Le but de ce TP est de finir le dernier TP en écrivant l'ensemble des algorithmes sur les matrices vus dans le cours de mathématiques.

Dans tout le TP, on travaillera sur des matrices inversibles, si ce n'est pas le cas les fonctions renverront une erreur.

Avant d'écrire sur l'ordinateur, pensez à écrire ce que vous connaissez sur une feuille. Vous savez résoudre des systèmes 3×3 **et** 4×4**, comment faire pour un système** $n \times n$ **?**

Exercice 1 :

Les deux opérations de base utilisée utilisées pour inverser une matrice sont l'échange de deux lignes et l'ajout une ligne à une autre.

Écrivons ces fonctions.

(1) Écrire la fonction **permuter** qui prend en entrée une matrice M et deux entiers i et j et renvoie une nouvelle matrice M_2 qui a les mêmes valeurs que M avec les lignes i et j échangées.

(2) Écrire la fonction **ajouter** qui prend en entrée une matrice M, un flottant lambda et deux entiers i et j. Cette fonction renvoie la matrice M_2 qui contient les éléments de M sauf pour la j ème ligne à qui on a ajoutée la i ème multipliée par lambda.

Exercice 2 :

Ensuite, on applique le pivot de Gauss pour rendre la matrice triangulaire supérieure. (On utilisera les fonctions de base **permuter** et **ajouter**)

(1) Utiliser la méthode du cours pour écrire la fonctions **rendre_triangulaire** qui prend une matrice M et renvoie les matrices M_2 (matrice triangulaire supérieure) et I_2 (matrice construite à partir de la matrice identité) tels que $M_2 \cdot M^{-1} = I_2$. *On vérifiera que le pivot est non nul. Si il l'est, on inversera les lignes.*

(2) Tester ce code sur des matrices différentes (on se contentera de matrices 3×3)

Exercice 3 :

Il ne reste plus qu'à appliquer la méthode de la remontée. On prend deux matrices, une triangulaire supérieure M et une quelconque B et on calcule B_2 tel que si X est une matrice tel que $M \cdot X = B$ alors $X = B_2$.

Il s'agit en fait de rendre la matrice triangulaire inférieure. On reprend donc l'algorithme de l'exercice 2 avec le pivot en bas à gauche au lieu d'en haut à droite.

Écrire la fonction **remontee** qui permet de faire ça.

Tester cette fonction.

Exercice 4 :

Écrire la fonction **diagonale_1** qui prend en compte deux matrices M et $M2$ et qui modifie les matrices de la même façon afin d'avoir des 1 sur la diagonale.

Exercice 5 :

Conclure en écrivant la fonction **pivot_Gauss** qui prend une matrice M et renvoie son inverse.

Tester cette fonction pour résoudre un système.

Exercice 6 :

Toutes ces opérations sont bien sûr possible en Python dans la bibliothèque **numpy**

Dans un nouveau fichier, on va tester les fonctions des deux derniers TP avec **numpy**. On pourra comparer l'efficacité de celles-ci. Pour cela on utilisera la bibliothèque **time** qui permet de gérer le temps.

En programmation, sauf si le professeur vous le demande explicitement, on ne réinvente jamais la roue. On utilise les bibliothèques et les fonctions existantes répondant à notre problème

(1) Il existe deux façons de déclarer une matrice avec **array** que l'on a déja vu et avec **matrix** que l'on va découvrir.

a. Déclarer deux matrices 2×2 avec les deux fonctions.

```
A1 = np.array([[1,2],[3,4]])
B1 = np.array([[4,5],[6,7]])
A2 = np.matrix([[1,2],[3,4]])
B2 = np.matrix([[4,5],[6,7]])
```

b. Tester les commandes A1+B1, A1∗B1 et A1∗∗2 avec les deux types de matrices.

c. Quel est le type qui donne le bon résultat ?

(2) Tester les commandes suivants, dire à quoi elles servent :

a. A.shape

b. A.T

c. A.I

Il est facile aussi de déclarer une matrice aléatoire np.matrix(np.random.randin(100,size=(3,3))), nulle np.matrix(np.zeros([10,10])) ou identité np.matrix(np.eye(10))