L3 E - S5 2018-19 --- TD ordinateurs — séance 5 — solutions

1 Exercice 1

Cherchez par vous-même (i.e. avec une feuille et un stylo) les vecteurs propres de la matrice

$$A = \begin{pmatrix} -7 & 1 & 9 \\ -12 & 3 & 12 \\ -6 & 1 & 8 \end{pmatrix}$$

et comparez avec les vecteurs propres trouvés par python.

On voit que python renvoie des approximations numériques (pas des valeurs exactes) et que les vecteurs propres approximés ne sont pas ceux auxquels on pense "naturellement".

Résolution d'un système linéaire

```
In [2]: M = array([[2,1,1],[1,1,1], [1,2,1]])
        X = array([1,2,3])
        Y = solve(M, X)
        print (Y)
```

```
[-1.  1.  2.]
```

```
In [3]: MM = array([[2,1,1],[1,1,1], [5,2,2]])
        X = array([1,2,3])
        Y = solve(MM, X)
        print (Y)
```

```
        ---------------------------------------------------------------------------

        LinAlgError                               Traceback (most recent call last)

        <ipython-input-3-7465219481c3> in <module>()
          1 MM = array([[2,1,1],[1,1,1], [5,2,2]])
          2 X = array([1,2,3])
    ----> 3 Y = solve(MM, X)
          4 print (Y)


        /Users/ef_mac2/anaconda/lib/python3.6/site-packages/numpy/linalg/linalg.py in solve(a, b
        382     signature = 'DD->D' if isComplexType(t) else 'dd->d'
        383     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
    --> 384     r = gufunc(a, b, signature=signature, extobj=extobj)
        385
        386     return wrap(r.astype(result_t, copy=False))
```

```
/Users/ef_mac2/anaconda/lib/python3.6/site-packages/numpy/linalg/linalg.py in _raise_lin
 88
 89 def _raise_linalgerror_singular(err, flag):
---> 90     raise LinAlgError("Singular matrix")
 91
 92 def _raise_linalgerror_nonposdef(err, flag):


LinAlgError: Singular matrix
```

```
In [4]: MM = array([[2,1,1],[1,1,1], [5,2,2]])
        X = array([2,-1,7])
        Y = solve(MM, X)
        print (Y)
```

```
---------------------------------------------------------------------------

LinAlgError                               Traceback (most recent call last)

<ipython-input-4-247617f0c572> in <module>()
  1 MM = array([[2,1,1],[1,1,1], [5,2,2]])
  2 X = array([2,-1,7])
----> 3 Y = solve(MM, X)
  4 print (Y)


/Users/ef_mac2/anaconda/lib/python3.6/site-packages/numpy/linalg/linalg.py in solve(a, b
382     signature = 'DD->D' if isComplexType(t) else 'dd->d'
383     extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 384     r = gufunc(a, b, signature=signature, extobj=extobj)
385
386     return wrap(r.astype(result_t, copy=False))


/Users/ef_mac2/anaconda/lib/python3.6/site-packages/numpy/linalg/linalg.py in _raise_lin
 88
 89 def _raise_linalgerror_singular(err, flag):
---> 90     raise LinAlgError("Singular matrix")
 91
 92 def _raise_linalgerror_nonposdef(err, flag):


LinAlgError: Singular matrix
```

2 Exercice 2

Dans les derniers exemples, on demande à python d'étudier les systèmes :

$$\begin{cases} 2x+y+z=1 \\ x+y+z=2 \\ 5x+2y+2z=3 \end{cases}$$

puis

$$\begin{cases} 2x+y+z=2 \\ x+y+z=-1 \\ 5x+2y+2z=7 \end{cases}$$

Dans les deux cas, il renvoie que ce n'est pas possible. Etes-vous d'accord avec lui ?

La matrice $B = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 1 & 1 \\ 5 & 2 & 2 \end{pmatrix}$ n'est pas inversible (on a , par exemple, $L_3 = 3L_1 - L_2$) et c'est "l'erreur" retournée par python (*singular matrix* : matrice singulière).

