第二讲　线性方程组直接解法

线性方程组的求解方法

解法概述

- **直接法**: LU 分解, Cholesky 分解, ...
- **迭代法**: 古典迭代法, Krylov 子空间迭代法

本讲介绍直接法, 即 **Gauss 消去法**, 即 **LU 分解**

直接法优点: 稳定可靠, 有限步终止 $\rightarrow$ 在工程界很受欢迎

直接法缺点: 运算量大 $O(n^3)$ $\rightarrow$ 不适合大规模稀疏线性方程组

(针对特殊结构矩阵的快速方法除外)

1 Gauss 消去法和 LU 分解

1.1 LU 分解

考虑线性方程组

$$\boxed{Ax = b} \tag{2.1}$$

其中 $A \in \mathbb{R}^{n\times n}$ 非奇异, $b \in \mathbb{R}^n$ 为给定的右端项.

Gauss 消去法本质上就是对系数矩阵 A 进行 LU 分解:

$$\boxed{A = LU} \tag{2.2}$$

其中 L 是**单位下三角矩阵**, U 为**非奇异上三角矩阵**.

$$Ax = b \iff \begin{cases} Ly = b, \\ Ux = y. \end{cases}$$ $\Longrightarrow$ 只需求解两个三角方程组

算法 1.1 Gauss 消去法

1: 将 A 进行 LU 分解: $\quad A = LU$;

2: 向前回代: 求解 $Ly = b$, 即得 $y = L^{-1}b$;

3: 向后回代: 求解 $Ux = y$, 即得 $x = U^{-1}y = (LU)^{-1}b = A^{-1}b$.

✍ A 非奇异, 则解存在唯一, 但并不一定存在 LU 分解!

定理 (LU 分解的存在性和唯一性) 设 $A \in \mathbb{R}^{n\times n}$. 则存在唯一的单位下三角矩阵 L 和非奇异上三角矩阵 U, 使得 $A = LU$ 的充要条件是 A 的所有顺序主子矩阵 $A_k = A(1:k, 1:k)$ 都非奇异, $k = 1, 2, \ldots, n$.

(板书)

1.2 LU 分解的实现 — 矩阵初等变换

给定一个矩阵

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ a_{31} & a_{32} & \cdots & a_{3n} \\ \vdots & & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \in \mathbb{R}^{n\times n}.$$

- 第一步: 假定 $a_{11} \neq 0$, 构造矩阵

$$L_1 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \\ l_{n1} & 0 & 0 & \cdots & 1 \end{bmatrix}, \quad \text{其中} \quad l_{i1} = \frac{a_{i1}}{a_{11}}, i = 2, 3, \ldots, n.$$

易知 L_1 的逆为

$$L_1^{-1} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ -l_{21} & 1 & 0 & \cdots & 0 \\ -l_{31} & 0 & 1 & \cdots & 0 \\ \vdots & & & \ddots & \\ -l_{n1} & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

用 L_1^{-1} 左乘 A, 并将所得到的矩阵记为 $A^{(1)}$, 则

$$A^{(1)} = L_1^{-1} A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} \end{bmatrix}.$$

- 第二步: 将上面的操作作用在 $A^{(1)}$ 的子矩阵 $A^{(1)}(2:n,2:n)$ 上, 将其第一列除第一个元素外都变为 0: 假定 $a_{22}^{(1)} \neq 0$, 构造矩阵

$$L_2 = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \\ 0 & l_{n2} & 0 & \cdots & 1 \end{bmatrix}, \quad \text{其中} \quad l_{i2} = \frac{a_{i2}^{(1)}}{a_{22}^{(1)}}, i = 3, 4, \ldots, n.$$

用 L_2^{-1} 左乘 $A^{(1)}$, 并将所得到的矩阵记为 $A^{(2)}$, 则

$$A^{(2)} = L_2^{-1}A = L_2^{-1}L_1^{-1}A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & a_{n3}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix}.$$

- 依此类推, 假定 $a_{kk}^{(k-1)} \neq 0\ (k=3,4,\dots,n-1)$,

  我们可以构造一系列的矩阵 $L_3, L_4, \dots, L_{n-1}$, 使得

$$L_{n-1}^{-1} \cdots L_2^{-1} L_1^{-1} A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \cdots & a_{2n}^{(1)} \\ 0 & 0 & a_{33}^{(2)} & \cdots & a_{3n}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n-1)} \end{bmatrix} \triangleq U \ \to \text{上三角}$$

  于是可得 $\boxed{A = LU}$ 其中

$$L = L_1 L_2 \cdots L_{n-1} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{bmatrix}$$

算法 1.2 LU 分解

for $k = 1$ to $n-1$ **do**
    for $i = k+1$ to n **do**
        $l_{ik} = a_{ik}/a_{kk}$ % 计算 L 的第 k 列
    end for
    for $j = k$ to n **do**
        $u_{kj} = a_{kj}$ % 计算 U 的第 k 行
    end for
    for $i = k+1$ to n **do**
        for $j = k+1$ to n **do**
            $a_{ij} = a_{ij} - l_{ik}u_{kj}$ % 更新 $A(k+1:n, k+1:n)$
        end for
    end for
end for

Gauss 消去法的运算量

由算法 1.2 可知, LU 分解的运算量 (加减乘除) 为

$$\sum_{i=1}^{n-1}\left(\sum_{j=i+1}^{n}1+\sum_{j=i+1}^{n}\sum_{k=i+1}^{n}2\right)=\sum_{i=1}^{n-1}\left(n-i+2(n-i)^2\right)=\frac{2}{3}n^3+O(n^2).$$

加上回代过程的运算量 $O(n^2)$, 总运算量为 $\boxed{\frac{2}{3}n^3+O(n^2)}$

✍ 评价算法的一个主要指标是**执行时间**, 但这依赖于计算机硬件和编程技巧等, 因此直接给出算法执行时间是不太现实的. 所以我们通常是统计算法中**算术运算 (加减乘除) 的次数**.

✍ 在数值算法中, 大多仅仅涉及加减乘除和开方运算. 一般地, 加减运算次数与乘法运算次数具有相同的量级, 而除法运算和开方运算次数具有更低的量级.

✍ 为了尽可能地减少运算量, 在实际计算中, 数, 向量和矩阵做乘法运算时的先后执行次序为: 先计算数与向量的乘法, 然后计算矩阵与向量的乘法, 最后才计算矩阵与矩阵的乘法.

矩阵 L 和 U 的存储

当 A 的第 i 列被用于计算 L 的第 i 列后, 在后面的计算中不再被使用.

同样, A 的第 i 行被用于计算 U 的第 i 行后, 在后面计算中也不再使用.

为了节省存储空间, 在计算过程中将 L 的第 i 列存放在 A 的第 i 列, 将 U 的第 i 行存放在 A 的第 i 行, 这样就不需要另外分配空间存储 L 和 U.

计算结束后, A 的上三角部分为 U, 其绝对下三角部分为 L 的绝对下三角部分 (L 的对角线全部为 1, 不需要存储).

