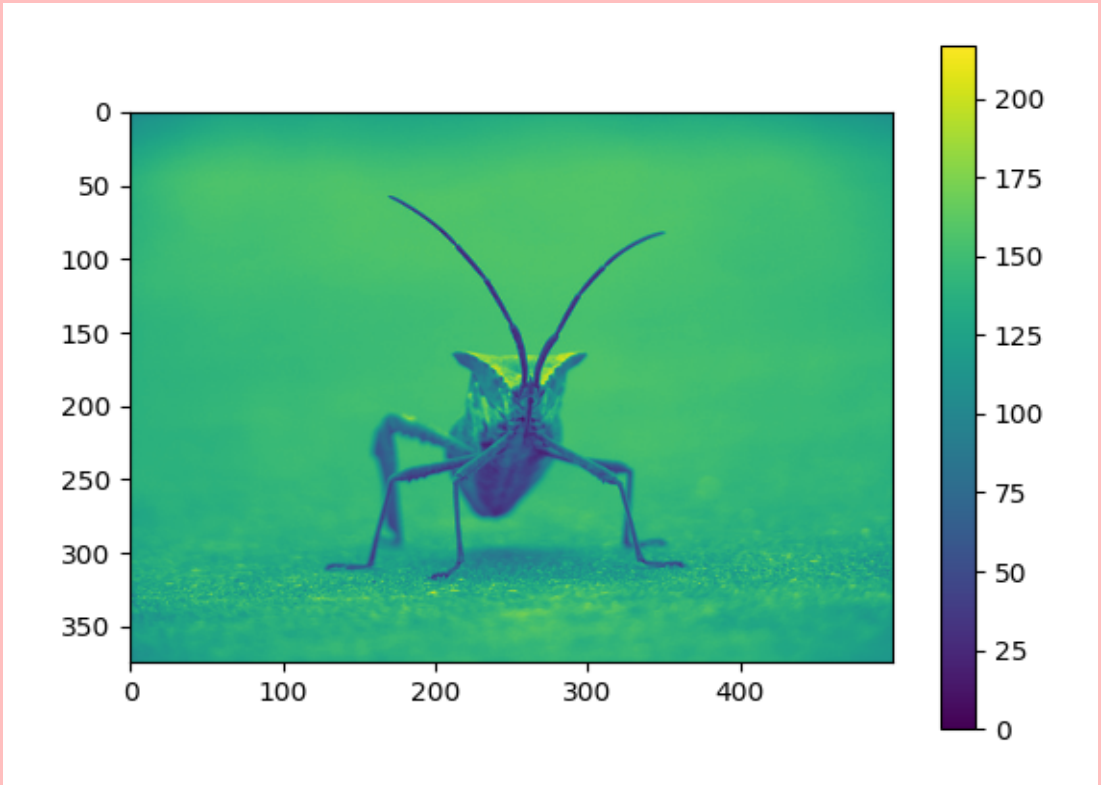
plt.imshow(lum_img, cmap="hot")



Référence de l'échelle de couleurs

Il est utile d'avoir une idée de la valeur que représente une couleur. Nous pouvons le faire en ajoutant une barre de couleur à votre silhouette :

