Cours Info - 15

Matrices

D.Malka

MPSI 2017-2018

Sommaire

Sommaire

Qu'est-ce qu'une matrice ?

Une matrice en mathématiques, ici (2,3) :

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Une matrice en mathématiques, ici (n,p) quelconque :

$$\begin{pmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1p} \\ \dots & \dots & \dots & \dots & \dots \\ a_{i1} & \dots & a_{ij} & \dots & a_{ip} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & \dots & a_{nj} & \dots & a_{np} \end{pmatrix}$$

Sommaire

Rappel sur les tableaux

Un tableau est une liste en Python (voir cours Info 5).

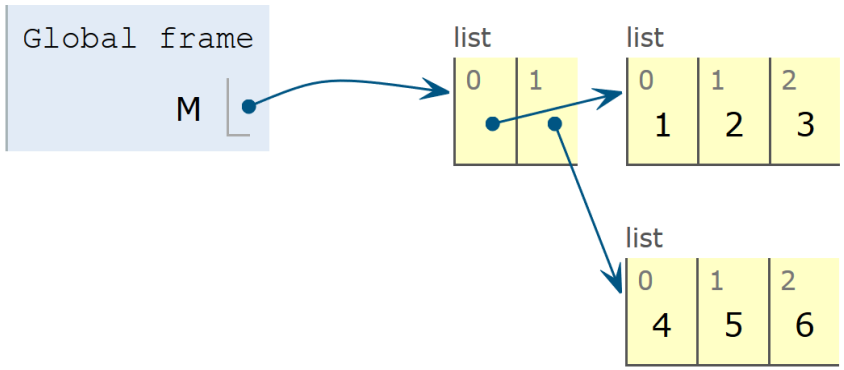
- **Création : `T=[1,2,5,9]`**
- **Accès à l'élément i : `T[i]` (ex `T[2]` vaut 5)**
- **Ecriture : `T[3]=0`$\rightarrow$ T=[1,2,5,0]**
- **Slicing : `T[i:j]` renvoie la liste [T[i],...,T[j-1]] (ex :T[0,2] renvoie [1,2])**
- **T contient la référence vers le tableau donc `T1=T` a pour effet de créer une nouvel référence vers le même tableau ! Manipuler T ou T1 est alors équivalent.**
- **Ajout d'un élément en queue : `T.append(elt)`**
- `[0]*5` $\rightarrow$ **[0,0,0,0,0]**

Implémentation d'une matrice en Python

Implémentation « maison » d'une matrice

On implémente la matrice comme un tableau de tableaux.

Pour la matrice $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$



Création d'une matrice particulière

Pour la matrice suivante :

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

On écrira :

PYTHON

```
M=[[1,2,3],
   [4,5,6]]
```

Remarquez la mise en forme qui rend plus clair le contenu de la matrice.

Création générique d'une matrice

Fonction `creer_ matrice` renvoyant une matrice quelconque dont les coefficients sont calculés par une fonction *f* de *i* et *j*.

