Résolution des systèmes linéaires

Les systèmes triangulaires

Approche matricielle

Les $a_{i,j}$ et les b_i sont des données et on cherche les x_i tels que

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n &= b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n &= b_2 \\ \vdots & \vdots \quad \vdots \\ a_n x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n &= b_n \\ . \end{cases}$$

C'est un système linéaire qui peut s'écrire :

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

Cas d'une matrice triangulaire

Le système est de la forme :

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_1 + \cdots + a_{1,n-1}x_{n-1} + a_{1,n}x_n &= b_1 \\ a_{2,2}x_2 + \cdots + a_{2,n-1}x_{n-1} + a_{2,n}x_n &= b_2 \\ \vdots \quad \vdots & \quad \vdots \quad \vdots \\ a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n &= b_{n-1} \\ a_{n,n}x_n &= b_n. \end{cases}$$

On suppose que $\forall\ 1 \leq i \leq n, a_{i,i} \neq 0$.

Les x_i se calculent explicitement en commençant par x_n et en « remontant » jusqu'à x_1.

Le code C

$$\begin{cases} a_{1,1}x_1 + a_{1,2}x_1 + \cdots + a_{1,n-1}x_{n-1} + a_{1,n}x_n &= b_1 \\ a_{2,2}x_2 + \cdots + a_{2,n-1}x_{n-1} + a_{2,n}x_n &= b_2 \\ \vdots \quad \vdots \quad \vdots & \\ a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n &= b_{n-1} \\ a_{n,n}x_n &= b_n \end{cases}$$

```
void ressystr(float *a, float *b, float *x, int n)
{
    int i,k;
    float aux;
    *(x + n - 1) = *(b + n - 1) / *(a + n*n - 1);
    for(i = n-2; i >= 0; i--)
    {
        aux = *(b + i);
        for(k = i+1; k < n; k++)
            aux -= *(a + i*n + k) * *(x + k);
        *(x + i) = aux / *(a + i*n + i);
    }
}
```

Exemple

$$\begin{pmatrix} 5 & 7 & 4 & 3 \\ 0 & 5 & 13 & 5 \\ 0 & 0 & 15 & 13 \\ 0 & 0 & 0 & 6 \end{pmatrix} \cdot X = \begin{pmatrix} -1 \\ 3 \\ 2 \\ -6 \end{pmatrix}$$

Calcul de la complexité

Le calcul de x_i est donné par

$$x_i = (b_i - a_{i,i+1}x_{i+1} - \ldots - a_{i,n}x_n)/a_{i,i}$$

De k à n, il y a $n+1-k$ termes.

Donc de $i+1$ à n, il y a $(n+1-(i+1) = n-i$ indices

donc le calcul de x_i nécessite $2 \times (n-i)+1$ opérations.

Le calcul total nécessite

$$\sum_{i=1}^{n}(2 \times (n-i)+1) = \sum_{k=0}^{n-1}(2 \times k+1) = 2 \times \frac{n(n-1)}{2} + n \approx n^2$$

Fin

L'algorithme de Gauß

Méthode de Gauß

La méthode Gauß se décompose en deux phases.

La première phase consiste à transformer un système linéaire général en un système triangulaire supérieur.

On appelle cette phase *la descente de Gauß*.

La seconde phase consiste à résoudre le système triangulaire supérieur.

On appelle cette phase *la remontée de Gauß*.

Le principe de la descente de Gauß

Soit le système $A \cdot X = B$. On crée la matrice $\left(\begin{array}{c|c} a_{i,j} & b_i \end{array} \right)$.

À l'étape i, on « réduit » la colonne i : on met un 1 sur la diagonale et des 0 dans la partie inférieure de la colonne i.

On part de

$$\left(\begin{array}{ccccc|c} 1 & \cdots & a'_{1,i} & \cdots & a'_{1,n} & b'_1 \\ \vdots & & \vdots & & \vdots & \vdots \\ 0 & & \boxed{a'_{i,i}} & & a'_{i,n} & b'_i \\ \vdots & \cdots & \boxed{\vdots} & \cdots & \vdots & \vdots \\ 0 & \cdots & \boxed{a'_{j,i}} & \cdots & a'_{j,n} & b'_j \\ \vdots & & \boxed{\vdots} & & \vdots & \vdots \\ 0 & \cdots & \boxed{a'_{n,i}} & \cdots & a'_{n,n} & b'_n \end{array} \right)$$

puis on divise la ligne i, b compris, par $a'_{i,i}$ qu'on appelle le i-ème pivot.

