# TP : Système linéaire d'équations - Pivot de Gauss

## 1. Introduction

On cherche à programmer l'algorithme du pivot de Gauss pour résoudre un système d'équations du type suivant :

$$\begin{cases} x_1+2x_2+3x_3=1 \\ 4x_1+5x_2+6x_3=4 \\ 7x_1+8x_2+10x_3=-1 \end{cases}$$

Le système peut s'écrire sous forme matricielle AX=B avec :

$$A=\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix} \qquad X=\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \qquad B=\begin{pmatrix} 1 \\ 4 \\ -1 \end{pmatrix}$$

Une matrice sera dans un premier temps définie en Python comme une liste de listes :

```
>>> A=[[1,2,3],[4,5,6],[7,8,10]]
>>> print(A)
[[1, 2, 3], [4, 5, 6], [7, 8, 10]]
```

Le programme `Gauss Elève Incomplet 1.py` (à enregistrer dans votre répertoire élève) présente quatre premières fonctions :

- La fonction **`Affiche(M)`** affiche une matrice sous la forme plus lisible suivante :

```
>>> Affiche(A)
1 2 3
4 5 6
7 8 10
```

- La fonction **`Copie(M)`** retourne une matrice R identique à la matrice M. Pour rappel, le type liste est mutable c'est à dire qu'il est possible de modifier une partie de l'objet sans changer l'adresse de l'objet (*modification sur place*). Pour travailler sur une matrice M sans la modifier, il faut en créer une copie. La simple affectation R=M ne convient pas car R et M pointent alors vers la même adresse : la modification de l'une des matrices se répercute sur l'autre.
- La fonction **`Colonne(M,i)`** extrait les termes de la colonne i de la matrice M (rappel : la numérotation des listes de Python commence à 0) :

```
>>> Colonne(A,1)
[2, 5, 8]
```

- La fonction **`Dilat(M,i,a)`** retourne la matrice dans laquelle a été effectuée la dilatation $L_i \leftarrow aL_i$.

```
>>> A=[[1, 2, 3, 1], [4, 5, 6, 4], [7, 8, 10, -1]]
>>> S=Dilat(A,1,10)
>>> Affiche(S)
1 2 3 1
40 50 60 40
7 8 10 -1
```

Ce TP vous propose :

- D'analyser les fonctions données.
- D'écrire les autres fonctions élémentaires permettant au final de programmer la méthode de résolution par pivot de Gauss de systèmes linéaires d'équations (sans recherche du pivot).

Par principe, chacune de ces fonctions commencera par faire une copie de la matrice donnée en argument.

## 2. Fonctions à écrire

Ecrire une fonction **`Augmente(M,N)`** qui retourne la matrice augmentée (M|N) et affiche un message d'erreur si les tailles des matrices M et N ne sont pas compatibles.

```
>>> A=[[1,2,3],[4,5,6],[7,8,10]]
>>> B=[1,4,-1]
>>> S=Augmente(A,B)
>>> Affiche(S)
1 2 3 1
4 5 6 4
7 8 10 -1
```

Ecrire une fonction **`Transvec(M,i,j,a)`** qui retourne la matrice dans laquelle a été effectuée la transvection $L_j \leftarrow L_j + aL_i$.

```
>>> A=[[1,2,3],[4,5,6],[7,8,10]]
>>> S=Transvec(A,0,1,-4)
>>> Affiche(S)
1 2 3
0 -3 -6
7 8 10
```

Ecrire une fonction **`Echelon(M)`** qui retourne la matrice échelonnée par lignes (MEL) par une suite d'opérations élémentaires sur les lignes. Cette fonction utilise principalement la fonction **`Transvec`**.

```
>>> A=[[1,2,3],[4,5,6],[7,8,10]]
>>> S=Echelon(A)
>>> Affiche(S)
1 2 3
0.0 -3.0 -6.0
0.0 0.0 1.0
```

Consigne : on programmera l'algorithme présenté dans le cours sans choix du pivot.

