Python pour le calcul scientifique

Loïc Gouarin

Laboratoire de Mathématiques d'Orsay
http://www.math.u-psud.fr/∼gouarin/pythonCS.pdf

22 mai 2014 - GT Sage Paris

Plan

Pourquoi utiliser Python pour le calcul scientifique ?

- Python peut être appris en quelques jours.
- De nombreux modules existent.
- L'intérêt pour ce langage ne cesse de croître.
- Tous les outils sont là pour faire des codes performants et parallèles.
- On peut faire bien plus que du calcul.

Pourquoi utiliser Python pour le calcul scientifique ?

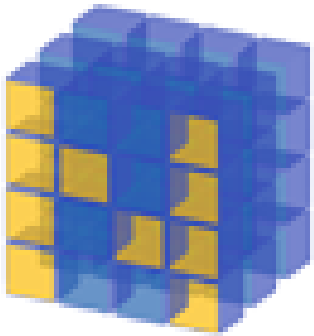
- Python peut être appris en quelques jours.
- De nombreux modules existent.
- L'intérêt pour ce langage ne cesse de croître.
- Tous les outils sont là pour faire des codes performants et parallèles.
- On peut faire bien plus que du calcul.

Langage utilisé maintenant dans les classes préparatoires scientifiques.

Comment commencer ?

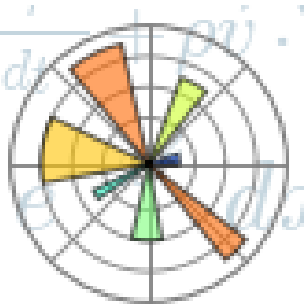
- en utilisant Sage
  http://www.sagemath.org/fr/
- en installant anaconda
  http://continuum.io/downloads

Les indispensables



NumPy

SciPy



Matplotlib

Le module numpy

- outils performants pour la manipulation de tableaux à N dimensions
- fonctions basiques en algèbre linéaire
- fonctions basiques pour les transformées de Fourier
- outils pour intégrer du code Fortran
- outils pour intégrer du code C/C++

Comparaison syntaxique NumPy, Matlab

on veut calculer

$u = 100 \exp\left(-100(x - 0.5)^2\right)$ avec $x \in [0, 1]$.

en Python

```
from numpy import *
x = linspace(0., 1., 100)
u = 100.*exp(-100.*(x - .5)**2)
```

en Matlab

```
x = linspace(0.,1.,100)
u=100.*exp(-100.*(x-.5).^2)
```

Création d'un tableau en connaissant sa taille

```
>>> import numpy as np
>>> a = np.zeros(4)
>>> a
array([ 0.,  0.,  0.,  0.])
>>> nx, ny = 2, 2
>>> a = np.zeros((nx, ny, 2))
>>> a
array([[[ 0.,  0.],
        [ 0.,  0.]],

       [[ 0.,  0.],
        [ 0.,  0.]]])
```

Il existe également

`np.ones`, `np.eye`, `np.identity`, `np.empty`, ...

Création d'un tableau avec une séquence de nombre

```
>>> a = np.linspace(-4, 4, 9)
>>> a
array([-4., -3., -2., -1.,  0.,  1.,  2.,  3.,  4.])
>>> a = np.arange(-4, 4, 1)
>>> a
array([-4, -3, -2, -1,  0,  1,  2,  3])
```

Création d'un tableau à partir d'une séquence

```
>>> a = np.array([1, 2, 3])
>>> a
array([1, 2, 3])
>>> b = np.array(range(10))
>>> b
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> L1, L2 = [1, 2, 3], [4, 5, 6]
>>> a = np.array([L1, L2])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
```

