Informatique TD 8

Faire un compte-rendu de l'ensemble du TD.

I Remarques générales

1. Pour créer et utiliser les matrices, on utilisera la bibliothèque `numpy`.

On commencera par l'instruction `import numpy as np`.

Les matrices sont des tableaux à deux dimensions, et pour les créer on utilise la commande `np.array`.

Exemple : la matrice $\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$, pourra être définie par `A=np.array([[1,2,3],[4,5,6]])`.

Attention, comme pour les listes, l'indexation commence à 0 : ici par exemple

$$1 = A[0,0], 2 = A[0,1], 3 = A[0,2], 4 = A[1,0], 5 = A[1,1] \text{ et } 6 = A[1,2]$$

La taille du tableau est obtenue par la commande `A.shape`. Ici `print(A.shape)` affiche (2,3).

2. Tester le programme suivant :

```
A=np.array([[2,3],[1,3]])
print(A)
for k in range(2):
        A[1,k]=A[1,k]-0.5*A[0,k]
print(A)
```

Que constatez vous ?

Lors de la création d'une matrice le *type* des éléments de la matrice est figé. Par exemple la matrice ci-dessus est une matrice d'entiers. Si on effectue des manipulations sur les lignes ou les colonnes on obtiendra que des entiers, même avec des coefficients réels non entiers. Pour éviter ce problème, on pourra imposer le *type* des éléments de la matrice ; par exemple si on veut imposer le *type* réel pour la matrice ci-dessus, on écrira `A=np.array([[2,3],[1,3]],float)`.

Tester le programme suivant :

```
A=np.array([[2,3],[1,3]],float)
print(A)
for k in range(2):
        A[1,k]=A[1,k]-0.5*A[0,k]
print(A)
```

3. Tester le programme suivant :

```
def double(M):
    taille=M.shape
    n=taille[0]
    p=taille[1]
    for i in range(n):
        for j in range(p):
            M[i,j]=2*M[i,j]
    return M
```

```
A=np.array([[1,2,3],[4,5,6]])
print(A)
resultat=double(A)
print(resultat)
print(A)
```

Que constatez vous ?

Pour éviter ce problème on travaillera sur une copie de la matrice ; on pourra utiliser la commande `C=np.copy(M)`, où C sera une copie de la matrice M.

Tester le programme suivant :

```
def double(M):
    C=np.copy(M)
    taille=C.shape
    n=taille[0]
    p=taille[1]
    for i in range(n):
        for j in range(p):
            C[i,j]=2*C[i,j]
    return C

A=np.array([[1,2,3],[4,5,6]])
print(A)
resultat=double(A)
print(resultat)
print(A)
```

II Exercice sur pivot de Gauss

On considère un système de n équations à n inconnues :

$$(S)\begin{cases} a_{0,0}\,x_0 & + \cdots + & a_{0,j}\,x_j & + \cdots + & a_{0,n-1}\,x_{n-1} & = & b_0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{i,0}\,x_1 & + \cdots + & a_{i,j}\,x_j & + \cdots + & a_{i,n-1}\,x_{n-1} & = & b_i \\ \vdots & & \vdots & & \vdots & & \vdots \\ a_{n-1,0}\,x_1 & + \cdots + & a_{n-1,j}\,x_j & + \cdots + & a_{n-1,n-1}\,x_{n-1} & = & b_{n-1} \end{cases}$$

Les $a_{i,j}$ sont des éléments fixés de $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$ et les x_j sont les inconnues.

On définit la matrice $A = \begin{pmatrix} a_{0,0} & \cdots & a_{0,n-1} \\ \vdots & & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,n-1} \end{pmatrix} \in \mathfrak{M}_n(\mathbb{K})$.

Le système (S) peut s'écrire :

$$A \begin{pmatrix} x_0 \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} b_0 \\ \vdots \\ b_{n-1} \end{pmatrix}$$

On se propose de programmer en Python la méthode du pivot de Gauss pour résoudre un tel système. On supposera que le système est de Cramer, donc qu'il admet un et un seul n-uplet solution $(x_0, \dots, x_{n-1})$.

1. Déterminer la complexité de la méthode du pivot qui permet de trigonaliser une matrice carrée en fonction de la taille de la matrice. On ne tiendra compte que des opérations algébriques.