算法 1.3 LU 分解

for $k = 1$ to $n - 1$ **do**
    for $i = k + 1$ to n **do**
        $a_{ik} = a_{ik}/a_{kk}$
        for $j = k + 1$ to n **do**
            $a_{ij} = a_{ij} - a_{ik}a_{kj}$
        end for
    end for
end for

```matlab
% Matlab code : LU 分解
function A = mylu(A)
n=size(A,1);
for k=1:n-1
    if A(k,k) == 0
        fprintf('Error: A(%d,%d)=0!\n', k, k);
        return;
    end
    for i=k+1:n
        A(i,k)=A(i,k)/A(k,k);
        for j=k+1:n
            A(i,j)=A(i,j)-A(i,k)*A(k,j);
        end
    end
end
```

为了充分利用 Matlab 向量运算优势, 提高效率, 可改写为

```
function A = mylu(A)
n=size(A,1);
for k=1:n-1
    if A(k,k) == 0
        fprintf('Error: A(%d,%d)=0!\n', k, k);
        return;
    end
    A(k+1:n,k)=A(k+1:n,k)/A(k,k);
    A(k+1:n,k+1:n)=A(k+1:n,k+1:n)-A(k+1:n,k)*A(k,k+1:n);
end
```

✍ 根据指标的循环次序, 算法 1.3 也称为 KIJ 型 LU 分解. 实际计算中一般不建议使用: 对指标 k 的每次循环, 都需要更新 A 的第 $k+1$ 至第 n 行, 这种反复读取数据的做法会使得计算效率大大降低.

若按行存储, 一般采用后面介绍的 IKJ 型 LU 分解.

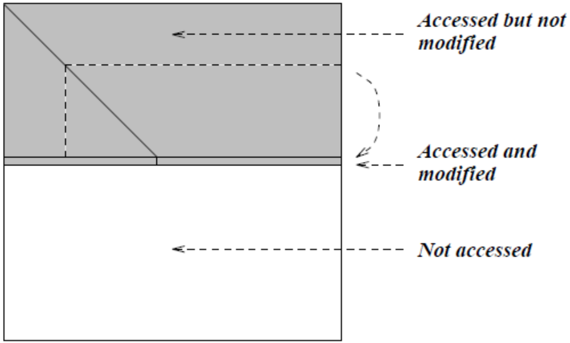
1.3 IKJ 型 LU 分解

如果数据是按行存储, 如 C/C++, 可采用下面的 IKJ 型 LU 分解.

算法 1.4 LU 分解 (IKJ 型)

for $i = 2$ to n **do**
  for $k = 1$ to $i - 1$ **do**
    $a_{ik} = a_{ik}/a_{kk}$
    for $j = k + 1$ to n **do**
      $a_{ij} = a_{ij} - a_{ik}a_{kj}$
    end for
  end for
end for

上述算法可以用下图来描述.



思考: 如果数据按列存储, 如 FORTRAN/MATLAB, 如何设计算法?

1.4 待定系数法计算 LU 分解

设 $A = LU$, 即

$$
\begin{bmatrix}
a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\
a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\
a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\
\vdots & & & \ddots & \vdots \\
a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn}
\end{bmatrix}
=
\begin{bmatrix}
1 & & & & \\
l_{21} & 1 & & & \\
l_{31} & l_{32} & 1 & & \\
\vdots & & & \ddots & \\
l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1
\end{bmatrix}
\begin{bmatrix}
u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\
 & u_{22} & u_{23} & \cdots & u_{2n} \\
 & & u_{33} & \cdots & u_{2n} \\
 & & & \ddots & \vdots \\
 & & & & u_{nn}
\end{bmatrix}
$$

(1) 比较等式两边的**第一行**, 可得

$$u_{1j} = a_{1j}, \quad j = 1, 2, \ldots, n.$$

再比较等式两边的**第一列**, 可得

$$a_{i1} = l_{i1} u_{11} \;\Rightarrow\; l_{i1} = a_{i1}/u_{11}, \quad i = 2, 3, \ldots, n.$$

(2) **比较等式两边的第二行, 可得**

$$a_{2j} = l_{21}u_{1j} + a_{2j} \ \Rightarrow \ u_{2j} = a_{2j} - l_{21}u_{1j}, \quad j = 2, 3, \ldots, n.$$

再比较等式两边的第二列, 可得

$$a_{i2} = l_{i1}u_{12} + l_{i2}u_{22} \ \Rightarrow \ l_{i1} = (a_{i2} - l_{i1}u_{12})/u_{22}, \quad i = 3, 4, \ldots, n.$$

(3) 以此类推, 第 k 步时, 比较等式两边的第 k 行, 可得

$$\boxed{u_{kj} = a_{kj} - (l_{k1}u_{1j} + \cdots + l_{k,k-1}u_{k-1,j}),}$$

$$j = k, k+1, \ldots, n.$$

比较等式两边的第 k 列, 可得

$$\boxed{l_{ik} = (a_{ik} - l_{i1}u_{1k} - \cdots - l_{i,k-1}u_{k-1,k})/u_{kk},}$$

$$i = k+1, k+2, \ldots, n.$$

直到第 n 步, 即可计算出 L 和 U 的所有元素.

同样, 我们可以利用 A 来存储 L 和 U. 算法描述如下:

算法 1.5 LU 分解 (待定系数法或 Doolittle 方法)

for $k = 1$ to n **do**
  for $j = k$ to n **do**
    $$a_{kj} = a_{kj} - \sum_{i=1}^{k-1} a_{ki} a_{ij},$$
  end for
  for $i = k+1$ to n **do**
    $$a_{ik} = \frac{1}{a_{kk}} \left(a_{ik} - \sum_{j=1}^{k-1} a_{ij} a_{jk} \right),$$
  end for
end for

```
function A = mylu2(A)
[n,n]=size(A);
for k=1:n
    A(k,k)=A(k,k)-A(k,1:k-1)*A(1:k-1,k);
    if (A(k,k)==0)
        fprintf('Error: A(%d,%d)=0!\n', i,i);
        return;
    end
    A(k,k+1:n)=A(k,k+1:n)-A(k,1:k-1)*A(1:k-1,k+1:n);
    A(k+1:n,k)=A(k+1:n,k)-A(k+1:n,1:k-1)*A(1:k-1,k);
    A(k+1:n,k)=A(k+1:n,k)/A(k,k);
end
```

1.5 三角方程求解

得到 A 的 LU 分解后, 我们最后需要用回代法求解两个三角方程组

$$Ly = b, \quad Ux = y.$$

算法 1.6 向前回代求解 $Ly = b$ (假定 L 是一般的非奇异下三角矩阵)

$y_1 = b_1 / l_{11}$
for $i = 2 : n$ **do**
  for $j = 1 : i - 1$ **do**
    $b_i = b_i - l_{ij} y_j$
  end for
  $y_i = b_i / l_{ii}$
end for

如果数据是按列存储的, 则采用列存储方式效率会高一些. 下面是按列存储方式求解上三角方程组.

算法 1.7 向后回代求解 $Ux = y$

1: **for** $i = n : -1 : 1$ **do**
2: $\quad x_i = y_i / u_{ii}$
3: $\quad$ **for** $j = i - 1 : -1 : 1$ **do**
4: $\quad\quad y_j = y_j - x_i u_{ji}$
5: $\quad$ **end for**
6: **end for**

这两个算法的运算量均为 $n^2 + O(n)$.

