Résolution de systèmes linéaires
Pivot de Gauss

1 Principe

1.1 Exemple introductif

Résoudre le système d'équations linéaires suivant :

$$(S) \Longleftrightarrow \left\{ \begin{array}{rcrcrcl} 2x & + & 2y & - & 3z & = & 2 \\ -2x & - & y & - & 3z & = & -5 \\ 6x & + & 4y & + & 4z & = & 16 \end{array} \right.$$

1.2 Etapes de résolution

1.3 Ecriture matricielle

Associer au système (S) la matrice augmentée $(A|B)$.

2 Mise en oeuvre numérique

On considère que le système S est de rang plein.

2.1 Programme principal

Etapes :

1. Ecriture du problème sous forme matricielle
2. Mise en forme triangulaire
3. Phase de remontée

2.2 Transvection

La transevection correspond à l'opération élémentaire :

$$L_i \leftarrow L_i + \mu L_j$$

```python
def transvection(A,B,i,j,mu):
    A[i,:]=A[i,:]+mu*A[j,:]
    B[i]=B[i]+mu*B[j]
    return
```

```python
def transvection_detail(A,B,i,j,mu):
    n=len(A)
    for k in range(0,n):
        A[i,k]= A[i,k] + mu * A[j,k]
    B[i]=B[i]+mu*B[j]
    return
```

3 Mise en forme triangulaire

A est une matrice $n \times n$, B est une matrice colonne $n \times 1$.

```python
def triangulation (A,B):
    n=len(A)
    for i in range(n-1):
        pivot=(A[i][i])
        for j in range(i+1,n):
            transvection (A,B,j,i,-A[j,i]/pivot)
    return
```

4 Phase de remontée

On s'appuie sur la forme triangulaire :

$$(S) \iff \begin{cases} \alpha_{1,1}x_1 + \alpha_{1,2}x_2 + \dots + \alpha_{1,n}x_n = \beta_1 \\ \alpha_{2,2}x_2 + \dots + \alpha_{2,n}x_n = \beta_2 \\ \ddots \quad \vdots \\ \alpha_{n-1,n-1}x_{n-1} + \alpha_{n-1,n}x_n = \beta_{n-1} \\ \alpha_{n,n}x_n = \beta_n \end{cases}$$

x_i se détermine, pour i variant de n à 1 :

$$x_i = \left(\beta_i - \sum_{k=i+1}^{n} \alpha_{i,k}x_k\right) \frac{1}{\alpha_{i,i}}$$

```python
def remontee(A,B):
    n=len(A)
    X=[0.]*n
    for i in range(n-1,-1,-1):
        temp= B[i,0]
        for k in range (i+1,n):
            temp= temp - A[i,k] * X[k]
        X[i]=temp/A[i,i]
    return X
```

5 Considération de précision

5.1 Cas d'étude

On cherche à résoudre le système :

$$(S) \Longleftrightarrow \begin{cases} x_1 & + & 2x_2 & + & 3x_3 & = & 6 \\ 4x_1 & + & 5x_2 & + & 6x_3 & = & 15 \\ 7x_1 & + & 8x_3 & + & 8,999999 & = & 23,999999 \end{cases}$$

Solution évidente : $(x_1, x_2, x_3) = (1, 1, 1)$
Par l'algorithme du pivot de Gauss on trouve :

	32 bits	64 bits
solution calculée		
écart		

5.2 Amélioration

L'erreur provient en majeure partie de la division numérique : plus $a_{i,i}$ est petit, plus l'erreur sur $\frac{1}{a_{i,i}}$ est grande. Une amélioration du programme consiste à choisir comme pivot la plus grande valeur, en valeur absolue, de la colonne considérée.

En conséquence, la ligne où apparait cette valeur doit être échangée avec la $i$ème ligne.

Cela permet aussi d'éviter tout problème en cas de pivot nul.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 7 & 8 & 9 \end{bmatrix} \qquad A^{(2)} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & -1 \\ 0 & -6 & -12 \end{bmatrix}$$

5.3 Complément de programme

```python
def chercher_pivot(A,i):
    n = len(A)
    j=i
    for k in range(i+1,n):
        if abs(A[k][i])>abs(A[j][i]):
            j=k
    return j
```

```python
def ech_ligne(A,B,i,j):
    n=len(A)
    for index in range(n):
        temp=A[i][index]
        A[i][index]=A[j][index]
        A[j][index]=temp
    temp=B[i,0]
    B[i,0]=B[j,0]
    B[j,0]=temp
    return(A)
```

5.4 Nouvelles performances

	32 bits	64 bits
solution calculée		
écart		

5.5 Malgré tout : conditionnement de matrice

Certaines matrices ne respectent pas certaines conditions qui limitent les erreurs numériques. Elles sont appelées «mal conditionnées ».

Moralité : vérifier le résultat !

6 Complexité

Fonctions	Notation	Nombre d'opération
Recherche pivot	C_{rp}	
Echange ligne	C_{el}	
Transvection	C_{trans}	
Triangulation	C_{tri}	
Remontéé	C_{rm}	

7 Programme complet

```python
import numpy as np

def transvection(A,B,i,j,mu):
    A[i,:]=A[i,:]+mu*A[j,:]
    B[i]=B[i]+mu*B[j]
    return

def echange_lignes(A,B,i,j):
    n=len(A)
    for index in range(n):
        temp=A[i][index]
        A[i][index]=A[j][index]
        A[j][index]=temp
    temp=B[i,0]
    B[i,0]=B[j,0]
    B[j,0]=temp
    return

def chercher_pivot(A,i):
    n = len(A)
    j=i
    for k in range(i+1,n):
        if abs(A[k][i])>abs(A[j][i]):
            j=k
    return j

def triangulation(A,B):
    n=len(A)
    for i in range(n-1):
        index_pivot=chercher_pivot(A,i)
        echange_lignes(A,B,i,index_pivot)
        pivot=(A[i][i])
        for j in range(i+1,n):
            transvection(A,B,j,i,-A[j,i]/pivot)
    return

def remontee(A,B):
    n=len(A)
    X=[0.]*n
    for i in range(n-1,-1,-1):
        temp= B[i,0]
        for k in range(i+1,n):
            temp= temp - A[i,k] * X[k]
        X[i]=temp/A[i,i]
    return X
A=np.array([[2,2,-3],[-2,-1,-3],[6,4,4],],'float64')
B=np.array([[2],[-5],[16]],'float64')
triangulation(A,B)
print((remontee(A,B)))
```