MIT 18.06 线性代数
第四讲：矩阵的 LU 分解

整理自 MIT 18.06 课程及相关公开笔记

引言

本讲的主要目的是从矩阵乘法的角度深入理解高斯消元法，进而找到一种将矩阵 A 分解为一个下三角矩阵 L 与一个上三角矩阵 U 的乘积形式，即 $A = LU$ 分解。这种分解不仅在理论分析上具有重要意义，在计算机数值计算中也非常高效。此外，本讲还将讨论消元过程的运算量，以及当主元位置出现 0 时，如何通过置换矩阵进行行交换。

1 逆矩阵与转置矩阵性质

1.1 矩阵乘积的逆

如果 A 和 B 均为可逆方阵，则矩阵乘积 AB 的逆矩阵为：

$$(AB)^{-1} = B^{-1}A^{-1}$$

验证：$(AB)(B^{-1}A^{-1}) = A(BB^{-1})A^{-1} = AIA^{-1} = AA^{-1} = I$。

1.2 转置矩阵的逆

矩阵的转置定义为将原矩阵的行变为列。对于可逆矩阵 A，转置与求逆的运算顺序可以互换：

$$(A^T)^{-1} = (A^{-1})^T$$

证明：由 $AA^{-1} = I$，两边同时取转置得到 $(AA^{-1})^T = (A^{-1})^T A^T = I$。根据逆矩阵的定义可知，$A^T$ 的逆矩阵正是 $(A^{-1})^T$。

2 A = LU 分解

2.1 从消元矩阵到 LU 分解

在无行交换的情况下，对矩阵 A 进行高斯消元得到上三角矩阵 U 的过程，可以用左乘一系列初等消元矩阵 E_{ij} 来表示。例如对于三阶矩阵：

$$E_{32}E_{31}E_{21}A = U$$

其中 E_{31} 若为单位阵 I，则可简化为 $E_{32}E_{21}A = U$。

若在等式两边同时左乘各消元矩阵的逆矩阵，则可得到 A 的分解形式：

$$A = E_{21}^{-1} E_{31}^{-1} E_{32}^{-1} U$$

将这一系列逆矩阵的乘积记为 L，即得到 $A = LU$ 分解。其中 U 是**上三角矩阵**（主元在对角线上），而 L 是**下三角矩阵**（对角线元素均为 1）。

$$A = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ * & 1 & 0 & 0 \\ * & * & 1 & 0 \\ * & * & * & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \blacksquare & * & * & * & * \\ 0 & \blacksquare & * & * & * \\ 0 & 0 & 0 & \blacksquare & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{U}$$

FIGURE 1 An LU factorization.

2.2 L 矩阵的构造与优势

我们通过一个具体的三阶矩阵示例来说明 L 矩阵的构造：

已知消元矩阵为 $E_{21} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$，$E_{31} = I$，$E_{32} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -5 & 1 \end{bmatrix}$.

首先计算总消元矩阵 $E = E_{32} E_{21}$：

$$E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -5 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 10 & -5 & 1 \end{bmatrix}$$

注意 E 左下角出现了元素 10，这是因为在第二步消元时，第三行减去了 5 倍的第二行，而此时的第二行已经是被第一步消元修改过的新的第二行（原第二行 -2 倍的第一行）。因此第三行最终受到的影响是 $-5 \times (-2) = 10$ 倍第一行的影响，产生了交叉项。

然而，我们再来计算 $L = E^{-1} = E_{21}^{-1} E_{32}^{-1}$：

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 5 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 5 & 1 \end{bmatrix}$$

可以看到，在矩阵 L 中，元素直接被填入了消元乘数（2 和 5），没有出现任何复杂的交叉项。

结论：$A = LU$ 分解优于 $EA = U$ 的形式，因为 L 直接记录消元乘数，不需要额外的存储和计算，这对计算机进行大规模数值计算极其有利。

2.3 LDU 分解

有时我们还会将 U 进一步分解，将主元单独提取出来，形成一个对角矩阵 D（Diagonal matrix），即 $A = LDU$。例如：$A = \begin{bmatrix} 2 & 1 \\ 8 & 7 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1/2 \\ 0 & 1 \end{bmatrix} = LDU$。其中 D 包含了消元后的主元（2 和 3），L 和 U 的对角线均为 1，这种形式更加规范对称。

3 消元法的运算量

当处理 $n \times n$ 的大型矩阵时，我们需要评估计算机进行消元所需的运算次数（将一行的倍数加到另一行定义为一次运算）。

- **第一列消元**：需要对第一列的 n 个元素进行操作，涉及 n 行，大约需要 n^2 次运算。
- **剩余子矩阵**：完成第一列后，左下角余下的矩阵大小为 $(n-1)\times(n-1)$，其消元需要约 $(n-1)^2$ 次运算。
- **总运算量**：依次类推，总运算次数约为：

$$1^2 + 2^2 + \cdots + (n-1)^2 + n^2 = \sum_{i=1}^{n} i^2 \approx \int_0^n x^2 dx = \frac{1}{3}n^3$$

（利用黎曼和与积分近似估算）。

这意味着矩阵规模 n 增加一倍，消元计算量将增加约八倍（$\frac{1}{3}n^3$ 级别）。而对右侧常数向量 $\mathbf{b}$ 进行同样变换的运算量仅为 n^2 级别，相比之下可以忽略不计。