1.6 选主元 LU 分解

- 在 LU 分解算法 1.2 中, 我们称 $a_{kk}^{(k-1)}$ 为 主元 .
  如果 $a_{kk}^{(k-1)} = 0$, 则算法就无法进行下去.
- 即使 $a_{kk}^{(k-1)}$ 不为零, 但如果 $|a_{kk}^{(k-1)}|$ 的值很小, 由于舍入误差的原因, 也可能会给计算结果带来很大的误差.
- 此时我们就需要通过 选主元 来解决这个问题.

例 用 LU 分解求解线性方程组 $Ax=b$, 其中

$$A=\begin{bmatrix}0.02 & 61.3\\ 3.43 & -8.5\end{bmatrix},\quad b=\begin{bmatrix}61.5\\ 25.8\end{bmatrix},$$

要求在运算过程中保留 3 位有效数字.

$(x_1\approx -20.7, x_2\approx 1.01)$ (精确解 $x_1=10.0$ 和 $x_2=1.00$) (板书)

选主元 LU 分解

定理 设 $A \in \mathbb{R}^{n\times n}$ 非奇异, 则存在置换矩阵 P_L, P_R, 以及单位下三角矩阵 L 和非奇异上三角矩阵 U, 使得 $P_L A P_R = LU$. 其中 P_L 和 P_R 中只有一个是必需的.

(板书)

第 k 步时, 如何选取置换矩阵 $P_L^{(k)}$ 和 $P_R^{(k)}$?

选法一. 使得主元为剩下的矩阵中绝对值最大, 这种选取方法称为 "全主元 Gauss 消去法", 简称 GECP (Gaussian elimination with complete pivoting)

选法二. 使得主元为第 k 列的第 k 到第 n 个元素中绝对值最大, 这种选取方法称为 "部分选主元 Gauss 消去法", 简称 GEPP (Gaussian elimination with partial pivoting), 此时 $P_R^{(k)} = I$, 因此也称为列主元 Gauss 消去法.

✍ 在绝大多数情况下, GEPP 是向后稳定的, 但理论上也存在失败的例子.

✍ GECP 比 GEPP 更稳定, 但工作量大.

✍ 事实上, 在大部分实际应用中, GEPP 与 GECP 具有同样的数值稳定性.

算法 1.8 部分选主元 LU 分解

```
p = 1 : n    % 用于记录置换矩阵
for k = 1 to n − 1 do
    [a_max, l] = max_{k≤i≤n} |a_ik|    % 选列主元, l 表示主元所在的行
    if l ≠ k then
        for j = 1 to n do
            tmp = a_kj, a_kj = a_lj, a_lj = tmp    % 交换两行
        end for
        tmp = p(k), p(k) = p(l), p(l) = tmp    % 更新置换矩阵
    end if
    for i = k + 1 to n do
        a_ik = a_ik / a_kk    % 计算 L 的第 k 列
    end for
    for i = k + 1 to n do
        for j = k + 1 to n do
            a_ij = a_ij − a_ik * a_kj    % 更新 A(k + 1 : n, k + 1 : n)
        end for
    end for
end for
```

```
function [A,p] = myplu(A)
[n,n]=size(A); p=1:n;
for i=1:n-1
    [a,k]=max(abs(A(i:n,i)));
    if a==0
        error('Error: 第 %d 步的列主元为 0!\n', i);
    end
    k=k+i-1;
    if k~=i
        tmp=A(i,:); A(i,:)=A(k,:); A(k,:)=tmp;
        tmp=p(i); p(i)=p(k); p(k)=tmp;
    end
    A(i+1:n,i)=A(i+1:n,i)/A(i,i);
    A(i+1:n,i+1:n)=A(i+1:n,i+1:n)-A(i+1:n,i)*A(i,i+1:n);
end
```

例 用部分选主元 LU 分解求解线性方程组 $Ax = b$, 其中

$$A = \begin{bmatrix} 0.02 & 61.3 \\ 3.43 & -8.5 \end{bmatrix}, \quad b = \begin{bmatrix} 61.5 \\ 25.8 \end{bmatrix},$$

要求在运算过程中保留 3 位有效数字.

$(x_1 \approx 10.0, x_2 \approx 0.998)$ (精确解 $x_1 = 10.0$ 和 $x_2 = 1.00$)

(板书)

1.7 矩阵求逆

我们可以通过部分选主元 LU 分解来计算矩阵的逆. 设 $PA = LU$, 则

$$A^{-1} = P^T U^{-1} L^{-1},$$

等价于求解下面 $2n$ 个三角线性方程组

$$Ly_i = Pe_i, \quad Ux_i = y_i, \quad i = 1, 2, \ldots, n.$$

思考: 也可以分别计算 L^{-1} 和 U^{-1}, 然后相乘. 哪种方法划算?

2 特殊方程组的求解

2.1 对称正定线性方程组

我们首先给出对称正定矩阵的几个基本性质.

- A 对称正定当且仅当 A 对称且所有特征值都是正的;
- A 对称正定当且仅当 $X^\intercal AX$ 对称正定, 其中 $X \in \mathbb{R}^{n\times n}$ 是一个任意的非奇异矩阵;
- 若 A 对称正定, 则 A 的任意主子矩阵都对称正定;
- 若 A 对称正定, 则 A 的所有对角线元素都是正的, 且

$$\max_{i\neq j}\{|a_{ij}|\} < \max_{i}\{a_{ii}\},$$

即绝对值最大的元素出现在对角线上.

Cholesky 分解

定理 (Cholesky 分解) 设 $A \in \mathbb{R}^{n\times n}$ 对称正定, 则存在唯一的对角线元素为正的下三角矩阵 L, 使得

$$A = LL^{\intercal}.$$

该分解称为 **Cholesky 分解**.

(板书)

Cholesky 分解的实现

设 $A = LL^{\intercal}$, 即

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

直接比较等式两边的元素可得

$$a_{ij} = \sum_{k=1}^{n} l_{ik} l_{jk} = l_{jj} l_{ij} + \sum_{k=1}^{j-1} l_{ik} l_{jk}, \quad i, j = 1, 2, \ldots, n.$$

根据上面的计算公式, 可得下面的算法:

算法 2.1 Cholesky 分解算法

for $j = 1$ to n **do**

$$l_{jj} = \left(a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 \right)^{1/2}$$

  for $i = j+1$ to n **do**

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) / l_{jj}$$

  end for

end for

几点说明

- 与 LU 分解一样, 可以利用 A 的下三角部分来存储 L
- Cholesky 分解算法的运算量为 $\frac{1}{3}n^3 + O(n^2)$, 大约为 LU 分解的一半
- Cholesky 分解算法是稳定的, 稳定性与全主元 Gauss 消去法相当, 故不需要选主元
- 基于 Cholesky 分解的求解方法称为 **平方根法**

改进的 Cholesky 分解算法

为了避免开方运算, 我们可以将 A 分解为: $A = LDL^\intercal$, 即

$$
\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}
=
\begin{bmatrix} 1 & & & \\ l_{21} & 1 & & \\ \vdots & & \ddots & \\ l_{n1} & \cdots & l_{n,n-1} & 1 \end{bmatrix}
\begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix}
\begin{bmatrix} 1 & l_{21} & \cdots & l_{n1} \\ & 1 & \cdots & l_{n2} \\ & & \ddots & \vdots \\ & & & 1 \end{bmatrix}
$$

