

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 19, 2015

Description des paquets scientifiques

Distributions et Environnements de travail

Conclusion

# Outline

Description des paquets scientifiques

Distributions et Environnements de travail

Conclusion

# Quelques Paquets et Outils Scientifiques







- ▶ SciPy : scientific python
- ▶ Mayavi : objets 3D avancés
- ▶ Scikit-learn : machine learning.
- ▶ ...

► <http://www.scipy.org/>

SciPy.org ENTHOUGHT

[Install](#) [Getting Started](#) [Documentation](#) [Report Bugs](#) [Blogs](#)

SciPy (pronounced "Sigh Pie") is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

	<b>NumPy</b> Base N-dimensional array package		<b>SciPy library</b> Fundamental library for scientific computing
	<b>Matplotlib</b> Comprehensive 2D Plotting		<b>IPython</b> Enhanced Interactive Console
	<b>Sympy</b> Symbolic mathematics		<b>pandas</b> Data structures & analysis


**ABOUT SCI-PIE**

- [About SciPy](#)
- [Install](#)
- [Getting Started](#)
- [Documentation](#)
- [Bug Reports](#)
- [Topical Software](#)
- [Citing](#)
- [Cookbook](#)
- [SciPy Central](#)
- [Wiki](#)
- [SciPy Conferences](#)
- [Blogs](#)
- [NumFOCUS](#)

**CORE PACKAGES:**

- [NumPy](#)
- [SciPy library](#)

# Numpy

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

```
>>> import numpy
>>> help(numpy)

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 de B sont les mêmes que ceux de A, 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)
- ▶ suivant une étendue (arange)

```
>>> 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]]
>>> print(numpy.arange(0.5, 1.7, 0.1))
[ 0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2  1.3  1.4  1.5  1.6]
```

# 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 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

## Division

- ▶ Division par un scalaire (/)
- ▶ Division éléments par éléments (/)
- ▶ Attention au type en Python 2.7 !

```
>>> 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]]
```



# N-dimensional array Object

## Autres méthodes

- max, min, sum, mean, std, cumsum, cumprod ... sur tous les éléments ou sur une dimension particulière (kwarg axis).

```
>>> A = numpy.ones((2, 3, 4))
>>> print(A)
[[[ 1.  1.  1.  1.]
  [ 1.  1.  1.  1.]
  [ 1.  1.  1.  1.]]

 [[ 1.  1.  1.  1.]
  [ 1.  1.  1.  1.]
  [ 1.  1.  1.  1.]]]
>>> 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

## Sélection de sous-tableaux

- découpage, slicing, comme pour les séquences.

```
>>> A = numpy.array([[1, 2, 3, 4], [11, 12, 13, 14]])
>>> print(A)
[[ 1  2  3  4]
 [11 12 13 14]]
>>> print(A[1, :])
[11 12 13 14]
>>> print(A[:, 1:3])
[[ 2  3]
 [12 13]]
```

# N-dimensional array Object

## Sélection de sous-tableaux

- ▶ Comparaison et opérateurs logiques
- ▶ Opérations logiques pour sélectionner des éléments (masking)
- ▶ Récupérer les indices (where)

```
>>> A = numpy.array([[1, 2, 3, 4], [11, 12, 13, 14]])
>>> print(A)
[[ 1  2  3  4]
 [11 12 13 14]]
>>> B = A > 2
>>> print(B)
[[False False  True  True]
 [ True  True  True  True]]
>>> print(A[B])
[ 3  4 11 12 13 14]
>>> indices = numpy.where(B)
>>> print(indices[0])
array([0, 0, 1, 1, 1, 1])
>>> print(indices[1])
array([2, 3, 0, 1, 2, 3])
>>> (i, j) = numpy.where(A > 2)
```

# N-dimensional array Object

## Concaténations

- ▶ Concaténation horizontale (hstack)
- ▶ Concaténation verticale (vstack)
- ▶ concatenate (concatenate, kwarg axis)

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[3, 4], [13, 14]])
>>> C = numpy.vstack((A, B))
>>> print(C)
[[ 1  2]
 [11 12]
 [ 3  4]
 [13 14]]
>>> D = numpy.hstack((A, B, A, A))
>>> print(D)
[[ 1  2  3  4  1  2  1  2]
 [11 12 13 14 11 12 11 12]]
>>> E = numpy.concatenate((A, B, A, A), axis=1)
>>> print(E)
[[ 1  2  3  4  1  2  1  2]
 [11 12 13 14 11 12 11 12]]
```

# Arrays

- ▶ Possibilité de mettre des éléments de types différents.
- ▶ Possibilité de tableaux structurés ...

