

Les bases de NumPy

NumPy est une extension du langage de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux.

NumPy permet la manipulations des vecteurs, matrices et polynômes.

Directive d'importation

- standard :

```
import numpy as np
```

Tableaux numériques

On convertit facilement des listes Python en tableau numpy. Essayez ceci :

```
<sxh python;> import numpy as np a=np.array(1,2],[3,4) print a print a.dtype </sxh>
```

 Sortie :

```
[[1 2]
 [3 4]]
<type 'numpy.ndarray'>
```

Pour définir un tableau, appelez simplement la fonction `.array` avec une liste ou un tuple. Des fonctions spéciales **zero**, **ones**, **rand** permettent d'initialiser à des valeurs particulières (0 ou 1), ou aléatoires.

Les fonctions `arange` et `shape` sont bien pratiques pour générer des nombres en séquences et réarranger des listes de nombres. La fonction `linspace` est utile parce qu'elle impose exactement le nombre de valeurs créées entre un minimum et un maximum.

Vous pouvez consulter [cette page](#) pour consulter d'autres fonctionnalités, ou [celle-ci](#), plus documentée.

```
<sxh python; title : arrays_01.py> #! /usr/bin/env python # -*- coding: utf-8 -*- """ Divers codes à essayer pour créer des tableaux "array" """ import numpy as np
```

```
a=np.array(1) # on peut créer un "array" à partir d'un tuple # afficher a, le nombre de dimensions, les dimensions, le type de donnée print a, a.ndim, a.shape, a.dtype # avec des "floats" : b= np.array(
```

```
[[1.1, 2.2, 3.3, 4.4],
 [5.5, 6.6, 7.7, 8.8],
```

```
[9.9, 0.2, 1.3, 2.4]])
```

```
print b, b.ndim, b.shape, b.dtype # un tableau de zéros c=np.zeros2) print c, c.ndim, c.shape, c.dtype  
# un tableau tridimensionnel de 1 "complexe" d=np.ones3) print e, e.ndim, e.shape, e.dtype  
f=np.random.rand(3,3) print f, f.ndim, f.shape, f.dtype # utilisation de linspace pour imposer le  
nombre d'éléments générés : g=np.linspace(0.,np.pi,11) print g, g.ndim, g.shape, g.dtype </sxh>
```

Quelques manipulations élémentaires : <sxh python; title : arrays_02.py> `#!/usr/bin/env python # -
*- coding: utf-8 *- """ Divers codes à essayer pour manipuler des tableaux "array" """ import numpy
as np`

```
a=np.array(1,2],[3,4) b=np.array(1,1],[1,1) c=a+b # addition terme à terme print c, c.ndim, c.shape,  
c.dtype d=a*b # multiplication terme à terme print d, d.ndim, d.shape, d.dtype e=np.dot(a,b) #  
multiplication matricielle print e, e.ndim, e.shape, e.dtype f=np.sin(np.pi*0.5*a) # fonction  
mathématique et adaptation automatique du type print f, f.ndim, f.shape, f.dtype g=np.transpose(a)  
# transposition print g, g.ndim, g.shape, g.dtype print np.sum(a),np.min(a), np.max(a) # somme des  
éléments, minimum, maximum </sxh>
```

Fonctions mathématiques principales :

- abs, sign, sqrt
- logarithmes/exponentielles : log, log10, exp
- trigonométriques et inverses : sin, cos, tan, arcsin, arccos, arctan
- hyperboliques et inverses : sinh, cosh, tanh, arcsinh, arccosh, arctanh
- entiers inférieur, supérieur ou le plus proche : floor, ceil, rint

Autres fonctions

- min et max rendent le minimum et le maximum, argmin et argmax rendent les indices de ce
éléments dans un tableau 1D (consulter la documentation pour les dimensions supérieures.
- sorted : tri
- clip : cliping permettant d'éliminer des valeurs inférieures à une borne minimale donnée ou
supérieures à une borne maximale
- unique : élimine les "doublons"
- fonctions booléennes, pour des conditions, ou pour filtrer suivant des conditions (voir la
documentation)

Algèbre linéaire

```
<sxh python; title : simple_linear_system.py> #!/usr/bin/env python # -*- coding: utf-8 -*- """ Solve a  
system of simultaneous equation in two variables of the form
```

$$\begin{aligned} 2x + 7y &= 17. \\ 3x - 5y &= -21. \end{aligned}$$

reference : <http://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.solve.html> """ #

```
import numpy as np
a = np.array(2.,7.,[3.,-5.]) # coefs matrice
b = np.array(17.,[-21.]) # independent coef vector
print np.linalg.solve(a,b) # solution
```