Mais s'il est vrai que le 1er système n'a pas de solution, le second système a une infinité de solutions :

$$S = \left\{ \begin{pmatrix} 3 \\ 0 \\ -4 \end{pmatrix} + a \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix},\ a \in \mathbf{R} \right\}$$

3 Exercice 3

Notations : Si $M = (m_{ij})_{1 \leq i \leq n, 1 \leq j \leq p}$ est une matrice à n lignes et p colonnes, on notera L_i (pour $1 \leq i \leq n$) la i-ème ligne de M et C_j (pour $1 \leq j \leq p$), la j-ème colonne de M (Evidemment, s'il ya un risque de confusion, on pourra préciser $L_i(M)$ ou $C_j(M)$ pour préciser qu'il s'agit bien des lignes et colonnes de la matrice M.

Ecrire une fonction `multiplie(M,i,a)` qui renvoie la matrice M avec la ligne L_i multipliée par le nombre a.

Voici une première version

```
In [5]: def multiplie(A,i,a):    #### opération élémentaire Li -> a * Li dans  la matrice M
            for j in range(0,len(A[i])):
                A[i][j] = a * A[i][j]
            return A
```

Et une version plus courte :

```
In [9]: def AutreMultiplie(A,i,a):    #### opération élémentaire Li -> a * Li dans  la matrice M
            A[i] = a * A[i]
            return A

In [10]: M = array([[2,1,1],[1,1,1], [6,7,8]])
         print(M,'\n')
         print(multiplie(M,0,12),'\n')
         print(multiplie(M,2,-1))
         print(AutreMultiplie(M,1,0))
```

```
[[2 1 1]
 [1 1 1]
 [6 7 8]]

[[24 12 12]
 [ 1  1  1]
 [ 6  7  8]]

[[24 12 12]
 [ 1  1  1]
 [-6 -7 -8]]
[[24 12 12]
 [ 0  0  0]
 [-6 -7 -8]]
```

4 Exercice 4

1- Si M est une matrice, l'instruction `M[2]` va donner la ligne L_3 de M (si cette ligne existe...). Et, de façon générale, l'instruction `M[i]` va donner la ligne L_{i+1} de M (si cette ligne existe...). Pouvez-vous expliquer pourquoi~?

2- Ecrire une fonction `echange(M,i,j)` qui échange les lignes L_i et L_j de la matrice M.

3- En déduire une instruction qui retourne la colonne i d'une matrice et une fonction qui échange les colonnes C_i et C_j d'une matrice.

```
In [11]: def echange(M,i,j): #### opération élémentaire Li <-> Lj dans  la matrice M
             n = len(M[0])   #### nombre de colonnes de A
             for k in range(n):
                M[i][k],M[j][k]=M[j][k],M[i][k]
             return M

In [31]: M = array([[2,0,1],[-1,-8,-13], [6,7,8]])
         print(M,'\n')
         M=echange(M,0,2)
         print('on a échangé L1 et L3 : \n\n',M)
         M=echange(M,1,2)
         print('\n \non a échangé L1 et L2 : \n\n',M,'\n')
         M=echange(M,1,2)
         print('\n \non a échangé L1 et L2 : \n\n',M,'\n')

[[  2   0   1]
 [ -1  -8 -13]
 [  6   7   8]]

on a échangé L1 et L3 :

 [[  6   7   8]
 [ -1  -8 -13]
```

```
 [  2   0   1]]


on a échangé L1 et L2 :

 [[  6   7   8]
 [  2   0   1]
 [ -1  -8 -13]]



on a échangé L1 et L2 :

 [[  6   7   8]
 [ -1  -8 -13]
 [  2   0   1]]
```

4.0.1 Attention !!

La fonction qui suit ne donne pas le même résultat...

```
In [32]: def AutreEchange(M,i,j): #### opération élémentaire Li <-> Lj dans  la matrice M
             M[i],M[j]=M[j],M[i]
             return M

In [33]: print(M,'\n')
         M=AutreEchange(M,0,2)
         print('\n \non n\'a pas échangé L0 et L2 !!! : \n\n',M,'\n')
```

```
[[  6   7   8]
 [ -1  -8 -13]
 [  2   0   1]]



on n'a pas échangé L0 et L2 !!! :

 [[  2   0   1]
 [ -1  -8 -13]
 [  2   0   1]]
```

5 Exercice 5

Pour i et j fixés et distincts, $E_{ij} = (e_{ij})$ est la matrice dont l'entrée e_{ij} vaut 1 et toutes les autres entrées sont nulles. Pour tout nombre réel a et pour $1 \leq i, j \leq n$, la matrice carrée d'ordre n égale

à $I_n + aE_{ij}$ est appelée matrice de transvection~; par exemple~:

$$I_4 - 3E_{42} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -3 & 0 & 1 \end{pmatrix}$$

1- Si M est une matrice carrée d'ordre n, quel est le résultat du produit $(I_n + aE_{ij}).M$ (avec $i \neq j$ et $1 \leq i, j \leq 4$) ?

