ALGORITHME DU PIVOT DE GAUSS

But. Implémenter en Python des algorithmes, basés sur la méthode du pivot de Gauss, permettant :

1. d'établir l'inversibilité d'une matrice,
2. de calculer l'inverse d'une matrice inversible,
3. de résoudre un système d'équations linéaires.

1 Implémentation des matrices et des opérations élémentaires sur les lignes d'une matrice

1.1 Implémentation des matrices

Dans tout ce TP, on représentera une matrice par la liste de ses lignes, et chacune de ses lignes par une liste (donc une matrice sera représentée par une liste de listes de même longueur).

Exemple. Par exemple, la matrice $M = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ sera représentée par :

```
>>> M = [[1,2,3],[4,5,6]]
```

On accède alors à la ligne i d'une matrice M ainsi encodée par `M[i]`, et à son terme d'indice (i, j) par `M[i][j]`, en sachant que **l'indexation des listes en Python commence à** 0.

Remarque. Le module `numpy` dispose d'un type `matrix`, analogue au type `array`, mais avec des commandes adaptées aux matrices (comme la multiplication). On ne l'utilisera pas ici.

1.2 Implémentation des opérations élémentaires

On rappelle que les opérations élémentaires sur les lignes d'une matrice sont les suivantes :

Nom	Codage	Transformation
Permutation	$L_i \leftrightarrow L_j$	Pour $i \neq j$, échanger L_i et L_j.
Dilatation	$L_i \leftarrow aL_i$	Pour $a \in \mathbb{K}$ non nul, remplacer L_i par aL_i.
Transvection	$L_i \leftarrow L_i + aL_j$	Pour $i \neq j$ et $a \in \mathbb{K}$, remplacer L_i par $L_i + aL_j$.

L'encodage choisi pour les matrices permet l'implémentation suivante de ces opérations :

- *Permutation* $L_i \leftrightarrow L_j$.

```
def permutation(M,i,j):
    M[i], M[j] = M[j], M[i]
```

- *Dilatation* $L_i \leftarrow aL_i$. Deux possibilités :

```
def dilatation(M,i,a):
    for k in range(len(M[i])):
    M[i][k] *= a
```

```
def dilatation(M,i,a):
    M[i] = [a*x for x in M[i]]
```

- *Transvection* $L_i \leftarrow L_i + aL_j$. Deux possibilités :

```
def transvection(M,i,j,a):
    for k in range(len(M[i])):
        M[i][k] += a*M[j][k]
```

```
def transvection(M,i,j,a):
    Li, Lj, p = M[i], M[j], len(M[i])
    M[i] = [Li[k]+a*Lj[k] for k in range(p)]
```

On prendra garde au fait que ces fonctions ne renvoient rien, mais modifient la matrice (liste de listes) `M` donnée en argument.

2 Implémentation des algorithmes

2.1 Algorithme du pivot de Gauss

- Voici l'algorithme du pivot de Gauss pour échelonner en ligne une matrice $M = (m_{i,j})_{\substack{0 \leqslant i \leqslant n-1 \\ 0 \leqslant j \leqslant p-1}}$, et une implémentation possible en Python, avec l'encodage choisi pour les matrices :

Pseudo code

Algorithme du pivot simple
$i_0 = 0$ # indice de ligne du prochain pivot
pour j **allant** de 0 à $p-1$:
    # recherche d'un pivot en colonne j
    chercher $k \in [\![i_0; n]\!]$ tel que $m_{k,j} \neq 0$
    # traitement de la colonne j si besoin
    si un tel k existe:
        faire l'opération $L_{i_0} \leftrightarrow L_k$
        pour i **allant** de $i_0 + 1$ à $n-1$:
            faire l'opération $L_i \leftarrow L_i - \frac{m_{i,j}}{m_{i_0,j}} L_{i_0}$
        incrémenter i_0 de 1

Python

```
def pivot(M):
    n, p, i0 = len(M), len(M[0]), 0
    for j in range(p):
        # recherche d'un pivot en colonne j
        k = i0
        while k < n and M[k][j] == 0:
            k += 1
        # traitement de la colonne j si besoin
        if k < n:
            permutation(M,i0,k)
            for i in range(i0+1,n):
                a = -float(M[i][j])/M[i0][j]
                transvection(M,i,i0,a)
            i0 += 1
```

- Pour le calcul de l'inverse d'une matrice inversible et la résolution d'un système d'équations linéaires, il faut continuer l'algorithme en transformant chaque pivot en 1 (par dilatations) et en faisant apparaître des 0 au dessus de chaque pivot (par transvections). On peut faire cela au moment du traitement de chaque colonne en complétant le code précédent :

