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：，，其中是矩阵A的特征值。

（A为对称正定阵）最速下降法的收敛性：

（A为对称正定阵）CG方法的收敛性：

1. P为对称正定阵）PCG方法的收敛性：



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，从降低到。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,)) 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]。

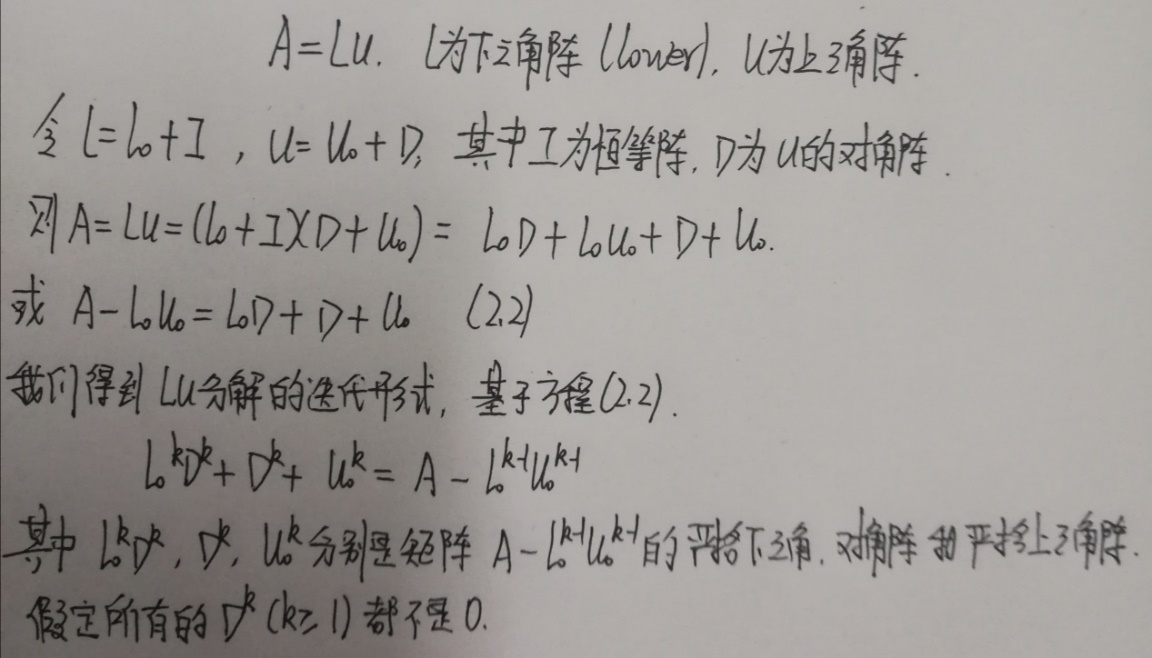
在这篇论文中，我们构造和分析一种给定的矩阵的迭代ILU分解。这个新算法的主要程序基础是稀疏矩阵的乘法。

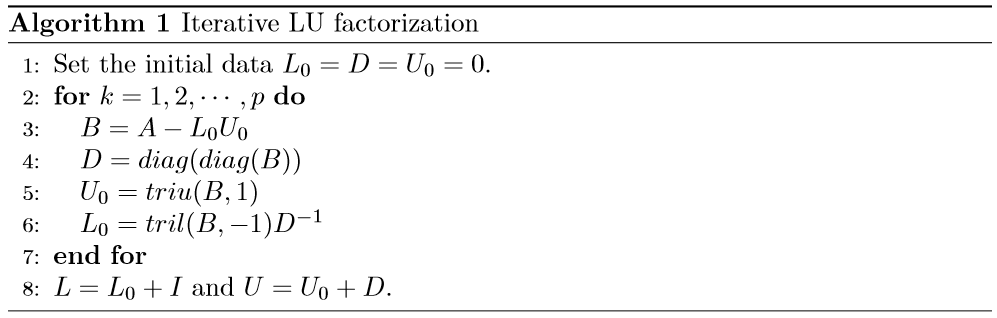
这篇论文不再涉及到新算法的并行性，因为这个算法仅会涉及到矩阵和矩阵乘法以及矩阵和向量的乘法。这个已经被证实和使用过了。

