numpy

October 14, 2020

[1]:
```
import numpy as np
```

[50]:
```
# Examples
a = np.arange(10)
np.sin(a)
```

[50]:
```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849])
```

[51]:
```
# Il ne faut pas utiliser le module math avec des tableaux numpy
import math
math.sin(a)
```

```
---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-51-8f708a3d7cef> in <module>
      1 # Il ne faut pas utiliser le module math avec des tableaux numpy
      2 import math
----> 3 math.sin(a)


    TypeError: only size-1 arrays can be converted to Python scalars
```

[52]:
```
# Comparaison de la vitesse entre une liste et un tableau
a = np.random.rand(1000000)

def carre_np(x):
    return x**2

%timeit carre_np(a)

def carre_liste(x):
    return [elm**2 for elm in x]
```

```
a_liste = list(a)

%timeit carre_liste(a_liste)
```

```
478 µs ± 35.8 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
352 ms ± 8.13 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[58]:
```
%%prun
def f(x):
    return 2*x


for i in range(10):
    a = f(i)
```

[59]:
```
# Boucle for avec un tableau
def carre_boucle(x):
    out = np.zeros(len(x))
    for i in range(len(x)):
        out[i] = x[i]**2
    return out

%timeit carre_boucle(b)
```

```
445 ms ± 9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

[60]:
```
# Numpy : code plus simple (et plus rapide)
a = np.random.rand(1000000)
b = a**2
```

[ ]:
```

```

1 Avantages (et inconvénients) des tableaux

- La taille et le type de donnée est fixé à la création du tableau

[64]:
```
a = np.array([1, 2, 4], dtype='complex')
a[1] = 3.14


a
```

[64]:
```
array([1.  +0.j, 3.14+0.j, 4.  +0.j])
```

[67]:
```python
a = np.array([1, 2.3, 4])
a[0] = 12424
a
```

[67]:
```
array([1.2424e+04, 2.3000e+00, 4.0000e+00])
```

2 Création d'un tableau

Il existe plusieurs fonctions pour créer un tableau.

- array : partir d'une liste (ou générateur)
- zeros, ones, eye
- arange
- linspace, logspace
- loadtxt
- load/save

Le type est déterminé automatiquement. On peut le forcer avec l'argument dtype

[68]:
```python
a = np.array([1, 2.3, 4])
```

[73]:
```python
np.zeros(10)
np.zeros(10, dtype='int')
np.zeros((2, 10), dtype='int')
```

[73]:
```
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

[72]:
```python
np.ones(10)
```

[72]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[74]:
```python
np.eye(10)
```

[74]:
```
array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

[83]:
```
# Attention pour linspace
# Si on veut contrôler delta_x, c'est mieux ainsi
np.arange(start=3, stop=10, step=2)

np.linspace(0, 1, 10)

a = 0
b = 1
N = 10
delta_x = (b-a)/N

a + np.arange(N)*delta_x
```

[83]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

[84]:
```
np.logspace(-1, 1, 11)
```

[84]:
```
array([ 0.1       ,  0.15848932,  0.25118864,  0.39810717,  0.63095734,
        1.        ,  1.58489319,  2.51188643,  3.98107171,  6.30957344,
       10.        ])
```

