CNRS Autrans 2010

NumPy

Marc Poinot
Numerical Simulation
For
Aerodynamics and Aeroacoustics Dept.

ONERA
THE FRENCH AEROSPACE LAB

retour sur innovation

NumPy

- Le module incontournable
  - Héritier de *Numeric* et *numarray*
  - Classes de base pour SciPy
- Installation
  - Module Python standard
  - Optimisation plateforme: blas, lapack...
- Utilisation
  - Traitement Python des tableaux pour calcul numérique
    - Nombreuses fonctions de manipulation
    - Bibliothèque mathématique importante
  - Support pour vos propres bibliothèques
    - Interface Python pur
    - API pour encapsulation de codes Fortran, C/C++

Le tableau NumPy

ndarray

L'objet Tableau

- 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
- Peut être une vue d'un autre objet Python implémentant une interface *array* ou bien *buffer*

Multi-dimensionnel, tous types de données

- Les dimensions et parcours sont modifiables, les indexations souples
- Optimisations internes pour les 1D, 2D et 3D
- Paramétrage complet de la représentation interne des éléments du tableau

Interfaçable avec les codes, en particulier Fortran

- Permet l'encapsulation de codes fortran
- Gestion possible des interfaçages multiples Fortran/C/C++

```
>>> import numpy
```

Création - 1

La création d'un tableau définit...

Son contenu

- Par exemple sous forme d'une liste de valeurs

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

Ses dimensions

- Une liste: dimension 1, une liste de listes: dimension 2, une liste de listes de listes: dimension 3...

```
a=array([0.1, 0.0, 0.2])
b=array([[1,2,3],[4,5,6]])
```

Son type d'élément

- Un entier Python est un long, un réel est un double
- Il est nécessaire de préciser si les valeurs en arguments ne sont pas les types Python

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

Création - 2

- Diverses méthodes de création

```
>>> a=arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a=zeros((2,4),dtype='f')
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]], dtype=float32)
>>> a=ones((3,5))*nan
array([[ nan,  nan,  nan,  nan,  nan],
       [ nan,  nan,  nan,  nan,  nan],
       [ nan,  nan,  nan,  nan,  nan]])
>>> a=identity(3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> a=mat([[1,0],[0,1]])
matrix([[1, 0],
        [0, 1]])
```

Création - 3

- Fonction des indices

```
>>> def initfunction(i,j):
...   return 100+10*i+j

>>> c=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")
```

Types d'éléments - 1

Les types traditionnels

- entiers et réels simple et double précision
  - la représentation dépend de votre machine
- complexes
  - attention complex64 pour deux *float32*s
- chaîne de caractères
- objet Python quelconque

`int32`	**`i`**	
`uint64`	**`L`**	
`float_`	**`f`**	*FloatType*
`complex128`	**`G`**	*float64 + float64 j*
`object_`	**`O`**	
`str_`	**`S#`**	*nombre de chars #*

Types d'éléments - 2

Les objets

- Votre classe ou tout autre objet Python

```
>>> a=Vector(1,2,3)
>>> b=Vector(4,5,3)
>>> c=Vector(2,5,2)
>>> v=array([a,b,c])
>>> v
array([<__main__.Vector instance at 0x102d05a8>,
       <__main__.Vector instance at 0x102d0560>,
       <__main__.Vector instance at 0x102d0518>], dtype=object)
>>>
```

Les records

- Ne contient que des types de base NumPy

```
>>> v=ones((200,),dtype=(('Masse','f',(1,),)('Vitesse','f',(2,)))
>>> v[0]=(2.0,(0.2,0.1))
>>> m=v[1]['Masse']
```

Identifier un type

- dtype
  - Un des attributs du tableau

```
>>> a=array([1,2,3])
>>> a.dtype
dtype('int64')
>>> a.dtype.char
'l'
```

  - Diverses fonctions pour construire et manipuler les types

```
>>> numpy.sctype2char(numpy.int32)
'i'
```

  - La définition des types permet portabilité et interopérabilité

```
>>> v.dtype
dtype([('Masse', '<f4', (1,)), ('Vitesse', '<f4', (2,))])
```

