

线性系统和特征问题的迭代 ILU 预处理器

摘要：迭代的 ILU 分解被构造、分析，被视为求解线性系统和特征值问题的预处理器。它的计算核心是稀疏矩阵的乘法，在串行和并行结构上可简单且高效的实施。我们也介绍了一种高层次和阈值的算法来提高提出的迭代的 ILU 分解算法的准确率，并基于数值实验给出了证明。

一、引言

LU 分解法是求解线性系统的一类直接方法。对于稀疏问题，L 和 U 这两个分解因子可能会提高或降低原先矩阵的稀疏程度。对于大规模问题，LU 直接法产生的因子同样需要花费时间来计算和花费空间来储存。在这种情况下，迭代方法就体现出了他们的优势，因为它们会需要更少的空间内存和浮点计算量。但是，迭代方法也有缺乏鲁棒性，收敛速度慢的缺点，所以为了使迭代方法更快且更稳健，预处理技术是必要的。很多预处理技术是针对特定的问题提出的，因为不完全的 LU 分解的基础是 LU 分解，所以是用 ILU 作为预处理器是相对普遍的。

应用很广泛。大部分的迭代方法的收敛速度取决于矩阵的条件数。

矩阵 A 的条件数： $\kappa(A) = \|A\| \cdot \|A^{-1}\|$ ， $\kappa_2(A) = \frac{\lambda_{\max}}{\lambda_{\min}}$ ，其中 λ 是矩阵 A 的特征值。

(A 为对称正定阵) 最速下降法的收敛性： $\|x^k - x\|_A \leq \left(\frac{\kappa_2(A)-1}{\kappa_2(A)+1}\right)^k \|x^0 - x\|_A$

(A 为对称正定阵) CG 方法的收敛性： $\|x^k - x\|_A \leq 2\left(\frac{\sqrt{\kappa_2(A)}-1}{\sqrt{\kappa_2(A)}+1}\right)^k \|x^0 - x\|_A$

(A, P 为对称正定阵) PCG 方法的收敛性：

$$\|x^k - x\|_A \leq 2\left(\frac{\sqrt{\kappa_2(P^{-1}A)}-1}{\sqrt{\kappa_2(P^{-1}A)}+1}\right)^k \|x^0 - x\|_A$$

在[7,29,34]中已经证明了不完全的因子分解预处理器可以减少有限差分离散化的条件数，对于二维椭圆问题，从 $O(n^2)$ 降低到 $O(n)$ 。基于不同技术，这里有很多标准的 ILU 分解的修正。例如：高层次的 ILU(p)通过允许更多的填充提高了分解的准确性，弥补对角线元素使得 lu 的乘积的行或列的和等于原始矩阵的行或列的和，被称为修正的 ILU(MILU)[19,26]。阈值的 ILU(ILUT(p, τ)) 在删

除规则中引入了容忍限度和填充数,这种方法提供了更多的选项来平衡因子的准确性和存储内存。[3,15,32]有很多关于 ILU 的稳定性和实现的文献。

高斯消去法是不完全的 LU 分解的核心,是一种具有高依赖的程序,它是 ILU 的并行化的一个障碍。因此,很多其他的技术已经被研究来提高 LU 分解的并行化。图染色方法和域分解方法将原来的大矩阵分解为很多的子矩阵,保证分解能在子矩阵上执行[20,21,28]。在[12],基于残差修正的分解方法可以并行计算。最近,一种基于不动点迭代的详细的并行不完全分解已经被提出,而且他已经在共享的内存环境中被成功执行[2,13]。

在这篇论文中,我们构造和分析一种给定的矩阵的迭代 ILU 分解。这个新算法的主要程序基础是稀疏矩阵的乘法。

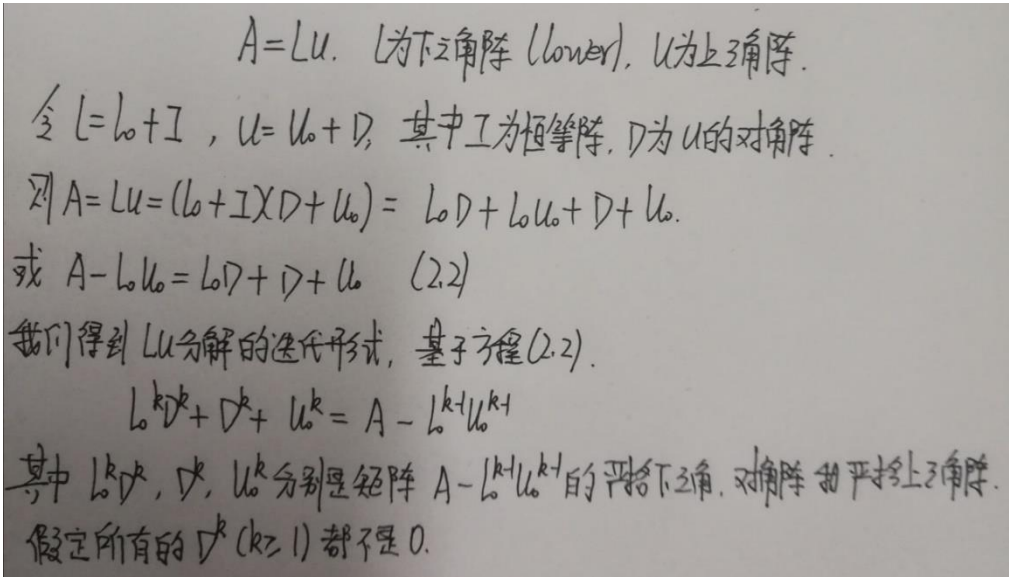
这篇论文不再涉及到新算法的并行性,因为这个算法仅会涉及到矩阵和矩阵乘法以及矩阵和向量的乘法。这个已经被证实和使用过了。

论文安排如下:在第 2 部分,推导矩阵形式的传统的 LU 分解并且回忆它的填充规则。在第 3 部分提出几个新的 ILU 迭代分解并且证明他们的收敛结果。第 4 部分讨论一些新的算法中涉及到的问题,例如矩阵和矩阵的乘法以及三角系统的求解。在 5,6 部分,我们利用这种新的方法来分别求解线性系统和特征值问题,展示几个数值实验的结果。在 7 部分是一些结论和限制(缺点)。

二、LU 分解的迭代法

令 A 是一个能被分解为下三角和上三角的方阵,所有的 L 的对角元素都是 1,我们将三角矩阵 L, U 写成如下形式:

I 是恒等矩阵, D 是 U 的对角阵,重新写 A 的分裂形式:



Handwritten mathematical derivation of the iterative ILU decomposition:

$$A = LU. \quad L \text{ 为下三角阵 (lower), } U \text{ 为上三角阵.}$$
$$\text{令 } L = L_0 + I, \quad U = U_0 + D, \quad \text{其中 } I \text{ 为恒等阵, } D \text{ 为 } U \text{ 的对角阵.}$$
$$\text{则 } A = LU = (L_0 + I)(U_0 + D) = L_0 D + L_0 U_0 + D + U_0.$$
$$\text{或 } A - L_0 U_0 = L_0 D + D + U_0 \quad (2.2)$$

我们得到 LU 分解的迭代形式, 基于方程 (2.2).

$$L_0^{k+1} D^k + D^k + U_0^k = A - L_0^{k+1} U_0^{k+1}$$

其中 L_0^k, D^k, U_0^k 分别是矩阵 $A - L_0^{k+1} U_0^{k+1}$ 的严格下三角, 对角阵和严格上三角阵.

假定所有的 $D^k (k \geq 1)$ 都不是 0.

算法 1 的伪代码如下：

Algorithm 1 Iterative LU factorization

```

1: Set the initial data  $L_0 = D = U_0 = 0$ .
2: for  $k = 1, 2, \dots, p$  do
3:    $B = A - L_0 U_0$ 
4:    $D = \text{diag}(\text{diag}(B))$ 
5:    $U_0 = \text{triu}(B, 1)$ 
6:    $L_0 = \text{tril}(B, -1)D^{-1}$ 
7: end for
8:  $L = L_0 + I$  and  $U = U_0 + D$ .

