- EXERCICE 1 -

1. **Construire la matrice $A = \begin{pmatrix} -2 & 1 & 1 \\ 1 & -2 & 1 \\ 1 & 1 & -2 \end{pmatrix}$ à l'aide des commandes `"np.ones"` et `"np.eye"`.**

```
A = np.ones(3,3) - 3 * np.eye(3)
A[0,0] = 3
```

2. **Quelle matrice obtient-on après l'exécution des commandes suivantes, `A` étant la matrice de la question 1 ?**

```
A[:,1] = np.zeros(3)
A[0,0] = 3
```

On obtient la matrice $\begin{pmatrix} 3 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & -2 \end{pmatrix}$

- EXERCICE 2 -

Soit B la matrice définie par : $B = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ et on considère les instructions Python suivantes et le résultat associé.

```
>>> B = np.array([[0,1,0],[1,0,0],[0,0,1]])
>>> P = np.array([[1,1,0],[1,-1,0],[0,0,1]])
>>> np.dot( np.dot(al.inv(P),B) , P )
array([[ 1.,  0.,  0.],
       [ 0., -1.,  0.],
       [ 0.,  0.,  1.]])
```

1. **Déduire de la séquence Python précédente que la matrice B est diagonalisable puis donner les valeurs propres de B ainsi que les vecteurs propres associés à ces valeurs propres.**

Le programme calcule $P^{-1}BP$ avec $P = \begin{pmatrix} 1 & 1 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ et renvoie comme résultat une matrice diagonale $D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.

On en déduit que $B = PDP^{-1}$ avec D diagonale donc B est diagonalisable, ses valeurs propres sont $-1, 1$, et les vecteurs propres associés sont les vecteurs colonnes de P dans le même ordre. Ainsi, $E_{-1}(B) = \text{Vect}\left(\begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}\right)$ et $E_1(B) = \text{Vect}\left(\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\right)$.

On considère à présent les instructions Python suivantes :

```
>>> D, V = al.eig(B)
>>> D
array([ 1., -1.,  1.])
>>> V
array([[ 0.70710678, -0.70710678,  0.        ],
       [ 0.70710678,  0.70710678,  0.        ],
       [ 0.        ,  0.        ,  1.        ]])
```

2. **Cette sortie Python est-elle cohérente avec vos résultats de la question 1 ?**

La commande `al.eig(B)` renvoie les valeurs propres de B et les vecteurs propres associés.

Ainsi, D contient les valeurs propres de B ce qui est cohérent avec le résultat de la question 1. Et on remarque que V contient des vecteur proportionnels aux vecteurs propres de B ce qui est cohérent avec le résultat de la question 1.

3. **Au vu des résultats précédentes, que renvoient les instructions suivantes ?**

```
>>> al.matrix_rank(B-np.eye(3,3))
>>> al.matrix_rank(B+np.eye(3,3))
>>> al.matrix_rank(B+4*np.eye(3,3))
```

Ces commandes calculent $rg(B-I)$, $rg(B+I)$ et $rg(B+4I)$.

Or, on a vu que $E_1(B) = \text{Vect}\left(\begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}\right)$ donc $\dim(E_1(B)) = \dim(\text{Ker}(B-I)) = 2$ (2 vecteurs non colinéaires) et d'après le théorème du rang, $rg(B-I) = 3-2 = 1$.

De même, on obtient $rg(B+I) = 3-1 = 2$.

Enfin comme -4 n'est pas valeur propre de B, la matrice $B+4I$ est inversible donc $rg(B+4I) = 3$.

1 Exercices

- EXERCICE 3 -

1. **Écrire une fonction Python `SommeCarrés(A)` qui prend en paramètre une matrice A représentée par un tableau numpy `A` et qui renvoie le réel $\sum_{1 \leqslant i,j \leqslant n} a_{i,j}^2$.**

```
import numpy as np
def SommeCarres(A) :
    (n,p) = np.shape(A)
    S = 0
    for i in range(n) : # parcours des lignes
        for j in range(p) : # parcours des colonnes
            S = S + A[i,j]**2
    return S
```

2. **Écrire une fonction Python `SommeLignes(A)` qui prend en paramètre une matrice A représentée par un tableau numpy `A` et qui renvoie une liste contenant la somme des termes de chaque ligne de A.**

```
def SommeLignes(A) :
    (n,p) = np.shape(A)
    L = np.zeros(n)
    for i in range(n) : # parcours des lignes
        L[i] = np.sum(A[i,:])
    return L
```

3. **On définit la trace d'une matrice A de $\mathcal{M}_n(\mathbb{R})$ comme la somme des ses éléments diagonaux. On note alors $tr(A)$ la trace de A : $tr(A) = \sum_{i=1}^{n} a_{i,i}$.**

Écrire une fonction Python `Trace(A)` qui prend en paramètre une matrice A représentée par un tableau numpy `A` et qui renvoie sa trace.

```
def Trace(A) :
    (n,p) = np.shape(A)
    S = 0
    for i in range(n) : # parcours des lignes
        S = S + A[i,i]
    return L
```

4. **On suppose que A est diagonalisable et on note $\lambda_1, \ldots, \lambda_n$ les valeurs propres de A. On définit alors l'énergie de A, notée $\mathcal{E}(A)$, comme la somme des valeurs abolues des valeurs propres de A : $\mathcal{E}(A) = \sum_{k=1}^{n} |\lambda_k|$.**

La fonction `al.eigvals(A)`, appliquée à un tableau `numpy` représentant une matrice diagonalisable, renvoie un tableau `numpy` contenant les valeurs propres A.
Écrire une fonction PYTHON `Energie(A)` qui prend en paramètre une matrice A représentée par un tableau numpy `A` et qui renvoie l'énergie de A.

```
import numpy.linalg as al
def Energie(A) :
  D = al.eigvals(A)
    return np.sum(abs(D))
```

- EXERCICE 4 -

1. **Écrire une fonction Python `minimax(A)` qui prend en paramètre une matrice A représentée par un tableau numpy `A` et qui renvoie le minimum de la liste contenant les maximums de chaque ligne de A.**

```
def minimax(A) :
    (n,p) = np.shape(A)
    Max = np.zeros(n)
    for i in range(n) : # parcours des lignes
        Max[i] = np.max(A[i,:])
    return np.min(Max)
```

2. **Écrire une fonction Python `maximin(A)` qui prend en paramètre une matrice A représentée par un tableau numpy `A` et qui renvoie le maximum de la liste contenant les minimums de chaque colonne de A.**

```
def maximin(A) :
    (n,p) = np.shape(A)
    Min = np.zeros(p)
    for j in range(p) : # parcours des colonnes
        Min[j] = np.min(A[:,j])
    return np.max(Min)
```

3. **Écrire une fonction Python `Test(A)` qui prend en paramètre une matrice A représentée par un tableau numpy `A` et qui renvoie `1` si son minimax est égal à son maximin et `0` sinon.**

```
def Test(A) :
    if minimax(A) == maximin(A) :
        return 1
    return 0
```