Eléments de correction de l'exercice 11, TP 5 matrices

1. (a) $A = \begin{array}{c} \begin{matrix} f((1,0,0)) & f(0,1,0) & f(0,0,1) \end{matrix} \\ \begin{pmatrix} 1 & 1 & -1 \\ 0 & 2 & 0 \\ -1 & 1 & 1 \end{pmatrix} \end{array} \begin{matrix} (1,0,0) \\ (0,1,0) \\ (0,0,1) \end{matrix}$

   (b)
   ```
   import numpy as np
   np.linalg.eig(A)
   ```
   On obtient ainsi deux valeurs propres distinctes 2 et 0. En lisant les colonnes de la matrices de passage, on obtient :
   deux vecteurs propres non colinéaires pour 2 : $\begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$, un vecteur propre pour 0 : $\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$.
   On conjecture donc $dim(E_2(A)) = 2$ et $dim(E_0(A)) = 1$: comme $2 + 1 = 3$ et A de taille 3×3, A est diagonalisable.

2. (a) $\mathcal{C} = ((1,-1,0),(1,-2,1)(1,0,0))$ donc $Card(\mathcal{C}) = 3 = dim(\mathbb{R}^3)$.
   Il reste à montrer que la famille $\mathcal{C}$ est libre (ou génératrice de $\mathbb{R}^3$).

   (b) $T = \begin{array}{c} \begin{matrix} g(u) & g(v)) & g(e_1) \end{matrix} \\ \begin{pmatrix} 2 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & -1 \end{pmatrix} \end{array} \begin{matrix} u \\ v \\ e_1 \end{matrix}$ car $g(u) = (2,-2,0) = 2u$, $g(v) = (-1,2,-1) = -v$ et $g(e_1) = (0,-2,1) = v - e_1$.

   (c) T est une matrice triangulaire donc les valeurs propres de g sont les coefficients diagonaux de T. $Sp(g) = \{2,-1\}$.
   Pour savoir si g est diagonalisable, soit on utilise python et la syntaxe `eig` (comme pour f), soit mathématiquement, on regarde les dimensions des 2 sous-espaces propres de g. Or $rg(T + I) = ... = 2$ et $rg(T - 2I) = ... = 2$ donc d'après le théorème du rang, $dim(E_{-1}(T) = 3 - 2 = 1$ et de même pour $dim(E_2(T)) = 1$. Donc $dim(E_{-1}(T) + dim(E_2(T) = 2 \neq 3$. Donc T n'est pas diagonalisable, donc g ne l'est pas.
   Attention, pour ceux qui auraient cherché les vecteurs propres de T, on a $dim(E_\lambda(T)) = dim(E_\lambda(g))$ mais les espaces propres ne sont pas égaux ! Donc à rédiger convenablement.

3. (a)
   ```
   def E(M):
       A=np.array([[1,1,-1],[0,2,0],[-1,1,1]])
       B=np.array([[0,-2,-5],[-2,0,4],[1,1,0]])
       X=np.dot(A,M)
       Y=np.dot(M,B)
       return (X==Y).all()
   ```

   (b) Avec les trois points.

   (c) Soit $M \in E$. Supposons que M est inversible. Alors $AM = MB \Rightarrow A = MBM^{-1}$. Autrement dit, A et B sont semblables. Mais A est diagonalisable, alors que B ne l'est pas. Contradiction (cf feuille d'exo). Donc M n'est pas inversible.

   (d) cf feuille d'exo

   (e) Posons $X \in \mathcal{M}_{3,1}(\mathbb{R})$ tel que $AX = 2X$, et $Y \in \mathcal{M}_{3,1}(\mathbb{R})$ tel que ${}^tBY = 2Y$. Alors $AX{}^tY = 2X{}^tY$ et $X{}^tYB = X{}^t({}^tBY) = X{}^t(2Y) = 2X{}^tY$ donc $X{}^tY \in E$.