Création d'un tableau à partir d'une fonction

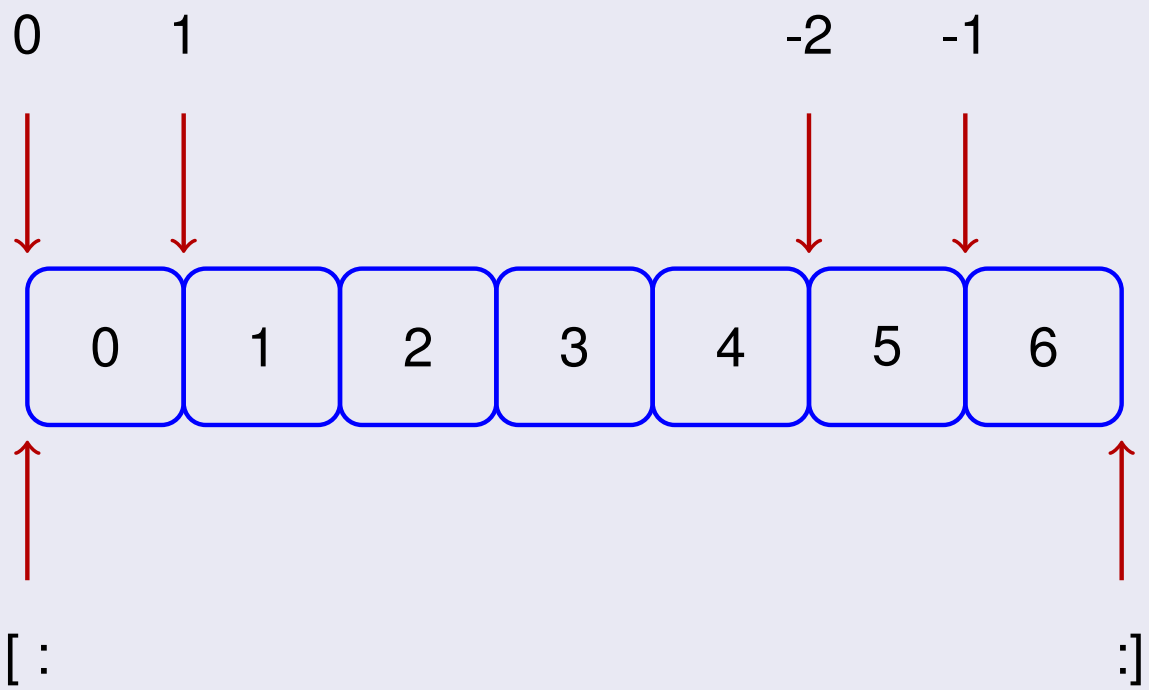
```
>>> def f(x, y):
...     return x**2 + np.sin(y)
...
>>> a = np.fromfunction(f, (2, 3))
>>> a
array([[ 0.        ,  0.84147098,  0.90929743],
       [ 1.        ,  1.84147098,  1.90929743]])
```

Caractéristiques d'un tableau a

- **a.shape** : retourne les dimensions du tableau
- **a.dtype** : retourne le type des éléments du tableau
- **a.size** : retourne le nombre total d'éléments du tableau
- **a.ndim** : retourne la dimension du tableau

Accès aux éléments d'un tableau

[debut : fin : pas]



Indexation

```
>>> L1, L2 = [1, -2, 3], [-4, 5, 6]
>>> a = np.array([L1, L2])
>>> a[1, 2]
6
>>> a[:, 1]
array([-2, 5])
>>> a[:, -1:0:-1]
array([[3, -2],
       [6, 5]])
>>> a[a < 0]
array([-2, -4])
```

Redimensionnement d'un tableau

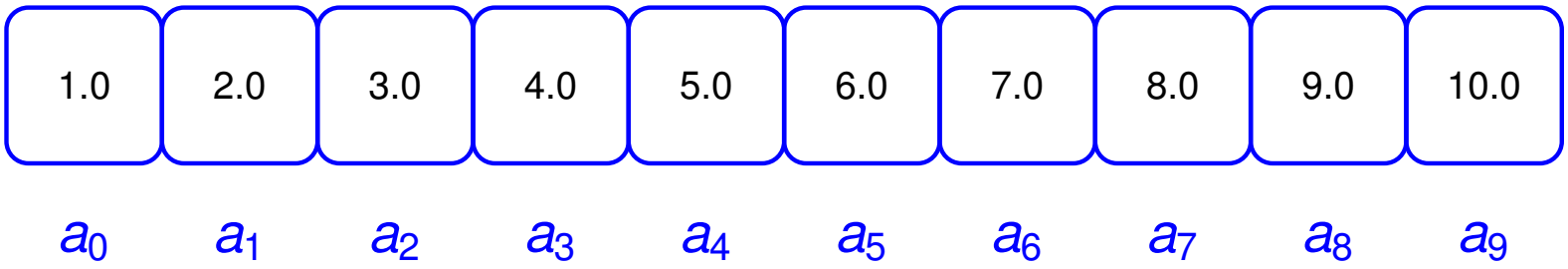
```
>>> a = np.linspace(1, 10, 10)
>>> a.shape = (2, 5)
>>> a
array([[  1.,   2.,   3.,   4.,   5.],
       [  6.,   7.,   8.,   9.,  10.]])
>>> a.shape = (a.size,)
>>> a.reshape((2, 5))
array([[  1.,   2.,   3.,   4.,   5.],
       [  6.,   7.,   8.,   9.,  10.]])
```

Attention : **reshape** ne fait pas une copie du tableau mais crée une nouvelle vue.

