Calcul Scientifique

Cours 3: Manipulation de tableau numpy

Alexis Lechervy

Introduction

Objectifs

Étudier les opérations de manipulation de tableau :

- Ajout/Suppression d'élément,
- Modification du nombre de dimension,
- Concaténation,
- Redimensionnement,
- Réarrangement.

Sommaire

Ajout d'élément

Matrice d'exemple

M = np.array([[1, 1], [2, 2], [3, 3]])
array([[1, 1],
[2, 2],
[3, 3]])

Ajout d'élément : np.insert

- Ajout d'un élément (e=5) à une position donnée (i=1) : np.insert(M,i,e) -> array([1,5, 1, 2, 2, 3, 3])
- Ajout d'un élément (e=5) à plusieurs positions données (i=[3,5,6]) : np.insert(M,i,e) -> array([1, 1, 2, 5, 2, 3, 5, 3, 5])
- Ajout d'un élément (e=5) à une position (i=1) sur tout un axe (a=1) :
  np.insert(M,i,e,axis=a) -> array([[1, 5, 1],
  [2, 5, 2],
  [3, 5, 3]])

Attention

La fonction *np.insert* peut changer la forme du tableau. Elle a tendance à retourner des vecteurs numpy.

Suppression d'élément

Matrice d'exemple

M = np.array([[1, 1], [2, 2], [3, 3]])
array([[1, 1],
[2, 2],
[3, 3]])

Ajout d'élément : np.delete

- Suppression d'un élément à une position donnée (i=1) : np.delete(M,inp) -> array([1, 2, 2, 3, 3])
- Suppression d'éléments à plusieurs positions données (i=[3,5]) : np.delete(M,i,e) -> array([1, 1, 2, 3])
- Ajout d'éléments selon un axe (i=0) et une direction (a=1) : np.delete(M,i,axis=a) -> array([[1], [2], [3]])

Attention

La fonction *np.delete* peut changer la forme du tableau. Elle a tendance à retourner des vecteurs numpy.

Recopie d'un tableau

Recopie de tableau : np.tile

La fonction *np.tile* permet de recopier un tableau selon un axe.

```
>>> a = np.array([0, 1, 2])
>>> np.tile(a, 2)
array([0, 1, 2, 0, 1, 2])

>>> np.tile(a, (2, 2))
array([[0, 1, 2, 0, 1, 2],
       [0, 1, 2, 0, 1, 2]])

>>> b = np.array([[1, 2], [3, 4]])
>>> np.tile(b, 2)
array([[1, 2, 1, 2],
       [3, 4, 3, 4]])

>>> np.tile(b, (2, 1))
array([[1, 2],
       [3, 4],
       [1, 2],
       [3, 4]])
```

Sommaire

Augmentation du nombre de dimension

Principe

On peut vouloir augmenter le nombre de dimension d'un tableau numpy pour par exemple transformer un vecteur en matrice.

np.expand_dims

```
>>> x = np.array([1,2])
>>> x.shape
(2,)

>>> y = np.expand_dims(x, axis=0)
>>> y
array([[1, 2]])
>>> y.shape
(1, 2)

>>> y = np.expand_dims(x, axis=1)
>>> y
array([[1],
       [2]])
>>> y.shape
(2, 1)
```

np.newaxis

```
>>> x = np.array([1,2])
>>> x.shape
(2,)

>>> y = x[np.newaxis,:] # ou x[np.newaxis]
>>> y
array([[1, 2]])
>>> y.shape
(1, 2)

>>> y = x[:,np.newaxis]
>>> y
array([[1],
       [2]])
>>> y.shape
(2, 1)
```

Réduction du nombre de dimension

Principe

On peut vouloir réduire le nombre de dimension en supprimant les axes ayant un seul élément.

```
>>> M.shape
(3, 1, 2, 1)
# Suppression de tout les axes a 1
>>> M2 = M.squeeze()
>>> M2.shape
(3, 2)
# Suppression d'un axe a 1
>>> M2 = M.squeeze(axis=1)
>>> M2.shape
(3, 2, 1)
# Tentative de suppression d'un axe qui n'est pas a 1
>>> M2 = M.squeeze(axis=0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: cannot select an axis to squeeze out which
 has size not equal to one
```

Sommaire

np.concatenate

Présentation

La fonction *np.concatenate* permet de concaténer plusieurs tableau numpy.

