Informatique 8
broadcasting, matplotlib, etc

Rémi Emonet – 2021
Université Jean Monnet – Laboratoire Hubert Curien

UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Informatique 8 : Plan

- Tableau Numpy : *broadcasting* (rappels)
- Tableau Numpy : *broadcasting* multiple (rappels)
- Reshape simplifié (rappels)
- Accès par tableau booléen (rappels)
- Broadcasting astucieux
- Matplotlib : principes appliqués
- Matplotlib : 2D
- Autres

Opérations élément par élément

- Créons deux tableaux de même dimensions

```python
a = np.arange(0, 1000, 100).reshape((2, 5))
b = np.arange(0, 10).reshape((2, 5))

assert_allclose(a, np.array([[  0, 100, 200, 300, 400],
                             [500, 600, 700, 800, 900]]))

assert_allclose(b, np.array([[  0,   1,   2,   3,   4],
                             [  5,   6,   7,   8,   9]]))
```

- Opérations éléments par élément

```python
c = a+b
assert_allclose(c, np.array([[  0, 101, 202, 303, 404],
                             [505, 606, 707, 808, 909]]))
d = a*b
assert_allclose(d, np.array([[  0, 100, 400, 900,1600],
                             [2500,3600,4900,6400,8100]]))
```

```python
print(a/b)
#   [[  nan  100.  100.  100.  100.]
#    [ 100.  100.  100.  100.  100.]]
# nan : « not a number » (valeur spéciale, « pas un nombre »)
```


Opérations entre un tableau et un nombre

- Créons un tableaux

```python
a = np.arange(0, 1000, 100).reshape((2, 5))
assert_allclose(a, np.array([[  0, 100, 200, 300, 400],
                             [500, 600, 700, 800, 900]]))
```

- Opérations avec un nombre

```python
print(a+1)                  # ou 1+a
#   [[  1 101 201 301 401]
#    [501 601 701 801 901]]

print(a/100)                # ou a*0.01 ou 0.01*a
#   [[ 0.  1.  2.  3.  4.]
#    [ 5.  6.  7.  8.  9.]]

print(a**2)
#   [[     0  10000  40000  90000 160000]
#    [250000 360000 490000 640000 810000]]

print(100/a)
#   [[ inf         1.      0.5         0.3333  0.25  ]
#    [ 0.2         0.1667  0.14285714  0.125   0.1111]]
#
# inf : « infinity » (infini)
```


Informatique 8 : Plan

- Tableau Numpy : *broadcasting* (rappels)
- Tableau Numpy : *broadcasting* multiple (rappels)
- Reshape simplifié (rappels)
- Accès par tableau booléen (rappels)
- Broadcasting astucieux
- Matplotlib : principes appliqués
- Matplotlib : 2D
- Autres

Broadcasting : tableau avec nombre

- *Broadcasting* = radiodiffusion = large diffusion = …

```python
a = np.arange(0, 1000, 100).reshape((2, 5))
print(a + 10)
print(10 / a)
```

- Opération entre un tableau et nombre
- … le nombre est « diffusé » vers tous les éléments du tableau
- … vu autrement,
  - un tableau virtuel est créé en répétant le nombre autant que nécessaire

```
a = np.arange(21).reshape((3, 7))
a + 3
```

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20

+

3	3	3	3	3	3	3
3	3	3	3	3	3	3
3	3	3	3	3	3	3

Broadcasting : entre tableaux compatibles

```python
a = np.arange(0, 1000, 100).reshape((2, 5))
b = np.array([[1, 3, 5, 8, 9]])
print(a + b)
c = np.array([[1], [9]])
print(a + c)
```

- … les élements du petit tableau sont « diffusés » vers ceux du grand
- … un tableau virtuel est créé en répétant le petit tableau

a = np.arange(21).reshape((3, 7))
b = np.arange(7).reshape((1, 7))
a + b

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20

+

0	1	2	3	4	5	6
0	1	2	3	4	5	6
0	1	2	3	4	5	6

Broadcasting : entre tableaux compatibles

a = np.arange(21).reshape((3, 7))
b = np.arange(7).reshape((1, 7))
a + b

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20

+

0	1	2	3	4	5	6
0	1	2	3	4	5	6
0	1	2	3	4	5	6

a = np.arange(21).reshape((3, 7))
b = np.arange(3).reshape((3, 1))
a + b

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20

+

0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	2	2	2	2	2	2

Exemple de broadcasting

```python
a = np.random.uniform(100, 150, (15, 200))
# 15×200 nombres au hasard tirés entre 100 et 150
```

