

# SciPy

Python pour le calcul scientifique

Pierre Navaro

IRMA Strasbourg

IMFS le 29 juin 2011

- Le module SciPy contient de nombreux algorithmes : fft, algèbre linéaire, intégration numérique,...
- On peut voir ce module comme une extension de Numpy car il contient toutes ses fonctions.

```
>>> import numpy as np
>>> import scipy as sp
>>> np.sqrt(-1.)
nan
>>> sp.sqrt(-1.)
1j
>>> np.log(-2.)
nan
>>> sp.log(-2.)
(0.69314718055994529+3.1415926535897931j)
>>> sp.exp(sp.log(-2.))
(-2+2.4492935982947064e-16j)
```

- Subpackages
  - 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
  - signal : Signal processing
  - sparse : Sparse matrices and associated routines
- SciKits <http://scikits.appspot.com>

# Interpolation 1D

```
import scipy as sp
import pylab as pl
from scipy.interpolate import interp1d
x = sp.linspace(-1, 1, num=5) # 5 points régulièrement espacés entre -1 et 1.
y = (x-1.)*(x-0.5)*(x+0.5) # x et y sont des tableaux numpy
f0 = interp1d(x,y, kind='zero')
f1 = interp1d(x,y, kind='linear')
f2 = interp1d(x,y, kind='quadratic')
f3 = interp1d(x,y, kind='cubic')
f4 = interp1d(x,y, kind='nearest')
xnew = sp.linspace(-1, 1, num=40)
ynew = (xnew-1.)*(xnew-0.5)*(xnew+0.5)
pl.plot(x,y, 'D', xnew, f0(xnew), ':', xnew, f1(xnew), '-.',
        xnew, f2(xnew), '-.-', xnew, f3(xnew), 's--',
        xnew, f4(xnew), '--', xnew, ynew, linewidth=2)
pl.legend(['data', 'zero', 'linear', 'quadratic', 'cubic', 'nearest', 'exact'],
         loc='best')
pl.show()
```

L'interpolation est linéaire par défaut.

[http://www-irma.u-strasbg.fr/~navaro/scipy/scipy\\_interp1d.py](http://www-irma.u-strasbg.fr/~navaro/scipy/scipy_interp1d.py)

# Interpolation 2D

```
import scipy, pylab as sp, pl
from scipy.interpolate import interp2d
x,y=sp.mgrid[0:1:20j,0:1:20j] #create the grid 20x20
z=sp.cos(4*sp.pi*x)*sp.sin(4*sp.pi*y) #initialize the field
T1=interp2d(x,y,z,kind='linear')
T2=interp2d(x,y,z,kind='cubic')
T3=interp2d(x,y,z,kind='quintic')
X,Y=sp.mgrid[0:1:100j,0:1:100j] #create the interpolation grid 100x100
pl.figure(1)
pl.subplot(221) #Plot original data
pl.contourf(x,y,z)
pl.title('20x20')
pl.subplot(222) #Plot linear interpolation
pl.contourf(X,Y,T1(X[:,0],Y[0,:]))
pl.title('100x100 linear')
pl.subplot(223) #Plot cubic interpolation
pl.contourf(X,Y,T2(X[:,0],Y[0,:]))
pl.title('100x100 cubic')
pl.subplot(224) #Plot quintic interpolation
pl.contourf(X,Y,T3(X[:,0],Y[0,:]))
pl.title('100x100 quintic')
pl.show()
```

- Algorithmes de Transformées de Fourier rapides (FFT) pour le calcul de transformées de Fourier discrètes en dimension 1, 2 et n (une exponentielle complexe pour noyau et des coefficients complexes) : `fft`, `ifft` (inverse), `rfft` (pour un vecteur de réels), `irfft`, `fft2` (dimension 2), `ifft2`, `fftn` (dimension n), `ifftn`.
- Transformées en cosinus discrètes de types I, II et III (un cosinus pour noyau et des coefficients réels) : `dct`
- Produit de convolution : `convolve`

```
>>> from scipy.fftpack import *
>>> x=sp.arange (5)
>>> sp.all(abs(x-fft(ifft(x))))<1.e-15)
True
```

[http://www-irma.u-strasbg.fr/~navaro/scipy/scipy\\_ffft1d.py](http://www-irma.u-strasbg.fr/~navaro/scipy/scipy_ffft1d.py)