   (f) En admettant notre conjecture au 1.(b), on dispose d'une base (X_1, X_2) de $E_2(A)$ avec $X_1 = \begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$ et $X_2 = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$. Par ailleurs comme $dim(E_2({}^tB) \geq 1$ (propriété vraie pour tout sous-espace propre), on dispose d'un vecteur $Y = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ non-nul tel que ${}^tBY = 2Y$.
   Alors en utilisant 3.(e), on sait que $X_1{}^tY \in E$ et $X_2{}^tY \in E$.
   Il reste à montrer que la famille $(X_1{}^tY, X_2{}^tY)$ est libre pour obtenir $dim(E) \geq 2$.
   Or $X_1{}^tY = \begin{pmatrix} a & b & c \\ 0 & 0 & 0 \\ a & b & c \end{pmatrix}$ et $X_2{}^tY = \begin{pmatrix} a & b & c \\ 2a & 2b & 2c \\ a & b & c \end{pmatrix}$. Comme (a,b,c) sont non tous nuls, on voit (en regardant la 2e ligne) que les matrices $X_1{}^tY$ et $X_2{}^tY$ ne sont pas colinéaires. Ce que l'on peut démontrer plus rigoureusement comme suit : Soit α, β tels que $\alpha X_1{}^tY + \beta X_2{}^tY = 0$. Alors en regardant la 2e ligne,

on obtient $\begin{cases} 0+2\beta a=0 \\ 0+2\beta b=0 \\ 0+2\beta c=0 \end{cases}$ donc $\beta=0$ (puisque a ou b ou c non nul). Puis en regardant la première ligne, on obtient pour les mêmes raisons $\alpha=0$.

Partie informatique de l'épreuve de modélisation 2019

Des éléments de syntaxe Python, et en particulier l'usage du module numpy, sont donnés en annexe à la fin. Dans tout ce qui suit, les variables n, p, A, M, i, j et c vérifient les conditions suivantes :

— n et p sont des entiers naturels que $p \geq n \geq 2$;

— A est une matrice carrée à n lignes inversible ;

— M est une matrice à n lignes et p colonnes ;

— i et j sont des entiers tels que $0 \leq i \leq n-1$ et $0 \leq j \leq n-1$;

— c est un réel non nul.

On note $L_i \leftarrow L_i + cL_j$ l'opération qui ajoute à la ligne i d'une matrice la ligne j multipliée par c

1. Soit la fonction `initialisation`

```
def initialisation(A):
    n = np.shape(A)[0]
    mat = np.zeros((n,2*n))
    for i in range(0, n):
        for j in range(0, n):
            mat[i,j] = A[i,j]
    return(mat)
```

Pour chacune des affirmations suivantes, indiquer si elle est vraie ou fausse, en justifiant.

L'appel `initialisation(A)` renvoie :

(a) une matrice rectangulaire à n lignes et $2n$ colonnes remplie de zéros ;

(b) une matrice de même taille que A ;

(c) une erreur au niveau d'un `range` ;

(d) une matrice rectangulaire telle que les n premières colonnes correspondent aux n colonnes de A, et les autres colonnes sont nulles.

2. Les trois fonctions `multip`, `ajout` et `permut` ne renvoie rien : elles *modifient* les matrices auxquelles elles s'appliquent.

(a) Que réalise la fonction `multip` ?

```
def multip(M, i, c):
    p = np.shape(M)[1]
    for k in range(0, p):
        M[i,k] = c*M[i,k]
```

(b) Compléter la fonction `ajout`, afin qu'elle effectue l'opération $L_i \leftarrow L_i + cL_j$.

```
def ajout(M, i, j, c):
    p = np.shape(M)[1]
    for k in range(0, p):
        _____ ligne(s) à compléter _____
```

(c) Écrire une fonction `permut` prenant comme argument M, i et j, et qui modifie M en échangeant les valeurs des lignes i et j.

Dans la suite du sujet, l'expression "opération élémentaire sur les lignes" fera référence à l'utilisation de `permut`, `multip` ou `ajout`.