Les vues

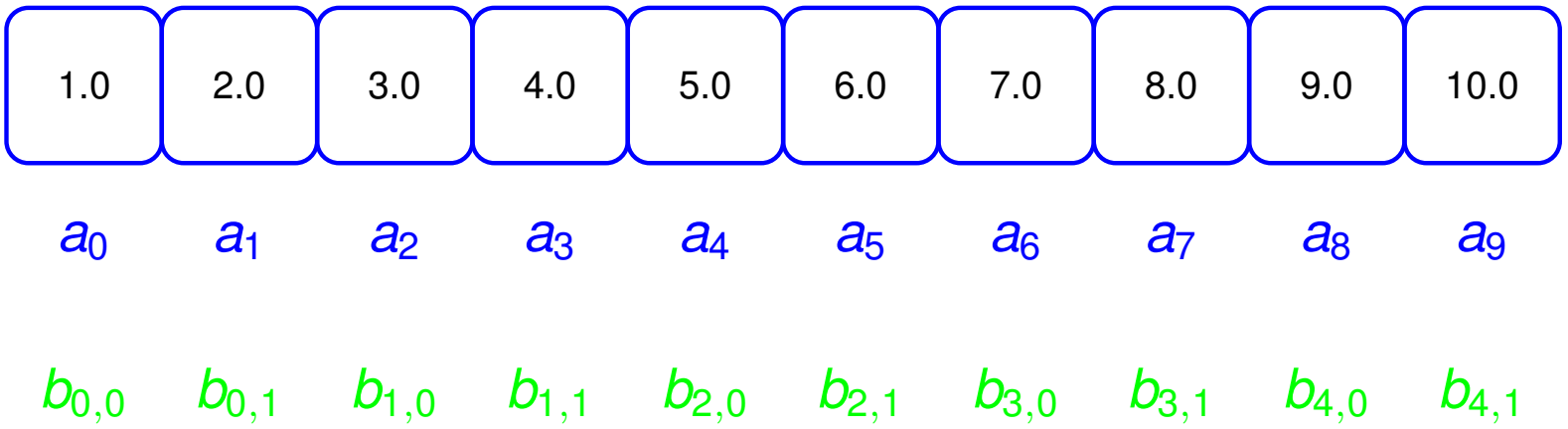
NumPy crée des tableaux alloués en mémoire selon l'alignement C ou Fortran. On peut ensuite y accéder selon différentes vues.

