第3章

线性方程组并行计算

本章主要介绍两种求解大型线性方程组的方法，第一种是乔勒斯基(Cholesky)分解法，另外一种是共轭梯度法(Conjugate Gradient Method)。本章对这两种方法进行讲述，并给出了程序实例予以讲解。

3.1 乔勒斯基分解法

3.1.1 乔勒斯基分解的分析

许多科学与工程的计算问题可归结为对线性方程组 $KU=I$ 的求解。大部分计算量也是花在解线性方程组上。对于系数矩阵 K 是大型、对称和正定的系数矩阵，最常用的解方程组的直接法是乔勒斯基分解法。故一般采用乔勒斯基分解的变形(LDL^T)分解法解线性方程组。

$$[K]=[L]\cdot[D]\cdot[L]^T \tag{3.1}$$

$[L]$为单位下三角矩阵，$[D]$为对角矩阵，各元素可按下式递推计算

$$\left.\begin{aligned}&\left.\begin{aligned}\tilde{a}_{ij}&=K_{ij}-\sum_{p=1}^{j-1}\tilde{a}_{ip}L_{jp}\\L_{ij}&=\tilde{a}_{ij}/ds\end{aligned}\right\}j<i\\&d_i=K_{ii}-\sum_{p=1}^{j-1}\tilde{a}_{ip}L_{jp}\quad j=i\end{aligned}\right\}$$

$$(i=1,2,\cdots,N) \tag{3.2}$$

在对$[K]$作了 LDL^T 分解之后，方程(3.1)变成

$$[L][D][L]^T\{U\}=\{I\}$$

它等价于以下两个方程

$$\left.\begin{aligned}&[L]\cdot\{W\}=\{I\}\\&[L]^T\{U\}=[D]^{-1}\cdot\{W\}\end{aligned}\right\} \tag{3.3}$$

分解实现的代码如下：

```
//Cholesky 分解
```

```
for(iTK = 0;iTK<Gbldata.m_PointNum;iTK++)
{
    for(jTK = 0;jTK<Gbldata.m_HalfBand;jTK++)
    {
        for(p = 0;p<iTK;p++)
        {
            if(iTK - p<Gbldata.m_HalfBand && jTK - p + iTK<Gbldata.m_HalfBand)
                pTK[iTK][jTK] -= pTK[p][iTK - p] * pTK[p][0] * pTK[p][jTK - p + iTK];
        }
        if(jTK! = 0)
            pTK[iTK][jTK] = pTK[iTK][jTK]/pTK[iTK][0];
    }
}
```

它们的解可按下列“回代”公式算出

$$W_i = I_i - \sum_{p=1}^{i-1} L_{ip} \cdot W_p \quad (i = 1,2,\cdots,N)$$

和

$$U_i = W_i/d_i - \sum_{p=i+1}^{N} L_{pi} \cdot U_p \quad (i = N,N-1,\cdots,1) \tag{3.4}$$

回代过程实现代码如下：

```
////////////////////////////////////////////////////////////////
//回代解方程
void UGet(unsigned int NodePosition,int iU)
{
    unsigned int p,iPoint;
    float * pI = new float [Gbldata.m_PointNum];
    for(iPoint = 0;iPoint<Gbldata.m_PointNum;iPoint++)
        pI[iPoint] = 0;

    //电极定位
    Gbldata.m_NodePosition = NodePosition +
        (Gbldata.m_yPointNum/2) * Gbldata.m_xPointNum;

    //电流向量赋值
    if(Gbldata.m_NodePosition % Gbldata.m_xPointNum == 0 ||
        (Gbldata.m_NodePosition + 1) % Gbldata.m_xPointNum == 0)
    {
        pI[Gbldata.m_NodePosition] = Gbldata.m_NodeValue/5;
    }
    else
    {
        pI[Gbldata.m_NodePosition] = Gbldata.m_NodeValue/10;
    }

    //解方程组
    for(iPoint = 0;iPoint<Gbldata.m_PointNum;iPoint++)
    {
        for(p = 0;p<iPoint;p++)
```

```
        {
            if(iPoint - p<Gbldata.m_HalfBand)
                pI[iPoint] = pI[iPoint] - pTK[p][iPoint - p] * pI[p];
        }
    }

    for(iPoint = Gbldata.m_PointNum - 1;
                iPoint>= 0 && iPoint<Gbldata.m_PointNum;iPoint -- )
    {
        pI[iPoint] = pI[iPoint]/pTK[iPoint][0];
        for(p = iPoint + 1;p<Gbldata.m_PointNum;p++)
        {
            if(p - iPoint<Gbldata.m_HalfBand)
                pI[iPoint] = pI[iPoint] - pTK[iPoint][p - iPoint] * pI[p];
        }
    }
}
```