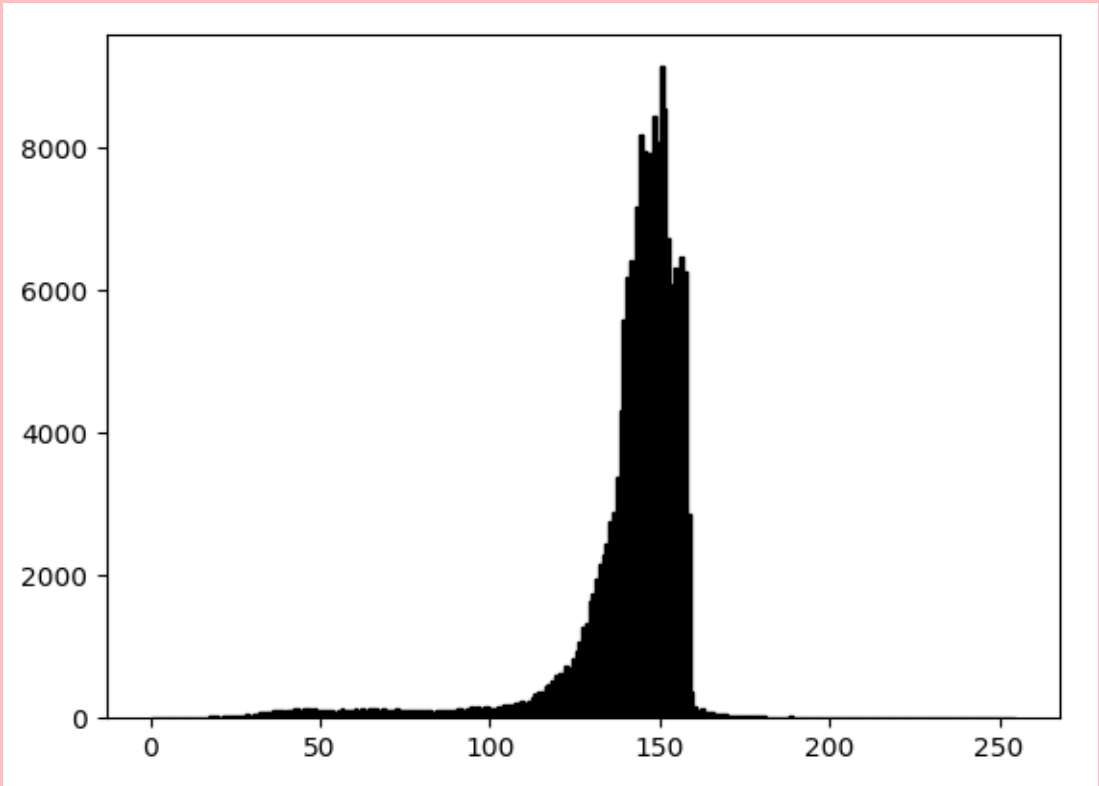
imgplot = plt.imshow(lum_img)
plt.colorbar()



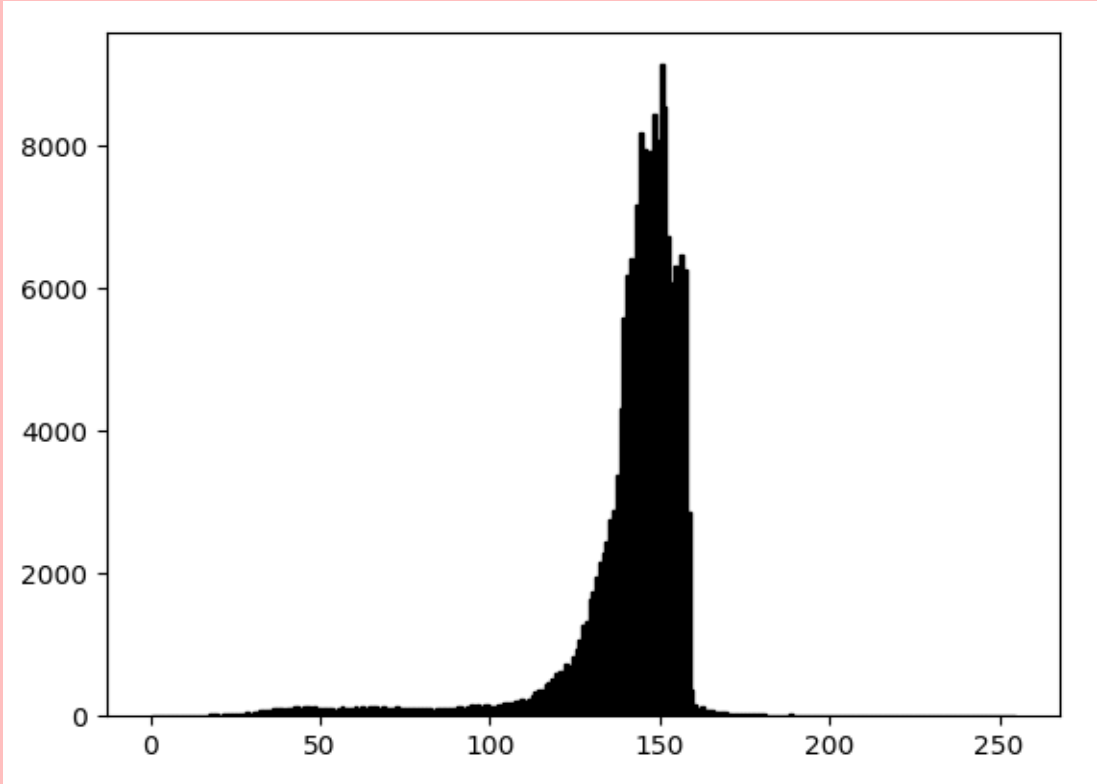
Examen d'une plage de données spécifique