3. Soit la colonne numéro j dans la matrice M. On cherche le numéro r d'une ligne où est situé le plus grand coefficient (en valeur absolue) de cette colonne parmi les lignes j à $n-1$. Autrement dit, r vérifie :

$$|A[r,j]| = \max\{|A[i,j]| \text{ pour } i \text{ tel que } j \leq i \leq n-1\}.$$

Écrire une fonction `rang_pivot` prenant pour argument M et j, et qui renvoie cette valeur de r. Lorsqu'il y a plusieurs réponses possibles pour r, dire (avec justification) si l'algorithme renvoie le plus petit r, le plus grand r ou un autre choix. *(L'utilisation d'une commande max déjà programmée dans Python est bien sûr proscrite.)*

4. Soit la fonction `mystere` :

```
def mystere(M):
    n = np.shape(M)[0]
    for j in range(0, n):
        r=rang\_pivot(M, j)
        permut(M, r, j)
        for k in range(j+1, n):
            ajout(M, k, j, -M[k,j]/M[j,j])
        print(M)
```

(a) On considère dans cette question l'algorithme `mystere` appliqué à la matrice $M_1 = \begin{pmatrix} 3 & 2 & 2 \\ -2 & 0 & 12 \\ 1 & 1 & -3 \end{pmatrix}$: indiquer combien de fois la ligne `print(M)` est exécutée ainsi que les différentes valeurs qu'elle affiche.

(b) De façon générale, que réalise cet algorithme ?

5. Soit la fonction `reduire`, qui modifie M :

```
def reduire(M):
    n = np.shape(M)[0]
    mystere(M)
    for i in range(0, n):
        multip(M, i, 1/M[i,i])
    #Les lignes suivantes sont à compléter :
    -------------------------
```

(a) Compléter la fonction afin que la portion de code manquante effectue les opérations élémentaires suivantes sur les lignes :

pour j prenant les valeurs n-1, n-2, ..., 1, faire :
    pour k prenant les valeurs j-1, j-2, ..., 0, faire :
        $L_k \leftarrow L_k - M[k,j]L_j$

(b) Indiquer ce que réalise cette fonction.

6. Inversion de A.

(a) Écrire une fonction `augmenter` prenant pour argument A et qui renvoie la matrice de taille $(n, 2n)$ définie ainsi :
- dans la partie gauche (composée des n lignes et n premières colonnes), elle contient les coefficients de A ;
- dans la partie droite (composée des n lignes et n dernières colonnes), elle contient les coefficients de la matrice identité de taille n.

Par exemple, pour $\begin{pmatrix} 1 & 2 \\ -1 & 3 \end{pmatrix}$ la fonction renvoie $\begin{pmatrix} 1 & 2 & 1 & 0 \\ -1 & 3 & 0 & 1 \end{pmatrix}$.

(b) À l'aide des fonctions précédentes, proposer un raisonnement permettant d'inverser A.

(c) Écrire une fonction `inverser` prenant pour argument A et qui renvoie la matrice inverse de A, en suivant le raisonnement décrit en question 6b.

(d) Quelle méthode connue venez-vous d'implémenter ?

Annexe : Rappels Python

On considère que le module `numpy`, permettant de manipuler des tableaux à deux dimensions, est importé via `import numpy as np`. Pour une matrice M à n lignes et p colonnes, les indices vont de 0 à $n-1$ pour les lignes et de 0 à $p-1$ pour les colonnes.

Python	Interprétation
`abs(x)`	Valeur absolue du nombre x
`M[i,j]`	Coefficient d'indice (i, j) de la matrice M
`np.zeros((n,p))`	Matrice à n lignes et p colonnes remplie de zéros
`T = np.shape(M)`	Dimensions de la matrice M
`T[0]` ou `np.shape(M)[0]`	Nombre de lignes
`T[1]` ou `np.shape(M)[1]`	Nombres de colonnes
`M[a:b,c:d]`	Matrice extraite de M constituée des lignes a à $b-1$ et des colonnes c à $d-1$: si a (resp. c) n'est pas précisé, l'extraction commence à la première ligne (resp. colonne) si b (resp. d) n'est pas précisé, l'extraction finit à la dernière ligne (resp. colonne) incluse