由于对称性，所以只需存储[K]/[L]的下半三角元素。采用图 3-1 所示的压缩存储方法，这样二维数组的列数等于半宽 S，且行与 N×M 相同。

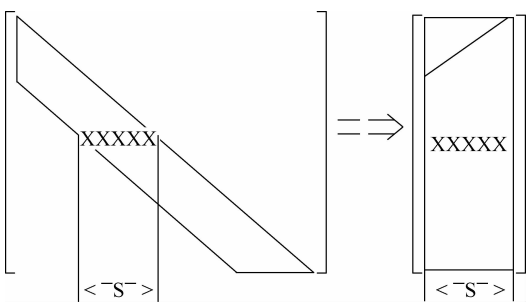

图 3-1　压缩存储示意图

在地球物理勘探中，解决电剖面法或电测深法的正演问题时，需对同一断面计算不同供电位置时的电场分布。供电位置不同只改变方程(3.1)的右端场源项{I}，而不影响刚度矩阵[K]，因而也不改变[K]的 LDL^T 分解。所以，整个解题过程中，只需计算一次刚度矩阵[K]和按下三角矩阵非零元素递推公式或对角矩阵非零元素递推公式对[K]作一次矩阵分解。在移动供电电极位置后，需要做的只是改变场源列矢量{I}，然后依次按下三角方程组回代公式和上三角方程组回代公式重新作“回代”，便可确定新供电位置时的电位分布{U}。由于改变{I}和作“回代”所花的计算量，比形成刚度矩阵[K]和作 LDL^T 分解的计算量小很多，所以，移动供电电极重新计算电位分布{U}，增加的计算量并不大。不像迭代法那样，每移动一次供电电极，就需重新作一次迭代求解，大约增加一倍的计算量。这是直接法相对于迭代法的一大优点。

3.1.2　主要算法

1. 自动剖分

接受参数：m，d，ab。

定义(输出)全局数组：TK[ ][ ],单元矩阵 EK[ ]。
考虑的问题：数组空间的节省存储；剖分时中间 1/3 等距,其余等比。
输出数据：TK[ ][ ],EK[ ][ ]数组。
算法如下。
首单元尺寸：

```
a1 = 2.0 * l/(nx - dx);b1 = 2 * m/(ny - dy)
```

等差步长：

```
dll = (a1 - dx)/(nx - l);dtt = (bl - dy)/(ny - l)
```

化为母单元：

```
tx = 2 * al1/l;ty = 2 * b1/m
  dll = 2 * dll/l;d2 = 2 * dtt/m
  for(iy = 1;iy<= jy;iy++)
    {ey = ey + ty - d2 * (iy - 2)
  for(ix = 1;ix<= jx;ix++)
    {ex = ex + tx - d1 + (ix - 2)
    调用 shap(ex,ey)
    节点坐标：x[mp - 1] = xx,y[mp - 1] = yy
    }
  }
for(iy = 1;iy<= ny;iy++)
  for(ix = 1;ix<= nx;ix++)
```