```

算法的实现：

<pre> function [L,U] = IterLU1(A,p) %LU分解迭代法 n=size(A,1); %矩阵A一定是一个方阵 L=zeros(n); U=zeros(n); D=zeros(n,1); for k=1:p B=A-L0*U0; D=diag(B); U0=triu(B,1); L0=tril(B,-1); L0=L0*diag(D,`(-1)`); end L=L0+speye(n); U=U0+diag(D); end </pre>	<pre> L = 1.0000000000000000 0 0 5.0000000000000000 1.0000000000000000 0 1.5000000000000000 -0.1250000000000000 1.0000000000000000 U = 2.0000000000000000 3.0000000000000000 2.0000000000000000 0 -12.0000000000000000 -6.0000000000000000 0 0 -2.7500000000000000 </pre>
--	---

定理 2.1：在算法 1 中的 L,U 至多 n 步收敛到 A 的准确 LU 分解。（已经验证过当 p 大于等于 3 时，L，U 的值是不发生变化的）

2.1 元素的填充

记 S_A 是索引对 (i, j) 对应于稀疏矩阵的稀疏模式的集合。定义了两个矩阵 L 和 U 的和与内积的集合的表达形式。

L 和 U 两个矩阵的非零元的个数随着迭代步数 k 的增加而增加，因为它在算法 1 的第三步中涉及到了矩阵的乘法。我们称相比于原来的矩阵的因子中新增加的非零元为 A 的填充。为了研究因子的填充的性质，在求解稀疏矩阵的直接解方法中，我们介绍填充路和填充层级的概念。将一个矩阵看作一个有向图，两个顶点 i, j 的一个 fill-path 是一个通路，例如在这条通路的所有的顶点，除了终点 i 和 j ，被编号少于 i 和 j 的编号数。如果 i 和 j 之间的最短路径是 $p+1$ ，位置 (i, j) 的填充层级就是 p （当 ILU 分解完全进行完毕后，如果 (i, j) 位置的填充层级为 p ，当且仅当从顶点 i 到顶点 j 之间存在着长度为 $p+1$ 的一条填充路）。如果在稀疏矩阵模式下，所有位置的填充层级小于等于 p ，我们称这个模式为这个矩阵的 level- p 模式（保留填充因子小于 p 的元素，丢弃大于 p 的元素）。若一个矩阵的非零位置的填充层级是 0，则由所有的非零位置组成 L 和 U 的稀疏模式是这个矩阵的零元填充。（零填充分解法是在原矩阵零元位置上将分解得到的非

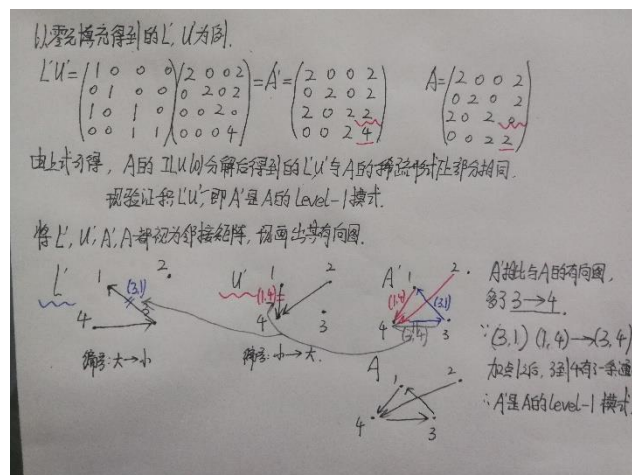
零元抛弃，分解后的 L 矩阵和 U 矩阵具有原来稀疏矩阵相同的分布结构。0 表示因子分解时产生的新的非零元素个数为零。)

$$A = \begin{bmatrix} 2 & 0 & 0 & 2 \\ 0 & 2 & 0 & 2 \\ 2 & 0 & 2 & 0 \\ 0 & 0 & 2 & 2 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & 0 & 0 & 2 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & -2 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

根据删除原则，得到

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 2 & 0 & 0 & 2 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 4 \end{bmatrix}$$

引理 2.2，若 L_0 和 U_0 是 A 的 ILU(0)模式分解的矩阵，则积 $L_0 U_0$ 的稀疏模式是 A 的 level-1 模式。



定理 2.3，当 k 大于等于 2 时， $L_0^k + D^k + U_0^k$ 的稀疏模式是 $L_0^{k-1} + D^{k-1} + U_0^{k-1}$ 的 level-1 模式。

3. 不完全的 LU 迭代

因为 step3 的矩阵和矩阵乘法，当迭代次数增加时算法的填充数快速增加，为了控制 L 和 U 因子的存储内存，在每次迭代中我们根据以下两个规则删除一些元素，设置最大迭代次数来停止迭代，这也是被称为不完全的 LU 分解原因。我们总结概括在算法二中的思想。在接下来的这部分，我们介绍两个删除策略，分别是基于模式和阈值。

3.1 基于模式

Algorithm 2 Iterative ILU factorization

- 1: Set the initial data $L_0 = D = U_0 = 0$.
 - 2: **for** $k = 1, 2, \dots, p$ **do**
 - 3: $B = L_0 U_0$
 - 4: $B = A - B$
 - 5: $D = \text{diag}(\text{diag}(B))$
 - 6: $U_0 = \text{triu}(B, 1)$
 - 7: $L_0 = \text{tril}(B, -1)D^{-1}$
 - 8: execute dropping strategies on L_0 and U_0 .
 - 9: **end for**
 - 10: $L = L_0 + I$ and $U = U_0 + D$.
-

在算法 3 中，我们首先设置一个稀疏矩阵 S ，然后在每一次迭代中仅删除 S 中除 L_s 和 U_s 之外的项。如果 S 包含 A 的稀疏形式，在第一次迭代之后。 $L_0 = A_{L_0} A_D^{-1}$ and $U_0 = A_{U_0}$ ， A_{L_0} ， A_D 和 A_{U_0} 是原始矩阵 A 的严格下三角矩阵、对角矩阵和严格上三角矩阵。相应的 ILU 预处理器 $LU = (A_{L_0} + A_D) A_D^{-1} (A_D + A_{U_0})$ 被称为 SSOR，这种方法已经很好地被应用于一些问题。

接下来，我们证明算法 3 的收敛性，我们用图 3.1 来阐述证明的过程。

Algorithm 3 Iterative ILU factorization

- 1: Set the initial data $L_0 = D = U_0 = 0$ and the sparsity pattern S .
 - 2: **for** $k = 1, 2, \dots, p$ **do**
 - 3: $B = L_0 U_0$
 - 4: $B = A - B$
 - 5: $D = \text{diag}(\text{diag}(B))$
 - 6: $U_0 = \text{triu}(B, 1)$
 - 7: $L_0 = \text{tril}(B, -1)D^{-1}$
 - 8: drop the entries of L_0 and U_0 out of the pattern S .
 - 9: **end for**
 - 10: $L = L_0 + I$ and $U = U_0 + D$.
-

引理 3.1 在算法 3 中，如果稀疏模式 s 的前 k 行和前 k 列的乘积 LU 等于

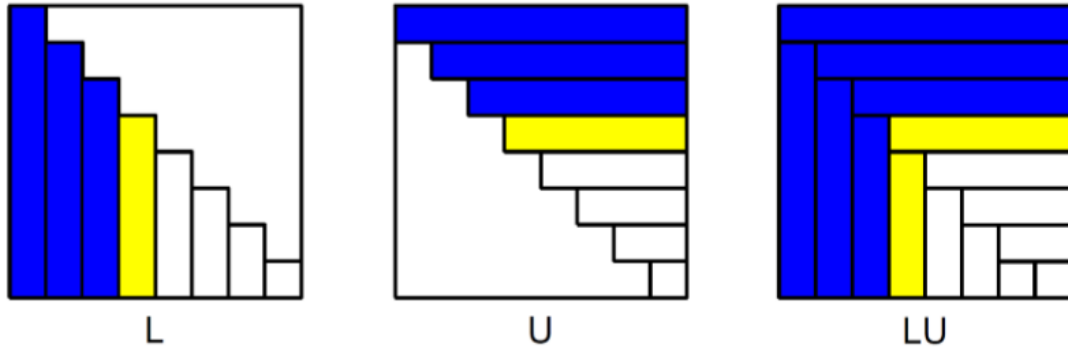


Fig. 3.1: The entries in the blue part of each matrix are exact values. In the next iteration, these exact entries will not change (Lemma 3.1), and the yellow rows and columns will become exact (Lemma 3.2).

A, 则在下一次迭代中, L 的前 k 列和 U 的前 k 行不会发生变化。

引理 3.2 在算法 3 中, 如果稀疏模式 s 的前 k 行和前 k 列的乘积 LU 等于 A, 则在下一次迭代中, 稀疏模式 s 的前 k+1 行和前 k+1 列乘积 LU 等于 A。

定理 3.3 算法 3 中的因子在 n 次迭代内收敛到稀疏模式 S 上的标准不完全因子。(理解: 在稀疏模式 $S(k, \tau)$ 下, 算法 3 的第 k 次迭代得到的 L_k 和 U_k 收敛于不完全的 $ILU(k, \tau)$ 分解得到的 L 和 U。)

当 S 包含了矩阵 A 的所有位置, 定理 2.1 是定理 3.3 的一种特殊例子,。

Level-0 模式在 ILU 分解中是最普遍的。为了包含更多准确的元素, 我们能扩大稀疏的模式来允许更多的填充。如果我们不删除任何元素, 使用算法 3 迭代几次, 通过定理 2.3 和引理 2.1 得到的因子是一个更大的稀疏模式。得到的因子和稀疏模式能被视作原始数据和一个固定的模式。我们在算法 4 中用了这个原则, 称它为 IterILU (p,m)。P 表示模式的尺寸, m 是迭代次数。算法 1 和 3 都能被看作 4 的特殊例子。

- $p = 1$ and $m = 0$, SSOR(1) preconditioner;
- $p = 1$ and $m \geq 1$, Algorithm 3 with m iterations on level-0 sparsity pattern;
- $p \geq 2$ and $m = 0$, Algorithm 1 with p iterations;
- $p \geq 2$ and $m \geq 1$, Algorithm 1 with p iterations, then on the sparsity pattern of the results, execute Algorithm 3 with m enhancement iterations.

