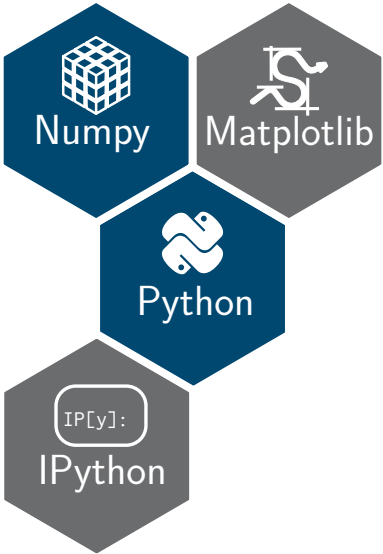


Option « Programmation en Python »

`matplotlib` : librairie pour la représentation graphique

Librarie `matplotlib`

Kozmetix sous `matplotlib`

Les modes de représentation

Interaction avec `matplotlib`

`matplotlib` et plus si affinités

matplotlib ?

- La librairie `matplotlib` est **la** bibliothèque graphique de Python
- Étroitement liée à `numpy` et `scipy`
- Grande variété de **format de sortie (`png, pdf, svg, eps, pgf`)** ainsi que support de LaTeX pour le texte
- *Graphical User Interface* pour l'exploration interactive des figures (zoom, sélection,...)
- Tous les aspects d'une figure (taille, position,...) peuvent être contrôlés d'un point de vue *programmatique* → **reproductibilité** des figures et des résultats scientifiques

matplotlib ?

- La librairie `matplotlib` est **la** bibliothèque graphique de Python
- Étroitement liée à `numpy` et `scipy`
- Grande variété de **format de sortie (`png`, `pdf`, `svg`, `eps`, `pgf`)** ainsi que support de LaTeX pour le texte
- *Graphical User Interface* pour l'exploration interactive des figures (zoom, sélection,...)
- Tous les aspects d'une figure (taille, position,...) peuvent être contrôlés d'un point de vue *programmatique* → **reproductibilité** des figures et des résultats scientifiques

matplotlib ?

- La librairie `matplotlib` est **la** bibliothèque graphique de Python
- Étroitement liée à `numpy` et `scipy`
- Grande variété de **format de sortie (`png`, `pdf`, `svg`, `eps`, `pgf`)** ainsi que support de LaTeX pour le texte
- *Graphical User Interface* pour l'exploration interactive des figures (zoom, sélection,...)
- Tous les aspects d'une figure (taille, position,...) peuvent être contrôlés d'un point de vue *programmatique* → **reproductibilité** des figures et des résultats scientifiques

- La librairie `matplotlib` est **la** bibliothèque graphique de Python
- Étroitement liée à `numpy` et `scipy`
- Grande variété de **format de sortie (`png`, `pdf`, `svg`, `eps`, `pgf`)** ainsi que support de LaTeX pour le texte
- *Graphical User Interface* pour l'exploration interactive des figures (zoom, sélection,...)
- Tous les aspects d'une figure (taille, position,...) peuvent être contrôlés d'un point de vue *programmatique* → **reproductibilité** des figures et des résultats scientifiques

matplotlib ?

- La librairie `matplotlib` est **la** bibliothèque graphique de Python
- Étroitement liée à `numpy` et `scipy`
- Grande variété de **format de sortie (`png`, `pdf`, `svg`, `eps`, `pgf`)** ainsi que support de LaTeX pour le texte
- *Graphical User Interface* pour l'exploration interactive des figures (zoom, sélection,...)
- Tous les aspects d'une figure (taille, position,...) peuvent être contrôlés d'un point de vue *programmatique* $\rightarrow$ **reproductibilité** des figures et des résultats scientifiques

Installation & importation de `matplotlib`

- Installation *via* `pip`

```
>_ pip install matplotlib
```

- Convention d'importation

```
In [1]: import matplotlib as mpl
In [2]: import matplotlib.pyplot as plt

In [3]: mpl.__version__
Out[3]: '2.0.0'
```

Installation & importation de `matplotlib`

- Installation *via* `pip`

```
>_ pip install matplotlib
```

- Convention d'importation

```
In [1]: import matplotlib as mpl
In [2]: import matplotlib.pyplot as plt

In [3]: mpl.__version__
Out[3]: '2.0.0'
```

Comment afficher vos figures : `show()` or not `show()`

- Affichage depuis un script python

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

x = np.linspace(0, 3*np.pi, 100)

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```

Comment afficher vos figures : `show()` or not `show()`

- Affichage depuis un script python

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  x = np.linspace(0, 3*np.pi, 100)
5
6  plt.plot(x, np.sin(x))
7  plt.plot(x, np.cos(x))
8
9  plt.show()
```

Utilisation de **plt.show()**

Comment afficher vos figures : `show()` or not `show()`

- Affichage depuis la console `ipython`

```
In [1]: %matplotlib
Using matplotlib backend: TkAgg

In [2]: import matplotlib.pyplot as plt
In [3]: import numpy as np

In [4]: x = np.linspace(0, 3*np.pi, 100)

In [6]: plt.plot(x, np.sin(x))
In [7]: plt.plot(x, np.cos(x))
```

Comment afficher vos figures : `show()` or not `show()`

- Affichage depuis la console `ipython`

```
In [1]: %matplotlib
Using matplotlib backend: TkAgg

In [2]: import matplotlib.pyplot as plt
In [3]: import numpy as np

In [4]: x = np.linspace(0, 3*np.pi, 100)

In [6]: plt.plot(x, np.sin(x))
In [7]: plt.plot(x, np.cos(x))
```

Utilisation de **%matplotlib**

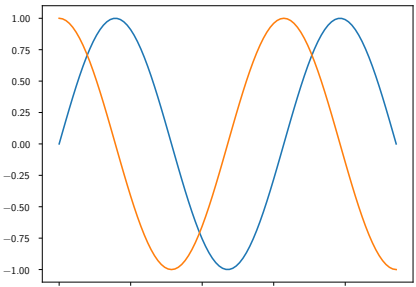
- Possibilité également de lancer la commande `ipython` avec l'option `--matplotlib`

Première figure sous `matplotlib`

```
In [1]: %matplotlib
In [2]: import matplotlib.pyplot as plt
In [3]: import numpy as np

In [4]: x = np.linspace(0, 3*np.pi, 100)

In [5]: plt.plot(x, np.sin(x))
In [6]: plt.plot(x, np.cos(x))
```

- Sauvegarder la figure (eps, pdf, png)

```
In [7]: plt.savefig("/tmp/mpl1.pdf")
```

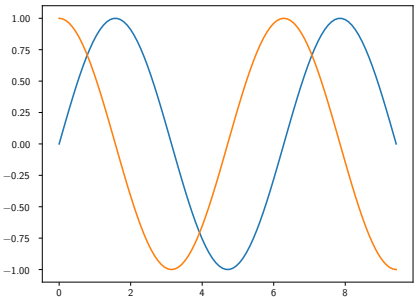


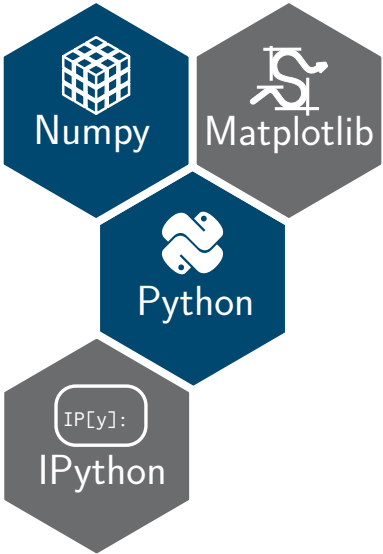
Première figure sous `matplotlib`

```
In [1]: %matplotlib
In [2]: import matplotlib.pyplot as plt
In [3]: import numpy as np

In [4]: x = np.linspace(0, 3*np.pi, 100)

In [5]: plt.plot(x, np.sin(x))
In [6]: plt.plot(x, np.cos(x))
```

- Sauvegarder la figure (eps, pdf, png)

```
In [7]: plt.savefig("/tmp/mpl1.pdf")
```





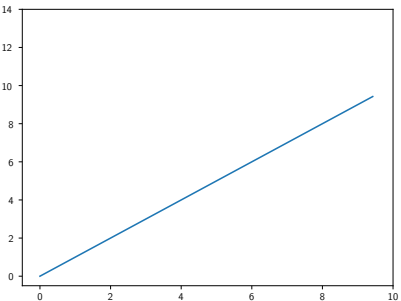
Option « Programmation en Python »

Épisode 1 : *Kozmetix* sous `matplotlib`

```
In [8]: plt.plot(x, x + 0, linestyle="solid")
In [9]: plt.plot(x, x + 1, linestyle="dashed")
In[10]: plt.plot(x, x + 2, linestyle="dashdot")
In[11]: plt.plot(x, x + 3, linestyle="dotted")
```

- Il est également possible d'utiliser les notations raccourcies

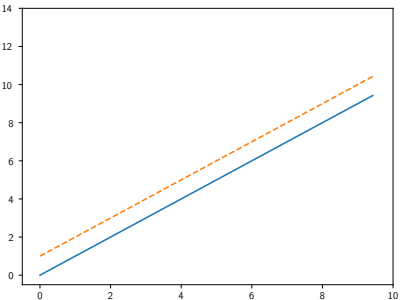
`-`	≡	solid
`--`	≡	dashed
`-.`	≡	dashdot
`:`	≡	dotted



```
In [8]: plt.plot(x, x + 0, linestyle="solid")
In [9]: plt.plot(x, x + 1, linestyle="dashed")
In[10]: plt.plot(x, x + 2, linestyle="dashdot")
In[11]: plt.plot(x, x + 3, linestyle="dotted")
```

- Il est également possible d'utiliser les notations raccourcies

  `-` ≡ solid
  `--` ≡ dashed
  `-.` ≡ dashdot
  `:` ≡ dotted



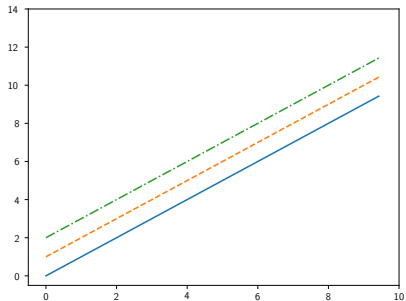
Kozmetix sous `matplotlib`

👉 Lignes, marqueurs : styles & couleurs

```
In [8]: plt.plot(x, x + 0, linestyle="solid")
In [9]: plt.plot(x, x + 1, linestyle="dashed")
In[10]: plt.plot(x, x + 2, linestyle="dashdot")
In[11]: plt.plot(x, x + 3, linestyle="dotted")
```

- Il est également possible d'utiliser les notations raccourcies

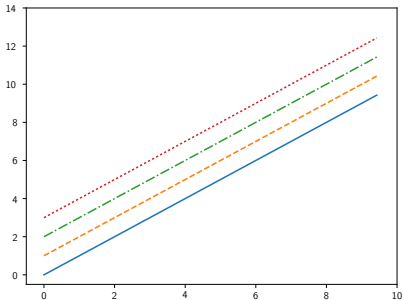
```
-   ≡  solid
--  ≡  dashed
-.  ≡  dashdot
:   ≡  dotted
```



```
In [8]: plt.plot(x, x + 0, linestyle="solid")
In [9]: plt.plot(x, x + 1, linestyle="dashed")
In[10]: plt.plot(x, x + 2, linestyle="dashdot")
In[11]: plt.plot(x, x + 3, linestyle="dotted")
```

- Il est également possible d'utiliser les notations raccourcies

`-`	≡	solid
`--`	≡	dashed
`-.`	≡	dashdot
`:`	≡	dotted



Kozmetix sous `matplotlib`

👉 Lignes, marqueurs : styles & couleurs

- En spécifiant le nom de la couleur

```
In [8]: plt.plot(x, np.sin(x - 0), color="blue")
```

- Nom raccourci (`rgbcmyk`)

```
In [9]: plt.plot(x, np.sin(x - 1), color="g")
```

- Échelle de gris `[0; 1]`

```
In[10]: plt.plot(x, np.sin(x - 2), color="0.75")
```

- Code héxadécimal (`RRGGBB`)

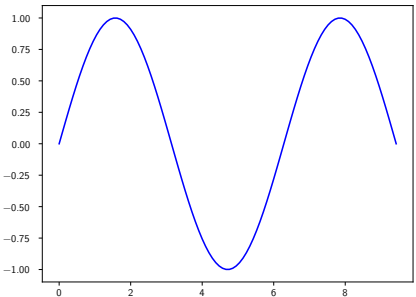
```
In[11]: plt.plot(x, np.sin(x - 3),
                 color="#FFDD44")
```

- RGB *tuple* `[0; 1]`

```
In[12]: plt.plot(x, np.sin(x - 4),
                 color=(1.0,0.2,0.3))
```

- Couleur du cycle `C0-9`

```
In[13]: plt.plot(x, np.sin(x - 5), color="C4")
```



Kozmetix sous `matplotlib`

👉 Lignes, marqueurs : styles & couleurs

- En spécifiant le nom de la couleur

```
In [8]: plt.plot(x, np.sin(x - 0), color="blue")
```

- Nom raccourci (`rgbcmyk`)

```
In [9]: plt.plot(x, np.sin(x - 1), color="g")
```

- Échelle de gris `[0; 1]`

```
In[10]: plt.plot(x, np.sin(x - 2), color="0.75")
```

- Code héxadécimal (`RRGGBB`)

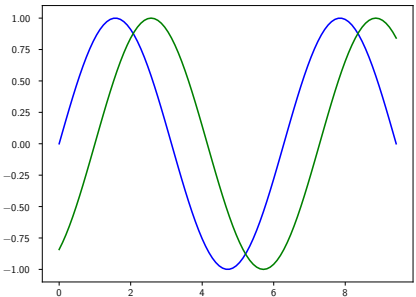
```
In[11]: plt.plot(x, np.sin(x - 3),
                 color="#FFDD44")
```

- RGB *tuple* `[0; 1]`

```
In[12]: plt.plot(x, np.sin(x - 4),
                 color=(1.0,0.2,0.3))
```

- Couleur du cycle `C0-9`

```
In[13]: plt.plot(x, np.sin(x - 5), color="C4")
```



Kozmetix sous `matplotlib`

👉 Lignes, marqueurs : styles & couleurs

- En spécifiant le nom de la couleur

```
In [8]: plt.plot(x, np.sin(x - 0), color="blue")
```

- Nom raccourci (`rgbcmyk`)

```
In [9]: plt.plot(x, np.sin(x - 1), color="g")
```

- Échelle de gris `[0; 1]`

```
In[10]: plt.plot(x, np.sin(x - 2), color="0.75")
```

- Code héxadécimal (`RRGGBB`)

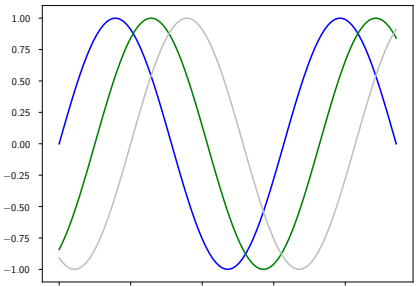
```
In[11]: plt.plot(x, np.sin(x - 3),
                 color="#FFDD44")
```

- RGB *tuple* `[0; 1]`

```
In[12]: plt.plot(x, np.sin(x - 4),
                 color=(1.0,0.2,0.3))
```

- Couleur du cycle `C0-9`

```
In[13]: plt.plot(x, np.sin(x - 5), color="C4")
```



Kozmetix sous `matplotlib`

👉 Lignes, marqueurs : styles & couleurs

- En spécifiant le nom de la couleur

```
In [8]: plt.plot(x, np.sin(x - 0), color="blue")
```

- Nom raccourci (`rgbcmyk`)

```
In [9]: plt.plot(x, np.sin(x - 1), color="g")
```

- Échelle de gris `[0; 1]`

```
In[10]: plt.plot(x, np.sin(x - 2), color="0.75")
```

- Code héxadécimal (`RRGGBB`)

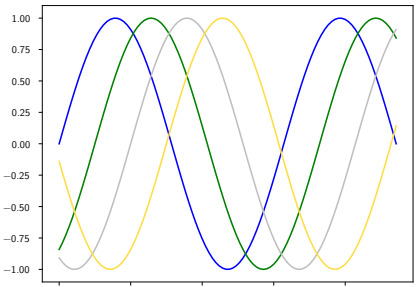
```
In[11]: plt.plot(x, np.sin(x - 3),
                 color="#FFDD44")
```

- RGB *tuple* `[0; 1]`

```
In[12]: plt.plot(x, np.sin(x - 4),
                 color=(1.0,0.2,0.3))
```

- Couleur du cycle `C0-9`

```
In[13]: plt.plot(x, np.sin(x - 5), color="C4")
```



Kozmetix sous `matplotlib`

👉 Lignes, marqueurs : styles & couleurs

- En spécifiant le nom de la couleur

```
In [8]: plt.plot(x, np.sin(x - 0), color="blue")
```

- Nom raccourci (`rgbcmyk`)

```
In [9]: plt.plot(x, np.sin(x - 1), color="g")
```

- Échelle de gris `[0; 1]`

```
In[10]: plt.plot(x, np.sin(x - 2), color="0.75")
```

- Code héxadécimal (`RRGGBB`)

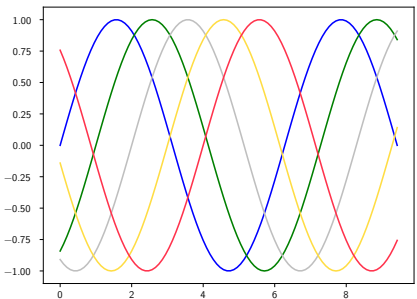
```
In[11]: plt.plot(x, np.sin(x - 3),
                 color="#FFDD44")
```

- RGB *tuple* `[0; 1]`

```
In[12]: plt.plot(x, np.sin(x - 4),
                 color=(1.0,0.2,0.3))
```

- Couleur du cycle `C0-9`

```
In[13]: plt.plot(x, np.sin(x - 5), color="C4")
```



- En spécifiant le nom de la couleur

```
In [8]: plt.plot(x, np.sin(x - 0), color="blue")
```

- Nom raccourci (rgbcmyk)

```
In [9]: plt.plot(x, np.sin(x - 1), color="g")
```

- Échelle de gris [0; 1]

```
In[10]: plt.plot(x, np.sin(x - 2), color="0.75")
```

- Code héxadécimal (RRGGBB)

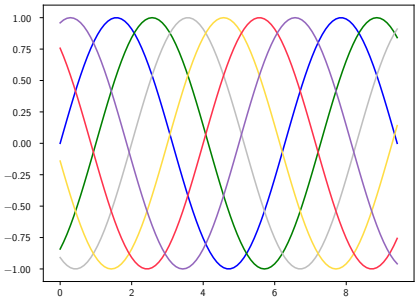
```
In[11]: plt.plot(x, np.sin(x - 3),
                 color="#FFDD44")
```

- RGB *tuple* [0; 1]

```
In[12]: plt.plot(x, np.sin(x - 4),
                 color=(1.0,0.2,0.3))
```

- Couleur du cycle C0-9

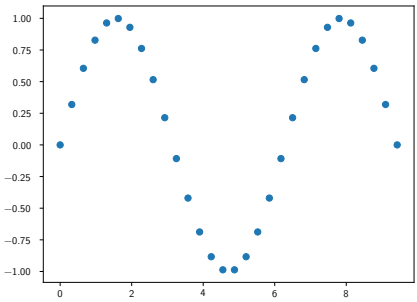
```
In[13]: plt.plot(x, np.sin(x - 5), color="C4")
```



Kozmetix sous `matplotlib`

👉 Lignes, marqueurs : styles & couleurs

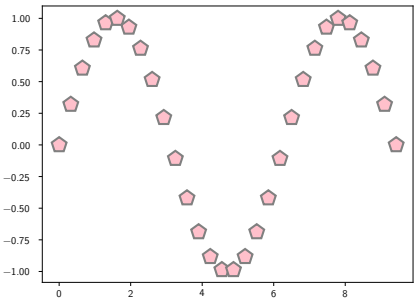
```
In [4]: x = np.linspace(0, 3*np.pi, 30)
In [5]: plt.plot(x, np.sin(x), "o")
In [6]: plt.plot(x, np.sin(x), "p",
   ...:          markersize=15,
   ...:          markerfacecolor="pink",
   ...:          markeredgecolor="gray",
   ...:          markeredgewidth=2)
```



Kozmetix sous `matplotlib`

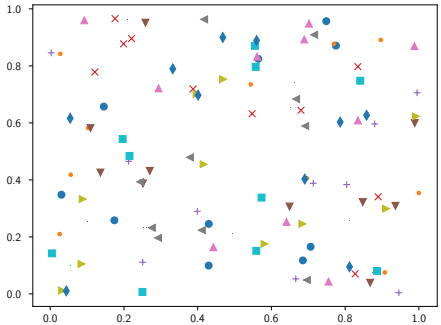
👉 Lignes, marqueurs : styles & couleurs