[88]:
```
a = np.random.rand(100, 3)
np.savetxt('/tmp/data.txt', a)
```

[91]:
```
b = np.loadtxt('/tmp/data.txt')
```

[95]:
```
np.save('/tmp/data.npy', a)
```

[97]:
```
np.load('/tmp/data.npy')
```

[97]:
```
array([[0.78756817, 0.77012707, 0.05666322],
       [0.67136447, 0.60927142, 0.71244226],
       [0.69509332, 0.60695902, 0.74593475],
       [0.81585006, 0.9538861 , 0.24334687],
       [0.38587552, 0.61876649, 0.0925133 ],
       [0.91710963, 0.40455482, 0.59341542],
       [0.83478973, 0.89127025, 0.23135619],
       [0.72371378, 0.41725607, 0.00672618],
       [0.75095482, 0.12960859, 0.73287232],
       [0.40283277, 0.78154341, 0.07110174],
       [0.85811748, 0.22409391, 0.1311706 ],
       [0.94658983, 0.33588581, 0.03539164],
       [0.07562765, 0.88150634, 0.71306811],
       [0.13863186, 0.15960147, 0.25445536],
       [0.00928996, 0.99026128, 0.95841149],
       [0.32415086, 0.27974203, 0.60310655],
       [0.97943222, 0.83000033, 0.68863756],
```

```
[0.2519884 , 0.90246805, 0.82075206],
[0.62937308, 0.02990324, 0.32944257],
[0.10305788, 0.81250143, 0.19434663],
[0.60563818, 0.79480923, 0.62457926],
[0.19817039, 0.7996571 , 0.42775064],
[0.03280126, 0.75296205, 0.54173969],
[0.33218053, 0.33011986, 0.77883891],
[0.13662123, 0.25383605, 0.41212962],
[0.42860105, 0.91337087, 0.33527679],
[0.44298795, 0.89396229, 0.4095222 ],
[0.32232169, 0.84180097, 0.28233382],
[0.38733562, 0.79255267, 0.98738567],
[0.07956617, 0.07757092, 0.37384693],
[0.32005714, 0.38103562, 0.52184046],
[0.84586888, 0.05998274, 0.87884175],
[0.06504126, 0.0818162 , 0.55530438],
[0.7998377 , 0.82457296, 0.09377761],
[0.65251177, 0.78827269, 0.14454776],
[0.46374061, 0.24042457, 0.13251871],
[0.82869638, 0.58725287, 0.47397153],
[0.96859355, 0.98417812, 0.08623702],
[0.73137586, 0.32639384, 0.1296064 ],
[0.03577448, 0.87722062, 0.1921185 ],
[0.49986172, 0.30937824, 0.9392147 ],
[0.3514998 , 0.95239796, 0.081281  ],
[0.83823574, 0.81437782, 0.51208033],
[0.2764049 , 0.64393839, 0.18020943],
[0.99539912, 0.98068704, 0.12607748],
[0.74427195, 0.41804779, 0.02445015],
[0.32684491, 0.85414684, 0.84464978],
[0.97673584, 0.14265542, 0.32821   ],
[0.42209541, 0.62348673, 0.84635387],
[0.2498476 , 0.58249422, 0.21309915],
[0.29935556, 0.85637654, 0.5910164 ],
[0.43075296, 0.97896021, 0.20219201],
[0.43707849, 0.33383581, 0.93293368],
[0.87510341, 0.68593899, 0.46572294],
[0.99744821, 0.69853663, 0.68345975],
[0.47783884, 0.58699341, 0.6288762 ],
[0.37607936, 0.83059447, 0.99398892],
[0.92825563, 0.34733092, 0.66668606],
[0.87765059, 0.38966542, 0.29939969],
[0.53339434, 0.2092734 , 0.04169819],
[0.66638038, 0.57608262, 0.82795793],
[0.17502075, 0.10558464, 0.01312669],
[0.93676031, 0.41835127, 0.90180574],
[0.75711222, 0.96357073, 0.73900114],
```

```
       [0.55612778, 0.32890029, 0.54491606],
       [0.49389977, 0.17947587, 0.16542156],
       [0.28788143, 0.09712776, 0.38363691],
       [0.02591453, 0.6722559 , 0.23014609],
       [0.9981691 , 0.35828297, 0.76715069],
       [0.16959178, 0.05641021, 0.54531874],
       [0.73643972, 0.3103241 , 0.90260863],
       [0.46211861, 0.92805322, 0.07535316],
       [0.91294927, 0.96586798, 0.75752538],
       [0.90649778, 0.55934242, 0.26032192],
       [0.11340406, 0.80091215, 0.48723619],
       [0.07675461, 0.5128126 , 0.08216031],
       [0.43158755, 0.01887168, 0.19439949],
       [0.71220391, 0.60987434, 0.98711333],
       [0.701058  , 0.91205995, 0.68282703],
       [0.37174364, 0.29478575, 0.21970335],
       [0.67984354, 0.24867109, 0.12902062],
       [0.16889869, 0.4510749 , 0.13250429],
       [0.33545453, 0.94005564, 0.0992588 ],
       [0.97665016, 0.60375181, 0.76254835],
       [0.88250911, 0.54450568, 0.45495728],
       [0.4305777 , 0.36864267, 0.26954182],
       [0.90390107, 0.50593545, 0.98280452],
       [0.20879689, 0.33561092, 0.95203336],
       [0.14028104, 0.08735022, 0.23846181],
       [0.94650735, 0.10389015, 0.55187484],
       [0.55444188, 0.91022777, 0.48945749],
       [0.48674463, 0.59581233, 0.8580212 ],
       [0.92891588, 0.74137559, 0.92187484],
       [0.48667986, 0.44977637, 0.88829084],
       [0.24980565, 0.7196141 , 0.27012974],
       [0.31274598, 0.50915686, 0.07215133],
       [0.62568132, 0.15736749, 0.97430344],
       [0.28453849, 0.84992917, 0.35594837],
       [0.02933084, 0.10117464, 0.60817023],
       [0.95473961, 0.54709794, 0.05766416]])
```