Attributs - 1

- flat
  - Vue 1D d'un tableau
  - Pas de modification du tableau
  - Itérateur

```
>>> 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
<numpy.flatiter object at 0xc9db80>
>>> a.flat[0]
0
>>> a.flat[:]
array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 2, 3, 4, 0, 1, 2, 3, 4])
>>>
```

Attributs - 2

- shape
  - Tuple des dimensions d'un tableau
    - Attribut accessible en lecture et en écriture

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

  - Le nombre d'éléments doit rester inchangé

len(a.flat) = *Constante*

  - La méthode *reshape* permet aussi de changer le shape

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

Attributs - 3

Fortran

- Type d'implantation mémoire
  - C (par défaut) line major, Boucle en i,j,k
  - Fortran column major, Boucle en k,j,i
- Pas d'impact sur le shape

```
>>> a=ones((2,3,4),order='Fortran')
>>> isfortran(a)
True
```

OwnData

- Python est propriétaire de la zone mémoire

```
>>> a.flags.owndata
True
```

Attributs - 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]]])
>>> 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]]])
```

Transposée

- Retourne une vue du tableau, pensez à la copie

```
b=a.T
```

- Existe aussi la fonction, elle retourne ou pas une vue du tableau suivant sa structure

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

Indexation - 1

Un élément dans le tableau

- Syntaxe similaire aux séquences

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

- Syntaxe avec implémentation optimisée pour l'accès

```
>>> a[0,2,1]
9
>>> a[0,0:-1]
array([[0, 1, 2, 3],[4, 5, 6, 7]])
```

- La syntaxe fonctionne pour la la référence et l'assignation

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

Indexation - 2

- Prendre un *axe* complet

```
>>> a=arange(24).reshape(2,3,2,2)
>>> a.tolist()
[[[[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[0,:,:,0]
array([[ 0,  2],
       [ 4,  6],
       [ 8, 10]])
>>> a[0,...,0]
```

- Double indirection

  - L'argument de l'indexation est un ndarray 1D (ou une liste)

```
>>> a=arange(13)*3
array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27, 30, 33, 36])
>>> a[array([2,4,5,9,11])]
array([ 6, 12, 15, 27, 33])
>>> a[[2,4,5,9,11]]
array([ 6, 12, 15, 27, 33])
```

Indexation - 3

- Ajout du step
  - *[<start>:<stop>:<step>]*

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

Méthodes - 1

Différent types de méthodes

- Associée au module

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

- Associée au ndarray
  - Appliquée élément par élément, retourne un ndarray

```
>>> a=arange(6).reshape(3,2)
>>> 1+sin(a)*1/2
array([[ 1.        ,  1.42073549],
       [ 1.45464871,  1.07056   ],
       [ 0.62159875,  0.52053786]])
>>> a>3
array([[False, False],
       [False, False],
       [ True,  True]], dtype=bool)
```

Méthodes - 2

Associée au ndarray

- Appliquée sur la totalité du tableau

```
>>> a.max()
5
>>> a.sum()
15
```

- Sur une partie du tableau

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

Associée au module

- Retourne plusieurs tableaux

```
>>> a=arange(24)
>>> split(a,3)
[array([0, 1, 2, 3, 4, 5, 6, 7]),
 array([ 8,  9, 10, 11, 12, 13, 14, 15]),
 array([16, 17, 18, 19, 20, 21, 22, 23])]
```

Méthodes - 3

Copie ou référence

Copie explicite

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

Copie implicite

```
>>> 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])
```

Itérateur

```
a.flat
```

Vue

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

Ufuncs - 1

Universal functions

Opère sur un tableau élément par élement

- `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

Broadcasting

- Il est parfois nécessaire de faire des adaptations pour que les vecteurs en entrée aient la même taille
- `a=array([1,2,3,4,5])*2`
- Provoque le broadcasting `b=array([2,2,2,2,2])`

Syntaxe compacte

- Pas de boucles
- Parfois difficile à lire