Fonctionnement de concatenate

np.concatenate((M,M2,M3) , axis=0)

$$\text{->} \begin{bmatrix} M1 \\ M2 \\ M3 \end{bmatrix}$$

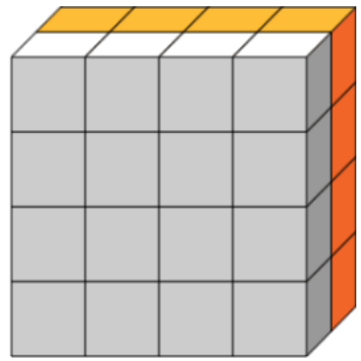
np.concatenate((M,M2,M3) , axis=1)

-> [M,M2,M3]

Empiler des matrices

np.concatenate((M[:, :,np.newaxis],M2[:, :,np.newaxis]) , axis=2)

Rq : L'axe d'empilement doit être présent dans les tableaux que l'on empile.



np.append

Présentation

La fonction *np.append* permet d'ajouter un tableau numpy à la fin d'un autre tableau numpy. Elle est équivalente à la fonction *np.concatenate* avec deux éléments.

```
>>> M = np.array([[1, 2, 3], [4, 5, 6]])
>>> v = np.array([7,8,9])
>>> np.append(M,v, axis=0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python3.7/site-packages/numpy/lib/fu
    return concatenate((arr, values), axis=axis)
ValueError: all the input arrays must have same number
 of dimensions
>>> np.append(M,v[np.newaxis,:],axis=0)
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

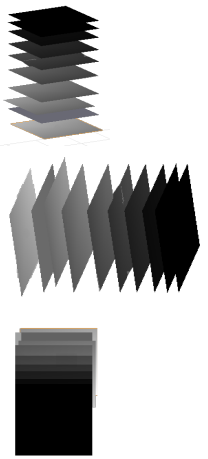
np.stack et ses variantes

Présentation

Permet de transformer une liste de tableau numpy de même dimension en un seul tableau les empilant tous selon un axe.