```
In [4]: x = np.linspace(0, 3*np.pi, 30)
In [5]: plt.plot(x, np.sin(x), "o")
In [6]: plt.plot(x, np.sin(x), "p",
   ...:          markersize=15,
   ...:          markerfacecolor="pink",
   ...:          markeredgecolor="gray",
   ...:          markeredgewidth=2)
```



Kozmetix sous matplotlib

👉 Lignes, marqueurs : styles & couleurs

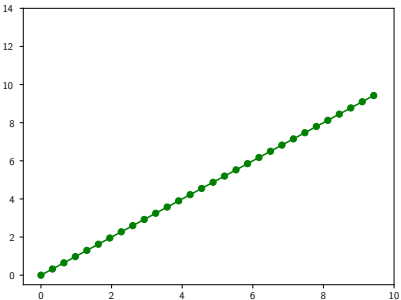



```
In [7]: for marker in ["o", ".", ",", "x", "+", "v", "^", "<", ">", "s", "d"]:
  ...:     plt.plot(np.random.rand(10), np.random.rand(10), marker)
```

- Il est finalement possible de combiner style & couleur au sein d'une syntaxe minimaliste

```
In [8]: plt.plot(x, x + 0, "-og")
In [9]: plt.plot(x, x + 1, "--xc")
In[10]: plt.plot(x, x + 2, "-.k")
In[11]: plt.plot(x, x + 3, ":sr")
```

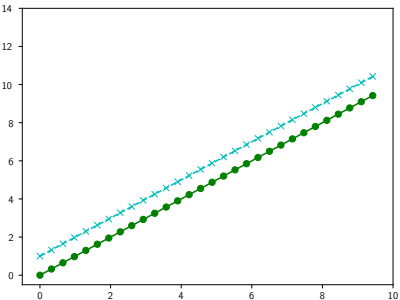
- Pour découvrir l'ensemble des options d'affichage `plt.plot?` ou `help(plt.plot)`



- Il est finalement possible de combiner style & couleur au sein d'une syntaxe minimaliste

```
In [8]: plt.plot(x, x + 0, "-og")
In [9]: plt.plot(x, x + 1, "--xc")
In[10]: plt.plot(x, x + 2, "-.k")
In[11]: plt.plot(x, x + 3, ":sr")
```

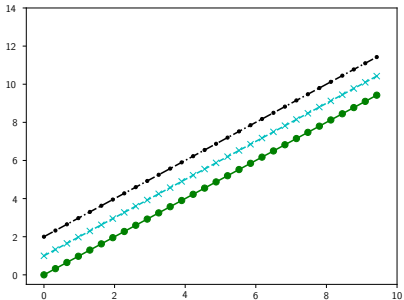
- Pour découvrir l'ensemble des options d'affichage `plt.plot?` ou `help(plt.plot)`



- Il est finalement possible de combiner style & couleur au sein d'une syntaxe minimaliste

```
In [8]: plt.plot(x, x + 0, "-og")
In [9]: plt.plot(x, x + 1, "--xc")
In[10]: plt.plot(x, x + 2, "-..k")
In[11]: plt.plot(x, x + 3, ":sr")
```

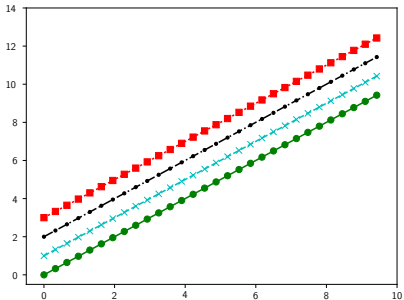
- Pour découvrir l'ensemble des options d'affichage `plt.plot?` ou `help(plt.plot)`



- Il est finalement possible de combiner style & couleur au sein d'une syntaxe minimaliste

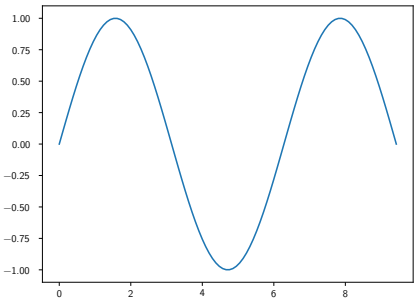
```
In [8]: plt.plot(x, x + 0, "-og")
In [9]: plt.plot(x, x + 1, "--xc")
In[10]: plt.plot(x, x + 2, "-..k")
In[11]: plt.plot(x, x + 3, ":sr")
```

- Pour découvrir l'ensemble des options d'affichage **`plt.plot?`** ou **`help(plt.plot)`**



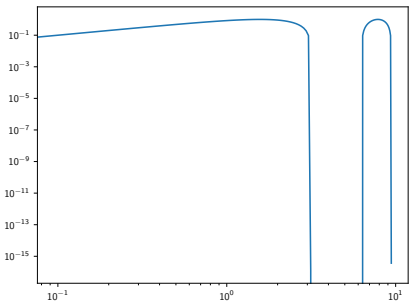
Kozmetix sous `matplotlib`

👉 Axes : échelle, limites & *ticks*



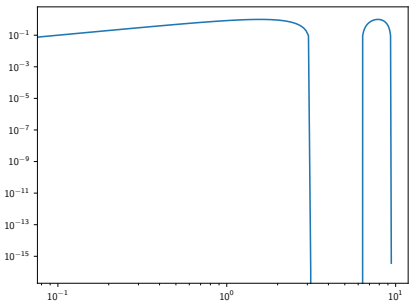

```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x))
In [6]: plt.xscale("log")
In [7]: plt.yscale("log")
In [8]: plt.loglog(x, np.sin(x))
```

- Pour découvrir l'ensemble des options d'affichage `plt.xscale?` ou `help(plt.xscale)`



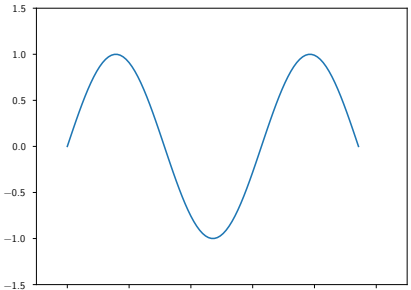

```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x))
In [6]: plt.xscale("log")
In [7]: plt.yscale("log")
In [8]: plt.loglog(x, np.sin(x))
```

- Pour découvrir l'ensemble des options d'affichage `plt.xscale?` ou `help(plt.xscale)`



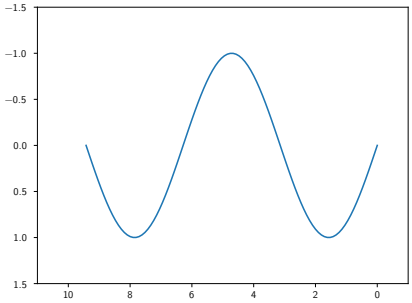

```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x))
In [6]: plt.xscale("log")
In [7]: plt.yscale("log")
In [8]: plt.loglog(x, np.sin(x))
```

- Pour découvrir l'ensemble des options d'affichage **`plt.xscale?`** ou **`help(plt.xscale)`**



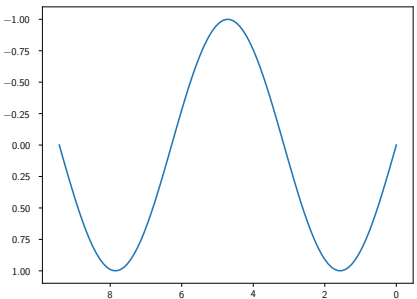

```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x))

In [6]: plt.xlim(-1, 11)
In [7]: plt.ylim(-1.5, 1.5)
In [8]: plt.axis([11, -1, 1.5, -1.5])
In [9]: plt.axis("tight")
In[10]: plt.axis("equal")
```

- Pour découvrir l'ensemble des options d'affichage `plt.axis?` ou `help(plt.axis)`



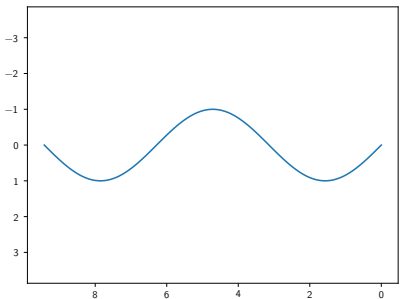

```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x))

In [6]: plt.xlim(-1, 11)
In [7]: plt.ylim(-1.5, 1.5)
In [8]: plt.axis([11, -1, 1.5, -1.5])
In [9]: plt.axis("tight")
In[10]: plt.axis("equal")
```

- Pour découvrir l'ensemble des options d'affichage `plt.axis?` ou `help(plt.axis)`



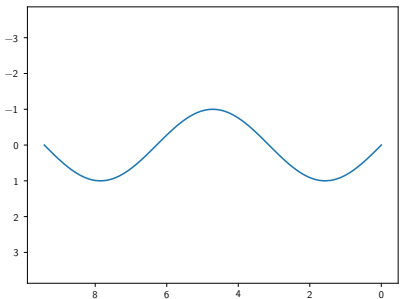

```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x))

In [6]: plt.xlim(-1, 11)
In [7]: plt.ylim(-1.5, 1.5)
In [8]: plt.axis([11, -1, 1.5, -1.5])
In [9]: plt.axis("tight")
In[10]: plt.axis("equal")
```

- Pour découvrir l'ensemble des options d'affichage `plt.axis?` ou `help(plt.axis)`



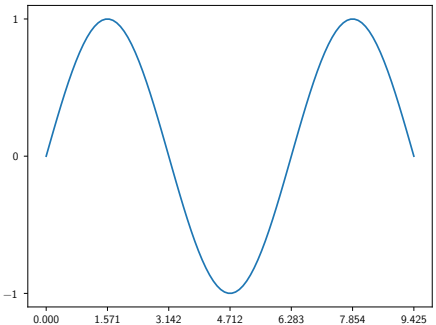

```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x))

In [6]: plt.xlim(-1, 11)
In [7]: plt.ylim(-1.5, 1.5)
In [8]: plt.axis([11, -1, 1.5, -1.5])
In [9]: plt.axis("tight")
In[10]: plt.axis("equal")
```

- Pour découvrir l'ensemble des options d'affichage **`plt.axis?`** ou **`help(plt.axis)`**

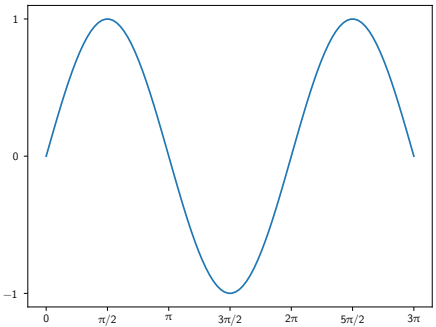



```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x))

In [6]: plt.xlim(-1, 11)
In [7]: plt.ylim(-1.5, 1.5)
In [8]: plt.axis([11, -1, 1.5, -1.5])
In [9]: plt.axis("tight")
In[10]: plt.axis("equal")
```

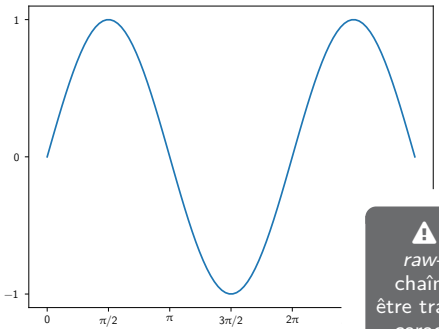
- Pour découvrir l'ensemble des options d'affichage **`plt.axis?`** ou **`help(plt.axis)`**




```
In[11]: plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi, 5*np.pi/2, 3*np.pi])
In[12]: plt.yticks([-1. 0. +1])
In[13]: plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi, 5*np.pi/2, 3*np.pi],
                   [r"$0$", r"$\pi/2$", r"$\pi$", r"$3\pi/2$", r"$2\pi", r"$5\pi/2$", r"$3\pi$"])
```

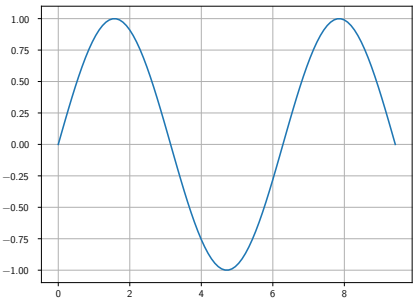



```
In[11]: plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi, 5*np.pi/2, 3*np.pi])
In[12]: plt.yticks([-1. 0. +1])
In[13]: plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi, 5*np.pi/2, 3*np.pi],
                   [r"$0$", r"$\pi/2$", r"$\pi$", r"$3\pi/2$", r"$2\pi", r"$5\pi/2$", r"$3\pi$"])
```



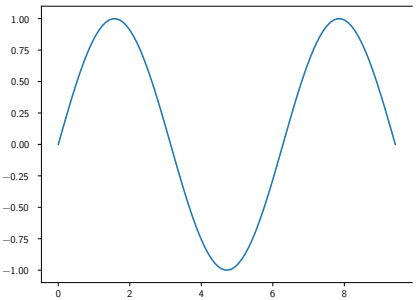
⚠ Le prefixe r pour *raw-text* indique que la chaîne de caractères doit être traiter sans échapper les caractères précédés de \

```
In[11]: plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi, 5*np.pi/2, 3*np.pi])
In[12]: plt.yticks([-1, 0, +1])
In[13]: plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi, 5*np.pi/2, 3*np.pi],
                   [r"$0$", r"$\pi/2$", r"$\pi$", r"$3\pi/2$", r"$2\pi", r"$5\pi/2$", r"$3\pi$"])
```



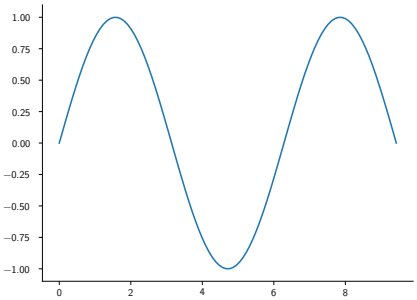
- Accéder aux axes de la figure (gca ≡ get current axis)

```
In [4]: ax = plt.gca()
In [5]: ax.grid()
In [6]: ax.spines["right"].set_color("none")
In [7]: ax.spines["top"].set_color("none")
In [8]: ax.spines["bottom"].set_position(("data",0))
```



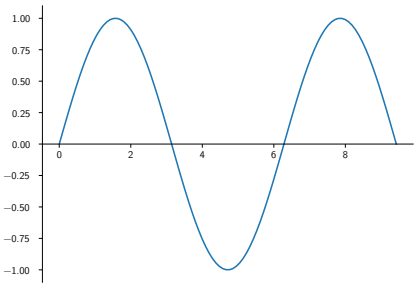
- Accéder aux axes de la figure (gca ≡ get current axis)

```
In [4]: ax = plt.gca()
In [5]: ax.grid()
In [6]: ax.spines["right"].set_color("none")
In [7]: ax.spines["top"].set_color("none")
In [8]: ax.spines["bottom"].set_position(("data",0))
```



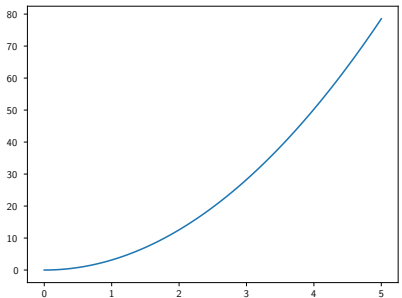
- Accéder aux axes de la figure (gca ≡ get current axis)

```
In [4]: ax = plt.gca()
In [5]: ax.grid()
In [6]: ax.spines["right"].set_color("none")
In [7]: ax.spines["top"].set_color("none")
In [8]: ax.spines["bottom"].set_position(("data",0))
```

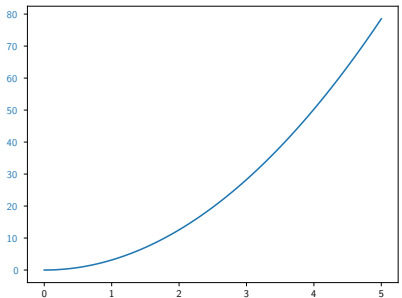


- Accéder aux axes de la figure (gca ≡ get current axis)

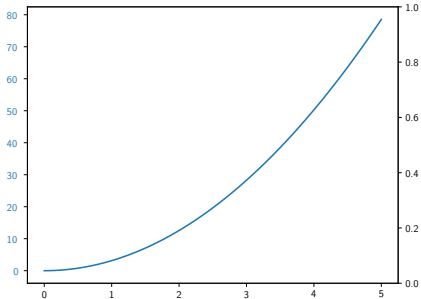
```
In [4]: ax = plt.gca()
In [5]: ax.grid()
In [6]: ax.spines["right"].set_color("none")
In [7]: ax.spines["top"].set_color("none")
In [8]: ax.spines["bottom"].set_position(("data",0))
```




```
In [1]: r = np.linspace(0, 5, 100)
In [2]: plt.plot(r, np.pi*r**2, color="C0")
In [3]: ax = plt.gca()
In [4]: for label in ax.get_yticklabels():
   ...:     label.set_color("C0")
In [5]: plt.twinx()
In [6]: plt.plot(r, 4/3*np.pi*r**3, color="C1")
In [7]: ax = plt.gca()
In [8]: for label in ax.get_yticklabels():
   ...:     label.set_color("C1")
```



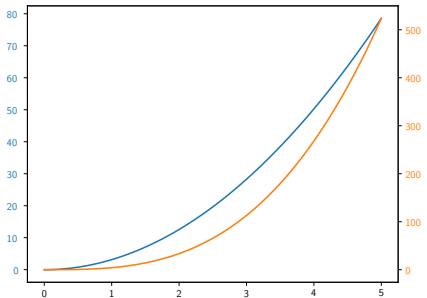

```
In [1]: r = np.linspace(0, 5, 100)
In [2]: plt.plot(r, np.pi*r**2, color="C0")
In [3]: ax = plt.gca()
In [4]: for label in ax.get_yticklabels():
   ...:     label.set_color("C0")
In [5]: plt.twinx()
In [6]: plt.plot(r, 4/3*np.pi*r**3, color="C1")
In [7]: ax = plt.gca()
In [8]: for label in ax.get_yticklabels():
   ...:     label.set_color("C1")
```



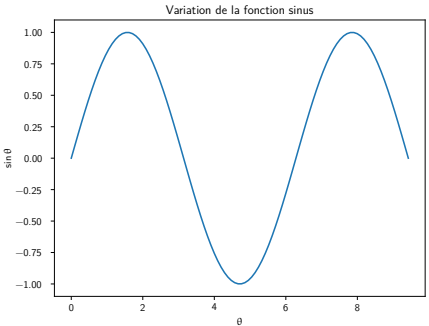

```
In [1]: r = np.linspace(0, 5, 100)
In [2]: plt.plot(r, np.pi*r**2, color="C0")
In [3]: ax = plt.gca()
In [4]: for label in ax.get_yticklabels():
   ...:     label.set_color("C0")
In [5]: plt.twinx()
In [6]: plt.plot(r, 4/3*np.pi*r**3, color="C1")
In [7]: ax = plt.gca()
In [8]: for label in ax.get_yticklabels():
   ...:     label.set_color("C1")
```

Kozmetix sous `matplotlib`

👉 Axes : échelles, limites & *ticks*



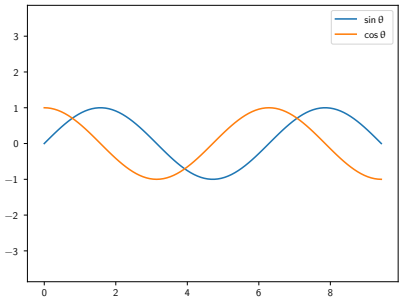