3 Fonctions vectorisées

C'est une fonction qui calcule sur un tableau élément par élément

[98]:
```
x = np.linspace(-1, 1, 51, endpoint=False)*np.pi
```

[101]:
```
# Souvent il n'y a rien a faire
np.sin(x)
```

```python
def ma_fonction(x):
    return np.sin(x)**2 + np.cos(x)**2
ma_fonction(x)
```

[101]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[105]:
```python
# Sinon, on utilise le décorateur vectorize
# Mais il existe des solutions pour éviter d'avoir à
# l'utiliser (c.f. prochaine partie)
@np.vectorize
def mafonction(a):
#    print('Bonjour')
    if a>0:
        return a
    else:
        return -a

mafonction(x)
```

[105]:
```
array([3.14159265, 3.01839294, 2.89519323, 2.77199352, 2.64879381,
       2.52559409, 2.40239438, 2.27919467, 2.15599496, 2.03279525,
       1.90959553, 1.78639582, 1.66319611, 1.5399964 , 1.41679669,
       1.29359698, 1.17039726, 1.04719755, 0.92399784, 0.80079813,
       0.67759842, 0.5543987 , 0.43119899, 0.30799928, 0.18479957,
       0.06159986, 0.06159986, 0.18479957, 0.30799928, 0.43119899,
       0.5543987 , 0.67759842, 0.80079813, 0.92399784, 1.04719755,
       1.17039726, 1.29359698, 1.41679669, 1.5399964 , 1.66319611,
       1.78639582, 1.90959553, 2.03279525, 2.15599496, 2.27919467,
       2.40239438, 2.52559409, 2.64879381, 2.77199352, 2.89519323,
       3.01839294])
```

[106]:
```python
def mafonction_simple(a):
#    print('Bonjour')
    if a>0:
        return a
    else:
        return -a

mafonction = np.vectorize(mafonction_simple)
```

[109]:
```python
x>0
```

[109]:
```
array([False, False, False, False, False, False, False, False, False,
       False, False, False, False, False, False, False, False, False,
```

```
       False, False, False, False, False, False, False, False,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True])
```

[107]:
```
# Il faut connaitre l'origine de cette erreur
if x>0:
    print('Bonjour')
```

```
                                                                          ␣
↪---------------------------------------------------------------------------

    ValueError                                Traceback (most recent call last)

    <ipython-input-107-f039710a8284> in <module>
      1 # Il faut connaitre l'origine de cette erreur
----> 2 if x>0:
      3     print('Bonjour')


    ValueError: The truth value of an array with more than one element is␣
↪ambiguous. Use a.any() or a.all()
```

[110]:
```
np.where(x>0, x, -x)
```

[110]:
```
array([3.14159265, 3.01839294, 2.89519323, 2.77199352, 2.64879381,
       2.52559409, 2.40239438, 2.27919467, 2.15599496, 2.03279525,
       1.90959553, 1.78639582, 1.66319611, 1.5399964 , 1.41679669,
       1.29359698, 1.17039726, 1.04719755, 0.92399784, 0.80079813,
       0.67759842, 0.5543987 , 0.43119899, 0.30799928, 0.18479957,
       0.06159986, 0.06159986, 0.18479957, 0.30799928, 0.43119899,
       0.5543987 , 0.67759842, 0.80079813, 0.92399784, 1.04719755,
       1.17039726, 1.29359698, 1.41679669, 1.5399964 , 1.66319611,
       1.78639582, 1.90959553, 2.03279525, 2.15599496, 2.27919467,
       2.40239438, 2.52559409, 2.64879381, 2.77199352, 2.89519323,
       3.01839294])
```

4 Indexer un tableau

[112]:
```
x = np.linspace(-1, 1, 51, endpoint=False)*np.pi
x[2:9]
```

[112]:
```
array([-2.89519323, -2.77199352, -2.64879381, -2.52559409, -2.40239438,
       -2.27919467, -2.15599496])
```