通常, 随着 p 的增加空间也在增加。当固定了 p, 解三角系统的花费的时间和空间也应该被考虑在其中, 我们在 matlab 中写一个简短的程序, 粘贴在附录 A 中。

Algorithm 4 Iterative ILU factorization IterILU(p, m)

```

1: Set the initial data  $L_0 = D = U_0 = 0$ .
2: for  $k = 1, 2, \dots, p$  do
3:    $B = A - L_0 U_0$ 
4:    $D = \text{diag}(\text{diag}(B))$ 
5:    $U_0 = \text{triu}(B, 1)$ 
6:    $L_0 = \text{tril}(B, -1)D^{-1}$ 
7: end for
8: get the sparsity pattern  $S$  of  $B$ 
9: for  $k = 1, 2, \dots, m$  do
10:   $B = L_0 U_0$ 
11:   $B = A - B$ 
12:   $D = \text{diag}(\text{diag}(B))$ 
13:   $U_0 = \text{triu}(B, 1)$ 
14:   $L_0 = \text{tril}(B, -1)D^{-1}$ 
15:  drop the entries of  $L_0$  and  $U_0$  out of the pattern  $S$ .
16: end for
17:  $L = L_0 + I$  and  $U = U_0 + D$ .

```

3.2 基于阈值

我们提出另一个删除原则，仅仅基于 L_0 和 U_0 的每行或者每列的大小。

(1.) 在 L 的每行中找到最大的绝对值 T ，在相应的 L_0 的行中删除比 τT 值小的元素。

(2.) 在 U 的每列中找到最大的绝对值 T ，在相应的 U_0 的列中删除比 τT 值小的元素。

阈值 T 是为了 L_0 和 U_0 统一设置的。我们将算法 5 中的算法称为 $\text{IterILU}(T, p)$ ，其中 p 是迭代次数。

Algorithm 5 Iterative threshold ILU factorization $\text{IterILUT}(\tau, p)$

```
1: Set the initial data  $L_0 = D = U_0 = 0$ .
2: for  $k = 1, 2, \dots, p$  do
3:    $B = L_0 U_0$ 
4:    $B = A - B$ 
5:    $D = \text{diag}(\text{diag}(B))$ 
6:    $U_0 = \text{triu}(B, 1)$ 
7:    $L_0 = \text{tril}(B, -1)D^{-1}$ 
8:   execute dropping strategies on  $L_0$  and  $U_0$  according to the threshold  $\tau$ .
9: end for
10:  $L = L_0 + I$  and  $U = U_0 + D$ .
```

基于阈值的 ILU 应用没有基于模式的应用广泛。因为当矩阵元素的值集中于一个值的周围，这个方法是不合适的。而且值 T 的选择是一个难点，选的太大，矩阵的重要部分可能被删除，选的太小，可能在减少空间存储方面是没有意义的。而且这个阈值应该根据算法的想要达到准确性和因素的空间存储来设计。当然了，基于位置、数字和元素的大小的删除策略可以被混合使用。

4 在新的算法中出现的一些其他的问题

4.1 矩阵和矩阵的乘法

实现这个操作的一个风险是，即使两个矩阵是非常稀疏的，他们的乘积可能会包含很多的非零元素。在极端情况下，当 L_0 的第一列和 U_0 的第一行是满向量时， L_0 和 U_0 的乘积会产生一个满矩阵，见下图：

$$\begin{pmatrix} l_{21} & 1 & & & \\ l_{31} & 0 & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ 0 & 0 & \dots & 0 & \\ 0 & \dots & 0 & \vdots & \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & \dots & 0 \end{pmatrix} = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ l_{21}u_{11} & l_{21}u_{12} & l_{21}u_{13} & \dots & l_{21}u_{1n} \\ l_{31}u_{11} & l_{31}u_{12} & l_{31}u_{13} & \dots & l_{31}u_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1}u_{11} & l_{n1}u_{12} & l_{n1}u_{13} & \dots & l_{n1}u_{1n} \end{pmatrix}$$

其中 $u_{1j}, l_{2i}, i=1, 2, \dots, n$ 都是非零数。

重新排序的方法在某些情况下可以减少填充的数字,可能也可以提高准确性。另一种方法是仅计算在矩阵的乘法过程中被给定的模式,我们称此为 IMMM。算法 6 用这个方法实现,数学上与算法 3 等价,但是理论上比 3 更简单。

Algorithm 6 Iterative ILU factorization

```

1: Set the initial data  $L_0 = D = U_0 = 0$ .
2: for  $k = 1, 2, \dots, p$  do
3:    $B = L_0 U_0$  (IMMM on  $S$ )
4:    $B = A - B$  (on  $S$ )
5:    $D = \text{diag}(\text{diag}(B))$ 
6:    $U_0 = \text{triu}(B, 1)$ 
7:    $L_0 = \text{tril}(B, -1)D^{-1}$ 
8: end for
9:  $L = L_0 + I$  and  $U = U_0 + D$ .
```

这个算法操作在现存的线性运算库函数中没有找到,虽然根据共享内存的定义在并行上容易实现,但是在分配的内存平台,使用者必须处理不同操作者之间的交流。如果内存和运行时间都可以支持填充,则更推荐算法 3 和 4。

4.2 三角系统求解器

最终问题会归结到求解两个三角系统 $L(Uy)=x$ 。一种方法是向前向后替换,这是可以根据等级表被并行计算的。**第二种方法是雅克比迭代。**

$$\begin{aligned} z^k &= x - L_0 z^{k-1}, \\ y^k &= D^{-1} z - D^{-1} U_0 y^{k-1}, \end{aligned} \tag{4.1}$$

$L(Uy)=x$ 是两个线性系统,分为 $\begin{cases} Lz=x & ① \\ z=Uy & ② \end{cases}$
 对 ①, 由 $L=L_0+I$, 得 $(L_0+I)z=x \Rightarrow z=L_0z + (x-L_0z)$ 迭代公式为: $z^k = x - L_0 z^{k-1}$
 ②, 由 $U=U_0+D$, 得 $z=(U_0+D)y \Rightarrow y=D^{-1}z - D^{-1}U_0y$ 迭代公式为: $y^k = D^{-1}z^k - D^{-1}U_0 y^{k-1}$

Algorithm 7 Jacobi method for the systems $L(Uy) = x$ with p iterations.

```

1: Set the initial data  $z^0 = 0$  and  $y^0 = 0$ .
2: for  $k = 1, 2, \dots, q$  do
3:    $z^k = x - L_0 z^{k-1}$ 
4: end for
5: for  $k = 1, 2, \dots, q$  do
6:    $y^k = D^{-1} z^k - D^{-1} U_0 y^{k-1}$ 
7: end for
8: output  $y = y^p$ 
```

算法 7 用了这种方法，特殊的情况是 L_0 和 U_0 这两个矩阵都为 0，则意味着方案的最快的收敛。通过使用这个方法，主要的风险转化为了矩阵和向量的乘法。而且，预处理的解决方法得到的解只是精确解的一个近似值，当预处理达到很小的误差时才需要迭代方案 4.1。通常，每种方案的几个迭代步骤对于预处理而言都足够准确。

```
function [D,U0,z,y] = Jacobi(L0,U,x,p)
%初始化
n=size(L0,2);%矩阵L0的列数
z=zeros(n,1);
y=zeros(n,1);
D=diag(diag(U));
U0=U-D;

for k=1:p
    z=x-L0*z;
end

for k=1:p
    y=(D^(-1))*z-(D^(-1))*U0*y;
end

>> A=[2 3 2;10 3 4;3 6 1];
>> [L,U] = IterLU(A,3)
L =
    1.000000000000000    0    0
    5.000000000000000    1.000000000000000    0
    1.500000000000000   -0.125000000000000    1.000000000000000
U =
    2.000000000000000    3.000000000000000    2.000000000000000
    0   -12.000000000000000   -6.000000000000000
    0    0   -2.750000000000000
>> L0=L-eye(3)
L0 =
    0    0    0
    5.000000000000000    0    0
    1.500000000000000   -0.125000000000000    0
>> x=[1;2;3];
>> b=inv(A)*x
b =
    0.227272727272727
    0.454545454545455
   -0.409090909090909

>> [D,U0,z,y] = Jacobi(L0,U,x,3)
D =
    2.000000000000000    0    0
    0   -12.000000000000000    0
    0    0   -2.750000000000000
U0 =
    0    3    2
    0    0   -6
    0    0    0
z =
    1.000000000000000
   -3.000000000000000
    1.125000000000000
y =
    0.227272727272727
    0.454545454545455
   -0.409090909090909
```

5 数值实验 1：线性系统

在这部分，我们使用新的预条件处理来求解线性系统，用预条件共轭梯度方法。主要目的是确定新算法的参数，且用新的预条件处理器与传统的 ILU 比较。

5.1 IterILU(p,m)

在这部分，考虑齐次狄利克雷边值条件的拉普拉斯有限差分离散格式的系数矩阵，我们探讨算法 4 的 ILU 迭代预处理 IterILU(p,m)的性质和影响。二维和三维的网格数分别是 102×102 ， $102 \times 102 \times 102$ ，相应的方阵有 10^4 ， 10^6 行。

	2D Laplace 100×100		3D Laplace $100 \times 100 \times 100$	
	nnz	ratio	nnz	ratio
$p = 1$	29800	1	3970000	1
$p = 2$	39601	1.33	6910300	1.74
$p = 3$	49303	1.65	12721996	3.20
$p = 4$	68608	2.30	28972351	7.30
$p = 5$	97025	3.26	72694564	18.31
$p = 6$	143276	4.81	201462286	50.75

Table 5.1: Numbers of nonzeros in the lower triangular factor L generated by IterILU(p,m) for increasing p and $m = 0$ for the 2D and 3D finite difference Laplacian.