```
>>> arrays = [np.random.randn(3, 4) for _ in range(10)]
```

```
>>> np.stack(arrays, axis=0).shape
(10, 3, 4)
>>> np.stack(arrays, axis=1).shape
(3, 10, 4)
>>> np.stack(arrays, axis=2).shape
(3, 4, 10)
```



Variantes

- *np.vstack* Empile selon l'axe 0 (vertical). Résultat de taille (30,4)
- *np.hstack* Empile selon l'axe 1 (horizontal). Résultat de taille (3,40)
- *np.dstack* Empile selon l'axe 2 (profondeur). Résultat de taille (3,4,10)

Sommaire

flatten et ravel

Présentation

Permet de déplier un tableau numpy *m* en un vecteur dont la dimension est (*m.size*,) . *np.flatten* retourne un nouveau tableau numpy tandis que *np.ravel* retourne une vue sur le tableau initial. Une modification du contenu du tableau retourné par *np.ravel* modifiera donc également les valeurs du tableau initial.

Exemple

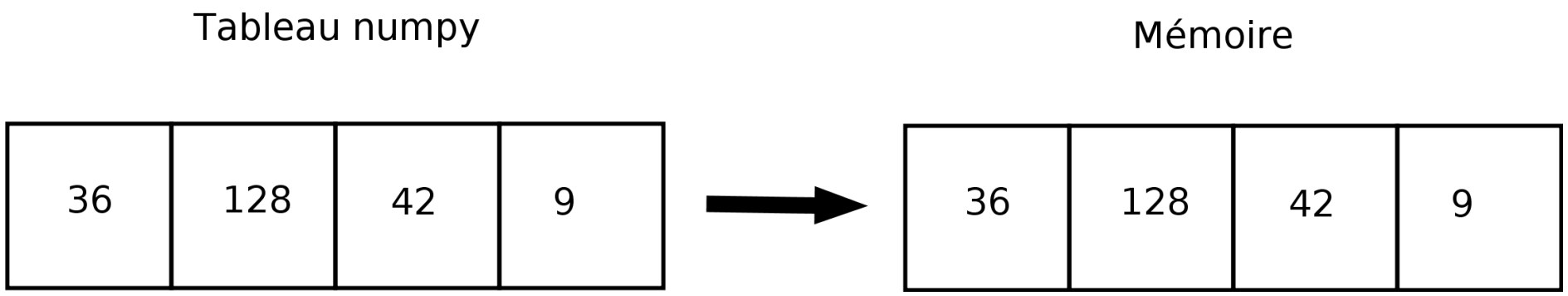
```
>>> M = np.random.random((20,3))
>>> M.shape
(20, 3)
>>> M.flatten().shape
(60,)
>>> M.ravel().shape
(60,)
```

Ordre de stockage des valeurs des tableaux numpy

Ordre de stockage

Les valeurs des tableaux numpy sont stocké dans le sens opposé des dimensions. Il faut d'abord parcourir la dernière dimension, puis l'avant dernière...

Cas 1 axe

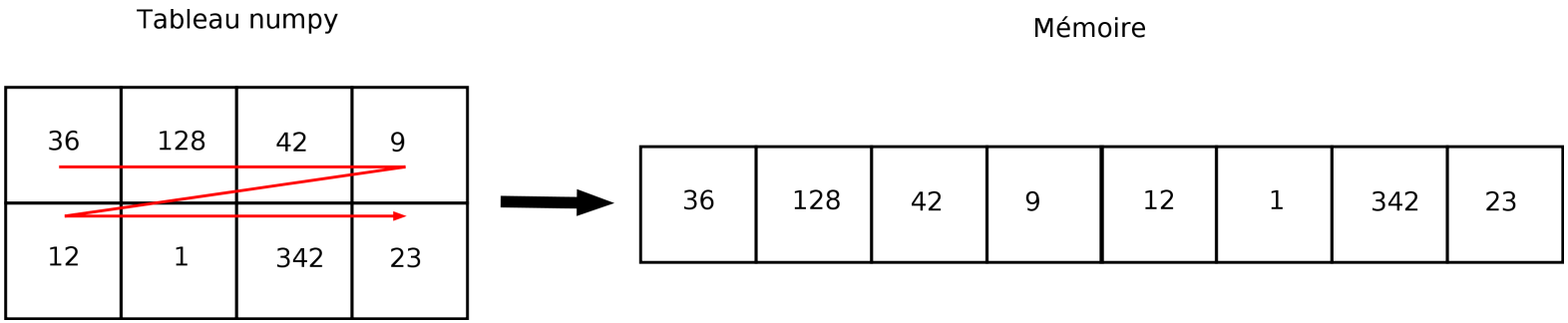


Ordre de stockage des valeurs des tableaux numpy

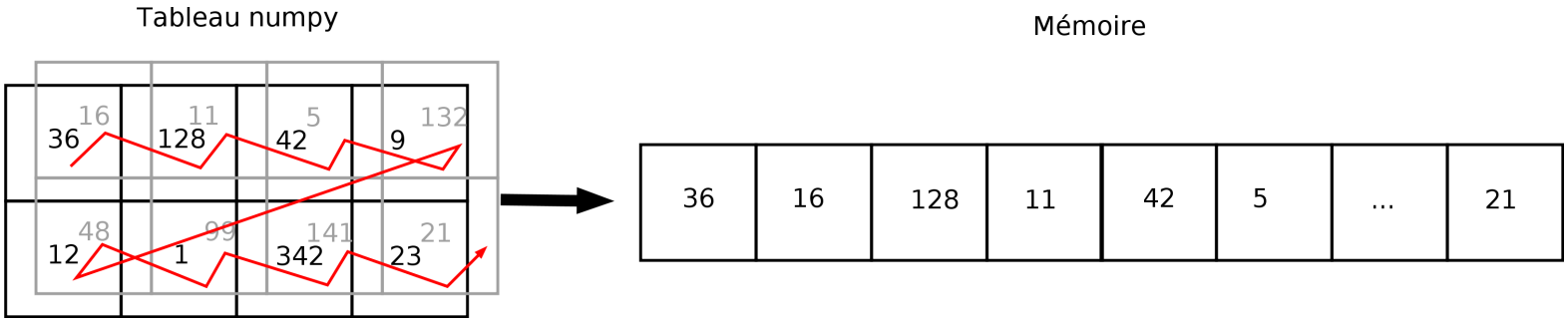
Ordre de stockage

Les valeurs des tableaux numpy sont stocké dans le sens opposé des dimensions. Il faut d'abord parcourir la dernière dimension, puis l'avant dernière...

Cas 2 axes



Cas 3 axes...



reshape et resize

reshape

La méthode *reshape* permet de changer la forme d'un tableau numpy. Elle permet par exemple de passer d'un tableau (6,4) à un tableau (8,3). Le redimensionnement se fait en partant d'un tableau entièrement déplié.

Fonctionnement de reshape