Parfois, vous souhaitez améliorer le contraste de votre image ou élargir le contraste dans une région particulière tout en sacrifiant les détails dans des couleurs qui ne varient pas beaucoup ou n'ont pas d'importance.
L'histogramme est un bon outil pour trouver des régions intéressantes.
Pour créer un histogramme de nos données d'image, nous utilisons la fonction hist().

plt.hist(lum_img.ravel(), bins=range(256), fc='k', ec='k')



plt.hist(lum_img.ravel(), bins=range(256), fc='k', ec='k')

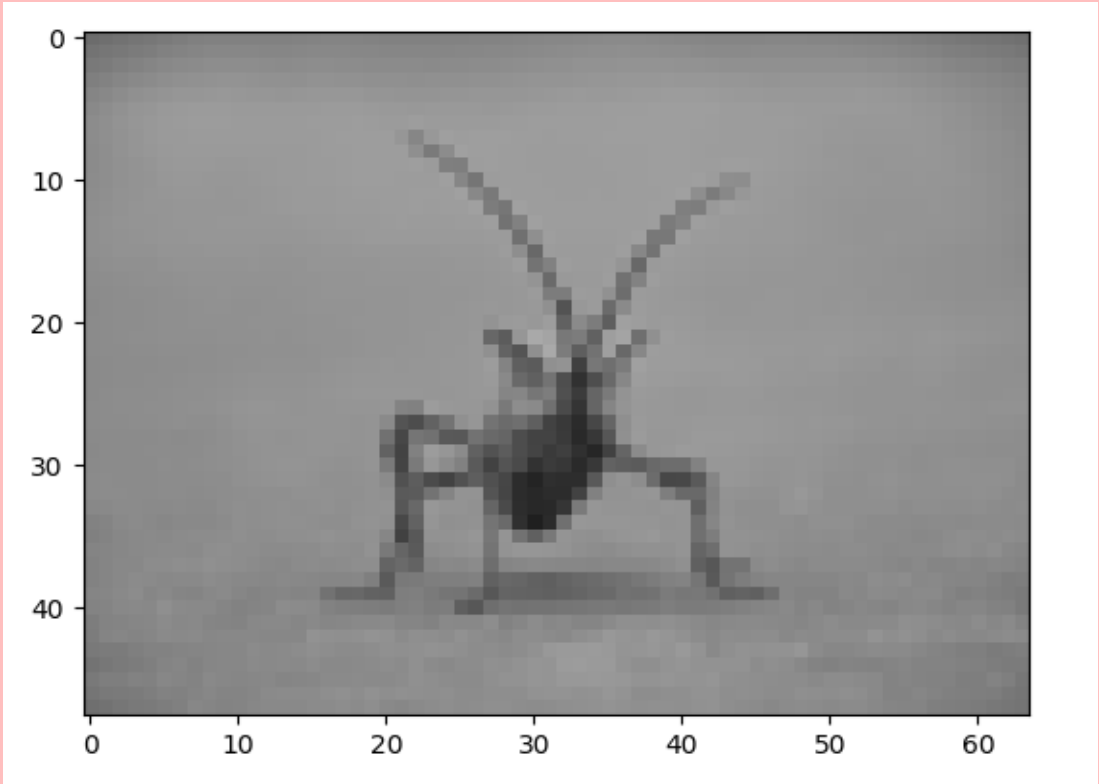


Most often, the "interesting" part of the image is around the peak, and you can get extra contrast by clipping the regions above and/or below the peak. In our histogram, it looks like there's not much useful information in the high end (not many white things in the image). Let's adjust the upper limit, so that we effectively "zoom in on" part of the histogram. We do this by setting clim, the colormap limits.