Quelques possibilités supplémentaires : `<sxh python; title : arrays_linalg_03.py> #! /usr/bin/env python # -*- coding: utf-8 -*- """ Divers codes à essayer pour de l'algèbre linéaire avec des tableaux "array" """ import numpy as np`

```
a=np.array(1,2,[3,4])
print a, a.ndim, a.shape, a.dtype
b=np.linalg.inv(a) # matrice inverse
print b, b.ndim, b.shape, b.dtype
unit = np.eye(2) # matrice unitaire
print unit, unit.ndim, unit.shape, unit.dtype
v = np.array(10., [14.]) # vecteur colonne
x1=np.dot(b,v) # multiplication de l'inverse de a par v
x2=np.linalg.solve(a,v) # solution du système linéaire de coefficients # des inconnues a et de coefficients indépendants b
# les deux techniques donnent évidemment le même résultat !
print x1, x1.ndim, x1.shape, x1.dtype
print x2, x2.ndim, x2.shape, x2.dtype # valeurs propres et vecteurs propres de matrices :
d=np.array(1,1,[-1,1])
print np.linalg.eig(d)
```

Numpy dispose aussi d'une classe particulière de "arrays" pour des matrices.

Autres fonctions

- inner : produit scalaire (équivalent à dot sur des tableaux 1D)
- cross : produit vectoriel
- det : déterminant

Statistiques élémentaires

```
<sxh python; title : arrays_stats_elem_04.py> #! /usr/bin/env python # -*- coding: utf-8 -*- """ Divers codes à essayer pour des statistiques élémentaires sur des tableaux "array" """ import numpy as np
```

```
a=np.array([1.,2.,3.5,5.,6.,7.,7.4,7.8,8.2,8.4,8.5,9.,10.2,12.5])
print a, a.ndim, a.shape, a.dtype
print "médiane = ", np.median(a)
print "moyenne = ", np.mean(a)
print "variance = ", np.var(a)
print "Écart-type = ", np.std(a)
```

Itérations sur les tableaux

```
<sxh python; title : arrays_iteration_05.py> #! /usr/bin/env python # -*- coding: utf-8 -*- """ itérations sur des tableaux "array" """ import numpy as np
```

```
a=np.array([1.,2.,3.5,5.,6.,7.,7.4,7.8,8.2,8.4,8.5,9.,10.2,12.5])
for x in a:
```

```
    print x
```

l'itération sur un tableau multidimensionnel se fait sur un premier niveau de sous-listes

```
b= np.array(
```

```
[[1.1, 2.2, 3.3, 4.4],
 [5.5, 6.6, 7.7, 8.8],
 [9.9, 0.2, 1.3, 2.4]])
```

for x in b:

```
print x
for y in x:
    print y, ", ",
print
```

</sxh>

Manipulation de polynômes

Une nouvelle bibliothèque [polynomial](#) devrait remplacer l'ancien "poly1d"

<sxh python; title : arrays_polynomes_06.py> `#!/usr/bin/env python # -*- coding: utf-8 -*- """
Utilisation de tableaux "array" pour des polynômes """ import numpy as np`

`# les coefficients du polynômes sont donnés par ordre décroissance des degrés
a=np.poly1d([1.,2.,3.,4.]) # = $x^3 + 2x^2 + 3x + 4$`

`print "polynôme : \n",a, type(a) # les coefficients de a : print "coefficients : ",a.coefs # les racines de
a : print "racines : ",a.roots # l'ordre du polynôme : print "ordre : ",a.order # évaluations sur un
vecteur x=np.linspace(0,2,21) print "x = ",x print "évaluation en x : ",np.polyval(a,x) # dérivation
print "dérivée : \n",np.polyder(a) # intégration print "intégrale : \n",np.polyint(a) # création d'un
polynôme par ses racines b=a.roots c=np.poly1d(b,True) print "Polynômes recrées par les racines
:\n", c # fitting polynomial xd=np.array([0.,1.,2.,3.,4.,5.]) yd1=np.array([0.05,0.99,3.95,
9.17,15.86,24.93]) pfit=np.poly1d(np.polyfit(xd,yd1,2)) print "fit d'une parabole (polynôme d'ordre 2)
sur ces x et y : " print xd print yd1 print "polynôme de fit : \n",pfit </sxh>`

Autres fonctions : voir [ici](#)

Transformées de Fourier

À compléter ...

Références

- [Site officiel](#)
- [NumPy reference](#)
- [Page Wikipédia](#)
- [Guide to NumPy](#)
- [Tutoriel via l'exemple du jeu de la vie](#)
- http://wiki.scipy.org/Tentative_NumPy_Tutorial

```
1)
(1,2),(3,4
2)
4,2
3)
2,3,4),dtype=complex) print d, d.ndim, d.shape, d.dtype # un tableau avec arange, et ensuite
reshape e1= np.arange(1,36,1) e=np.reshape(e1,(5,7
```

From:

<https://dvillers.umons.ac.be/wiki/> - **Didier Villers, UMONS - wiki**

Permanent link:

https://dvillers.umons.ac.be/wiki/teaching:progappchim:numpy_simple?rev=1392036091

Last update: **2014/02/10 13:41**