[113]:
```python
x[4:10:2]
# x[start:stop:step], comme range ou arange
```

[113]:
```
array([-2.64879381, -2.40239438, -2.15599496])
```

[ ]:
```python

```

[114]:
```python
# C'est un racourcis pour créer une slice
x[slice(4, 10, 2)]
```

[114]:
```
array([-2.64879381, -2.40239438, -2.15599496])
```

[128]:
```python
x = np.linspace(-1, 1, 51)
x
# tout sauf le dernier
x[:-1]
```

[128]:
```
array([-1.  , -0.96, -0.92, -0.88, -0.84, -0.8 , -0.76, -0.72, -0.68,
       -0.64, -0.6 , -0.56, -0.52, -0.48, -0.44, -0.4 , -0.36, -0.32,
       -0.28, -0.24, -0.2 , -0.16, -0.12, -0.08, -0.04,  0.  ,  0.04,
        0.08,  0.12,  0.16,  0.2 ,  0.24,  0.28,  0.32,  0.36,  0.4 ,
        0.44,  0.48,  0.52,  0.56,  0.6 ,  0.64,  0.68,  0.72,  0.76,
        0.8 ,  0.84,  0.88,  0.92,  0.96])
```

[120]:
```python
# Les deux derniers
x[-2:]
```

[120]:
```
array([0.96, 1.  ])
```

[ ]:
```python

```

[130]:
```python
# La différence entre deux éléments consécutifs (dérivée numérique)
x[1:] - x[:-1]
```

[130]:
```
array([0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04,
       0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04,
       0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04,
       0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04, 0.04,
       0.04, 0.04, 0.04, 0.04, 0.04, 0.04])
```

[131]:
```python
# Indexer avec un tableau d'entier
x[np.array([1, 5, 10])]
```

[131]:
```
array([-0.96, -0.8 , -0.6 ])
```

[133]:
```
# Par exemple : argsort
# Les trois éléments les plus petits
x = np.random.rand(10)
x
```

[133]:
```
array([0.96396784, 0.81229321, 0.99438251, 0.81246584, 0.17294009,
       0.06299503, 0.04981811, 0.58266196, 0.50673473, 0.72339183])
```

[136]:
```
x[x.argsort()][:3]
```

[136]:
```
array([0.04981811, 0.06299503, 0.17294009])
```

[141]:
```
# Avec un tableau de booléens
x = np.random.rand(5)
x
```

[141]:
```
array([0.89334533, 0.06966841, 0.504351  , 0.57275771, 0.94950988])
```

[142]:
```
x[np.array([True, False, False, True, True])]
```

[142]:
```
array([0.89334533, 0.57275771, 0.94950988])
```

[143]:
```
x[x>0.5]
```

[143]:
```
array([0.89334533, 0.504351  , 0.57275771, 0.94950988])
```

[144]:
```
def val_abs(x):
    res = np.zeros(len(x))
    res[x>0] = x[x>0]
    res[x<=0] = -x[x<=0]
    return res
```

[127]:
```
x[:, 1]
```

[127]:
```
array([0.06279038, 0.04918709, 0.76216172, 0.95101346, 0.21152223,
       0.41467358, 0.81026453, 0.2698764 , 0.0638469 , 0.86332402])
```

5 Tableau dans la mémoire

- strides

[192]:
```
x = np.random.rand(3, 4)
x.strides
```

[192]:
```
(32, 8)
```

[196]:
```
x = np.linspace(0, 1)
print(x.strides)
b = x[::2]
b.strides
```

```
(8,)
```

[196]:
```
(16,)
```

[197]:
```
(1,)
```

[197]:
```
(1,)
```

[200]:
```
type(())
```

[200]:
```
tuple
```

6 Modifier un tableau

[148]:
```
x = np.random.rand(10)
x[2:4] = np.array([1, 2])
x
```

[148]:
```
array([0.79888963, 0.77060231, 1.        , 2.        , 0.1850596 ,
       0.6057666 , 0.70432775, 0.9227318 , 0.46778595, 0.99036249])
```

[21]:
```
x = np.random.rand(10)
```

[150]:
```
x = np.random.rand(10)
x[x>.5] = .5
x
```

[150]:
```
array([0.06508248, 0.17733062, 0.5       , 0.5       , 0.4551478 ,
       0.5       , 0.5       , 0.5       , 0.32803591, 0.5       ])
```

