

Une pas si courte introduction au langage de programmation Python comme alternative à Matlab pour réaliser des calculs scientifiques ou d'autres applications.

G. Becq, N. Le Bihan

January 15, 2015

Description des paquets scientifiques

Distributions et Environnements de travail

Conclusion

# Outline

Description des paquets scientifiques


Distributions et Environnements de travail

Conclusion

# Principales Paquets Scientifiques

- ▶ Numpy : essentiel en calcul numérique.
- ▶ Matplotlib, Pylab : tracé de figures et fonctions à la matlab.
- ▶ Scipy : outils scientifiques spécifiques.
- ▶ Maiavi : 3D avancée.

# Numpy

- ▶  NumPy: <http://www.numpy.org>
- ▶ *NumPy is the fundamental package for scientific computing with Python*

```
>>> import numpy as np
>>> help(np)

Help on package numpy:

NAME
    numpy

FILE
    /Users/becqg/Library/Enthought/Canopy_64bit/User/lib/python2.7/site-packages
    /numpy/__init__.py

DESCRIPTION
    NumPy
    =====

    Provides
        1. An array object of arbitrary homogeneous items
        2. Fast mathematical operations over arrays
        3. Linear Algebra, Fourier Transforms, Random Number Generation

    ...
```

# N-dimensional array Object

- ▶ N-dimensional array : ndarray
- ▶ Création d'un tableau vide et réservation de l'espace (empty)
- ▶ Accès aux éléments :  $A[i, j, \dots]$

```
>>> A = numpy.empty((2, 2))
>>> print(A)
[[ -1.28822975e-231    2.68678092e+154]
 [  2.24497156e-314    2.24499315e-314]]
>>> A[0, 0] = 1
>>> A[1, 0] = 2
>>> A[0, 1] = 11
>>> A[1, 1] = 12
>>> print(A)
[[  1.   2.]
 [ 11.  12.]]
>>> type(A)
<type 'numpy.ndarray'>
```

# N-dimensional array Object

- Rappel : accès aux propriétés et méthodes (dir)

```
>>> dir(A)
'T', ..., 'all', 'any', 'argmax', 'argmin', 'argpartition', 'argsort', 'astype',
'base', 'byteswap', 'choose', 'clip', 'compress', 'conj', 'conjugate', 'copy',
'ctypes', 'cumprod', 'cumsum', 'data', 'diagonal', 'dot', 'dtype', 'dump',
'dumps', 'fill', 'flags', 'flat', 'flatten', 'getfield', 'imag', 'item',
'itemset', 'itemsize', 'max', 'mean', 'min', 'nbytes', 'ndim', 'newbyteorder',
'nonzero', 'partition', 'prod', 'ptp', 'put', 'ravel', 'real', 'repeat',
'reshape', 'resize', 'round', 'searchsorted', 'setfield', 'setflags', 'shape',
'size', 'sort', 'squeeze', 'std', 'strides', 'sum', 'swapaxes', 'take',
'tofile', 'tolist', 'tostring', 'trace', 'transpose', 'var', 'view'
```

# N-dimensional array Object

## Attributs sur la forme du tableau

- ▶ Forme du tableau (shape), c'est un tuple.
- ▶ Nombre de dimension (ndim)
- ▶ Type des éléments (dtype)
- ▶ Taille du tableau (size), c'est le nombre de cellules totales.

```
>>> A.shape
(2, 2)
>>> (nRow, nCol) = A.shape
>>> nRow = A.shape[0]
>>> nCol = A.shape[1]
>>> A.ndim
2
>>> A.dtype
dtype('float64')
>>> A.size
4
```



# N-dimensional array Object

## Changement de forme

- Pour changer la forme (reshape)
- Transposition (T)

```
>>> B = A.reshape((4, 1))
array([[ 1.],
       [ 2.],
       [11.],
       [12.]])
>>> B.ndim
2
>>> B.size
4
>>> B.T
array([[ 1.,  2., 11., 12.]])
```

# N-dimensional array Object

## Copie de tableaux

- ▶ Les éléments pointés sont liés, seule la forme change.
- ▶ Si on veut une copie (copy)

```
>>> B[0, 0] = 21
>>> print(A)
[[ 21.   2.]
 [ 11.  12.]]
>>> B[1, 0]
>>> C = A.copy()
>>> C[0, 0] = 31
>>> print(A[0,0], C[0,0])
(21.0, 31.0)
```