形成单元节点码：

```
ei[m - 1][],ei[m - 2][]
```

2. 并行化构造

输入数组 TK[ ][ ],EK[ ][ ]；进行并行元素的无关构造。根据参加运算的机器数,将划分的单元按不相关原则构筑成新的数组。

3. 对刚度矩阵[K]作乔勒斯基(LL^T 或 LDL^T)分解

LDL 方法求解方程：输入数组 TK[ ][ ],EK[ ][ ]。
算法如下。

```
for(i = 1;i<= nj2;i++)
  {   io = id[i - 1] - 1
    if i! = 1
      {for(j = mi;j<= I;j++)
        for(k = mij;k<= jm1;k++)
            if(mjj<= jm1)
            tk[ij - 1] = tk[ij - 1] - tk[ik - 1] * tk[kk - 1] * tk[jk - 1]
            If(j = i)
              {jj = id[j - 1];tk[ij - 1] = tk[ij - 1]/tk[jj - 1]
              for(k = 1;k<= m;k++)
```

```
                {p[i-1][k-1]=p[I-1][k-1]-tk[ij-1]*tk[jj-1]p[j-1][k-1]
                }
            }
        }
      for(k=1;k<=m;k++)
      p[I-1][k-1]=p[I-1][k-1]/tk[ij-1]
    }

for(l1=2;li<=nj2;l1++)
  for(j=mi;j<=im1;j++)
    if(mi<=im1)
  {for(k=0;k<=m-1;k++)
p[j-1]k[]=p[j-1][k]-tk[ij-1*p[I-1][k]
}
```

3.2　共轭梯度法简介

共轭梯度法是一种特殊的共轭方向法，用该方法求解线性代数方程组时，它能有效地生成关于求解矩阵共轭的方向向量[77]。由于共轭梯度法对矩阵的元素结构没有特殊要求，所以，它常用于求解一般大型稀疏矩阵的线性方程组。共轭梯度法从理论上讲属于直接法，但是在实际计算过程中，由于不可避免地会出现舍入误差，因此常常作为迭代法来使用，对于高阶方程组更是如此。当方程组的阶数很高时，往往只要经过比阶数小得多的迭代次数，就能得到满足精度要求的近似解。

3.3　梯度法的基本思想

3.3.1　算法原理

如果线性代数方程组

$$AX = B \tag{3.5}$$

的系数矩阵 A 是对称正定的(在本章中如无特殊说明，总认为 A 是对称正定的)，则梯度法的基本思想可以叙述如下：对于任意一个初始向量 X_0(本章所讨论的向量均是列向量)，依次构造一组向量 p_i 及数量 a_i，使得经过公式

$$X_{i+1} = X_i + a_i P_i, i = 0,1,\cdots \tag{3.6}$$

迭代后，得到的迭代值序列

$$X_0, X_1, \cdots, X_i, \cdots$$

收敛于原方程组的精确解 X^*。

由此可见，这种方法的关键是如何根据系数矩阵 A 的对称正定性，逐步构造 a_i 和 p_i。

首先，构造一个二次函数

$$F(X) = (A(X^* - X), X^* - X) \tag{3.7}$$

显然，由 A 的对称正定性，有 $F(X)\geqslant 0$，并且只有当 $X=X^*$ 时，才有 $F(X)=0$。其中 X

即为方程组的精确解。现在，如果用第 $i+1$ 次的迭代值

$$X_{i+1} = X_i + a_i P_i$$

来代替 X，则有

$$\begin{aligned} F(X_i + a_i P_i) &= (A(X^* - X_i - a_i P_i), X^* - X_i - a_i P_i) \\ &= F(X_i) - 2a_i(P_i, A(X^* - X_i)) + a_i^2(P_i, AP_i) \end{aligned}$$

在这一步中，利用了 A 自共轭的性质，即

$$(P_i, A(X^* - X_i)) = (AP_i, X^* - X_i)$$

同时也利用了内积的交换律，即

$$(P_i, A(X^* - X_i)) = (A(X^* - X_i), P_i)$$

其中，$(X^* - X_i)$是第 i 次迭代值与精确的误差向量。令

$$R_i = A(X^* - X_i)$$

且称为残向量。显然有

$$R_i = A(X^* - X_i) = AX^* - AX_i = B - AX_i$$

最后，上述所构造的二次函数就变成

$$F(X_i + a_i P_i) = F(X_i) - 2a_i(P_i, R_i) + a_i^2(P_i, AP_i)$$

接下来选取 a_i，使 $F(X_i + a_i P_i)$达到极小值，即 a_i 满足条件

$$\frac{\partial F(X_i + a_i P_i)}{\partial a_i} = 0$$

由此解出

$$a_i = \frac{(P_i, R_i)}{(P_i, AP_i)} \tag{3.8}$$

同时，还需寻找一个方向(即向量 P_i)，使函数 $F(X)$在 X_i 点最快地下降到 X_{i+1} 点。而函数 $F(X)$在 $X = X_i$ 点的变化率最大的方向是 $F(X)$在这一点的梯度，即

$$gradF\Big|X_i = \left(\frac{\partial F}{\partial x_1}, \frac{\partial F}{\partial x_2}, \cdots, \frac{\partial F}{\partial x_n}\right)^T \Big| X_i$$