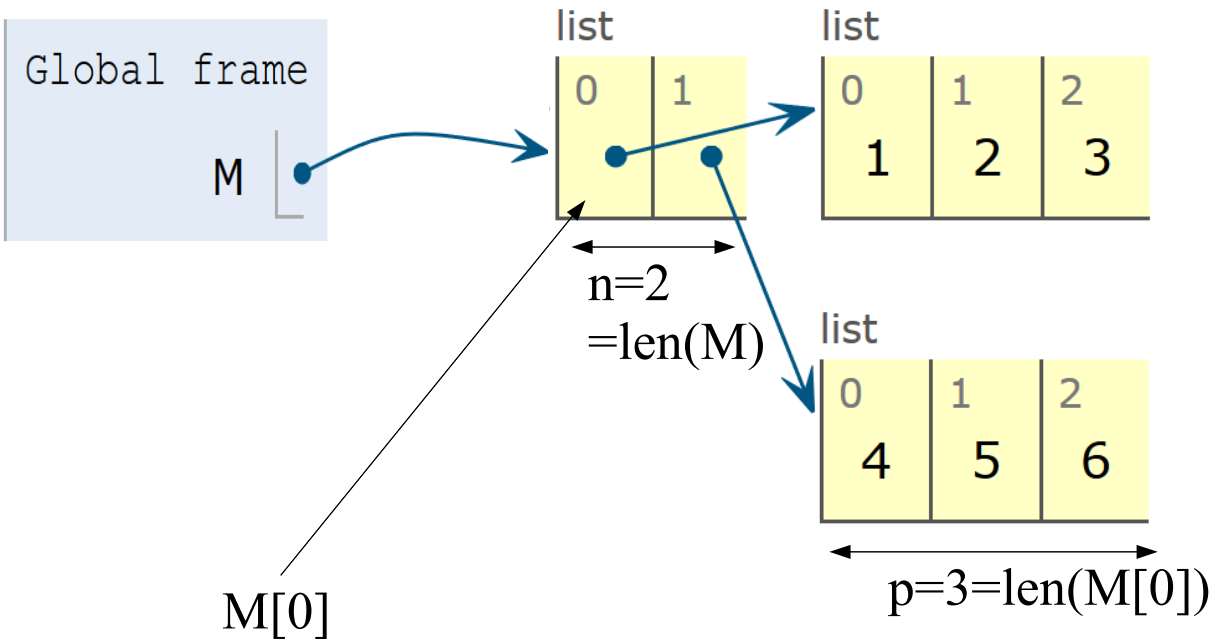
```
1  def creer_matrice(n,p,f):
2      M=[]#M est un tableau de tableaux
3
4      for i in range(n):
5          ligne=[]#ligne a remplir
6          for j in range(p):#remplissage de la ligne i
7              ligne.append(f(i,j))
8          M.append(ligne)#ajout de la ligne i a M
9
10     return M
```

Création générique d'une matrice

Test de la fonction.

```
1 f=lambda i,j:i+j
2 n=4
3 p=5
4 M0=creer_matrice(n,p,f)
```

Dimensions d'une matrice



Accès aux éléments

Accès au coefficient a_{ij}.

`M[i][j]` : en écriture et en lecture !
ex :`M[3][2]=0` met a_{32} à 0.

Décalage de une unité des indices : en maths i et j commencent à 1, en informatique à 0. Il faut donc avoir une convention clair sur la numérotation (i, j). Le mieux est de commencer à 0 aussi en maths. En informatique, on n'a pas le choix !

Accès aux éléments

Accès à une ligne $\rightarrow$ *slicing*

`Li=M[i][:]`

Pour la matrice suivante :

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

`M[1][:]` renvoie `[4,5,6]`

`Li=M[i]` ne remplit pas la fonction attendue. Pourquoi ? Car alors Li contient la référence vers [4,5,6] et non pas une copie de ce tableau ! Si on insère `Li` dans une matrice `M2` alors `M` et `M2` partageront la ligne `Li` ce qui est rarement voulu !

Accès aux éléments

Accès à une colonne : est-ce aussi simple que pour une ligne ?

NON !

```
1 def copie_colonne(M,j):
2     n=len(M)
3     c=[]
4     for i in range(n):
5         c.append(M[i][j])
6     return c
```

Copie d'une matrice

On souhaite souvent travailler sur une copie plutôt que de la modifier *in place*.

Comment copier une matrice `M` ?

`M2=M` ? Bonne idée ?

REPONSE EN TD !

Autres opérations

On peut vouloir calculer la trace, le rang, la commatrice, la transposée, l'inverse, le déterminant d'une matrice. . . ou afficher une matrice ou encore calculer une somme ou un produit de matrice !

Solution 1 : implémenter ces fonctions soit même.

Solution 2 : utiliser les matrices de type `ndarray` des fonctions prédéfinies en ayant bien soin de lire la documentation.

Sommaire

Matrice numpy : le type array

La bibliothèque `numpy` comprend un type `array` permettant de créer des tableaux en mémoire et de réaliser des calculs matriciels. Les éléments ont tous le même type (conversion à la volée si calcul ou modification).

Toutes les opérations usuelles (déterminant, inversion, trace, produit ...) sont déjà implémentés $\rightarrow$ se référer à la documentation.

Nous voyons dans la suite comment manipuler des objets de type `array`.