# N-dimensional array Object

## Création de tableaux

- ▶ tableau vide et réservation de l'espace (empty)
- ▶ initialisation à zeros (zeros)
- ▶ initialisation avec des uns (ones)
- ▶ tableau identité (eye) avec la dimension.
- ▶ à partir de listes (array)

```
>>> A = numpy.zeros((2, 4))
>>> print(A)
[[ 0.  0.  0.  0.]
 [ 0.  0.  0.  0.]]
>>> A = numpy.ones((3, 2))
>>> print(A)
[[ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]]
>>> A = numpy.eye(2)
>>> print(A)
[[ 1.  0.]
 [ 0.  1.]]
>>> A = numpy.array([[1, 2], [11, 12]])
>>> print(A)
[[ 1  2]
 [11 12]]
```

# N-dimensional array Object

## Types

- ▶ Définition du type à la création
- ▶ Changement de type (astype)
- ▶ Multiplication ou addition avec un scalaire typé.

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> print(A.dtype)
int64
>>> A = numpy.array([[1., 2], [11, 12]])
>>> print(A.dtype)
float64
>>> A = numpy.array([[1, 2], [11, 12]], dtype="float")
>>> print(A.dtype)
float64
>>> A = A.astype("complex")
>>> print(A)
[[ 1.+0.j  2.+0.j]
 [11.+0.j 12.+0.j]]
>>> A = numpy.array([[1, 2], [11, 12]]) * 1.
>>> print(A.dtype)
float64
```

# N-dimensional array Object

Additions, soustractions, multiplications sur les tableaux

- ▶ Addition, soustraction de matrices ou d'un scalaire (+, -)
- ▶ Multiplication par un scalaire (\*)
- ▶ Produit de matrices élément par élément (\*)

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[3, 4], [13, 14]])
>>> print(A + 10)
[[ 11.  12.]
 [ 21.  22.]]
>>> print(A + B)
[[ 4.   6.]
 [ 24.  26.]]
>>> print(A * 10)
[[ 10.  20.]
 [ 110. 120.]]
>>> print(A * B)
[[ 3.   8.]
 [ 143. 168.]]
>>> C = numpy.ones((10, ))
>>> print(A * C)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together with shapes (2,2) (10)
```

# N-dimensional array Object

## Produit scalaire

- ▶ Produit scalaire (dot)
- ▶ See also `numpy.dot` : en général, pour chaque méthode associée à un `ndarray`, il existe une fonction équivalente dans `numpy`.

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[3, 4], [13, 14]])
>>> print(A.dot(B))
[[ 29.  32.]
 [189. 212.]]
>>> print(numpy.dot(A, B))
[[ 29.  32.]
 [189. 212.]]
>>> C = numpy.ones((10, ))
>>> print(A.dot(C))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: matrices are not aligned
>>>
```

# N-dimensional array Object

## Autres méthodes

- ▶ nonzero, max, min ...
- ▶ sum, mean, std, cumsum, cumprod ...

```
>>> A = numpy.ones((2, 3, 4))
>>> print(A.cumsum())
[[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15.
 16. 17. 18. 19. 20. 21. 22. 23. 24.]
>>> print(A.cumsum(axis=0))
[[[ 1.  1.  1.  1.]
  [ 1.  1.  1.  1.]
  [ 1.  1.  1.  1.]]

 [[ 2.  2.  2.  2.]
  [ 2.  2.  2.  2.]
  [ 2.  2.  2.  2.]]]
print(A.cumsum(axis=1))
[[[ 1.  1.  1.  1.]
  [ 2.  2.  2.  2.]
  [ 3.  3.  3.  3.]]

 [[ 1.  1.  1.  1.]
  [ 2.  2.  2.  2.]
  [ 3.  3.  3.  3.]]]
>>> print(A.cumsum(2))
[[[ 1.  2.  3.  4.]
  [ 1.  2.  3.  4.]
  [ 1.  2.  3.  4.]]

