TP 11 Python – Matrices comme listes de listes

1. Généralités

Il existe plusieurs manières de représenter les matrices en Python. La première, qui n'utilise aucun nouvel outil, est celle des listes de listes.
Une matrice est alors représentée par une liste dont les éléments sont des listes, qui représentent chacune des lignes de la matrice, dans l'ordre.
Par exemple, la matrice $\begin{pmatrix} 1 & -3 \\ 2 & 5 \\ 6 & 7 \end{pmatrix}$ est représentée par la liste : `[[1, -3], [2, 5], [6, 7]]`.

Ainsi, `M[i]` désigne la i-ème ligne de la matrice M.
On accède à l'élément à la position i, j en écrivant : $\boxed{M[i][j]}$.
⚠ La numérotation des lignes et colonnes commence alors à 0.

Exercice 1. Définir en Python des variables représentant chacune des matrices suivantes :

a) $\begin{pmatrix} 0 & 6 \\ -3 & 2 \end{pmatrix}$ b) I_4 c) $\begin{pmatrix} 3 \\ -1 \\ 1 \end{pmatrix}$ d) $\begin{pmatrix} 1 & 1 & 0 & 5 \end{pmatrix}$

Vous trouverez sur le site une fonction `affiche_matrice(M)` qui affiche joliment une matrice donnée sous la forme d'une liste de liste. Utilisez cette fonction pour vérifier vos résultats !

Exercice 2. Pour qu'une liste de listes de nombres représente réellement une matrice, il faut que chacune des lignes fasse la même longueur.

1. Écrire une fonction `vérifie_matrice(L)` qui prend en entrée une liste de listes et qui vérifie si elle représente bien une matrice.
2. Écrire une fonction `nblignes(M)` qui prend en entrée une matrice et qui renvoie son nombre de lignes.
3. Écrire une fonction `nbcolonnes(M)` qui prend en entrée une matrice et qui renvoie son nombre de colonnes.

Exercice 3 (Un grand classique)**.**
1. Écrire une fonction `valeur_max(M)` qui prend en entrée une matrice et qui renvoie la **valeur** de son coefficient maximal.
2. Écrire une fonction `position_min(M)` qui prend en entrée une matrice et qui renvoie la **position** de son coefficient *minimal*, c'est-à-dire l'indice de ligne et l'indice de colonne correspondants.

Exercice 4.
1. Écrire une fonction `matrice_nulle(n, p)` qui prend en argument deux entiers $n, p \in \mathbb{N}^*$ et qui renvoie la liste de liste représentant la matrice nulle $0_{n,p}$.
2. Écrire une fonction `matrice_identité(n)` qui prend en argument un entier $n \in \mathbb{N}^*$ et qui renvoie la matrice carrée I_n.
   On pourra partir de la matrice `matrice_nulle(n, n)`.
3. Écrire une fonction `matrice_aléatoire(n, p)` qui prend en argument deux entiers $n, p \in \mathbb{N}^*$ et qui renvoie une matrice de taille $n \times p$ dont les coefficients sont choisis uniformément dans l'intervalle $[\![-10, 10]\!]$.

2. Opérations sur les matrices

Pour programmer les opérations matricielles en Python, on commencera toujours par créer une matrice résultat de la bonne taille à l'aide de `matrice_nulle`, puis on modifiera ses coefficients un à un. Par exemple :

```
def somme(M,N):
    n = nblignes(M)
    p = nbcolonnes(N)
    if nblignes(N) != n or nbcolonnes(N) != p :
        print("Les dimensions doivent être les mêmes !")
        return None
    else:
        S = matrice_nulle(n, p)
        for i in range(n):
            for j in range(p):
                S[i][j] = M[i][j] + N[i][j]
        return S
```

Exercice 5.
1. Écrire une fonction `produit_par_scalaire(M,a)` qui prend en argument une matrice M et un scalaire a, et qui renvoie la matrice aM.
2. Écrire une fonction `transposée(M)` qui renvoie la transposée d'une matrice donnée en argument.
3. Écrire une fonction `multiplie(M,N)` qui réalise le produit matriciel lorsque c'est possible, et renvoie `None` précédé d'un message d'erreur sinon.
4. À l'aide de vos fonctions, testez sur plusieurs matrices aléatoires si on a bien l'égalité : $(AB)^\top = B^\top A^\top$.

Exercice 6.
1. Écrire une fonction `puissance(M,n)` qui prend en entrée une matrice M et un entier naturel n, et qui renvoie M^n si la matrice M est carrée, et une erreur sinon.
2. Écrire une fonction `commutent(M,N)` qui vérifie si deux matrices commutent.

3. Pivot de Gauss

Exercice 7 (Matrices échelonnées)**.**
1. Écrire une fonction `compte_zéros_début(L)` qui prend en argument une liste de nombres, et qui compte le nombre de zéros présents **au début** de la liste.
   ⚠ Les matrices seront très vite remplies de flottants, sur lesquels il est vain de faire des tests d'égalité (rappelez vous le résultat de `(0.1+0.1+0.1)==0.3`…)
   On comptera donc plutôt le nombre de coefficients qui sont plus petits que 10^{-6} en valeur absolue !
2. En déduire une fonction `est_échelonnée(M)` qui prend en argument une matrice M et qui renvoie un booléen indiquant s'il s'agit ou non d'une matrice échelonnée.
   Attention aux éventuelles lignes de zéros à la fin !
3. Écrire une fonction `rang_éch(M)` qui prend en argument une matrice, et qui renvoie :
   - Une erreur si la matrice n'est pas échelonnée ;
   - Le rang de la matrice sinon.

Exercice 8 (Opérations élémentaires)**.** Écrire les fonctions suivantes, qui prennent en entrée une matrice et les paramètres décrivant une opération élémentaire, et qui renvoie la matrice obtenue après l'opération élémentaire *sur les lignes* de la matrice. On modifiera directement la matrice passée en argument.

1. `permute(M, i, j)`, qui fait $L_i \leftrightarrow L_j$
2. `dilate(M, i, lamb)`, qui fait $L_i \leftarrow \lambda L_i$, avec $\lambda \neq 0$
3. `transvecte(M, i, j, lamb)`, qui fait $L_i \leftarrow L_i + \lambda L_j$

Exercice 9. Vous trouverez sur le site un fichier contenant une fonction Python à compléter.
Cette fonction prend en argument une matrice, et renvoie :

- une erreur si elle n'est pas carrée ;
- son rang et la mention `None` si elle n'est pas inversible ;
- Son rang et son inverse si elle est inversible.

À vous de compléter cette fonction, qui suit l'algorithme du Pivot de Gauss étendu !