- Tableau ⇹ nombre

```python
b = a - np.mean(a)  # que vaut b ?
# les valeurs de a moins la moyenne de ces valeurs
```

- Tableau ⇹ tableau

```python
c = a - np.mean(a, axis=1)  # que vaut c ?
# ERREUR : (15, 200) est incompatible avec (15,)
```

- Tableau ⇹ tableau

```python
d = a - np.mean(a, axis=1, keepdims=True)  # que vaut d ?
# les valeurs de a moins la moyenne la ligne correspondante
```

- Tableau ⇹ tableau

```python
e = a - np.mean(a, axis=0, keepdims=True)  # que vaut e ?
# les valeurs de a moins la moyenne la colonne correspondante
```


Informatique 8 : Plan

- Tableau Numpy : *broadcasting* (rappels)
- Tableau Numpy : *broadcasting* multiple (rappels)
- Reshape simplifié (rappels)
- Accès par tableau booléen (rappels)
- Broadcasting astucieux
- Matplotlib : principes appliqués
- Matplotlib : 2D
- Autres

Broadcasting Multiple

- Broadcasting sur les deux tableaux

a = np.arange(7).reshape((1, 7))
b = np.arange(3).reshape((3, 1))
a + b

0	1	2	3	4	5	6
0	1	2	3	4	5	6
0	1	2	3	4	5	6

+

0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	2	2	2	2	2	2

- NB: Marche aussi avec plus de 2 dimensions

Tableaux compatibles ?

- Condition requise pour que `a + b` marche
  - pour chaque axe i (NB symmétrie: on peut inverser a et b)
    - soit on a le même nombre d'éléments (`a.shape[i] == b.shape[i]`)
    - soit l'un des deux est `1` (`a.shape[i] == 1 or b.shape[i] == 1`)
- Exemple

```python
a = np.random.uniform(0, 1, (10, 42,  1,  1, 12, 98))
b = np.random.uniform(0, 1, (10,  1, 21,  1,  1,  1))
c = np.random.uniform(0, 1, (10,  1, 21,  1,  4,  1))
d = a + b  # Ok
e = a + c  # pas Ok (4 contre 12 ──────────────)
f = c + b  # Ok
```

- Cas de tableaux de dimension différentes
  - si un tableau est « plus petit » (moins de dimensions),
  - numpy accèpte de lui ajouter des axes au début, avec `1` élément

```python
a = np.random.uniform(0, 1, (              12, 98))
b = np.random.uniform(0, 1, (       21,  1,  1,  1))
c = np.random.uniform(0, 1, (10,  1, 21,  1,  4,  1))
d = a + b  # Ok, a mis en          ( 1,  1, 12, 98)
e = a + c  # Pb 4/12 avec a ( 1,  1,  1,  1, 12, 98)
f = c + b  # Ok, b mis en   ( 1,  1, 21,  1,  1,  1)
```

Informatique 8 : Plan

- Tableau Numpy : *broadcasting* (rappels)
- Tableau Numpy : *broadcasting* multiple (rappels)
- Reshape simplifié (rappels)
- Accès par tableau booléen (rappels)
- Broadcasting astucieux
- Matplotlib : principes appliqués
- Matplotlib : 2D
- Autres

Pseudo-Indices `None`

- Étendre un tableau en ajoutant des axes
- Comme reshape avec
  - un `1` en plus pour chaque `None`
- Exemple simple

```python
a = np.random.uniform(-1, 1, (51, 42))
# ajouter un axe (avec 1 élément) à la fin des axes de a
b = a.reshape((51, 42, 1))
c = a[:,:,None]
# ^ b et c sont identiques
```

- Exemple plus complexe

```python
a = np.random.uniform(-1, 1, (49, 51, 42))
b = a.reshape((1, 49, 1, 1, 51, 1, 42, 1, 1))
c = a[None, :, None, None, :, None, :, None, None]
# ^ b et c sont identiques
```


