L'inversion matricielle

Un cas particulier

Si $B = I_d$ alors la résolution de $A.X = B$ conduit à calculer A^{-1}.
Exemple :

$$\left(\begin{array}{ccc|ccc} 2 & 3 & -1 & 1 & 0 & 0 \\ 4 & 4 & -3 & 0 & 1 & 0 \\ -2 & 3 & -1 & 0 & 0 & 1 \end{array}\right)$$

Un cas particulier

Exemple :

$$\left(\begin{array}{ccc|ccc} 2 & 3 & -1 & 1 & 0 & 0 \\ 4 & 4 & -3 & 0 & 1 & 0 \\ -2 & 3 & -1 & 0 & 0 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{ccc|ccc} 1 & 3/2 & -1/2 & 1/2 & 0 & 0 \\ 0 & -2 & -1 & -2 & 1 & 0 \\ 0 & 6 & -2 & 1 & 0 & 1 \end{array}\right)$$

Un cas particulier

Exemple :

$$\left(\begin{array}{ccc|ccc} 2 & 3 & -1 & 1 & 0 & 0 \\ 4 & 4 & -3 & 0 & 1 & 0 \\ -2 & 3 & -1 & 0 & 0 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{ccc|ccc} 1 & 3/2 & -1/2 & 1/2 & 0 & 0 \\ 0 & -2 & -1 & -2 & 1 & 0 \\ 0 & 6 & -2 & 1 & 0 & 1 \end{array}\right)$$

$$\left(\begin{array}{ccc|ccc} 1 & 0 & -5/4 & -1 & 3/4 & 0 \\ 0 & 1 & 1/2 & 1 & -1/2 & 0 \\ 0 & 0 & -5 & -5 & 3 & 1 \end{array}\right)$$

Un cas particulier

Exemple :

$$\left(\begin{array}{ccc|ccc} 2 & 3 & -1 & 1 & 0 & 0 \\ 4 & 4 & -3 & 0 & 1 & 0 \\ -2 & 3 & -1 & 0 & 0 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{ccc|ccc} 1 & 3/2 & -1/2 & 1/2 & 0 & 0 \\ 0 & -2 & -1 & -2 & 1 & 0 \\ 0 & 6 & -2 & 1 & 0 & 1 \end{array}\right)$$

$$\left(\begin{array}{ccc|ccc} 1 & 0 & -5/4 & -1 & 3/4 & 0 \\ 0 & 1 & 1/2 & 1 & -1/2 & 0 \\ 0 & 0 & -5 & -5 & 3 & 1 \end{array}\right) \Rightarrow \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 1/4 & 0 & -1/4 \\ 0 & 1 & 0 & 1/2 & -1/5 & 1/10 \\ 0 & 0 & 1 & 1 & -3/5 & -1/5 \end{array}\right)$$

Remarque : On peut donc inverser « sur place ».

Le code C

```
double inv_gj(double *a, int n)
{
  double aux;
  int i,j,k;
  for(i=0;i<n;i++)
    {
      aux = *(a + n*i +i);
      for(j=0;j<n;j++)
        *(a + n*i + j) /= aux;
      *(a + n*i +i) = 1.0/aux;
      for(k=0;k<n;k++)
        if (k!=i)
          {
            aux = *(a +n*k + i);
            for(j=0;j<n;j++)
              *(a +n*k + j) -= aux * *(a +n*i + j);
            *(a +n*k + i) = - aux* *(a +n*i + i);
          }
    }
}
```

Complexité de l'algorithme

A chaque étape : n divisions et $(n-1)^2$ substitutions (2 opérations)

Au total :

$$n \times (n + 2(n-1)^2 \approx 2 \times n^3 \text{ opérations}$$

C'est comparable à un produit matricielle.

Exemple

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 3 & -3 \\ 4 & -2 & 2 \end{pmatrix}$$

Exemple

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 3 & -3 \\ 4 & -2 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 2 \\ 0 & 3 & -3 \\ -4 & -2 & -6 \end{pmatrix}$$

Exemple

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 3 & -3 \\ 4 & -2 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 2 \\ 0 & 3 & -3 \\ -4 & -2 & -6 \end{pmatrix}$$
$$\rightarrow \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1/3 & -1 \\ -4 & 2/3 & -8 \end{pmatrix}$$

Exemple

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 3 & -3 \\ 4 & -2 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 2 \\ 0 & 3 & -3 \\ -4 & -2 & -6 \end{pmatrix}$$

$$\rightarrow \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1/3 & -1 \\ -4 & 2/3 & -8 \end{pmatrix} \rightarrow \begin{pmatrix} 0 & 1/6 & 1/4 \\ 1/2 & 1/4 & -1/8 \\ 1/2 & -1/12 & -1/8 \end{pmatrix}$$

On a bien $\begin{pmatrix} 1 & 0 & 2 \\ 0 & 3 & -3 \\ 4 & -2 & 2 \end{pmatrix} \times \begin{pmatrix} 0 & 1/6 & 1/4 \\ 1/2 & 1/4 & -1/8 \\ 1/2 & -1/12 & -1/8 \end{pmatrix} = I_d$

Fin

La décomposition LU

Suppons que, dans un algorithme itératif, il faut résoudre le système linéaire $A.X = B_k$ à chaque itération.