通过待定系数法可得

$$
a_{ij} = \sum_{k=1}^{n} l_{ik} d_k l_{jk} = d_j l_{ij} + \sum_{k=1}^{j-1} l_{ik} d_k l_{jk}, \quad i, j = 1, 2, \ldots, n.
$$

基于该分解的求解对称正定线性方程组的算法称为 **改进的平方根法**

算法 2.2 改进的平方根法

for $j = 1$ to n **do** % 先计算分解
  $d_j = a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 d_k$
  for $i = j + 1$ to n **do**
    $l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_k l_{jk}) / d_j$
  end for
end for
$y_1 = b_1$ % 解方程组: $Ly = b$ 和 $DL^{\intercal} x = y$
for $i = 2$ to n **do**
  $y_i = b_i - \sum_{k=1}^{i-1} l_{ik} y_k$
end for
$x_n = y_n / d_n$
for $i = n - 1$ to 1 **do**
  $x_i = y_i / d_i - \sum_{k=i+1}^{n} l_{ki} x_k$
end for

2.2 对称不定线性方程组

$A \to$ 非奇异, **对称** 但 **不定**

若 A 存在 LU 分解, 即 $A = LU$, 则可写成 $\boxed{A = LDL^{\mathsf{T}}}$

然而, 当 A 不定时, 其 LU 分解不一定存在.

若采用选主元 LU 分解, 则其对称性将被破坏.

为了保持对称性, 在选主元时必须对行列进行同样的置换, 即选取置换矩阵 P, 使得

$$PAP^{\mathsf{T}} = LDL^{\mathsf{T}}. \tag{2.3}$$

通常称 (2.3) 为对称矩阵的 $\mathrm{LDL}^{\mathrm{T}}$ 分解.

不幸的是, 这样的置换矩阵可能不一定存在, 即分解 (2.3) 不一定存在.

例 设对称矩阵

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}.$$

由于 A 的对角线元素都是 0, 对任意置换矩阵 P, 矩阵 $PAP^{\intercal}$ 的对角线元素仍然都是 0. 因此, 矩阵 A 不存在 $\mathrm{LDL}^{\mathrm{T}}$ 分解.

方法一：Aasen 算法

1971 年, Aasen 提出了下面的分解

$$\boxed{PAP^{\intercal} = LTL^{\intercal}},$$

其中 P 为置换矩阵, L 为单位下三角矩阵, T **为对称三对角矩阵**.

该分解本质上与部分选主元 LU 分解是一样的,

思考：该分解怎么实现?

方法二：块 LDL$^{\mathrm{T}}$ 分解

设 A 对称非奇异, 则存在置换矩阵 P 使得

$$PAP^{\intercal} = \begin{bmatrix} B & E^{\intercal} \\ E & C \end{bmatrix}, \qquad B \in \mathbb{R} \text{ 或 } B \in \mathbb{R}^{2\times 2}, \text{且非奇异.}$$

思考：如何实现?

于是可以对 $PAP^\intercal$ 进行块对角化, 即

$$PAP^\intercal = \begin{bmatrix} I & 0 \\ EB^{-1} & I \end{bmatrix} \begin{bmatrix} B & 0 \\ 0 & C - EB^{-1}E^\intercal \end{bmatrix} \begin{bmatrix} I & B^{-1}E^\intercal \\ 0 & I \end{bmatrix},$$

其中 $C - EB^{-1}E^\intercal$ 是 Schur 补, 对称非奇异.

不断重复以上过程, 就可以得到 A 的块 $\mathrm{LDL^T}$ 分解:

$$\boxed{PAP^\intercal = L\tilde{D}L^\intercal},$$

其中 $\tilde{D}$ 是 **拟对角矩阵** , 即块对角矩阵且对角块的大小为 1×1 或 2×2.

选主元块 $\mathrm{LDL}^{\mathsf{T}}$ 分解

实际使用时，需要考虑选主元策略, 目前常用的策略有:

- 全主元策略: 由 Bunch 和 Parlett 于 1971 年提出, 并证明了其稳定性. 但需要进行 $n^3/6$ 次比较运算, 代价比较昂贵.
- 部分选主元策略: 由 Bunch 和 Kaufman 于 1977 年提出, 将比较运算复杂度降低到 $O(n^2)$ 量级, 而且具有较满意的向后稳定性. 因此被广泛使用.
- Rook 策略: 由 Ashcraft, Grimes 和 Lewis 于 1998 年提出, 整体上与部分选主元类似, 但在选主元时加了一层迭代, 精度更高.

✍ **目前大部分软件都采用部分选主元块 $\mathrm{LDL}^{\mathsf{T}}$ 分解算法.**

2.3 三对角线性方程组

$$A = \begin{bmatrix} b_1 & c_1 & & \\ a_1 & \ddots & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ & & a_{n-1} & b_n \end{bmatrix}.$$

我们假定

$$|b_1| > |c_1| > 0, \quad |b_n| > |a_{n-1}| > 0, \tag{2.4}$$

$$|b_i| \geq |a_{i-1}| + |c_i|, \quad a_i c_i \neq 0, \quad i = 1, \ldots, n-1. \tag{2.5}$$

即 A 是 **不可约弱对角占优**

思考：如果 A 可约, 怎么处理? (什么情况下，三对角矩阵可约？)

此时, 我们可以得到下面的三角分解

$$A = \begin{bmatrix} b_1 & c_1 & & \\ a_1 & \ddots & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ & & a_{n-1} & b_n \end{bmatrix} = \begin{bmatrix} \alpha_1 & & & \\ a_1 & \alpha_2 & & \\ & \ddots & \ddots & \\ & & a_{n-1} & \alpha_n \end{bmatrix} \begin{bmatrix} 1 & \beta_1 & & \\ & 1 & \ddots & \\ & & \ddots & \beta_{n-1} \\ & & & 1 \end{bmatrix} \triangleq LU.$$

得递推公式:

$$\alpha_1 = b_1, \quad \beta_1 = c_1/\alpha_1 = c_1/b_1,$$

$$\begin{cases} \alpha_i = b_i - a_{i-1}\beta_{i-1}, \\ \beta_i = c_i/\alpha_i = c_i/(b_i - a_{i-1}\beta_{i-1}), \quad i = 2, 3, \ldots, n-1 \end{cases}$$

$$\alpha_n = b_n - a_{n-1}\beta_{n-1}.$$

为了使得算法能够顺利进行下去, 我们需要证明 $\alpha_i \neq 0$.

定理 设三对角矩阵 A 满足条件 (2.4) 和 (2.5). 则 A 非奇异, 且

(1) $|\alpha_1| = |b_1| > 0$;

(2) $0 < |\beta_i| < 1, i = 1, 2, \ldots, n-1$;

(3) $0 < |c_i| \leq |b_i| - |a_{i-1}| < |\alpha_i| < |b_i| + |a_{i-1}|, i = 2, 3, \ldots, n$;

(板书)

算法 2.3 追赶法

$\beta_1 = c_1/b_1$

$y_1 = f_1/b_1$

for $i = 2$ to $n-1$ **do**

  $\alpha_i = b_i - a_{i-1}\beta_{i-1}$

  $\beta_i = c_i/\alpha_i$

  $y_i = (f_i - a_{i-1}y_{i-1})/\alpha_i$

end for

$\alpha_n = b_n - a_{n-1}\beta_{n-1}$

$y_n = (f_n - a_{n-1}y_{n-1})/\alpha_n$

$x_n = y_n$

for $i = n-1$ to 1 **do**

  $x_i = y_i - \beta_i x_{i+1}$

end for

✍ **追赶法** (也称为 **Thomas 算法**) 的运算量大约为 $8n$.