# Algèbre linéaire : scipy.linalg

- Solveurs linéaires, décompositions, valeurs propres. ( fonctions communes avec numpy).
- Des fonctions de matrices : expm, sinm, sinhm,...
- Matrices par blocs diagonales, triangulaires, circulantes,...

```
import scipy as sp
import scipy.linalg as spl
b=sp.ones(5)
A=sp.array([[1.,3.,0., 0.,0.],
            [ 2.,1.,-4, 0.,0.],
            [ 6.,1., 2,-3.,0.],
            [ 0.,1., 4.,-2.,-3.],
            [ 0.,0., 6.,-3., 2.]])
print "x=",spl.solve(A,b,sym_pos=False) # LAPACK ( gesv ou posv si matrice sym)
AB=sp.array([[0.,3.,-4.,-3.,-3.],
            [1.,1., 2.,-2., 2.],
            [2.,1., 4.,-3., 0.],
            [6.,1., 6., 0., 0.]])
print "x=",spl.solve_banded((2,1),AB,b) # LAPACK ( gbsv )
P,L,U=spl.lu(A) # written for scipy
```

[http://www-irma.u-strasbg.fr/~navaro/scipy/scipy\\_linalg.py](http://www-irma.u-strasbg.fr/~navaro/scipy/scipy_linalg.py)

# Stockage CSC des matrices creuses

## Compressed Sparse Column format

```
import scipy.sparse as spsp
row = sp.array([0,2,2,0,1,2])
col = sp.array([0,0,1,2,2,2])
data = sp.array([1,2,3,4,5,6])
Mcsc1 = spsp.csc_matrix((data,(row,col)),shape=(3,3))
indptr = sp.array([0,2,3,6])
indices = sp.array([0,2,2,0,1,2])
data = sp.array([1,2,3,4,5,6])
Mcsc2 = spsp.csc_matrix ((data,indices,indptr),shape=(3,3))
print Mcsc1.todense(), Mcsc2.todense()
```

```
[[1 0 4]
 [0 0 5]
 [2 3 6]]
```

- Les opérations matricielles sont optimisées
- Le "slicing" suivant les colonnes est efficace.
- produit matrice vecteur efficace.
- La conversion dans d'autres formats peut être coûteuse.



## Compressed Sparse Row format

```
row = sp.array([0,0,1,2,2,2])
col = sp.array([0,2,2,0,1,2])
data = sp.array([1,2,3,4,5,6])
Mcsr1 = spsp.csr_matrix((data,(row,col)),shape=(3,3))
indptr = sp.array([0 ,2 ,3 ,6])
indices = sp.array([0 ,2 ,2 ,0 ,1 ,2])
data = sp.array([1,2,3,4,5,6])
Mcsr2=spsp.csr_matrix((data,indices,indptr),shape=(3,3))
print Mcsr1.todense (), Mcsr2.todense()
```

```
[[1 0 2]
 [0 0 3]
 [4 5 6]]
```

- Les opérations matricielles sont optimisées
- Le "slicing" suivant les lignes est efficace.
- produit matrice vecteur efficace.
- La conversion dans d'autres formats peut être coûteuse.

## Block Sparse Row format

*#exemple au format BSR*

```
indptr = sp.array([0,2,3,6])
indices = sp.array([0,2,2,0,1,2])
data = sp.array([1,2,3,4,5,6]).repeat(4)
print data
data = data.reshape(6,2,2) # 6 blocs de taille 2x2
Mbsr = spsp.bsr_matrix((data,indices,indptr),shape=(6,6))
print Mbsr.todense()
```

```
[[1 1 0 0 2 2]
 [1 1 0 0 2 2]
 [0 0 0 0 3 3]
 [0 0 0 0 3 3]
 [4 4 5 5 6 6]
 [4 4 5 5 6 6]]
```

- Format approprié pour des matrices creuses à blocs denses.
- Très proche du format csr.
- Peut permettre une accélération des opérations arithmétiques et des produits matrices vecteurs.