On aimerait ne pas avoir à appliquer un Gauss complet à chaque itération.

La solution est la décomposition LU.

Appliquons la descente de Gauss sans recherche de pivot comme le ferait l'ordinateur.

Soit la matrice $A = \begin{pmatrix} 2 & 3 & -1 \\ 4 & 4 & -3 \\ -2 & 3 & -1 \end{pmatrix}$.

$$\Rightarrow \begin{pmatrix} 2 & 3/2 & -1/2 \\ 4 & -2 & -1 \\ -2 & 6 & -2 \end{pmatrix} \Rightarrow \begin{pmatrix} 2 & 3/2 & -1/2 \\ 4 & -2 & 1/2 \\ -2 & 6 & -5 \end{pmatrix}$$

Soit $L = \begin{pmatrix} 2 & 0 & 0 \\ 4 & -2 & 0 \\ -2 & 6 & -5 \end{pmatrix}$ et $U = \begin{pmatrix} 1 & 3/2 & -1/2 \\ 0 & 1 & 1/2 \\ 0 & 0 & 1 \end{pmatrix}$

Alors

$$L.U = A$$

Avec Gauss, $P_l ... P_1 . A = U$, donc $A = P_1^{-1} ... P_l^{-1} . U$

$P_1^{-1} ... P_l^{-1}$ est exactement ce que contient la partie inférieure de la matrice à la fin de l'algorithme.

La première transformations est la division par le premier pivot $a_{1,1}$ donc

$$Q_{\mathbf{1}} = \begin{pmatrix} 1/a_{\mathbf{1},\mathbf{1}} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 1 \end{pmatrix} \text{ et } Q_{\mathbf{1}}^{-\mathbf{1}} = \begin{pmatrix} a_{\mathbf{1},\mathbf{1}} & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 1 \end{pmatrix}$$

La première substitution correspond à la multiplication à gauche par

$$P_{\mathbf{2},\mathbf{1}} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ -a_{\mathbf{2},\mathbf{1}} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 1 \end{pmatrix} \text{ et } P_{\mathbf{2},\mathbf{1}}^{-\mathbf{1}} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ a_{\mathbf{2},\mathbf{1}} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} a_{1,1} & \dots \\ \dots & \dots \\ \dots & \dots \\ \dots & \dots \end{pmatrix} \times \begin{pmatrix} 1 & \dots \\ a_{2,1} & \dots \\ \dots & \dots \\ \dots & \dots \end{pmatrix} \times \dots \times \begin{pmatrix} 1 & \dots \\ \dots & \dots \\ \dots & \dots \\ a_{n,1} & \dots \end{pmatrix} = \begin{pmatrix} a_{1,1} & \dots \\ a_{2,1} & \dots \\ \dots & \dots \\ a_{n,1} & \dots \end{pmatrix}$$

$$\begin{pmatrix} a_{\mathbf{1},\mathbf{1}} & \dots \\ a_{\mathbf{2},\mathbf{1}} & \dots \\ \dots & \dots \\ a_{n,\mathbf{1}} & \dots \end{pmatrix} \times \begin{pmatrix} 1 & .. & .. \\ 0 & a^{(\mathbf{1})}_{\mathbf{2},\mathbf{2}} & .. \\ .. & .. & .. \\ 0 & .. & .. \end{pmatrix} \times \dots \times \begin{pmatrix} 1 & .. & .. \\ 0 & 1 & \dots \\ .. & .. & .. \\ 0 & a^{(\mathbf{1})}_{n,\mathbf{2}} & .. \end{pmatrix} = \begin{pmatrix} a_{\mathbf{1},\mathbf{1}} & 0 & .. \\ a_{\mathbf{2},\mathbf{1}} & a^{(\mathbf{1})}_{\mathbf{2},\mathbf{2}} & .. \\ .. & .. & .. \\ a_{n,\mathbf{1}} & a^{(\mathbf{1})}_{n,\mathbf{2}} & .. \end{pmatrix}$$

Au final

$$P_1^{-1} \dots P_l^{-1} = \begin{pmatrix} a_{1,1} & 0 & .. & .. & .. & 0 \\ a_{2,1} & a^{(1)}_{2,2} & 0 & .. & .. & 0 \\ a_{3,1} & a^{(1)}_{3,2} & a^{(2)}_{3,3} & 0 & .. & 0 \\ .. & .. & .. & .. & .. & .. \\ a_{n,1} & a^{(1)}_{n,2} & a^{(2)}_{n,3} & .. & .. & a^{(n-2)}_{n,n} \end{pmatrix}$$

Le code C

```
void decomp-LU(float *a, 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;
      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);
        }
    }
}
```

Fin