TP n°16

Applications de l'algorithme de Gauß-Jordan

Présentation des objectifs du TP
Au cours du TP n°15, nous avons implémenté l'algorithme de Gauß-Jordan donnant l'unique matrice échelonnée réduite équivalente par lignes à une matrice donnée, cf. fonction

`echelonnee_reduite`.

Pour cela, nous avons préalablement construit 4 fonctions auxiliaires :

`permutation` , `dilatation` , `transvection` , `index_non_zero_coeff_after`.

La matrice échelonnée réduite équivalente par lignes à une matrice donnée joue un rôle prépondérant dans nombre de questions. L'objectif de ce TP est d'utiliser la fonction `echelonnee_reduite` ou d'en construire des variantes, pour résoudre les problèmes d'algèbre linéaire suivants.

1. Calculer le rang d'une matrice.
2. Déterminer si une matrice est inversible.
3. Calculer l'inverse d'une matrice inversible.
4. Déterminer le nombre de solutions d'un système linéaire (aucune, une unique ou une infinité).

Conseil
Les 5 fonctions construites au cours du TP n°15 ont été écrites dans un (seul) fichier. Celles-ci étant utiles pour résoudre les exercices de ce TP, il est conseillé d'écrire les fonctions demandées ici à la suite de celles du TP n°15 (dans le même fichier donc).

Exercice 1

1. Construire une fonction nommée `rang` d'argument
   - `M0` un tableau bidimensionnel de type `numpy.ndarray`

   qui retourne
   - le rang de la matrice `M0`.

   Indication : On pourra écrire une variante de la fonction `echelonnee_reduite`.

2. Tester la fonction `rang` construite (cf. exemples listés ci-dessous).

$$\text{rang}\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}=2 \quad ; \quad \text{rang}\begin{pmatrix}1&1&2&3\\2&1&2&1\\1&1&1&2\end{pmatrix}=3 \quad ; \quad \text{rang}\begin{pmatrix}1&1&1&1\\2&2&2&2\\3&3&3&4\\4&4&4&5\end{pmatrix}=2$$

Exercice 2

1. Construire une fonction nommée `inversible` d'argument
   - `M0` un tableau bidimensionnel de type `numpy.ndarray`

   qui retourne
   - `True` si la matrice `M0` est inversible (en particulier carrée donc) ;
   - `False` sinon.

   Indication : Une matrice carrée est inversible si et seulement si son rang est égal à son format.

2. Tester la fonction `inversible` construite (cf. exemples listés ci-dessous).

$$\begin{pmatrix}0&1&1\\1&0&1\\1&1&0\end{pmatrix} \text{ est inversible} \quad ; \quad \begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix} \text{ n'est pas inversible} \quad ; \quad \begin{pmatrix}1&1&2&3\\2&1&2&1\\1&1&1&2\end{pmatrix} \text{ n'est pas inversible}$$

Ci-après, nous donnons quelques lignes de codes qui pourraient aider à résoudre l'exercice suivant. Elles illustrent des appels des fonctions `eye` et `concatenate` de la bibliothèque `numpy`. Pour avoir des informations sur ces fonctions, on peut saisir `help(np.eye)` et `help(np.concatenate)`.

```
>>> import numpy as np
>>> A=np.array([[2,3,4],[4,2,3],[3,4,2]]) # construction et affectation dans A d'une matrice 3x3
>>> print(A)
[[2 3 4]
 [4 2 3]
 [3 4 2]]
>>> B=np.eye(3) # construction et affectation dans B de la matrice I_3
>>> print(B)
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
>>> np.concatenate((A,B),axis=1) # on accole A et B (dans cet ordre) suivant les colonnes
array([[ 2.,  3.,  4.,  1.,  0.,  0.],
       [ 4.,  2.,  3.,  0.,  1.,  0.],
       [ 3.,  4.,  2.,  0.,  0.,  1.]])
```

Exercice 3

1. Construire une fonction nommée `inverse` d'argument
   - `M0` un tableau bidimensionnel de type `numpy.ndarray` modélisant une matrice inversible

   qui retourne
   - l'inverse de la matrice `M0`.

   Indication : On pourra accoler à `M0` *la matrice identité d'un format ad hoc et s'appuyer sur la fonction* `echelonnee_reduite`*. L'utilisation du slicing pourra également s'avérer utile.*

2. Tester la fonction `inverse` construite (cf. exemples listés ci-dessous).

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}^{-1} = \frac{1}{3}\begin{pmatrix} -1 & 2 \\ 2 & -1 \end{pmatrix} \quad ; \quad \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}^{-1} = \frac{1}{2}\begin{pmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & -1 \end{pmatrix} \quad ; \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \\ 3 & 4 & 1 & 2 \\ 2 & 3 & 4 & 1 \end{pmatrix}^{-1} = \frac{1}{40}\begin{pmatrix} -9 & 11 & 1 & 1 \\ 1 & -9 & 11 & 1 \\ 1 & 1 & -9 & 11 \\ 11 & 1 & 1 & -9 \end{pmatrix}$$

Exercice 4

1. Construire une fonction nommée `number_solutions` d'argument
   - `M0` un tableau bidimensionnel de type `numpy.ndarray` modélisant une matrice augmentée

   qui retourne
   - une des chaînes de caractères `"aucune"`, `"une unique"` , `"une infinité"`, suivant le nombre de solutions que possède le système linéaire associé à la matrice augmentée `M0`.

   Indication : On pourra écrire une variante de la fonction `echelonnee_reduite`*, s'intéresser à d'éventuelles équations d'incompatibilité et au rang de la matrice des coefficients du système.*

2. Tester la fonction `number_solutions` construite (cf. exemples listés ci-dessous).

$$(S_1) : \begin{cases} x - y + 2z = 1 \\ x + 2y - z = 2 \\ 2x + y + z = 3 \end{cases} \quad \text{possède une infinité de solutions}$$

$$(S_2) : \begin{cases} x - y + 2z = 1 \\ x + 2y - z = 2 \\ 2x + y + z = 1 \end{cases} \quad \text{ne possède aucune solution}$$

$$(S_3) : \begin{cases} x - y + z = 1 \\ x + 2y - z = 2 \\ 2x + y + z = 1 \end{cases} \quad \text{possède une unique solution}$$