Schémas d'interpolation de tableaux

L'interpolation calcule ce que "devrait" être la couleur ou la valeur d'un pixel, selon différents schémas mathématiques. Cela se produit souvent lorsque vous redimensionnez une image. Le nombre de pixels change, mais vous voulez les mêmes informations. Puisque les pixels sont discrets, il manque de l'espace. L'interpolation est la façon dont vous remplissez cet espace. C'est pourquoi vos images semblent parfois pixellisées lorsque vous les agrandissez. L'effet est d'autant plus prononcé que la différence entre l'image originale et l'image agrandie est plus grande. Prenons notre image et réduisons-la. Nous supprimons effectivement les pixels, n'en conservant que quelques-uns. Désormais, lorsque nous les traçons, ces données sont agrandies à la taille de votre écran. Les anciens pixels ne sont plus là et l'ordinateur doit dessiner en pixels pour remplir cet espace.
Nous utiliserons également la bibliothèque Pillow que nous avons utilisée pour charger l'image pour redimensionner l'image.

```
img = Image.open('../../doc/_static/stinkbug.png')
img.thumbnail((64, 64))  resizes image in-place
imgplot = plt.imshow(img)
```



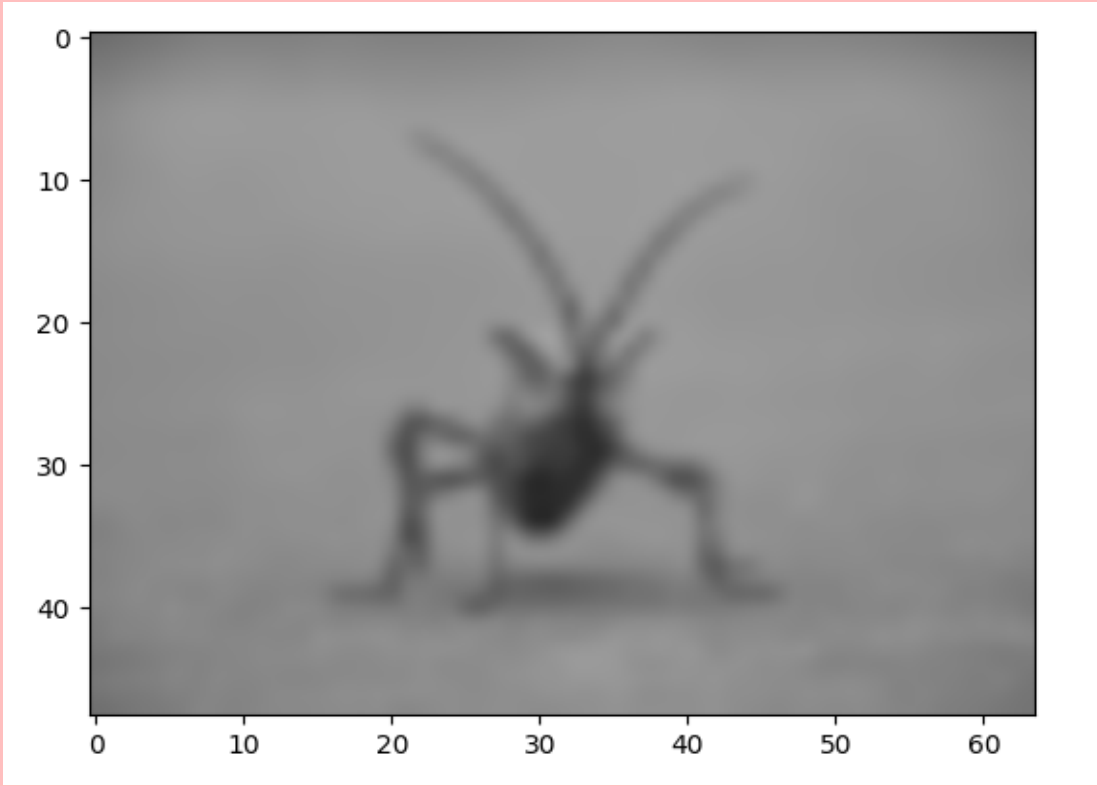
Ici, nous utilisons l'interpolation par défaut ("la plus proche"), puisque nous n'avons donné à imshow() aucun argument d'interpolation.

Essayons-en d'autres. Voici "bilinéaire" :

imgplot = plt.imshow(img, interpolation="bilinear")

et bicubique :

imgplot = plt.imshow(img, interpolation="bicubic")



L'interpolation bicubique est souvent utilisée pour agrandir des photosBicubic interpolation is often used when blowing up photos - people tend to prefer blurry over pixelated..