## Block Sparse Row format

*#exemple au format BSR*

```
indptr = sp.array([0,2,3,6])
indices = sp.array([0,2,2,0,1,2])
data = sp.array([1,2,3,4,5,6]).repeat(4)
data = data.reshape(6,2,2) # 6 blocs de taille 2x2
Mbsr = spsp.bsr_matrix((data,indices,indptr),shape=(6,6))
print Mbsr.todense()
```

```
[[1 1 0 0 2 2]
 [1 1 0 0 2 2]
 [0 0 0 0 3 3]
 [0 0 0 0 3 3]
 [4 4 5 5 6 6]
 [4 4 5 5 6 6]]
```

- Format approprié pour des matrices creuses à blocs denses.
- Très proche du format csr.
- Peut permettre une accélération des opérations arithmétiques et des produits matrices vecteurs.

# Les formats dédiés a l'assemblage

- `lil_matrix` : Row-based linked list matrix. Format agréable pour l'assemblage mais il faut convertir dans un autre format avant de calculer.
- `dok_matrix` : A dictionary of keys based matrix. Format idéal pour un assemblage incrémental et la conversion vers un autre format est efficace.
- `coo_matrix` : coordinate list format. Conversion tres rapide vers les formats CSC/CSR.

<http://docs.scipy.org/doc/scipy/reference/sparse.html>

# Matrice avec un stockage diagonal : dia\_matrix

```
>>> import scipy as sp
>>> import scipy.sparse as spsp
>>> data = sp.array ([[1 ,2 ,3 ,4]])
>>> print data.repeat (3)
[1 1 1 2 2 2 3 3 3 4 4 4]
>>> print data.repeat (3 , axis=0)
[[1 2 3 4]
 [1 2 3 4]
 [1 2 3 4]]
>>> data=data.repeat(3,axis=0)
>>> offsets=sp.array([0, -1 ,2])
>>> Mdia=spsp.dia_matrix((data,offsets),shape=(4,4))
>>> Mdia.todense ()
matrix([[1, 0, 3, 0],
        [1, 2, 0, 4],
        [0, 2, 3, 0],
        [0, 0, 3, 4]])
>>> Mdia
<4x4 sparse matrix of type '<type 'numpy.int64'>'
with 9 stored elements (3 diagonals) in DIAgonal format>
```

# Matrices diagonales : identity, eye

```
>>> Id=spsp.identity(3)
>>> Id
<3x3 sparse matrix of type '<type 'numpy.float64''>'
with 3 stored elements in Compressed Sparse Row format>
>>> Id.todense()
matrix([[ 1.,  0.,  0.],
        [ 0.,  1.,  0.],
        [ 0.,  0.,  1.]])
>>> Idmn=spsp.eye (3 ,4)
>>> Idmn.todense ( )
matrix([[ 1.,  0.,  0.,  0.],
        [ 0.,  1.,  0.,  0.],
        [ 0.,  0.,  1.,  0.]])
>>> Idmn
<3x4 sparse matrix of type '<type 'numpy.float64''>'
with 3 stored elements (1 diagonals) in DIAgonal format>
```

# Matrices diagonales : spdiags, triu

```
>>> data=sp.array ([[10 ,20 ,30 ,40] ,[1 ,2 ,3 ,4] ,[100 ,200 ,300 ,400]])
>>> diags=sp.array ([ -1 ,0 ,2])
>>> M=spsp.spdiags(data, diags, 4, 4)
>>> M.todense()
matrix([[ 1,  0, 300,  0],
        [10,  2,  0, 400],
        [ 0, 20,  3,  0],
        [ 0,  0, 30,  4]])
>>> print spsp.isspmatrix_csc(M), spsp.isspmatrix_dia(M)
False True
>>> spsp.triu(M).todense()
matrix([[ 1,  0, 300,  0],
        [ 0,  2,  0, 400],
        [ 0,  0,  3,  0],
        [ 0,  0,  0,  4]])
>>> r,c,d=spsp.find(M)
>>> print r,c,d
[0 0 1 1 1 2 2 3 3] [0 2 0 1 3 1 2 2 3] [ 1 300  10  2 400  20  3 30  4]
```

[http://www-irma.u-strasbg.fr/~navaro/scipy/scipy\\_sparse\\_diag.py](http://www-irma.u-strasbg.fr/~navaro/scipy/scipy_sparse_diag.py)