Les vues



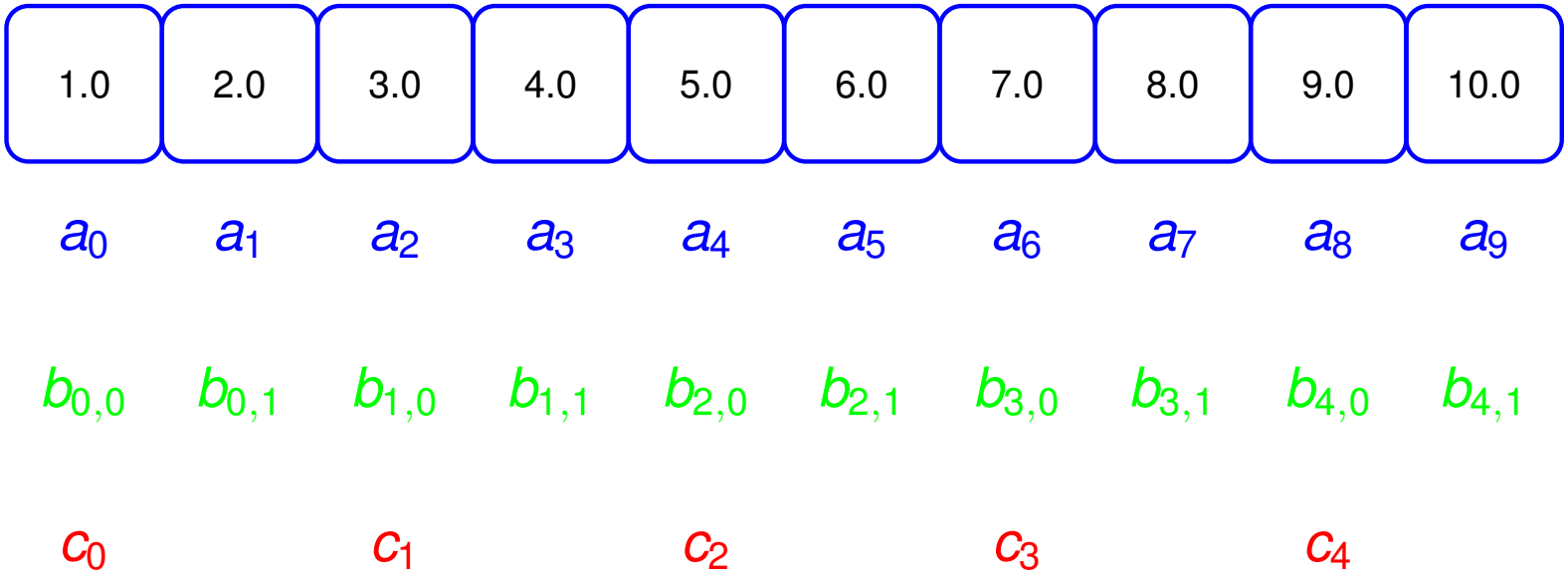

```
>>> a = np.linspace(1, 10, 10)
```

Les vues



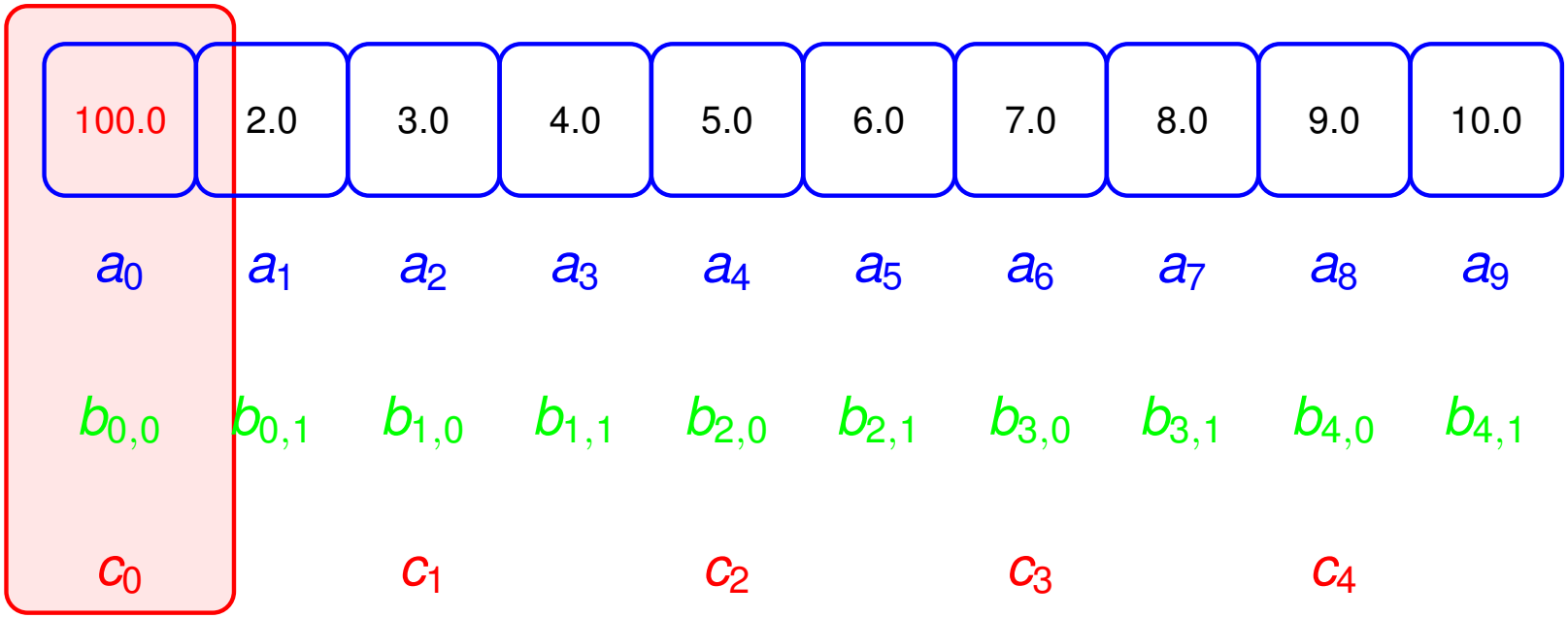

```
>>> a = np.linspace(1, 10, 10)
>>> b = a.reshape((5, 2))
```

Les vues




```
>>> a = np.linspace(1, 10, 10)
>>> b = a.reshape((5, 2))
>>> c = b[:, 0]
```

Les vues




```
>>> a = np.linspace(1, 10, 10)
>>> b = a.reshape((5, 2))
>>> c = b[:, 0]
>>> a[0] = 100
```