Création d'une matrice particulière

Pour la matrice suivante :

$$M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

On écrira :

PYTHON

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

Remarquez la mise en forme qui rend plus clair le contenu de la matrice.

Array particuliers

Quelques matrices particulières très utiles !

- **`np.zeros((n,p))` : créé une matrice de taille (n,p) dont tous les coefficients sont nuls.**
- **`np.ones((n,p))` : créé une matrice de taille (n,p) dont tous les coefficients valent 1.**
- **`np.diag(v)` : matrice dont la diagonale est le vecteur v**
- **`np.random.rand((n,p))` : matrice dont les coefficients prennent une valeur aléatoire entre 0 et 1. Très utile pour tester des algorithmes matriciels.**

Accès à un coefficient

Accès à a_{ij} : `M[i,j]`

PYTHON

```
In [18]: M=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7],[5,6,7,8]])

In [19]:M
Out[19]:
array([[1, 2, 3, 4],
       [2, 3, 4, 5],
       [3, 4, 5, 6],
       [4, 5, 6, 7],
       [5, 6, 7, 8]])

In [25]: M[2,2]
Out[25]: 5
```

Accès à une ligne

Slicing : pour accéder à la ligne L_i :`M[i,:]`.

PYTHON

```
In [18]: M=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7],[5,6,7,8]])

In [19]: M
Out[19]:
array([[1, 2, 3, 4],
       [2, 3, 4, 5],
       [3, 4, 5, 6],
       [4, 5, 6, 7],
       [5, 6, 7, 8]])

In [22]: M[1,:]
Out[22]: array([2, 3, 4, 5])
```

Accès à une colonne

Slicing : pour accéder à la colonne C_j :`M[:,j]`.

PYTHON

```
In [18]: M=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7],[5,6,7,8]])

In [19]:M
Out[19]:
array([[1, 2, 3, 4],
       [2, 3, 4, 5],
       [3, 4, 5, 6],
       [4, 5, 6, 7],
       [5, 6, 7, 8]])

In [21]: M[:,3]
Out[21]: array([4, 5, 6, 7, 8])
```

Extraction d'une sous-matrice de matrice

Slicing : `[k:l,m:n]` renvoie la sous-matrice (a_{ij}) avec $k \leq i < l$ et $m \leq j < n$.

PYTHON

```
In [18]: M=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7],[5,6,7,8]])

In [19]:M
Out[19]:
array([[1, 2, 3, 4],
       [2, 3, 4, 5],
       [3, 4, 5, 6],
       [4, 5, 6, 7],
       [5, 6, 7, 8]])

In [24]: M[0:2,2:4]
Out[24]:
array([[3, 4],
       [4, 5]])
```

Dimension d'un array

Utiliser l'attribut `shape`

PYTHON

```
In [10]: M=np.ones((4,5))

In [11]: M
Out[11]:
array([[ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])

In [12]: M.shape
Out[12]: (4, 5)
```

Création générique d'une matrice

Idée :

- **Créer une matrice de taille initialisée à** 0 **avec** `np.zeros`
- **Modifier les valeurs des coefficients suivant leurs positions** (i, j)

```
1  def creer_matrice(f,n,p):
2      M=np.zeros((n,p))#initialisation
3
4      #remplissage
5      for i in range(n):
6          for j in range(p):
7              M[i,j]=f(i,j)
8
9      return M
```

Copie d'une matrice

Slicing intégral ! `M[:,:]` renvoie une copie de l'intégralité de la matrice

PYTHON

```
In [18]: M=np.array([[1,2,3,4],[2,3,4,5],[3,4,5,6],[4,5,6,7],[5,6,7,8]])

In [19]:M
Out[19]:
array([[1, 2, 3, 4],
       [2, 3, 4, 5],
       [3, 4, 5, 6],
       [4, 5, 6, 7],
       [5, 6, 7, 8]])

In [26]: M2=M[:,:]
In [27]: M2
Out[27]:
array([[1, 2, 3, 4],
       [2, 3, 4, 5],
       [3, 4, 5, 6],
       [4, 5, 6, 7],
       [5, 6, 7, 8]])
```

Autres opérations

Voir la documentation !