Pseudo code

Algorithme du pivot complet
$i_0 = 0$ # indice de ligne du prochain pivot
pour j **allant** de 0 à $p-1$:
    # recherche d'un pivot en colonne j
    chercher $k \in [\![i_0; n]\!]$ tel que $m_{k,j} \neq 0$
    # traitement de la colonne j si besoin
    si un tel k existe:
        faire l'opération $L_{i_0} \leftrightarrow L_k$
        faire l'opération $L_{i_0} \leftarrow \frac{1}{m_{i_0,j}} L_{i_0}$
        pour i **allant** de 0 à $n-1$:
            si $i \neq i_0$:
                faire l'opération $L_i \leftarrow L_i - m_{i,j} L_{i_0}$
        incrémenter i_0 de 1

Python

```
def pivot_complet(M):
    n, p, i0 = len(M), len(M[0]), 0
    for j in range(p):
        # recherche d'un pivot en colonne j
        k = i0
        while k < n and M[k][j] == 0:
            k += 1
        # traitement de la colonne j si besoin
        if k < n:
            permutation(M,i0,k)
            dilatation(M,i0,1./M[i0][j])
            for i in range(n):
                if i != i0:
                    transvection(M,i,i0,-M[i][j])
            i0 += 1
```

Remarque. Pour limiter les erreurs d'arrondis causés par les calculs sur les `float` et pour éviter le problème du test d'égalité à zéro, il est préférable de chercher, dans chaque colonne, le plus grand pivot possible en valeur absolue (au lieu de prendre le premier qui se présente).

Les trois lignes de code correspondant à cette recherche de pivot dans les fonctions `pivot` et `pivot_complet` ci-dessus doivent alors être remplacées par les lignes suivantes :

```
pivot, k = 0, n
for i in range(i0,n):
    if abs(M[i][j]) > abs(pivot):
        pivot, k = M[i][j], i
```

2.2 Inversibilité, et le cas échéant calcul de l'inverse, d'une matrice carrée

Dans toute cette partie, la matrice M est supposée carrée de taille n. Pour que les algorithmes du pivot de la partie précédente ne modifie pas la matrice M, on va les faire agir sur une copie de M, qu'on peut obtenir via la commande `deepcopy` du module `copy`.

```
from copy import deepcopy
```

- La matrice carrée M est inversible si et seulement si elle est équivalente par ligne à une matrice échelonnée de rang n (i.e. une matrice échelonnée dont le dernier terme diagonal est non nul).

  On peut alors implémenter un test d'inversibilité des matrices carrées comme suit :

```
def inversible(M):
    A = deepcopy(M)
    pivot(A)
    return A[-1][-1] != 0
```

- Si M est inversible, alors l'algorithme du pivot complet appliqué à la matrice augmentée $M|I_n$ transforme cette matrice en la matrice $I_n|M^{-1}$.

  On peut alors implémenter un algorithme de calcul de l'inverse d'une matrice inversible comme suit, en commençant par une fonction définissant la matrice I_n et une autre créant la matrice augmenter $A|B$ à partir de deux matrices A et B ayant le même nombre de lignes :

```
def indentite(n):
    return [[int(i==j) for j in range(n)] for i in range(n)]
    # int(bool) renvoie 0 ou 1 selon que bool est faux ou vrai

def augmente(A,B):
    return [A[i]+B[i] for i in range(len(A))]

def inverse(M): # M est ici supposée inversible
    n = len(M)
    A = augmente(M,identite(n))
    pivot_complet(A)
    return [A[i][n:] for i in range(n)]
```

2.3 Résolution d'un système d'équations linéaires

- Si M est inversible (donc carrée) de taille n, alors l'algorithme du pivot complet appliqué à la matrice augmentée $M|B$, où B est une matrice colonne à n lignes, transforme cette matrice en la matrice $I_n|X$ où X est l'unique solution du système $MX = B$ (exercice : pourquoi ?).

  On peut alors implémenter un algorithme de résolution des systèmes de Cramer comme suit :

```
def resolution_Cramer(M,B): # M est ici supposée inversible
    n = len(M)              # B est une matrice colonne
    A = augmente(M,B)
    pivot_complet(A)
    return [A[i][n:] for i in range(n)]
```

- Dans le cas général, l'algorithme du pivot complet appliqué à la matrice augmentée $M|B$ transforme cette matrice en une matrice échelonnée, dont la dernière ligne non nulle indique la compatibilité du système $MX = B$, et qui permet le cas échéant d'exprimer les inconnues principales de ce système en fonction des inconnues secondaires.

  L'exploitation de ces résultats pour un affichage automatisé des solutions est plus délicat, et est laissé en exercice.