- `speigen`, `speigen_symmetric`, `lobpcg` pour le calcul de valeurs et vecteurs propres (ARPACK).
- `svd` pour une décomposition en valeurs singulières (ARPACK).
- Méthodes directes (UMFPACK si présent ou SUPERLU) ou itératives (<http://www.netlib.org/templates/>) pour la résolution de  $Ax = b$ . Le format `csc` ou `csr` est conseillé.
  - `spsolve` pour les non initiés
  - `dsolve` ou `isolve` pour les moyennement initiés
  - Pour les initiés : méthodes directes `splu` et `spilu` ; méthodes itératives `cg`, `cgs`, `bicg`, `bicgstab`, `gmres`, `lgmres` et `qmr`.
- Algorithmes de minimisation : `lsqr` et `minres`

[http://www-irma.u-strasbg.fr/~navaro/scipy/scipy\\_sparse\\_linalg.py](http://www-irma.u-strasbg.fr/~navaro/scipy/scipy_sparse_linalg.py)

<http://docs.scipy.org/doc/scipy/reference/sparse.linalg.html>



# Matrices creuses : LinearOperator

Pour des méthodes itératives telles que `cg`, `gmres`, il n'est pas nécessaire de connaître la matrice du système, le produit matrice vecteur est suffisant. La classe `LinearOperator` permet d'utiliser ces méthodes sans leur donner la matrice du système mais en leur donnant l'opérateur permettant de faire le produit matrice vecteur.

```
>>> import scipy as sp
>>> import scipy.sparse.linalg as spspl
>>> def mv(v):
...     return sp.array([2*v[0],3*v[1]])

>>> A=spspl.LinearOperator((2 ,2),matvec=mv,dtype=float )
>>> A
<2x2 LinearOperator with dtype=float64>
>>> A*sp.ones(2)
array([ 2.,  3.])
>>> A.matmat(sp.array([[1,-2],[3,6]]))
array([[ 2, -4],
       [ 9, 18]])
```

[http://www-irma.u-strasbg.fr/~navaro/scipy/scipy\\_exemples.py](http://www-irma.u-strasbg.fr/~navaro/scipy/scipy_exemples.py)

# Matrices creuses : Factorisation LU

```
>>> import scipy as sp
>>> import scipy.linalg as spl
>>> import scipy.sparse as spsp
>>> import scipy.sparse.linalg as spspl
>>> N=50
>>> un=sp.ones(N)
>>> w=sp.rand(N+1)
>>> A=spsp.spdiags([w[1:], -2*un, w[:-1]], [-1,0,1], N, N)
>>> A=A.tocsc()
>>> b = un
>>> op=spspl.splu(A)
>>> print op
<factored_lu object at 0x523d6b0>
>>> x=op.solve(b)
>>> spl.norm(A*x-b)
2.112343365827712e-15
```

[http://www-irma.u-strasbg.fr/~navaro/scipy/scipy\\_lu.py](http://www-irma.u-strasbg.fr/~navaro/scipy/scipy_lu.py)

# Matrices creuses : Gradient conjugué

*à la suite du script précédent...*

```
>>> global k
>>> k=0
>>> def f(xk):
...     global k
...     print "iteration ",k," residu=",spl.norm(A*xk-b)
...     k=k+1
>>>
>>> x,info=spspl.cg(A,b,x0=sp.zeros(N),tol=1.0e-12,maxiter=N,M=None,callback=f)
iteration 0 residu= 2.71695646088
iteration 1 residu= 1.23113341645
iteration 2 residu= 0.669850704913
iteration 3 residu= 0.268084915616
...
iteration 24 residu= 5.2476767966e-11
iteration 25 residu= 1.34905427044e-11
iteration 26 residu= 5.41798161014e-12
```

[http://www-irma.u-strasbg.fr/~navaro/scipy/scipy\\_cg.py](http://www-irma.u-strasbg.fr/~navaro/scipy/scipy_cg.py)

