Visualisation avec Python

Initiation à Matplotlib

Dominique Benielli[1] Eulalio Torres Garcia[3] Denis Arrivault[1 2]
François-Xavier Dupé[2]

[1]Institut Archimède
Aix Marseille Université

[2]Laboratoire Informatique et Systèmes

[3]Cedre Aix Marseille Université

Formation Python Scientifique
@AMU

Introduction

La communauté Python propose un grand nombre de modules pour faire des graphiques ou de la visualisations, dont

1. `matplotlib` (le plus utilisé),
2. `bokeh` qui propose une interface différente,
3. `plot.ly` un concurrent du précédent.

C'est `matplotlib` que l'on va étudier aujoud'hui.

Outline

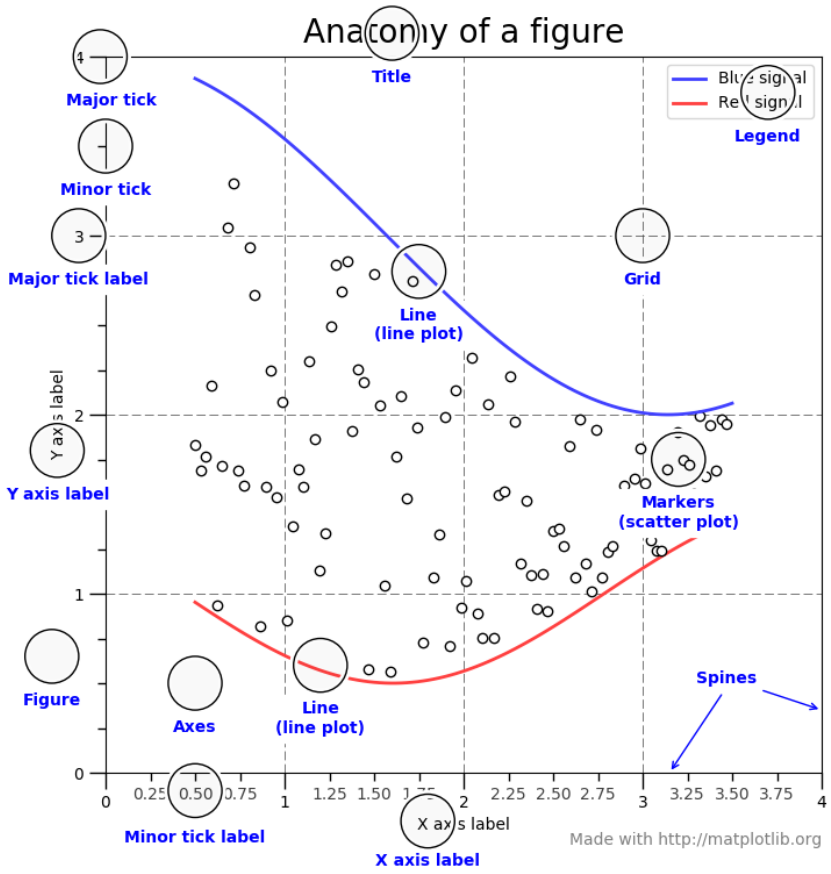
Introduction

Matplotlib est un module de visualisation scientifique créé par John Hunter en 2007. Il permet notamment de dessiner,

- les contours 2D de fonctions,
- les histogrammes et autres,
- les spectrogrammes,
- des animations,
- ...
- de mettre des légendes.

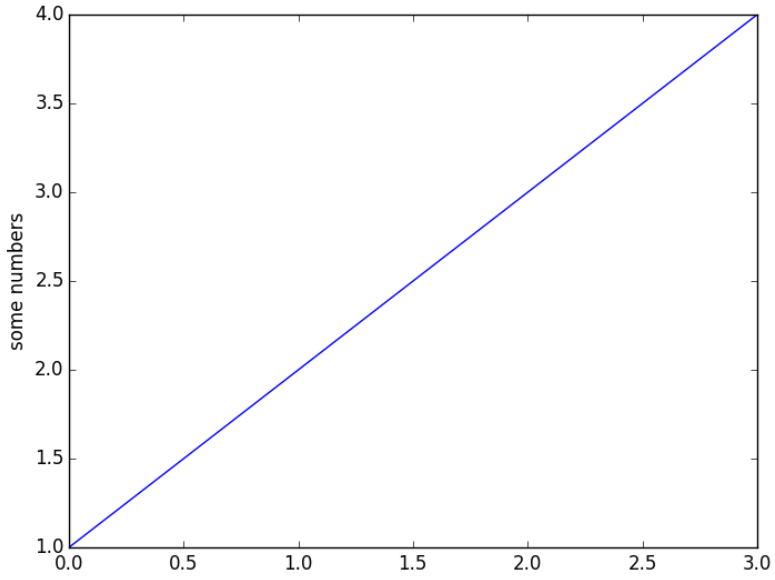
Anatomie d'une figure Pyplot

- Title
- line plot
- Legend
- Axes
- Grid
- …



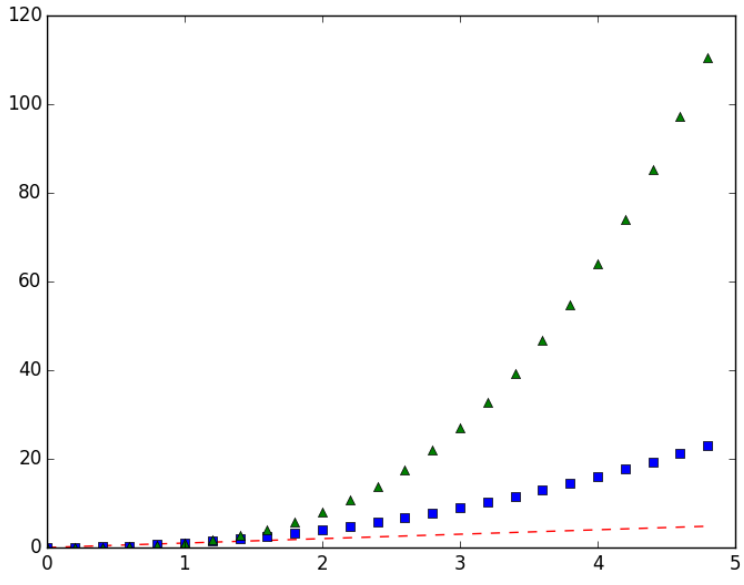
Une rapide introduction

```
# Import du module
import matplotlib.pyplot as plt
# Une courbe
plt.plot([1,2,3,4])
# Un label pour l'axe y
plt.ylabel('des nombres')
# On montre le tout
plt.show()
```



