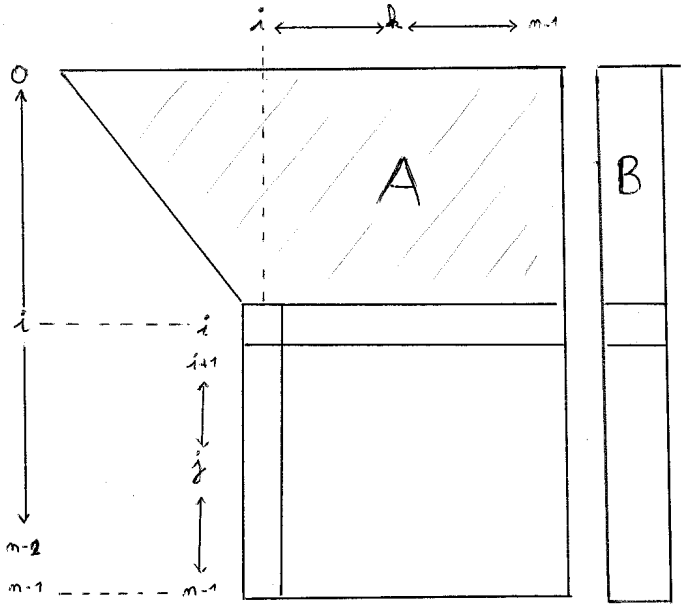
# **TD n°07** : Le Pivot de Gauss

12 décembre 2017




Récupérer le fichier `gauss.py` qui se trouve dans l'espace de partage du réseau de l'établissement.
Copiez-le sur le bureau de votre ordinateur.
Vous y trouverez les consignes suivantes :

```
##########################################################################################################

# Programmation de l'algorithme du Pivot de Gauss pour un système de Cramer

# On souhaite résoudre le système de Cramer AX=B
# La matrice A sera codée sous la forme d'une liste de listes A de taille nxn
# Le terme aij sera alors obtenu grace à la commande A[i][j] pour i et j dans [|0,n-1|]
# le vecteur B sera codé sous la forme d'une liste de valeurs : B = [b0,b1..., bn-1]
# On pourra tester les programmes suivants avec les matrices A et B suivantes :

A = [[1,0,-1],[1,2,-1],[0,2,1]]
B = [0,2,-1]

##########################################################################################################
```

```
# I] Programmation des opérations élémentaires

# 1) Recherche d'un pivot

# Compléter la fonction suivante qui dans une colonne j d'une matrice A, renvoie
# l'indice du premier coefficient aij non nul pour j>=i. Ce coefficient sera le pivot.
# Le système étant de Cramer, on est assuré qu'un tel coeffient existe bien

def pivot(A,i):
    n = len(A)                # Ne sert à rien !!
    j=i
    while A[j][i] == 0:
        ...

pivot(A,0)

# 2) Echange de lignes : Li <-> Lj

# Compléter la fonction suivante qui échange les lignes Li et Lj d'une matrice A
# Attention, l'indexation des lignes et colonnes commence à 0

def echange_lignes (A,i,j):
    n = len(A)
    for k in range(...) :
        ...

echange_lignes(A,0,2)

# 3) Transvection : Lj <- Lj + mu.Li

# Compléter la fonction suivante qui dans une matrice A, remplace la ligne Lj par la ligne Lj + mu.Li

def transvection(A,i,j,mu):
    n = len(A)
    for k in range(...) :
        ...

transvection(A,0,1,2)

#############################################################################################

# II] Programmation de la résolution du système


# 1) ETAPE 1 : Triangularisation du système

# Complèter la procédure suivante qui transforme le système AX = B en un système triangulaire
# On pensera à bien effectuer sur le vecteur B les mêmes opérations d'échange et de transvection
# que sur la matrice A

def triangularisation(A,B):
    n = len(A)
    for i in range(0,n-1):          # i parcourt ici toutes les lignes de 0 à n-2
        j = pivot(...)
        ...                         # on échange les lignes Li et Lj puis
                                    # on annule tous les coefficients sous a[i][i]

triangularisation(A,B)


# 2) ETAPE 2 : Résolution du système triangulaire
```

```
# Construire la fonction qui, à partir d'un système triangulaire, renvoie le vecteur solution.

def soltrig(A,B):
    n = ln(A)
    l =[]                       # l est la liste dans lesquelles nous allons stocker les solutions
    for i in range(0,n-1):      # (n-1)-i représente l'indice de la solution recherchée
        ...

soltrig(A,B)


# 3) ETAPE 3 : Programme de résolution globale

# A l'aide des procédures précédentes, construire une fonction de résolution du système AX = B
# par la méthode de Gauss.

#############################################################################################
```

Rappel des fonctions `solve()` de sympy et numpy pour résoudre : $\begin{cases} ax+by+cz=0 \\ a'x+b'y+c'z=0 \\ a''x+b''y+c''z=0 \end{cases}$.

Python

```
# Avec sympy ##############################

from sympy import var, solve
var('x,y,z')
solve([a*x+b*y+c*z,a'*x+b'*y+c'*z,a"*x+b"*y+c"*z],[x,y,z])

# Avec numpy ##############################

from numpy.linalg import solve
from numpy import array
A = array([[a,b,c],[a',b',c'],[a",b",c"]])
B = array([u,v,w])
solve(A,B)
```