Informatique 8 : Plan

- Tableau Numpy : *broadcasting* (rappels)
- Tableau Numpy : *broadcasting* multiple (rappels)
- Reshape simplifié (rappels)
- Accès par tableau booléen (rappels)
- Broadcasting astucieux
- Matplotlib : principes appliqués
- Matplotlib : 2D
- Autres

Accès par masque (tableau booléen)

Exemple :

```
a = np.arange(21).reshape((3, 7))
b = a < 10
c = a[b]
```

a (3, 7)

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20

b (3, 7)

T	T	T	T	T	T	T
T	T	T	F	F	F	F
F	F	F	F	F	F	F

c (10,)

0	1	2	3	4	5	6	7	8	9

- NB: marche aussi avec plus (ou moins) de 2 dimensions
- NB: renvoie toujours un tableau 1D

Accès par masque (tableau booléen)

Exemple 2 :

```
a = np.arange(21).reshape((3, 7))
b = a%5 == 0
c = a[b]
```

a (3, 7)

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20

b (3, 7)

T	F	F	F	F	T	F
F	F	F	T	F	F	F
F	T	F	F	F	F	T

c (5,)

0	5	10	15	20

Accès par masque (tableau booléen)

- Si `a` et `b` sont des tableaux, `a[b]` marche uniquement si
  - `a` et `b` ont la même shape
  - `b` est un tableau de booléen (vrai/faux, True/False)
- `a[b]` est alors un tableau
  - 1D (toujours)
  - avec les valeurs de `a` uniquement aux positions où `b` est True
- `a[b]` peut aussi être utilisé à gauche de `=`
  - `a[b] = 12` remplace dans `a` toutes les positions où `b` est True, par 12
- Utilisations typiques
  - `a[a==10] = 100` : remplace par 100 toutes les valeurs égales à 10
  - `a[a%3==0] = a[a%3==0]**2`
    - remplace chaque valeur multiple de 3 par son carré
- Raccourci « plus égal » (et autres)
  - `x += y` presque équivalent à `x = x + y`
  - `«truc» «op»= «machin»` presque équivalent à `«truc» = «truc» «op» «machin»`
  - (marche avec tous les opérateurs)
  - au dessus, `a[a%3==0] **= 2`

Informatique 8 : Plan

- Tableau Numpy : *broadcasting* (rappels)
- Tableau Numpy : *broadcasting* multiple (rappels)
- Reshape simplifié (rappels)
- Accès par tableau booléen (rappels)
- Broadcasting astucieux
- Matplotlib : principes appliqués
- Matplotlib : 2D
- Autres

Broadcasting Multiple (reprise)

- Broadcasting sur les deux tableaux

```
a = np.arange(7).reshape((1, 7))
b = np.arange(3).reshape((3, 1))
a + b
```

0	1	2	3	4	5	6
0	1	2	3	4	5	6
0	1	2	3	4	5	6

+

0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	2	2	2	2	2	2

Broadcasting Multiple (reprise)

- Broadcasting sur les deux tableaux

```
a = 10*np.arange(1, 4).reshape((3, 1))
b = np.arange(1, 8).reshape((1, 7))
z = a + b
```

b

1	2	3	4	5	6	7

a

10	11	12	13	14	15	16	17
20	21	22	23	24	25	26	27
30	31	32	33	34	35	36	37

Broadcasting « Astucieux »

- Broadcasting classique, en choisissant
  - quels axes sont à étirer
  - quels axes sont à partager

```
a = np.random.randint(0, 100, (N, 2))
b = a.reshape((N, 1, 2))
c = a.reshape((1, N, 2))
diff = b - c
print(diff.shape)
#            └──────> (N, N, 2)
```

c

90	7	52	30	47	43

b

90
7
52
30
47
43

0	83	38	60	43	47
-83	0	-45	-23	-40	-36
-38	45	0	22	5	9
-60	23	-22	0	-17	-13
-43	40	-5	17	0	4
-47	36	-9	13	-4	0

- Ici, à partir d'un seul tableau de données 2D de shape (N, 2)
  - on partage l'axe à deux valeurs, en tant que nouvel axe **2**,
  - on duplique et broadcaste l'axe à N valeurs, en tant que deux axes (**0**, **1**)

Broadcasting etc : exemples ?