Un exemple plus complexe

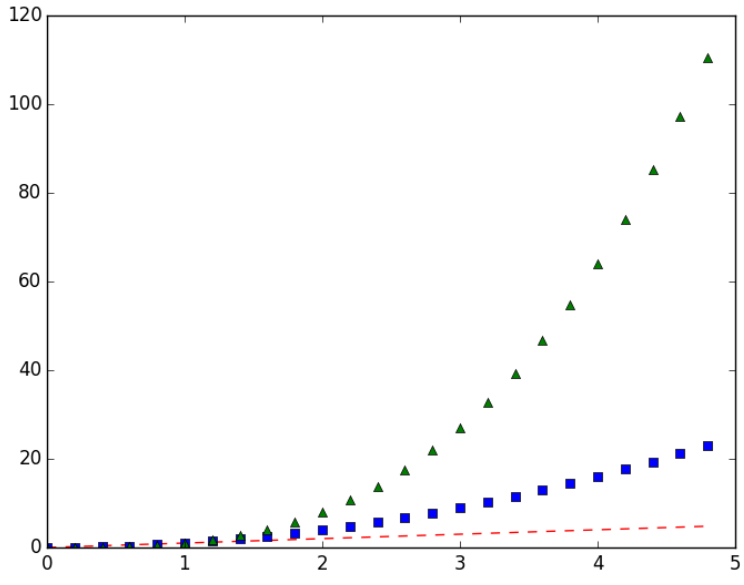
```
import numpy as np
import matplotlib.pyplot as plt

# Des données
t = np.arange(0., 5., 0.2)

# Des courbes différentes
plt.plot(t, t, 'r--', t, t**2,
    'bs', t, t**3, 'g^')
plt.show()
```



Les paramètres sont proches de Matlab.

Un exemple plus complexe

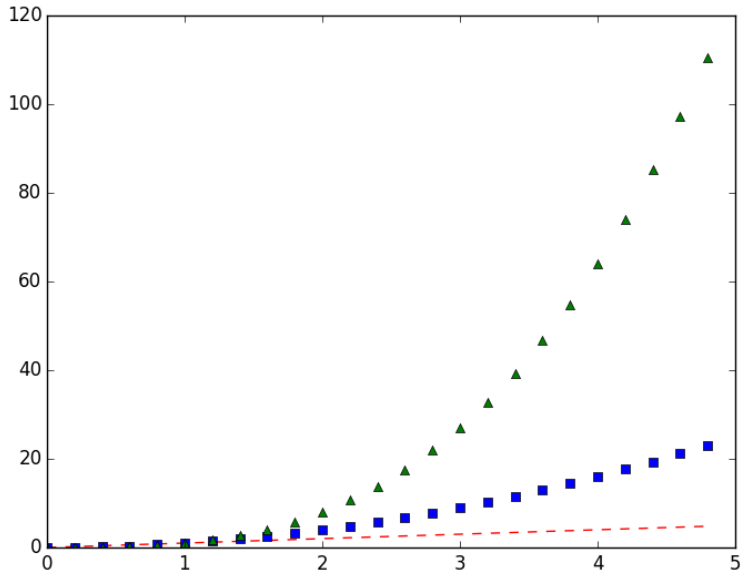
```
import numpy as np
import matplotlib.pyplot as plt

# Des données
t = np.arange(0., 5., 0.2)

# Des courbes différentes
plt.plot(t, t, 'r--', t, t**2,
    'bs', t, t**3, 'g^')
plt.show()
```



Les paramètres sont proches de Matlab.

Un exemple plus complexe

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

# Des données
t = np.arange(0., 5., 0.2)

# Des courbes différentes
plt.plot(t, t, 'r--', t, t**2,
    'bs', t, t**3, 'g^')
plt.show()
```



Les paramètres sont proches de Matlab.

Remarque

A partir de maintenant, nous supposons que les importations suivantes sont faites,

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

Outline

Le module *pyplot*

Ce module propose un grand nombre de fonctions pour tracer des graphiques. Entre autre, nous y trouvons,

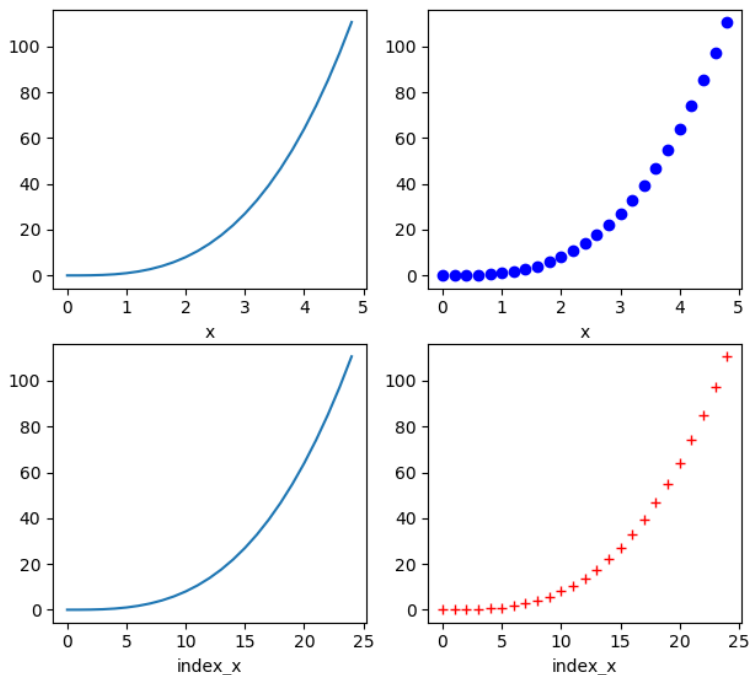
- `figure` pour créer des figures (comme dans Matlab),
- `axes` region de l'image avec les données.
- `plot` pour les tracers classiques,
- `bar` pour les graphes en barres,
- `contour` pour tracer les contours de fonctions,
- `errorbar` pour afficher avec les barres d'erreurs,
- `hist` pour les histogrammes,
- `acorr` pour tracer les autocorrélations,
- `annotate` pour placer une annotation,
- `axes` pour gérer les axes,
- `grid` pour la grille,
- `legend` pour les légendes,
- `semilogx`, `subplot`...

Tracer des lignes

Les lignes sont tracées via la fonction `plot`. Cette fonction a un comportement similaire à celle de Matlab.

Exemple d'invocations :