首先，我们测试由 IterILU(p,m)产生的因子的填充的性质。表格 5.1 展示了不同的 p 值的 L 的非零元个数和当 $p=1$ 时这些数字和非零数的比值。非零元的数字个数随着 p 的增加而增加，且因为三维的拉普拉斯在每一行或者每一列中的非零元数比二维的多，所以三维的比值增长得更快。为了保证因子的内存在控制

之内，我们限制我们的测试在 $p=1,2,3$ 时。

接下来的两个测试将重点放在改变增强数 m 的效果上，测试矩阵是三维的拉普拉斯方程。为了展示不完全的因子和准确因子的不同，我们介绍一个相对误差：

$$relative\ error(A, LU) = \max_i \left\{ \frac{\sum_j^n |A_{ij} - (LU)_{ij}|}{\sum_j^n |A_{ij}|} \right\} \quad (5.1)$$

图 5.1 左侧展示随着 m 的增加，相对误差也增加，但是在几次迭代 m 之后相对误差会达到一个稳定状态。然后，我们利用迭代数 m 后的右手侧的值为随机值的 PCG 方法中的 $IterILU(p,k)$ 因子作为预处理器。图 5.1 右侧展示，在几次迭代之后，PCG 迭代数达到一个稳定的状态。从图 5.1 我们可以得出，对于相对误差和迭代数，三次迭代之后就可以达到稳定值。

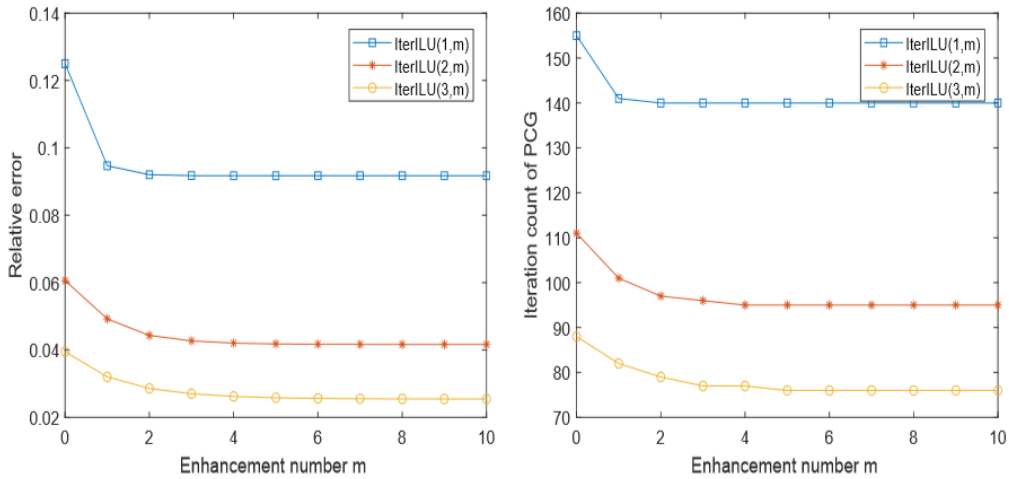


Fig. 5.1: Left: relative errors of $IterILU(p,m)$ with $p = 1, 2, 3$ and different enhancement numbers m . Right: PCG iteration counts with different values of p and m .

在上述测试中，三角系统的求解使用的是直接解法。在接下来的数值实验中，我们使用算法 7 的雅可比迭代方法来求解两个三角系统。图 5.2 展示了 PCG 和雅可比迭代的关系。

从结果中，我们得出带有 $IterILU(p,m)$ 预处理器的 PCG 迭代数在 $(p,m)=(1,3)$ 时 6 次迭代后达到稳定， $(p,m)=(2,3)$ 时 8 次迭代， $(p,m)=(3,3)$ 时 12 次迭代。因此，在雅可比方法中为迭代数 q 设置一致的值是不合适的。如果 q 太小，对于某些问题，预处理器的精确度是不够的，但是如果它太大，可能会浪费计算量。为了减轻不同数字的影响，在余下的实验中我们对于求解三角系统

使用精确的求解器。

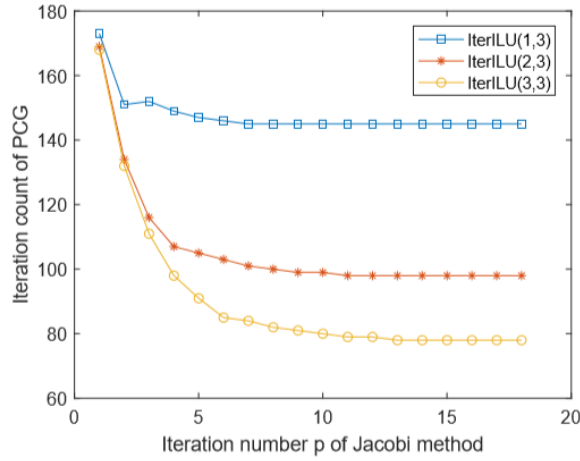


Fig. 5.2: PCG iteration counts using the Jacobi method of Algorithm 7 with different iteration numbers for the preconditioner triangular solves.

我们使用带有不同参数的 $\text{IterILU}(p,m)$ 绘制 PCG 收敛历史，将它们与标准的 level-0 的 ILU 预处理器进行比较。

图 5.3， $\text{IterILU}(1,3)$ 和标准的 ILU 产生了同样的结果，这意味着它们对于这个问题有着相似的影响。不同之处是 $\text{IterILU}(1,3)$ 能更轻易的实现而且具有良好的并行性， $\text{IterILU}(2,3)$ 和 $\text{IterILU}(3,3)$ 以允许更多的元素在他们的三角因子中为代价，比 level-0 更快地达到 PCG 收敛。

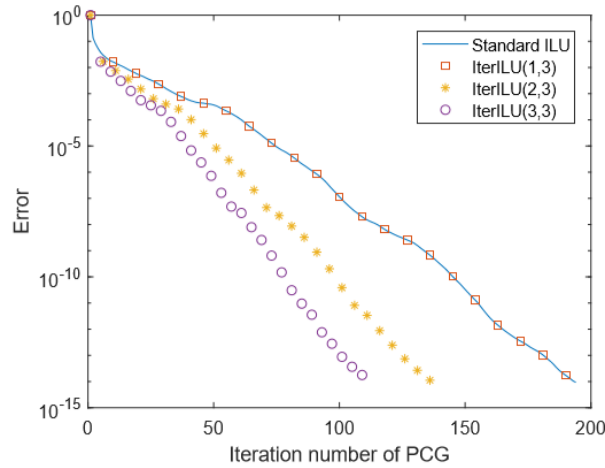


Fig. 5.3: Convergence history for solving the 3D Laplace problem using PCG with different ILU preconditioners.

接下来，我们对佛罗里达大学稀疏矩阵集合中的一些对称矩阵做同样的研究。

表格 5.2 列出了由不同的 p 产生的 $\text{IterILU}(p,m)$ 因子的维数、非零元的个数以及非零元的比值。

	row/column	nnz	$p = 1$	$p = 2$	$p = 3$
apache1	80800	542184	1	1.70	3.01
apache2	715176	4817870	1	1.71	3.02
thermal1	82654	574458	1	1.61	2.39
thermal2	1228045	8580313	1	1.62	2.42
parabolic_fem	525825	3674625	1	1.65	2.59
G3_circuit	1585478	7660826	1	1.31	1.69
thermomech_dM	204316	1423116	1	1.68	2.77
thermomech_TC	102158	711558	1	1.68	2.77
ecology2	999999	4995991	1	1.33	1.67
offshore	259789	4242673	1	2.21	6.16
af_shell3	504855	17562051	1	1.42	1.97

Table 5.2: The second and third columns in this table are the rows and numbers of nonzeros of some SPD matrices from the University of Florida sparse matrix collection [14], respectively. The rightmost two columns are the ratios of nonzeros of the IterILU(p, m) lower factor L with different p and the nonzeros with $p = 1$.