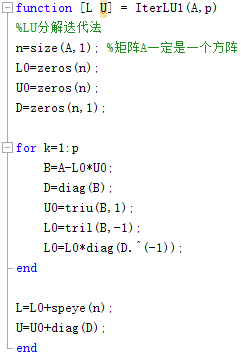
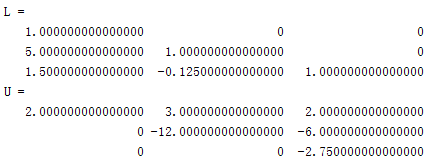
论文安排如下：在第2部分，推导矩阵形式的传统的LU分解并且回忆它的填充规则。在第3部分提出几个新的ILU迭代分解并且证明他们的收敛结果。第4部分讨论一些新的算法中涉及到的问题，例如矩阵和矩阵的乘法以及三角系统的求解。在5,6部分，我们利用这种新的方法来分别求解线性系统和特征值问题，展示几个数值实验的结果。在7部分是一些结论和限制(缺点)。

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：

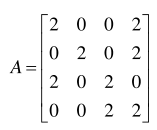
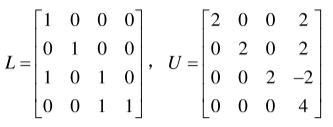
算法1的伪代码如下：

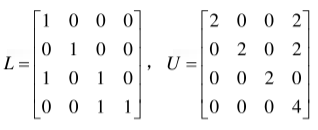
算法的实现：

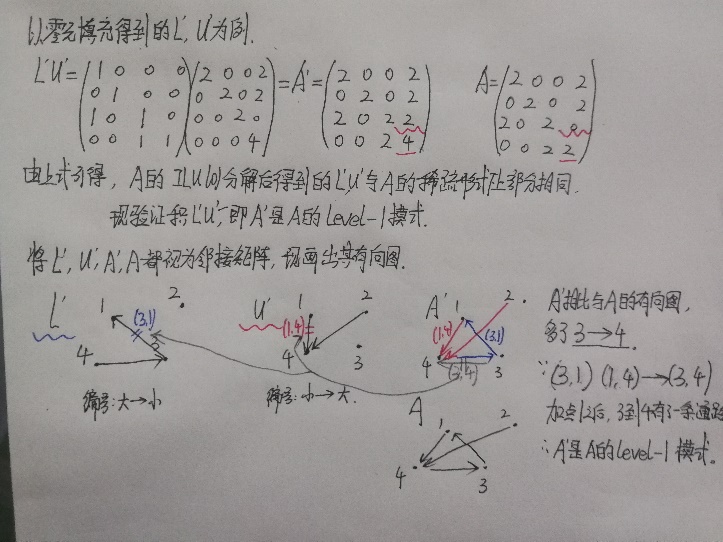
定理2.1：在算法1中的L,U至多n步收敛到A的准确LU分解。（已经验证过当p大于等于3时，L，U的值是不发生变化的）

2.1元素的填充

记SA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的和与內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。）

根据删除原则，得到

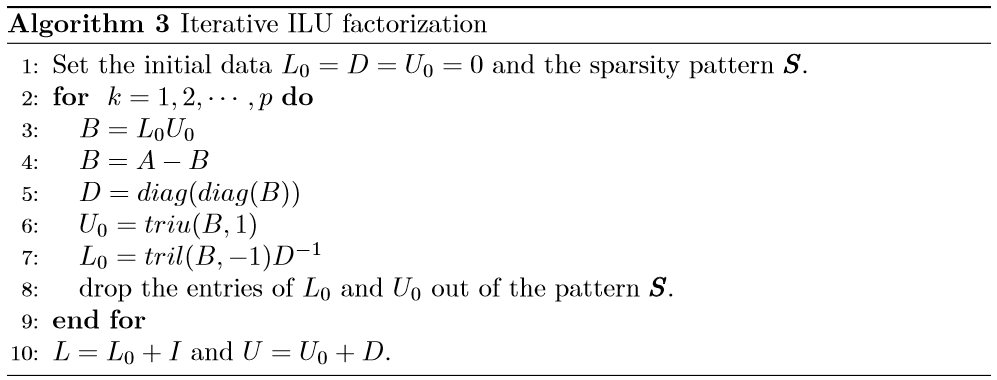
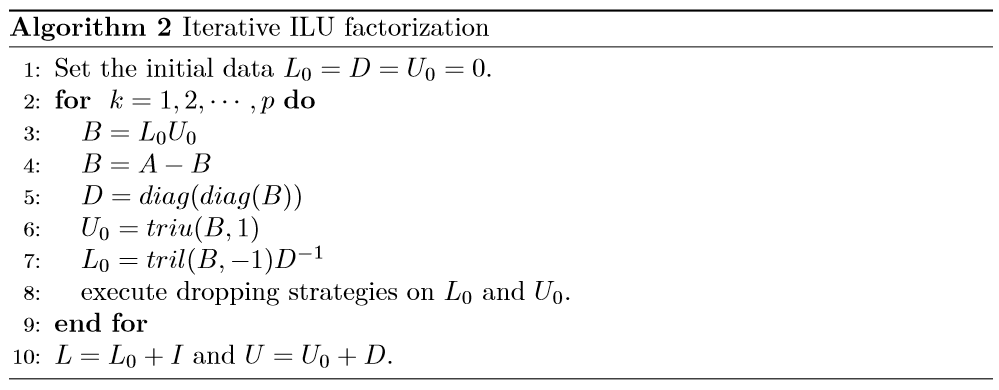
引理2.2，若和是A的ILU(0)模式分解的矩阵，则积的稀疏模式是A的level-1模式。