```
>>> M=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
>>> M.reshape((2,6))
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12]])
```

Remarques

- Le nombre d'élément d'un tableau est conservé avec *reshape*. Il n'est pas possible de transformer un tableau (5,2) en tableau (3,1).
- La valeur -1 permet d'indiquer qu'il faut choisir un nombre de dimensions tel que la fonction *reshape* puisse fonctionner. Par exemple (6,4) -> (8,-1) et équivalent à (8,3).
- *resize* a le même effet que *reshape* mais créé un nouveau tableau.

Sommaire

np.transpose

Principe

La fonction permet de réorganiser les axes d'un tableau numpy.

Syntaxe de np.transpose

```
M = np.ones((12,4,2))
print(M.shape)
-> (12,4,2)
M2 = np.transpose(M,(2,1,0))
print(M2.shape)
-> (2,4,12)
```

Cas particulier

Si l'on souhaite mettre l'axe 0 en dernière axe, il est possible d'utiliser le code $M.T$. Cette opération revient à faire une transposé (au sens mathématique) sur un vecteur (*np.transpose(M,(1,0))*) ou une matrice (*np.transpose(M,(1,2,0))*).

np.transpose versus np.reshape

(3,5,2)

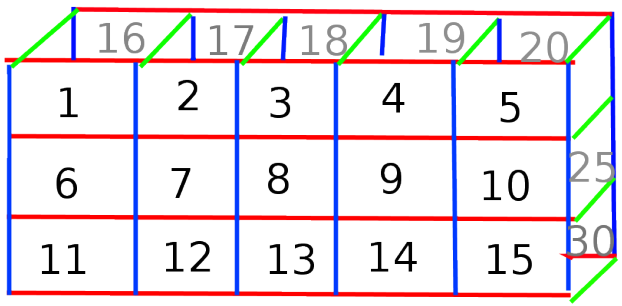
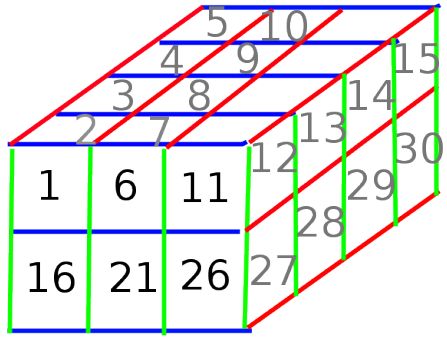


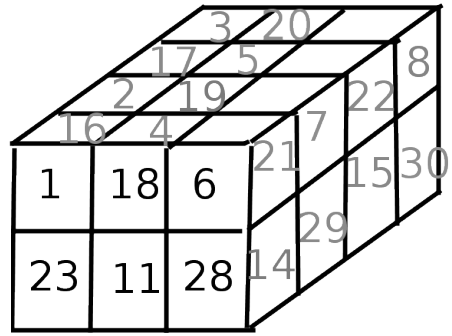
Tableau initial M

(2,3,5)



np.transpose(M,(2,0,1))

(2,3,5)



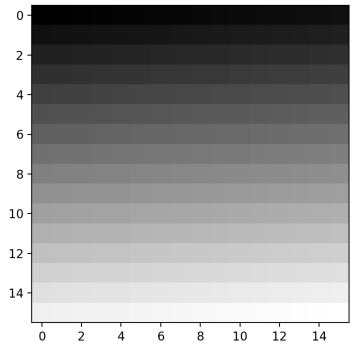
M.reshape((2,3,5))

Sommaire

Exemple : création d'un dégradé rouge

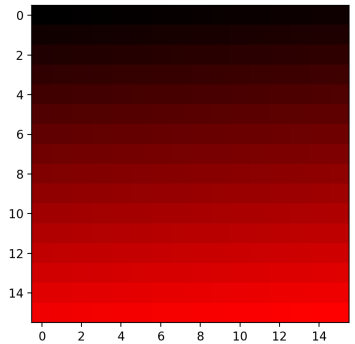
Création d'un dégradé noir et blanc sur les lignes