Ensuite, pour j allant de $i+1$ à n, on multiplie la nouvelle ligne i par $a'_{j,i}$ et on la soustrait à la ligne j.

Un exemple

On veut résoudre $\begin{pmatrix} -6 & 1 & -7 \\ 9 & 3 & 5 \\ 2 & 0 & 2 \end{pmatrix} \cdot X = \begin{pmatrix} -14 \\ 11 \\ 4 \end{pmatrix}$

Première étape :

Un exemple

On veut résoudre $\begin{pmatrix} -6 & 1 & -7 \\ 9 & 3 & 5 \\ 2 & 0 & 2 \end{pmatrix} \cdot X = \begin{pmatrix} -14 \\ 11 \\ 4 \end{pmatrix}$

Première étape : $\begin{pmatrix} 1 & -1/6 & 7/6 & 7/3 \\ 0 & 9/2 & -11/2 & -10 \\ 0 & 1/3 & -1/3 & -2/3 \end{pmatrix}$.

Deuxième étape :

Un exemple

On veut résoudre $\begin{pmatrix} -6 & 1 & -7 \\ 9 & 3 & 5 \\ 2 & 0 & 2 \end{pmatrix} \cdot X = \begin{pmatrix} -14 \\ 11 \\ 4 \end{pmatrix}$

Première étape : $\begin{pmatrix} 1 & -1/6 & 7/6 & 7/3 \\ 0 & 9/2 & -11/2 & -10 \\ 0 & 1/3 & -1/3 & -2/3 \end{pmatrix}$.

Deuxième étape : $\begin{pmatrix} 1 & -1/6 & 7/6 & 7/3 \\ 0 & 1 & -11/9 & -20/9 \\ 0 & 0 & 2/27 & 2/27 \end{pmatrix}$

La remontée

Un exemple

On veut résoudre $\begin{pmatrix} -6 & 1 & -7 \\ 9 & 3 & 5 \\ 2 & 0 & 2 \end{pmatrix} \cdot X = \begin{pmatrix} -14 \\ 11 \\ 4 \end{pmatrix}$

Première étape : $\begin{pmatrix} 1 & -1/6 & 7/6 & 7/3 \\ 0 & 9/2 & -11/2 & -10 \\ 0 & 1/3 & -1/3 & -2/3 \end{pmatrix}$.

Deuxième étape : $\begin{pmatrix} 1 & -1/6 & 7/6 & 7/3 \\ 0 & 1 & -11/9 & -20/9 \\ 0 & 0 & 2/27 & 2/27 \end{pmatrix}$

La remontée donne $X_s = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}$

Transformation de type 2

On multiplie la ligne k de A par λ.

Cela revient à multiplier A à gauche par

$$P^2_{k,\lambda} = \begin{array}{c} \\ \vdots \\ k \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccc} & & k & & \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ \dots & 0 & \lambda & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 \end{pmatrix} \end{array} .$$

Le déterminant de $P^2_{k,\lambda}$ vaut λ et son inverse est $P^2_{k,1/\lambda}$, c'est-à-dire la même matrice mais en changeant λ par $1/\lambda$.

Transformation de type 3

On soustrait la ligne ℓ multipliée par λ à la ligne k de A.

Cela revient à multiplier A à gauche par

$$P^3_{k,\ell,\lambda} = \begin{array}{c} \\ \vdots \\ k \\ \vdots \\ \ell \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} \ldots & & k & \ldots & \ell & \ldots & \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 1 & \ldots & -\lambda & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 0 & 1 \end{pmatrix} \end{array}$$

Le déterminant de $P^3_{k,\ell,\lambda}$ vaut 1 et son inverse est $P^3_{k,\ell,-\lambda}$, c'es-à-dire la même matrice mais en changeant λ par $-\lambda$.

Théorème : Si les pivots successifs sont non nuls, la descente de Gauß transforme un système linéaire général en un système triangulaire supérieur équivalent avec des 1 sur la diagonale.

En effet,

$$A \cdot X = B$$
$$\Leftrightarrow$$
$$P_1 \cdot P_2 \cdots P_k \cdot A \cdot X = P_1 \cdot P_2 \cdots P_k \cdot B$$

où P_i est une matrice élémentaire de type 2 ou 3.

