NumPy

Python pour le calcul scientifique

Pierre Navaro

IRMA Strasbourg

IMFS le 28 juin 2011

NumPy

- Le module incontournable du calcul scientifique avec Python.
- Héritier de Numeric et Numarray.
- Classes de base pour SciPy.
- S'installe comme un module Python standard.
- Contient les fonctions de manipulation des tableaux pour le calcul numérique.
- Bibliothèque mathématique importante.
- API pour l'encapsulation de codes Fortran, C/C++.

Le tableau NumPy ndarray

- Collection indexable et contigüe d'éléments de même type
- Implémentation avec un vrai tableau en mémoire optimisé pour les performances
- Manipulation similaire à tout autre objet Python
- Multi-dimensionnel pour tous les types de données
- Les dimensions et parcours sont modifiables, les indexations souples
- Permet l'encapsulation de tableaux fortran.
- Gestion possible de l'interface avec des programmes Fortran/C/C++

```
>> import numpy as np
```

Le tableau NumPy ndarray

Tableau de taille fixée, multidimensionnel contenant des élements de méme type et de même taille. Ce tableau peut être définit par :

- son contenu sous forme d'une liste de valeurs :

```
a=np.array([1,3,5,7,9,11,13,17])
```

- ses dimensions et son profil :

```
b=np.array([[1,2,3],[4,5,6]])
```

- son type d'élément :

```
a=np.array([0.1, 0.0, 0.2],dtype='f')
b=np.array([[1,2,3],[4,5,6]],dtype='i')
```

http://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html

Création d'un tableau numpy

- Fonction de ces indices

```
>>> def initfunction(i,j):
...     return 100+10*i+j
>>> c=numpy.fromfunction(initfunction,(5,3))
array([[ 100., 101., 102.],
       [ 110., 111., 112.],
       [ 120., 121., 122.],
       [ 130., 131., 132.],
       [ 140., 141., 142.]])
```

- A partir d'un fichier

```
>>> import numpy
>>> a=numpy.ones((3,5,7))
>>> numpy.save("data.npy",a)
>>> b=numpy.load("data.npy")
```

Les tableaux numpy structurés (Record arrays)

Numpy propose des fonctions pour créer facilement des tableaux.

```
>>> x = np.zeros((2,),dtype=('i4,f4,a10'))
>>> x[:] = [(1,2.,'Hello'),(2,3.,"World")]
>>> x
array([(1, 2.0, 'Hello'), (2, 3.0, 'World')],
      dtype=[('f0', '>i4'), ('f1', '>f4'), ('f2', '|S10')])
```

Dans cet exemple, nous avons créé un tableau 1D de 2 éléments. Chaque élément possède trois composantes dont les types sont entier 32 bits, flottant 32 bits et une chaine de 10 caractères.

```
>>> x[1]
(2,3.,"World")
```

`empty, empty_like, ones, ones_like, zeros, zeros_like`

```
>>> N=10
>>> import numpy
>>> A = numpy.zeros(4*N*N,dtype=numpy.double).reshape(2*N,2*N)
>>> B = numpy.zeros((2*N,2*N),dtype=numpy.double)
```

`http://docs.scipy.org/doc/numpy/reference/routines.array-creation.html`

Identifier un type

- Diverses fonctions pour construire et manipuler les types
- La définition des types permet portabilité et interopérabilité.

```
>>> a=array(range(4))
>>> a
array([0, 1, 2, 3])
>>> a.dtype
dtype('int32')
>>> a.dtype.char
'l'
>>> b=array(a,dtype='d')
>>> b
array([ 0.,  1.,  2.,  3.])
>>> b.dtype
dtype('float64')
```

http://docs.scipy.org/doc/numpy/reference/arrays.dtypes.html

Manipulation de profil

- `flat` : vue 1D d'un tableau sans modification

```
>>> a = indices((2,5))
>>> a
array([[[0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1]],

       [[0, 1, 2, 3, 4],
        [0, 1, 2, 3, 4]]])
>>> a.flat[:]
array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4])
```

- `shape` : n-uplet des dimensions d'un tableau

```
>>> a=ones((3,5,7))
>>> a.shape
(3,5,7)
>>> a.shape=(21,5)
>>> shape(a)
(21,5)
```

- La méthode `reshape`

```
>>> a=arange(105).reshape((3,5,7))
```

Transposition

```
>>> a
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7]],
       [[ 8,  9, 10, 11],
        [12, 13, 14, 15]],
       [[16, 17, 18, 19],
        [20, 21, 22, 23]]])
>>> a.T
array([[[ 0,  8, 16],
        [ 4, 12, 20]],
       [[ 1,  9, 17],
        [ 5, 13, 21]],
       [[ 2, 10, 18],
        [ 6, 14, 22]],
       [[ 3, 11, 19],
        [ 7, 15, 23]]])
>>> b=a.T
```

Retourne une vue du tableau, pensez à la copie !. Existe aussi la fonction :

```
>>> a.transpose()
>>> transpose(a)
```

Indexation

```
>>> a=arange(24).reshape(2,3,4)
>>> a
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

- Pour un élément dans le tableau.