```
>>> A = numpy.array([[ "a", 1], [ "b", 2]], dtype="object")
>>> print(A)
[[ 'a' 1]
 [ 'b' 2]]
>>> print(A.dtype)
object

>>> A = numpy.array([(1, "abc"), (2, "def")], dtype=[("index", "int"),
("name", "S8")])
>>> print(A)
[(1, 'abc') (2, 'def')]
>>> A["index"]
array([1, 2])
>>> A["name"]
array(['abc', 'def'],
      dtype='<S8')
```

# Sauvegarde et lecture de données

- ▶ Enregistrement d'un tableau (save) dans un fichier ".npy"
- ▶ Enregistrement compressé de plusieurs tableaux (savez) au format ".npz"
- ▶ Enregistrement (savetxt) et lecture (loadtxt) de fichier texte ".txt"
- ▶ Lecture (load) des fichiers ".npy", ".npz", ou ".txt"

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> numpy.save("save_A", A)
>>> del(A)
>>> A = numpy.load("save_A.npy")
>>> print(A)
[[ 1  2]
 [11 12]]
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[21, 22], [31, 32]])
>>> numpy.savez("save_AB", tab1=A, B=B)
>>> del(A, B)
>>> data = numpy.load("save_AB.npz")
>>> print(data["tab1"])
[[ 1  2]
 [11 12]]
>>> print(data["B"])
[[21 22]
 [31 32]]
```

```
>>> A = numpy.loadtxt("data.txt")
>>> A
```

# Matrix

## Définition

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

```
>>> 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).
...
```

# Matrix

## Saisie

- ▶ Saisie directe de type ndarray avec des listes imbriquées.
- ▶ Possibilité de saisie type Matlab.

```
>>> 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]]
```



# Matrix

## Multiplication et exposant

- ▶ Produit de matrices (\*)
- ▶ Exposant de matrice (\*\*)

```
>>> A = numpy.matrix([[1, 2], [11, 12]])  
>>> B = numpy.matrix([[3, 4], [13, 14]])  
>>> print(A * B)  
[[ 29  32]  
 [189 212]]  
>>> print(A ** 2)  
[[ 23  26]  
 [143 166]]
```

# Matrix

## Opérateurs matriciels courants

- ▶ Transposition (T)
- ▶ Inversion (I)
- ▶ Opérateur Hermitien (H)

```
>>> A = numpy.matrix([[1, 2], [11, 12]])
>>> print(A)
[[ 1  2]
 [11 12]]
>>> print(A.T)
[[ 1 11]
 [ 2 12]]
>>> print(A.I)
print(A.I)
[[-1.2  0.2]
 [ 1.1 -0.1]]
>>> B = numpy.matrix([[1, 2+1j], [11+1j, 12]])
>>> print(B)
[[ 1.+0.j  2.+1.j]
 [11.+1.j 12.+0.j]]
>>> print(B.H)
[[ 1.-0.j 11.-1.j]
 [ 2.-1.j 12.-0.j]]
```

# Sous paquet linalg

## Algèbre Linéaire

### ► Interface vers Lapack (numpy.linalg)

```
>>> help(numpy.linalg)
...
Linear algebra basics:

- norm          Vector or matrix norm
- inv           Inverse of a square matrix
- solve         Solve a linear system of equations
- det           Determinant of a square matrix
- lstsq         Solve linear least-squares problem
- pinv          Pseudo-inverse (Moore-Penrose)...
- matrix_power  Integer power of a square matrix

Eigenvalues and decompositions:

- eig           Eigenvalues and vectors of a square matrix
- eigh          Eigenvalues and eigenvectors of a Hermitian matrix
- eigvals       Eigenvalues of a square matrix
- eigvalsh      Eigenvalues of a Hermitian matrix
- qr            QR decomposition of a matrix
- svd           Singular value decomposition of a matrix
- cholesky      Cholesky decomposition of a matrix

Tensor operations:

- tensorsolve   Solve a linear tensor equation
- tensorinv     Calculate an inverse of a tensor
...
```

# Autres paquets de numpy

```
>>> help(numpy)
...
doc
    Topical documentation on broadcasting, indexing, etc.
lib
    Basic functions used by several sub-packages.
random
    Core Random Tools
linalg
    Core Linear Algebra Tools
fft
    Core FFT routines
polynomial
    Polynomial tools
testing
    Numpy testing tools
f2py
    Fortran to Python Interface Generator.
distutils
    Enhancements to distutils with support for
    Fortran compilers support and more.
...
```