Les vues

```
>>> a = np.linspace(1, 10, 10)
>>> b = a.reshape((5, 2))
>>> c = b[:, 0]
>>> a[0] = 100
>>> a
array([ 100.,    2.,    3.,    4.,    5.,    6.,    7.,    8.,
>>> b
array([[ 100.,    2.],
       [   3.,    4.],
       [   5.,    6.],
       [   7.,    8.],
       [   9.,   10.]])
>>> c
array([ 100.,    3.,    5.,    7.,    9.])
```

Copie d'un tableau

```
>>> a = np.linspace(1, 5, 5)
>>> b = a
>>> c = a.copy()
>>> d = np.zeros(a.shape, a.dtype)
>>> d[:] = a
>>> b[1] = 9
>>> a; b; c; d
array([ 1.,  9.,  3.,  4.,  5.])
array([ 1.,  9.,  3.,  4.,  5.])
array([ 1.,  2.,  3.,  4.,  5.])
array([ 1.,  2.,  3.,  4.,  5.])
```

Opérations sur les tableaux

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a.sum()
45
>>> np.sum(a)
45
>>> a + a
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
>>> a*a
array([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
>>> np.dot(a, a)
285
```

SciPy

SciPy reprend l'ensemble de NumPy et contient un certain nombre de modules spécialisés.

Subpackage	Description
cluster	Clustering algorithms
constants	Physical and mathematical constants
fftpack	Fast Fourier Transform routines
integrate	Integration and ordinary differential equation solvers
interpolate	Interpolation and smoothing splines
io	Input and Output
linalg	Linear algebra
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	Optimization and root-finding routines
signal	Signal processing
sparse	Sparse matrices and associated routines
spatial	Spatial data structures and algorithms
special	Special functions
stats	Statistical distributions and functions
weave	C/C++ integration

FIGURE – SciPy Reference Guide

integrate

Ce module fournit un ensemble d'outils pour l'intégration. Ces outils se découpent en 3 grandes familles

- calcul d'intégrale à partir de fonctions Python
  quadpack, linpack, mach
- calcul d'intégrale à partir d'un ensemble de points
  fonctions Python
- résolution d'équations différentielles ordinaires
  odepack, dop, vode, linpack, mach

Toutes ces librairies sont en Fortran. Les interfaces sont réalisées directement via la C API sauf vode qui utilise f2py.

linalg

Ce module fournit un ensemble d'outils en algèbre linéaire. Il reprend l'ensemble des fonctions de numpy.linalg et apporte quelques fonctionnalités supplémentaires.

Il s'appuie sur les librairies blas et lapack. L'interface est réalisée en utilisant f2py.