- remplacement des pas négatif + recalcul somme cumulée
- développement limité (avec courbe)
- bilan des forces en 1D

Informatique 8 : Plan

- Tableau Numpy : *broadcasting* (rappels)
- Tableau Numpy : *broadcasting* multiple (rappels)
- Reshape simplifié (rappels)
- Accès par tableau booléen (rappels)
- Broadcasting astucieux
- Matplotlib : principes appliqués
- Matplotlib : 2D
- Autres

Pyplot

- Matplotlib permet de
  - tracer des figures
  - tracer des courbes, histogrammes, etc
  - sauver dans un fichier image
  - afficher de manière interactive
- Pyplot : `from matplotlib import pyplot as plt`
  - `plt` : pour simplifier les taches simples
- Quelques fonctions `plt.truc(...)`
  - plot, savefig, show, ...
  - xlabel, ylabel, xticks, yticks, xlim, ylim, xscale, yscale, legend, grid, title, ...
  - text, annotate, ...
  - subplot
  - scatter, bar, barh, contour, contourf,
  - imshow, hist, ...
- http://matplotlib.org/api/pyplot_summary.html

Rappel... Illustration de Numpy + Matplotlib : affichage des pas (quand on marche)

Retour sur... affichage des pas (quand on marche)

- remplacement des valeurs négatives?
- calcul de la somme cumulée?

Matplotlib : apparence des courbes

- Paramètres de base de `plt.plot()`
  - codes pour la couleurs
  - codes pour le type de trait
  - codes pour le type de marqueurs (points)
- Paramètres nommés
  - pour les éléments ci-dessus (`color`, `linestyle`, `marker`)
  - pour d'autres choses : `markerfacecolor` (couleur des points), `fillstyle` (remplissage sous la courbe), `drawstyle` (comment relier les points : segment oblique, marche, ...), `markersize` (tailles des points), `label` (nom de la courbe), `linewidth` (épaisseur du trait),

Tracer de courbe avec `plt.plot`

- Plusieurs combinaisons de paramètres possibles
  - 2 listes "x" et "y" donnant les coordonnées des points
  - uniquement "y", un tableau de valeur ("x" est alors considéré comme `range(len(y))`
  - optionnellement, en plus, un code couleur+trait+marqueur
- + Paramètres nommés

Python Library Documentation: function plot in matplotlib.pyplot

matplotlib.pyplot.plot = plot(*args, **kwargs)

```
Plot lines and/or markers to the
:class:`~matplotlib.axes.Axes`.  *args* is a variable length
argument, allowing for multiple *x*, *y* pairs with an
optional format string.  For example, each of the following is
legal::

    plot(x, y)        # plot x and y using default line style and color
    plot(x, y, 'bo')  # plot x and y using blue circle markers
    plot(y)           # plot y using x as index array 0..N-1
    plot(y, 'r+')     # ditto, but with red plusses
```


Afficher la figure avec `plt.show`

- Aucun paramètre

Python Library Documentation: function show in matplotlib.pyplot

matplotlib.pyplot.show = show(*args, **kw)

```
Display a figure.
When running in ipython with its pylab mode, display all
figures and return to the ipython prompt.

In non-interactive mode, display all figures and block until
the figures have been closed; in interactive mode it has no
effect unless figures were created prior to a change from
non-interactive to interactive mode (not recommended).  In
that case it displays the figures but does not block.

A single experimental keyword argument, *block*, may be
set to True or False to override the blocking behavior
described above.
```

Exporter la figure `plt.savefig`

- Premier paramètre : nom du fichier à exporter
- ^ type en fonction de l'extension `.pdf`, `.png`, `.jpg`, ...