综上所述，取梯度作为 P_i 而得到的新的近似值

$$X_{i+1} = X_i + a_i P_i \tag{3.9}$$

的方法称为梯度法，也称为最速下降法。其中 a_i 的计算公式为

$$a_i = \frac{(P_i, R_i)}{(P_i, AP_i)} \tag{3.10}$$

R_i 为残向量，即

$$R_i = B - AX \tag{3.11}$$

但在实际应用中，梯度法并不十分理想，因为在某些情况下可能收敛得很慢。如果能合理地挑选方向向量 P_i，则只要经过有限步(至多 n 步)迭代后，就可得到方程组的精确解。由此提出了共轭梯度法。

3.3.2 对称正定矩阵、向量的正交与共轭变换

设矩阵

$$A=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

中的元素满足下列关系

$$a_{ij}=a_{ji},i,j=1,2,\cdots,n$$

则称矩阵 A 为对称矩阵。也就是说，对称矩阵中的元素是关于主对角线对称的。

对于一个对称矩阵 A 及向量 $X=(x_1,x_2,\cdots,x_n)^T$，可以构造一个二次函数

$$F(X)=X^TAX=(x_1,x_2,\cdots,x_n)A\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

$$=a_{11}x_1^2+\cdots+a_{nn}x_n^2+2a_{12}x_1x_2+\cdots+2a_{n-1,n}x_{n-1}x_n$$

如果对于任意的 $X=(x_1,x_2,\cdots,x_n)^T$，有

$$F(X)\geqslant 0$$

且只有当任意的 $X=(0,0,\cdots,0)^T$ 时，才有 $F(X)=0$，则称矩阵 A 为对称正定矩阵。

二次函数

$$F(X)=X^TAX$$

也可以写成向量的内积形式，即

$$F(X)=(X,AX)$$

设有两个互异的向量

$$X=(x_1,x_2,\cdots,x_n)^T$$

$$Y=(y_1,y_2,\cdots,y_n)^T$$

如果它们的内积为 0，即

$$(X,Y)=0$$

则称向量 X 与 Y 是正交的。

对于两个任意向量 X 和 Y

$$X=(x_1,x_2,\cdots,x_n)^T$$

$$Y=(y_1,y_2,\cdots,y_n)^T$$

如果矩阵 A 与 A^* 满足关系

$$(X,AY)=(A^*X,Y)$$

则称 A^* 为 A 的共轭变换。如果

$$A^*=A$$

则称 A 是自共轭的。显然，根据向量内积的定义，对称矩阵 A 是自共轭的，即对于任意的向量 X 和 Y，有

$$(X,AY)=(AX,Y)$$

其中 A 为对称矩阵。

如果对于向量 X 和 Y，有

$$(X,AY)=(AX,Y)=0$$

则称 X 和 Y 为 A 共轭正交。

3.4 共轭梯度法的基本原理

共轭梯度法的基本思想与梯度法差不多，只是在方向向量 P_i 的选取上有所不同。在共轭梯度法中，所选取的 P_i 为一组 A 共轭正交的向量。构造 A 共轭正交的向量组$\{P_i\}$的计算公式的方法如下。

设 A 共轭正交的向量

$$P_0, P_1, \cdots, P_{i-1}$$

已经产生，并且由此可以算出一系列系数

$$a_0, a_1, \cdots, a_{i-1}$$

从而得到近似解序列

$$X_0, X_1, \cdots, X_i$$

进一步可以得到残向量序列

$$R_0, R_1, \cdots, R_i$$

其中，$R_i = B - AX_i = B - A(X_{i-1} + a_{i-1}P_{i-1}) = \cdots = B - AX_0 - a_0AP_0 - a_1AP_1 - \cdots - a_{i-1}AP_{i-1}$