```
>>> import numpy as np
>>> from numpy.random import rand as rn
>>> from scipy.linalg import lu as lu
>>> A = rn(100, 100)
>>> P, L, U = lu(A)
```

sparse

Ce module fournit un ensemble d'outils permettant de manipuler des matrices creuses.

Les formats de matrices creuses sont écrits en C et l'interface est réalisée en utilisant swig.

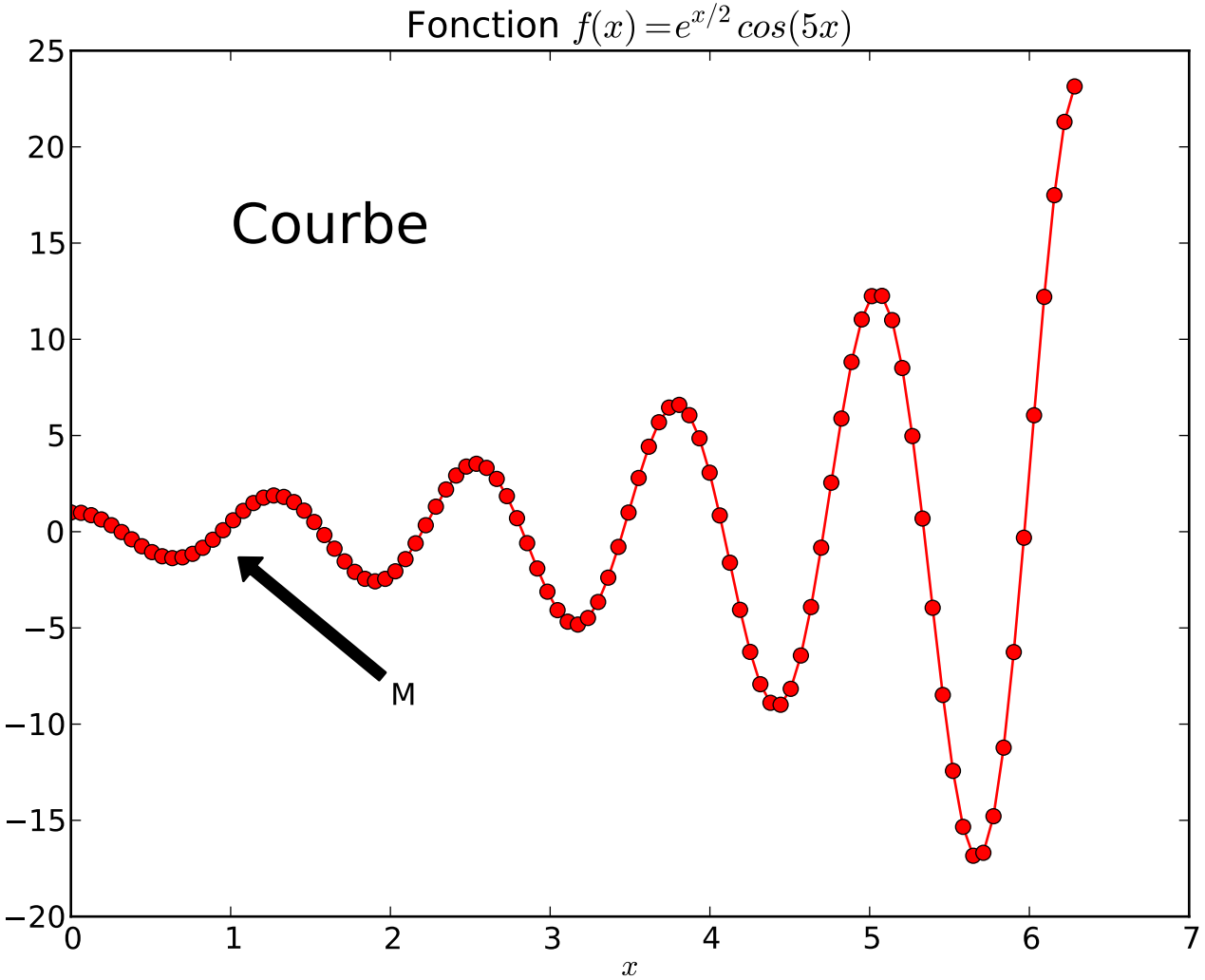
```
>>> import scipy.sparse as spsp
>>> data = array([[1,2,3,4],[1,2,3,4],[1,2,3,4]])
>>> diags = array([0,-1,2])
>>> spsp.spdiags(data, diags, 4, 4).todense()
matrix([[1, 0, 3, 0],
        [1, 2, 0, 4],
        [0, 2, 3, 0],