# Autres paquets de numpy

## Random


- Sous paquet random : générateurs de nombres aléatoires.

```
>>> A = numpy.random.seed(0)
>>> A = numpy.random.randn(2, 3, 4)
>>> print(A)
[[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
  [ 1.86755799 -0.97727788  0.95008842 -0.15135721]
  [-0.10321885  0.4105985   0.14404357  1.45427351]]

 [[ 0.76103773  0.12167502  0.44386323  0.33367433]
  [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
  [-2.55298982  0.6536186   0.8644362  -0.74216502]]]
```

# Scipy library

## Librairie scientifique

- ▶  <http://www.scipy.org/scipylib/index.html>
- ▶ *It provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization.*

```
>>> import scipy
>>> help(scipy)
...
    cluster                --- Vector Quantization / Kmeans
    fftpack                --- Discrete Fourier Transform algorithms
    integrate              --- Integration routines
    interpolate             --- Interpolation Tools
    io                     --- Data input and output
    lib                    --- Python wrappers to external libraries
    lib.lapack              --- Wrappers to LAPACK library
    linalg                 --- Linear algebra routines
    misc                   --- Various utilities that don't have
                           another home.
    ndimage                --- n-dimensional image package
    odr                    --- Orthogonal Distance Regression
    optimize               --- Optimization Tools
    signal                 --- Signal Processing Tools
    sparse                 --- Sparse Matrices
    sparse.linalg           --- Sparse Linear Algebra
    ...
```

- Optimisation, Intégration, Interpolation, Algèbre linéaire, Algèbre linéaire creuse, Signal, Image, Statistiques, Fonctions spéciales ( $\Gamma$ ,  $\psi$ )...

```
...
sparse.linalg.dsolve          --- Linear Solvers
sparse.linalg.dsolve.umfpack  --- :Interface to the UMFPACK library:
                                Conjugate Gradient Method (LOBPCG)
sparse.linalg.eigen.lobpcg    --- Locally Optimal Block Preconditioned
                                Conjugate Gradient Method (LOBPCG) [*]

special                       --- Airy Functions [*]
lib.blas                      --- Wrappers to BLAS library [*]
sparse.linalg.eigen           --- Sparse Eigenvalue Solvers [*]
stats                         --- Statistical Functions [*]
lib                           --- Python wrappers to external libraries
                                [*]

lib.lapack                    --- Wrappers to LAPACK library [*]
integrate                     --- Integration routines [*]
ndimage                       --- n-dimensional image package [*]
linalg                        --- Linear algebra routines [*]
spatial                       --- Spatial data structures and algorithms
special                       --- Airy Functions
stats                         --- Statistical Functions
...
```

# Scipy


## lecture de fichiers Matlab

- ▶ Exemple, lectures de fichiers Matlab dans le subpackage io (scipy.io)

```
>>> help(scipy)
>>> import scipy.io
>>> data = scipy.io.loadmat('file.mat')
```



# Matplotlib

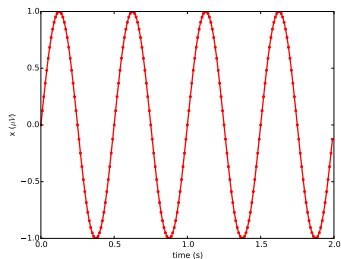
- ▶  **matplotlib**: <http://www.matplotlib.org>
- ▶ *matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.*
- ▶ Programmation orientée objets avec différents backends pour différents graphical user interfaces (GUI) : agg, gtk, qt, svg, ps, pdf...

# Matplotlib

## pyplot

- Fonctions procédurales dans le sous-paquet pyplot (matplotlib.pyplot)

```
>>> import matplotlib.pyplot
>>> t = numpy.arange(0, 10, 0.01)
>>> x = numpy.sin(2 * numpy.pi * 3 * t)
>>> matplotlib.pyplot.plot(t, x)
>>> matplotlib.pyplot.xlabel("time (s)")
>>> matplotlib.pyplot.ylabel("x ( $\mu$  V)")
>>> matplotlib.pyplot.show()
```