 [[ 1.  2.  3.  4.]
  [ 1.  2.  3.  4.]
  [ 1.  2.  3.  4.]]]
```

# N-dimensional array Object

## Division

- ▶ Division par un scalaire (/)
- ▶ Division de matrices éléments par éléments (/)
- ▶ Attention au type !

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[3, 4], [13, 14]])
>>> print(A/2)
[[0 1]
 [5 6]]
>>> print(A / B)
[[0 0]
 [0 0]]
>>> print(A / B.astype(float))
[[ 0.33333333  0.5      ]
 [ 0.84615385  0.85714286]]
```



# Matrix

- ▶ classe héritée de ndarray avec  $\text{ndim} = 2$  avec propriétés spéciales.

```
>>> help(numpy.matrix)
class matrix(numpy.ndarray)
|   matrix(data, dtype=None, copy=True)
|
|   Returns a matrix from an array-like object, or from a string of data.
|   A matrix is a specialized 2-D array that retains its 2-D nature
|   through operations. It has certain special operators, such as "*"
|   (matrix multiplication) and "**" (matrix power).
...

>>> A = numpy.matrix([[1, 2], [11, 12]])
>>> print(A)
[[ 1  2]
 [11 12]]
>>> type(A)
<class 'numpy.matrixlib.defmatrix.matrix'>
>>> A = numpy.matrix("[1,2,3,4;11,12,13,14]")
>>> print(A)
[[ 1  2  3  4]
 [11 12 13 14]]
```

# Numpy

## Tableaux ND et fonctions associées

- Les exemples sont donnés dans IPython avec les fonctions Pylab chargées.

```
>>> A = array([[1,2,3],[4,5,6],[7,8,9]])
>>> whos

Variable      Type           Data/Info
-----
A             ndarray      3x3: 9 elems, type 'int64', 72 bytes

>>> A
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

>>> A.size
9

>>> A.shape
(3,3)

>>> B = array([[1,0,0],[0,1,0],[0,0,1]])

>>> A*B

array([[ 1.,  0.,  0.],
       [ 0.,  5.,  0.],
       [ 0.,  0.,  9.]])
```

# Numpy

## Tableaux ND et fonctions associées

- Méthodes
  - ▶ nonzero, max, min, mean, std ...
  - ▶ sum, cumprod, cumsum ...
  - ▶ reshape, resize, flatten, transpose ...
- Fonctions
  - ▶  $*$  : produit élt./élt.
  - ▶ *dot*(.,.) : produit matriciel
  - ▶ ...

# Numpy

## NDarray vs. matrix

```
>>> A = array([[1.,2,3],[4,5,6],[7,8,9]])
>>> B = array([[1,0,0],[0,1,0],[0,0,1]])
>>> C = matrix([[1.,2,3],[4,5,6],[7,8,9]])
>>> D = matrix([[1,0,0],[0,1,0],[0,0,1]])

>>> whos
Variable      Type          Data/Info
-----
A             ndarray      3x3: 9 elems, type 'float64', 72 bytes
B             ndarray      3x3: 9 elems, type 'int64', 72 bytes
C             matrix       [[ 1.  2.  3.]\n [ 4.  5.  6.]\n [ 7.  8.  9.]]
D             matrix       [[1 0 0]\n [0 1 0]\n [0 0 1]]

>>> dot(A,B)

array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [ 7.,  8.,  9.]])

>>> C*D

matrix([[ 1.,  2.,  3.],
        [ 4.,  5.,  6.],
        [ 7.,  8.,  9.]])
```

# Numpy

## matrix

- Méthodes
  - ▶ min, max, mean, std, ...
  - ▶ .T, .H, .I, ...
  - ▶ reshape, flatten, ...
- Fonctions
  - ▶ inv, svd, eig, ...

## Différences entre array et matrix

```
>>> A = array([[1.,2,3],[4,5,6],[7,8,9]])
>>> B = matrix([[1.,2,3],[4,5,6],[7,8,9]])

>>> A**2
array([[ 1.,  4.,  9.],
       [16., 25., 36.],
       [49., 64., 81.]])

>>> B**2
matrix([[ 30.,  36.,  42.],
        [ 66.,  81.,  96.],
        [102., 126., 150.]])
```

# Numpy

## NDarray vs. matrix

```
>>> A = array([[1.,2,3],[4,5,6],[7,8,9]])
>>> B = array([[1,0,0],[0,1,0],[0,0,1]])
>>> C = matrix([[1.,2,3],[4,5,6],[7,8,9]])
>>> D = matrix([[1,0,0],[0,1,0],[0,0,1]])

>>> whos
Variable      Type           Data/Info
-----
A             ndarray       3x3: 9 elems, type 'float64', 72 bytes
B             ndarray       3x3: 9 elems, type 'int64', 72 bytes
C             matrix        [[ 1.  2.  3.]\n [ 4.  5.  6.]\n [ 7.  8.  9.]]
D             matrix        [[1 0 0]\n [0 1 0]\n [0 0 1]]

>>> dot(A,B)

array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [ 7.,  8.,  9.]])

>>> C*D

matrix([[ 1.,  2.,  3.],
        [ 4.,  5.,  6.],
        [ 7.,  8.,  9.]])
```

# Numpy

## matrix

- Méthodes
  - ▶ min, max, mean, std, ...
  - ▶ .T, .H, .I, ...
  - ▶ reshape, flatten, ...
- Fonctions
  - ▶ inv, svd, eig, ...