2- Ecrire une fonction `tranvection(M,i,j,a)` qui renvoie la matrice $(I_n + aE_{ij}).M$ si M est une matrice carrée d'ordre n (et $i \neq j$, $1 \leq i, j \leq n$ et $a \in \mathbf{R}$).

```
In [34]: def transvection(M,i,j,a): #### opération élémentaire Li -> Li + a * Lj dans  la matric
             for k in range(0,len(M[i])):
                 M[i][k] = M[i][k] + a * M[j][k]
             return M
```

```
In [43]: M = array([[1,1,1],[-2,-2,-2], [6,6,6]])
         print(M,'\n\n')
         M=transvection(M,0,2,-2)
         print(M,'\n')
         M=transvection(M,1,2,10)
         print(M,'\n')
```

```
[[ 1  1  1]
 [-2 -2 -2]
 [ 6  6  6]]


[[-11 -11 -11]
 [ -2  -2  -2]
 [  6   6   6]]

[[-11 -11 -11]
 [ 58  58  58]
 [  6   6   6]]
```

Voici une version plus courte :

```
In [35]: def AutreTransvection(M,i,j,a): #### opération élémentaire Li -> Li + a * Lj dans  la n
             M[i] = M[i] + a * M[j]
             return M
```

```
In [44]: M = array([[1,1,1],[-2,-2,-2], [6,6,6]])
         print(M,'\n\n')
         M=AutreTransvection(M,0,2,-2)
         print(M,'\n')
         M=AutreTransvection(M,1,2,10)
         print(M,'\n')
```

```
[[ 1  1  1]
 [-2 -2 -2]
 [ 6  6  6]]


[[-11 -11 -11]
 [ -2  -2  -2]
 [  6   6   6]]

[[-11 -11 -11]
 [ 58  58  58]
 [  6   6   6]]
```

6 Exercice 6 : la méthode du pivot de Gauss

6.1 Transformations élémentaires

• $T_1 : L_i \to L_i + \alpha L_j$ (avec $i \neq j$: ajouter à une ligne un multiple d'une autre ligne),
•• $T_2 : L_i \to \alpha L_i$, $\alpha \neq 0$ (multiplier une ligne par un scalaire non nul),
••• $T_3 : L_i \leftrightarrow L_j$ (échanger deux lignes).
Si M est inversible, ces transformations élémentaires permettent de passer de M à I_n :

$$\begin{pmatrix} * & * & * \\ * & * & * \\ & * & * \end{pmatrix} \xrightarrow{T_1} \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \end{pmatrix} \xrightarrow{T_1} \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & * \end{pmatrix} \xrightarrow{T_1} \begin{pmatrix} * & * & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{pmatrix} \xrightarrow{T_1} \begin{pmatrix} * & 0 & 0 \\ 0 & * & 0 \\ 0 & 0 & * \end{pmatrix} \xrightarrow{T_2} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

6.2 Remarques sur ces transformations

1- Cette méthode n'aboutit que si la matrice M de départ est inversible...

2- La transformation T_1 nécessite d'avoir un {*pivot*} *(l'élément diagonal sur lequel on s'appuie) non nul ; lorsqu'il est nul, on utilise la transformation T_3 pour faire apparaître un pivot non nul.*

3- Le schéma ci-dessus est théorique : dans la pratique, il y aura souvent intérêt à utiliser T_2 pour se ramener à des matrices d'entiers ou à effectuer directement des manipulations du type "remplacer L_i par $\alpha L_i + \beta L_j$".

On note que les transformations T_i sont inversibles et que si

$$I_n = T_k T_{k-1} \dots T_2 T_1 M$$

cela signifie que

$$M^{-1} = T_k T_{k-1} \dots T_2 T_1 I_n$$

Avec $B = \begin{pmatrix} 1 & -1 & 2 \\ 2 & 0 & 4 \\ 1 & 1 & 3 \end{pmatrix}$, *on trouve* $B^{-1} = \begin{pmatrix} -2 & 5/2 & -2 \\ -1 & 1/2 & 0 \\ 1 & -1 & 1 \end{pmatrix}$.