Algorithme de Gauß en C

```
void gauss(float *a, float *b, int n)
{
  int i,j,k;
  float aux, aux2;
  for(i=0;i<n-1;i++)
  {
    aux = *(a + i*n + i);
    for(k=i+1; k<n; k++)
      *(a + i*n + k) /= aux;
    *(b + i) /= aux;
    for(k=i+1;k<n;k++)
    {
      aux2 = *(a + k*n + i);
      for(j=i+1;j<n;j++)
        *(a + k*n + j) -= aux2 * *(a + i*n + j);
      *(b + k) -= aux2 * *(b+i);
    }
  }
  *(b + n-1) /= *(a + n*n -1);
  for(i=n-2;i>=0;i--)
  {
    aux= *(b + i);
    for(k=i+1;k<n;k++)
      aux -= *(a + i*n + k) * *(b + k);
    *(b + i) = aux;
  }
}
```

Complexité

La complexité de l'algorithme de Gauß est en $\frac{2}{3}n^3$.

En effet, à l'étape i

- $n+1-i$ divisions ;
- pour les substitutions : $2(n-i)(n+1-i)$ opérations .

Soit au total

$$\sum_{i=1}^{n}(2(n-i)+1)(n+1-i) = \sum_{k=1}^{n}(2k-1)k$$
$$= 2 \times \frac{n(n+1)(2n+1)}{6} - n(n+1)$$
$$\approx \frac{2}{3}n^3$$

Fin

L'algorithme de Gauß
avec recherche partiel de pivot maximum

La recherche partiel de pivot maximum

L'algorithme de Gauß n'est applicable que si les pivots sont non nuls.

$\Rightarrow$ Algorithme de Gauß avec recherche du pivot maximum.

- A l'étape i, on cherche i_0 tel que

$$|a_{i_0,i}| = \max_{k=i}^{n} (|a_{k,i}|)$$

et on echange les lignes i et i_0.

$$\left(\begin{array}{ccccccc|c} 1 & & \dots & & a'_{1,i} & \dots & a'_{1,n} & b'_1 \\ \vdots & & & & \vdots & & \vdots & \vdots \\ 0 & & & & a'_{i,i} & & a'_{i,n} & b'_i \\ \vdots & & \dots & & \vdots & \dots & \vdots & \vdots \\ 0 & & \dots & & a'_{j,i} & \dots & a'_{j,n} & b'_j \\ \vdots & & \dots & & \vdots & \dots & \vdots & \vdots \\ 0 & & & & a'_{n,i} & & a'_{n,n} & b'_n \end{array}\right)$$

Transformation de type 1

On échange les lignes k et ℓ de la matrice $A = (a_{i,j}), \forall A$.

Cela revient à multiplier A à gauche par

$$P^1_{k,\ell} = \begin{array}{c} \\ \vdots \\ k \\ \vdots \\ l \\ \vdots \end{array} \begin{array}{c} \begin{array}{ccccccc} \dots & k & \dots & \ell & \dots & & \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} \end{array}.$$

$P^1_{k,\ell}$ est obtenue à partir de l'identité en échangeant les lignes k et ℓ.

Le déterminant de $P^1_{k,\ell}$ vaut -1 et $P^1_{k,\ell} \times P^1_{k,\ell} = I_d$.

Théorème : La descente de Gauß avec recherche partiel de pivot maximum transforme un système linéaire général en un système triangulaire supérieur équivalent avec des 1 sur la diagonale si et seulement si la matrice du système est inversible

En effet, si Gauß s'applique

$$A \cdot X = B$$
$$\Leftrightarrow$$
$$P_1 \cdot P_2 \cdots P_k \cdot A \cdot X = P_1 \cdot P_2 \cdots P_k \cdot B.$$

où P_i est une matrice élémentaire de type 2 ou 3.

Et $\det(P_1 \cdot P_2 \cdots P_k \cdot A) = \prod \det(P_i) \times \det(A) = 1 \Rightarrow \det(A) \neq 0.$

Réciproquement, si Gauß ne s'applique pas, c'est qu'à une étape i on tombe sur

$$\left(\begin{array}{ccccc|c} 1 & \dots & a'_{1,i} & \dots & a'_{1,n} & b'_1 \\ \vdots & & \vdots & & \vdots & \vdots \\ 0 & & 0 & & a'_{i,n} & b'_i \\ \vdots & \dots & \vdots & \dots & \vdots & \vdots \\ 0 & \dots & 0 & \dots & a'_{j,n} & b'_j \\ \vdots & \dots & \vdots & \dots & \vdots & \vdots \\ 0 & & 0 & & a'_{n,n} & b'_n \end{array}\right).$$

Les i premières colonnes de cette matrice sont liées car seules leurs $i-1$ premières coordonnées sont non nuls donc elles sont dans un espace de dimension au plus $i-1$ donc la matrice $A' = (a'_{i,j})$ est singulière.