# Matrices creuses : Gradient conjugué avec préconditionneur

```
>>> pc=spspl.spilu(A,drop_tol=0.1)
>>> xp=pc.solve(b)
>>> spl.norm(A*xp-b)
0.36434925345295482
>>> def mv(v):
...     return pc.solve(v)
>>> lo = spspl.LinearOperator((N,N),matvec=mv)
>>> k=0
>>> x,info=spspl.cg(A,b,x0=sp.zeros(N),tol=1.e-12,maxiter=N,M=lo,callback=f)
iteration 0  residu= 0.327201220193
iteration 1  residu= 0.00973829257822
iteration 2  residu= 0.000619041802146
iteration 3  residu= 1.76504773146e-05
iteration 4  residu= 7.28076292035e-07
iteration 5  residu= 3.35185106542e-08
iteration 6  residu= 2.16425999372e-09
iteration 7  residu= 7.15477674159e-11
iteration 8  residu= 3.67681615701e-12
```

[http://www-irma.u-strasbg.fr/~navaro/scipy/scipy\\_pc.py](http://www-irma.u-strasbg.fr/~navaro/scipy/scipy_pc.py)

- Intégration numérique de fonctions : `quad`, `dblquad`, `tplquad`, ... Les intégrales doivent être définies. La librairie Fortran utilisée est QUADPACK.

```
>>> import scipy.integrate as spi
>>> x2=lambda x: x**2
>>> x2(4)
16
>>> spi.quad(x2,0.,4.)
(21.333333333333336, 2.368475785867001e-13)
>>> 4.**3/3
21.333333333333332
```

- Intégration numérique de données discrètes : `trapz` (dans Numpy), `simps`,...

# Intégration : odeint

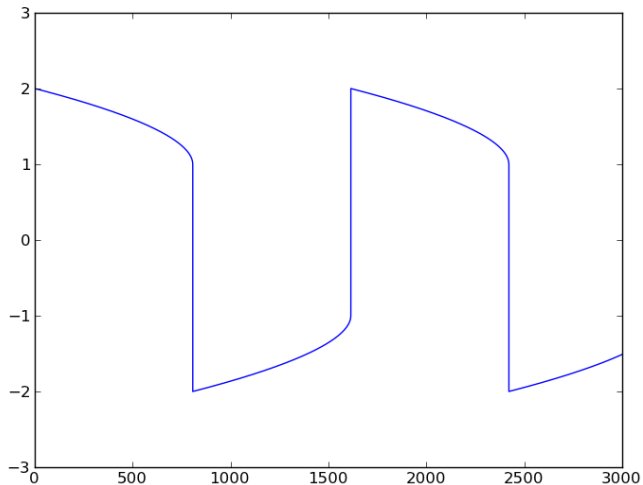
Résolution d'équations aux dérivées ordinaires. Utilise `lsoda` de la librairie Fortran ODEPACK. Exemple : l'oscillateur de van der Pol

$$\begin{aligned}y_1'(t) &= y_2(t), \\y_2'(t) &= 1000(1 - y_1^2(t))y_2(t) - y_1(t)\end{aligned}$$

avec  $y_1(0) = 2$  et  $y_2(0) = 0$ .

```
>>> def vdp1000(y,t):
...     dy=sp.zeros(2)
...     dy[0]=y[1]
...     dy[1]=1000.*(1.-y[0]**2)*y[1]-y[0]
...     return dy
>>> t0=0
>>> tf=3000
>>> N=300000
>>> dt=(tf-t0)/N
>>> tgrid=sp.linspace(t0,tf,num=N)
>>> y=spi.odeint(vdp1000,[2.,0.],tgrid)
>>> import pylab as plt
>>> plt.plot(tgrid,y[:,0])
[<matplotlib.lines.Line2D object at 0x7447230>]
>>> plt.show()
```

# Intégration : odeint



[http://www-irma.u-strasbg.fr/~navaro/scipy/scipy\\_odeint.py](http://www-irma.u-strasbg.fr/~navaro/scipy/scipy_odeint.py)

- Non abordé : Optimisation
- Les sous-paquets cluster, constants, io, maxentropy, misc, ndimage, odr, signal, spatial, stats, weave.
- <http://www.scipy.org/>
- Les SciKits : [www.scipy.org/scipy/scikits](http://www.scipy.org/scipy/scikits).
- Transparents de Sylvain Faure  
[http://calcul.math.cnrs.fr/Documents/Ecoles/2010/cours\\_Python\\_Scipy.pdf](http://calcul.math.cnrs.fr/Documents/Ecoles/2010/cours_Python_Scipy.pdf)