```
plot(x, y)         # dessine x et
    y avec le style par défaut
plot(x, y, 'bo') # dessine x et
    y avec des cercles bleus
plot(y)            # dessine y en
    utilisant le tableau 0..N
    -1 pour x
plot(y, 'r+')      # idem mais
    avec des plus rouges
```



Tracer des lignes (2)

- avec les mots-clefs :

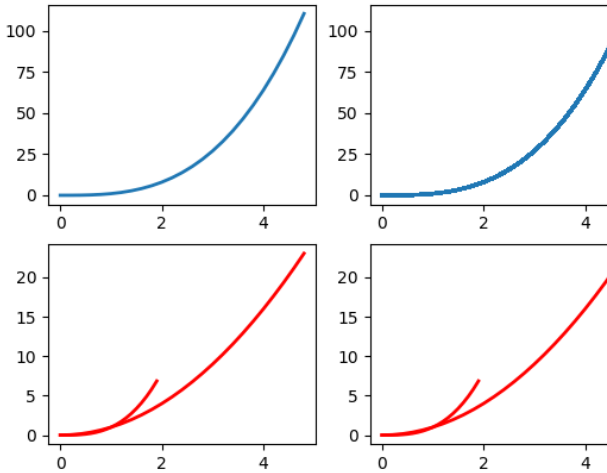
```
plt.plot(x, y, linewidth=2.0)
```

- en utilisant les accesseurs de `Line2D`,

```
line, = plt.plot(x, y, '-',
    linewidth=2.0)
line.set_antialiased(False) #
    supprime l'antialising
```

- en utilisant la fonction `setp`,

```
lines = plt.plot(x1, y1, x2, y2)
# Par mots-clefs
plt.setp(lines, color='r',
    linewidth=2.0)
# En utilisant un style Matlab
plt.setp(lines, 'color', 'r', '
    linewidth', 2.0)
```

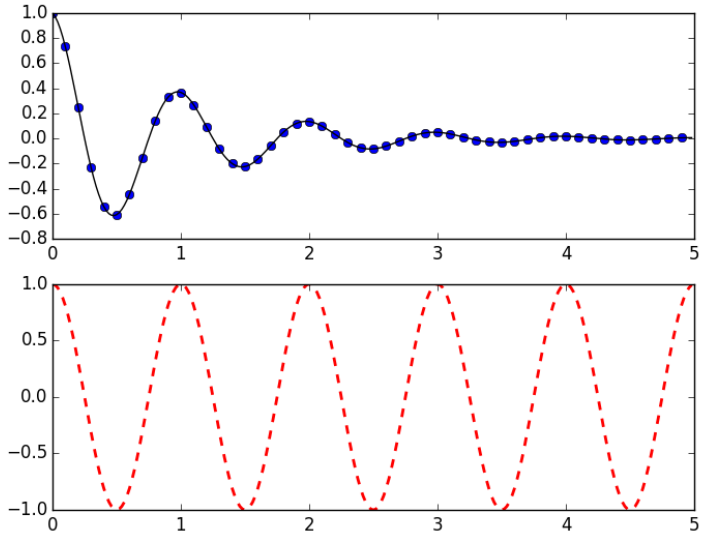


Sélection de paramètres

- `alpha` pour la transparence,
- `label` une légende,
- `axes` les options sur les axes,
- `antialiasing` pour mettre ou non l'antialiasing,
- `linestyle` le style de la ligne,
- `linewidth` la largeur de la ligne,
- `marker` le type de marqueur,
- `xdata` les données en x (tableau 1D),
- `ydata` les données en y (tableau 1D),
- . . . (voir la documentation)

Exemple plus complexe avec Axes.plot

```
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, (ax1, ax2) = plt.subplots(2, 1)
ax1.plot(t1, f(t1), 'bo', t2, f(t2),
    'k')

