Tableaux à plusieurs dimensions

1 Liste de listes.

Pour construire un tableau à plusieurs dimensions, il existe plusieurs méthodes sous Python

On peut construire un tableau – une matrice, si le tableau est à 2 dimensions - comme une liste de listes :

Exemple : pour construire $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$

```
>>> a=[[1, 2] ,[3, 4] ,[5, 6]]
>>> a
[[1, 2], [3, 4], [5, 6]]
>>> len(a)
3
>>> a[1]
[3, 4]
>>> a[1][0]
3
```

Un tableau de dimension 2 (liste de listes) contiendra des éléments qui devront être appelé par la double indexation `[i][j]` :

- Le premier index renvoie donc à l'index de la ligne.
- Le deuxième index renvoie donc à l'index de la colonne.

La commande *len()* renvoie la longueur de la liste.

```
>>> len(a)
3
>>> len(a[1])
2
```

2 Type array sous Numpy

La bibliothèque de fonction Numpy propose un type de tableau homogène dédié noté *`array`*, et les fonctions dédiées. Les éléments contenus dans un *`array`* seront de même type. Les types proposés sont plus nombreux que sous python. On y trouve par exemple le type *`uint8`* : *unsigned integer 8* bits (comprendre « codage en 8 bits non signés », soit de 0 à 255)

2.1 Construction

Une première méthode consiste à convertir une liste en un tableau via la commande *`array`* . Le deuxième argument est optionnel et spécifie le type des éléments du tableau.

```
>>> a = np.array([1, 4, 5, 8], float)
>>> a
array([ 1., 4., 5., 8.])
>>> type(a)
<type 'numpy.ndarray'
```

Un *array* peut être multidimensionnel :

```
>>> a=np.array([[1,2],[3,4],[5,6]])
>>> a
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> a[1,0]
3
```

Nous limiterons l'utilisation du type *array* aux tableaux {uni,bi}-dimensionnels (vecteurs et matrice), mais ils peuvent s'étendre au delà.

2.2 Opération sur les *array*

- Copie

Attention : En Python modifier une donnée d'une extraction d'un tableau entraîne aussi une modification du tableau initial ! Si nécessaire la fonction `np.copy(a)` ou `a.copy()` permet de faire une copie d'un tableau *a.*

```
>>> a=np.array([1, 2, 3, 4, 5])
>>> c=np.array([1, 2, 3, 4, 5])
>>> b=a[1:3]
>>> b
array([2, 3])
>>> b[1]=0
>>> a
array([1, 2, 0, 4, 5])   # a  modifié
>>> b=c[1:3].copy()      # première méthode
>>> b[1]=0
>>> bb=np.copy(c[1:3])   # seconde méthode
>>> bb[1]=0
>>> c
array([1, 2, 3, 4, 5])   # c non modifié
```

- Extraction

La technique dite du *slicing* permet d'extraire des sous-tableaux (cette technique fonctionne aussi avec les listes)

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[2:9:3] # [début:fin:pas]
array([2, 5, 8])
>>> a[2:8:3] # le dernier élément n'est pas inclus
array([2, 5])
>>> a[:5]    # le dernier élément n'est pas inclus
array([0, 1, 2, 3, 4])
```

- Construction diagonales

```
>>> a=np.eye(5,5)
>>> a
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])
```

- Multiplications de matrices (attention aux dimensions!)

```
>>> a=np.array([[1,2],[3,4],[5,6]])
>>> b=np.array([[1,2,3],[4,5,6]])
>>> c=dot(a,b)
>>> print c
[[ 9 12 15]
 [19 26 33]
 [29 40 51]]
```

3 Type *mat* sous Numpy

Numpy/Scipy proposent aussi le type `mat` comme matrice, exclusivement un tableau bi-dimensionnel. Comme les fonctions telles que `ones`, `eye` retournent un objet de type `array` nous n'utiliserons pas dans la suite le type `mat`. Il permet cependant de saisir les matrices à-la-Matlab/Scilab et de faire le produit matriciel par le simple symbole *

```
>>> a=np.mat('[1 2 4 ; 3 4 5.]')
>>> b=np.mat('[2. ; 4 ; 6]')
>>> print a
[[ 1.  2.  4.]
 [ 3.  4.  5.]]
>>> print b
[[ 2.]
 [ 4.]
 [ 6.]]
>>> print a*b
[[ 34.]
 [ 52.]]
```

Remarque :

Tout objet array est convertible en type mat et réciproquement (sous la condition que le tableau (array) soit {uni,bi}-dimensionnel)

```
>>> a=np.array([1, 2, 4])
>>> np.mat(a)
matrix([[1, 2, 4]])
```