Python Library Documentation: function savefig in matplotlib.pyplot

matplotlib.pyplot.savefig = savefig(*args, **kwargs)

```
Save the current figure.

Call signature::

  savefig(fname, dpi=None, facecolor='w', edgecolor='w',
          orientation='portrait', papertype=None, format=None,
          transparent=False, bbox_inches=None, pad_inches=0.1,
          frameon=None)

The output formats available depend on the backend being used.

Arguments:

  *fname*:
    A string containing a path to a filename, or a Python
    file-like object, or possibly some backend-dependent object
    such as :class:`~matplotlib.backends.backend_pdf.PdfPages`.
```


Nom de l'axe horizontal avec `plt.xlabel`

- Un seul paramètre (chaîne de caractères)
- idem avec `ylabel`

Python Library Documentation: function xlabel in matplotlib.pyplot

matplotlib.pyplot.xlabel = xlabel(s, *args, **kwargs)

```
Set the *x* axis label of the current axis.

Default override is::

  override = {
      'fontsize'            : 'small',
      'verticalalignment'   : 'top',
      'horizontalalignment' : 'center'
      }

.. seealso::

    :func:`~matplotlib.pyplot.text`
        For information on how override and the optional args work
```


Graduations horizontales avec `plt.xticks`

- Paramètres
  - soit la liste des valeurs des graduatons,
  - soit deux listes (valeurs et texte de la graduation)
- idem avec `yticks`

Python Library Documentation: function xticks in matplotlib.pyplot

matplotlib.pyplot.xticks = xticks(*args, **kwargs)

```
Get or set the *x*-limits of the current tick locations and labels.

::

  # return locs, labels where locs is an array of tick locations and
  # labels is an array of tick labels.
  locs, labels = xticks()

  # set the locations of the xticks
  xticks( arange(6) )

  # set the locations and labels of the xticks
  xticks( arange(5), ('Tom', 'Dick', 'Harry', 'Sally', 'Sue') )
```


Choisir l'interval d'affichage avec `plt.xlim`

- Paramètre 1 : valeur minimale selon l'axe x
- Paramètre 2 : valeur maximale selon l'axe x
- idem avec `ylim`

Python Library Documentation: function xlim in matplotlib.pyplot

matplotlib.pyplot.xlim = xlim(*args, **kwargs)

```
Get or set the *x* limits of the current axes.

::

  xmin, xmax = xlim()   # return the current xlim
  xlim( (xmin, xmax) )  # set the xlim to xmin, xmax
  xlim( xmin, xmax )    # set the xlim to xmin, xmax

If you do not specify args, you can pass the xmin and xmax as
kwargs, e.g.::

  xlim(xmax=3) # adjust the max leaving min unchanged
  xlim(xmin=1) # adjust the min leaving max unchanged

Setting limits turns autoscaling off for the x-axis.
```


`plt.xscale`

- Un paramètre : type d'échelle parmi 'linear', 'log', 'logit', 'symlog'
- Paramètres nommés selon le type d'échelle
- idem pour `yscale`

Python Library Documentation: function xscale in matplotlib.pyplot

matplotlib.pyplot.xscale = xscale(*args, **kwargs)

```
Set the scaling of the *x*-axis.

call signature::

  xscale(scale, **kwargs)

The available scales are: 'linear' | 'log' | 'logit' | 'symlog'

Different keywords may be accepted, depending on the scale:

    'linear'



    'log'
```


Afficher la légende avec `plt.legend`

- Paramètres nommés intérresants : `loc` (placement de la légende), `ncol` (nombre de colonnes), `fontsize` (taille du texte)

Python Library Documentation: function legend in matplotlib.pyplot

matplotlib.pyplot.legend = legend(*args, **kwargs)

```
Places a legend on the axes.

To make a legend for lines which already exist on the axes
(via plot for instance), simply call this function with an iterable
of strings, one for each legend item. For example::

    ax.plot([1, 2, 3])
    ax.legend(['A simple line'])

However, in order to keep the "label" and the legend element
instance together, it is preferable to specify the label either at
artist creation, or by calling the
:meth:`~matplotlib.artist.Artist.set_label` method on the artist::

    line, = ax.plot([1, 2, 3], label='Inline label')
    # Overwrite the label by calling the method.
    line.set_label('Label via method')
    ax.legend()
```