```
In [1]: r = np.linspace(0, 5, 100)
In [2]: plt.plot(r, np.pi*r**2, color="C0")

In [3]: ax = plt.gca()
In [4]: for label in ax.get_yticklabels():
   ...:     label.set_color("C0")

In [5]: plt.twinx()

In [6]: plt.plot(r, 4/3*np.pi*r**3, color="C1")
In [7]: ax = plt.gca()
In [8]: for label in ax.get_yticklabels():
   ...:     label.set_color("C1")
```




```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x))

In [6]: plt.title("Variation de la fonction sinus")
In [7]: plt.xlabel(r"$\theta$")
In [8]: plt.ylabel(r"$\sin\theta$")
```

Kozmetix sous `matplotlib`

👉 *Labelling* : titre, axes, légendes et autres annotations

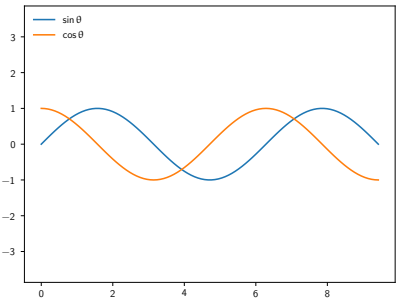



```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x), label=r"$\sin\theta$")
In [6]: plt.plot(x, np.cos(x), label=r"$\cos\theta$")
In [7]: plt.axis("equal")

In [8]: plt.legend()
In [9]: plt.legend(loc="upper left", frameon=False)
In[10]: plt.legend(loc="lower center", frameon=False,
                   ncol=2)
```

- Pour découvrir l'ensemble des options d'affichage `plt.legend?` ou `help(plt.legend)`

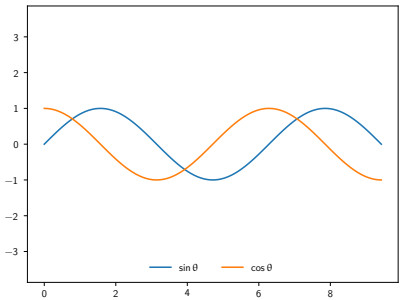
Kozmetix sous `matplotlib`

👉 *Labelling* : titre, axes, légendes et autres annotations



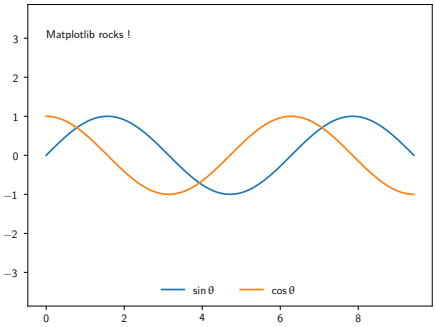

```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x), label=r"$\sin\theta$")
In [6]: plt.plot(x, np.cos(x), label=r"$\cos\theta$")
In [7]: plt.axis("equal")

In [8]: plt.legend()
In [9]: plt.legend(loc="upper left", frameon=False)
In[10]: plt.legend(loc="lower center", frameon=False,
                   ncol=2)
```

- Pour découvrir l'ensemble des options d'affichage `plt.legend?` ou `help(plt.legend)`

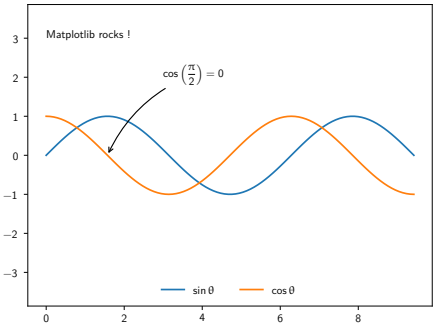



```
In [4]: x = np.linspace(0, 3*np.pi, 100)
In [5]: plt.plot(x, np.sin(x), label=r"$\sin\theta$")
In [6]: plt.plot(x, np.cos(x), label=r"$\cos\theta$")
In [7]: plt.axis("equal")

In [8]: plt.legend()
In [9]: plt.legend(loc="upper left", frameon=False)
In[10]: plt.legend(loc="lower center", frameon=False,
                   ncol=2)
```

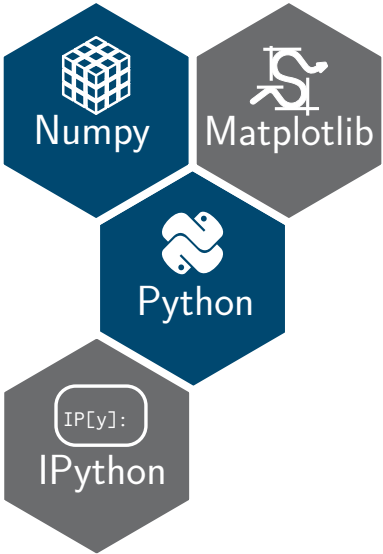
- Pour découvrir l'ensemble des options d'affichage **`plt.legend?`** ou **`help(plt.legend)`**




```
In[11]: plt.text(0, 3, "Matplotlib rocks !")
In[12]: plt.annotate(r"$\cos\left(\frac{\pi}{2}\right)=0$",
                     xy=(np.pi/2, np.cos(np.pi/2)), xytext=(3, 2),
                     arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
```



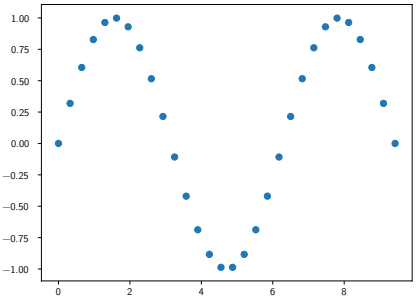

```
In[11]: plt.text(0, 3, "Matplotlib rocks !")
In[12]: plt.annotate(r"$\cos\left(\frac{\pi}{2}\right)=0$",
                     xy=(np.pi/2, np.cos(np.pi/2)), xytext=(3, 2),
                     arrowprops=dict(arrowstyle="->", connectionstyle="arc3,rad=.2"))
```



Option « Programmation en Python »

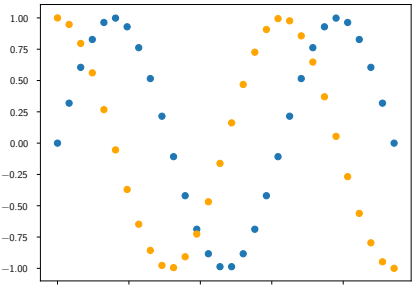
Épisode 2 : Les modes de représentation

Scatter plot



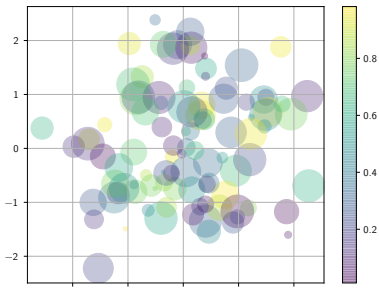

```
In [1]: x = np.linspace(0, 3*np.pi, 30)
In [2]: plt.scatter(x, np.sin(x), marker="o")
In [3]: plt.plot(x, np.cos(x), "o", color="orange")
```

Scatter plot




```
In [1]: x = np.linspace(0, 3*np.pi, 30)
In [2]: plt.scatter(x, np.sin(x), marker="o")
In [3]: plt.plot(x, np.cos(x), "o", color="orange")
```

- Le mode *scatter* permet de contrôler (taille, couleur) chaque point/marqueur individuellement

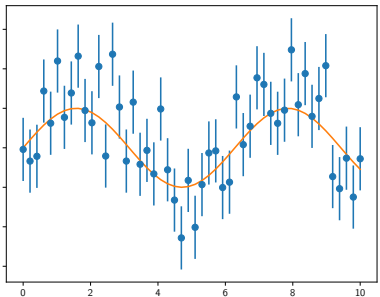



```
In [1]: rng = np.random
In [2]: x = rng.randn(100)
In [3]: y = rng.randn(100)
In [4]: colors = rng.rand(100)
In [5]: sizes = 1000 * rng.rand(100)

In [6]: plt.grid()
In [7]: plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
                    cmap="viridis")
In [8]: plt.colorbar()
```

Barres d'erreur

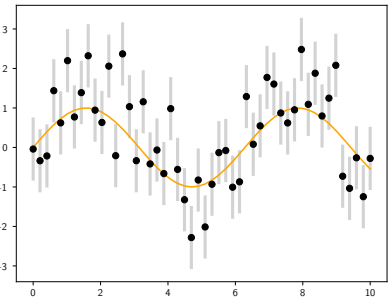
```
In [1]: x = np.linspace(0, 10, 50)
In [2]: dy = 0.8
In [3]: y = np.sin(x) + dy * np.random.randn(50)

In [4]: plt.errorbar(x, y, yerr=dy, fmt="o")
In [5]: plt.plot(x, np.sin(x))
```

```
In [6]: plt.errorbar(x, y, yerr=dy,
                     fmt="o", color="black",
                     ecolor="lightgray",
                     elinewidth=3,
                     capsize=0)

In [7]: plt.fill_between(x, np.sin(x)-dy, np.sin(x)+dy,
                         alpha=0.2, color="gray")
```



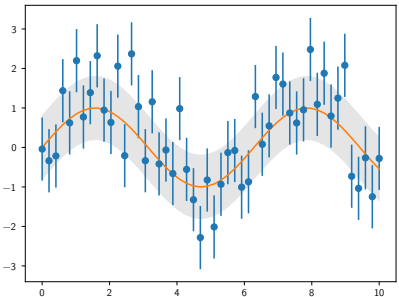
Barres d'erreur

```
In [1]: x = np.linspace(0, 10, 50)
In [2]: dy = 0.8
In [3]: y = np.sin(x) + dy * np.random.randn(50)

In [4]: plt.errorbar(x, y, yerr=dy, fmt="o")
In [5]: plt.plot(x, np.sin(x))

In [6]: plt.errorbar(x, y, yerr=dy,
                     fmt="o", color="black",
                     ecolor="lightgray",
                     elinewidth=3,
                     capsize=0)

In [7]: plt.fill_between(x, np.sin(x)-dy, np.sin(x)+dy,
                         alpha=0.2, color="gray")
```



Barres d'erreur

```
In [1]: x = np.linspace(0, 10, 50)
In [2]: dy = 0.8
In [3]: y = np.sin(x) + dy * np.random.randn(50)

In [4]: plt.errorbar(x, y, yerr=dy, fmt="o")
In [5]: plt.plot(x, np.sin(x))

In [6]: plt.errorbar(x, y, yerr=dy,
                     fmt="o", color="black",
                     ecolor="lightgray",
                     elinewidth=3,
                     capsize=0)

In [7]: plt.fill_between(x, np.sin(x)-dy, np.sin(x)+dy,
                         alpha=0.2, color="gray")
```

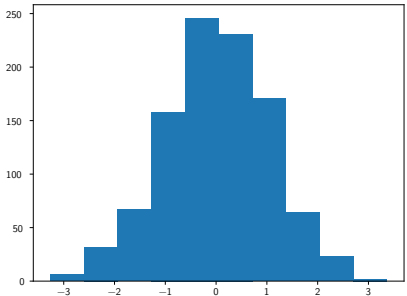


Histogramme 1D

```
In [1]: data = np.random.randn(1000)
In [2]: plt.hist(data)
```

```
In [3]: plt.hist(data, bins=30, normed=True)
```

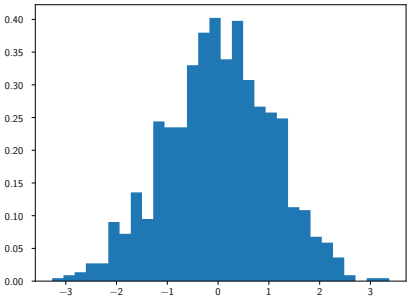
- Pour découvrir l'ensemble des options d'affichage **`plt.hist?`** ou **`help(plt.hist)`**



Histogramme 1D

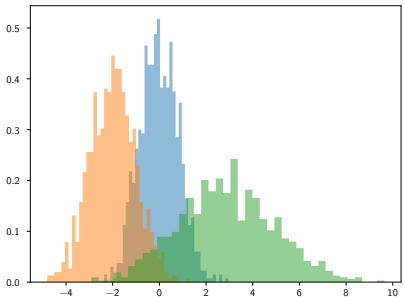
```
In [1]: data = np.random.randn(1000)
In [2]: plt.hist(data)
In [3]: plt.hist(data, bins=30, normed=True)
```

- Pour découvrir l'ensemble des options d'affichage **`plt.hist?`** ou **`help(plt.hist)`**



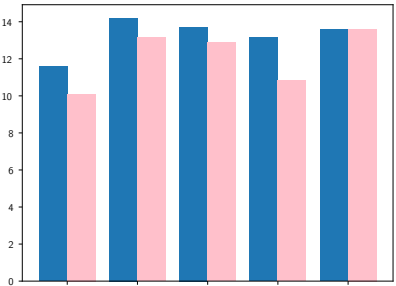
Histogramme 1D

```
In [0]: x1 = np.random.normal(0, 0.8, 1000)
In [1]: x2 = np.random.normal(-2, 1, 1000)
In [2]: x3 = np.random.normal(3, 2, 1000)

In [3]: kwargs = dict(histtype="stepfilled", alpha=0.5,
                      normed=True, bins=40)

In [4]: plt.hist(x1, **kwargs)
In [5]: plt.hist(x2, **kwargs)
In [6]: plt.hist(x3, **kwargs);
```



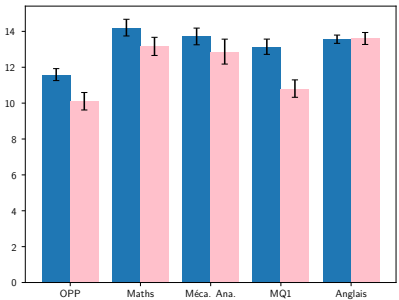
Histogramme 1D

```
In [1]: data = np.loadtxt("data/pv_2016_2017.tsv")

In [2]: men_mask   = (data[:,-1] == 0)
In [3]: women_mask = (data[:,-1] == 1)

In [4]: men_means   = np.mean(data[men_mask], axis=0)
In [5]: women_means = np.mean(data[women_mask], axis=0)

In [6]: dx = 0.4
In [7]: x = np.arange(5)
In [8]: plt.bar(x-dx/2, men_means[:-1], dx)
In [9]: plt.bar(x+dx/2, women_means[:-1], dx, color="pink")
In[10]: plt.xticks(x,
        ["OPP", "Maths", "Méca. Ana.", "MQ1", "Anglais"])
```



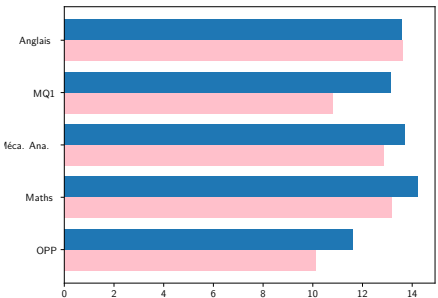
Histogramme 1D

```
In [1]: data = np.loadtxt("data/pv_2016_2017.tsv")

In [2]: men_mask   = (data[:,-1] == 0)
In [3]: women_mask = (data[:,-1] == 1)

In [4]: men_means   = np.mean(data[men_mask], axis=0)
In [5]: women_means = np.mean(data[women_mask], axis=0)

In [6]: men_errs   = np.std(data[men_mask], axis=0) \
                     /np.sqrt(np.sum(men_mask))
In [7]: women_errs = np.std(data[women_mask], axis=0) \
                     /np.sqrt(np.sum(women_mask))

In [8]: dx = 0.4
In [9]: x = np.arange(5)
In[10]: plt.bar(x-dx/2, men_means[:-1], dx,
                yerr=men_errs[:-1])
In[11]: plt.bar(x+dx/2, women_means[:-1], dx, color="pink",
                yerr=women_errs[:-1])
In[12]: plt.xticks(x,
        ["OPP", "Maths", "Méca. Ana.", "MQ1", "Anglais"])
```



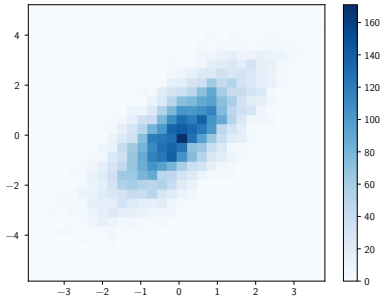
Histogramme 1D

```
In [1]: data = np.loadtxt("data/pv_2016_2017.tsv")

In [2]: men_mask   = (data[:,-1] == 0)
In [3]: women_mask = (data[:,-1] == 1)

In [4]: men_means   = np.mean(data[men_mask], axis=0)
In [5]: women_means = np.mean(data[women_mask], axis=0)

In [6]: dx = 0.4
In [7]: x = np.arange(5)
In [8]: plt.barh(x-dx/2, men_means[:-1], dx)
In [9]: plt.barh(x+dx/2, women_means[:-1], dx, color="pink")
In[10]: plt.yticks(x,
        ["OPP", "Maths", "Méca. Ana.", "MQ1", "Anglais"])
```



Histogramme 2D

```
In [1]: mean = [0, 0]
In [2]: cov = [[1, 1], [1, 2]]
In [3]: x, y = np.random.multivariate_normal(mean, cov, 10000).T

In [4]: plt.hist2d(x, y, bins=30, cmap="Blues")
In [5]: plt.colorbar()
```



Contours & densités

$$z = f(x, y) = \sin^{10} x + \cos(x \cdot y) \cdot \cos x$$
$$= \sin^{10} \begin{bmatrix} x_0 & \cdots \end{bmatrix} + \cos \left(\begin{bmatrix} x_0 & \cdots \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ \vdots \end{bmatrix} \right) \cdot \cos \begin{bmatrix} x_0 & \cdots \end{bmatrix}$$

```
In [1]: def f(x, y):
   ...:     return np.sin(x)**10 + np.cos(x*y) * np.cos(x)

In [2]: x = np.linspace(0, 5, 500)
In [3]: y = np.linspace(0, 5, 500)

In [4]: X, Y = np.meshgrid(x, y)
In [5]: Z = f(X, Y)

In [6]: contours = plt.contour(X, Y, Z, 3, colors="black")
In [7]: plt.clabel(contours, inline=True, fontsize=8)

In [8]: plt.imshow(Z, extent=[0, 5, 0, 5], origin="lower",
                   cmap="RdGy", alpha=0.5)
In [9]: plt.colorbar();
```

$$z = f(x, y) = \sin^{10} x + \cos(x \cdot y) \cdot \cos x$$

$$= \sin^{10} \begin{bmatrix} x_0 & \cdots \end{bmatrix} + \cos \left(\begin{bmatrix} x_0 & \cdots \end{bmatrix} \cdot \begin{bmatrix} y_0 \\ \vdots \end{bmatrix} \right) \cdot \cos \begin{bmatrix} x_0 & \cdots \end{bmatrix}$$

```
In [1]: def f(x, y):
   ...:     return np.sin(x)**10 + np.cos(x*y) * np.cos(x)

In [2]: x = np.linspace(0, 5, 500)
In [3]: y = np.linspace(0, 5, 500)

In [4]: X, Y = np.meshgrid(x, y)
In [5]: Z = f(X, Y)

In [6]: contours = plt.contour(X, Y, Z, 3, colors="black")
In [7]: plt.clabel(contours, inline=True, fontsize=8)

In [8]: plt.imshow(Z, extent=[0, 5, 0, 5], origin="lower",
                   cmap="RdGy", alpha=0.5)
In [9]: plt.colorbar();
```

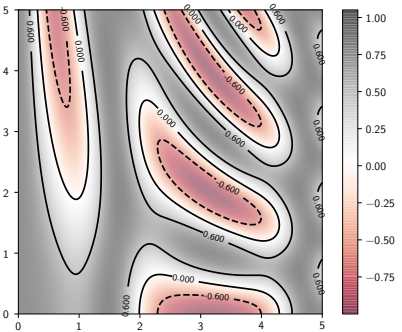


Image 2D