```
>>> x=arange(10)
>>> x[(x**3-9*x**2+23*x-15)==0]
array([1, 3, 5])
```

Ufuncs - 2

add(x1) Add arguments element-wise.
subtract(x1) Subtract arguments, element-wise.
multiply(x1) Multiply arguments element-wise.
divide(x1) Divide arguments element-wise.
logaddexp(x1) Logarithm of the sum of exponentiations of the inputs.
logaddexp2(x1) Logarithm of the sum of exponentiations of the inputs in base-2.
true_divide(x1) Returns a true division of the inputs, element-wise.
floor_divide(x1) Return the largest integer smaller or equal to the division of the inputs.
negative() Returns an array with the negative of each element of the original array.
power(x1) First array elements raised to powers from second array, element-wise.
remainder(x1) Return element-wise remainder of division.
mod(x1) Return element-wise remainder of division.
fmod(x1) Return the element-wise remainder of division.
absolute() Calculate the absolute value element-wise.
rint() Round elements of the array to the nearest integer.
sign() Returns an element-wise indication of the sign of a number.
conj() Return the complex conjugate, element-wise.
exp() Calculate the exponential of all elements in the input array.
exp2() Calculate 2**p for all p in the input array.
log() Natural logarithm, element-wise.
log2() Base-2 logarithm of x.
log10() Return the base 10 logarithm of the input array, element-wise.
expm1() Calculate exp(x) - 1 for all elements in the array.
log1p() Return the natural logarithm of one plus the input array, element-wise.
sqrt() Return the positive square-root of an array, element-wise.
square() Return the element-wise square of the input.
reciprocal() Return the reciprocal of the argument, element-wise.
ones_like() Returns an array of ones with the same shape and type as a given array

greater(x1) Return the truth value of (x1 > x2) element-wise.
greater_equal(x1) Return the truth value of (x1 >= x2) element-wise.
less(x1) Return the truth value of (x1 < x2) element-wise.
less_equal(x1) Return the truth value of (x1 =< x2) element-wise.
not_equal(x1) Return (x1 != x2) element-wise.
equal(x1) Return (x1 == x2) element-wise.

sin() Trigonometric sine, element-wise.
cos() Cosine elementwise.
tan() Compute tangent element-wise.
arcsin() Inverse sine, element-wise.
arccos() Trigonometric inverse cosine, element-wise.
arctan() Trigonometric inverse tangent, element-wise.

isreal(x) Returns a bool array, where True if input element is real.
iscomplex(x) Returns a bool array, where True if input element is complex.
isfinite() Test element-wise for finite-ness (not infinity or not Not a Number).
isinf() Test element-wise for positive or negative infinity.
isnan() Test element-wise for Not a Number (NaN), return result as a bool array.
signbit() Returns element-wise True where signbit is set (less than zero).
copysign(x1) Change the sign of x1 to that of x2, element-wise.
 nextafter(x1) Return the next representable floating-point value after x1 in the direction of x2 element-wise.
modf(out1) Return the fractional and integral parts of an array, element-wise.
ldexp(x1) Compute y = x1 * 2**x2.
frexp(out1) Split the number, x, into a normalized fraction (y1) and exponent (y2)
fmod(x1) Return the element-wise remainder of division.
floor() Return the floor of the input, element-wise.
ceil() Return the ceiling of the input, element-wise.
trunc() Return the truncated value of the input, element-wise.

logical_and(x1) Compute the truth value of x1 AND x2 elementwise.
logical_or(x1) Compute the truth value of x1 OR x2 elementwise.
logical_xor(x1) Compute the truth value of x1 XOR x2, element-wise.
logical_not() Compute the truth value of NOT x elementwise.

Matrices

- Dérivée de ndarray
  - Toute **Matrix** est au moins un *ndarray*
- Utilisée dans *numpy.linalg*
- Quelques spécificités
  - 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])
```

Masquages

- Dérivée de ndarray
  - Tout **MaskedArray** est au moins un *ndarray*
  - Module particulier *numpy.ma*
- Tableaux dont une partie des données est masquée
  - Données invalides, incomplètes, hors intervalle...
  - Basé sur un tableau existant auquel on ajoute un masque
- Les opérations ne s'appliquent que sur les valeurs non-masquées

```
>>> a=sin(arange(24).reshape(2,3,4))
>>> m=numpy.ma.masked_greater(a,0.5)
masked_array(data =
 [[[0.0 -- -- 0.14]
  [-0.75 -0.95 -0.27 --]
  [-- 0.41 -0.54 -0.99]]
 [[-0.53 0.42 -- --]
  [-0.28 -0.96 -0.75 0.14]
  [-- -- -0.00 -0.84]]],
             mask =
 [[[False  True  True False]
  [False False False  True]
  [ True False False False]]
 [[False False  True  True]
  [False False False False]
  [ True  True False False]]],
       fill_value = 1e+20)
>>> for v,t in zip(m.data.flat,m.mask.flat):
...   if t: print v
...
0.841470984808
0.909297426826
0.656986598719
0.989358246623
0.990607355695
0.650287840157
0.912945250728
0.836655638536
```