```
degNB = np.arange(256)
degNB = degNB.reshape((16,16))
```



Création d'un dégradé rouge sur les lignes

```
degRouge = np.zeros((16,16,3))
degRouge[ :, :,0] = degNB
```



Exemple : création d'un dégradé rouge

Transformation d'un dégradé sur les lignes en dégradé sur les colonnes

degRouge2 = np.transpose(im,(1,0,2))

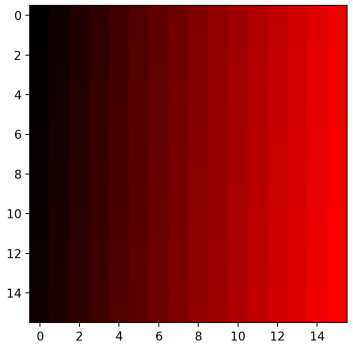
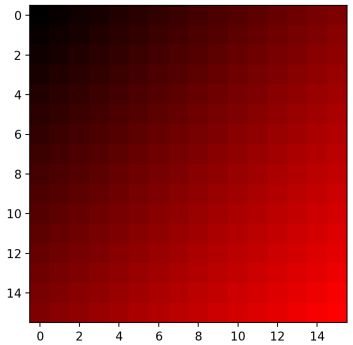


Image finale : dégradé rouge sur les lignes et les colonnes

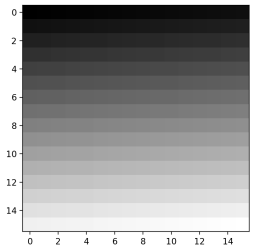
im = (degRouge1+degRouge2)//2



Exemple : Création d'un dégradé multicolore

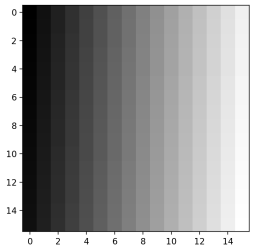
Création du canal rouge

```
imR = np.arange(256)
imR = imR.reshape((16,16))
```



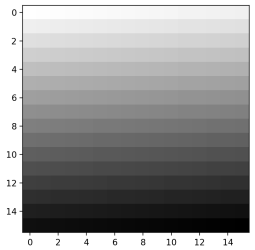
Création du canal vert

```
imG = np.arange(256)
imG = imG.reshape((16,16)).T
```



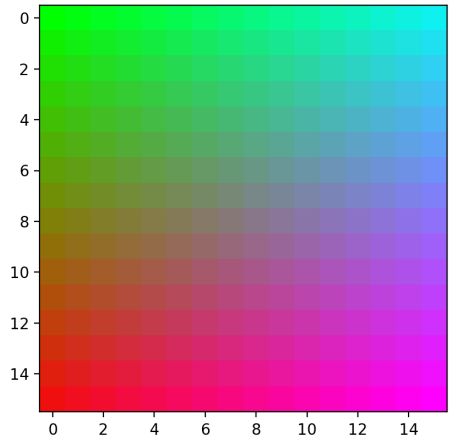
Création du canal bleu

```
imB = 255-np.arange(256)
imB = imB.reshape((16,16))
```



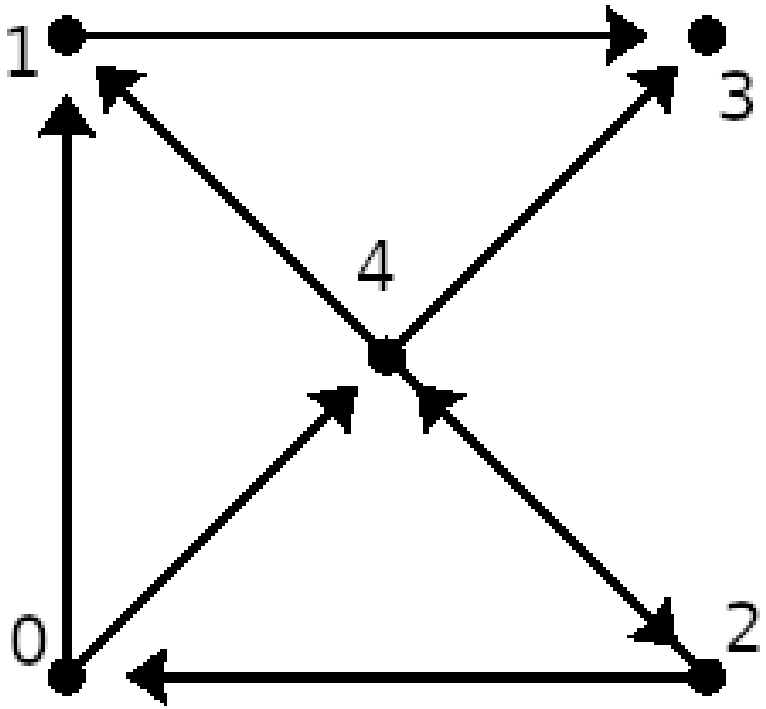
Assemblage des trois canaux