✍ 具体计算时, 由于求解 $Ly = f$ 与矩阵 LU 分解是同时进行的, 因此, α_i 可以不用存储. 但 β_i 需要存储.

✍ 由于 $|\beta_i| < 1$, 因此在回代求解 x_i 时, 误差可以得到有效控制.

需要指出的是, 我们也可以考虑下面的分解

$$A=\begin{bmatrix} b_1 & c_1 & & \\ a_1 & \ddots & \ddots & \\ & \ddots & \ddots & c_{n-1} \\ & & a_{n-1} & b_n \end{bmatrix}=\begin{bmatrix} 1 & & & \\ \gamma_1 & 1 & & \\ & \ddots & \ddots & \\ & & \gamma_{n-1} & 1 \end{bmatrix}\begin{bmatrix} \alpha_1 & c_1 & & \\ & \alpha_2 & \ddots & \\ & & \ddots & c_{n-1} \\ & & & \alpha_n \end{bmatrix}. \tag{2.6}$$

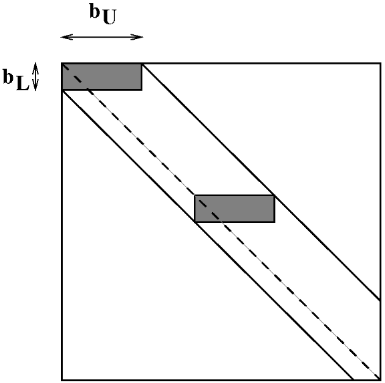
但此时 $|\gamma_i|$ 可能大于 1. 比如当 $|b_1|<|a_1|$ 时, $|\gamma_1|=|a_1/b_1|>1$. 所以在回代求解时, 误差可能得不到有效控制. 同时, 计算 γ_i 时也可能会产生较大误差 (大数除小数). **如果 A 列对角占优, 则可以保证 $|\gamma_i|<1$.**

✍ 所以，如果 A 是 (行) 对角占优, 则采用前面的分解; 如果 A 是**列对角占优**, 则采用分解 (2.6).

2.4 带状线性方程组

设 $A \in \mathbb{R}^{n\times n}$ 是带状矩阵, 其下带宽为 b_L, 上带宽为 b_U, 即

$$a_{ij} = 0 \quad \text{for} \quad i > j + b_L \ \text{ or } \ i < j - b_U \,.$$



对于带状矩阵, 其 LU 分解有如下性质:

定理 设 $A \in \mathbb{R}^{n\times n}$ 是带状矩阵, 其下带宽为 b_L, 上带宽为 b_U. 若 $A = LU$ 是**不选主元的 LU 分解**, 则 L 为下带宽为 b_L 的带状矩阵, U 为上带宽为 b_U 的带状矩阵.

练习：统计求解带状矩阵 $Ax = b$ 的运算量.

若采用部分选主元的 LU 分解, 则有

定理 设 $A \in \mathbb{R}^{n\times n}$ 是带状矩阵, 其下带宽为 b_L, 上带宽为 b_U. 若 $PA = LU$ 是**部分选主元的 LU 分解**, 则 U 为上带宽不超过 $b_L + b_U$ 的带状矩阵, L 为下带宽为 b_L 的 "基本带状矩阵", 即 L 每列的非零元素不超过 $b_L + 1$ 个.

2.5 Toeplitz 线性方程组

Toeplitz 矩阵

$$T_n = \begin{bmatrix} t_0 & t_{-1} & \cdots & t_{-n+1} \\ t_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_{-1} \\ t_{n-1} & \cdots & t_1 & t_0 \end{bmatrix}$$

✍ 易知, T_n 是反向对称 (persymmetric) 矩阵.

记 J_n 为 n 阶 **反向单位矩阵**:

$$J_n = \begin{bmatrix} & & & 1 \\ & & 1 & \\ & \iddots & & \\ 1 & & & \end{bmatrix}$$

✍ 易知 $J_n^\intercal = J_n^{-1} = J_n$.

引理 矩阵 $A \in \mathbb{R}^{n\times n}$ 是反向对称矩阵当且仅当

$$A = J_n A^\intercal J_n \quad 或 \quad J_n A = A^\intercal J_n.$$

若 A 可逆, 则可得

$$A^{-1} = J_n^{-1}\left(A^{\intercal}\right)^{-1} J_n^{-1} = J_n \left(A^{-1}\right)^{\intercal} J_n,$$

即 **反向对称矩阵的逆也是反向对称矩阵**.

Toeplitz 矩阵的逆

Toeplitz 矩阵的逆是反向对称矩阵, 但不一定是 Toeplitz 矩阵.

Yule-Walker 方程组

假定 T_n **对称正定**, 考虑线性方程组

$$\boxed{T_n x = -r_n}$$

其中 $r_n = [t_1, t_2, \ldots, t_{n-1}, t_n]^\intercal$. 这类线性方程组称为 **Yule-Walker 方程组**, 其中 t_n 为任意给定的实数.

由于 T_n 对称正定, 所以 $t_0 > 0$.

因此我们可以对 T_n 的对角线元素进行单位化.

不失一般性, 我们假定 T_n 的对角线元素为 1, 即

$$T_n = \begin{bmatrix} 1 & t_1 & \cdots & t_{n-1} \\ t_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_1 \\ t_{n-1} & \cdots & t_1 & 1 \end{bmatrix}.$$

由于方程组右端项的特殊性, 我们可以通过 递推/递归 技巧来求解.

求解 Yule-Walker 方程组的递推方法

记 $\boxed{T_k x = -r_k}$ 的解为 $x^{(k)}$ $\Longrightarrow$ 假定已经求出

考虑 $\boxed{T_{k+1} x = -r_{k+1}}$, 设解为 $x^{(k+1)} = \begin{bmatrix} z^{(k)} \\ \alpha_k \end{bmatrix}$.

代入后可得 递推公式:

$$\boxed{\alpha_k = \frac{-t_{k+1} - r_k^{\intercal} J_k x^{(k)}}{1 + r_k^{\intercal} x^{(k)}}, \quad z^{(k)} = x^{(k)} + \alpha_k J_k x^{(k)}} \qquad k = 1, 2, \ldots.$$

因此, 可以从一阶 Yule-Walker 方程出发, 利用递推公式计算 $T_n x = -r_n$ 的解.

思考: 总的运算量 (乘法与加减运算) 大约是多少?

减少运算量

引入辅助变量 $\boxed{\beta_k \triangleq 1 + r_k^\intercal x^{(k)}}$, 则

$$\begin{aligned}\beta_{k+1} &= 1 + r_{k+1}^\intercal x^{(k+1)} \\ &= 1 + [r_k^\intercal, t_{k+1}] \begin{bmatrix} x^{(k)} + \alpha_k J_k x^{(k)} \\ \alpha_k \end{bmatrix} \\ &= 1 + r_k^\intercal x^{(k)} + \alpha_k (t_{k+1} + r_k^\intercal J_k x^{(k)}) \\ &= (1 - \alpha_k^2)\beta_k.\end{aligned}$$

总运算量降为 $2n^2$.