表格 5.3 展示了使用不同的预处理器的 PCG 的迭代数。在大多数例子中，IterILU(1,3)预处理器和标准的 level-0 模式 ILU 产生了相似的数字，迭代数随着 p 的增加而减少。IterILU(p, m)预处理器对最后两个测试例子不起作用。

	no p.c.	level-0 ILU	IterILU(1,3)	IterILU(2,3)	IterILU(3,3)
apache1	3774	364	359	281	249
apache2	5468	874	874	592	473
thermal1	1734	659	659	369	252
thermal2	6727	2598	2599	1443	984
parabolic_fem	3495	1432	1432	853	553
G3_circuit	19975	1152	1178	667	550
thermomech_dM	89	10	10	6	4
thermomech_TC	89	10	10	6	4
ecology2	7154	2125	2127	1338	1085
	no p.c.	level-0 ILU	IterILU(1,10)	IterILU(2,10)	IterILU(3,10)
offshore	-	567	574	-	-
af_shell3	4700	1172	-	-	-

Table 5.3: PCG iteration counts of different ILU preconditioners for linear systems from the University of Florida sparse matrix collection [14].

5.2 IterILUT(τ, k)

我们测试在算法 5 中的阈值分解 $\text{IterILUT}(\tau, k)$ 。就像之前讨论的，要仔细的确定参数。这里，我们考虑 *parabolic_fem* 矩阵，选择 $\tau=0.025$ 和 $\tau=0.006$ 。我们选择这两个参数的原因是因为用这两个参数产生的因子与 $\text{IterILU}(2, m)$ 和 $\text{IterILU}(3, m)$ 相比有相似的非零元数。

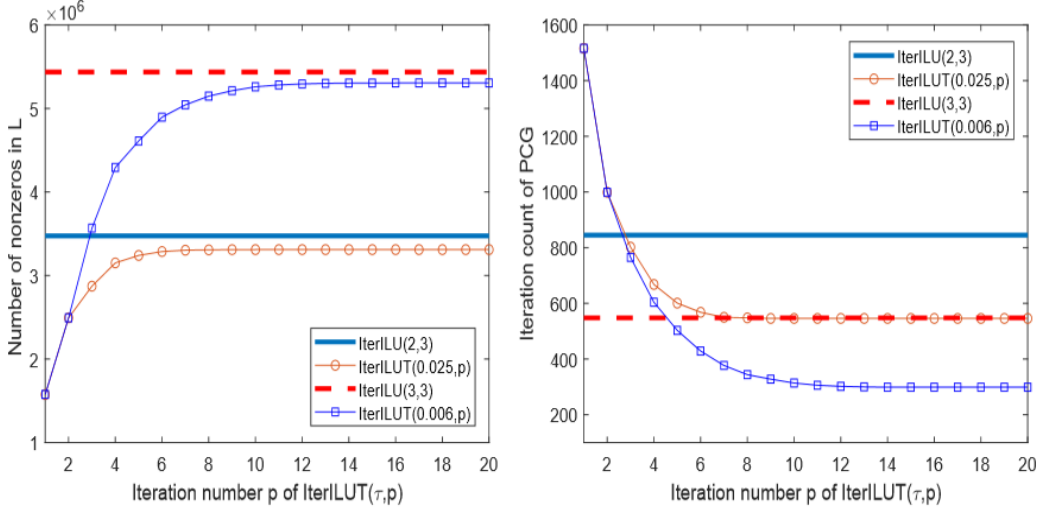


Fig. 5.4: Left: the number of nonzeros in the lower factor L generated by $\text{IterILUT}(\tau, p)$ with different threshold parameters and iteration numbers. Right: PCG iteration numbers using the $\text{IterILUT}(\tau, p)$ preconditioners with different τ and p .

图 5.4 左边展示了下三角因子 L 的非零数与增强数 k 之间的关系。我们设置停止的上限为 10^{-12} 。图 5.4 右边展示了 PCG 迭代数和增强数 m 之间的关系。在因子中非零数会在 $\tau=0.025$ 时, $p=5$ 之后趋于平稳, PCG 迭代数会在 $\tau=0.006$ 时, $p=10$ 之后趋于平稳。然后我们为每个 τ 选择这些 p 值, 我们在图 5.5 中画出不同的预处理器的 PCG 的收敛历史。

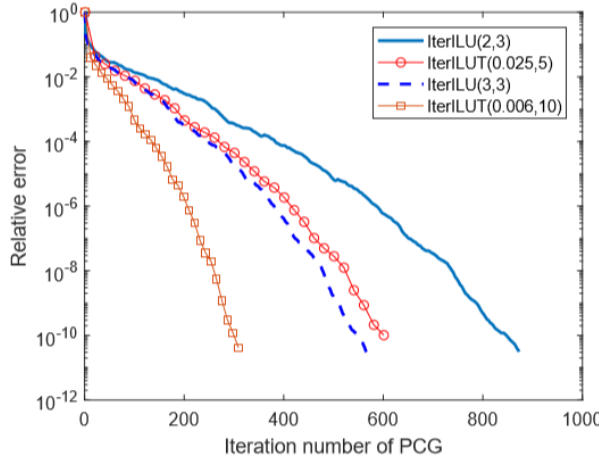


Fig. 5.5: PCG convergence history with $\text{IterILUT}(\tau, p)$ and $\text{IterILU}(p, m)$ preconditioners for the *parabolic_fem* problem.

这些结果表明尽管非零数是相似的，对于这个问题， $\text{IterILUT}(\tau, k)$ 比 $\text{IterILU}(p, m)$ 表现得更好。

6 数值实验 2：特征值问题

在这部分我们用算法 4 的 $\text{IterILU}(p, m)$ 来求解特征值问题。我们使用局部最优快预处理器共轭梯度法来计算谱的部分。除了预处理器，初始值，停止标准，同时计算的特征值和随机误差，这些都会影响特征值求解的收敛行为。当特征函数收敛退化或者甚至失败，可以用不同的初始值和参数重新开始。

我们计算 10^6 行三维有限差分的拉普拉斯矩阵的最小的特征值问题。我们同时计算四个最小特征值，设置停止准则是 10^{-12} ，使用随机初始数据。

图 6.1 给出了不同的预处理器的 LOBPCG 的收敛历史。这个结果和用 PCG 求解线性系统是相似的。Level-0 迭代预处理器和标准的 Level-0 迭代预处理器有同样的表现，收敛速率随着填充层数的增加而提高。

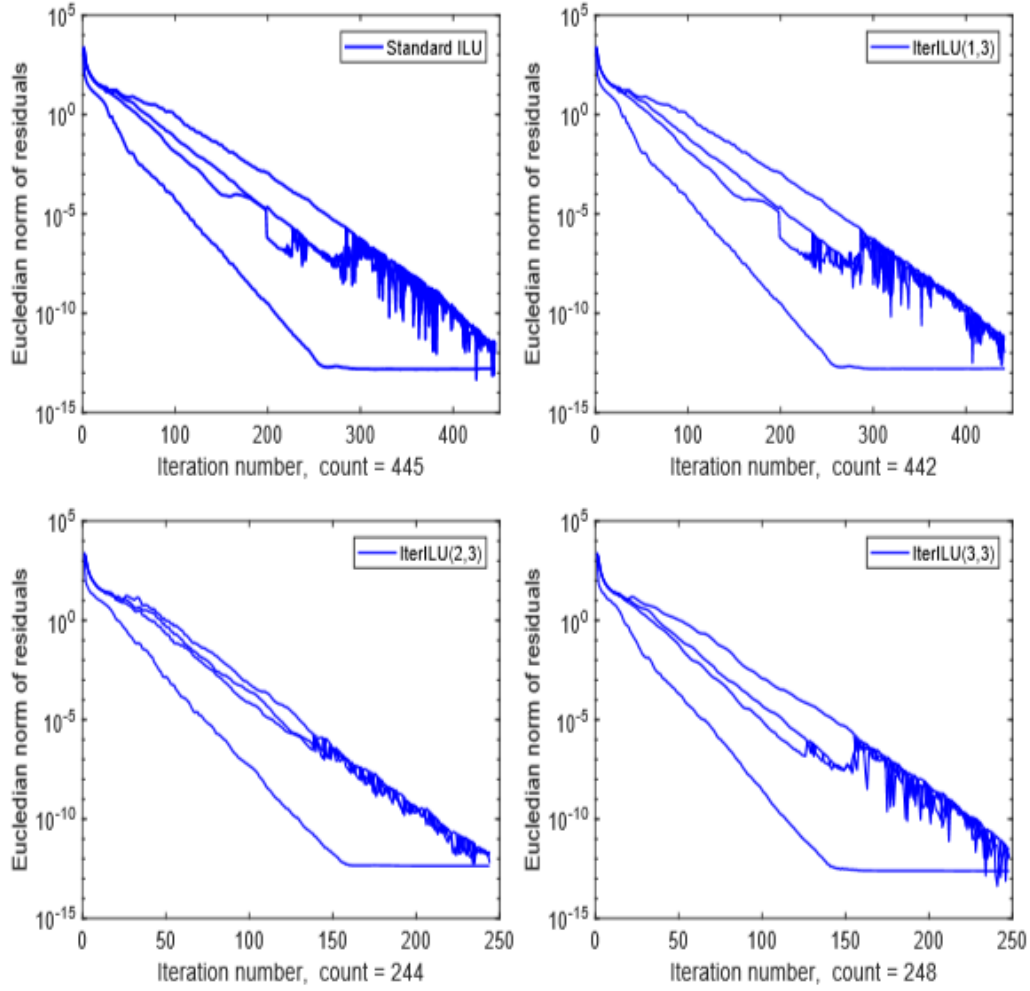


Fig. 6.1: LOBPCG convergence history in solving the Laplace eigenvalue problem with 10^6 rows using different ILU preconditioners.