```
im = np.concatenate(
(imR[ :, :,np.newaxis],
 imB[ :, :,np.newaxis],
imG[ :, :,np.newaxis]),
axis=2)
```



Exemple : Manipulation d'un graphe

Représentation informatique d'un graphe orienté




```
array([[0, 1, 0, 0, 1],
       [0, 0, 0, 1, 0],
       [1, 0, 0, 0, 1],
       [0, 0, 0, 0, 0],
       [0, 1, 1, 1, 0]])
```

Si la case à la ligne i et la colonne j contient un 1 cela indique qu'il existe un arc du noeud i vers le noeud j.

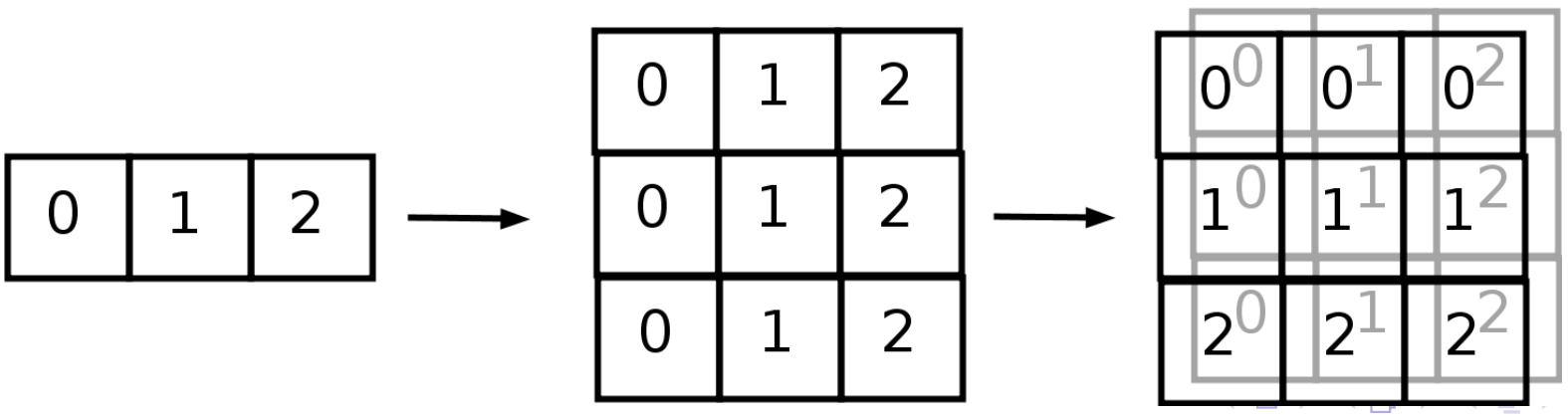
Définition numpy

```
g = np.array([[0,1,0,0,1],[0,0,0,1,0],[1,0,0,0,1],[0,0,0,0,0],[0,1,1,1,0]])
```

Récupération de la liste des arcs d'un graphe (1/2)

Liste de tout les arcs possibles

```
# création de la liste des noeuds existants
noeud = np.arange(g.shape[0])
# création d'une matrice de la même taille que g indiquant le noeuds source des arcs
a2 = np.tile(noeud,(g.shape[0],1))
# création d'un tensor indiquant les noeuds sources et destinations des arcs
a = np.stack((a2.T,a2),axis=2)
```



Récupération de la liste des arcs d'un graphe (2/2)

Récupérer juste les arcs du graphe

```
a = a.reshape((-1,2))
# on ne garde que les flèches du graphe
fleches = a[g.flatten()==1, :]
```