Ajouter une grille avec `plt.grid`

Python Library Documentation: function grid in matplotlib.pyplot

matplotlib.pyplot.grid = grid(b=None, which='major', axis='both', **kwargs)

```
Turn the axes grids on or off.

Call signature::

   grid(self, b=None, which='major', axis='both', **kwargs)

Set the axes grids on or off; *b* is a boolean.  (For MATLAB
compatibility, *b* may also be a string, 'on' or 'off'.)

If *b* is *None* and ``len(kwargs)==0``, toggle the grid state.  If
*kwargs* are supplied, it is assumed that you want a grid and *b*
is thus set to *True*.

*which* can be 'major' (default), 'minor', or 'both' to control
whether major tick grids, minor tick grids, or both are affected.

*axis* can be 'both' (default), 'x', or 'y' to control which
set of gridlines are drawn.

*kwargs* are used to set the grid line properties, e.g.,::

   ax.grid(color='r', linestyle='-', linewidth=2)
```


Ajouter un titre avec `plt.title`

- ...

Python Library Documentation: function title in matplotlib.pyplot

matplotlib.pyplot.title = title(s, *args, **kwargs)

```
Set a title of the current axes.

Set one of the three available axes titles. The available titles are
positioned above the axes in the center, flush with the left edge,
and flush with the right edge.

.. seealso::
    See :func:`~matplotlib.pyplot.text` for adding text
    to the current axes

Parameters
----------
label : str
    Text to use for the title

fontdict : dict
    A dictionary controlling the appearance of the title text,
    the default `fontdict` is:

        {'fontsize': rcParams['axes.titlesize'],
```


Du texte dans un graphique avec `plt.text`

- Paramètre 1 : position "x" du texte
- Paramètre 2 : position "y" du texte
- Paramètre 3 : texte à afficher (chaîne de caractères)
- Paramètres nommés intéressants : `fontsize` (taille du texte), `horizontalalignment` (alignement du texte horizontalement, par rapport à x,y), `verticalalignment`

Python Library Documentation: function text in matplotlib.pyplot

matplotlib.pyplot.text = text(x, y, s, fontdict=None, withdash=False, **kwargs)

```
Add text to the axes.

Add text in string `s` to axis at location `x`, `y`, data
coordinates.

Parameters
----------
x, y : scalars
    data coordinates

s : string
```


Annoter un point avec `plt.annotate`

- Paramètre 1 : texte à afficher (chaîne de caractères)
- Paramètre 2 : paire (2-uplet) position "x,y" du point à annoter
- Paramètre 3 : paire (2-uplet) position "x,y" du texte
- Paramètres nommés intéressants : `arrowprops=dict(arrowstyle="->")` (pour tracer une flêche), `xycoords` (repère pour les coordonnées du point annoté), `textcoords` (repère pour les coordonnées du texte)

Python Library Documentation: function annotate in matplotlib.pyplot

matplotlib.pyplot.annotate = annotate(*args, **kwargs)

```
Create an annotation: a piece of text referring to a data
point.

Parameters
----------
s : string
    label

xy : (x, y)
    position of element to annotate. See *xycoords* to control what
```


Retour sur... affichage des pas (quand on marche)

- annoter le maximum ?

Tracer un nuage de points avec `plt.scatter`

- Paramètre 1 : liste/tableau des positions horizontales
- Paramètre 2 : liste/tableau des positions verticales

Python Library Documentation: function scatter in matplotlib.pyplot

matplotlib.pyplot.scatter = scatter(x, y, s=20, c=None, marker='o', cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, hold=None, data=None, **kwargs)

```
Make a scatter plot of x vs y, where x and y are sequence like objects
of the same lengths.

Parameters
----------
x, y : array_like, shape (n, )
    Input data

s : scalar or array_like, shape (n, ), optional, default: 20
    size in points^2.

c : color or sequence of color, optional, default : 'b'
    `c` can be a single color format string, or a sequence of color
    specifications of length `N`, or a sequence of `N` numbers to be
    mapped to colors using the `cmap` and `norm` specified via kwargs
    (see below). Note that `c` should not be a single numeric RGB or
    RGBA sequence because that is indistinguishable from an array of
```

Tracer un diagramme en barres avec `plt.bar`

- Paramètre 1 : liste/tableau des positions horizontales (gauche de chaque barre)
- Paramètre 2 : liste/tableau des hauteurs (valeur verticale)
- Paramètre nommé `width` : largeur de l'ensemble des barres