6.1 混合形式的一个特征值问题

在这个小部分, 我们考虑一个普遍的由混合有限维分解的产生拉普拉斯特征值问题。我们将拉普拉斯特征值问题重新写为一阶系统:

$$\begin{cases} -\Delta u = \lambda u & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega \end{cases} \quad \begin{cases} \sigma - \text{grad } u = 0 & \text{in } \Omega \\ \text{div } \sigma = -\lambda u & \text{in } \Omega \\ u = 0 & \text{on } \partial\Omega. \end{cases}$$

然后考虑二维的情况, 用 Raviat-Thomas 元素来近似这个差分系统, [8,9,10]

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \sigma \\ u \end{pmatrix} = \mu \begin{pmatrix} 0 & 0 \\ 0 & M \end{pmatrix} \begin{pmatrix} \sigma \\ u \end{pmatrix}, \quad (6.1)$$

$$\lambda = -\mu,$$

这个参考文献中有更多的细节。得到的代数系统能被写作块矩阵的形式:

A, M 都是埃尔米特矩阵。

标准的 LOBPCG 方法通过使用 Raviat-Thomas 进程在埃尔米特矩阵谱的两端找到了特征值。但是，分散的谱如下：

$$\mu_M \leq \cdots \leq \mu_2 \leq \mu_1 (\leq 0) \leq \infty_1 = \infty_2 = \cdots = \infty_N.$$

有限的特征值通过特征对 $(\sigma, u)^T$ ， 10^6 ， $u=0$ 产生。想要求得的特征值是最小的值，是内部的特征值。为了逼近这些特征值，我们用齐次的 Rayleigh_Ritz 来代替 Rayleigh_Ritz 过程。另一个问题是左侧左下角的块为零，但是新算法要求对角元素全是非零的。之后我们使用特殊的转化方法，在 6.1 左右两边同乘以一个质量阵，得到图 6.2 和一个等价的广义特征值问题：

$$\begin{pmatrix} A & B^T \\ B & M \end{pmatrix} \begin{pmatrix} \sigma \\ u \end{pmatrix} = \mu \begin{pmatrix} 0 & 0 \\ 0 & M \end{pmatrix} \begin{pmatrix} \sigma \\ u \end{pmatrix}, \quad (6.2)$$

$$\lambda = -\mu + 1.$$

然后我们使用算法 4 的 IterILU(p,m) 预处理器为左侧矩阵产生预处理器。

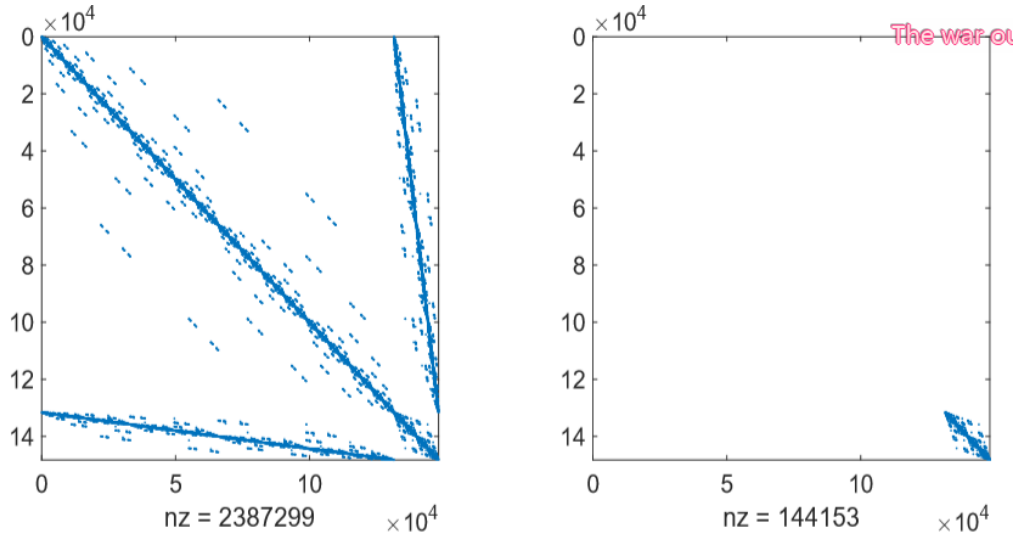


Fig. 6.2: The sparsity patterns of the matrices generated by mixed FEM.

在这个测试例子中，块矩阵被网格数为 128×128 ，行数为 148225 行的平方域 $[0, \Pi]^2$ 生成。我们设置 10^{-12} 为停止准则，计算六个最小的特征值。结果报告在图 6.3 中，表明 IterILU(1,3) 预处理器和标准的 ILU 预处理器有同样的表现，收敛速率随着填充层数的增加而提高。

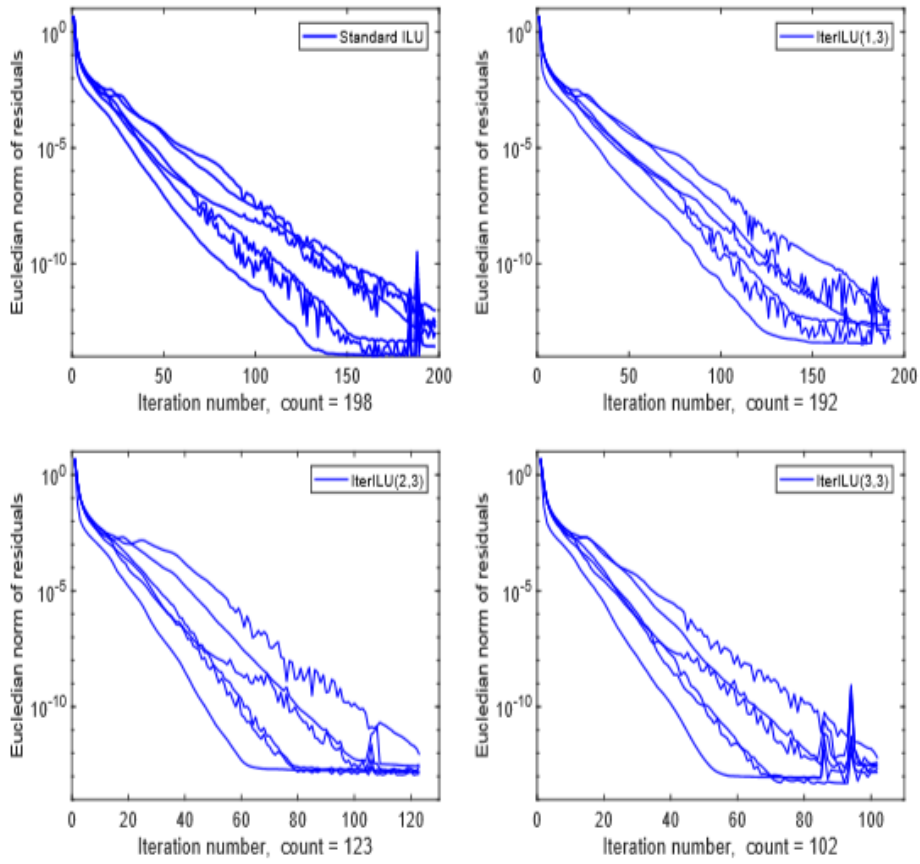


Fig. 6.3: The convergence history in solving mixed eigenvalue problem with different ILU preconditioners.