4 置换矩阵与行交换

4.1 置换矩阵的定义与数量

在消元过程中，若主元位置恰好出现 0，我们需要进行 “行交换” 来寻找非零主元，这种行交换操作可以通过左乘置换矩阵 P 来实现。置换矩阵是从单位阵 I 经过行交换得到的，其每一行和每一列都只有一个元素为 1，其余全为 0。

对于一个 $n \times n$ 的矩阵，其行排列共有 $n!$ 种方式，因此共有 $n!$ 个不同的置换矩阵。例如，三阶矩阵共有 $3! = 6$ 个置换矩阵，它们构成了一个乘法群：任意两个置换矩阵的乘积仍在集合中，且每个置换矩阵都有唯一的逆矩阵。

4.2 置换矩阵的性质与 PA=LU

置换矩阵最重要的一条性质是：置换矩阵的逆矩阵等于它的转置矩阵，即：

$$P^{-1} = P^T$$

当我们引入行交换后，LU 分解的形式需要稍作调整。为了便于后续求解线性方程组 $Ax = b$，我们可以先将 A 的行顺序进行调整，使得消元过程中永远不会出现主元为 0 的情况，这等价于在分解前左乘一个置换矩阵 P：

$$PA = LU$$

4.3 LU 分解的 Python 与 MATLAB 实现

在 4.2 节中我们引入了置换矩阵 P，并指出当消元过程需要行交换时，分解形式写为 $PA = LU$。本节给出两种常用科学计算环境下的具体实现，以便将理论算法直接应用于实际数值问题。

MATLAB 中的 `lu`

MATLAB 的 `lu` 函数支持满矩阵与稀疏矩阵，常用调用形式为：

```
[L, U] = lu(A)              % A = L*U
[L, U, P] = lu(A)           % 返回置换矩阵 P，满足 P*A = L*U
[L, U, P] = lu(A, 'vector') % 返回置换向量 P，满足 A(P,:) = L*U
[L,U,P,Q] = lu(S)           % P*S*Q = L*U
```

当只返回两个输出时，将满矩阵或稀疏矩阵 A 分解为一个上三角矩阵 U 和一个经过置换的下三角矩阵 L，使得 A = LU。当返回三个输出时，还返回一个置换矩阵 P，并满足 A = P'LU，其中 L 是单位下三角矩阵，U 是上三角矩阵。还可以将稀疏矩阵 S 分解为一个单位下三角矩阵 L、一个上三角矩阵 U、一个行置换矩阵 P 以及一个列置换矩阵 Q，并满足 PSQ = LU。

Python（SciPy）中的 `scipy.linalg.lu`

SciPy 的 `lu` 函数默认返回置换矩阵 P、下三角矩阵 L 和上三角矩阵 U，满足 $A = PLU$。可通过参数 `permute_l=True` 直接返回已置换的 L，使分解满足 $A = LU$；也可通过 `p_indices=True` 获取置换向量的形式。

```
P, L, U = lu(A)                     # 返回置换矩阵
PL, U = lu(A, permute_l=True)       # 返回置换后的 L
p, L, U = lu(A, p_indices=True)     # 返回置换向量
```

示例：考虑矩阵 $A = \begin{bmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{bmatrix}$，分别用 MATLAB 与 Python 计算 LU 分解（带行置换）。

MATLAB 代码与输出：

```
>> A = [10 -7 0; -3 2 6; 5 -1 5];
>> [L, U, P] = lu(A)

L =
    1.0000         0         0
    0.5000    1.0000         0
   -0.3000   -0.0400    1.0000

U =
   10.0000   -7.0000         0
         0    2.5000    5.0000
         0         0    6.2000

P =
     1     0     0
     0     0     1
     0     1     0
```

```
>> P'*L*U   % 验证 PA = LU 等价于 A = P'LU
ans =
    10    -7     0
    -3     2     6
     5    -1     5
```

Python 代码与输出：

```
>>> import numpy as np
>>> from scipy.linalg import lu
>>> A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
>>> P, L, U = lu(A)
>>> print(P)
[[1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]]
>>> print(L)
[[ 1.   0.   0. ]
 [ 0.5  1.   0. ]
 [-0.3 -0.04 1. ]]
>>> print(U)
[[10.  -7.   0. ]
 [ 0.   2.5  5. ]
 [ 0.   0.   6.2]]
>>> np.allclose(A, P @ L @ U) # 验证理论分解的正确性（实际计算中有数值误差）
True
```

两种环境返回的 P、L、U 结果一致，行交换发生在第 2、3 行。使用置换向量可节省内存，尤其在大规模稀疏矩阵时优势明显，例如用 `[L,U,p] = lu(A,'vector')` 或 `p, L, U = lu(A, p_indices=True)` 重新组合。

总结

- 矩阵的逆与转置满足：$(AB)^{-1} = B^{-1}A^{-1}$ 及 $(A^T)^{-1} = (A^{-1})^T$。
- $A = LU$ 分解是将矩阵 A 分解为一个下三角矩阵 L（消元乘数）和一个上三角矩阵 U（消元后的结果）。相比 $EA = U$，L 矩阵能直接记录消元乘数，避免了交叉项，在数值计算上更优。
- 高斯消元的运算量约为 $\frac{1}{3}n^3$ 次，而右侧向量的处理约为 n^2 次。
- 当遇到主元为 0 的情况时，使用置换矩阵 P 进行行交换，此时通用的分解形式为 $PA = LU$。