TP 11 : Méthode de Gauss

1 Rappel : formalisation de l'algorithme du pivot de Gauss

On cherche à résoudre AX=Y avec A=$(a_{i,j})_{1\leqslant i,j\leqslant n}$ inversible.

Étape 1 : Mise sous forme triangulaire

pour i de 0 à (n-2) faire
    Trouver k entre i et n-1 tel que $|a_{k,i}|$ soit maximal
    Échanger L_k et L_i
    Pour m variant de i+1 à n-1 faire
        $L_m \leftarrow L_m - \frac{a_{m,i}}{a_{i,i}} L_i$

Étape 2 : Phase de remontée

On a $x_i = \frac{1}{a_{i,i}} \left(y_i - \sum_{k=i+1}^{k=n-1} a_{i,k} x_k \right)$

pour i de (n-1) à 0 faire
    pour k de i+1 à n-1 faire
        $y_i \leftarrow y_i - a_{i,k} x_k$
    $y_i \leftarrow \frac{y_i}{a_{i,i}}$

2 Écriture des fonctions auxiliaires

*Vous prendrez soin de tester au fur et à mesure de leur rédaction ces différentes fonctions sur des matrices simples (2*2) pour lesquelles vous aurez préalablement effectué les différents calculs à la main.*

2.1 Copie de matrices

Écrire une fonction **copie_matrice** qui permet de réaliser la copie d'une matrice afin que nos programmes ne modifient pas les matrices initiales. Cette fonction a pour entrée une matrice et pour sortie la copie à l'identique de cette matrice.

2.2 Recherche du pivot

Écrire une fonction **chercher_pivot** qui prend comme argument la matrice A et l'indice i, indice de la ligne à partir de laquelle on cherche le coefficient maximal qui servira de pivot pour l'étape.

Cette fonction retourne l'indice k de la ligne de ce coefficient maximal en valeur absolue

2.3 Échange de lignes

Écrire une fonction **echange_lignes** qui permet d'échanger deux lignes de la matrice A ou Y afin de placer la ligne k de coefficient maximal en valeur absolue en position i.

Elle aura pour argument les deux indices i et k de position des lignes à échanger, une matrice A et son nombre de colonnes nc.

Cette fonction ne retourne pas d'objets mais modifie la matrice d'entrée.

2.4 Transvection

Écrire une fonction **transvection_ligne** qui permet de modifier, à la i-ème étape la ligne m avec $i+1 \leqslant m \leqslant n-1$ et de réaliser $L_m \leftarrow L_m + \mu L_i$

Elle prendra comme arguments une matrice A à modifier ; nc son nombre de colonnes ; les indices i et m des lignes et le coefficient mu.

Cette fonction ne retourne pas d'objets mais modifie la matrice d'entrée.

3 Écriture du programme principal

Écrire une fonction **resolution** qui prenne en entrée les deux matrices A0 et Y0 du système à résoudre $A_0X = Y_0$ et retourne une matrice colonne X solution de ce système.

Ce programme :

- copie les matrices initiales
- triangularise la matrice A copie de A0 (ne pas oublier que toutes les opérations d'échange de ligne et de transvections se font aussi sur la matrice Y)
- effectue la phase de remontée

4 Exploitation

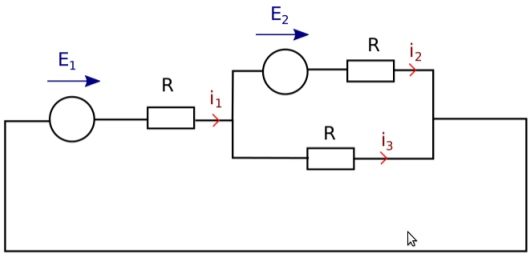
4.1 Cas d'école

Utiliser votre fonction et la fonction linalg.solve de la bibliothèque numpy pour résoudre le système suivant ; comparer les résultats en terme de précision et en terme de temps.

$$(S) : \begin{cases} 2x & + & 2y & - & 3z & = & 2 \\ -2x & - & y & - & 3z & = & -5 \\ 6x & + & 4y & + & 4z & = & 16 \end{cases}$$

4.2 Un exemple en Physique

À l'aide de votre programme principal résoudre le problème suivant. On se posera la question de la validation des résultats. Comparer le résultat obtenu avec la fonction linalg.solve de la bibliothèque numpy ; tant en terme de résultats qu'en termes de temps.



$$\begin{vmatrix} -E_1 + Ri_1 - E_2 + Ri_2 = 0 \\ -E_1 + Ri_1 + Ri_3 = 0 \\ i_1 = i_2 + i_3 \end{vmatrix} \qquad \begin{vmatrix} R = 10k\Omega \\ E_1 = 1V \\ E_2 = 0.5V \end{vmatrix}$$

Calculer i1, i2, i3

4.3 Matrices de Hilbert

Un test classique de performance numérique en calcul matriciel consiste à inverser les matrices de Hilbert. Comme on le verra en deuxième année, ces matrices sont très mal conditionnées, et tout calcul numérique portant sur de telles matrices peut induire d'assez grosses erreurs d'approximation.

La matrice de Hilbert de rang n est définie par :

$$H_n = \left(\frac{1}{i+j-1}\right)_{1\leqslant i,j\leqslant n}$$

On peut définir la matrice de Hilbert d'ordre n en Python en tenant compte du décalage d'indice :

```
def hilbert(n) :
    return [[1./(i+j+1) for j in range(n)] for i in range(n)]
```

On note C la matrice de $\mathcal{M}_{n,1}(\mathbb{R})$ avec des zéros partout sauf en dernière position. La résolution de $H_nX = C$ va renvoyer théoriquement la dernière colonne de H_n^{-1}. Un résultat classique est le caractère entier de H_n^{-1} .

Par exemple, la dernière composante de H_{10}^{-1} (calculée avec un logiciel de calcul formel) est 44914183600. Pour n=20 le résultat attendu est 48722219250572027160000

1. Implémenter la fonction qui renvoie une matrice de Hilbert d'ordre n et la fonction qui renvoie le vecteur colonne C tel que défini plus haut de rang n
2. Utiliser votre fonction pour tester les cas n=10 et n=20 tant en terme de précision qu'en terme de temps
3. Utiliser la fonction linalg.solve de numpy dans ces deux cas et comparer les précisions et les coûts en temps.