ax2.plot(t2, np.cos(2*np.pi*t2), 'r
    --', linewidth=2.0)
plt.show()
```



Les types de lignes

Matplotlib propose plusieurs type de ligne (changeable avec le paramètre `linestyle`),

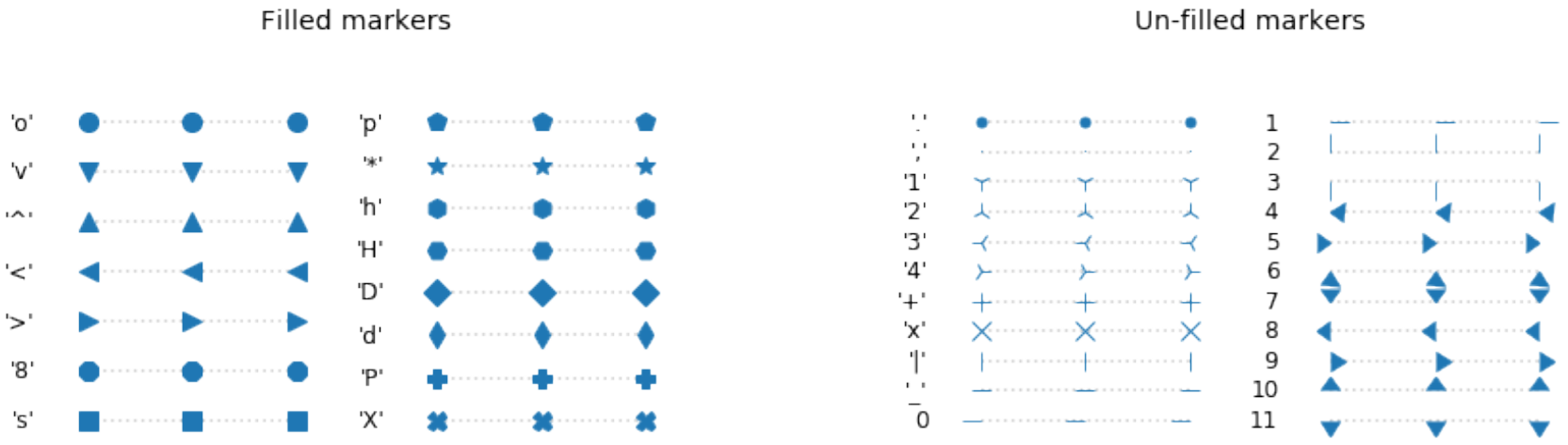
- `-` pour la ligne pleine,
- `--` pour la ligne en tirets,
- `-.` pour la ligne en tirets et pointillés,
- `:` pour la ligne en pointillés,
- `None` ou `' '` ou `''` pour ne rien afficher.

Les types de marques

De même plusieurs type de marques sont disponibles,

- `.` un point,
- `,` un pixel,
- `o` un cercle,
- `v` un triangle bas,
- `^` un triangle haut,
- `<` un triangle à gauche,
- `>` un triangle à droite,
- `8` un octogone,
- `s` un carré,
- `p` un pentagone,
- `*` une étoile,
- `h` ou `H` un hexagone,
- `+` un plus,
- `x` une croix,
- `D` un diamant,
- `None` rien,
- ...

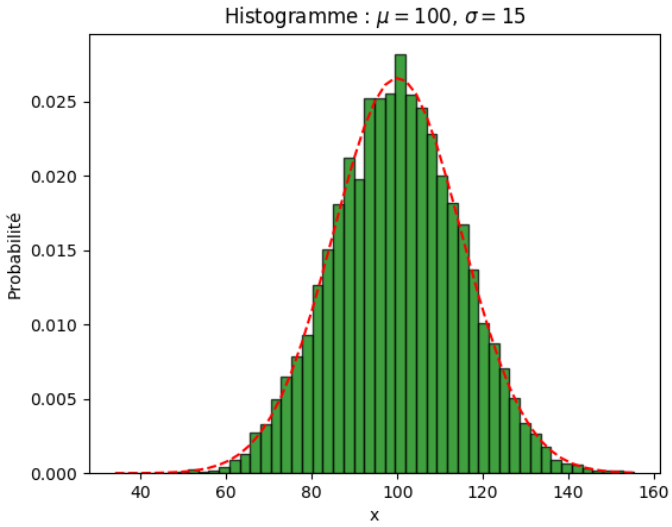
Les types de marques



Outline

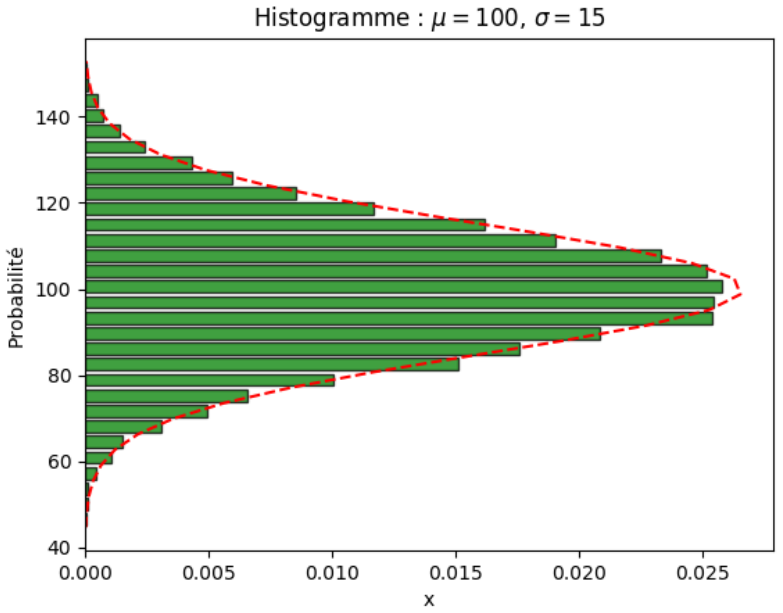
Histogramme

```
mu = 100 # la moyenne
sigma = 15 # ecart-type
x = mu + sigma * np.random.randn
    (10000)
# Histogramme
num_bins = 50
n, bins, patches = plt.hist(x,
    num_bins, normed=1, facecolor='
    green', histtype='bar' , alpha=0
    .5)
plt.setp(patches, 'alpha', 0.75, '
    edgecolor' , 'k' )
# Ajout d'une ligne "best fit"
y = mlab.normpdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')
plt.xlabel('x')
plt.ylabel('Probabilité')
plt.title(r'Histogramme  : $\mu=100
    $  $\sigma=15$')
```



Histogramme(2)

```
n, bins, patches = plt.hist(x,
    num_bins, normed=1, facecolor=
    'g', rwidth  =0.8, orientation
    ='horizontal',histtype='bar' ,
     alpha=0.75, edgecolor= 'k')
```

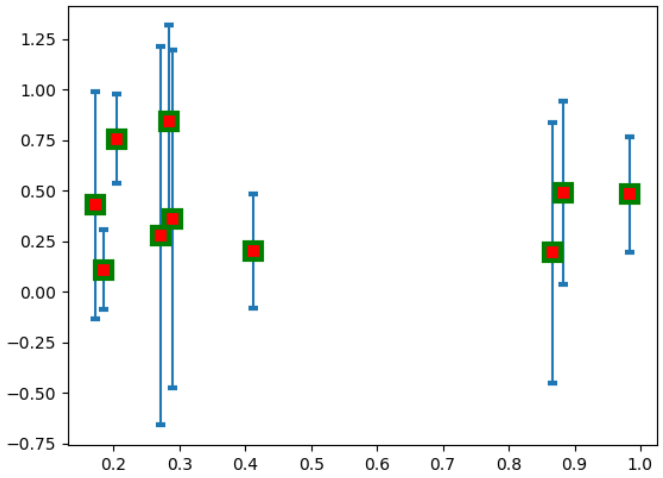


Tracer avec des barres d'erreurs