Python Library Documentation: function bar in matplotlib.pyplot

matplotlib.pyplot.bar = bar(left, height, width=0.8, bottom=None, hold=None, data=None, **kwargs)

```
Make a bar plot.

Make a bar plot with rectangles bounded by:

  `left`, `left` + `width`, `bottom`, `bottom` + `height`
       (left, right, bottom and top edges)

Parameters
----------
left : sequence of scalars
    the x coordinates of the left sides of the bars

height : sequence of scalars
    the heights of the bars
```


... barres horizontales avec `plt.barh`

- Paramètre 1 : liste/tableau des positions verticale (bas de chaque barre)
- Paramètre 2 : liste/tableau des largeur (valeur horizontale)
- Paramètre nommé `height` : hauteur de l'ensemble des barres

Python Library Documentation: function barh in matplotlib.pyplot

matplotlib.pyplot.barh = barh(bottom, width, height=0.8, left=None, hold=None, **kwargs)

```
Make a horizontal bar plot.

Make a horizontal bar plot with rectangles bounded by:

  `left`, `left` + `width`, `bottom`, `bottom` + `height`
       (left, right, bottom and top edges)

`bottom`, `width`, `height`, and `left` can be either scalars
or sequences

Parameters
----------
bottom : scalar or array-like
    the y coordinate(s) of the bars

width : scalar or array-like
```


Calculer un histogramme avec `plt.hist`

- Paramètre 1 : tableau de valeurs dont on veut l'histogramme
- Paramètre 2 (nommé `bins`) : nombre de cases dans l'histogramme
- ...

Python Library Documentation: function hist in matplotlib.pyplot

matplotlib.pyplot.hist = hist(x, bins=10, range=None, normed=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, hold=None, data=None, **kwargs)

```
Plot a histogram.

Compute and draw the histogram of *x*. The return value is a
tuple (*n*, *bins*, *patches*) or ([*n0*, *n1*, ...], *bins*,
[*patches0*, *patches1*,...]) if the input contains multiple
data.

Multiple data can be provided via *x* as a list of datasets
of potentially different length ([*x0*, *x1*, ...]), or as
a 2-D ndarray in which each column is a dataset.  Note that
the ndarray form is transposed relative to the list form.

Masked arrays are not supported at present.

Parameters
```


Retour sur... affichage des pas (quand on marche)

- histogramme des pas par minute ?

Faire des sous graphiques avec `plt.subplot`

- Paramètre 1 : nombre de lignes de graphiques
- Paramètre 2 : nombre de colones de graphiques
- Paramètre 3 : numéro/position du graphique à tracer avec le prochain `plt.plot` (début à 1, sense de la lecture (lignes d'abord))

Python Library Documentation: function subplot in matplotlib.pyplot

matplotlib.pyplot.subplot = subplot(*args, **kwargs)

```
Return a subplot axes positioned by the given grid definition.

Typical call signature::

  subplot(nrows, ncols, plot_number)

Where *nrows* and *ncols* are used to notionally split the figure
into ``nrows * ncols`` sub-axes, and *plot_number* is used to identify
the particular subplot that this function is to create within the notional
grid. *plot_number* starts at 1, increments across rows first and has a
maximum of ``nrows * ncols``.

In the case when *nrows*, *ncols* and *plot_number* are all less than 10,
a convenience exists, such that the a 3 digit number can be given instead,
where the hundreds represent *nrows*, the tens represent *ncols* and the
```


`plt.xlabel` vs `ax.set_xlabel`

- Deux « interfaces » pour matplotlib
  - l'historique `plt.`, ...
  - le plus fin `ax.`, `fig.`, ...
- Ressources
  - https://matplotlib.org/3.2.1/tutorials/introductory/lifecycle.html

Retour sur... affichage des pas (quand on marche)

- les pas et un histogrammes, etc

Informatique 8 : Plan

- Tableau Numpy : *broadcasting* (rappels)
- Tableau Numpy : *broadcasting* multiple (rappels)
- Reshape simplifié (rappels)
- Accès par tableau booléen (rappels)
- Broadcasting astucieux
- Matplotlib : principes appliqués
- Matplotlib : 2D
- Autres