        [0, 0, 3, 4]])
```

Matplotlib

- Permet de faire principalement des graphes 1*D* et 2*D*, "façon *Matlab*",
- Possibilité d'interagir avec les graphes,
- Sauvegarde des figures sous différents formats : *pdf*, *ps*, *png*,...
- Graphes facilement intégrables dans une interface graphique utilisateur (*GUI*),
- Code extensible.

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

Premier exemple



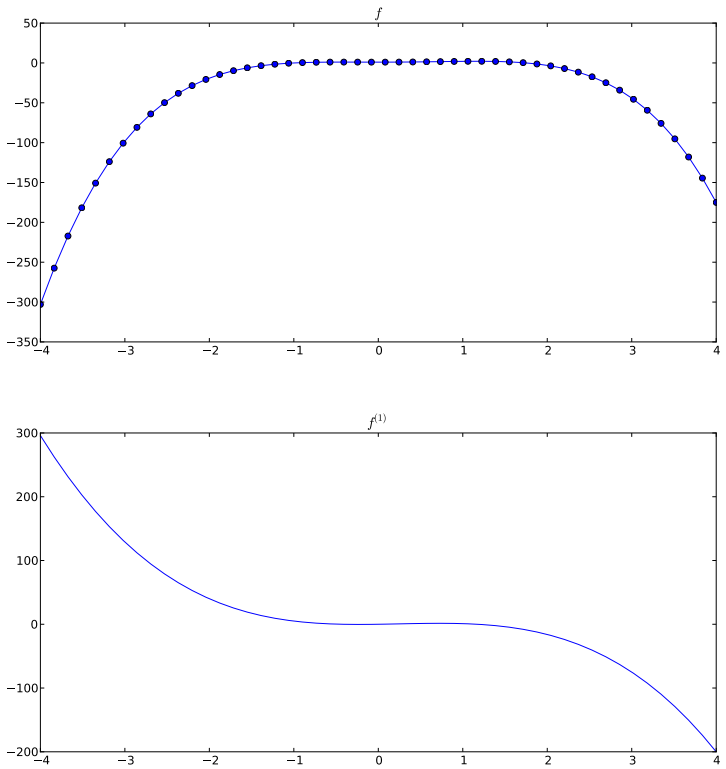
Premier exemple

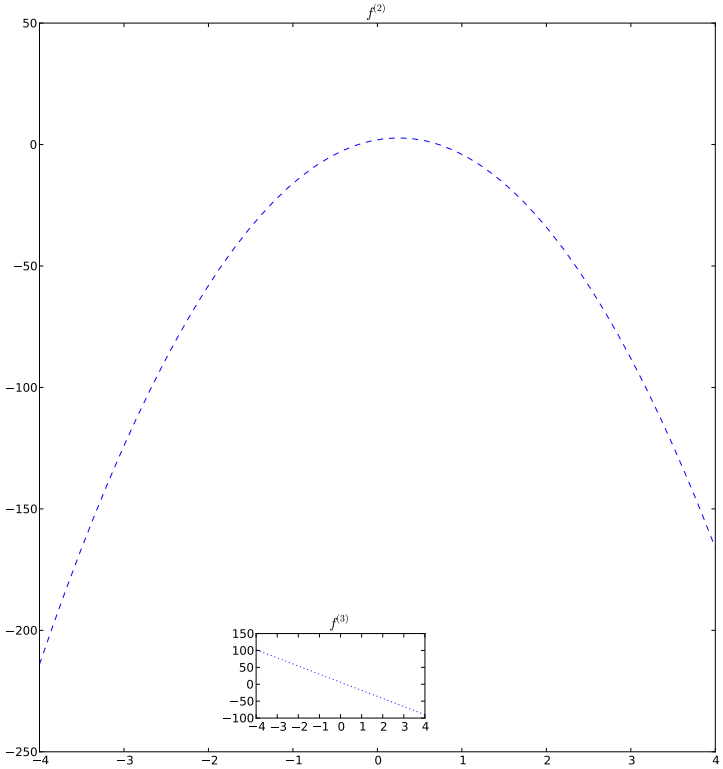
```
x = np.linspace(0., 2*np.pi, 100)
plt.plot(x, np.exp(x/2)*np.cos(5*x), '-ro')