```
x,y,yerr = scipy.rand(3,10)
plt.errorbar(x, y, yerr, marker='s
    ', mfc='red', mec='green', ms
    =10, mew=3, linestyle='',
    capsize=3)
```

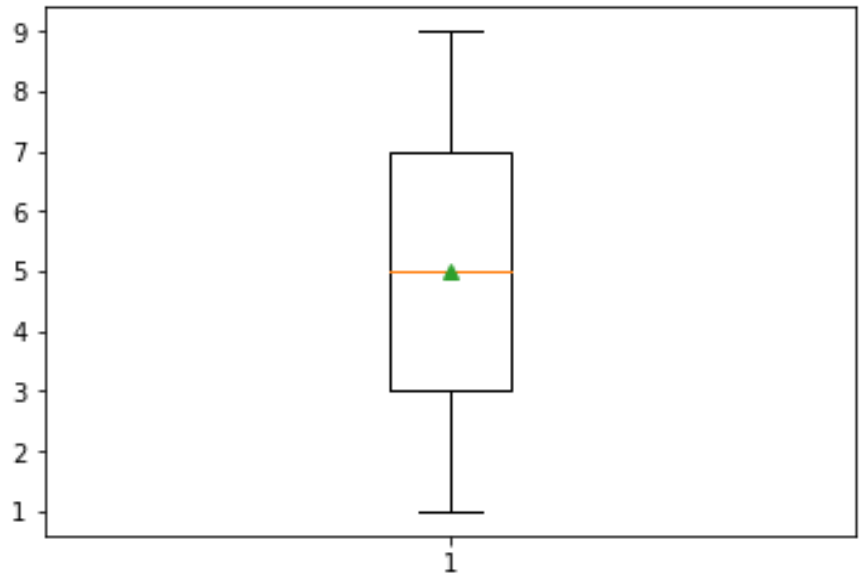
avec,

- `mfc` pour la couleur des marques,
- `mec` pour la couleur du bord des marques,
- `ms` pour la taille des marques,
- `mew` pour la largeur des bords.
- `capsize` taille des terminaisons des barres erreur



Boites à moustaches

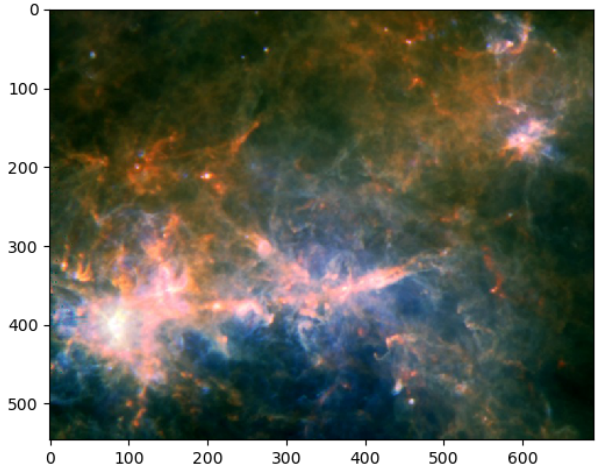
```
data = [1,2,3,4,5,6,7,8,9]
plt.boxplot(data, showmeans=
    True)
```

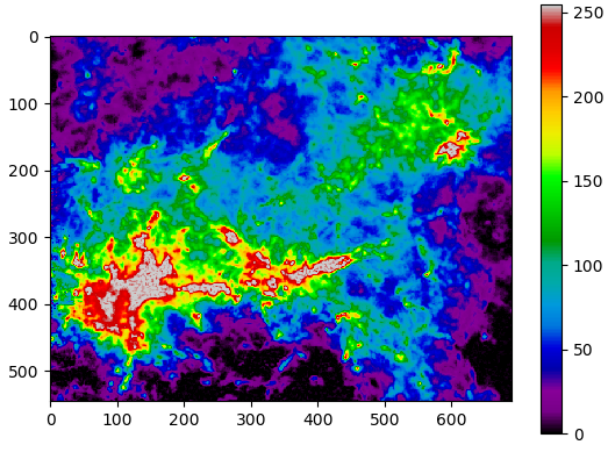


Outline

Visualisation d'Images

```
import matplotlib.image as mpimg
img=mpimg.imread('../images/
    g49.jpg')
imgplot = plt.imshow(img)

lum_img = img[:,:,0]
imgplot = plt.imshow(lum_img)
imgplot.set_cmap('nipy_spectral')
plt.colorbar()
```





Cartographie Basemap (1)

```
from mpl_toolkits.basemap
    import Basemap
import matplotlib.pyplot as
    plt
m = Basemap(width=12000000
    ,height=9000000
    ,projection='lcc',
            resolution='c'
                ,lat_1=44
                ,lat_2=52
                ,lat_0=48
                ,lon_0=2.34)
m.bluemarble()
```

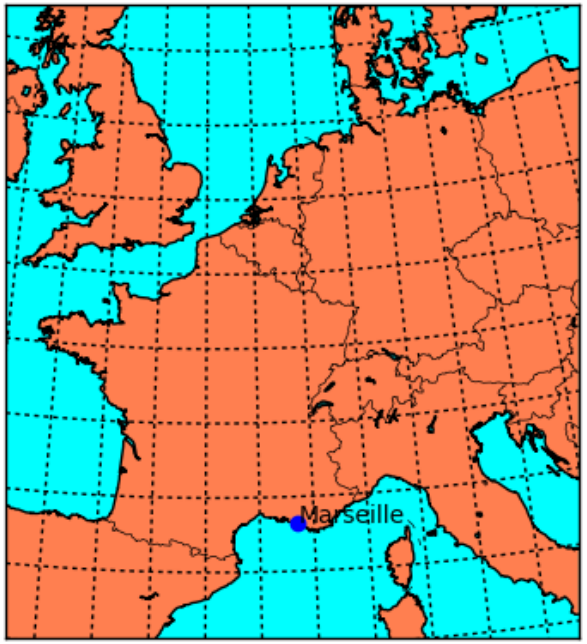
Cartographie Basemap (2)

```
from mpl_toolkits.basemap
    import Basemap
m = Basemap(llcrnrlon=-5
    ,llcrnrlat=40,urcrnrlon=20
    ,urcrnrlat=56, resolution='
    i',projection='cass',lon_0
    =2.34,lat_0=48)
m.drawcoastlines()
m.drawcountries()
m.fillcontinents(color='coral',
     lake_color='aqua')
m.drawparallels(np.arange(-40
    ,61.,2.))
m.drawmeridians(np.arange(-20
    .,21.,2.))
m.drawmapboundary(fill_color='
    aqua')
```

