

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 16, 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
>>> 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 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

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]]
```

N-dimensional array Object

Autres méthodes

- max, min, sum, mean, std, cumsum, cumprod ... sur tous les éléments ou sur un axe donné.

```
>>> 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

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])
```

N-dimensional array Object

Concaténations

- ▶ Concaténation horizontale (hstack)
- ▶ Concaténation verticale (vstack)
- ▶ Réduction (squeeze)

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> B = numpy.array([[3, 4], [13, 14]])
>>> C = numpy.hstack((A, B))
>>> print(C)
[[ 1  2  3  4]
 [11 12 13 14]]
>>> D = numpy.array([[[21, 22], [31, 32]], [[41, 42], [51, 52]]])
>>> 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])
```

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

► n

```
>>> A = numpy.array([[1, 2], [11, 12]])
>>> numpy.save("save_A", A)
>>> del(A)
>>> A = numpy.load("saveA.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]]
```

Matrix

- ▶ classe héritée de ndarray avec $\text{ndim} = 2$ et 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]]
```

Matrix

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

```
>>> help(numpy.matrix)
>>> 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



▶ numpy.random

```
>>> 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.
...
>>> 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]
```

Sauvegarde et lecture de données

- ▶ Un tableau (save) : ".npy"
- ▶ Plusieurs tableaux (savez) : ".npz"
- ▶ Fichier texte (loadtxt, savetxt)

```
>>> 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
array([[ 1.,  2.,  3.,  4.,  5.],
       [11., 12., 13., 14., 15.],
       [21., 22., 23., 24., 25.]])
>>> numpy.savetxt("data.txt", A)
```

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 (μV)")
>>> matplotlib.pyplot.show()
```

Matplotlib

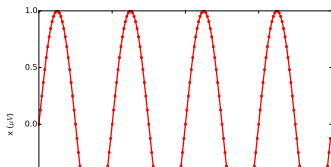
- ▶ Fonctions à la Matlab dans le sous-paquet pylab (matplotlib.pylab)
- ▶ Beaucoup de fonctions sous formes abrégées ...

```
>>> from matplotlib.pylab import *
>>> len(dir(mpl))
955
>>> x = sin(2 * numpy.pi * 3 * t)
>>> plot(t, x)
>>> xlabel("time (s)")
>>> ylabel("x (V)")
>>> show()
```

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.*
- ▶ fonction procédurale dans le sous-paquet pyplot
- ▶ programmation orientée objets avec différents backends pour différents graphical user interfaces (GUI) : agg, gtk, qt, svg, ps, pdf...

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



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.]])

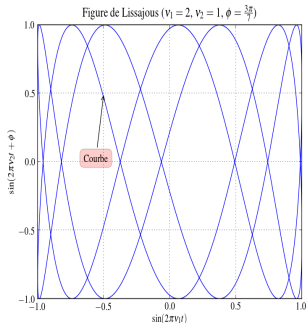
>>> B**2
matrix([[ 30.,  36.,  42.],
        [ 66.,  81.,  96.],
        [102., 126., 150.]])
```

Matplotlib

Librairie graphique

- ▶ Outils graphiques 2D
- ▶ Toolkits : Basemap, Axesgrid, mplot3d
- ▶ Beaucoup d'exemples : www.matplotlib.org

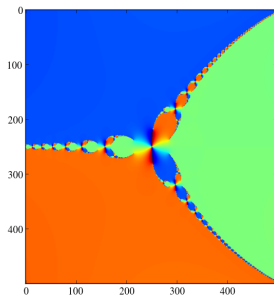
```
>>> t=arange(0,1,10e-5)
>>> S1=sin(2*pi*2*t);S2=sin(2*pi*7*t+(3*pi/7)),
>>> fig=figure()
>>> ax=fig.add_subplot(111,xlim=(-1,1),ylim=(-1,1))
>>> courbe=ax.plot(S1,S2);ax.grid('on')
>>> annotate('Courbe',xy=(-0.5,0.5),xytext=(-0.65,0),\
           bbox=dict(boxstyle='round,pad=0.5',fc='red',\
           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

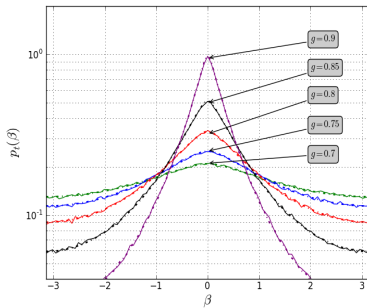
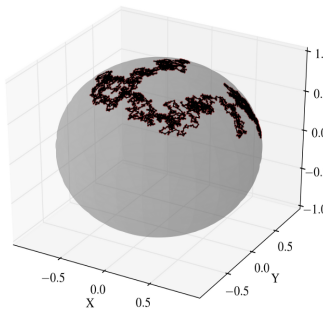
Exemples

```
>>> [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

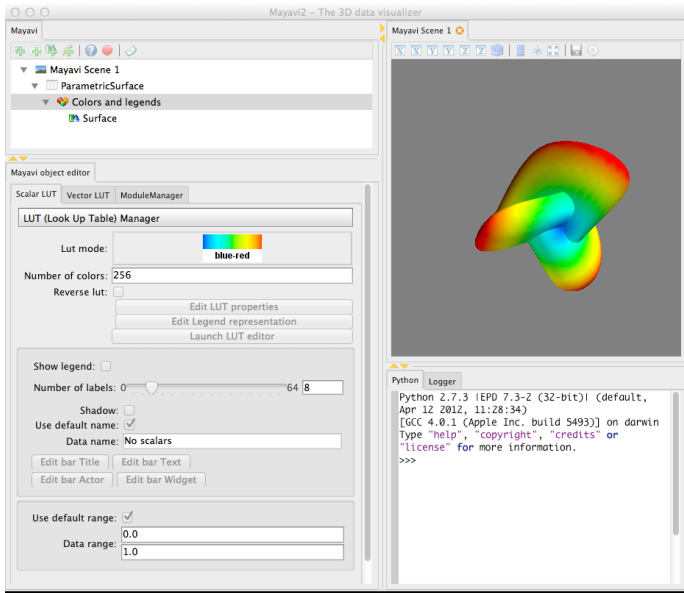
 scipy.sparse
 scipy.sparse.csgraph

 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)
```