TP n°12

Matrices

Matrices et Python : premiers pas

Une représentation des matrices en Python

Une matrice est un tableau bidimensionnel de nombres. Cet objet peut se représenter sous diverses formes en Python. Nous choisissons, ici, d'utiliser la fonction

`array(·)`

qui se trouve dans la bibliothèque `numpy`, afin de modéliser les matrices en Python. Nous pouvons motiver ce choix par le fait que la bibliothèque `numpy` contient certaines fonctions permettant d'opérer sur les matrices (e.g. la multiplication, la transposition, ...). Le type associé sera `numpy.ndarray`. Afin d'illustrer quelques manipulations de matrices, nous en stockons une dans une variable notée `A`, puis nous l'affichons et vérifions son type.

```
>>> import numpy as np # chargement de la bibliothèque numpy avec comme alias np
>>> A=np.array([[1,2,3],[4,5,6]])
>>> print(A)
[[1 2 3]
 [4 5 6]]
>>> type(A)
<class 'numpy.ndarray'>
```

Accès aux coefficients d'une matrice via une adresse : attention à l'indexation

Si i est un indice de ligne d'une matrice et j un indice de colonne (avec la numérotation usuelle des lignes et des colonnes en mathématiques), alors nous obtenons coefficient d'adresse (i, j) de cette matrice grâce à l'instruction suivante.

```
matrice[i-1,j-1] # attention aux décalages d'indices
```

Ainsi, lorsque l'on exécute la ligne de commande

`matrice[i,j]`

les indices i et j doivent vérifier

$$0 \leq i \leq \text{nombre de lignes de la matrice} - 1 \qquad \text{et} \qquad 0 \leq j \leq \text{nombre de colonnes de la matrice} - 1$$

sinon on « sort » de la matrice et une erreur se produit.

```
>>> A[0,0]
1
>>> A[1,1]
5
>>> A[1,2]
6
>>> A[2,1]
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    A[2,1]
IndexError: index 2 is out of bounds for axis 0 with size 2
>>> A[1,3]
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    A[1,3]
IndexError: index 3 is out of bounds for axis 1 with size 3
```

Les deux messages d'erreurs précédents renvoyés sont intéressants à analyser : que signifient `axis 0` et `axis 1` ?

Lignes et colonnes d'une matrice : attention à l'indexation (une nouvelle fois)
Si i est un indice de ligne d'une matrice (avec la numérotation usuelle des lignes en mathématiques) alors nous obtenons la ligne n°i grâce à l'instruction suivante.

```
matrice[i-1,:] # attention au décalage d'indice  - utilisation du slicing
```

De manière analogue, si j est un indice de colonne d'une matrice (avec la numérotation usuelle des colonnes en mathématiques) alors nous obtenons la colonne n°j grâce à l'instruction suivante.

```
matrice[:,j-1] # attention au décalage d'indice - utilisation du slicing
```

Les objets obtenus par ces outils de prélèvement, sur un objet de type `numpy.ndarray`, sont encore de type `numpy.ndarray`. Les problèmes de dépassement de format évoqués dans le cas des adresses des coefficients existent également, quand nous prélevons une ligne ou une colonne.

```
>>> A[0,:]
array([1, 2, 3])
>>> type(A[0,:])
<class 'numpy.ndarray'>
>>> A[:,1]
array([2, 5])
>>> type(A[:,1])
<class 'numpy.ndarray'>
>>> A[2,:]
Traceback (most recent call last):
  File "<pyshell#35>", line 1, in <module>
    A[2,:]
IndexError: index 2 is out of bounds for axis 0 with size 2
>>> A[:,3]
Traceback (most recent call last):
  File "<pyshell#36>", line 1, in <module>
    A[:,3]
IndexError: index 3 is out of bounds for axis 1 with size 3
```

Nombre de lignes et nombre de colonnes d'une matrice
Le nombre de lignes d'une matrice est la longueur de sa première colonne. Nous obtenons donc le nombre de lignes d'une matrice grâce à l'instruction suivante.

```
len(matrice[:,0]) # nombre de lignes
```

Le nombre de colonnes d'une matrice est la longueur de sa première ligne. Nous obtenons donc le nombre de colonnes d'une matrice grâce à l'instruction suivante.

```
len(matrice[0,:]) # nombre de colonnes
```

```
>>> len(A[:,0]) # nombre de lignes de A
2
>>> len(A[0,:]) # nombre de colonnes de A
3
```

Deux boucles imbriquées pour parcourir une matrice
Pour parcourir tous les coefficients d'une matrice, nous utiliserons souvent deux boucles imbriquées : une boucle pour le parcours des lignes qui contient elle-même une boucle de parcours des colonnes. Nous donnons ci-dessous un exemple de fonction, dans laquelle est mise en œuvre cette technique.

```
def somme_coefficients(A): # argument: une matrice
                           # retour: la somme des coefficients de la matrice

    nombre_lignes=len(A[:,0]) # calcul du nombre de lignes de A
    nombre_colonnes=len(A[0,:]) # calcul du nombre de colonnes de A

    somme=0 # variable initialisée à 0
            # servant à calculer la somme des coefficients de A de proche en proche

    for i in range(nombre_lignes):
        for j in range(nombre_colonnes): # une boucle dans une autre boucle: boucles imbriquées
            somme=somme+A[i,j]

    return(somme) # la somme est calculée ; on la renvoie
```

Exercices

Exercice 1 (Matrices carrées)
Écrire une fonction `is_square`
- d'argument `A` une matrice ;
- qui retourne `True` si `A` est une matrice est carrée, et `False` sinon.

Exercice 2 (Trace d'une matrice carrée)
Écrire une fonction `trace`
- d'argument `A` une matrice ;
- qui retourne la somme des coefficients diagonaux de `A` (appelée trace de `A`) si `A` est une matrice carrée et la chaîne de caractères `'trace of a non square matrix is not defined'` sinon.

Exercice 3 (Matrices diagonales)
Écrire une fonction `is_diagonal`
- d'argument `A` une matrice ;
- qui retourne `True` si `A` est une matrice est diagonale (donc en particulier carrée), et `False` sinon.

Exercice 4 (Matrices triangulaires supérieures)
Écrire une fonction `is_upper_triangular`
- d'argument `A` une matrice ;
- qui retourne `True` si `A` est une matrice est triangulaire supérieure (donc en particulier carrée), et `False` sinon.

Exercice 5 (Coefficient d'un produit matriciel)
Écrire une fonction `coefficient_produit`
- d'argument `(A,B,i,j)` où `A`, `B` sont des matrices et `i`, `j` sont des entiers ;
- qui retourne le coefficient d'adresse `(i,j)` (avec la numérotation usuelle des lignes et des colonnes en mathématiques) du produit de `A` par `B`, si celui-ci existe, et `'coefficient doesn't exist'` sinon.

Ici, nous n'utiliserons pas la fonction `dot(.)` de la bibliothèque `numpy` pour construire la fonction `coefficient_produit`, mais nous pourrons nous en servir dans les tests pour vérifier nos résultats.