```
lon =  5.400000
lat = 43.300000
x,y = m(lon, lat)
m.plot(x, y, 'bo',
    markersize=6)   plt.text
    (x, y, "Marseille")
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(
    LinearLocator(10))
ax.zaxis.set_major_formatter
    (FormatStrFormatter('%
    .02f'))
fig.colorbar(surf, shrink=0
    .5, aspect=5)
plt.show()
```

Cartographie Basemap (3)

On obtient



Annotations

Avec `pyplot.annotate`, nous pouvons ajouter des annotations dans les figures. Cette fonction possède les arguments suivants,

- `s` un label ;
- `xy` les coordonnées de l'élément à annoter ;
- `xytext` la position du label (facultatif) ;
- `xycoords` le type de coordonnées utilisées pour le positionnement de l'élément à annoter (par exemple `data`, `figure points`, `figure pixels`, `axes points`) ;
- `textcoords` le type de coordonnées utilisées pour le positionnement du texte ;
- `arrowprops` un dictionnaire contenant le type de flèche à utiliser (ceci est un objet particulier, par défaut une simple ligne) et ses propriétés.

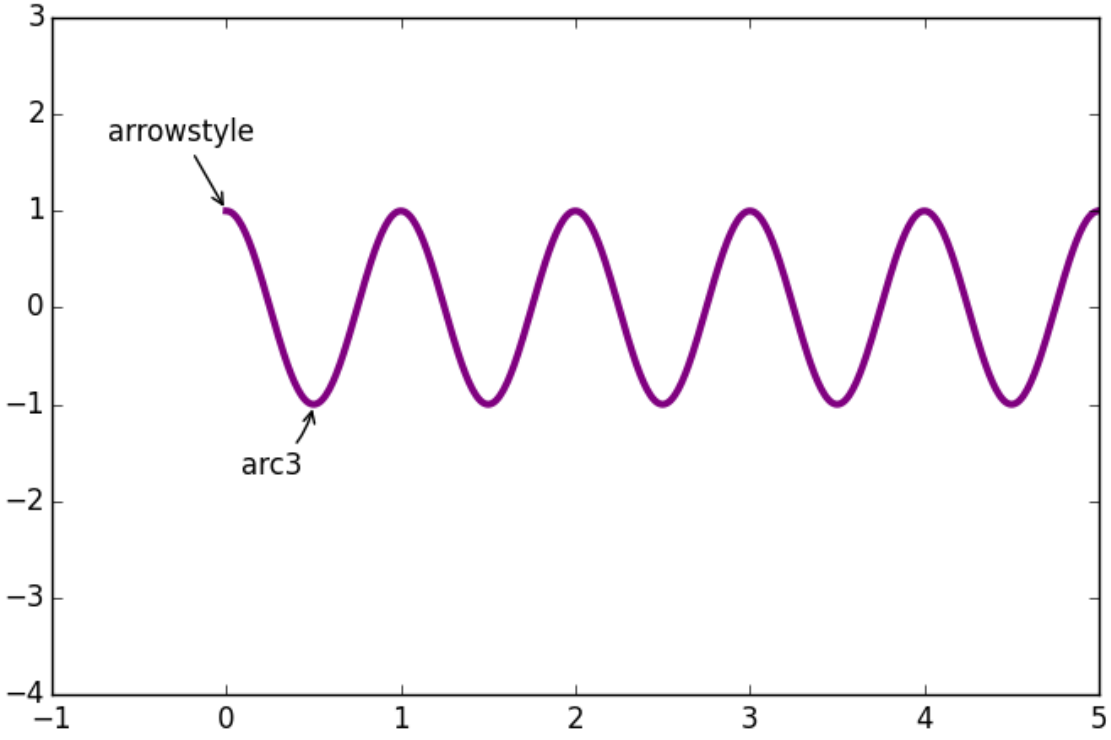
Note : si le dictionnaire pour `arrowprops` contient une clef `arrowstyle` alors une simple flèche est créée.

Annotations (2)

```
from matplotlib.pyplot import figure, show
fig = figure(1,figsize=(8,5))
ax = fig.add_subplot(111, autoscale_on=False, xlim=(-1,5
    ), ylim=(-4,3))
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = ax.plot(t, s, lw=3, color='purple')

ax.annotate('arrowstyle', xy=(0, 1),  xycoords='data',
             xytext=(-50, 30), textcoords='offset points
                 ',
             arrowprops=dict(arrowstyle="->"))
ax.annotate('arc3', xy=(0.5, -1),  xycoords='data',
            xytext=(-30, -30), textcoords='offset points
                ',
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="arc3,rad=.2
                                "))
show()
```

Annotations (3)

Nous obtenons,



Outline

Sous-figures avec pyplot

`pyplot.subplot` permet de faire des sous-figures dans une même figure. La signature de cette fonction est la suivante,

subplot(*nrows*, *ncols*, *plot_number*)

- *nrows* est le nombre de rangs dans la grille ;
- *ncols* est le nombre de colonne ;
- *plot_number* est le numéro de la figure dans la grille (le numéro commence à 1, puis parcourt les rangs).

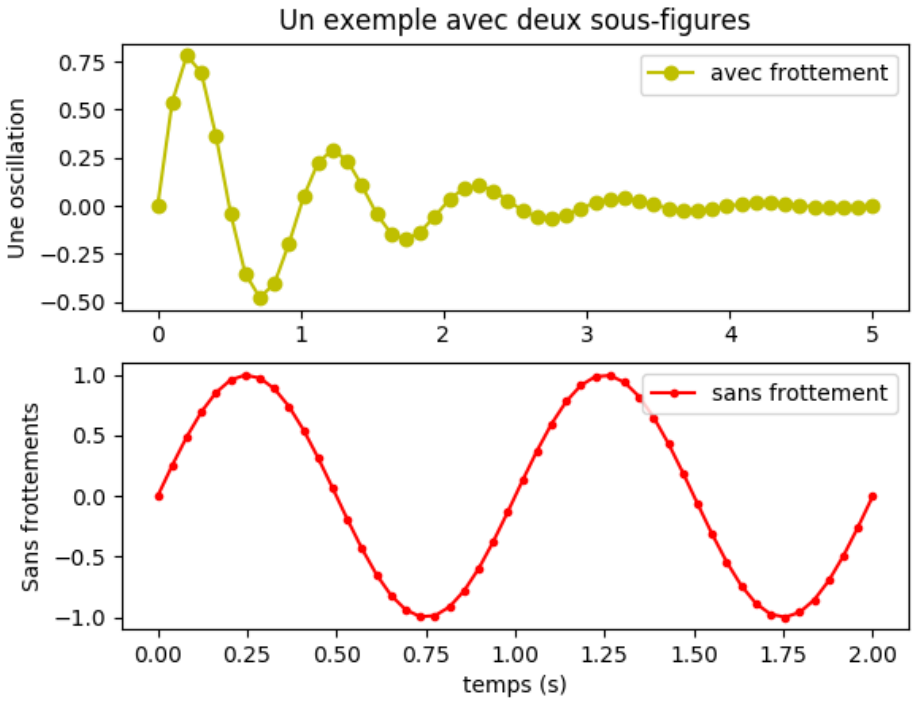
Sous-figures (2)

Voici un exemple,