这就是求解 Yule-Walker 方程组的 **Durbin 算法**

算法 2.4 求解 Yule-Walker 方程组的 Durbin 算法

输入数据: $t = [t_1, t_2, \ldots, t_n]$ % 注: 这里假定 $t_0 = 1$

$x(1) = -t_1, \beta = 1, \alpha = -t_1$

for for $k = 1$ to $n - 1$ **do**

$\quad \beta = (1 - \alpha^2)\beta$

$\quad \alpha = -\left(t_{k+1} - \sum_{i=1}^{k} t_{k+1-i} x(i)\right) / \beta$

$\quad x(1:k) = x(1:k) + \alpha x(k:-1:1)$

$\quad x(k+1) = \alpha$

end for

一般右端项的对称正定 Toeplitz 线性方程组

考虑一般右端项的方程组 $\boxed{T_n x = b}$, 其中 T_n 对称正定.

我们仍然利用递推方法来求解: 假定 $x^{(k)}$ 和 $y^{(k)}$ 是下面两个方程组的解

$$\boxed{T_k x = [b_1, b_2, \ldots, b_k]^\intercal}, \qquad \boxed{T_k y = -[t_1, t_2, \ldots, t_k]^\intercal}$$

设 $x^{(k+1)} = \begin{bmatrix} z^{(k)} \\ \mu_k \end{bmatrix}$ 是 $\boxed{T_{k+1} x = b^{(k+1)}}$ 的解, 则可得

$$\boxed{z^{(k)} = x^{(k)} + \mu_k J_k y^{(k)}, \quad \mu_k = \frac{b_{k+1} - r_k^\intercal J_k x^{(k)}}{1 + r_k^\intercal y^{(k)}}}$$

Levinson 算法

于是, 我们可以先计算 $T_k x = b^{(k)}$ 和 $T_k x = -r_k$ 的解, 然后利用上述公式得到 $T_{k+1} x = b^{(k+1)}$ 的解, 这就是 **Levinson 算法**, 总运算量约 $4n^2$.

算法 2.5 求解对称正定 Toeplitz 线性方程组的 Levinson 算法

$$
\begin{array}{l}
\textbf{输入数据: } t = [t_1, t_2, \ldots, t_{n-1}] \textbf{ 和 } b = [b_1, b_2, \ldots, b_n] \qquad \color{red}{\%\ \text{假定 } t_0 = 1} \\
y(1) = -t_1,\ x(1) = b_1,\ \beta = 1,\ \alpha = -t_1 \\
\textbf{for } k = 1 \text{ to } n-1 \textbf{ do} \\
\qquad \beta = (1-\alpha^2)\beta, \quad \mu = \left(b_{k+1} - \displaystyle\sum_{i=1}^{k} t_{k+1-i}\, x(i) \right) / \beta \\
\qquad x(1:k) = x(1:k) + \mu y(k:-1:1), \quad x(k+1) = \mu \\
\qquad \textbf{if } k < n-1 \textbf{ then} \\
\qquad\qquad \alpha = -\left(t_{k+1} + \displaystyle\sum_{i=1}^{k} t_{k+1-i}\, y(i) \right) / \beta \\
\qquad\qquad y(1:k) = y(1:k) + \alpha y(k:-1:1), \quad y(k+1) = \alpha \\
\qquad \textbf{end if} \\
\textbf{end for}
\end{array}
$$

注记

在数学与工程的许多应用中都会出现 Toeplitz 线性方程组. Levinson 算法是较早的快速算法, 但并不稳定 (只具有弱稳定性). 后来人们提出了各种各样的快速和超快速算法:

方法	运算量	存储量
Fast stable	$\geq 20n^2$	$\geq n^2/2$
Fast but unstable	$\geq 3n^2$	$\geq 4n$
Superfast and “unstable”	$O(n\log^2 n)$	$O(n)$
Superfast preconditioner	$O(n\log n)$	$O(n)$

- Fast : Levinson-Durbin (1946), Trench (1964), ...
- Fast stable: Bareiss (1969), Chandrasekaran and Sayed (1998), Gu (1998), ...
- Superfast: Brent, Gustavson and Yun (1980), Bitmead and Anderson (1980), Morf (1980), de Hoog (1987), Ammar and Gragg (1988), ...
- Superfast Preconditioners: Strang, Chan, Chan, Tyrtyshnikov, ...

3 扰动分析

扰动方程

考虑线性方程组

$$Ax = b$$

设 x_* 是精确解, $\hat{x}$ 是通过数值计算得到的近似解.

向后误差分析 假定 $\hat{x}$ 满足线性方程组

$$(A + \delta A)\hat{x} = b + \delta b$$

下面讨论 $\delta x \triangleq \hat{x} - x_*$ 的大小.

3.1 δx 与 $\hat{x}$ 的关系

定理 设 $\|\cdot\|$ 是任一向量范数（当该范数作用在矩阵上时就是相应的导出范数），则 δx 与 $\hat{x}$ 满足下面的关系式

$$\frac{\|\delta x\|}{\|\hat{x}\|} \leq \|A^{-1}\| \cdot \|A\| \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|A\| \cdot \|\hat{x}\|} \right).$$

当 $\delta b = 0$ 时, 有

$$\boxed{\frac{\|\delta x\|}{\|\hat{x}\|} \leq \kappa(A) \frac{\|\delta A\|}{\|A\|}}$$

(板书)

3.2 δx 与 x_* 的关系

引理 设 $\|\cdot\|$ 是任一算子范数, $X \in \mathbb{R}^{n\times n}$. 若 $\|X\| < 1$, 则 $I - X$ 可逆, 且有

$$(I-X)^{-1} = \sum_{k=0}^{\infty} X^k \qquad \text{和} \qquad \|(I-X)^{-1}\| \leq \frac{1}{1-\|X\|}.$$

(板书)

定理 设 $A \in \mathbb{R}^{n\times n}$ 非奇异且 $\|A^{-1}\| \cdot \|\delta A\| < 1$, 则

$$\frac{\|\delta x\|}{\|x_*\|} \leq \frac{\kappa(A)}{1 - \kappa(A) \cdot \frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta b\|}{\|b\|} \right).$$

如果 $\|\delta A\| = 0$, 则

$$\frac{1}{\kappa(A)} \frac{\|\delta b\|}{\|b\|} \leq \frac{\|\delta x\|}{\|x_*\|} \leq \kappa(A) \frac{\|\delta b\|}{\|b\|}$$

(板书)

定理 设 $A \in \mathbb{R}^{n\times n}$ 非奇异, 则有

$$\min\left\{\frac{\|\delta A\|_2}{\|A\|_2} \;:\; A+\delta A \text{ 奇异}\right\} = \frac{1}{\kappa_2(A)}$$

(证明见讲义, 留作自习)

✍ 上述定理中的结论对所有 p-范数都成立.

✍ **度量**

$$\mathrm{dist}_p(A) \triangleq \min\left\{\frac{\|\delta A\|_p}{\|A\|_p} \;:\; A+\delta A \text{ 奇异}\right\} = \frac{1}{\kappa_p(A)},$$

表示 A 距离奇异矩阵集合的相对距离.