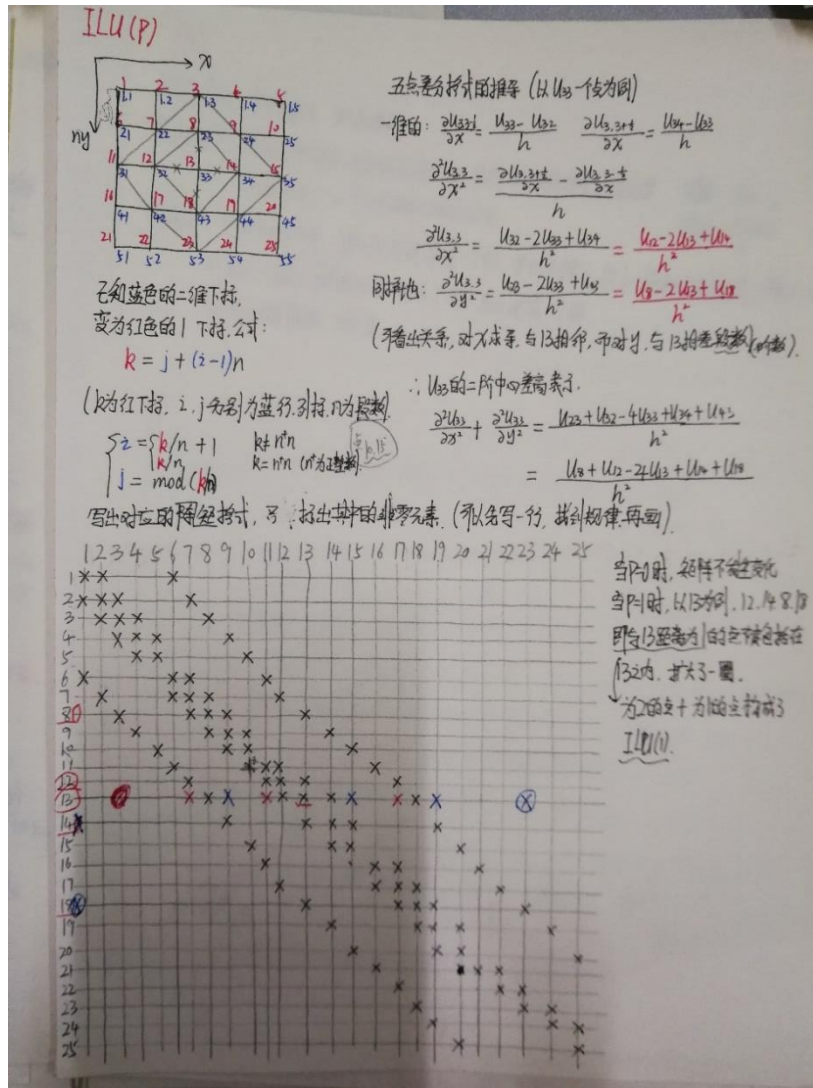
7 结论和限制

在这篇论文中，我们推导了传统的 LU 分解的迭代矩阵形式，而且基于这个迭代的过程构造了几个迭代的 ILU 预处理器。这个新的预处理器能被成功地应用于求解线性系统和特征值问题上。这个算法的一个主要的工作是矩阵和矩阵的乘法，这个预处理器能够从良好的并行中受益。利用现有数值库中的基本矩阵运算，在串行和并行计算平台中，这个算法能被有效的执行。

局限性：对一些矩阵，这个算法当有很高的填充层数时，可能不稳定甚至求解不出来，表格 5.3 中的后两个矩阵就是这样的情况。未来的工作是处理这些特殊的情况，而且提高迭代分解的鲁棒性。我们仅提出了基于位置和大小两个删除原则，考虑位置、大小、非零元个数的混合算法也是令人感兴趣的。

附录 1: ILU (p)

1. ILU(0)指的是矩阵所对应的五点差分格式的矩阵形式。
2. ILU(1)指的是原来的矩阵形式向外扩展一圈之后的形式。以 13 行 13 列的这个点为例，与它距离为 1 的点分别为 8,12,14,18，在五点差分格式的矩阵形式中找到这几个数字代表的行，将这四行的非零值移动到 13 行，则与 13 距离为 1 的点和距离为 2 的点的总和构成了这个点的 ILU(1)。以此类推所有的点，最终形成的所有的非零元组成的矩阵形式则为 ILU(1)。
3. ILU(p)指的是原来的矩阵形式向外扩展 p 圈之后的形式，矩阵中的每个点，与每个点距离为 1,2,...,p+1 的点构成的矩阵形式为 ILU(p)。
4. 一维下标 k 与二维下标(i,j)之间的关系式： $k=j+(i-1)n$
5. 五点差分格式的推导，见下图。



附录 2：LU 直接法的过程

$$A = LU$$

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

$$= \begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ l_{21}u_{11} & l_{21}u_{12} + u_{22} & l_{21}u_{13} + u_{23} & \cdots & l_{21}u_{1n} + u_{2n} \\ l_{31}u_{11} & l_{31}u_{12} + l_{32}u_{22} & l_{31}u_{13} + l_{32}u_{23} + l_{33}u_{33} & \cdots & l_{31}u_{1n} + l_{32}u_{2n} + l_{33}u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1}u_{11} & l_{n1}u_{12} + l_{n2}u_{22} & l_{n1}u_{13} + l_{n2}u_{23} + l_{n3}u_{33} & \cdots & \sum_{i=1}^n l_{ni}u_{in} \end{pmatrix}$$

$$a_{ij} = LU_j = (l_{i1}, l_{i2}, l_{i3}, \dots, l_{i,i-1}, 1, 0, \dots, 0) \begin{pmatrix} u_{1j} \\ u_{2j} \\ \vdots \\ u_{jj} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$a_{ij} = \sum_{m=1}^n l_{im} u_{mj} = \sum_{m=1}^{\min(i,j)} l_{im} u_{mj}$$

a_{ij} 由 l_{im} 和 u_{mj} 向量乘的最小非零元素个数决定。

附录 3：迭代次数 k 与网格单元尺寸最大直径的关系

有限元、有限差分的核心思想与定积分类似，都是分割、求和、取极限。

CG方法中, 已知条件数与网格单元尺寸最大直径的关系式是: $k_2(A) = O(h^{-2})$

已知 $\|x^k - x\|_A \leq 2 \left(\frac{\sqrt{k_2(A)} - 1}{\sqrt{k_2(A)} + 1} \right)^k \|x^0 - x\|_A$

令 $\|x^k - x\|_A \cdot \frac{1}{\|x^0 - x\|_A} = 10^{-6} = 2 \left(\frac{\sqrt{k_2(A)} - 1}{\sqrt{k_2(A)} + 1} \right)^k$, 分析 k 与 h 之间关系

记: $t = \sqrt{k_2(A)} \approx h^{-1}$

$\therefore 2 \left(\frac{t-1}{t+1} \right)^k = 10^{-6}$

$\left(\frac{t-1}{t+1} \right) = 1 - \frac{2}{t+1} \xrightarrow{t \rightarrow \infty} 1 - \frac{2}{t} \approx 1 - 2h$ 或 $1 - \frac{2}{t+1} \approx 1 - \frac{2}{h+1} = 1 - \frac{2h}{1+h} \xrightarrow{h \rightarrow 0} 1 - 2h$

$\therefore 2(1-2h)^k = 10^{-6}$

$(1-2h)^k = 1 - 2kh + O(h^2)$

$\therefore 1 - 2kh = \frac{10^{-6}}{2} \Rightarrow kh = 2 \cdot 10^{-6} \Rightarrow k = O(h^{-1})$

\therefore 当网格尺寸数加密一倍, CG达到收敛的迭代次数也增加一倍

附录 4：传统的 LU 分解法与此论文的迭代法的比较

1. 杜立特尔直接法求解 L 和 U

杜立特尔直接法解 LU

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 2 & 0 & -1 & 0 & 3 \\ 1 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 4 & -2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & \frac{1}{3} & 1 \\ 0 & 0 & 0 & -\frac{4}{3} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -3 & 0 & 3 \\ 0 & 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 0 & -\frac{6}{5} \end{pmatrix}$$

$A = L \cdot U$

2. 高斯消元法求解 L 和 U

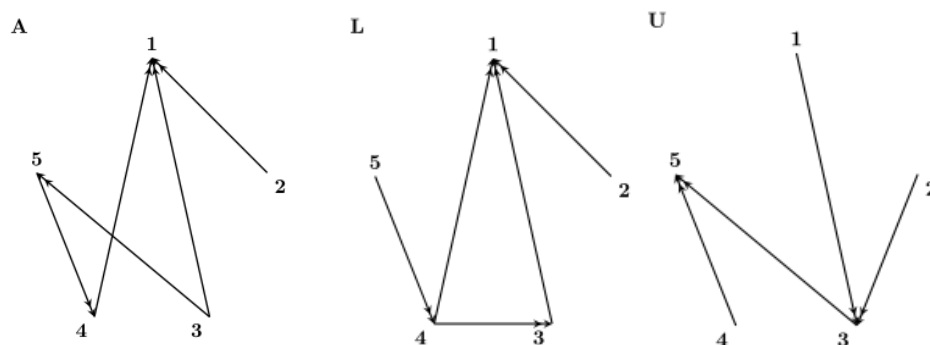
$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 & 0 \\ 2 & 0 & -1 & 0 & 3 \\ 1 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 4 & -2 \end{pmatrix} \xrightarrow[l_1, l_2, l_3]{\substack{R_2 - R_1 \\ R_3 - 2R_1 \\ R_4 - R_1}} \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -3 & 0 & 3 \\ 0 & 0 & -1 & 5 & 0 \\ 0 & 0 & 0 & 4 & -2 \end{pmatrix} \xrightarrow[l_4]{R_4 - \frac{1}{3}R_3} \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -3 & 0 & 3 \\ 0 & 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 4 & -2 \end{pmatrix}$$

$$\xrightarrow[l_5]{R_5 - \frac{4}{5}R_4} \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -3 & 0 & 3 \\ 0 & 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 0 & -\frac{6}{5} \end{pmatrix}$$

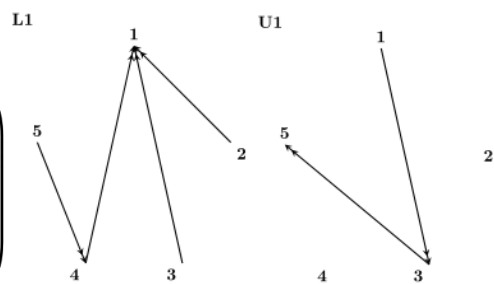
$l_5 l_4 l_3 l_2 l_1 A = U$
 $A = (l_5 l_4 l_3 l_2 l_1)^T U$
 $L = (l_5 l_4 l_3 l_2 l_1)^{-T}$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -\frac{4}{5} & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -\frac{4}{5} & 1 \end{pmatrix}$$