```
In [1]: img = plt.imread("./data/puzo_patrick.png")
In [2]: plt.imshow(img)
In [3]: plt.title("P. Puzo après le partiel d'EM")
In [4]: plt.scatter(725, 1000, c="red", s=1000)
In [5]: plt.title("P. Puzo après son anniversaire")
```

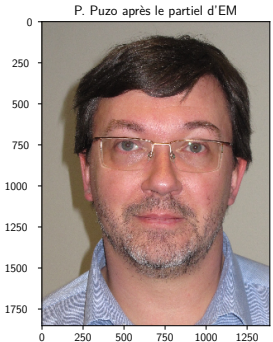


Image 2D

```
In [1]: img = plt.imread("./data/puzo_patrick.png")
In [2]: plt.imshow(img)
In [3]: plt.title("P. Puzo après le partiel d'EM")
In [4]: plt.scatter(725, 1000, c="red", s=1000)
In [5]: plt.title("P. Puzo après son anniversaire")
```

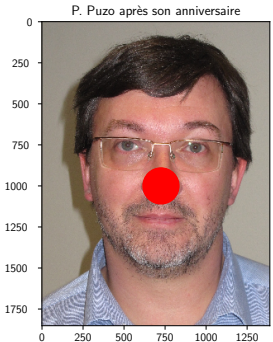


Figure 3D

- La représentation 3D suppose le chargement de l'outil **mplot3d** inclus par défaut dans `matplotlib`

```
In [1]: from mpl_toolkits import mplot3d
```

- Une vue 3D est initialisée en spécifiant le type de projection

```
In [2]: ax = plt.axes(projection="3d")
```

Figure 3D

- La représentation 3D suppose le chargement de l'outil **mplot3d** inclus par défaut dans `matplotlib`

```
In [1]: from mpl_toolkits import mplot3d
```

- Une vue 3D est initialisée en spécifiant le type de projection

```
In [2]: ax = plt.axes(projection="3d")
```

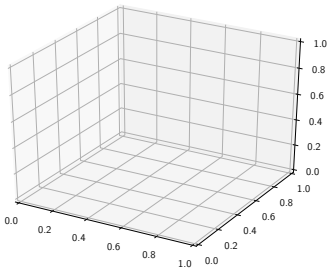


Figure 3D

👉 plot3D & scatter3D

```
In [2]: ax = plt.axes(projection="3d")

In [3]: # Data for a three-dimensional line
In [4]: zline = np.linspace(0, 15, 1000)
In [5]: xline = np.sin(zline)
In [6]: yline = np.cos(zline)
In [7]: ax.plot3D(xline, yline, zline, "gray")
In [8]: # Data for three-dimensional scattered points
In [9]: zdata = 15 * np.random.random(100)
In[10]: xdata = np.sin(zdata) + 0.1*np.random.randn(100)
In[11]: ydata = np.cos(zdata) + 0.1*np.random.randn(100)
In[12]: ax.scatter3D(xdata, ydata, zdata, c=zdata)
```

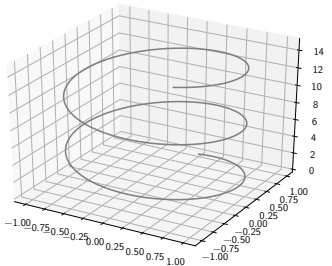


Figure 3D

plot3D & scatter3D

```
In [2]: ax = plt.axes(projection="3d")

In [3]: # Data for a three-dimensional line
In [4]: zline = np.linspace(0, 15, 1000)
In [5]: xline = np.sin(zline)
In [6]: yline = np.cos(zline)
In [7]: ax.plot3D(xline, yline, zline, "gray")

In [8]: # Data for three-dimensional scattered points
In [9]: zdata = 15 * np.random.random(100)
In[10]: xdata = np.sin(zdata) + 0.1*np.random.randn(100)
In[11]: ydata = np.cos(zdata) + 0.1*np.random.randn(100)
In[12]: ax.scatter3D(xdata, ydata, zdata, c=zdata)
```

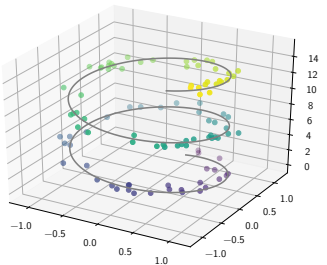


Figure 3D

plot_wireframe & plot_surface

- $f(x, y) = \sin(\sqrt{x^2 + y^2})$

```
In [2]: ax = plt.axes(projection="3d")
In [3]: def f(x, y):
   ...:     return np.sin(np.sqrt(x**2 + y**2))

In [4]: x = np.linspace(-6, 6, 30)
In [5]: y = np.linspace(-6, 6, 30)

In [6]: X, Y = np.meshgrid(x, y)
In [7]: Z = f(X, Y)
In [8]: ax.plot_wireframe(X, Y, Z, linewidth=0.5
                          color="gray")
In [9]: ax.plot_surface(X, Y, Z, cmap="viridis")
```

Figure 3D

plot_wireframe & plot_surface

- $f(x, y) = \sin(\sqrt{x^2 + y^2})$

```
In [2]: ax = plt.axes(projection="3d")
In [3]: def f(x, y):
   ...:     return np.sin(np.sqrt(x**2 + y**2))

In [4]: x = np.linspace(-6, 6, 30)
In [5]: y = np.linspace(-6, 6, 30)

In [6]: X, Y = np.meshgrid(x, y)
In [7]: Z = f(X, Y)

In [8]: ax.plot_wireframe(X, Y, Z, linewidth=0.5
                          color="gray")
In [9]: ax.plot_surface(X, Y, Z, cmap="viridis")
```

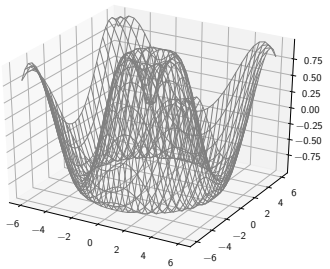


Figure 3D

plot_wireframe & plot_surface

- $f(x, y) = \sin(\sqrt{x^2 + y^2})$

```
In [2]: ax = plt.axes(projection="3d")
In [3]: def f(x, y):
   ...:     return np.sin(np.sqrt(x**2 + y**2))

In [4]: x = np.linspace(-6, 6, 30)
In [5]: y = np.linspace(-6, 6, 30)

In [6]: X, Y = np.meshgrid(x, y)
In [7]: Z = f(X, Y)

In [8]: ax.plot_wireframe(X, Y, Z, linewidth=0.5
                          color="gray")

In [9]: ax.plot_surface(X, Y, Z, cmap="viridis")
```

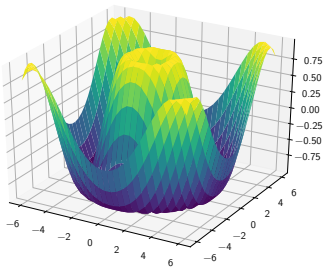


Figure 3D

plot_wireframe & plot_surface

- $f(x, y) = \sin(\sqrt{x^2 + y^2})$

```
In [2]: ax = plt.axes(projection="3d")
In [3]: def f(x, y):
   ...:     return np.sin(np.sqrt(x**2 + y**2))

In [4]: x = np.linspace(-6, 6, 30)
In [5]: y = np.linspace(-6, 6, 30)

In [6]: X, Y = np.meshgrid(x, y)
In [7]: Z = f(X, Y)
In [9]: ax.plot_surface(X, Y, Z, alpha=0.25,
                        edgecolor="k", linewidth=0.1)
```

```
In[10]: ax.contour(X, Y, Z, zdir="z", offset=+1)
In[11]: ax.contour(X, Y, Z, zdir="y", offset=-7)
In[12]: ax.contour(X, Y, Z, zdir="x", offset=+7)
In[13]: ax.set_zlim3d(-1, 1)
In[14]: ax.set_ylim3d(-7, 7)
In[15]: ax.set_xlim3d(-7, 7)
```

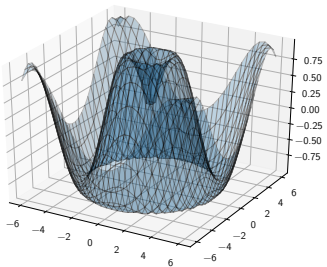


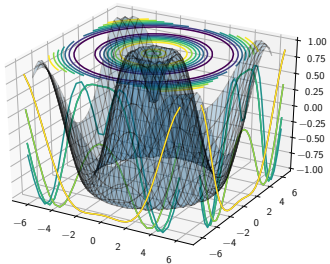
Figure 3D

plot_wireframe & plot_surface

- $f(x, y) = \sin(\sqrt{x^2 + y^2})$

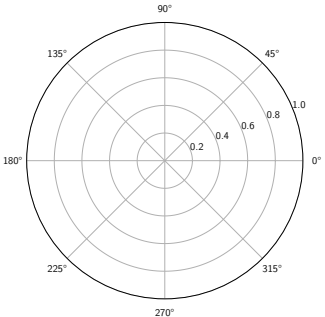
```
In [2]: ax = plt.axes(projection="3d")
In [3]: def f(x, y):
   ...:     return np.sin(np.sqrt(x**2 + y**2))

In [4]: x = np.linspace(-6, 6, 30)
In [5]: y = np.linspace(-6, 6, 30)

In [6]: X, Y = np.meshgrid(x, y)
In [7]: Z = f(X, Y)
In [9]: ax.plot_surface(X, Y, Z, alpha=0.25,
                        edgecolor="k", linewidth=0.1)
```

```
In[10]: ax.contour(X, Y, Z, zdir="z", offset=+1)
In[11]: ax.contour(X, Y, Z, zdir="y", offset=-7)
In[12]: ax.contour(X, Y, Z, zdir="x", offset=+7)
In[13]: ax.set_zlim3d(-1, 1)
In[14]: ax.set_ylim3d(-7, 7)
In[15]: ax.set_xlim3d(-7, 7)
```



Autres modes de représentation

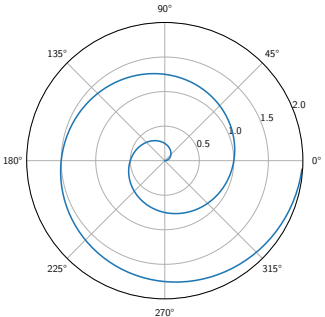
Polar & Pie charts




```
In [2]: ax = plt.axes(projection="polar")
In [3]: r = np.arange(0, 2, 0.01)
In [4]: theta = 2 * np.pi * r

In [5]: ax.plot(theta, r)
In [6]: ax.set_rmax(2)
In [7]: ax.set_rticks([0.5, 1, 1.5, 2])
```

Autres modes de représentation

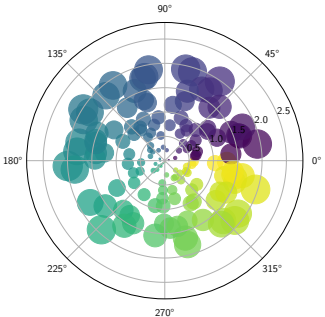
Polar & Pie charts




```
In [2]: ax = plt.axes(projection="polar")
In [3]: r = np.arange(0, 2, 0.01)
In [4]: theta = 2 * np.pi * r

In [5]: ax.plot(theta, r)
In [6]: ax.set_rmax(2)
In [7]: ax.set_rticks([0.5, 1, 1.5, 2])
```

Autres modes de représentation

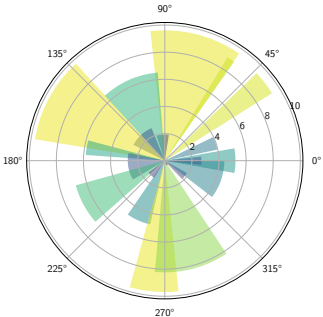
Polar & Pie charts




```
In [2]: ax = plt.axes(projection="polar")
In [3]: N = 150
In [4]: r = 2 * np.random.rand(N)
In [5]: theta = 2 * np.pi * np.random.rand(N)
In [6]: area = 200 * r**2

In [7]: ax.scatter(theta, r, c=theta, s=area, alpha=0.75)
```

Autres modes de représentation

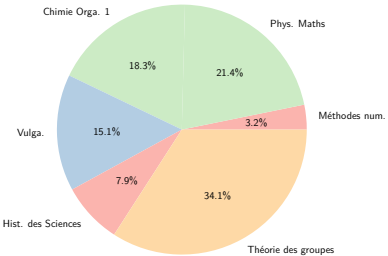
Polar & Pie charts




```
In [2]: ax = plt.axes(projection="polar")
In [3]: N = 20
In [4]: theta = np.linspace(0.0, 2 * np.pi, N)
In [5]: radii = 10 * np.random.rand(N)
In [6]: width = np.pi / 4 * np.random.rand(N)

In [7]: bars = ax.bar(theta, radii, width=width)

In [8]: for r, bar in zip(radii, bars):
   ...:     bar.set_facecolor(plt.cm.viridis(r/10))
   ...:     bar.set_alpha(0.5)
```

Autres modes de représentation

Polar & Pie charts

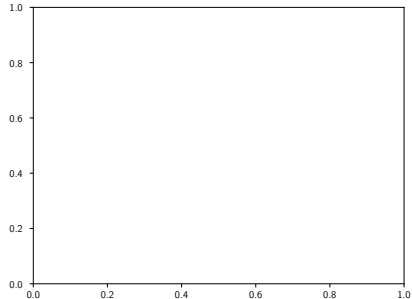



```
In [2]: labels = "Méthodes num.", "Phys. Maths", \
                 "Chimie Orga. 1", "Vulga.",     \
                 "Hist. des Sciences", "Théorie des groupes"
In [3]: percent = np.array([4.2, 28.1, 24.0, 19.8, 10.4, 44.8])

In [4]: plt.pie(percent, labels=labels, autopct="%1.1f%%",
                colors=plt.cm.Pastel1(percent))
In [5]: plt.axis("equal")
```

- `matplotlib` permet une gestion relativement aisée du placement des figures et de leurs sous-figures

```
In [2]: ax1 = plt.axes()
In [3]: ax2 = plt.axes([0.65, 0.65, 0.2, 0.2])
```

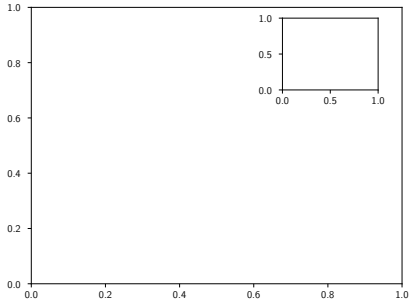
avec `axes([x, y, w, h])` et **x,y,w,h exprimés en fraction du canevas initial**



- `matplotlib` permet une gestion relativement aisée du placement des figures et de leurs sous-figures

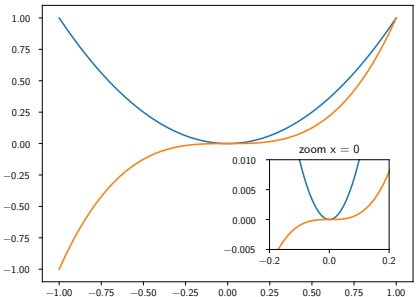
```
In [2]: ax1 = plt.axes()
In [3]: ax2 = plt.axes([0.65, 0.65, 0.2, 0.2])
```

avec `axes([x, y, w, h])` et **`x,y,w,h` exprimés en fraction du canevas initial**



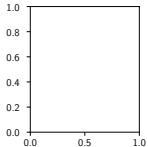
- `matplotlib` permet une gestion relativement aisée du placement des figures et de leurs sous-figures

```
In [2]: x = np.linspace(-1, 1, 1000)
In [3]: plt.plot(x, x**2, x, x**3)

In [4]: inset = plt.axes([0.6, 0.2, 0.25, 0.25])

In [5]: inset.plot(x, x**2, x, x**3)
In [6]: inset.set_title("zoom x = 0")
In [7]: inset.set_xlim(-0.2, +0.2)
In [8]: inset.set_ylim(-0.005, +0.01)
```



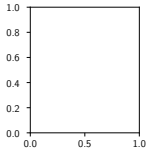
- La commande **`subplot`** permet la génération **sous-figure par sous-figure** selon une représentation matricielle

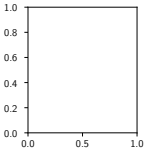
```
In [2]: plt.subplot(2, 3, 1)
In [3]: plt.subplot(2, 3, 3)
In [4]: plt.subplot(2, 3, 5)
```



- La commande **`subplot`** permet la génération **sous-figure par sous-figure** selon une représentation matricielle

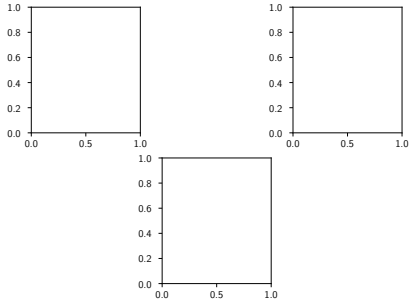
```
In [2]: plt.subplot(2, 3, 1)
In [3]: plt.subplot(2, 3, 3)
In [4]: plt.subplot(2, 3, 5)
```





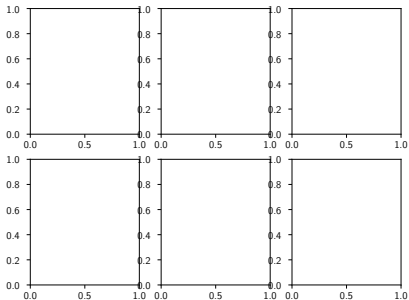
- La commande **`subplot`** permet la génération **sous-figure par sous-figure** selon une représentation matricielle

```
In [2]: plt.subplot(2, 3, 1)
In [3]: plt.subplot(2, 3, 3)
In [4]: plt.subplot(2, 3, 5)
```



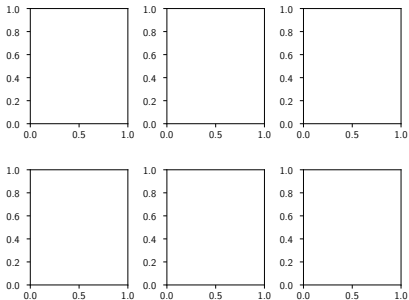
- La commande **`subplots`** permet la génération de **l'ensemble des sous-figures** selon une représentation matricielle

```
In [2]: plt.subplots(2, 3)
In [3]: plt.subplots_adjust(hspace=0.4, wspace=0.4)
In [4]: plt.subplots(2, 3, sharex="col", sharey="row")
In [5]: fig, ax = plt.subplots(2, 3, sharex="col",
                               sharey="row")
In [6]: for i in range(2):
   ...:     for j in range(3):
   ...:         ax[i, j].text(0.5, 0.5, str((i, j)),
   ...:                       fontsize=18, ha="center")
```



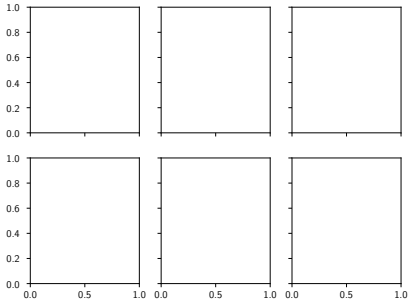
- La commande **`subplots`** permet la génération de **l'ensemble des sous-figures** selon une représentation matricielle

```
In [2]: plt.subplots(2, 3)
In [3]: plt.subplots_adjust(hspace=0.4, wspace=0.4)
In [4]: plt.subplots(2, 3, sharex="col", sharey="row")
In [5]: fig, ax = plt.subplots(2, 3, sharex="col",
                               sharey="row")
In [6]: for i in range(2):
   ...:     for j in range(3):
   ...:         ax[i, j].text(0.5, 0.5, str((i, j)),
   ...:                       fontsize=18, ha="center")
```



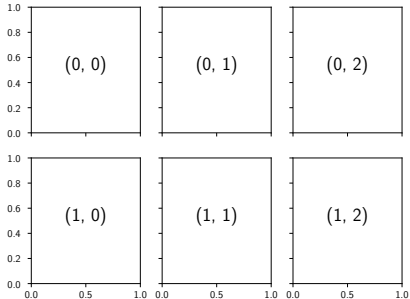
- La commande **`subplots`** permet la génération de **l'ensemble des sous-figures** selon une représentation matricielle