2. On se propose dans un premier temps de se placer dans le cas où, à chaque étape de la méthode, le coefficient $a_{i,i}$ n'est pas nul. Il convient donc comme pivot.

a. On considère deux entiers i et j de $[\![0, n-1]\!]$.
Écrire une fonction qui prend en entrée une matrice carrée A, les entiers i et j, et un scalaire μ et qui retourne la matrice issue de A par l'opération élémentaire sur les lignes : $L_j \leftarrow L_j + \mu\, L_i$.
On appellera la fonction `transvection_ligne`.

b. Écrire une fonction qui prend en entrée une matrice carrée A de taille n et qui retourne la matrice triangulaire supérieure issue de A par la méthode du pivot de Gauss.
On appellera `triangulation` cette fonction.

c. Appliquer la méthode ci-dessus pour la matrice $A = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 2 & 1 \\ 1 & -2 & 3 \end{pmatrix}$.

d. Écrire une fonction qui prend en entrée une matrice carrée A de taille n, une matrice unicolonne B ayant n lignes, et qui retourne la matrice triangulaire supérieure issue de A par la méthode du pivot de Gauss et la matrice unicolonne issue de B par les mêmes opérations sur les lignes. On appellera la fonction `pivot_Gauss`.

e. Appliquer la méthode ci-dessus pour la matrice $A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 1 & 3 & 4 \end{pmatrix}$ et la matrice $B = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$.

f. Écrire une fonction qui prend en entrée une matrice carrée A de taille n et qui sera, dans l'utilisation de la fonction, triangulaire supérieure, une matrice unicolonne B ayant n lignes, et qui retourne la solution de l'équation $A\,X = B$. On appellera la fonction `resolution`.

g. Résoudre le système $\begin{cases} x + 3y + 4z + t = 48 \\ 3x + 5y - 4z + 2t = -2 \\ 4x + 7y - 2z - 3t = 37 \\ x + 2y + z - 4t = 29 \end{cases}$

3. La méthode ci-dessus est inapplicable si un des $a_{i,i}$ est nul.

Par exemple elle est inapplicable pour le système $\begin{cases} y + z = 1 \\ x + 2y + 2z = 2 \\ x + 3y + 4z = 3 \end{cases}$

On va donc améliorer la méthode en effectuant des permutations sur des lignes, afin de sélectionner si nécessaire un pivot non nul. Pour des problèmes de précision, il est même souhaitable de choisir parmi les pivots possibles celui qui a la plus grande valeur absolue.

a. On considère deux entiers i et j de $[\![0, n-1]\!]$.

Écrire une fonction qui prend en entrée une matrice carrée A, les entiers i et j, et qui retourne la matrice issue de A par l'opération élémentaire sur les lignes : $L_i \longleftrightarrow L_j$.

On appellera la fonction `echange_ligne`.

b. On considère un entier i de $[\![0, n-1]\!]$.

Écrire une fonction qui prend en entrée une matrice carrée A, l'entier i, et qui retourne l'indice p tel que, parmi tous les coefficients $a_{i,i}, a_{i+1,i}, \ldots, a_{n-1,i}$, le coefficient $a_{p,i}$ soit celui qui a la plus grande valeur absolue.

On appellera la fonction `chercher_pivot`.

c. Résoudre le système $\begin{cases} 4x + 2y + 2z = 6 \\ 2x + y + 2z = 5 \\ -x + 3y + 4z = 4 \end{cases}$

III Un exercice proposé à l'ENSAM

On pourra utiliser les deux commandes suivantes de la bibliothèque `numpy` :

- `np.dot(A,B)` qui retourne la produit des deux matrices A et B.
- `np.eye(n)` qui retourne la matrice identité de taille n.

1. Écrire une fonction `trace` prenant pour argument une matrice M et retournant sa trace.

2. Écrire une fonction `puis` de paramètres une matrice A et un entier naturel p et retournant A^p. Cette fonction est à faire en récursif.

3. Écrire une fonction `puisR` de paramètres une matrice A et un entier naturel p et retournant A^p en utilisant :

$$A^{2p} = (A^p)^2 \quad \text{et} \quad A^{2p+1} = A\ (A^p)^2$$

Laquelle des ces deux fonctions `puis` et `puisR` est la plus coûteuse en calcul ?