- *Cela permet aussi de résoudre des systèmes ; par exemple,* $BX = (7, 10, 4)$ *a pour solution* $X = (3, -2, 1)$.
- *Cette méthode permet aussi de calculer le déterminant (en faisant attention aux cas où on a multiplié une ligne par un scalaire) ; par exemple, ici on voit que* $\det B = 2$.
- *Cette méthode s'applique également de la même façon en agissant sur les colonnes. Par contre, il ne faudra pas l'appliquer en mélangeant les 2 méthodes (par les lignes et par les colonnes).*

6.3 Enoncé de l'exercice

1- Ecrire une fonction `gauss(M)` *qui par suites de transformations élémentaires transforme la matrice carrée* M *en la matrice de l'identité.*

2- Ecrire une fonction `gauss_inv(M)` *qui retourne l'inverse de la matrice carrée* M *en appliquant la méthode de Gauss.*

3- Tester votre programme sur la matrice B *donnée en exemple ci-dessus.*

```
In [45]: def gauss(B):
             A=B
             multiplie(A,0,1/A[0][0])
             transvection(A,1,0,-A[1][0])
             transvection(A,2,0,-A[2][0])
             multiplie(A,1,1/float(A[1][1]))
             transvection(A,0,1,-A[0][1])
             transvection(A,2,1,-A[2][1])
             multiplie(A,2,1/A[2][2])
             transvection(A,0,2,-A[0][2])
             transvection(A,1,2,-A[1][2])
             return A
```

```
In [46]: M = array([[2.0,1.0,1.0],[1.0,1.0,1.0], [6.0,7.0,8.0]])
         print(gauss(M),'\n')
```

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

```
In [47]: def gauss_inv(Z):
             A=Z.copy()
             B=eye(3)
             multiplie(B,0,1/A[0][0])
             multiplie(A,0,1/A[0][0])
             transvection(B,1,0,-A[1][0])
             transvection(A,1,0,-A[1][0])
             transvection(B,2,0,-A[2][0])
             transvection(A,2,0,-A[2][0])
```

```
            multiplie(B,1,1/float(A[1][1]))
            multiplie(A,1,1/float(A[1][1]))
            transvection(B,0,1,-A[0][1])
            transvection(A,0,1,-A[0][1])
            transvection(B,2,1,-A[2][1])
            transvection(A,2,1,-A[2][1])
            multiplie(B,2,1/A[2][2])
            multiplie(A,2,1/A[2][2])
            transvection(B,0,2,-A[0][2])
            transvection(A,0,2,-A[0][2])
            transvection(B,1,2,-A[1][2])
            transvection(A,1,2,-A[1][2])
            return B
```

In [54]:
```
M = array([[1, -1, 2], [2,0,4],[1,1,3]])
MM=M.copy()
H=gauss_inv(MM)
print('\n\n Voici la matrice inverse : \n')
print('H=inverse de M =\n',H,'\n')
print('\n Vérification : \n')
print('M H=\n',dot(M,H))
```

```
 Voici la matrice inverse :

H=inverse de M =
 [[-2.   2.5 -2. ]
 [-1.   0.5  0. ]
 [ 1.  -1.   1. ]]


 Vérification :

M H=
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

In [55]:
```
M = array([[3, 0, 0], [0,4,0],[0,0,2]])
MM=M.copy()
H=gauss_inv(MM)
print('\n\n Voici la matrice inverse : \n')
print('H=inverse de M =\n',H,'\n')
print('\n Vérification :\n')
print('M H=\n',dot(M,H))
```

```
 Voici la matrice inverse :

H=inverse de M =
 [[ 0.33333333  0.          0.        ]
 [ 0.          0.25        0.        ]
 [ 0.          0.          0.5       ]]


 Vérification :

M H=
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

```
In [56]: M = array([[1, 0, 1], [0,1,0],[-1,1,1]])
         MM=M.copy()
         H=gauss_inv(MM)
         print('\n\n Voici la matrice inverse : \n')
         print('H=inverse de M =\n',H,'\n')
         print('\n Vérification : \n')
         print('M H=\n',dot(M,H))
```

```
 Voici la matrice inverse :

H=inverse de M =
 [[ 0.5  0.5 -0.5]
 [ 0.   1.   0. ]
 [ 0.5 -0.5  0.5]]


 Vérification :

M H=
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```