Chaînes de caractères

- Pas de classe
  - Un tableau de caractéres est un chaîne de caractéres
  - Manipulations avec *numpy.core.defchararray*
  - Eviter la classe *Chararray* qui n'est présente que pour raisons historiques
- Fonctions similaires au *strings* Python

```
>>> capitalize(a)
array(['Autrans', 'Paris', 'Bordeaux', 'Rennes', 'Lille'],
      dtype='|S8')
>>> a[0]
'autrans'
>>> a[0].dtype
dtype('|S7')
>>> type(a[0])
<type 'numpy.string_'>
>>> a[0].shape
()
```

- Passage taille fixe du Fortran

```
a=array(map(lambda x: "%-32s"%x,['Autrans', 'Paris', 'Bordeaux']))
```

Random

Divers générateurs de nombres

- Module numpy.random
  - Fonctions adaptées au ndarrays et tous les types d'éléments

```
>>> a=randint(100,size=(3,2,4))
array([[[67,  1, 36, 28],
        [90, 91, 27,  9]],
       [[28, 85, 33,  1],
        [ 7, 81, 87, 91]],
       [[51, 72, 94, 49],
        [ 9, 82, 32, 76]]])
>>> shuffle(a.flat)
>>>
```

- Très nombreuses distributions

```
>>> poisson(100,10)
array([102, 104,  94, 102,  88,  87, 105, 118, 109, 102])
```

Entrées/Sorties

- Formats Numpy
  - Binaire et texte
    - .npy .npz
    - Fonctions load/save/savez/loadtxt/savetxt
- Autres formats
  - Pickle
  - Cherchez une interface NumPy
    - load/save sur format pivot Python (array, buffer)
    - SciPy
    - HDF5, NetCDF, ADF...
  - Utiliser l'API
    - Directe ou bien *cython*...

Représentation machine

- Le numérique est très lié à la machine
  - Calculs hétérogènes
    - NumPy utilise le standard IEEE754
    - Passage en mémoire
      - Traductions parfois faites par les couches réseaux
    - Passage sur disque
      - Cluster pour le calcul
      - Stations locales pour le pré/post traitement
      - Laptop pour les mises au points
- Outillage NumPy
  - Identification plateforme et types
    - dtype: >u4 <f8
    - finfo (min, max, eps...)
  - Des méthodes pour traduire
    - can cast, mintypecode...

Divers

- Options pour l'affichage des tableaux
  - set_printoptions: précision, tailles de lignes...

```
>>> a=sin(arange(5))
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ])
>>> numpy.set_printoptions(precision=2,threshold=sys.maxint)
>>> a
array([ 0.  ,  0.84,  0.91,  0.14, -0.76])
```

- Documentation en ligne
  - info,lookfor

```
>>> lookfor('determinant')
Search results for 'determinant'
--------------------------------
numpy.linalg.det
    Compute the determinant of an array.
```

- IEEE754: Gestion des cas d'erreurs
  - Afficher, ignorer, appeler une fonction utilisateur

```
>>> seterr(divide='ignore')
{'over': 'ignore', 'divide': 'ignore', 'invalid': 'ignore', 'under': 'ignore'}
>>> a=zeros((2,2))
>>> b=ones((2,2))
array([[ inf,  inf],[ inf,  inf]])
```

Pour finir

- Il reste au moins...
  - Les librairies: FFT, Algèbre linéaire, Statistiques, Finances...
  - L'API: Une interface Python pour vos propres outils C et Fortran
  - La migration Numeric, numarray...
  - Les outils interopérables: Matlab, Ensight...

- Maintenant:
  - Tout le monde les mains sur le clavier avec une page blanche (aaaaargh !)

- Plus tard:
  - Contribuez à la force de Python NumPy avec votre expertise
  - Construisez des modules installables et utilisables
  - Diffusez en Open Source