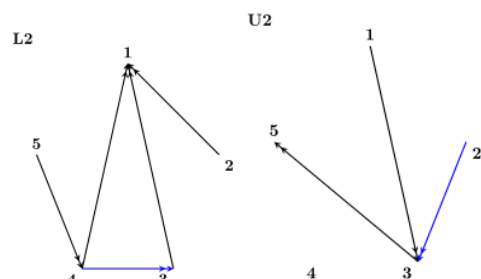
3. 本论文的迭代法求解 L 和 U



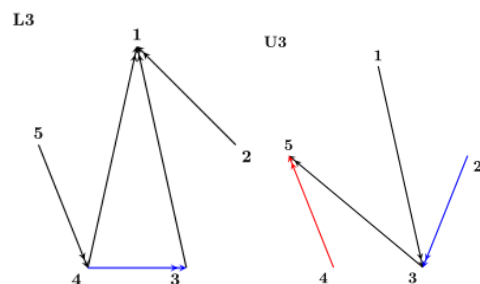
$$U_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 3 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$



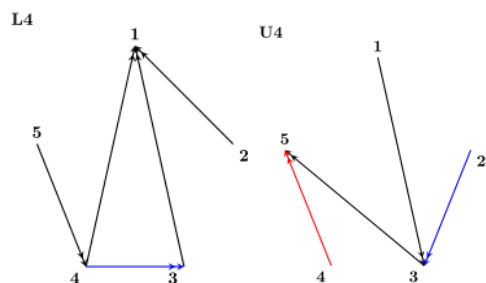
$$U_2 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -3 & 0 & 3 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$



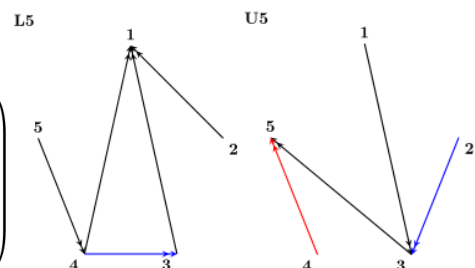
$$U_3 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -3 & 0 & 3 \\ 0 & 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$



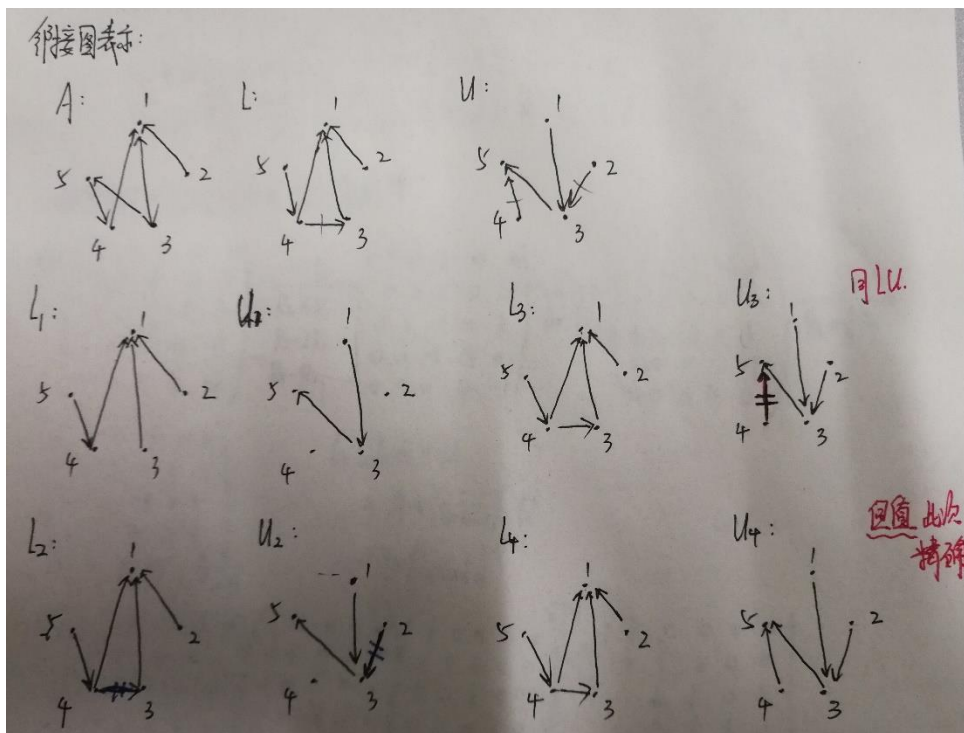
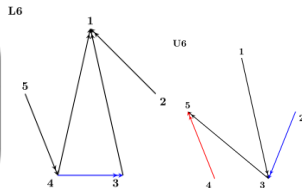
$$U_4 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -3 & 0 & 3 \\ 0 & 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 0 & -6/5 \end{pmatrix}$$



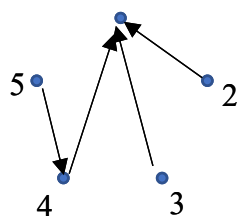
$$U_5 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & \textcolor{blue}{1} & 0 & 0 \\ 0 & 0 & \textbf{-3} & 0 & 3 \\ 0 & 0 & 0 & 5 & \textcolor{red}{-1} \\ 0 & 0 & 0 & 0 & \textcolor{green}{*-6/5} \end{pmatrix}$$



$$L_6 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1/3 & 1 & 0 \\ 0 & 0 & 0 & 4/5 & 1 \end{pmatrix}, \quad U_6 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & -3 & 0 & 3 \\ 0 & 0 & 0 & 5 & -1 \\ 0 & 0 & 0 & 0 & -6/5 \end{pmatrix}$$



L1:



$$L_{01} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4/5 & 0 \end{pmatrix}, \quad U_{01} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad D_{01} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$

$$L_02 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & \textcolor{blue}{1/3} & 0 & 0 \\ 0 & 0 & 0 & 4/5 & 0 \end{pmatrix}, \quad U_02 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \textcolor{blue}{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad D_02 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & \textbf{-3} & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$

$$L_03 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & \textcolor{blue}{1/3} & 0 & 0 \\ 0 & 0 & 0 & 4/5 & 0 \end{pmatrix}, \quad U_03 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \textcolor{blue}{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & \textcolor{red}{-1} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad D_03 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & \textbf{-3} & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & -2 \end{pmatrix}$$

$$L_04 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & \textcolor{blue}{1/3} & 0 & 0 \\ 0 & 0 & 0 & 4/5 & 0 \end{pmatrix}, \quad U_04 = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \textcolor{blue}{1} & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & \textcolor{red}{-1} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad D_04 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & \textbf{-3} & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & \textbf{-1.2} \end{pmatrix}$$

```

D =
 1
 2
-1
 5
-2
U0 =
 0 0 1 0 0
 0 0 0 0 0
 0 0 0 0 3
 0 0 0 0 0
 0 0 0 0 0
L0 =
 0 0 0 0
-1.000000000000000 0 0 0
 2.000000000000000 0 0 0
 1.000000000000000 0 0 0
 0 0 0.800000000000000

```

```

D =
 1
 2
-3
 5
-2
U0 =
 0 0 1 0 0
 0 0 1 0 0
 0 0 0 0 3
 0 0 0 0 0
 0 0 0 0 0
L0 =
 0 0 0 0
 0 -1.000000000000000 0 0 0
 0 2.000000000000000 0 0 0
 0 1.000000000000000 0 0 0
 0 0 0.333333333333333 0 0.800000000000000

```

```

D =
 1
 2
-3
 5
-2
U0 =
 0 0 1 0 0
 0 0 1 0 0
 0 0 0 0 3
 0 0 0 0 -1
 0 0 0 0 0
L0 =
 0 0 0 0
-1.000000000000000 0 0 0
 2.000000000000000 0 0 0
 1.000000000000000 0 0.333333333333333 0
 0 0 0 0.800000000000000

```

```

D =
 1.000000000000000
 2.000000000000000
-3.000000000000000
 5.000000000000000
-1.200000000000000
U0 =
 0 0 1 0 0
 0 0 1 0 0
 0 0 0 0 3
 0 0 0 0 -1
 0 0 0 0 0
L0 =
 0 0 0 0
 0 -1.000000000000000 0 0 0
 0 2.000000000000000 0 0 0
 0 1.000000000000000 0 0 0
 0 0 0.333333333333333 0 0.800000000000000

```

D =

1.0000000000000000
2.0000000000000000
-3.0000000000000000
5.0000000000000000
-1.2000000000000000

U0 =

0 0 1 0 0
0 0 1 0 0
0 0 0 0 3
0 0 0 0 -1
0 0 0 0 0

L0 =

0	0	0	0	0
-1.0000000000000000	0	0	0	0
2.0000000000000000	0	0	0	0
1.0000000000000000	0	0.3333333333333333	0	0
0	0	0	0.8000000000000000	0