```
# Fichier subplot.py
import numpy as np
import matplotlib.pyplot as plt
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)
y1 = np.sin(2 * np.pi * x1) * np.exp(-x1)
y2 = np.sin(2 * np.pi * x2)

plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'yo-', label ='avec frottement')
plt.title('Un exemple avec deux sous-figures')
plt.ylabel('Une oscillation')
plt.legend()
plt.subplot(2, 1, 2)
plt.plot(x2, y2, 'r.-', label ='sans frottement')
plt.xlabel('temps (s)')
plt.ylabel('Sans frottements')
plt.legend()
```

Sous-figures (3)

Voici le résultat,



Sous-figures avec axes

```
# Fichier subplot2.py
import numpy as np
import matplotlib.pyplot as plt
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)
y1 = np.sin(2 * np.pi * x1) * np.exp(-x1)
y2 = np.sin(2 * np.pi * x2)
fig, (ax1, ax2) = plt.subplots(2, 1)
plt.suptitle('Un exemple avec deux sous-figures')
ax1.plot(x1, y1, 'yo-', label ='avec frottement')

ax1.set_ylabel('Une oscillation')
ax1.legend()
ax2.plot(x2, y2, 'r.-', label ='sans frottement')
ax2.set_xlabel('temps (s)')
ax2.set_xlabel('Sans frottements')
ax2.legend()
plt.show()
```

Outline

Utilisation avec LaTeX

Il est possible d'utiliser LaTeX pour afficher les textes (et donc de mettre des équations). Pour cela il faut activer l'option `text.usetex` en la mettant `True` avec la fonction `rc`. Soit par exemple,

```
from matplotlib import rc
rc('font',**{'family':'sans-serif','sans-serif':['
    Helvetica']})
#rc('font',**{'family':'serif','serif':['Palatino']})
rc('text', usetex=True)
```

Note : L'option `font` permet de choisir la police de caractères.

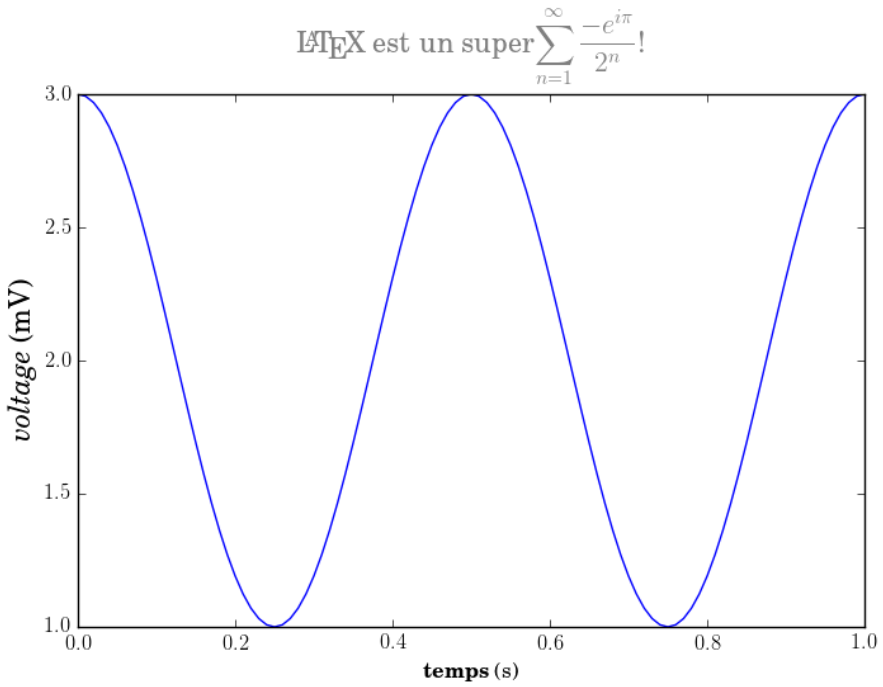
Utilisation avec LaTeX (2)

Un exemple plus complexe,

```
# Fichier latex.py
import numpy as np
import matplotlib.pyplot as plt


t = np.arange(0.0, 1.0 + 0.01, 0.01)
s = np.cos(4 * np.pi * t) + 2

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.plot(t, s)

plt.xlabel(r'\textbf{temps} (s)')
plt.ylabel(r'\textit{voltage} (mV)', fontsize=16)
plt.title(r"\LaTeX\ est un super"
          r"$\displaystyle\sum_{n=1}^\infty\frac{-e^{i\
              pi}}{2^n}$!",
          fontsize=16, color='gray')
```

Utilisation avec LaTeX (3)

Et voici le résultat,



Outline

Sauvegarder une figure

Il est possible de sauvegarder une figure dans un fichier `.png`, `.pdf` (selon les modules disponibles). Pour cela il suffit d'utiliser la fonction `savefig` qui prend comme argument le nom du fichier. Les options sont les suivantes,

- `dpi` pour changer la densité de pixels (donc changer la qualité de l'image) ;
- `facecolor` la couleur de la face du rectangle contenant la figure ;
- `edgecolor` la couleur des bords de la figure ;
- `papertype` le type de papier (A4, A3...) ;
- `format` le format de l'image (`.png`, `.pdf`...) ;
- `transparent` à mettre à `True` pour avoir une image transparente.

D'autres options sont disponibles, mais elles sont moins utiles.

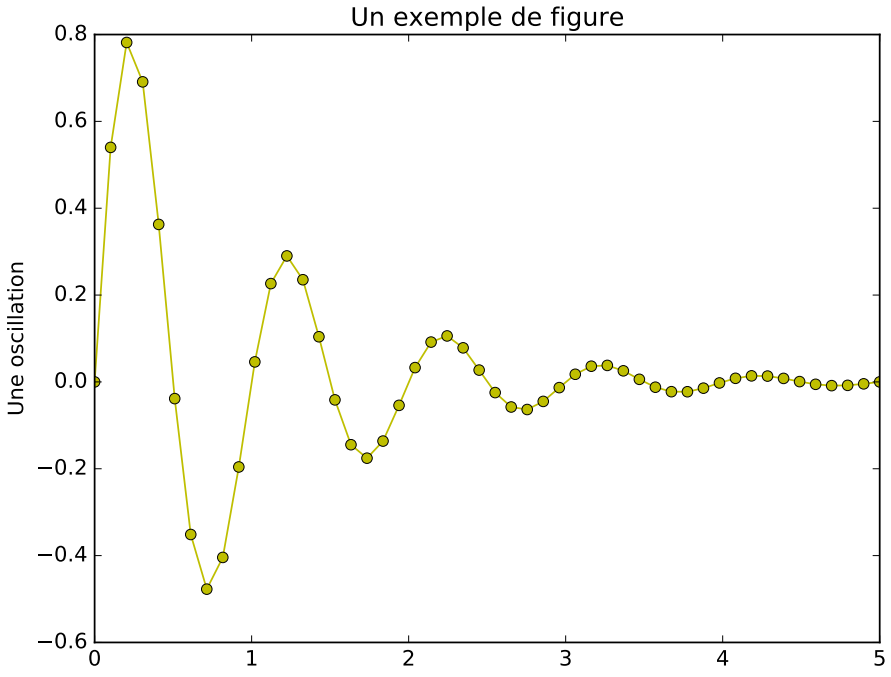
Sauvegarder une figure (2)

Voici un petit exemple,