[ ]:
```

```

[23]:
```
# Valeur absolue?
```

7 Tableaux nD

[154]:
```
a = np.array([[1,2], [3, 4]])
# l'index est un tuple
a[(1, 0)]
```

[154]:
```
3
```

[155]:
```
x = np.random.rand(5, 5)
# Récupérer une colonne
x[:,2]
```

[155]:
```
array([0.72250763, 0.57289555, 0.52329195, 0.82939019, 0.33549713])
```

[159]:
```
x.shape
```

[159]:
```
(5, 5)
```

[160]:
```
x.flatten()
```

[160]:
```
array([0.55024183, 0.77205978, 0.72250763, 0.12884491, 0.10148173,
       0.94730005, 0.65878693, 0.57289555, 0.57455849, 0.77861644,
       0.05765653, 0.57710239, 0.52329195, 0.48279831, 0.10392   ,
       0.91754299, 0.53944164, 0.82939019, 0.52004975, 0.71194507,
       0.4582634 , 0.32965476, 0.33549713, 0.33800232, 0.95859682])
```

[163]:
```
# méthode reshape
x = np.random.rand(25)
x.reshape((5, 5))
```

[163]:
```
array([[0.76174163, 0.56789357, 0.87887519, 0.62665396, 0.23743087],
       [0.56139622, 0.68744969, 0.20495418, 0.06028479, 0.01212552],
       [0.58186348, 0.60524986, 0.76859713, 0.85347495, 0.56368311],
       [0.82153901, 0.0726592 , 0.94946988, 0.90512198, 0.10372617],
       [0.59029934, 0.17514464, 0.51423141, 0.64776727, 0.31844386]])
```

[27]:
```
x = np.random.rand(5, 3)
x.strides
```

[27]:
```
(24, 8)
```

[164]:
```
# Attention, numpy évite de recopier la memoire
x = np.arange(10)
b = x[1::2]
b[2] = 100
x
```

[164]: array([0,   1,   2,   3,   4, 100,   6,   7,   8,   9])

[165]:
```
x = np.arange(10)
b = x[x%2==0]
b[2] = 100
x
```

[165]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

[30]:
```
# meshgrid
```

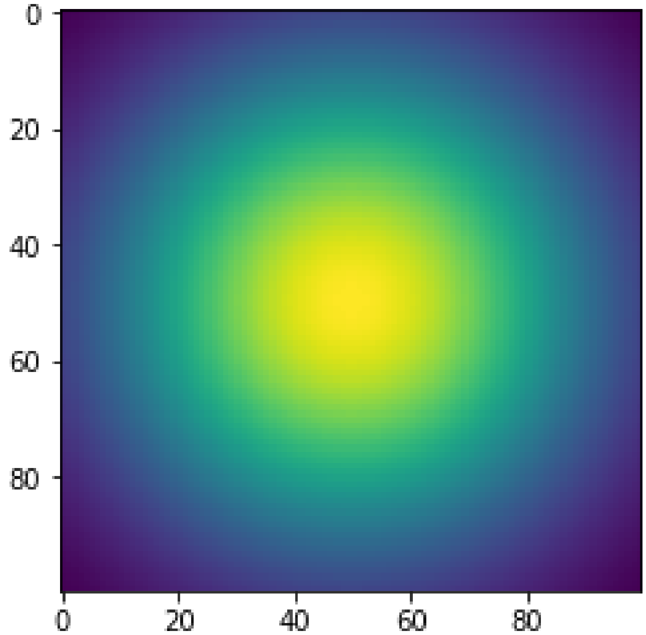
[166]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

[168]:
```
x = np.array([1, 2])
y = np.array([45, 56, 67])

X, Y = np.meshgrid(x, y)
print(X)
print(Y)
```

```
[[1 2]
 [1 2]
 [1 2]]
[[45 45]
 [56 56]
 [67 67]]
```

[170]:
```
# Création d'une image 2D avec meshgrid

x = np.linspace(-.5, .5, 100)*4*np.pi
y = np.linspace(-.5, .5, 100)*4*np.pi
X, Y = np.meshgrid(x, y)
R = np.sqrt(X**2 + Y**2)
plt.imshow(np.exp(-R**2/30))
```