```
In [2]: plt.subplots(2, 3)
In [3]: plt.subplots_adjust(hspace=0.4, wspace=0.4)
In [4]: plt.subplots(2, 3, sharex="col", sharey="row")
In [5]: fig, ax = plt.subplots(2, 3, sharex="col",
                               sharey="row")
In [6]: for i in range(2):
   ...:     for j in range(3):
   ...:         ax[i, j].text(0.5, 0.5, str((i, j)),
   ...:                       fontsize=18, ha="center")
```



- La commande **`subplots`** permet la génération de **l'ensemble des sous-figures** selon une représentation matricielle

```
In [2]: plt.subplots(2, 3)
In [3]: plt.subplots_adjust(hspace=0.4, wspace=0.4)
In [4]: plt.subplots(2, 3, sharex="col", sharey="row")
In [5]: fig, ax = plt.subplots(2, 3, sharex="col",
                               sharey="row")
In [6]: for i in range(2):
   ...:     for j in range(3):
   ...:         ax[i, j].text(0.5, 0.5, str((i, j)),
   ...:                       fontsize=18, ha="center")
```

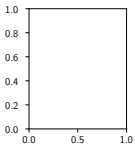


- La commande **GridSpec** ne génère pas de figures ou sous figures mais facilite la gestion et notamment la fusion d'espaces réservés aux sous-figures

```
In [2]: grid = plt.GridSpec(2, 3, hspace=0.4, wspace=0.4)
In [3]: plt.subplot(grid[0, 0])
In [4]: plt.subplot(grid[0, 1:])
In [5]: plt.subplot(grid[1, :2])
In [6]: plt.subplot(grid[1, 2])
```

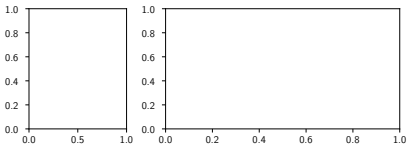
- La commande **GridSpec** ne génère pas de figures ou sous figures mais facilite la gestion et notamment la fusion d'espaces réservés aux sous-figures

```
In [2]: grid = plt.GridSpec(2, 3, hspace=0.4, wspace=0.4)
In [3]: plt.subplot(grid[0, 0])
In [4]: plt.subplot(grid[0, 1:])
In [5]: plt.subplot(grid[1, :2])
In [6]: plt.subplot(grid[1, 2])
```



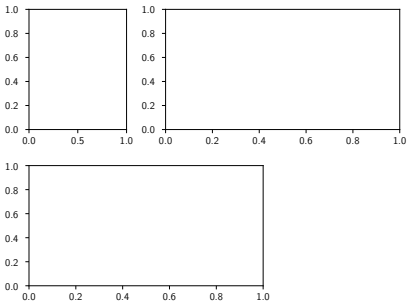
- La commande **GridSpec** ne génère pas de figures ou sous figures mais facilite la gestion et notamment la fusion d'espaces réservés aux sous-figures

```
In [2]: grid = plt.GridSpec(2, 3, hspace=0.4, wspace=0.4)
In [3]: plt.subplot(grid[0, 0])
In [4]: plt.subplot(grid[0, 1:])
In [5]: plt.subplot(grid[1, :2])
In [6]: plt.subplot(grid[1, 2])
```



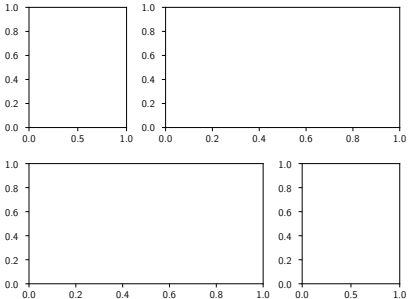
- La commande **`GridSpec`** ne génère pas de figures ou sous figures mais facilite la gestion et notamment la fusion d'espaces réservés aux sous-figures

```
In [2]: grid = plt.GridSpec(2, 3, hspace=0.4, wspace=0.4)
In [3]: plt.subplot(grid[0, 0])
In [4]: plt.subplot(grid[0, 1:])
In [5]: plt.subplot(grid[1, :2])
In [6]: plt.subplot(grid[1, 2])
```



- La commande **`GridSpec`** ne génère pas de figures ou sous figures mais facilite la gestion et notamment la fusion d'espaces réservés aux sous-figures

```
In [2]: grid = plt.GridSpec(2, 3, hspace=0.4, wspace=0.4)
In [3]: plt.subplot(grid[0, 0])
In [4]: plt.subplot(grid[0, 1:])
In [5]: plt.subplot(grid[1, :2])
In [6]: plt.subplot(grid[1, 2])
```

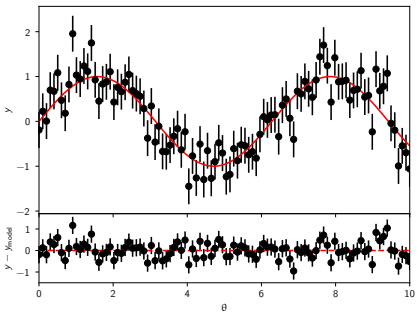


Subplot

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

x = np.linspace(0, 10, 100)
dy = 0.4
y = np.sin(x) + dy * np.random.randn(100)

grid = plt.GridSpec(4, 1, hspace=0, wspace=0)

main = plt.subplot(grid[0:3], xticklabels=[], xlim=[0, 10])
main.plot(x, np.sin(x), "r")
main.errorbar(x, y, yerr=dy, fmt="ok")
main.set_ylabel(r"$y$")

dev = plt.subplot(grid[3], xlim=[0, 10])
dev.errorbar(x, y - np.sin(x), yerr=dy, fmt="ok")
dev.plot([0, 10], [0, 0], "--r")
dev.set_ylabel(r"$y-y_\mathrm{model}$")
dev.set_xlabel(r"$\theta$")

plt.show()
```

Subplot

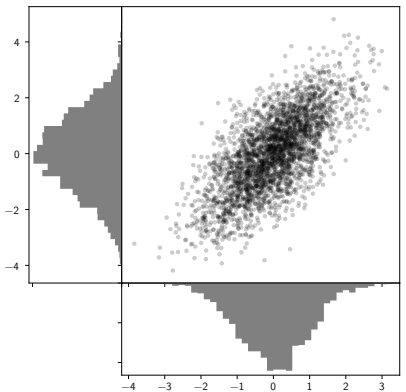
```python
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
dy = 0.4
y = np.sin(x) + dy * np.random.randn(100)

grid = plt.GridSpec(4, 1, hspace=0, wspace=0)

main = plt.subplot(grid[0:3], xticklabels=[], xlim=[0, 10])
main.plot(x, np.sin(x), "r")
main.errorbar(x, y, yerr=dy, fmt="ok")
main.set_ylabel(r"$y$")

dev = plt.subplot(grid[3], xlim=[0, 10])
dev.errorbar(x, y - np.sin(x), yerr=dy, fmt="ok")
dev.plot([0, 10], [0, 0], "--r")
dev.set_ylabel(r"$y-y_\mathrm{model}$")
dev.set_xlabel(r"$\theta$")

plt.show()
```

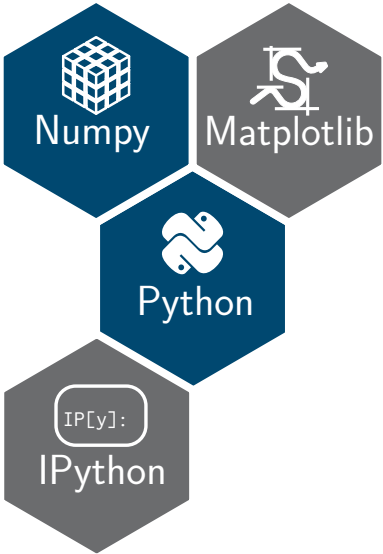


```
import numpy as np
import matplotlib.pyplot as plt

# Create some normally distributed data
mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 3000).T

# Set up the axes with gridspec
plt.figure(figsize=(6, 6))
grid = plt.GridSpec(4, 4, hspace=0, wspace=0)
main_ax = plt.subplot(grid[:-1, 1:], xticklabels=[],
                      yticklabels=[])
y_hist = plt.subplot(grid[:-1, 0], xticklabels=[])
x_hist = plt.subplot(grid[-1, 1:], yticklabels=[])

# scatter points on the main axes
main_ax.plot(x, y, "ok", markersize=3, alpha=0.2)

# histogram on the attached axes
x_hist.hist(x, 40, histtype="stepfilled",
            orientation="vertical", color="gray")
x_hist.invert_yaxis()

y_hist.hist(y, 40, histtype="stepfilled",
            orientation="horizontal", color="gray")
y_hist.invert_xaxis()

plt.show()
```

Subplot

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

# Create some normally distributed data
mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 3000).T

# Set up the axes with gridspec
plt.figure(figsize=(6, 6))
grid = plt.GridSpec(4, 4, hspace=0, wspace=0)
main_ax = plt.subplot(grid[:-1, 1:], xticklabels=[],
                      yticklabels=[])
y_hist = plt.subplot(grid[:-1, 0], xticklabels=[])
x_hist = plt.subplot(grid[-1, 1:], yticklabels=[])

# scatter points on the main axes
main_ax.plot(x, y, "ok", markersize=3, alpha=0.2)

# histogram on the attached axes
x_hist.hist(x, 40, histtype="stepfilled",
            orientation="vertical", color="gray")
x_hist.invert_yaxis()

y_hist.hist(y, 40, histtype="stepfilled",
            orientation="horizontal", color="gray")
y_hist.invert_xaxis()

plt.show()
```





Option « Programmation en Python »

Épisode 3 : Interaction avec `matplotlib`

Interface graphique sous `matplotlib`

- `matplotlib` propose une interface relativement rudimentaire pour interagir avec les objets graphiques
- Parmi les *widgets* ou objets de contrôle
  - `Cursor`/`MultiCursor` permet l'affichage des valeurs dans la barre d'état
  - `Slider` permet la variation d'une quantité numérique
  - `Button` de générer une action définie par l'utilisateur lors du clic souris
  - `CheckButtons`/`RadioButtons` permet l'activation/désactivation de fonctionnalités graphiques
  - `Selector`, `Menu`, ...
- Importation des *widgets*

```
from matplotlib.widgets import Cursor, Slider, Button
```

Interface graphique sous `matplotlib`

- `matplotlib` propose une interface relativement rudimentaire pour interagir avec les objets graphiques
- Parmi les *widgets* ou objets de contrôle
  - **`Cursor/MultiCursor`** permet l'affichage des valeurs dans la barre d'état
  - **`Slider`** permet la variation d'une quantité numérique
  - **`Button`** de générer une action définie par l'utilisateur lors du clic souris
  - **`CheckButtons/RadioButtons`** permet l'activation/désactivation de fonctionnalités graphiques
  - `Selector, Menu,…`
- Importation des *widgets*

```
from matplotlib.widgets import Cursor, Slider, Button
```

Interface graphique sous `matplotlib`

- `matplotlib` propose une interface relativement rudimentaire pour interagir avec les objets graphiques
- Parmi les *widgets* ou objets de contrôle
  - **`Cursor/MultiCursor`** permet l'affichage des valeurs dans la barre d'état
  - **`Slider`** permet la variation d'une quantité numérique
  - **`Button`** de générer une action définie par l'utilisateur lors du clic souris
  - **`CheckButtons/RadioButtons`** permet l'activation/désactivation de fonctionnalités graphiques
  - `Selector, Menu,…`
- Importation des *widgets*

```
from matplotlib.widgets import Cursor, Slider, Button
```

Interface graphique sous `matplotlib`

- `matplotlib` propose une interface relativement rudimentaire pour interagir avec les objets graphiques
- Parmi les *widgets* ou objets de contrôle
  - **`Cursor/MultiCursor`** permet l'affichage des valeurs dans la barre d'état
  - **`Slider`** permet la variation d'une quantité numérique
  - **`Button`** de générer une action définie par l'utilisateur lors du clic souris
  - **`CheckButtons/RadioButtons`** permet l'activation/désactivation de fonctionnalités graphiques
  - `Selector, Menu,…`
- Importation des *widgets*

```
from matplotlib.widgets import Cursor, Slider, Button
```

Interface graphique sous `matplotlib`

- `matplotlib` propose une interface relativement rudimentaire pour interagir avec les objets graphiques
- Parmi les *widgets* ou objets de contrôle
  - **`Cursor/MultiCursor`** permet l'affichage des valeurs dans la barre d'état
  - **`Slider`** permet la variation d'une quantité numérique
  - **`Button`** de générer une action définie par l'utilisateur lors du clic souris
  - **`CheckButtons/RadioButtons`** permet l'activation/désactivation de fonctionnalités graphiques
  - `Selector, Menu,…`
- Importation des *widgets*

```
from matplotlib.widgets import Cursor, Slider, Button
```

Interface graphique sous `matplotlib`

- `matplotlib` propose une interface relativement rudimentaire pour interagir avec les objets graphiques
- Parmi les *widgets* ou objets de contrôle
  - **`Cursor/MultiCursor`** permet l'affichage des valeurs dans la barre d'état
  - **`Slider`** permet la variation d'une quantité numérique
  - **`Button`** de générer une action définie par l'utilisateur lors du clic souris
  - **`CheckButtons/RadioButtons`** permet l'activation/désactivation de fonctionnalités graphiques
  - `Selector, Menu,…`
- Importation des *widgets*

```
from matplotlib.widgets import Cursor, Slider, Button
```

Interface graphique sous `matplotlib`

- `matplotlib` propose une interface relativement rudimentaire pour interagir avec les objets graphiques
- Parmi les *widgets* ou objets de contrôle
  - **`Cursor/MultiCursor`** permet l'affichage des valeurs dans la barre d'état
  - **`Slider`** permet la variation d'une quantité numérique
  - **`Button`** de générer une action définie par l'utilisateur lors du clic souris
  - **`CheckButtons/RadioButtons`** permet l'activation/désactivation de fonctionnalités graphiques
  - `Selector`, `Menu`,...
- Importation des *widgets*

```
from matplotlib.widgets import Cursor, Slider, Button
```

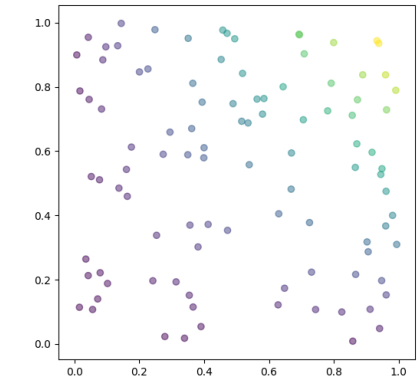
Interface graphique sous `matplotlib`

- `matplotlib` propose une interface relativement rudimentaire pour interagir avec les objets graphiques
- Parmi les *widgets* ou objets de contrôle
  - **`Cursor/MultiCursor`** permet l'affichage des valeurs dans la barre d'état
  - **`Slider`** permet la variation d'une quantité numérique
  - **`Button`** de générer une action définie par l'utilisateur lors du clic souris
  - **`CheckButtons/RadioButtons`** permet l'activation/désactivation de fonctionnalités graphiques
  - `Selector`, `Menu`,...
- Importation des *widgets*

```
from matplotlib.widgets import Cursor, Slider, Button
```

Interface graphique sous `matplotlib`

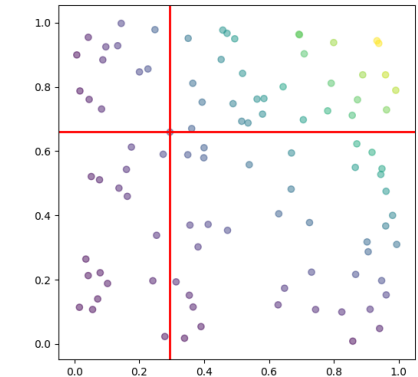
Utilisation de curseurs



```
In [1]: import numpy as np
In [2]: x, y = np.random.rand(2, 100)

In [3]: import matplotlib.pyplot as plt
In [4]: plt.figure(figsize=(6,6))
In [5]: plt.scatter(x, y, c=x*y, alpha=0.5)

In [6]: from matplotlib.widgets import Cursor
In [7]: cursor = Cursor(plt.gca(), color="red", lw=2)
```

Interface graphique sous `matplotlib`

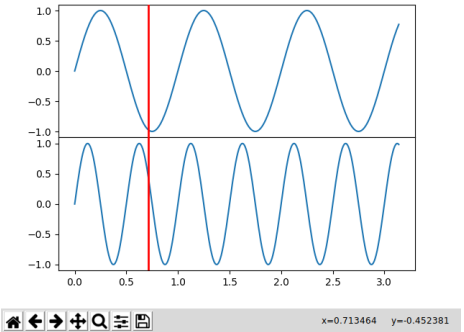
Utilisation de curseurs




```
In [1]: import numpy as np
In [2]: x, y = np.random.rand(2, 100)

In [3]: import matplotlib.pyplot as plt
In [4]: plt.figure(figsize=(6,6))
In [5]: plt.scatter(x, y, c=x*y, alpha=0.5)

In [6]: from matplotlib.widgets import Cursor
In [7]: cursor = Cursor(plt.gca(), color="red", lw=2)
```

Interface graphique sous `matplotlib`

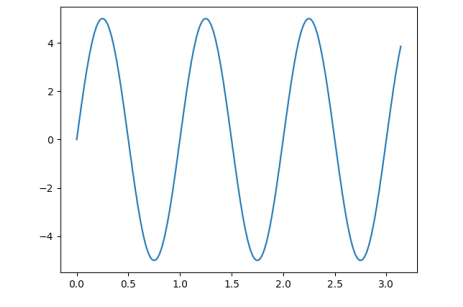
Utilisation de curseurs




```
In [1]: import numpy as np
In [2]: t = np.arange(0.0, np.pi, 0.01)

In [3]: import matplotlib.pyplot as plt
In [4]: fig, ax = plt.subplots(2, 1, sharex="col")
In [5]: plt.subplots_adjust(hspace=0)
In [6]: ax[0].plot(t, np.sin(2*np.pi*t))
In [7]: ax[1].plot(t, np.sin(4*np.pi*t))

In [8]: from matplotlib.widgets import MultiCursor
In [9]: multi = MultiCursor(fig.canvas, (ax[0], ax[1]),
   ...:                     color="red", lw=2)
```

Interface graphique sous matplotlib

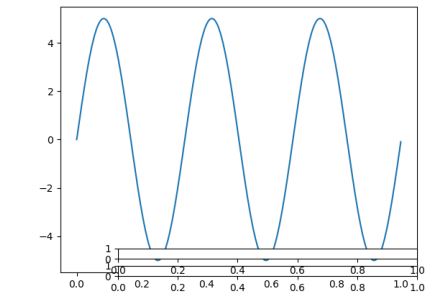
Utilisation de sliders




```
In [1]: import numpy as np
In [2]: t = np.arange(0.0, np.pi, 0.01)

In [3]: a0, f0 = 5, 1
In [4]: def signal(t, a=a0, f=f0):
   ...:     return a*np.sin(2*np.pi*f*t)

In [5]: import matplotlib.pyplot as plt
In [6]: fig, ax = plt.subplots()
In [7]: l, = plt.plot(t, signal(t))
In [8]: axfreq = plt.axes([0.25, 0.10, 0.65, 0.03])
In [9]: axamp  = plt.axes([0.25, 0.15, 0.65, 0.03])
In[10]: plt.subplots_adjust(bottom=0.25)
In[11]: from matplotlib.widgets import Slider
In[12]: sfreq = Slider(axfreq, "Fréquence", 0.1, 30.0,
                       valinit=f0)
In[13]: samp = Slider(axamp, "Amplitude", 0.1, 10.0,
                      valinit=a0)
```

Interface graphique sous matplotlib

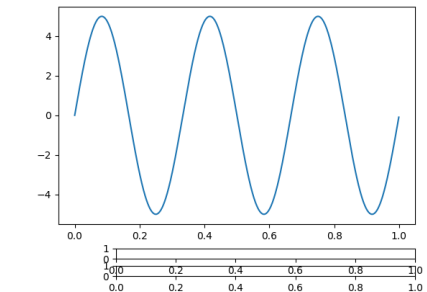
Utilisation de sliders