3.3 δx 与残量的关系

记残量 (残差) 为 $r = b - A\hat{x}$, 则有

$$\delta x = \hat{x} - x_* = \hat{x} - A^{-1}b = A^{-1}(A\hat{x} - b) = -A^{-1}r,$$

所以可得

$$\boxed{\|\delta x\| \leq \|A^{-1}\| \cdot \|r\|}$$

实际计算中 r 是可以计算的, 因此比较实用.

3.4 相对扰动分析

前面给出的误差 δx 与条件数 $\kappa(A)$, δA 和 δb 成比例.

许多情况下, 这个界是令人满意的.

但有时相差很大, 不能很好的反映实际计算中解的误差.

例 设 $A = \begin{bmatrix} \gamma & 0 \\ 0 & 1 \end{bmatrix}$, $b = \begin{bmatrix} \gamma \\ 1 \end{bmatrix}$, 其中 $\gamma > 1$. 则 $Ax = b$ 的精确解为 $x_* = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, 任何合理的直接法求得的解的误差都很小. 但系数矩阵的谱条件数为 $\kappa_2(A) = \gamma$, 当 γ 很大时, $\kappa_2(A)$ 也很大, 因此前面定理中的上界可以是很大.

针对这个问题, 我们按分量进行分析. 记

$$\delta A = \begin{bmatrix} \delta a_{11} & \\ & \delta a_{22} \end{bmatrix}, \quad \delta b = \begin{bmatrix} \delta b_1 \\ \delta b_2 \end{bmatrix},$$

并设 $|\delta a_{ij}| \le \varepsilon |a_{ij}|, |\delta b_i| \le \varepsilon |b_i|$. 则

$$\delta x = \begin{bmatrix} \hat{x}_1 - x_1 \\ \hat{x}_2 - x_2 \end{bmatrix} = \begin{bmatrix} \dfrac{\delta b_1 + b_1}{\delta a_{11} + a_{11}} - 1 \\ \dfrac{\delta b_2 + b_2}{\delta a_{22} + a_{22}} - 1 \end{bmatrix} = \begin{bmatrix} \dfrac{\delta b_1 + \gamma}{\delta a_{11} + \gamma} - 1 \\ \dfrac{\delta b_2 + 1}{\delta a_{22} + 1} - 1 \end{bmatrix} = \begin{bmatrix} \dfrac{\delta b_1 - \delta a_{11}}{\delta a_{11} + \gamma} \\ \dfrac{\delta b_2 - \delta a_{22}}{\delta a_{22} + 1} \end{bmatrix}$$

故

$$\|\delta x\|_\infty \le \frac{2\varepsilon}{1-\varepsilon}.$$

如果 $\delta b = 0$, 则

$$\|\delta x\|_\infty \le \frac{\varepsilon}{1-\varepsilon}.$$

这个界前面定理中的上界相差约 γ 倍.

相对条件数

为了得到更好误差界, 我们引入**相对条件数** $\kappa_{cr}(A)$, 即

$$\kappa_{cr}(A) \triangleq \| \, |A^{-1}| \cdot |A| \, \| \, ,$$

有时也称为 Bauer 条件数或 Skeel 条件数.

假定 δA 和 δb 满足 $|\delta A| \leq \varepsilon |A|$ 和 $|\delta b| \leq \varepsilon |b|$. 由 $(A + \delta A)\hat{x} = b + \delta b$ 可得

$$\begin{aligned}
|\delta x| &= |A^{-1}(-\delta A \hat{x} + \delta b)| \\
&\leq |A^{-1}| \cdot (|\delta A| \cdot |\hat{x}| + |\delta b|) \\
&\leq |A^{-1}| \cdot (\varepsilon |A| \cdot |\hat{x}| + \varepsilon |b|) \\
&= \varepsilon |A^{-1}| \cdot (|A| \cdot |\hat{x}| + |b|).
\end{aligned}$$

若 $\delta b = 0$, 则有

$$\|\delta x\| = \| \, |\delta x| \, \| \leq \varepsilon \| \, |A^{-1}| \cdot |A| \cdot |\hat{x}| \, \| \leq \varepsilon \| \, |A^{-1}| \cdot |A| \, \| \cdot \|\hat{x}\|,$$

即

$$\frac{\|\delta x\|}{\|\hat{x}\|} \leq \| \, |A^{-1}| \cdot |A| \, \| \cdot \varepsilon = \kappa_{cr}(A) \cdot \varepsilon$$

相对条件数有下面的性质

引理 设 $A \in \mathbb{R}^{n \times n}$ 非奇异, $D \in \mathbb{R}^{n \times n}$ 为非奇异对角矩阵, 则

$$\kappa_{cr}(DA) = \kappa_{cr}(A).$$

定理 设 $A \in \mathbb{R}^{n\times n}$ 非奇异. 使得 $|\delta A| \le \varepsilon |A|$, $|\delta b| \le \varepsilon |b|$ 成立, 且满足

$$(A + \delta A)\hat{x} = b + \delta b$$

的最小的 $\varepsilon > 0$ 称为按分量的相对向后误差, 其表达式为

$$\varepsilon = \max_{1\le i\le n} \frac{|r_i|}{(|A| \cdot |\hat{x}| + |b|)_i},$$

其中 $r = b - A\hat{x}$.

更多数值计算稳定性和矩阵扰动分析方面的知识, 可参考相关资料.

4 误差分析

4.1 LU 分解的舍入误差分析

关于 LU 分解的舍入误差分析, 我们有下面的结果.

定理 假定 $A \in \mathbb{R}^{n\times n}$ 的所有顺序主子式都不为 0, 则带舍入误差的 LU 分解可表示为

$$A = LU + E,$$

其中误差 E 满足

$$|E| \leq \gamma_n |L| \cdot |U|.$$

这里 $\gamma_n = \dfrac{n\varepsilon_u}{1 - n\varepsilon_u}$, ε_u 表示机器精度.

4.2 Gauss 消去法的舍入误差分析

引理 设 $\hat{y}$ 和 $\hat{x}$ 分别是由向前回代和向后回代得到的数值解, 则

$$
\begin{aligned}
(L+\delta L)\hat{y} &= b, \quad |\delta L| \leq \gamma_n |L| \\
(U+\delta U)\hat{x} &= \hat{y}, \quad |\delta U| \leq \gamma_n |U|.
\end{aligned}
$$

该引理表明, 向前回代算法和向后回代算法都是稳定的.

- 在绝大多数情况下, 部分选主元 Gauss 消去法是向后稳定的, 但理论上也存在失败的例子.
- 全主元 Gauss 消去法是数值稳定的. 在大部分实际应用中, 部分选主元 Gauss 消去法与全主元 Gauss 消去法具有同样的数值稳定性.

5 解的改进

5.1 高精度运算

在计算中, 尽可能采用高精度的运算.

比如, 原始数据是单精度的, 但在计算时都采用双精度运算, 或者更高精度的运算. 但更高精度的运算会带来更大的开销.

5.2 矩阵元素缩放 (Scaling)

如果 A 的元素在数量级上相差很大, 则在计算过程中很可能会出现大数与小数的加减运算, 这样就可能会引入更多的舍入误差.