[170]: <matplotlib.image.AxesImage at 0x7f7b74d39f50>



8 Broadcast

[171]:
```
a = np.random.rand(10)
b = np.random.rand(11)
a+b
```

```
↪---------------------------------------------------------------------------

    ValueError                                Traceback (most recent call last)

    <ipython-input-171-f42d7a067895> in <module>
      1 a = np.random.rand(10)
      2 b = np.random.rand(11)
----> 3 a+b


    ValueError: operands could not be broadcast together with shapes (10,)
↪(11,)
```

[172]:
```
a + 1
```

[172]:
```
array([1.75569737, 1.95839138, 1.63598944, 1.79815267, 1.26852414,
       1.75350651, 1.49225329, 1.88060177, 1.46355395, 1.14904252])
```

[176]:
```python
# Si sur une dimension la taille du tableau vaut 1, alors numpy peut l'étendre
# pour qu'elle ait la même valeur que celle de l'autre tableau
x = np.random.rand(5, 5)
a = np.array([np.arange(5)])
print(a.shape)
x[1:4, :] = a
x
```

```
(1, 5)
```

[176]:
```
array([[0.05071439, 0.78901826, 0.41781107, 0.46556661, 0.891322  ],
       [0.        , 1.        , 2.        , 3.        , 4.        ],
       [0.        , 1.        , 2.        , 3.        , 4.        ],
       [0.        , 1.        , 2.        , 3.        , 4.        ],
       [0.12950603, 0.58421764, 0.14433857, 0.20371821, 0.34703876]])
```

[179]:
```python
# Il y a une syntaxe simple pour rajouter une dimension de taille 1
# C'est newaxis
a = np.arange(5)
b = a[np.newaxis, :]
print(b.shape)
x = np.random.rand(5, 5)
x[2:4, :] = b
x
```

```
(1, 5)
```

[179]:
```
array([[0.37072389, 0.26571337, 0.75382869, 0.49241768, 0.80372115],
       [0.2019896 , 0.50292453, 0.39788085, 0.13131222, 0.71631248],
       [0.        , 1.        , 2.        , 3.        , 4.        ],
       [0.        , 1.        , 2.        , 3.        , 4.        ],
       [0.05295748, 0.9398645 , 0.75268115, 0.84297014, 0.88985663]])
```

[183]:
```python
# Exemple : calculer une moyenne pondérée
# Chaque ligne est un élève, chaque colonne un examen
notes = np.random.rand(10, 5)*20
#print(notes)
coef = np.array([1, 4, 2, 5, 8])
```

[187]:
```python
# Il est inutile de faire des boucles
np.sum(notes*coef[np.newaxis,:], axis=1)/np.sum(coef)
```

[187]:
```
array([11.121959  , 11.98277735,  8.90538059,  7.74284719,  9.44101068,
        6.04703322, 11.51351397,  9.31315847, 12.89224939, 14.52960175])
```

[ ]:

```python

```

[189]:

```python
x = np.random.rand(3, 4)
x
```

[189]:

```
array([[0.71460449, 0.24918298, 0.06893493, 0.38753238],
       [0.70728353, 0.74218708, 0.71399197, 0.37982629],
       [0.15277431, 0.87126192, 0.92394228, 0.49147835]])
```

[191]:

```python
np.max(x, axis=1)
```

[191]:

```
array([0.71460449, 0.74218708, 0.92394228])
```

9 Au delà de numpy : numba

- Calculer π (avec une formule très très lente!!!)

$$\frac{\pi}{4} = \sum_i \frac{(-1)^i}{2i+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$$

[29]:

```python
def pi_python(N):
    res = 0
    coef = 1
    for i in range(N):
        res += coef/(2*i+1)
        coef = -coef
    return 4*res

%timeit pi_python(1000000) # 155 ms

def pi_np(N):
    Ti = arange(N)
    return 4*np.sum((1-2*(Ti%2))/(2*Ti+1))

%timeit pi_np(1000000) # 28.3ms

from numba import jit, int64, float64
numba_pi = jit(float64(int64))(pi_python)

@jit( float64(int64) )
def pi_python(N):
    res = 0
    coef = 1
    for i in range(N):
        res += coef/(2*i+1)
        coef = -coef
```

```
    return 4*res
%timeit numba_pi(1000000)
```

```
10 loops, best of 3: 160 ms per loop
10 loops, best of 3: 31.6 ms per loop
100 loops, best of 3: 8.87 ms per loop
```