## Différences entre array et matrix

```
>>> A = array([[1.,2,3],[4,5,6],[7,8,9]])
>>> B = matrix([[1.,2,3],[4,5,6],[7,8,9]])

>>> A**2
array([[ 1.,  4.,  9.],
       [16., 25., 36.],
       [49., 64., 81.]])

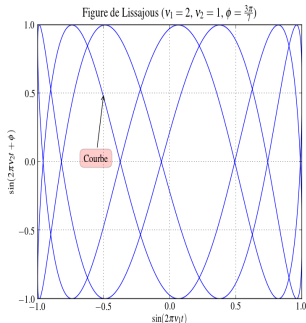
>>> B**2
matrix([[ 30.,  36.,  42.],
        [ 66.,  81.,  96.],
        [102., 126., 150.]])
```

# Matplotlib

## Librairie graphique

- ▶ Outils graphiques 2D
- ▶ Toolkits : Basemap, Axesgrid, mplot3d
- ▶ Beaucoup d'exemples : [www.matplotlib.org](http://www.matplotlib.org)

```
>>> t=arange(0,1,10e-5)
>>> S1=sin(2*pi*2*t);S2=sin(2*pi*7*t+(3*pi/7)),
>>> fig=figure()
>>> ax=fig.add_subplot(111,xlim=(-1,1),ylim=(-1,1))
>>> courbe=ax.plot(S1,S2);ax.grid('on')
>>> annotate('Courbe',xy=(-0.5,0.5),xytext=(-0.65,0),\
           bbox=dict(boxstyle='round,pad=0.5',fc='red',\
           alpha=0.2),arrowprops=dict(arrowstyle='->',\
           connectionstyle='arc3,rad=0'))
>>> title(r"Figure de Lissajous ($\nu_1=2$, $\nu_2=1$, $\phi=\frac{3\pi}{7}$)")
>>> xlabel(r"$\sin(2\pi\nu_1t)$")
>>> ylabel(r"$\sin(2\pi\nu_2t+\phi)$")
>>> show()
```

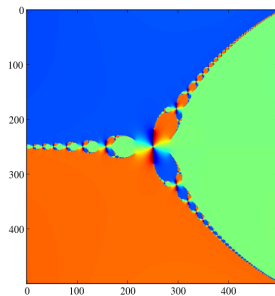




# Matplotlib

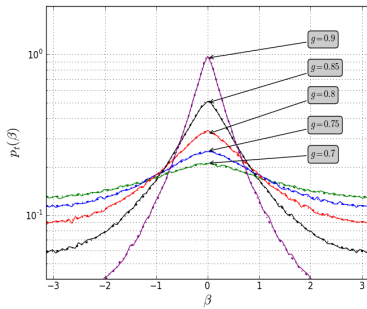
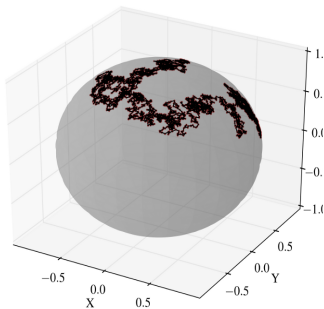
## Exemples

```
>>> [X,Y] = meshgrid(linspace(-2,2,500),\
linspace(-2,2,500))
>>> Z=X+Y*1j
>>> k=1
>>> while k<=75:
>>>     Z = Z - (Z/3 - 1) / (3*Z**2)
>>>     k=k+1
>>> close('all')
>>> imshow(angle(Z), cmap=cm.gray)
>>> #savefig('/MonChemin/Lenom.pdf',format='pdf')
>>> show()
```