```
# Fichier sauvegarde.py
import numpy as np
import matplotlib.pyplot as plt


x1 = np.linspace(0.0, 5.0)
y1 = np.sin(2 * np.pi * x1) * np.exp(-x1)

plt.plot(x1, y1, 'yo-')
plt.title('Un exemple de figure')
plt.ylabel('Une oscillation')
plt.savefig('fig.pdf', format='pdf')
```

Sauvegarder une figure (3)

Voici le résultat,



Outline

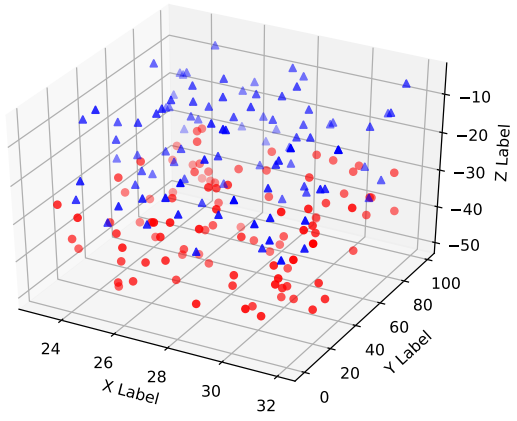
Matplotlib mplot3d toolkit

Cette extension de Matplotlib permet de faire des dessins 3D de base dont,

- traçage de nuage de points ;
- affichage de formes maillées ;
- traçage de surfaces ;
- affichage de contours ;
- ...

Nuages de points

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3
    d')
n = 100
for c, m, zlow, zhigh in [('r', 'o',
    -50, -25), ('b', '^', -30, -5)]:
    xs = (32-23)*np.random.rand(n)+23
    ys = 100*np.random.rand(n)
    zs = (zhigh-zlow)*np.random.rand(n)
        +zlow
    ax.scatter(xs, ys, zs, c=c, marker=
        m)
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.show()
```



Surfaces (1)

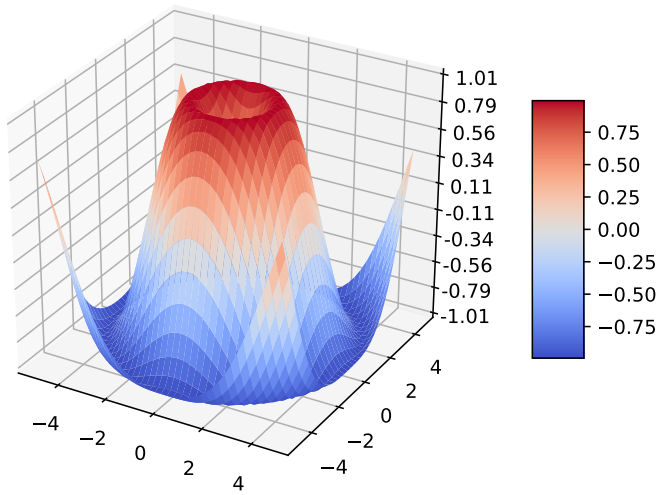
En entête, nous allons avoir :

```
1 from mpl_toolkits.mplot3d import Axes3D
2 import matplotlib.pyplot as plt
3 from matplotlib import cm
4 from matplotlib.ticker import LinearLocator,
      FormatStrFormatter
5 import numpy as np
```

- ligne 1 nécessaire pour avoir l'affichage 3d ;
- ligne 3 pour récupérer une gamme de couleurs (colormap) ;
- ligne 4 pour la gestion de la numérotation sur les axes.

Surfaces (2)

```
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

fig = plt.figure()
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap
    =cm.coolwarm, linewidth=0,
    antialiased=False)
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(
    LinearLocator(10))
ax.zaxis.set_major_formatter(
    FormatStrFormatter('%.02f'))
fig.colorbar(surf, shrink=0.5,
    aspect=5)
plt.show()
```



Extensions et autres

Des extensions sont disponibles de bases. Elles permettent entre autre,

- annotations complexes
- gestion de carte/atlas...

Notez aussi l'existence de modules proches de Matplotlib, comme,

- *prettyplot* dont les choix de couleurs par défauts sont souvent meilleurs,
- *seaborn* qui étend les possibilités de Matplotlib,
- *plotly* qui permet d'avoir des figures interactives dans son navigateur...

Conclusion

Il existe encore d'autres modules pour faire des graphiques scientifiques,

- `Seaborn` pour la gestion de graphiques 2D et 3D,
- `Plot.ly` concurrent aussi à `Matplotlib` propose un bonne intégration avec IPython,
- `Chaco` basé sur VTK pour des graphiques 2D,
- `Veusz` proposant une bonne intégration avec QT,
- `PyX` pour créer des figures en SVG ou PDF,
- `pyla` basé sur Gnuplot pour l'affichage
- . . . beaucoup d'autres modules existent.