Matrices, matrices diagonalisables et algorithme du pivot de Gauss

Code de partage avec Capytale : 0ac6-7821270 **Corrigé**

On utilisera les bibliothèques suivantes :

```
import numpy as np
import numpy.linalg as al
```

Exercice 1 - opérations usuelles sur les matrices

On pose :

$$A = \begin{pmatrix} -3 & -3 & 0 \\ 4 & 5 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

1. Définir A avec Python.

   ```
   A=np.array([[-3,-3,0],[4,5,0],[0,1,2]])
   ```

2. Avec Python, calculer A^2, A^3 puis montrer que $A^3 = 4A^2 - A - 6I_3$

   ```
   print(al.matrix_power(A,2),al.matrix_power(A,3))
   al.matrix_power(A,3)==4*al.matrix_power(A,2)-A-6*np.eye(3)
   ```

   La dernière commande renvoie une matrice de « vrai » ou « faux », qui doit donc ne contenir que la valeur « vrai » si l'égalité est vérifiée.

3. Déterminer A^{-1} en fonction de I_3, A et A^2 Vérifier le résultat avec Python.

   De $A^3 = 4A^2 - A - 6I_3$ on déduit $A^3 - 4A^2 + A = 6I_3$ et donc $A\left[-\frac{1}{6}(A^2 - 4*4 + I_3)\right] = I_3$

   donc A est inversible $A^{-1} = -\frac{1}{6}(A^2 - 4A + I_3)$

   on peut alors vérifier avec Python mais on ne peut pas forcément tester l'égalité comme plus haut, car les fractions génèrent des valeurs approchées (par exemple $\frac{1}{3}$ devient $0.333333\ldots$) et donc certaines égalités ne seront pas vérifiées.

   On calcule donc `np.dot(A,-1/6*(al.matrix_power(A,2)-4*A+np.eye(3))` et le résultat doit être « très proche » de I_3

4. Soit $P = \begin{pmatrix} -1 & 0 & 9 \\ 2 & 0 & -6 \\ 2 & 1 & 2 \end{pmatrix}$ Montrer que P est inversible et avec Python calculer $P^{-1}AP$

   Que peut-on en déduire ?

   Avec les opérations $L_2 \leftarrow L_2 + 2L_1$ et $L_3 \leftarrow L_3 + 2L_1$ dans un premier temps, puis en échangeant les nouvelles lignes 2 et 3, on obtient une matrice réduite de P qui est triangulaire et dont les coefficients diagonaux sont non nuls, donc P est inversible.

   Puis avec Python, après avoir défini P, la commande `np.dot(al.inv(P),np.dot(A,P))` on obtient une matrice diagonale, aux arrondis près, et on en déduit que A est diagonalisable.

Exercice 2

On considère le graphe G de sommets $(0, 1, 2, 3, 4)$ et de liste d'adjacence :

$$L = [[1, 2, 4], [0, 2, 3, 4], [0, 1, 3, 4], [1, 2], [0, 1, 2]]$$

1. Ecrire un programme qui permet de remplir la matrice d'adjacence A à partir de `L`. Justifier que A est diagonalisable.

   Comme nous l'avions vu au T.P. 1, il suffit de parcourir la liste d'adjacence dont la liste i donne pour la ligne i de la matrice d'adjacence, les colonnes à remplir par des 1

```
L=[[1, 2, 4], [0, 2, 3, 4], [0, 1, 3, 4], [1, 2], [0, 1, 2]]
A=np.zeros((len(L),len(L)))
for i in range(0,len(L)) :
  for j in L[i]:
    A[i,j]=1
```

   A est symétrique, on peut le vérifier avec Python grâce à la commande `A==np.transpose(A)`, de fait A est diagonalisable.

2. Soit $D = \text{diag}(d_0, \dots d_4)$ la matrice diagonale où l'élément d_i en ligne i et colonne i vaut le degré de i dans le graphe G
   Ecrire un programme qui permet de remplir la matrice D

   Le degré d'un sommet est obtenu en additionnant les coefficients de la ligne correspondante de la matrice d'adjacence, d'où le programme suivant :

```
D=np.zeros((5,5))
for i in range(0,5):
  D[i,i]=sum(A[i,:])
```

3. Soit $M = D - A$
   (a) Justifier que M est diagonalisable.

   L'addition de deux matrices symétriques est une matrice symétrique, donc M est symétrique et de fait diagonalisable.

   (b) Déterminer les valeurs propres de M

   On « triche » ici avec la commande `al.eig` (qui n'est pas au programme) qui donne une première liste qui contient les valeurs propres, puis ensuite les vecteurs propres et il est plus facile de vérifier ensuite qu'elles le sont effectivement et de déterminer les sous-espaces propres.

   (c) Déterminer les sous-espaces propres de M

Exercice 3 - algorithme du pivot de Gauss

1. Opération $L_i \leftrightarrow L_j$
   Ecrire une fonction `Echange(A, i, j)` renvoyant la matrice A avec les lignes i et j échangées.

   On extrait simplement les lignes i et j que l'on stocke sous forme de liste, puis on remplace par ces deux listes les lignes correspondantes de la matrice A. On utilise ici la commande `.copy()` pour contourner une subtilité de Python. La commande `L1=A[i,:]` ne crée en fait qu'une « vue » sur cette ligne et donc lorsque cette ligne est modifiée par la suite, les vues le sont aussi, ce qui fausse nos manipulations.

```
def Echange(A, i, j):
    L1=A[i,:].copy()
    L2=A[j,:].copy()
    A[i,:]=L2
    A[j,:]=L1
    return A
```

2. Opération $L_i \leftarrow L_i + aL_j$
   Ecrire une fonction `Elimine(A, i, j, a)` renvoyant la matrice A après l'opération $L_i \leftarrow L_i + aL_j$

   Comme on ne fait une opération que sur une ligne ici, la « vue » ne pose pas de problème.

```
def Elimine(A, i, j, a):
  A[i,:]=A[i,:]+a*A[j,:]
  return A
```

3. Opération $L_i \leftarrow aL_i$
   Ecrire une fonction `mult(A, i, a)` renvoyant la matrice A après l'opération $L_i \leftarrow aL_i$

   De même :

```
def Mult(A, i, a):
    A[i,:]=a*A[i,:]
    return A
```

4. Utiliser pas-à-pas les fonctions précédentes pour inverser la matrice $A = \begin{pmatrix} 1 & 2 & 1 \\ 1 & 1 & 2 \\ 0 & -1 & -1 \end{pmatrix}$
   On utilisera les opérations sur les lignes avec la matrice suivante :
   $M = \begin{pmatrix} 1 & 2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 2 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 & 0 & 1 \end{pmatrix}$
   On placera l'inverse de A dans une matrice `B` et on vérifiera le résultat.

   Les fonctions des questions **1.**, **2.** et **3.** modifient en fait les matrices, par exemple après `Echange(M,1,2)`, la matrice M aura ses lignes modifiées (autrement dit il n'est pas nécessaire de faire `M=Echange(M,1,2)`
   donc il suffit de faire successivement les opérations