数值线性代数———

矩阵计算

潘建瑜

线性方程组直接解法

线性方程组的求解方法

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

本章介绍直接法, 即Gauss 消去法 或PLU 分解

直接法优点: 稳定可靠 $\rightarrow$ 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)

考虑线性方程组

$$Ax = b \tag{1}$$

其中 $A\mathbb{R}^{nn}$ 非奇异, $b\mathbb{R}^{n}$ 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:

$$A = LU \tag{2}$$

其中 L 是单位下三角矩阵, U 为非奇异上三角矩阵.
分解 () 就称为 LU 分解

$$Ax = b \Longleftrightarrow \left\{ \begin{array}{l} Ly = b \\ Ux = y \end{array} \right| \Longrightarrow \text{只需求解两个三角方程组} \tag{3}$$

算法 1.1:	Gauss 消去法
1：	将 A 进行 LU 分解:
2：	$A = LU$, 其中 L 为单位下三角矩阵, U 为非奇异上三角矩阵;
3：	向前回代: 求解 $Ly = b$, 即得 $y = L^{-1}b$
4：	向后回代: 求解 $Ux = y$, 即得 $x = U^{-1}y = (LU)^{-1}b = A^{-1}b$.

需要指出的是: 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 $A\mathbb{R}^{nn}$. 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, $k = 1, 2, ..., n$.

给定一个矩阵

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

- 第一步: 假定 $a_{11} \neq 0$, 构造矩阵

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

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

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

即左乘 L_1^1 后, A 的第一列中除第一个元素外其它都变为 0.

- 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

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

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

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

1.2 LU 分解的实现—矩阵初等变换

- 依此类推, 假定 $a_{kk}^{(k-1)} \neq 0 (k = 3, 4, \ldots, n-1)$, 则我们可以构造一系列的矩阵 $L_3, L_4, \ldots, L_{n1}$, 使得

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

于是可得 $A = LU$ 其中

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

1.2 LU 分解的实现—矩阵初等变换

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

  end 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

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

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

Gauss 消去法的运算量

† 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.
同样地, 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).

矩阵 L 和 U 的存储

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
```

† 根据指标的循环次序, 算法 4.3 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. 对于按行存储的数据结构, 一般采用后面介绍的 IKJ 型 LU 分解.

矩阵 L 和 U 的存储

Matlab code 1: 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!’ , 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
```

矩阵 L 和 U 的存储

为了充分利用 Matlab 的向量运算优势, 提高运算效率, 程序可改写为

Matlab code 2: LU 分解 (KIJ 型)

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

1.3 IKJ 型 LU 分解

如果数据是按行存储的, 如 C/C++, 我们一般采用下面的 IKJ 型 LU 分解.

算法 1.4: LU 分解

```
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.

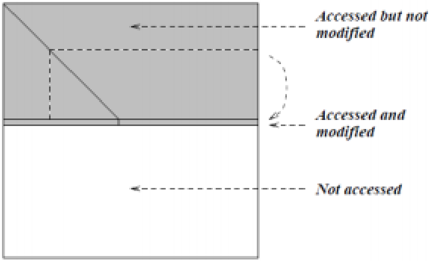

Figure:

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

设 $A = LU$, 即

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & & & \ddots & \vdots \\ a_{n1} & a_{x2} & a_{x3} & \cdots & a_{nn} \end{bmatrix} =$$

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

(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

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

$u_{kj} = a_{kj} - \left(l_{k1}u_{1j} + \cdots + l_{k,k-1}u_{k-1,j}\right), \quad j = k, k+1, \ldots, n$

比较等式两边的第 k 列, 可得

$l_{ik} = \left(a_{ik} - l_{i1}u_{1k} - \cdots - l_{i,k-1}u_{k-1,k}\right)/u_{kk}, \quad i = k+1, k+2, \ldots, n$

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)
1：	for k = 1 to n do
2：	$a_{kj} = a_{kj} - \sum_{i=1}^{k-1} a_{ki}a_{ij}, \quad j = k, k+1, \ldots, n$
3：	$a_{ik} = \frac{1}{a_{kk}} \left(a_{ik} - \sum_{j=1}^{k-1} a_{ij}a_{jk} \right), \quad i = k+1, k+2, \ldots, n$
4：	end for

1.4 待定系数法计算 LU 分解

Matlab code 3: 待定系数法 LU 分解

```
function A = mylu(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!’, 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
```

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)

1: $y_1 = b_1/l_{11}$
2: for i = 2 : n do
3:     for j = 1 : i - 1 do
4:         $b_i = b_i - l_{ij}y_j$
5:     end for
6:     $y_i = b_i/l_{ii}$
7: 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 向前回代求解 $Ly = b$(假定 L 是一般的非奇异下三角矩阵)

for i = n : -1 : 1 do
  $x_i = y_i / u_{ii}$
  for j = i - 1 : -1 : 1 do
    $y_j = y_j - x_i u_{ji}$
  end for
end for

这两个算法的运算量均为 $n^2 + O(n)$

- 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 $\left|a_{kk}^{(k-1)}\right|$ 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

$$A = \begin{bmatrix} 0.02 & 61.3 \\ 3.43 & -8.5 \end{bmatrix}, \quad b = \begin{bmatrix} 61.5 \\ 25.8 \end{bmatrix} \tag{4}$$

要求在运算过程中保留 3 位有效数字.

$$(x_1 20.7, x_2 1.01)$$

易知, 方程的精确解为 $x_1 = 10.0$ 和 $x_2 = 1.00$. 我们发现 x_1 的误差非常大. 导致这个问题的原因就是 $|a_{11}|$ 太小, 用它做主元时会放大舍入误差. 所以我们需要选主元.

选主元 LU 分解

定理

设 $A R^{nn}$ 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.

选法一. 选取 $P_L^{(k)}$ 和 $P_R^{(k)}$ 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. 选取 $P_L^{(k)}$ 和 $P_R^{(k)}$ 使得主元为第 k 列中第 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.

†(1) GECP 比 GEPP 更稳定, 但工作量太大, 在实际应用中通常使用 GEPP 算法.

(2) GEPP 算法能保证 L 所有的元素的绝对值都不超过 1.

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
    $[a_{max}, l] = \max_{k\ \mathrm{in}} |a_{ik}|$ % 选列主元，其中 l 表示主元所在的行
    if l k then
        for j =2 to n do
            $tmp = a_{kj}, a_{kj} = a_{lj}, a_{ij} = tmp$ % 交换第 k 行与第 l 行
        end for
        $tmp = p(k), p(k) = p(l), p(l) = tmp$ % 更新置换矩阵
    end for
    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
```