plt.title('Fonction $f(x)=e^{x/2} cos(5x)$')
plt.xlabel('$x$')
plt.text(1, 15., 'Courbe', fontsize=22)
plt.annotate('M', xy=(1, -1), xytext=(2, -9),
             arrowprops = {facecolor:'black',
                           \parallelshrink:0.05})

plt.savefig('1D_exemple.pdf')
plt.show()
```

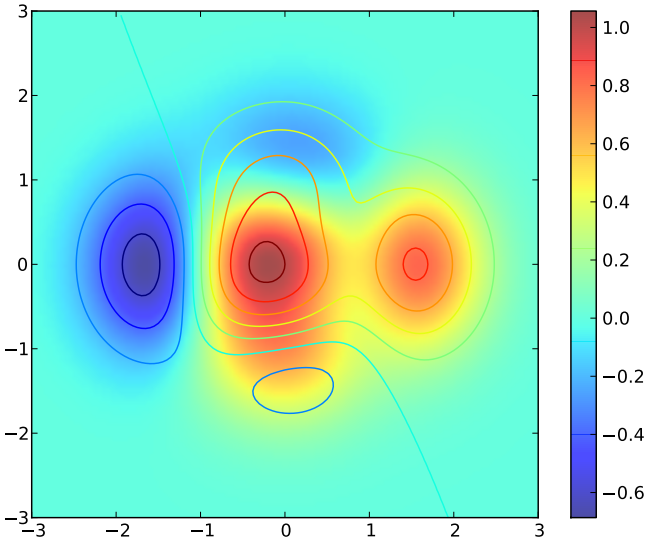
Subplot et axes

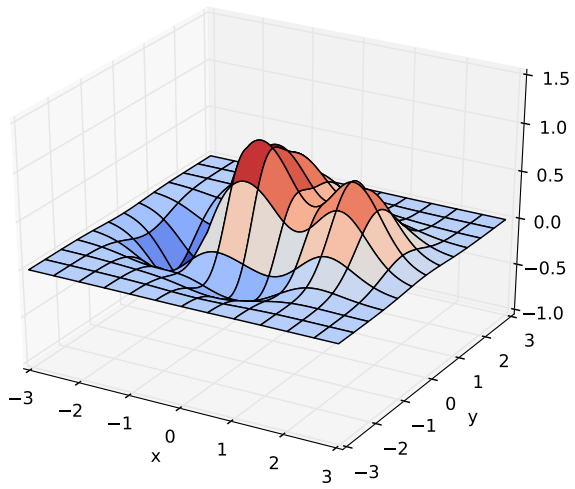


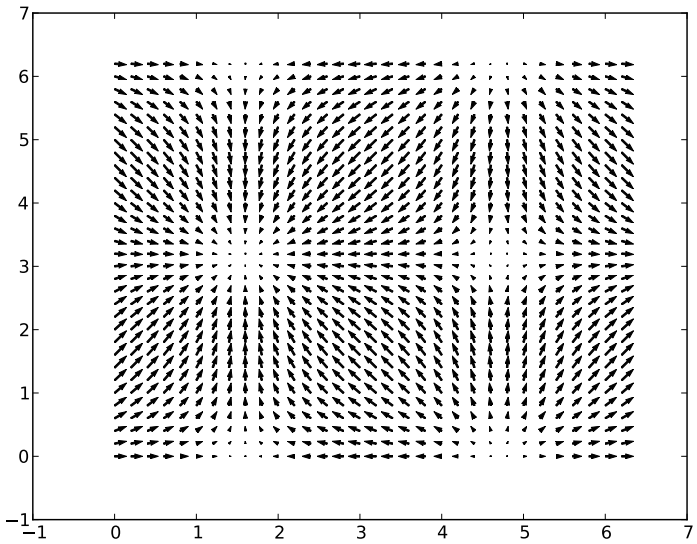


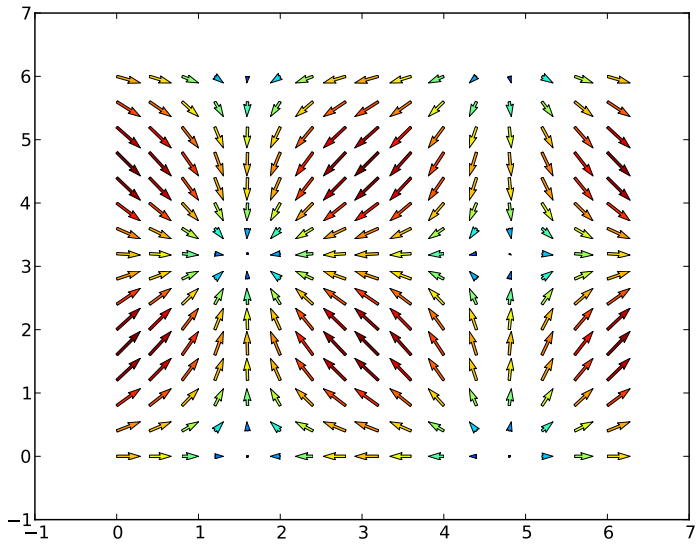
Subplot et axes

```
x = np.linspace(-4., 4., 50)
plt.subplot(2, 2, 1)
plt.plot(x, -x**4 + x**3 + x**2 + 1, 'o-')
plt.title("$f$")
plt.subplot(2, 2, 3)
plt.plot(x, -4*x**3 + 3*x**2 + 2*x, '-')
plt.title("$f^{(1)}$")
plt.subplot(1, 2, 2)
plt.plot(x, -12*x**2 + 6*x + 2, '--')
plt.title("$f^{(2)}$")
plt.axes([.7, .1, .1, .1])
plt.plot(x, -24*x + 6, ':')
plt.title("$f^{(3)}$")
plt.tight_layout()
plt.savefig("1D_subplot2.pdf")
plt.show()
```


3
2
1
0
−1
−2
−3
−3 −2 −1 0 1 2 3
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6


1.5
1.0
0.5
0.0
−0.5
−1.0
−3 −2 −1 0 1 2 3
x
−3 −2 −1 0 1 2 3
y


7
6
5
4
3
2
1
0
−1
−1 0 1 2 3 4 5 6 7


7
6
5
4
3
2
1
0
−1
−1 0 1 2 3 4 5 6 7

Les autres modules intéressants

- SymPy : calcul symbolique
- scikit-learn : machine learning
- pandas : structure de données et outils d'analyse de données haute performance
- pyTables : traitement de grandes données
- ...

Pour plus d'informations
http://numfocus.org/projects.html

Les interfaces

De nombreuses librairies écrites dans des langages bas niveau ont leur interface pour Python.

- mpi4py
- petsc4py
- FEniCS
- ...

Il y a également possibilité de faire du calcul sur carte graphique.

- PyCUDA
- PyOpenCL

Pour aller plus loin

Les formations organisées par le Groupe Calcul sur le sujet.

- Ecole Python de décembre 2010
  http://calcul.math.cnrs.fr/spip.php?rubrique85
- Ecole Python de décembre 2013
  http://calcul.math.cnrs.fr/spip.php?rubrique102