# Matplotlib

## Exemples



# Pylab

- ▶ Module de Matplotlib : matplotlib/pylab.py
- ▶ Redéfinit des fonctions à la Matlab.
- ▶ Raccourci : "import pylab" au lieu de "import matplotlib.pylab"
- ▶ En général, et exceptionnellement, s'utilise : "from pylab import \*"

```
>>> from pylab import *
>>> plot([1, 5, 2, 4, 3])
[<matplotlib.lines.Line2D object at 0x4ec05b0>]
>>> show()
>>> a = randn(100, 10)
>>> type(a)
<type 'numpy.ndarray'>
>>> a.shape
(100, 10)
>>> help(matplotlib.pylab)
```

# Scipy

Scientific library

►  SciPy.org : <http://www.scipy.org>

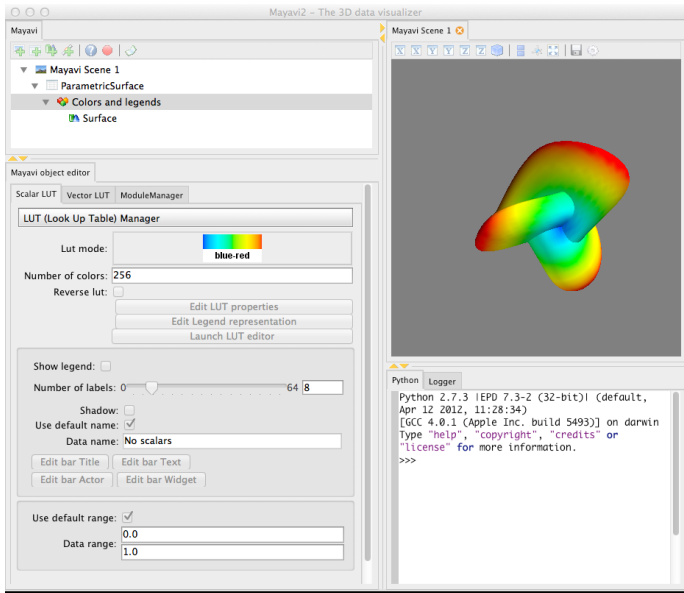
Clustering package  
Discrete Fourier transforms  
Interpolation  
Linear algebra  
Multi-dimensional image processing  
Optimization and root finding  
Sparse matrices  
Compressed Sparse Graph  
Routines  
Special functions  
Statistical functions for  
masked arrays

`scipy.cluster`  
`scipy.fftpack`  
`scipy.interpolate`  
`scipy.linalg`  
`scipy.ndimage`  
  
`scipy.optimize`  
  
`scipy.sparse`  
`scipy.sparse.csrgraph`  
  
`scipy.special`  
`scipy.stats.mstats`

Constants  
Integration and ODEs  
Input and output  
Miscellaneous routines  
Orthogonal distance regression  
Signal processing  
  
Sparse linear algebra  
Spatial algorithms and data  
structures  
Statistical functions  
C/C++ integration

`scipy.constants`  
`scipy.integrate`  
`scipy.io`  
`scipy.misc`  
`scipy.odr`  
  
`scipy.signal`  
  
`scipy.sparse.linalg`  
`scipy.spatial`  
  
`scipy.stats`  
`scipy.weave`

- Manipulation des objets 3D améliorée.



# Mayavi

import mayavi.engine

- Dans une console Python : import mayavi. ...

```
>>> from numpy import array
>>> from mayavi.api import Engine
>>> engine = Engine()
>>> engine.start()
>>> engine.new_scene()
>>> from mayavi.sources.parametric_surface import ParametricSurface
>>> parametric_surface1 = ParametricSurface()
>>> scene = engine.scenes[0]
>>> engine.add_source(parametric_surface1, scene)
>>> from mayavi.modules.surface import Surface
>>> surface1 = Surface()
>>> engine.add_filter(surface1, parametric_surface1)
```

# Importation de bibliothèques de fonction écrites en C

## Exemple using module ctypes

```
import ctypes

myLib = ctypes.LoadLibrary('myLib.lib')
fun_c = myLib.fun
fun_c.argtypes = [ctypes.c_double, ctypes.c_int]
fun_c.restype = ctypes.c_int (default)

n = len(s)
pathFile = os.path.dirname(__file__)
libName = os.path.join(pathFile, 'clz.lib')
print('libName + ' + libName)
libCLZ = ctypes.CDLL(libName)
clz_c = libCLZ.clz
clz_c.restype = ctypes.c_uint
sequence = numpy.ctypeslib.ndpointer(dtype=numpy.int)
clz_c.argtypes = ([sequence, ctypes.c_uint])
# conversion of s into sequence with numpy.asarray
c = clz_c(numpy.asarray(s, dtype='int'), n)
```