```
In [1]: import numpy as np
In [2]: t = np.arange(0.0, np.pi, 0.01)

In [3]: a0, f0 = 5, 1
In [4]: def signal(t, a=a0, f=f0):
   ...:     return a*np.sin(2*np.pi*f*t)

In [5]: import matplotlib.pyplot as plt
In [6]: fig, ax = plt.subplots()
In [7]: l, = plt.plot(t, signal(t))

In [8]: axfreq = plt.axes([0.25, 0.10, 0.65, 0.03])
In [9]: axamp  = plt.axes([0.25, 0.15, 0.65, 0.03])

In[10]: plt.subplots_adjust(bottom=0.25)

In[11]: from matplotlib.widgets import Slider
In[12]: sfreq = Slider(axfreq, "Fréquence", 0.1, 30.0,
                       valinit=f0)
In[13]: samp = Slider(axamp, "Amplitude", 0.1, 10.0,
                      valinit=a0)
```

Interface graphique sous matplotlib

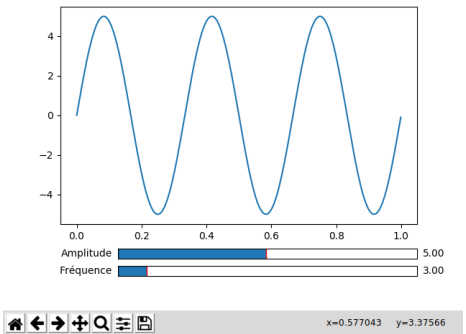
Utilisation de sliders




```
In [1]: import numpy as np
In [2]: t = np.arange(0.0, np.pi, 0.01)

In [3]: a0, f0 = 5, 1
In [4]: def signal(t, a=a0, f=f0):
   ...:     return a*np.sin(2*np.pi*f*t)

In [5]: import matplotlib.pyplot as plt
In [6]: fig, ax = plt.subplots()
In [7]: l, = plt.plot(t, signal(t))
In [8]: axfreq = plt.axes([0.25, 0.10, 0.65, 0.03])
In [9]: axamp  = plt.axes([0.25, 0.15, 0.65, 0.03])
In[10]: plt.subplots_adjust(bottom=0.25)
In[11]: from matplotlib.widgets import Slider
In[12]: sfreq = Slider(axfreq, "Fréquence", 0.1, 30.0,
                       valinit=f0)
In[13]: samp = Slider(axamp, "Amplitude", 0.1, 10.0,
                      valinit=a0)
```

Interface graphique sous matplotlib

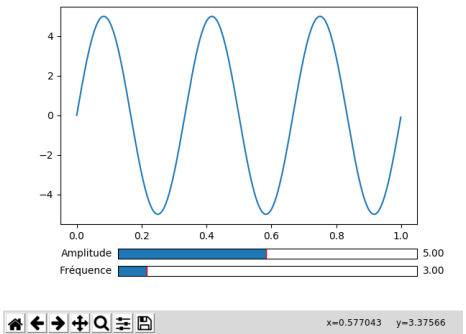
Utilisation de sliders




```
In [1]: import numpy as np
In [2]: t = np.arange(0.0, np.pi, 0.01)

In [3]: a0, f0 = 5, 1
In [4]: def signal(t, a=a0, f=f0):
   ...:     return a*np.sin(2*np.pi*f*t)

In [5]: import matplotlib.pyplot as plt
In [6]: fig, ax = plt.subplots()
In [7]: l, = plt.plot(t, signal(t))

In [8]: axfreq = plt.axes([0.25, 0.10, 0.65, 0.03])
In [9]: axamp  = plt.axes([0.25, 0.15, 0.65, 0.03])

In[10]: plt.subplots_adjust(bottom=0.25)

In[11]: from matplotlib.widgets import Slider
In[12]: sfreq = Slider(axfreq, "Fréquence", 0.1, 30.0,
                       valinit=f0)
In[13]: samp = Slider(axamp, "Amplitude", 0.1, 10.0,
                      valinit=a0)
```

Interface graphique sous matplotlib

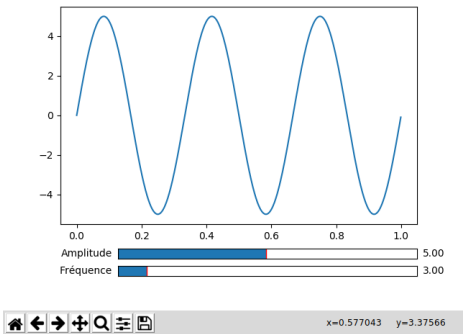
Utilisation de sliders




```
In[11]: from matplotlib.widgets import Slider
In[12]: sfreq = Slider(axfreq, "Fréquence", 0.1, 30.0,
                       valinit=f0)
In[13]: samp = Slider(axamp, "Amplitude", 0.1, 10.0,
                      valinit=a0)

In[14]: def update(val):
   ...:     l.set_ydata(signal(t, samp.val, sfreq.val))
   ...:     fig.canvas.draw_idle()

In[15]: sfreq.on_changed(update)
In[16]: samp.on_changed(update)
```

Interface graphique sous `matplotlib`

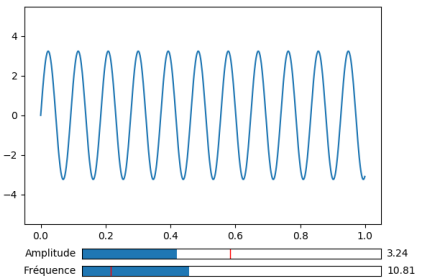
Utilisation de sliders




```
In[11]: from matplotlib.widgets import Slider
In[12]: sfreq = Slider(axfreq, "Fréquence", 0.1, 30.0,
                       valinit=f0)
In[13]: samp = Slider(axamp, "Amplitude", 0.1, 10.0,
                      valinit=a0)

In[14]: def update(val):
   ...:     l.set_ydata(signal(t, samp.val, sfreq.val))
   ...:     fig.canvas.draw_idle()

In[15]: sfreq.on_changed(update)
In[16]: samp.on_changed(update)
```

Interface graphique sous matplotlib

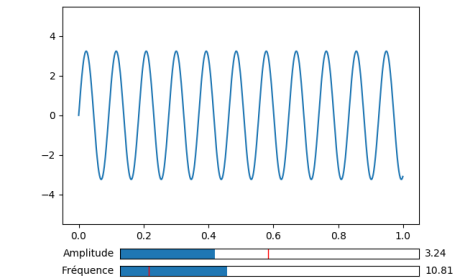
Utilisation de sliders




```
In[11]: from matplotlib.widgets import Slider
In[12]: sfreq = Slider(axfreq, "Fréquence", 0.1, 30.0,
                       valinit=f0)
In[13]: samp = Slider(axamp, "Amplitude", 0.1, 10.0,
                      valinit=a0)

In[14]: def update(val):
   ...:     l.set_ydata(signal(t, samp.val, sfreq.val))
   ...:     fig.canvas.draw_idle()

In[15]: sfreq.on_changed(update)
In[16]: samp.on_changed(update)
```

Interface graphique sous matplotlib

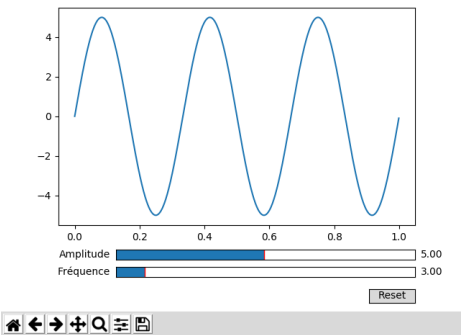
Utilisation de boutons




```
In[17]: from matplotlib.widgets import Button
In[18]: axreset = plt.axes([0.8, 0.025, 0.1, 0.04])
In[19]: button = Button(axreset, "Reset")
In[20]: def reset(event):
   ...:     sfreq.reset()
   ...:     samp.reset()
In[21]: button.on_clicked(reset)

In[22]: from matplotlib.widgets import RadioButtons
In[23]: axcolor = plt.axes([0.025, 0.5, 0.15, 0.15])
In[24]: plt.subplots_adjust(left=0.25)
In[25]: radio = RadioButtons(axcolor,
                             ("red", "blue", "green"),
                             active=1)
In[26]: def update_color(label):
   ...:     l.set_color(label)
   ...:     fig.canvas.draw_idle()
In[27]: radio.on_clicked(update_color)
```

Interface graphique sous matplotlib

Utilisation de boutons

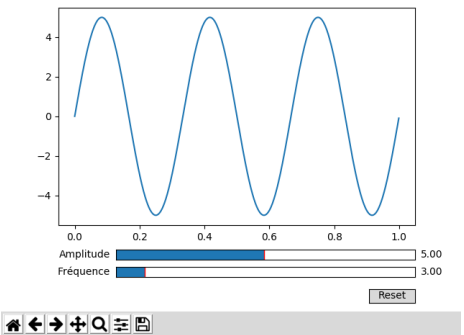



```
In[17]: from matplotlib.widgets import Button
In[18]: axreset = plt.axes([0.8, 0.025, 0.1, 0.04])
In[19]: button = Button(axreset, "Reset")
In[20]: def reset(event):
   ...:     sfreq.reset()
   ...:     samp.reset()
In[21]: button.on_clicked(reset)
```

```
In[22]: from matplotlib.widgets import RadioButtons
In[23]: axcolor = plt.axes([0.025, 0.5, 0.15, 0.15])
In[24]: plt.subplots_adjust(left=0.25)
In[25]: radio = RadioButtons(axcolor,
                             ("red", "blue", "green"),
                             active=1)
In[26]: def update_color(label):
   ...:     l.set_color(label)
   ...:     fig.canvas.draw_idle()
In[27]: radio.on_clicked(update_color)
```

Interface graphique sous matplotlib

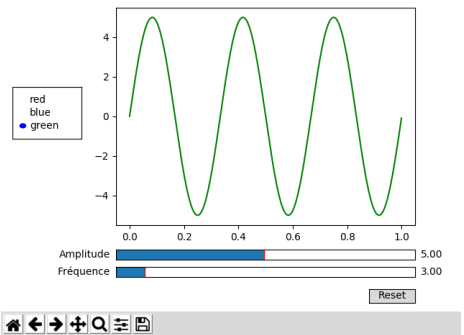
Utilisation de boutons




```
In[17]: from matplotlib.widgets import Button
In[18]: axreset = plt.axes([0.8, 0.025, 0.1, 0.04])
In[19]: button = Button(axreset, "Reset")

In[20]: def reset(event):
   ...:     sfreq.reset()
   ...:     samp.reset()
In[21]: button.on_clicked(reset)

In[22]: from matplotlib.widgets import RadioButtons
In[23]: axcolor = plt.axes([0.025, 0.5, 0.15, 0.15])
In[24]: plt.subplots_adjust(left=0.25)
In[25]: radio = RadioButtons(axcolor,
                             ("red", "blue", "green"),
                             active=1)

In[26]: def update_color(label):
   ...:     l.set_color(label)
   ...:     fig.canvas.draw_idle()
In[27]: radio.on_clicked(update_color)
```

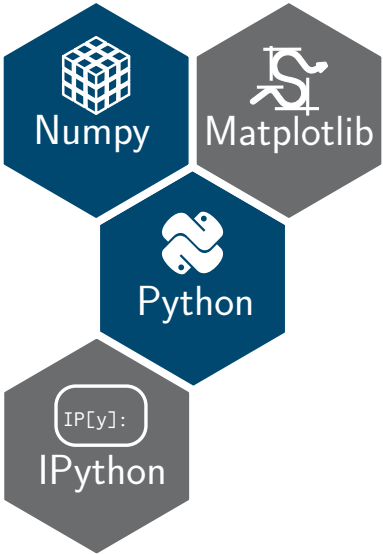
Interface graphique sous matplotlib

Utilisation de boutons




```
In[17]: from matplotlib.widgets import Button
In[18]: axreset = plt.axes([0.8, 0.025, 0.1, 0.04])
In[19]: button = Button(axreset, "Reset")

In[20]: def reset(event):
   ...:     sfreq.reset()
   ...:     samp.reset()
In[21]: button.on_clicked(reset)

In[22]: from matplotlib.widgets import RadioButtons
In[23]: axcolor = plt.axes([0.025, 0.5, 0.15, 0.15])
In[24]: plt.subplots_adjust(left=0.25)
In[25]: radio = RadioButtons(axcolor,
                             ("red", "blue", "green"),
                             active=1)

In[26]: def update_color(label):
   ...:     l.set_color(label)
   ...:     fig.canvas.draw_idle()
In[27]: radio.on_clicked(update_color)
```

- API Documentation : http://matplotlib.org/api/widgets_api.html
- Exemples & démo. : http://matplotlib.org/examples/widgets/index.html
- Pour rappel, les exemples proposés à la préparation à l'aggrégation à Montrouge : http://poisson.ens.fr/Ressources/index.php

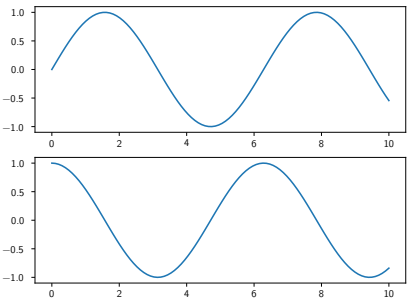


Option « Programmation en Python »

Épilogue : `matplotlib` et plus si affinités

- Historiquement, matplotlib a été développée comme un clone de MATLAB afin de faciliter la conversion des utilisateurs de MATLAB
- L'interface pyplot (plt) fournit ainsi des commandes identiques à celles de MATLAB

```
In [1]: import numpy as np
In [2]: x = np.linspace(0, 10, 100)

In [3]: import matplotlib.pyplot as plt
In [4]: plt.figure()

In [5]: plt.subplot(2, 1, 1)
In [6]: plt.plot(x, np.sin(x))

In [7]: plt.subplot(2, 1, 2)
In [8]: plt.plot(x, np.cos(x))
```

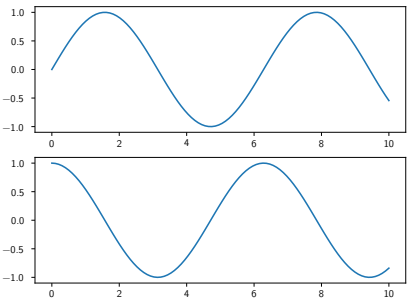


⚠ Comment accéder à la première sous-figure une fois la seconde affichée ?

matplotlib et la programmation orientée objet

- Historiquement, matplotlib a été développée comme un clone de MATLAB afin de faciliter la conversion des utilisateurs de MATLAB
- L'interface pyplot (plt) fournit ainsi des commandes identiques à celles de MATLAB

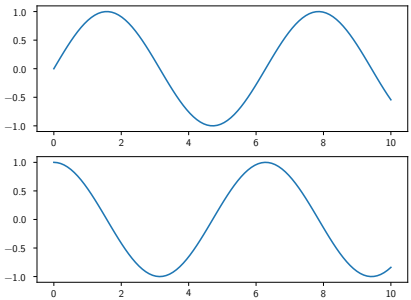
```
In [1]: import numpy as np
In [2]: x = np.linspace(0, 10, 100)

In [3]: import matplotlib.pyplot as plt
In [4]: plt.figure()

In [5]: plt.subplot(2, 1, 1)
In [6]: plt.plot(x, np.sin(x))

In [7]: plt.subplot(2, 1, 2)
In [8]: plt.plot(x, np.cos(x))
```



⚠ Comment accéder à la première sous-figure une fois la seconde affichée ?

- L'interface "orientée objet" fournit un accès simple aux différents éléments d'une figure

```
In [1]: import numpy as np
In [2]: x = np.linspace(0, 10, 100)

In [3]: import matplotlib.pyplot as plt
In [4]: fig, ax = plt.subplots(2)

In [5]: ax[0].plot(x, np.sin(x))
In [6]: ax[1].plot(x, np.cos(x))
```



`matplotlib` et la programmation orientée objet

- L'interface "orientée objet" fournit un accès simple aux différents éléments d'une figure
- Certaines fonctions de l'interface `pyplot` sont toutefois légèrement différentes
  - `plt.xlabel()` → `ax.set_xlabel()`
  - `plt.ylabel()` → `ax.set_ylabel()`
  - `plt.xlim()` → `ax.set_xlim()`
  - `plt.ylim()` → `ax.set_ylim()`
  - `plt.title()` → `ax.set_title()`
- L'avantage avec l'interface `Axes` réside dans la possibilité d'assigner un certain nombre de champ *via* la méthode `set`

```
In [3]: ax = plt.axes()
In [4]: ax.plot(x, np.sin(x))
In [5]: ax.set(xlim=(0, 10), ylim=(-2, 2),
               xlabel="x", ylabel="sin(x)",
               title="A Simple Plot");
```

`matplotlib` et la programmation orientée objet

- L'interface "orientée objet" fournit un accès simple aux différents éléments d'une figure
- Certaines fonctions de l'interface `pyplot` sont toutefois légèrement différentes
  - `plt.xlabel()` → `ax.set_xlabel()`
  - `plt.ylabel()` → `ax.set_ylabel()`
  - `plt.xlim()` → `ax.set_xlim()`
  - `plt.ylim()` → `ax.set_ylim()`
  - `plt.title()` → `ax.set_title()`
- L'avantage avec l'interface `Axes` réside dans la possibilité d'assigner un certain nombre de champ *via* la méthode `set`

```
In [3]: ax = plt.axes()
In [4]: ax.plot(x, np.sin(x))
In [5]: ax.set(xlim=(0, 10), ylim=(-2, 2),
               xlabel="x", ylabel="sin(x)",
               title="A Simple Plot");
```

- L'interface "orientée objet" fournit un accès simple aux différents éléments d'une figure
- Certaines fonctions de l'interface `pyplot` sont toutefois légèrement différentes
  - `plt.xlabel()` → `ax.set_xlabel()`
  - `plt.ylabel()` → `ax.set_ylabel()`
  - `plt.xlim()` → `ax.set_xlim()`
  - `plt.ylim()` → `ax.set_ylim()`
  - `plt.title()` → `ax.set_title()`
- L'avantage avec l'interface `Axes` réside dans la possibilité d'assigner un certain nombre de champ *via* la méthode `set`

```
In [3]: ax = plt.axes()
In [4]: ax.plot(x, np.sin(x))
In [5]: ax.set(xlim=(0, 10), ylim=(-2, 2),
               xlabel="x", ylabel="sin(x)",
               title="A Simple Plot");
```

- L'outil `Basemap` permet la représentation de données géographiques : divers modes de projection, données topographiques...
- L'installation de `Basemap` est compliquée par l'absence de module dans `pip`. Pour plus d'information *cf.* guide d'installation
- Importation du module `Basemap`

```
In [1]: from mpl_toolkits.basemap import Basemap
```

- L'outil `Basemap` permet la représentation de données géographiques : divers modes de projection, données topographiques...
- L'installation de `Basemap` est compliquée par l'absence de module dans `pip`. Pour plus d'information *cf.* guide d'installation
- Importation du module `Basemap`

```
In [1]: from mpl_toolkits.basemap import Basemap
```

- L'outil `Basemap` permet la représentation de données géographiques : divers modes de projection, données topographiques...
- L'installation de `Basemap` est compliquée par l'absence de module dans `pip`. Pour plus d'information *cf.* guide d'installation
- Importation du module `Basemap`

```
In [1]: from mpl_toolkits.basemap import Basemap
```

Goodies

Données géographiques avec `Basemap`

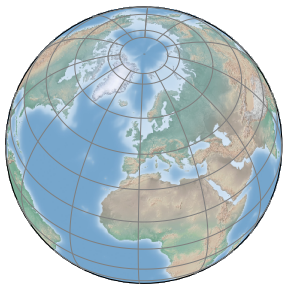
```
In [1]: from mpl_toolkits.basemap import Basemap