```
>>> a[0][2][1]
9
```

- Syntaxe avec implémentation optimisée pour l'accès

```
>>> a[0,:-1]
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

- La syntaxe fonctionne pour la référence et l'assignation

```
>>> b=a[0:2]
>>> a[0:2]=9
```

Méthodes

- Associée au module

```
>>> import numpy
>>> numpy.finfo(numpy.float32).eps
1.1920929e-07
```

- Associée à l'objet ndarray

```
>>> a=arange(6).reshape(3,2)
>>> 1+sin(a)
array([[ 1.        ,  1.84147098],
       [ 1.90929743,  1.14112001],
       [ 0.2431975 ,  0.04107573]])
>>> a>3
array([[False, False],
       [False, False],
       [ True,  True]], dtype=bool)
```

- Appliquée sur la totalité ou une partie du tableau .

```
>>> a.max(), a.sum()
(5, 15)
>>> a.sum(axis=1)
array([1, 5, 9])
```

Copie ou référence

- Copie explicite

```
b=numpy.copy(a)
b=array(a, copy=True)
```

- Copie implicite et vue

```
>>> a=array([5,3,6,1,6,7,9,0,8])
>>> sort(a)
array([0, 1, 3, 5, 6, 6, 7, 8, 9])
>>> a
array([5, 3, 6, 1, 6, 7, 9, 0, 8])
>>> a.sort()
>>> a
array([0, 1, 3, 5, 6, 6, 7, 8, 9])
```

- Vue

```
>>> b=a
>>> b.shape=(3,3)
>>> a[0]=-1
>>> a.T
array([[-1,  5,  7],
       [-1,  6,  8],
       [-1,  6,  9]])
>>> b
array([[-1, -1, -1],
       [ 5,  6,  6],
       [ 7,  8,  9]])
```

- Itérateur

```
>>> b.flat[4]
6
```

Ufuncs : Universal functions

- Opère sur un tableau élément par élément. Exemple : `add(a,b)` avec a et b ndarrays est plus performant qu'une boucle
- Vectorisable : Une fonction prend des vecteurs en entrée et produit un vecteur en sortie. L'utilisateur peut utiliser l'API pour créer ses propres ufuncs
- Syntaxe compacte sans boucle $\Rightarrow$ les ufuncs sont parfois difficiles à lire.

```
>>> x=arange(10)
>>> x[(x**3-9*x**2+23*x-15)==0]
array([1, 3, 5])
>>> sqrt(x)
array([ 0.        ,  1.        ,  1.41421356,  1.73205081,  2.        ,
        2.23606798,  2.44948974,  2.64575131,  2.82842712,  3.        ])
```

`http://docs.scipy.org/doc/numpy/reference/ufuncs.html`

Matrices

- Dérivée de ndarray. Toute **matrix** est au moins un ndarray
- Utilisée dans numpy.linalg
- Toujours 2D, attention un slice de Matrix est en 2D
- Certaines méthodes de Matrix masquent les méthodes équivalentes de ndarray

```
>>> a=arange(9).reshape(3,3)
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> b=matrix(a)
matrix([[0, 1, 2],
        [3, 4, 5],
        [6, 7, 8]])
>>> b[0]
matrix([[0, 1, 2]])
>>> a[0]
array([0, 1, 2])
```

Random : numpy.random

Fonctions adaptées au ndarrays pour tous les types d'éléments

```
>>> a=numpy.random.randint(100,size=(3,2,4))
>>> a
array([[[93, 19, 44, 54],
        [81, 63, 13, 83]],

       [[22,  7, 49, 82],
        [19,  8, 57,  7]],

       [[25, 40, 54, 13],
        [81, 16, 26, 72]]])
>>> numpy.random.shuffle(a.flat)
>>> a
array([[[26, 13, 19,  8],
        [19, 44, 72, 54]],

       [[25, 93, 16, 57],
        [81, 83, 13,  7]],

       [[ 7, 63, 81, 54],
        [49, 40, 22, 82]]])
```

http://docs.scipy.org/doc/numpy/reference/routines.random.html

Quelques fonctions bien utiles...

- trapz(y[, x, dx, axis]) : Intégration par la méthode des trapèzes.
- gradient(f, *varargs) : Gradient d'un tableau à N dimensions.
- diff(a[, n, axis]) : Différences finies.
- dot(a, b) : Produit de matrices.
- inner(a, b) : Produit scalaire.
- cholesky(a) : Factorisation de Cholesky.
- eig(a) : Valeurs et vecteurs propres.
- eigvals(a) : Valeurs propres.
- det(a) : Déterminant.
- solve(a, b) : Solution d'un système linéaire.
- inv(a) : Inversion de matrices.

Entrées/Sorties

- Formats Numpy Binaire et texte (.npy .npz) : `load/save/savez/loadtxt/savetxt`
- Pickle
- h5py

Pour finir

- fft : Fast Fourrier Transform.
- random : random numbers generation.
- linalg : Algèbre linéaire (Lapack, ATLAS ou MKL + BLAS).
- distutils : Outils de compilation pour les package numpy.
- f2py : interfacage Fortran pour Python/numpy.

Quelques logiciels utilisant numpy :

- Scipy `http://www.scipy.org`.
- pyopencl, pycuda `http://mathema.tician.de/software`.
- pyamg `http://code.google.com/p/pyamg/`.
- femhub `http://base.femhub.org/`.