为了避免由于这种情况而导致的舍入误差, 我们可以在求解之前先对矩阵元素进行缩放 (Scaling), 即在矩阵两边同时乘以两个适当的对角矩阵.

✍ 缩放技术在工程计算时经常会用到.

例 考虑线性方程组

$$\begin{bmatrix} -4000 & 2000 & 2000 \\ 2000 & 0.78125 & 0 \\ 2000 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 400 \\ 1.3816 \\ 1.9273 \end{bmatrix}.$$

用部分选主元 Gauss 法求解, 计算过程保留 8 位有效数字, 求得数值解

$$\tilde{x} = [0.00096365, -0.698496, 0.90042329]^{\mathsf{T}}.$$

与精确解 $x = [1.9273\cdots, -0.698496\cdots, 0.9004233\cdots]^{\mathsf{T}}$ 相差较大.

考虑矩阵元素缩放, 即同乘对角阵 $D = \mathrm{diag}(0.00005, 1, 1)$, 得新方程组

$$DADy = Db.$$

最后令 $\tilde{x} = Dy$, 即可求得比较精确的数值解.

思考: 试比较 A 和 DAD 的条件数.

5.3 迭代改进法

设近似解 $\hat{x}$, 如果 $\hat{x}$ 没达到精度要求, 可以考虑对其进行修正

$$\tilde{x} = \hat{x} + z,$$

即加上一个向量 z, 得到新的近似解 $\tilde{x}$.

怎么选取 z

为了使得 $\tilde{x}$ 尽可能地接近真解，我们希望 $\tilde{x}$ 能满足 $A\tilde{x} = b$，即

$$\boxed{A(\hat{x} + z) = b \Longleftrightarrow Az = r} \quad \to \text{残量方程}$$

其中 $\boxed{r = b - A\hat{x}}$ 是残量.

在实际计算中, 我们只能得到近似解 $\hat{z}$, 但 $\|r - A\hat{z}\|$ 会比较小, 特别地, 应该比 $\|r\|$ 更小. **因此 $\hat{x} + \hat{z}$ 应该比 $\hat{x}$ 更接近精确解.**

如果新的近似解 $\tilde{x} \triangleq \hat{x} + \hat{z}$ 还不满足精度要求, 则可重复以上过程.

这就是通过迭代来提高解的精度.

算法 5.1 通过迭代改进解的精度

1: 设 $PA = LU$, $\hat{x}$ 是 $Ax = b$ 的近似解
2: **while** 近似解 $\hat{x}$ 不满足精度要求, **do**
3: 计算 $r = b - A\hat{x}$
4: 求解 $Ly = Pr$, 即 $y = L^{-1}Pr$
5: 求解 $Uz = y$, 即 $z = U^{-1}y$
6: 令 $\hat{x} = \hat{x} + z$
7: **end while**

由于每次迭代只需计算一次残量和求解两个三角线性方程组, 因此运算量为 $O(n^2)$. 所以相对来讲还是比较经济的.

✍ 为提高计算精度, 在计算残量 r 时最好使用原始数据 A, 因此对 A 做 LU 分解时需要保留矩阵 A, 不能被 L 和 U 覆盖.

✍ 实际计算经验表明, 当 A 病态不是很严重时, 即 $\varepsilon_u \kappa_\infty(A) < 1$, 迭代法可以有效改进解的精度, 最后达到机器精度. 这里 ε_u 表示机器精度.

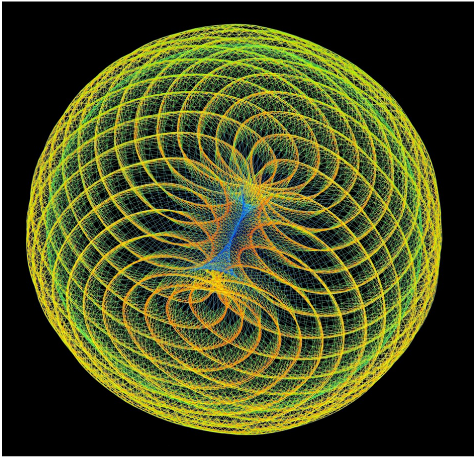
但 $\varepsilon_u \kappa_\infty(A) \geq 1$ 时, 效果可能不佳，得不到任何改善.

大规模稀疏线性方程组的直接解法

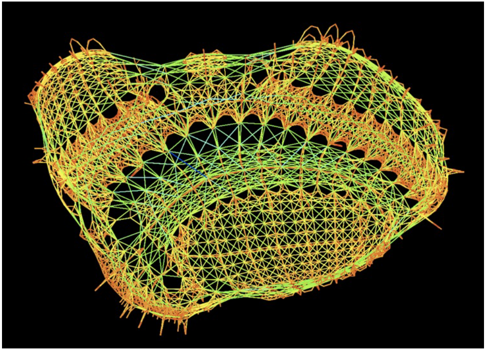
存在的困难

- 规模大, 百万量级以上或以亿为单位.
- LU/Cholesky 分解无法保持充分的稀疏性, 可能出现大量非零元, 导致存在困难, 即存储爆炸, 同时也无法有效降低运算量.

Example sparse matrices from THE SUITESPARSE MATRIX COLLECTION



A Barrier Hessian Matrix from Convex Quadratic Programming



The structural loading of an exhibition hall in Beijing

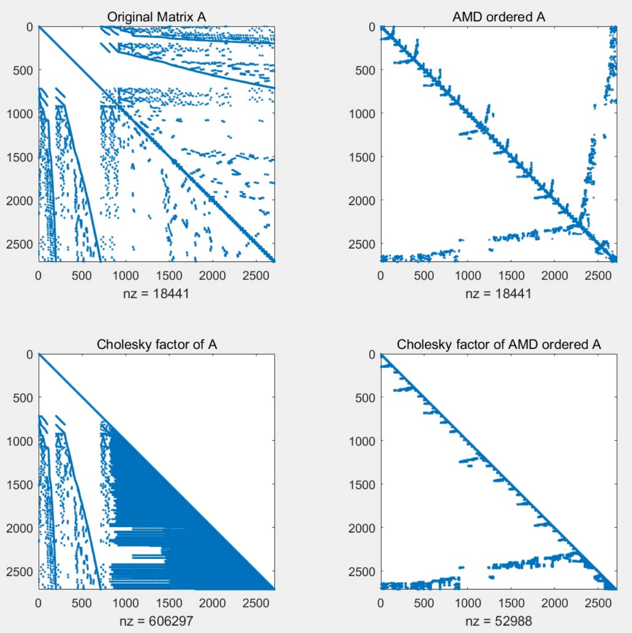
目前的策略

(1) 对矩阵进行重排序, 使得新矩阵的 LU/Cholesky 分解具有更好的稀疏性.

(2) 利用稀疏矩阵的重复结构: Supernodal, Frontal, Multifrontal methods

(3) 并行计算, GPU 加速, 低秩逼近, update/downdate methods

Davis, Rajamanickam and Sid-Lakhdar, A survey of direct methods for sparse linear systems, *Acta Numerica*, 2016.


Original Matrix A
nz = 18441
AMD ordered A
nz = 18441
Cholesky factor of A
nz = 606297
Cholesky factor of AMD ordered A
nz = 52988