In [2]: m = Basemap(projection="ortho", resolution=None,
                    lat_0=48.78, lon_0=2.34)
In [3]: m.bluemarble(scale=0.2)
In [4]: m.shadedrelief(scale=0.2)
In [5]: kwargs = dict(color="gray", linewidth=0.1,
                      dashes=(None, None))

In [6]: m.drawmeridians(np.linspace(-180, 180, 13), **kwargs)
In [7]: m.drawparallels(np.linspace(-90, 90, 13), **kwargs)
```

👉 Données géographiques avec `Basemap`

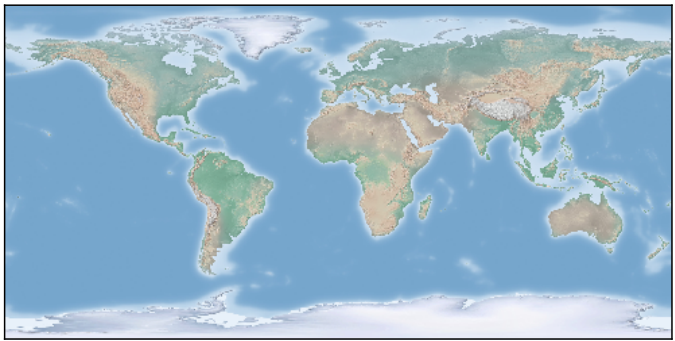
```
In [1]: from mpl_toolkits.basemap import Basemap

In [2]: m = Basemap(projection="ortho", resolution=None,
                    lat_0=48.78, lon_0=2.34)
In [3]: m.bluemarble(scale=0.2)
In [4]: m.shadedrelief(scale=0.2)
In [5]: kwargs = dict(color="gray", linewidth=0.1,
                      dashes=(None, None))

In [6]: m.drawmeridians(np.linspace(-180, 180, 13), **kwargs)
In [7]: m.drawparallels(np.linspace(-90, 90, 13), **kwargs)
```

👉 Données géographiques avec `Basemap`

```
In [1]: from mpl_toolkits.basemap import Basemap

In [2]: m = Basemap(projection="ortho", resolution=None,
                    lat_0=48.78, lon_0=2.34)
In [3]: m.bluemarble(scale=0.2)

In [4]: m.shadedrelief(scale=0.2)

In [5]: kwargs = dict(color="gray", linewidth=0.1,
                      dashes=(None, None))

In [6]: m.drawmeridians(np.linspace(-180, 180, 13), **kwargs)
In [7]: m.drawparallels(np.linspace(-90, 90, 13), **kwargs)
```



```
In [1]: from mpl_toolkits.basemap import Basemap
In [2]: m = Basemap(projection="cyl", resolution=None)
In [3]: m.shadedrelief(scale=0.2)
```

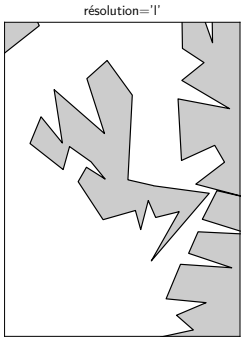


- Plusieurs modes de projection
  - projection cylindrique : `cyl`, Mercator (`merc`), *cylindrical equal area* (`cea`)
  - projection pseudo-cylindrique : Mollweide (`moll`), sinusoïdale (`sinu`), Robinson (`robin`)
  - projection en perspective : orthographique (`ortho`), gnomonique (`gnom`), stéréographique (`stere`)
  - projection conique : conique conforme de Lambert (`lcc`), ...
  - plus d'information : `http://matplotlib.org/basemap/users/mapsetup.html`

- Plusieurs modes de projection
  - projection cylindrique : `cyl`, Mercator (`merc`), *cylindrical equal area* (`cea`)
  - projection pseudo-cylindrique : Mollweide (`moll`), sinusoïdale (`sinu`), Robinson (`robin`)
  - projection en perspective : orthographique (`ortho`), gnomonique (`gnom`), stéréographique (`stere`)
  - projection conique : conique conforme de Lambert (`lcc`), ...
  - plus d'information : `http://matplotlib.org/basemap/users/mapsetup.html`

- Limites physiques (côtières, contientales…)
  - `drawcoastlines()` : affiche les limites côtières
  - `drawrivers()` : affiche les rivières, cours d'eau
  - `fillcontinents()` : change la couleur des continents (et des lacs)
- Limites politiques
  - `drawcountries()` : affiche les frontières des pays
  - `drawstates()` : affiche les états américains
- Caractérisitiques de cartes
  - `drawparallels()` : affiche les latitudes
  - `drawmeridians()` : affiche les longitudes
  - `drawgreatcircle()` : affiche un cercle entre deux points
  - `drawmapscale()` : affiche une échelle linéaire
- Fonds de cartes
  - `bluemarble()` : fond issu de la NASA
  - `shadedrelief()` : fond en relief
  - `etopo()` : fond selon le relevé topographique `etopo`

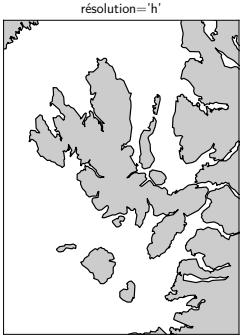
```
In [1]: for res in ["l", "h"]:
   ...:     m = Basemap(projection="gnom", lat_0=57.3, lon_0=-6.2,
   ...:                 width=90000, height=120000, resolution=res)
   ...:     m.drawcoastlines()
   ...:     m.fillcontinents()
   ...:     plt.set_title("résolution='{0}'".format(res));
```



Goodies

Données géographiques avec `Basemap`

```
In [1]: for res in ["l", "h"]:
   ...:     m = Basemap(projection="gnom", lat_0=57.3, lon_0=-6.2,
   ...:                 width=90000, height=120000, resolution=res)
   ...:     m.drawcoastlines()
   ...:     m.fillcontinents()
   ...:     plt.set_title("résolution='{0}'".format(res));
```



- Afficher des données sur cartes :
  - `contour()/contourf()` : dessiner des contours ou des contours remplis
  - `imshow()` : affiche une image
  - `plot()/scatter()` : représentation graphique sous forme de lignes et/ou marqueurs
  - `quiver()` : représente des vectors
  - `barbs()` : représentation des vents

- Densité de population dans l'état de Californie

```
# 1. Extract data
import pandas as pd
cities = pd.read_csv("./data/california_cities.csv")

lat = cities["latd"].values
lon = cities["longd"].values
population = cities["population_total"].values
area = cities["area_total_km2"].values
```

👉 Données géographiques avec `Basemap`

```
# 2. Draw the map background
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8, 8))

from mpl_toolkits.basemap import Basemap
m = Basemap(projection="lcc", resolution="h",
            lat_0=37.5, lon_0=-119,
            width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color="gray")
m.drawcountries(color="gray")
m.drawstates(color="gray")
```

```
# 3. Scatter city data, with color reflecting
# population and size reflecting area
import numpy as np
m.scatter(lon, lat, latlon=True,
          c=np.log10(population), s=area,
          cmap="Reds", alpha=0.5)

# 4. Create colorbar and legend
plt.colorbar(label=r"$\log_{10}(\mathrm{population})$")
plt.clim(3, 7)

for a in [100, 300, 500]:
    plt.scatter([], [], c="k", alpha=0.5, s=a,
                label=str(a) + " km$^2$")
plt.legend(scatterpoints=1, frameon=False,
           labelspacing=1, loc="lower left")
```

Goodies

👉 Données géographiques avec Basemap

```
# 2. Draw the map background
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8, 8))

from mpl_toolkits.basemap import Basemap
m = Basemap(projection="lcc", resolution="h",
            lat_0=37.5, lon_0=-119,
            width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color="gray")
m.drawcountries(color="gray")
m.drawstates(color="gray")
```

```
# 3. Scatter city data, with color reflecting
# population and size reflecting area
import numpy as np
m.scatter(lon, lat, latlon=True,
          c=np.log10(population), s=area,
          cmap="Reds", alpha=0.5)

# 4. Create colorbar and legend
plt.colorbar(label=r"$\log_{10}(\mathrm{population})$")
plt.clim(3, 7)

for a in [100, 300, 500]:
    plt.scatter([], [], c="k", alpha=0.5, s=a,
                label=str(a) + " km$^2$")
plt.legend(scatterpoints=1, frameon=False,
           labelspacing=1, loc="lower left")
```

Goodies

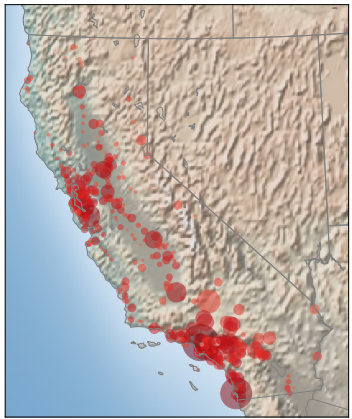
Données géographiques avec Basemap

```
# 2. Draw the map background
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8, 8))

from mpl_toolkits.basemap import Basemap
m = Basemap(projection="lcc", resolution="h",
            lat_0=37.5, lon_0=-119,
            width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color="gray")
m.drawcountries(color="gray")
m.drawstates(color="gray")

# 3. Scatter city data, with color reflecting
# population and size reflecting area
import numpy as np
m.scatter(lon, lat, latlon=True,
          c=np.log10(population), s=area,
          cmap="Reds", alpha=0.5)

# 4. Create colorbar and legend
plt.colorbar(label=r"$\log_{10}(\mathrm{population})$")
plt.clim(3, 7)

for a in [100, 300, 500]:
    plt.scatter([], [], c="k", alpha=0.5, s=a,
                label=str(a) + " km$^2$")
plt.legend(scatterpoints=1, frameon=False,
           labelspacing=1, loc="lower left")
```

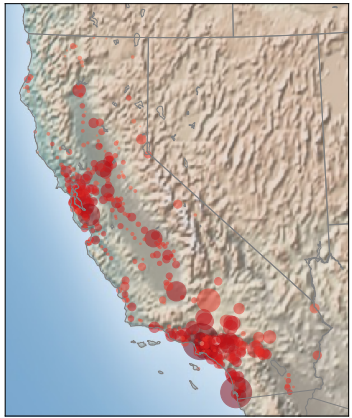
```
# 2. Draw the map background
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8, 8))

from mpl_toolkits.basemap import Basemap
m = Basemap(projection="lcc", resolution="h",
            lat_0=37.5, lon_0=-119,
            width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color="gray")
m.drawcountries(color="gray")
m.drawstates(color="gray")
```

```
# 3. Scatter city data, with color reflecting
# population and size reflecting area
import numpy as np
m.scatter(lon, lat, latlon=True,
          c=np.log10(population), s=area,
          cmap="Reds", alpha=0.5)
```

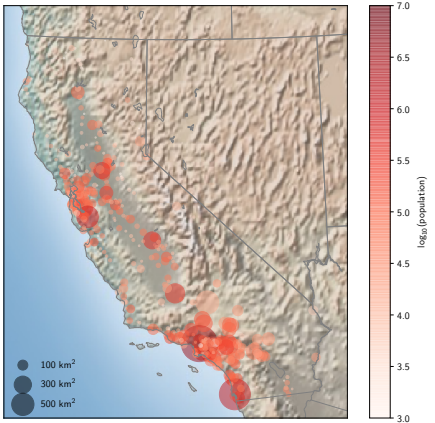
```
# 4. Create colorbar and legend
plt.colorbar(label=r"$\log_{10}(\mathrm{population})$")
plt.clim(3, 7)

for a in [100, 300, 500]:
    plt.scatter([], [], c="k", alpha=0.5, s=a,
                label=str(a) + " km$^2$")
plt.legend(scatterpoints=1, frameon=False,
           labelspacing=1, loc="lower left")
```



```
# 2. Draw the map background
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8, 8))

from mpl_toolkits.basemap import Basemap
m = Basemap(projection="lcc", resolution="h",
            lat_0=37.5, lon_0=-119,
            width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color="gray")
m.drawcountries(color="gray")
m.drawstates(color="gray")

# 3. Scatter city data, with color reflecting
# population and size reflecting area
import numpy as np
m.scatter(lon, lat, latlon=True,
          c=np.log10(population), s=area,
          cmap="Reds", alpha=0.5)

# 4. Create colorbar and legend
plt.colorbar(label=r"$\log_{10}(\mathrm{population})$")
plt.clim(3, 7)

for a in [100, 300, 500]:
    plt.scatter([], [], c="k", alpha=0.5, s=a,
                label=str(a) + " km$^2$")
plt.legend(scatterpoints=1, frameon=False,
           labelspacing=1, loc="lower left")
```



```
# 2. Draw the map background
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8, 8))

from mpl_toolkits.basemap import Basemap
m = Basemap(projection="lcc", resolution="h",
            lat_0=37.5, lon_0=-119,
            width=1E6, height=1.2E6)
m.shadedrelief()
m.drawcoastlines(color="gray")
m.drawcountries(color="gray")
m.drawstates(color="gray")

# 3. Scatter city data, with color reflecting
# population and size reflecting area
import numpy as np
m.scatter(lon, lat, latlon=True,
          c=np.log10(population), s=area,
          cmap="Reds", alpha=0.5)

# 4. Create colorbar and legend
plt.colorbar(label=r"$\log_{10}(\mathrm{population})$")
plt.clim(3, 7)

for a in [100, 300, 500]:
    plt.scatter([], [], c="k", alpha=0.5, s=a,
                label=str(a) + " km$^2$")
plt.legend(scatterpoints=1, frameon=False,
           labelspacing=1, loc="lower left")
```



- **`healpy`** est l'implémentation Python du programme de pixelisation HEALPix
- Le programme HEALPix permet d'échantillonner/pixeliser une sphère de telle sorte à ce que chaque pixel couvre la même surface

- La sphère est ainsi découpée en $12 \times N_{side}^2$ pixels où N_{side} est une puissance de 2
- Installation

```
>_ pip install healpy
```

- Importation

```
In [1]: import healpy as hp
```

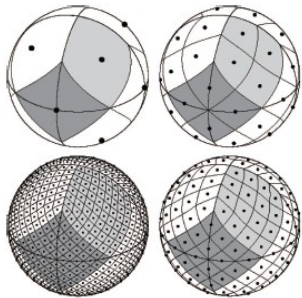
- **healpy** est l'implémentation Python du programme de pixelisation HEALPix
- Le programme HEALPix permet d'échantillonner/pixeliser une sphère de telle sorte à ce que chaque pixel couvre la même surface

- La sphère est ainsi découpée en $12 \times N_{side}^2$ pixels où N_{side} est une puissance de 2
- Installation

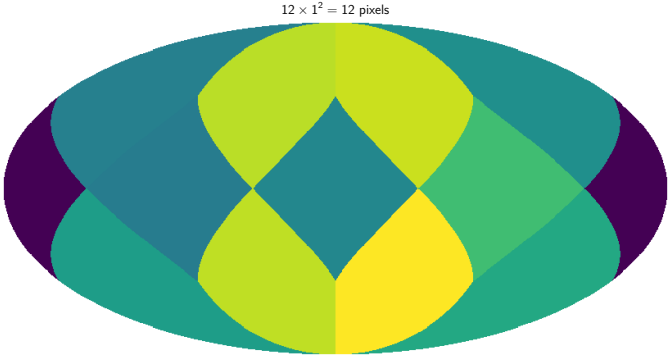
```
>_ pip install healpy
```

- Importation

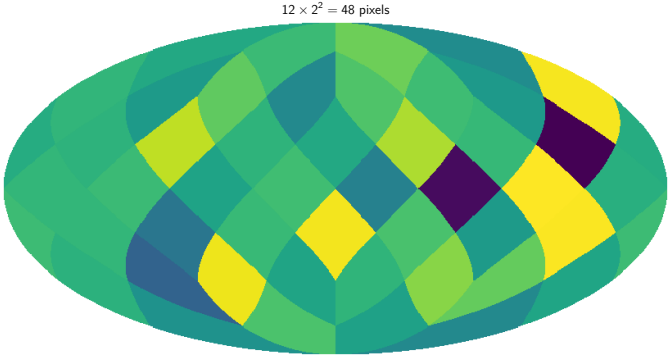
```
In [1]: import healpy as hp
```



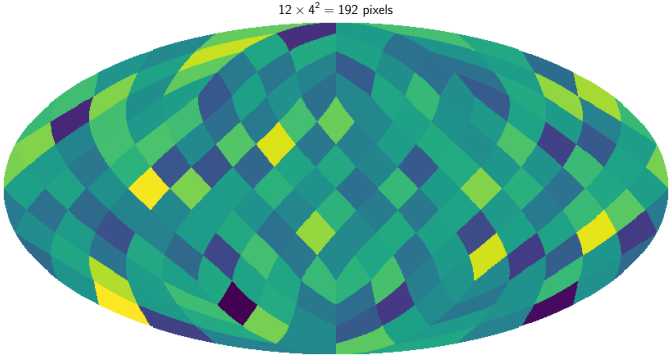
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



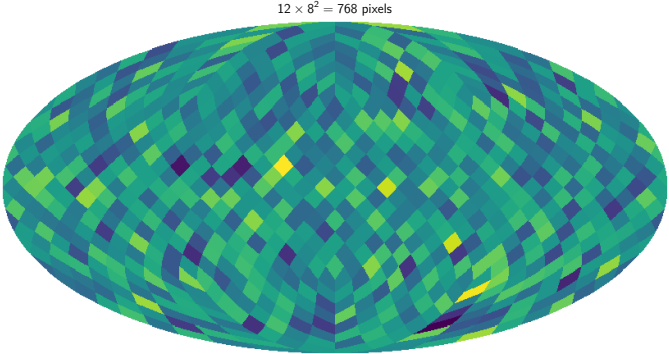
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



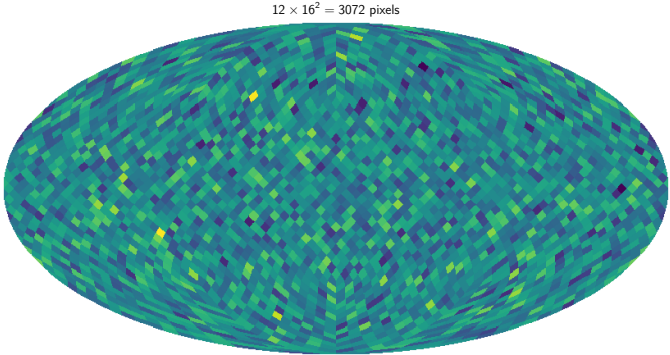
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



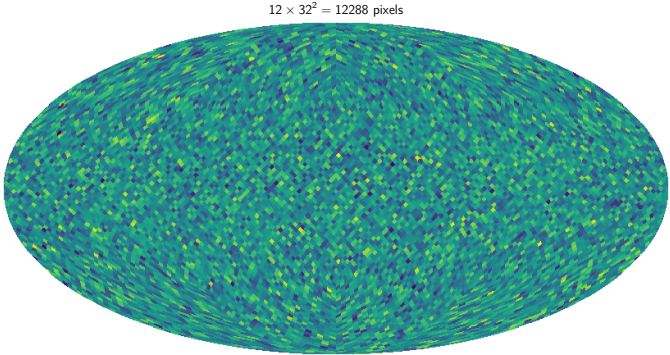
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



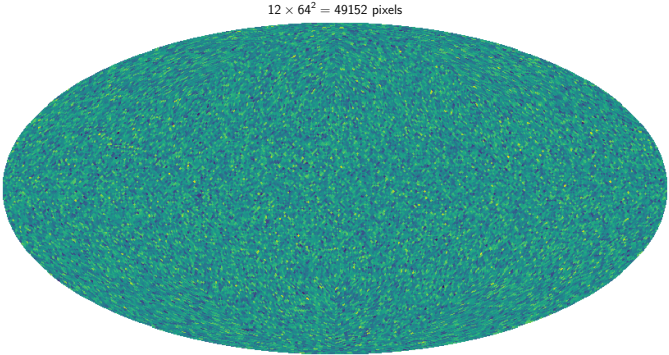
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



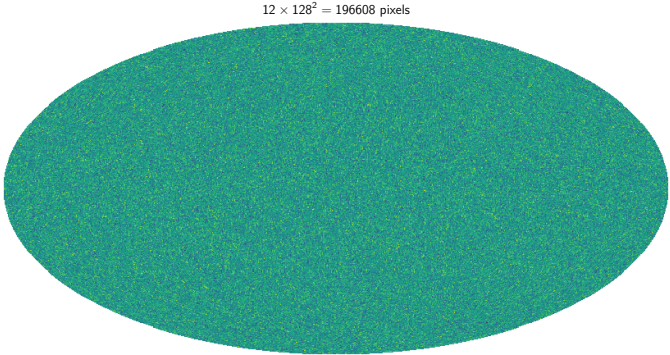
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



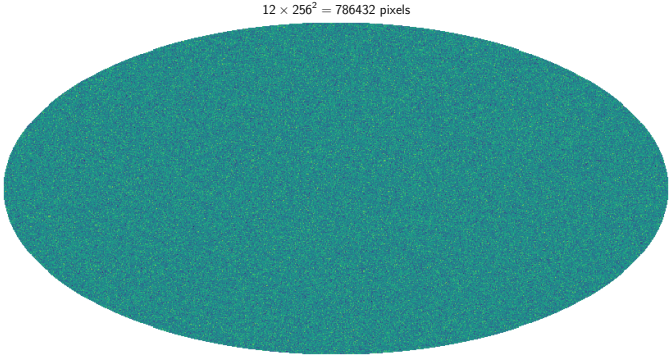
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



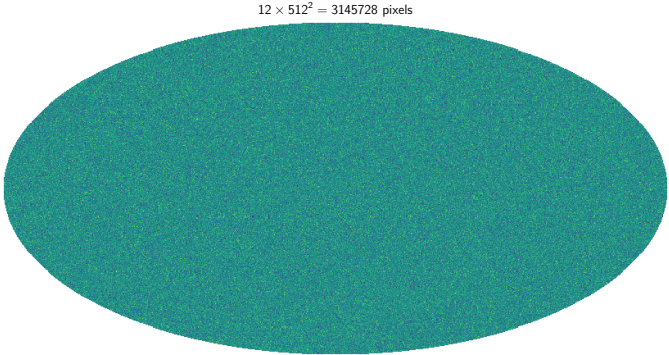
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



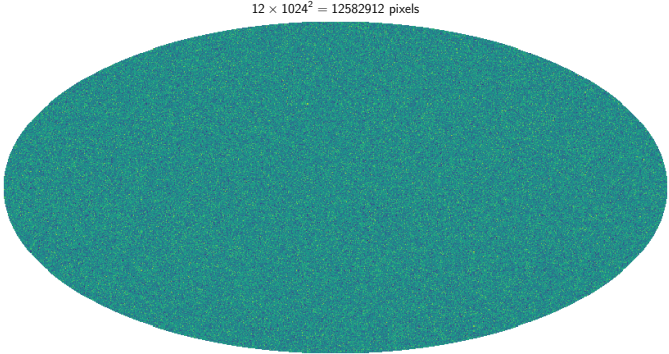
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



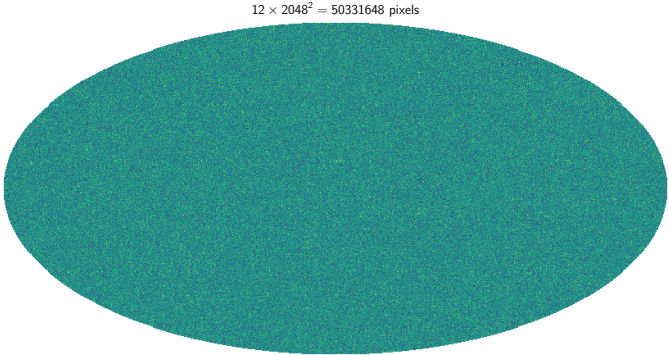
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



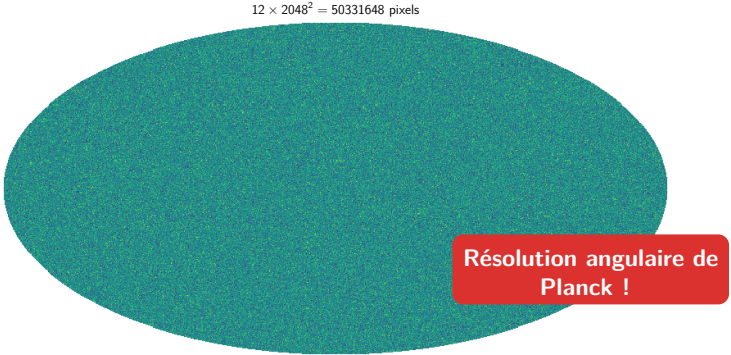
```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```

