TP Informatique commune 1A : Vecteurs et matrices

On importera la bibliothèque `numpy` au moyen de la commande:

```
import numpy as np
```

Exercice 1 : Des listes aux vecteurs

1. Définir les objets suivants:

   ```
   L1 = [1,2,3]
   L2 = [4,5,6]

   v1 = np.array(L1)
   v2 = np.array(L2)
   ```

   Prédire puis vérifier ce qui est renvoyé par les commandes suivantes:

   ```
   L1 + L2
   v1 + v2
   2 * L1
   2 * v1
   1.5 * L1
   1.5 * v1
   L1 ** 2
   v1 ** 2
   L1.append(7)
   v1.append(7)
   ```

2. Écrire une fonction `prod_scal(v1,v2)` prenant en entrée deux vecteurs de même taille v_1 et v_2 et renvoyant leur produit scalaire. Donner sa complexité.

3. En déduire une fonction `ortho(v1,v2)` renvoyant un booléen indiquant si les vecteurs sont orthogonaux. Votre fonction vous paraît-elle pertinente ? La tester sur les vecteurs:

   ```
   v1 = np.array([1.,1.,1.])
   v2 = np.array([0.1,0.2,-0.3])
   ```

4. Écrire une fonction `prod_vec(v1,v2)` prenant en entrée deux vecteurs de taille 3 et renvoyant leur produit vectoriel. On pourra utiliser la commande `np.zeros(n)` créant un vecteur de taille n rempli de 0.

Exercice 2 : En 2 dimensions : Matrices

Note : on appelle matrice un tableau à deux dimensions.

1. Exécuter et commenter le code suivant:

```
M1 = np.array([[1,2],[3,4],[5,6]])
M2 = np.zeros((3,2))
print(M1)
print(M2)
M2[1][1] = 7
M2[2][0] = 8
M2[0,1] = 9
print(M2)
print(len(M2))
print(len(M2[0]))
print(M2.shape)
print(M2[1])
print(M2[:,0])
print(M1 + 2*M2)
```

2. Écrire une fonction `nombre_zeros(M)` renvoyant le nombre de zéros d'une matrice M. Donner sa complexité.

3. Écrire une fonction `transpose(M)` prenant en entrée une matrice (sous forme de tableau 2D) et renvoyant sa transposée.

4. Écrire une fonction `diag(L)` prenant en entrée une liste L de nombres et renvoyant une matrice carrée dont les coefficients diagonaux sont donnés par les éléments de L et les autres coefficients sont nuls (on dit que c'est une matrice diagonale). Donner sa complexité.

5. Écrire une fonction `dilatation_ligne(M,i,l)` qui prend en entrée une matrice M, un indice de ligne i et un nombre l et qui multiplie tous les coefficients de la i-ème ligne de M par l (la fonction modifie M et ne renvoie rien). La fonction peut être écrite en une seule ligne ! Écrire de même la fonction `dilatation_colonne(M,i,l)`. Donner leurs complexités.

6. Notons L_i la i-ème ligne d'une matrice. Écrire une fonction `transvection_ligne(M,i,j,l)` qui prend en entrée une matrice M, deux indices de ligne i et j et un nombre l et qui remplace L_i par $L_i + l * L_j$ (la fonction modifie M et ne renvoie rien). Écrire de même la fonction `transvection_colonne(M,i,j,l)`. Donner leurs complexités.

7. Écrire une fonction `echange_lignes(M,i,j)` qui prend en entrée une matrice M et deux indices de lignes i et j et qui échange les lignes i et j de M (la fonction modifie M et ne renvoie rien). Tester votre fonction, il peut y avoir des surprises ! Note : on peut réaliser une copie d'un vecteur `v` (ou d'une matrice) comme pour les listes avec `v.copy()`. Écrire de même la fonction `echange_colonnes(M,i,j)`. Donner leurs complexités.

8. Écrire une fonction `produit(M,N)` renvoyant le produit de deux matrices (on supposera que leurs tailles sont compatibles). Donner sa complexité.

9. On dit qu'une matrice carrée est un carré magique si la somme des éléments de toute ligne, colonne ou diagonale est toujours la même. Écrire une fonction `carre_magique(M)` renvoyant un booléen indiquant si M est un carré magique. Donner sa complexité.