若取 $X_0 = (0, 0, \cdots, 0)^T$，则有

$$R_0 = B - AX_0 = B$$

并取初值 $P_0 = R_0 = B$，则有

$$R_i = P_0 - a_0AP_0 - a_1AP_1 - \cdots - a_{i-1}AP_{i-1}$$

不难验证，残向量序列 $R_i(i = 0, 1, \cdots)$是互相正交的，即

$$(R_i, R_j) = 0, i \neq j$$

并且

$$R_i = R_{i-1} - a_{i-1}AP_{i-1} \quad 或 \quad P_{i-1} = B - AX_i$$

由此可得

$$AP_{i-1} = -\frac{R_i - R_{i-1}}{a_{i-1}} \tag{3.12}$$

现在，为了构造 P_i，可以通过 $P_0, P_1, \cdots, P_{i-1}$ 及 R_i 的线性组合来得到，即可以假设

$$P_i = a_0P_0 + a_1P_1 + \cdots + a_{i-1}P_{i-1} + bR_i \tag{3.13}$$

然后根据 $P_i(i = 0, 1, 2, \cdots)$的 A 共轭正交的性质依次确定系数 $a_0, a_1, \cdots, a_{i-1}$ 及 b。

由

$$\begin{aligned}(AP_i, P_0) = {} & a_0(AP_0, P_0) + a_1(AP_1, P_0) + \cdots \\ & + a_{i-1}(AP_{i-1}, P_0) + b(AR_i, P_0) + b(AR_i, P_0) = 0\end{aligned}$$

可以解出

$$a_0 = -\frac{b(AR_i, P_0)}{(AP_0, P_0)}$$

同理，可以分别解出

$$a_1 = -\frac{b(AR_i, P_1)}{(AP_1, P_1)}$$

$$\vdots$$

$$a_{i-1} = -\frac{b(AR_i, P_{i-1})}{(AP_{i-1}, P_{i-1})}$$

最后将 $a_0, a_1, \cdots, a_{i-1}$ 代入式(3.13)可得

$$P_i = bR_i - b\left[\frac{(AR_i, P_0)}{(AP_0, P_0)}P_0 + \frac{(AR_i, P_1)}{(AP_1, P_1)}P_1 + \cdots + \frac{(AR_i, P_{i-1})}{(AP_{i-1}, P_{i-1})}P_{i-1}\right]$$