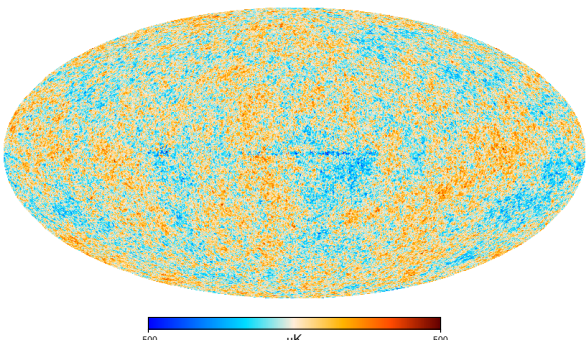


```
In [1]: import healpy as hp

In [2]: for n in np.arange(12):
   ...:     hp.mollview(np.random.randn(12*2**(2*n)), cbar=False,
                        title=r"$12\times%i^2 = %i$ pixels" % (2**n, 12*2**(2*n)))
```



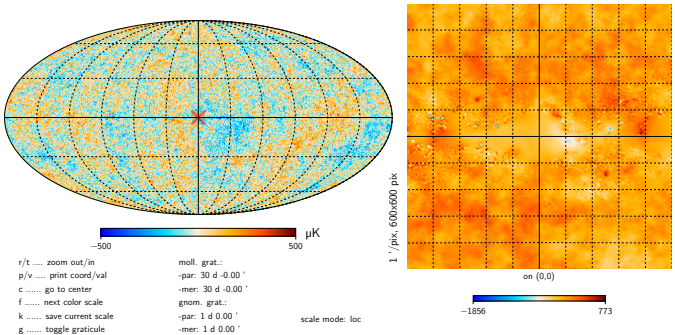
- Cartes CMB en libre accès sur *Planck Legacy Archive*

```
In [1]: import healpy as hp

In [2]: map = hp.read_map("COM_CMB_IQU-smica-field-Int_2048_R2.01_full.fits")

In [3]: hp.mollview(map*1e6, title="", min=-500, max=+500, unit="µK")
In [4]: hp.mollzoom(map*1e6, title="", min=-500, max=+500, unit="µK")
```

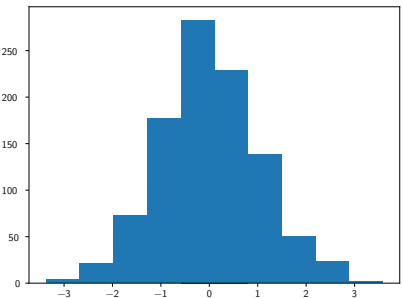


- Cartes CMB en libre accès sur *Planck Legacy Archive*

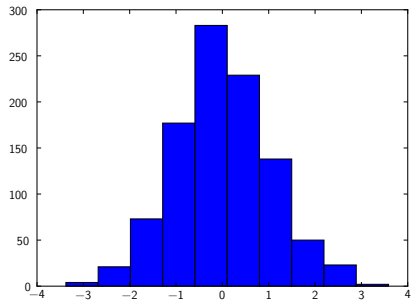
```
In [1]: import healpy as hp

In [2]: map = hp.read_map("COM_CMB_IQU-smica-field-Int_2048_R2.01_full.fits")

In [3]: hp.mollview(map*1e6, title="", min=-500, max=+500, unit="µK")
In [4]: hp.mollzoom(map*1e6, title="", min=-500, max=+500, unit="µK")
```



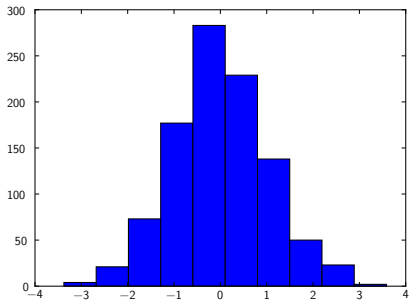
```
In [1]: import matplotlib.pyplot as plt
In [2]: import numpy as np
In [3]: x = np.random.randn(1000)
In [4]: plt.hist(x)
In [5]: plt.style.use("classic")
In [6]: plt.hist(x)
In [7]: with plt.style.context(("classic")):
   ...:     plt.hist(x)
In [8]: plt.style.available
Out[8]:
['seaborn-white',
 'bmh',
 'seaborn-dark',
 'seaborn-talk',
 'seaborn-colorblind',
 ...
```



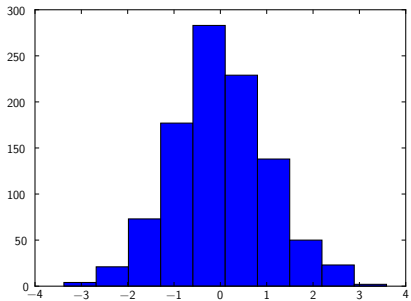
```
In [1]: import matplotlib.pyplot as plt
In [2]: import numpy as np
In [3]: x = np.random.randn(1000)
In [4]: plt.hist(x)
In [5]: plt.style.use("classic")
In [6]: plt.hist(x)
In [7]: with plt.style.context(("classic")):
   ...:     plt.hist(x)
In [8]: plt.style.available
Out[8]:
['seaborn-white',
 'bmh',
 'seaborn-dark',
 'seaborn-talk',
 'seaborn-colorblind',
 ...
```



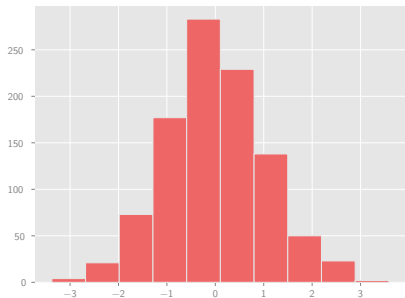
```
In [1]: import matplotlib.pyplot as plt
In [2]: import numpy as np
In [3]: x = np.random.randn(1000)
In [4]: plt.hist(x)
In [5]: plt.style.use("classic")
In [6]: plt.hist(x)
In [7]: with plt.style.context(("classic")):
   ...:     plt.hist(x)
In [8]: plt.style.available
Out[8]:
['seaborn-white',
 'bmh',
 'seaborn-dark',
 'seaborn-talk',
 'seaborn-colorblind',
 ...
```



```
In [1]: import matplotlib.pyplot as plt
In [2]: import numpy as np
In [3]: x = np.random.randn(1000)
In [4]: plt.hist(x)
In [5]: plt.style.use("classic")
In [6]: plt.hist(x)
In [7]: with plt.style.context(("classic")):
   ...:     plt.hist(x)
In [8]: plt.style.available
Out[8]:
['seaborn-white',
 'bmh',
 'seaborn-dark',
 'seaborn-talk',
 'seaborn-colorblind',
 ...
```



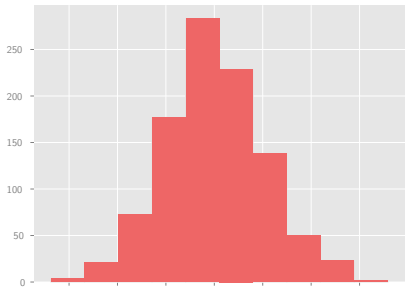
```
# Use a gray background
ax = plt.axes(facecolor="#E6E6E6")
ax.set_axisbelow(True)

# Draw solid white grid lines
ax.grid(color="w", linestyle="solid")

# Hide axis spines
for spine in ax.spines.values():
    spine.set_visible(False)

# Hide top and right ticks
ax.xaxis.tick_bottom()
ax.yaxis.tick_left()

# Lighten ticks and labels
ax.tick_params(colors="gray", direction="out")
for tick in ax.get_xticklabels():
    tick.set_color("gray")
for tick in ax.get_yticklabels():
    tick.set_color("gray")

# Control face and edge color of histogram
ax.hist(x, edgecolor="#E6E6E6", color="#EE6666");
```



- L'apparence et le style des figures peuvent être définis dans un script ou dans une session `ipython` *via* la fonction **rc** pour *runtime configuration*

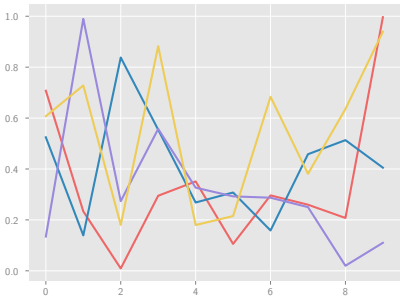
```
In [6]: from matplotlib import cycler
In [7]: colors = cycler("color",
                        ["#EE6666", "#3388BB", "#9988DD",
                         "#EECC55", "#88BB44", "#FFBBBB"])
In [8]: plt.rc("axes", facecolor="#E6E6E6", edgecolor="none",
               axisbelow=True, grid=True, prop_cycle=colors)
In [9]: plt.rc("grid", color="w", linestyle="solid")
In[10]: plt.rc("xtick", direction="out", color="gray")
In[11]: plt.rc("ytick", direction="out", color="gray")
In[12]: plt.rc("patch", edgecolor="#E6E6E6")
In[13]: plt.rc("lines", linewidth=2)

In[14]: plt.hist(x)
```

```
In[15]: for i in range(4):
   ...:     plt.plot(np.random.rand(10))
```



- L'apparence et le style des figures peuvent être définis dans un script ou dans une session `ipython` *via* la fonction **rc** pour *runtime configuration*

```
In [6]: from matplotlib import cycler
In [7]: colors = cycler("color",
                        ["#EE6666", "#3388BB", "#9988DD",
                         "#EECC55", "#88BB44", "#FFBBBB"])
In [8]: plt.rc("axes", facecolor="#E6E6E6", edgecolor="none",
               axisbelow=True, grid=True, prop_cycle=colors)
In [9]: plt.rc("grid", color="w", linestyle="solid")
In[10]: plt.rc("xtick", direction="out", color="gray")
In[11]: plt.rc("ytick", direction="out", color="gray")
In[12]: plt.rc("patch", edgecolor="#E6E6E6")
In[13]: plt.rc("lines", linewidth=2)

In[14]: plt.hist(x)

In[15]: for i in range(4):
  ...:     plt.plot(np.random.rand(10))
```



- Les préférences d'apparence peuvent être également définies de façon globale dans le fichier de configuration **matplotlibrc** du répertoire

```
In [1]: import matplotlib
In [2]: matplotlib.get_configdir()
```

Exemple de fichier de configuration (*cf.* lien pour plus de détails)

```
axes.prop_cycle : cycler("color",
                         ["#EE6666", "#3388BB", "#9988DD",
                          "#EECC55", "#88BB44", "#FFBBBB"])
axes.facecolor  : "#E6E6E6"
axes.edgecolor  : "none"
```

- Il est finalement possible de définir un fichier de style d'extension **.mplstyle** à sauvegarder dans le répertoire `matplotlib.get_configdir()`
  Exemple de fichier `python_l3.mplstyle`

```
axes.prop_cycle : cycler("color",
                         ["#EE6666", "#3388BB", "#9988DD",
                          "#EECC55", "#88BB44", "#FFBBBB"])
axes.facecolor  : "#E6E6E6"
axes.edgecolor  : "none"
```

- L'utilisation de ce fichier se fera par l'appel

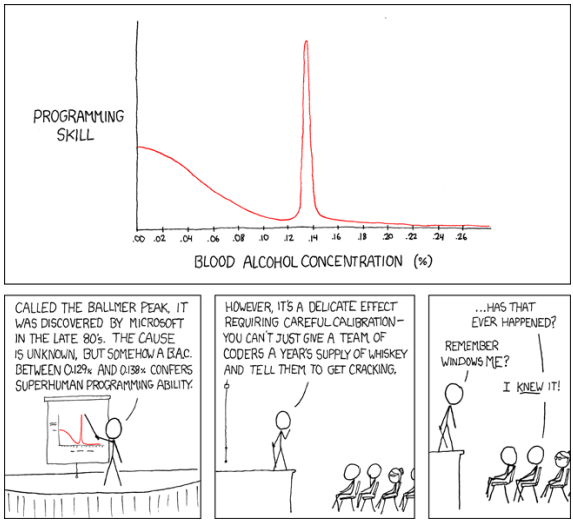
```
In [1]: import matplotlib.pyplot as plt
In [2]: plt.style.use("python_l3")
```

- Il est finalement possible de définir un fichier de style d'extension **.mplstyle** à sauvegarder dans le répertoire `matplotlib.get_configdir()`
  Exemple de fichier `python_l3.mplstyle`

```
axes.prop_cycle : cycler("color",
                         ["#EE6666", "#3388BB", "#9988DD",
                          "#EECC55", "#88BB44", "#FFBBBB"])
axes.facecolor  : "#E6E6E6"
axes.edgecolor  : "none"
```

- L'utilisation de ce fichier se fera par l'appel

```
In [1]: import matplotlib.pyplot as plt
In [2]: plt.style.use("python_l3")
```


PROGRAMMING SKILL
.00 .02 .04 .06 .08 .10 .12 .14 .16 .18 .20 .22 .24 .26
BLOOD ALCOHOL CONCENTRATION (%)
CALLED THE BALLMER PEAK. IT WAS DISCOVERED BY MICROSOFT IN THE LATE 80'S. THE CAUSE IS UNKNOWN, BUT SOMEHOW A B.A.C. BETWEEN 0.129% AND 0.138% CONFERS SUPERHUMAN PROGRAMMING ABILITY.
HOWEVER, IT'S A DELICATE EFFECT REQUIRING CAREFUL CALIBRATION— YOU CAN'T JUST GIVE A TEAM OF CODERS A YEAR'S SUPPLY OF WHISKEY AND TELL THEM TO GET CRACKING.
...HAS THAT EVER HAPPENED?
REMEMBER WINDOWS ME?
I KNEW IT!

† xkcd comic

```
In [1]: import matplotlib.pyplot as plt
In [2]: import numpy as np

In [3]: plt.figure(figsize=(8,4))
In [4]: plt.xkcd()
In [5]: def f(x, mu1=0.0, sig1=0.05, mu2=0.14, sig2=0.003):
   ...:     return np.exp(-np.power(x-mu1,2)/(2*sig1*sig1)) \
   ...:     + 3*np.exp(-np.power(x-mu2, 2)/(2*sig2*sig2))

In [6]: x = np.linspace(0, 0.28, 10000)
In [7]: plt.plot(x, f(x), "red", alpha=0.75)

In [8]: plt.xticks(np.arange(0, 0.27, 0.02))
In [9]: plt.yticks([])
In[10]: plt.gca().spines["right"].set_color("none")
In[11]: plt.gca().spines["top"].set_color("none")
In[12]: plt.xlabel("blood alcohol concentration (%)")
In[13]: plt.ylabel("programming\n skill", rotation=0)
In[14]: plt.gca().yaxis.set_label_coords(-0.15, 0.5)
In[15]: plt.tight_layout()
```

```
In [1]: import matplotlib.pyplot as plt
In [2]: import numpy as np

In [3]: plt.figure(figsize=(8,4))
In [4]: plt.xkcd()
In [5]: def f(x, mu1=0.0, sig1=0.05, mu2=0.14, sig2=0.003):
   ...:     return np.exp(-np.power(x-mu1,2)/(2*sig1*sig1)) \
   ...:     + 3*np.exp(-np.power(x-mu2, 2)/(2*sig2*sig2))

In [6]: x = np.linspace(0, 0.28, 10000)
In [7]: plt.plot(x, f(x), "red", alpha=

In [8]: plt.xticks(np.arange(0, 0.27, 0
In [9]: plt.yticks([])
In[10]: plt.gca().spines["right"].set_c
In[11]: plt.gca().spines["top"].set_col
In[12]: plt.xlabel("blood alcohol conce
In[13]: plt.ylabel("programming\n skill
In[14]: plt.gca().yaxis.set_label_coord
In[15]: plt.tight_layout()
```