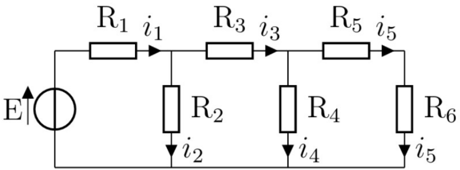
Résolution directe d'un système d'équations linéaires par pivot de Gauss

1

Principe de la méthode

Exemple : Déterminer les courants dans le circuit.



$E = 5.0\,\mathrm{V}$, $R_1 = 100\Omega$,
$R_2 = R_3 = 220\Omega$,
$R_4 = R_5 = R_6 = 100\Omega$

Les lois de Kirchhoff donnent :

$$\begin{cases} E = R_1 i_1 + R_2 i_2 \\ R_2 i_2 = R_3 i_3 + R_4 i_4 \\ R_4 i_4 = R_5 i_5 + R_6 i_5 \\ i_1 = i_2 + i_3 \\ i_3 = i_4 + i_5 \end{cases}$$

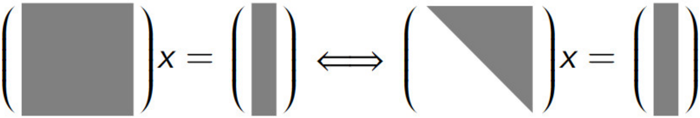
Déterminer tous les courants dans ce circuit, revient à résoudre l'équation matricielle :

$$\begin{pmatrix} R_1 & R_2 & 0 & 0 & 0 \\ 0 & -R_2 & R_3 & R_4 & 0 \\ 0 & 0 & 0 & -R_4 & R_5 + R_6 \\ -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \end{pmatrix} = \begin{pmatrix} E \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

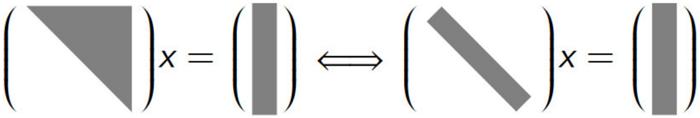
Principe de la méthode du pivot de Gauss

C'est une méthode de résolution d'un système linéaire :AX=B, où A est une matrice inversible : on ne modifie pas l'ensemble des solutions d'une équation linéaire en appliquant les mêmes opérations élémentaires sur les lignes de A et de B. La méthode du pivot de Gauss comporte trois étapes :

1. une première étape de descente : on transforme la matrice A en une matrice A' triangulaire tout en effectuant les mêmes opérations sur B,



2. une deuxième étape de remontée : on transforme la matrice A' en une matrice A" diagonale tout en effectuant les mêmes opérations sur B,



3. une dernière étape de résolution du système linéaire diagonal obtenu.

2

Obtention d'une matrice triangulaire

Exemple de système 2x2

$$\begin{cases} 0,003x + 59,14y = 59,17 \\ 5,291x - 6,13y = 46,78 \end{cases}$$

On suppose les calculs effectués en virgule flottante dans le système décimal avec une mantisse de quatre chiffres.

Ce système est équivalent à l'équation matricielle :

$$\begin{pmatrix} 3.000 \times 10^{-3} & 5.914 \times 10^{1} \\ 5.291 \times 10^{0} & -6.130 \times 10^{0} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5.917 \times 10^{1} \\ 4.678 \times 10^{1} \end{pmatrix}$$

La solution exacte de ce système est x=10 et y=1.

Si on choisit 0,003 pour pivot...

Si on choisit 5,291 pour pivot...

Méthode du « pivot partiel »

Si on choisit un pivot trop . . . . , des erreurs d'arrondis importantes peuvent apparaître au cours de la résolution et aboutir à une valeur qui est très éloignée de la valeur théorique.

$$A = \begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ a_{10} & \ddots & \cdots & a_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{pmatrix}$$

Le choix du « pivot partiel » pour une colonne j, consiste à choisir pour pivot le coefficient le plus grand en valeur absolue parmi les $\{a_{jj}, a_{j+1,j}, \ldots, a_{n-1,j}\}$ de la colonne j.

Rechercher le pivot partiel


```
def ChercherPivot(M,j):
    """Cherche le pivot le plus grand possible
     en valeur absolue pour la colonne j
    (sans toucher aux lignes deja traitees)"""

   n=len(M) #nombre de lignes
    imax=j
    for i in range(j+1,n):
        if abs(M[i][j])>abs(M[imax][j]):
            imax=i
    return imax
```

Procédures d'échange et de transvection


```python
def echange(M, i1, i2):
    """Echange la ligne i1 avec la ligne i2"""
    M[i1][:],M[i2][:]=M[i2][:],M[i1][:]

def transvection(M, i1, i2, mu):
    """Effectue la transvection
    Li1=Li1-mu*L2"""
    p=len(M[0])#nombre de colonnes
    for j in range(p):
        M[i1][j]=M[i1][j]-mu*M[i2][j]
```

Attentions aux manipulations des listes de listes

Attention aux alias... tester l'algorithme de copie si possible

```
def copieM(M):
    """Effectue une matrice"""
    n=len(M)
    copie=[None]*n
    for i in range(n):
        copie[i]=M[i][:]
    return copie
```

Autre possibilité :

```
from copy import deepcopy
Ac=deepcopy(A)
```

Obtenir une matrice triangulaire


```python
def triangulaire(A,B):
    """Applique la methode du pivot partiel
     au systeme AX=B pour obtenir
     une matrice triangulaire """

    Ac,Bc=copieM(A),copieM(B)
    n=len(A)
    for j in range(n-1):
        i=ChercherPivot(Ac,j)
        echange(Ac,i,j)
        echange(Bc,i,j)
        for isup in range(j+1,n):
            mu=Ac[isup][j]/Ac[j][j]
            transvection(Ac,isup,j,mu)
            transvection(Bc,isup,j,mu)
    return Ac,Bc
```

3

Phase de remontée

Après avoir réalisé l'ensemble des opérations précédentes sur le système AX=B, nous avons abouti au système triangulaire qui peut se mettre sous la forme :

$$\begin{pmatrix} \alpha_{00} & \dots & \dots & \alpha_{0,n-1} \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \alpha_{n-1,n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ \vdots \\ \vdots \\ x_{n-1} \end{pmatrix} = \begin{pmatrix} \beta_0 \\ \vdots \\ \vdots \\ \beta_{n-1} \end{pmatrix}$$

Les différentes composantes x_i du vecteur X s'obtiennent successivement en calculant d'abord x_{n-1} puis en remontant les indices jusqu'à x_0. Les différentes composantes x_i du vecteur X s'obtiennent avec la relation :

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

```
def ResolGauss(A,B):
    n=len(A)
    alpha,beta=triangulaire(A,B)
    X=[None]*n
    for i in range(n-1,-1,-1):
        X[i]=(beta[i][0]- \n
        sum(alpha[i][k]*X[k] for k in \n
        range(i+1,n)))/alpha[i][i]
    return X
```

- Quelle est la complexité de l'algorithme mis en oeuvre dans la focntion ResolGauss ?

4

Exemples

Exemples

- Retour sur le système 2×2
- Déterminer les courants du système linéaire introductif

- Utilisation de la fonction **linalg.solve**(A,B)... la puissance de numpy !