Afficher une « image » avec `plt.imshow`

- Premier paramètre : tableau 2D de valeurs (comme "z" pour `contour`)
- Paramètres nommés intéressants : `interpolation` (`"nearest"` pour éviter le flou), `extent` (bornes x et y), `cmap` (couleurs), `alpha` (opacité), `norm` (normalisation des "z"))

Python Library Documentation: function imshow in matplotlib.pyplot

matplotlib.pyplot.imshow = imshow(X, cmap=None, norm=None, aspect=None, interpolation=None, alpha=None, vmin=None, vmax=None, origin=None, extent=None, shape=None, filternorm=1, filterrad=4.0, imlim=None, resample=None, url=None, hold=None, data=None, **kwargs)

```
Display an image on the axes.

Parameters
----------
X : array_like, shape (n, m) or (n, m, 3) or (n, m, 4)
    Display the image in `X` to current axes.  `X` may be a float
    array, a uint8 array or a PIL image. If `X` is an array, it
    can have the following shapes:

    - MxN -- luminance (grayscale, float array only)
    - MxNx3 -- RGB (float or uint8 array)
    - MxNx4 -- RGBA (float or uint8 array)
```


Tr. des lignes de niveau avec `plt.contour`

- Plusieurs combinaisons de paramètres possibles
  - soit un tableau 2D de valeurs "z"
  - soit 3 tableaux 1D/1D/2D (ou 2D/2D/2D) de valeurs "x", "y", "z" (pour donner les positions sur les axes x et y)
  - dans les deux cas, optionellement en plus, une liste des valeurs des lignes de niveaux
- Paramètres nommés intéressants : `extent` (bornes x et y), `cmap` (couleurs), `alpha` (opacité), `norm` (normalisation des "z")

Python Library Documentation: function contour in matplotlib.pyplot

matplotlib.pyplot.contour = contour(*args, **kwargs)

```
Plot contours.

:func:`~matplotlib.pyplot.contour` and
:func:`~matplotlib.pyplot.contourf` draw contour lines and
filled contours, respectively.  Except as noted, function
signatures and return values are the same for both versions.

:func:`~matplotlib.pyplot.contourf` differs from the MATLAB
```


... et en remplissant, avec `plt.contourf`

- idem

Python Library Documentation: function contourf in matplotlib.pyplot

matplotlib.pyplot.contourf = contourf(*args, **kwargs)

```
Plot contours.

:func:`~matplotlib.pyplot.contour` and
:func:`~matplotlib.pyplot.contourf` draw contour lines and
filled contours, respectively.  Except as noted, function
signatures and return values are the same for both versions.

:func:`~matplotlib.pyplot.contourf` differs from the MATLAB
version in that it does not draw the polygon edges.
To draw edges, add line contours with
calls to :func:`~matplotlib.pyplot.contour`.


Call signatures::

  contour(Z)

make a contour plot of an array *Z*. The level values are chosen
automatically.
```


Informatique 8 : Plan

- Tableau Numpy : *broadcasting* (rappels)
- Tableau Numpy : *broadcasting* multiple (rappels)
- Reshape simplifié (rappels)
- Accès par tableau booléen (rappels)
- Broadcasting astucieux
- Matplotlib : principes appliqués
- Matplotlib : 2D
- Autres

Menu

- Rappels ? Broadcasting astucieux ?
- TD5
- Recherche « force brute » (ex : Regression linéaire, moindre carrés)
- Le retour des planetes
- Autours du Sudoku
- Notebook pour compte-rendu de TP (chimie, physique, ...)
- Retour sur un des TP (questions ? difficultés ?)
- Exam an passé
- autre ?
  - retour sur matplotlib (subplot, etc)
  - ...

Informatique 8 : Plan

- Tableau Numpy : *broadcasting* (rappels)
- Tableau Numpy : *broadcasting* multiple (rappels)
- Reshape simplifié (rappels)
- Accès par tableau booléen (rappels)
- Broadcasting astucieux
- Matplotlib : principes appliqués
- Matplotlib : 2D
- Autres