取

$$P_i = R_i - \sum_{j=0}^{i-1}\frac{(AR_i, P_j)}{(AP_j, P_j)}P_j \tag{3.14}$$

由于 A 为对称正定矩阵，所以 A 为自共轭，即有

$$(AR_i, P_j) = (R_i, AP_j), j = 0, 1, \cdots, i-1$$

根据式(3.12)，又有

$$AP_j = -\frac{R_{j+1} - R_j}{a_j}$$

所以，由 R_i 的正交性，当 $j \leqslant i-2$ 时有

$$(R_i, AP_j) = \left(R_i, -\frac{R_{j+1} - R_j}{a_j}\right)$$

$$= -\frac{1}{a_j}[(R_i, R_{j+1}) - (R_i, R_j)] = 0$$

这就说明，在式(3.14)的右端和式中，除了最后一项(即 $j=i-1$)不为 0 外，前面的 $i-1$ 项实际上均为 0，所以得到 P_i 的计算公式为

$$P_i = R_i - \frac{(AR_i, P_{i-1})}{(AP_{i-1}, P_{i-1})}P_{i-1} \tag{3.15}$$

若令

$$\beta_{i-1} = \frac{(AR_i, P_{i-1})}{(AP_{i-1}, P_{i-1})} \tag{3.16}$$

则有

$$P_i = R_i - \beta_{i-1}P_{i-1} \tag{3.17}$$

式(3.17)就是计算 A 共轭正交向量组$\{P_i\}$的递推公式。

由于$\{P_i\}(i=0,1,\cdots)$为 A 共轭正交向量组，因此，不难验证，在式(3.10)中

$$(P_i, R_i) = (P_i, B - AX_i) = (P_i, B) - (P_i, AX_i) = (P_i, B)$$

所以有

$$a_i = \frac{(P_i, B)}{(P_i, AP_i)} \tag{3.18}$$

综上所述，由式(3.18)、式(3.9)、式(3.10)、式(3.16)及式(3.17)可以得到共轭梯度法的递推计算公式如下。

取 $X_0 = (0, 0, \cdots, 0)^T$，且 $R_0 = P_0 = B$。

对于 $i = 0, 1, \cdots, n-1$

$$\begin{cases} a_i = \dfrac{(P_i, B)}{(P_i, AP_i)} \\ X_{i+1} = X_i + a_i P_i \\ R_{i+1} = B - AX_{i+1} \\ \beta_i = \dfrac{(R_{i+1}, AP_i)}{(P_i, AP_i)} \\ P_{i+1} = R_{i+1} - \beta_i P_i \end{cases} \tag{3.19}$$

在按照上述公式进行迭代的过程中，如果做到某步使 $R_i=0$ 时，则迭代结束，说明 X_i 已经等于精确解 X^*。并且由 R_n 与 $R_0, R_1, \cdots, R_{n-1}$ 的正交性可知：$R_n \equiv 0$，这就说明，这个迭代过程是有限的，最多迭代 n 次就能得到精确解。当然，在实际计算过程中，由于舍入误差的影响(递推公式(3.19)强烈地依赖于向量组的正交性，对舍入误差比较敏感)，可能迭代 n 次后还不能得到精确解，只能得到一个近似解。

下面是共轭梯度法的基本算法：

```
Compute R0 = B - AX0 ,P0 = R0
For i = 0,1,…,until convergence Do:
  Compute expressions(3.15)
End Do
```

3.5 共轭梯度法的 C++语言实现

3.5.1 背景说明

```
CG(B[],N,X[],RSQ)
```

- N：整型变量，输入参数，方程组阶数。
- B[ ]：实型数组，输入参数，存放方程组的右端向量 B。
- X[ ]：实型数组，输入、输出参数，输入初始向量 X_0，输出解向量。
- RSQ：实型变量，输出参数，输出 $\| AX-B \|_2^2$。如果 RSQ 较大，即不太小，则说明矩阵是数值奇异的，这时相应的解为最小二乘逼近。

3.5.2 函数定义

函数 CG(B[],N,X[],RSQ)定义程序清单如下：

```
void sparse(double b[],int n,double x[],double rsq)
{
    double eps2,eps = 0.000001;
    double * g,* h,* xi,* xj;
    g = new double [n];
    h = new double [n];
    xi = new double [n];
    xj = new double [n];
```

```
int j;
eps2 = n * pow(eps,2);
int irst = 0;
r1: irst = irst + 1;
asub(x,xi); //xi = Ax
double rp = 0.0;
double bsq = 0.0;
for(j = 0; j<= n - 1; j++)
{
    bsq = bsq + pow(b[j],2);
    xi[j] = xi[j] - b[j];
    rp = rp + pow(xi[j],2);
}
atsub(xi,g); //g = ATxi
for(j = 0; j<= n - 1; j++)
{
    g[j] = -g[j];
    h[j] = g[j];
}
for(int iter = 1; iter<= 10 * n; iter++)
{
    asub(h,xi);
    double anum = 0.0;
    double aden = 0.0;
    for(j = 0; j<= n - 1; j++)
    {
        anum = anum + g[j] * h[j];
        aden = aden + pow(xi[j],2);
    }
    if(aden == 0.0)
    {
        cout<<"very singular matrix"<<endl;
    }
    anum = anum / aden;
    for(j = 0; j<= n - 1; j++)
    {
        xi[j] = x[j];
        x[j] = x[j] + anum * h[j];
    }
    asub(x,xj);
    double rsq = 0.0;
    for(j = 0; j<= n - 1; j++)
    {
        xj[j] = xj[j] - b[j];
        rsq = rsq + pow(xj[j],2);
    }
    if(rsq == rp || rsq <= bsq * eps2)
    {
        delete [] g;
        delete [] h;
        delete [] xi;
```