Or $P_1 \cdot P_2 \cdots P_s \cdot A = A'$ et $\forall i, \det(P_i) \neq 0$ donc $\det(A) = 0$.

Algorithme de Gauß en C

```
void gauss(float *a, float *b, int n)
{
  int i,j,k,il;
  float aux, aux2;
  for(i=0;i<n-1;i++)
    {
      aux = *(a + i*n + i);
      for(k=i+1; k<n; k++)
        *(a + i*n + k) /= aux;
      *(b + i) /= aux;
      for(k=i+1;k<n;k++)
        {
          aux2 = *(a + k*n + i);
          for(j=i+1;j<n;j++)
            *(a + k*n + j) -= aux2 * *(a + i*n + j);
          *(b + k) -= aux2 * *(b+i);
        }
    }
  *(b + n-1) /= *(a + n*n -1);
  for(i=n-2;i>=0;i--)
    {
      aux= *(b + i);
      for(k=i+1;k<n;k++)
        aux -= *(a + i*n + k) * *(b + k);
      *(b + i) = aux;
    }
}
```

La recherche partielle de pivot en C

```
for(i=0;i<n-1;i++)
  {
    aux=fabs(*(a + i*n + i));
    il=i;
    for(k = i+1; k < n; k++)
      if (aux<fabs(*(a + k*n + i)))
        {
          aux = fabs(*(a + k*n + i));
          il = k;
        };
    if (il != i)
      {
        for(j=i;j<n;j++)
          {
            aux = *(a + il*n + j);
            *(a + il*n + j) = *(a + i*n + j);
            *(a + i*n + j) = aux;
          }
        aux = *(b + il);
        *(b + il) = *(b + i);
        *(b + i) = aux;
      }
    aux = *(a + i*n + i);
.....................
```

La recherche partielle de pivot minimise les erreurs d'arrondi ! !

Soit $A = \begin{pmatrix} \varepsilon & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$.

Après la première étape, on obtient :

$$A_1 = \begin{pmatrix} 1 & \frac{a_{1,2}}{\varepsilon} & \frac{a_{1,3}}{\varepsilon} \\ 0 & a_{2,2} - \frac{a_{2,1}a_{1,2}}{\varepsilon} & a_{2,3} - \frac{a_{2,1}a_{1,3}}{\varepsilon} \\ 0 & a_{3,2} - \frac{a_{3,1}a_{1,2}}{\varepsilon} & a_{3,3} - \frac{a_{3,1}a_{1,3}}{\varepsilon} \end{pmatrix}.$$

Si $\varepsilon << 1$ alors

$$\begin{pmatrix} a_{2,2} - \frac{a_{2,1}a_{1,2}}{\varepsilon} & a_{2,3} - \frac{a_{2,1}a_{1,3}}{\varepsilon} \\ a_{3,2} - \frac{a_{3,1}a_{1,2}}{\varepsilon} & a_{3,3} - \frac{a_{3,1}a_{1,3}}{\varepsilon} \end{pmatrix} \approx \begin{pmatrix} -\frac{a_{2,1}a_{1,2}}{\varepsilon} & -\frac{a_{2,1}a_{1,3}}{\varepsilon} \\ -\frac{a_{3,1}a_{1,2}}{\varepsilon} & -\frac{a_{3,1}a_{1,3}}{\varepsilon} \end{pmatrix}$$

Et la réduction suivante engendre des soustractions catastrophiques

Exemple

$$\begin{pmatrix} 21 & 130 & 0 & 2.1 \\ 13 & 80 & 4.74 & 752 \\ 0 & -0.4 & 3.9816 & 4.2 \\ 0 & 0 & 1.7 & 9.0e-1 \end{pmatrix} \cdot X = \begin{pmatrix} 153.1 \\ 849.74 \\ 7.7816 \\ 2.6 \end{pmatrix}$$

Fin