# Matplotlib

## Saving figures

- ▶ Sauvegarde manuelle à partir de la fenêtre ouverte sur l'icône save.
- ▶ sauvegarde en ligne de commande (savefig) sans passage par un affichage à l'écran.

```
...  
>>> matplotlib.pyplot.ylabel("x ( $\mu$  V)")  
>>> # matplotlib.pyplot.show()  
>>> matplotlib.pyplot.savefig("./sinus.ps")  
>>> matplotlib.pyplot.savefig("./sinus.pdf")  
>>> matplotlib.pyplot.savefig("./sinus.svg")  
>>> matplotlib.pyplot.savefig("./sinus.tiff")  
>>> matplotlib.pyplot.savefig("./sinus.png")  
>>> matplotlib.pyplot.savefig("./sinus.jpg")
```

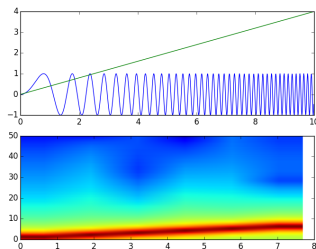
# Matplotlib

## Pylab

- ▶ Fonctions à la Matlab dans le sous-paquet pylab (matplotlib.pylab)
- ▶ Beaucoup de fonctions sous formes abrégées ...

```
>>> import matplotlib.pylab as mpl
>>> len(dir(mpl))
955

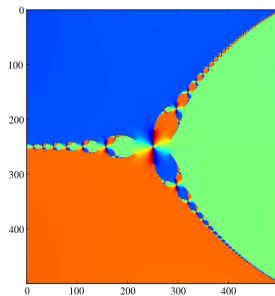
>>> from pylab import *
>>> t = arange(0, 10, 0.01)
>>> f = arange(0, 20, 0.02)
>>> x = sin(2 * pi * f * t)
>>> res = specgram(x, NFFT=16, Fs=100,
>>> show())
```



# Matplotlib

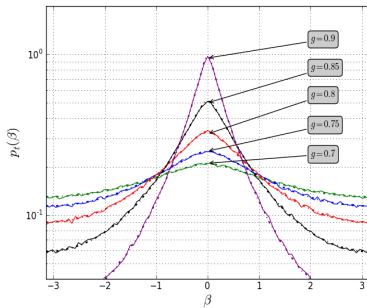
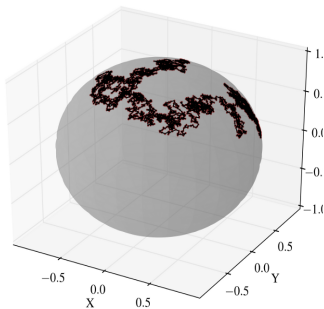
## Pylab

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



# Matplotlib

Exemples de Nicolas Le Bihan



# IPython

- ▶ **IP[y]:** IPython  
Interactive Computing  
<http://www.scipy.org/scipylib/index.html>
- ▶ *Enhanced python console*
- ▶ Attention, ce n'est pas un paquet mais une console améliorée !
- ▶ Accessible à partir d'un terminal (ipython)

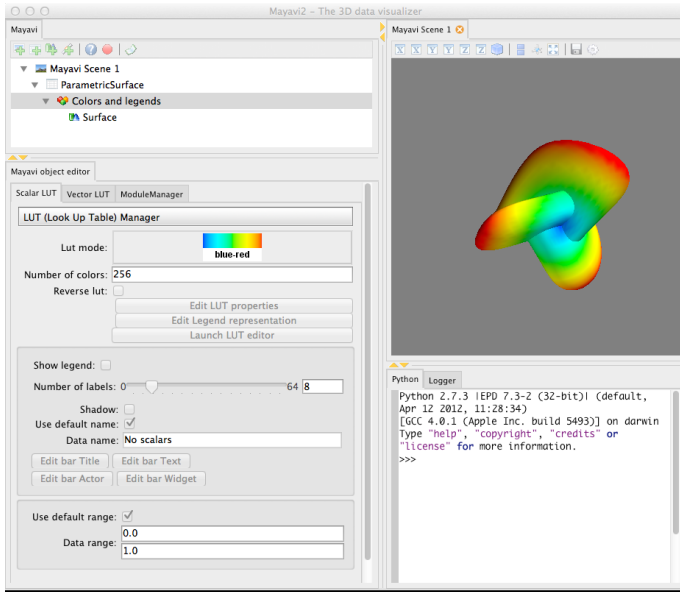
```
ipython
```

```
Python 2.7.3 | 64-bit | (default, Jun 14 2013, 18:17:36)
Type "copyright", "credits" or "license" for more information.

IPython 2.2.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]:
```

- 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)
```

# Scikit-learn

pas vu encore