Ecrire une fonction **`Diagnorm(M)`** qui a pour argument une matrice échelonnée par lignes M et qui retourne la matrice avec les coefficients diagonaux normalisés. Cette fonction utilise principalement la fonction **`Dilat`**.

```
>>> A=[[1,2,3],[0,2,4],[0,0,5]]
>>> S=Diagnorm(A)
>>> Affiche(S)
1.0 2.0 3.0
0.0 1.0 2.0
0.0 0.0 1.0
```

### 3. Pivot de Gauss

En utilisant les différentes fonctions précédentes, écrire une fonction **`Gauss(A,B)`** qui retourne la solution du système d'équations AX=B, la réduction se fera par opérations élémentaires sur les lignes.

```
>>> A=[[1,2,3],[4,5,6],[7,8,10]]
>>> B=[1,4,-1]
>>> Gauss(A,B)
[-7.0, 16.0, -8.0]
```

### 4. Exploitation du programme

**4.1** A l'aide de votre programme, donner les solutions du système :

$$\begin{cases} x_1 + 3x_2 + 5x_3 + 7x_3 = 12 \\ 3x_1 + 5x_2 + 7x_3 + x_3 = 0 \\ 5x_1 + 7x_2 + x_3 + 3x_3 = 4 \\ 7x_1 + x_2 + 3x_3 + 5x_3 = 16 \end{cases}$$

**4.2** Une usine fabrique trois produits P1, P2 et P3. Ces produits passent sur trois ateliers différents A, B et C pour y subir des opérations avec les temps de passages suivants :

- P1 passe 2h dans l'atelier A, 1h dans l'atelier B et 1h dans l'atelier C
- P2 passe 5h dans l'atelier A, 3h dans l'atelier B et 2h dans l'atelier C
- P3 passe 3h dans l'atelier A, 2h dans l'atelier B et 2h dans l'atelier C

Lors d'un programme de fabrication, la charge horaire des différents ateliers a été de 104h pour A, 64h pour B et 55h pour C. Quelles sont les quantités de produits P1, P2 et P3 fabriquées ?

## 5. Utilisation de la bibliothèque Numpy

Numpy ajoute à Python les tableaux (`array`) qui permettent de représenter les matrices.
Après avoir importé la bibliothèque Numpy (`>>> import numpy as np`), vous pouvez :

- Reprendre la programmation des différentes fonctions en gérant les matrices sous le format tableau :

```
>>> A=np.array([[1,2,3],[4,5,6],[7,8,10]])
>>> print(A)
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8 10]]
```

- Le module linalg de Numpy propose des fonctions préprogrammées d'algèbre linéaire. Ainsi, la fonction `linalg.inv(A)` retourne l'inverse de la matrice A. Par ailleurs, la fonction `dot(A,B)` effectue le produit entre les deux matrices A et B (*attention : A*B effectue le produit des matrices terme à terme*).
  A l'aide de ces fonctions, écrire une fonction `Solution_np(A,B)` qui retourne la solution du système d'équations AX=B.

```
>>> import numpy as np
>>> A=np.array([[1,2,3],[4,5,6],[7,8,10]])
>>> B=np.array([1,4,-1])
>>> Solution_np(A,B)
array([ -7.,  16.,  -8.])
```

- Finalement, consulter l'aide de la fonction `solve` (`help(np.linalg.solve)`)...

## 6. Complexité et temps de calcul

Le module time possède la fonction `time.clock()` qui renvoie le temps CPU (*Central Processing Unit = Processeur*) en secondes. On peut ainsi placer `tps1=time.clock()` au début d'un programme et `tps2=time.clock()` à la fin d'un programme et faire la différence pour avoir une estimation de son temps d'exécution.

- Evaluer le temps d'exécution de la fonction **`Gauss(A,B)`** pour un système de taille 3 et pour un système de taille 4.
- Analyser l'évolution de ces temps vis à vis de la complexité de l'algorithme en $n^3$.