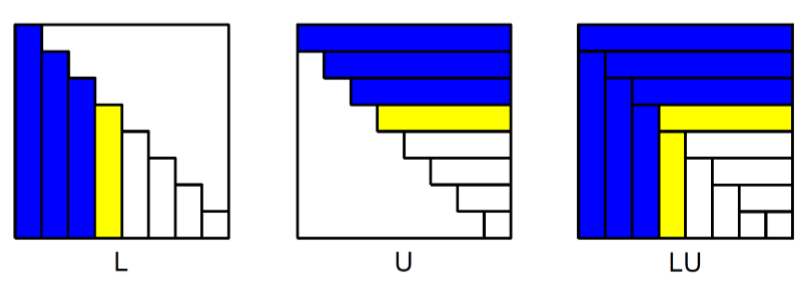
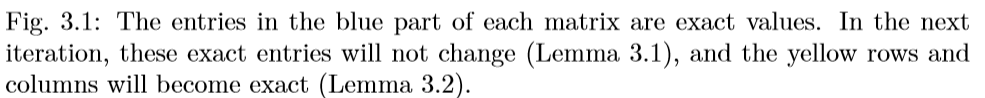
定理2.3，当k大于等于2时，的稀疏模式是的level-1模式。

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

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中除和之外的项。如果S包含A的稀疏形式，在第一次迭代之后。，AL0，AD和AU0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。

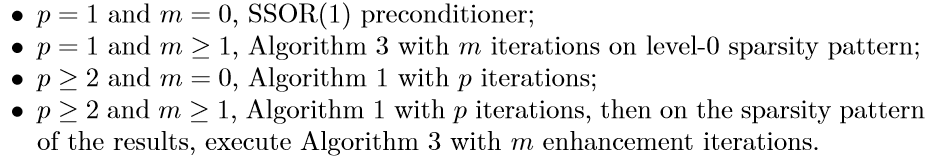
接下来，我们证明算法3的收敛性，我们用图3.1来阐述证明的过程。

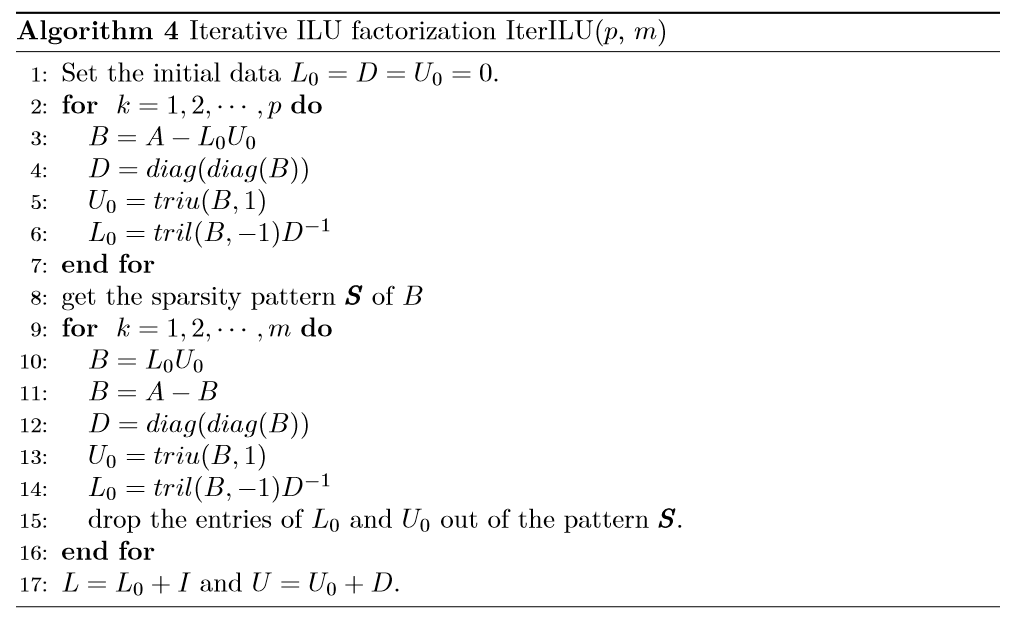
引理3.1 在算法3中，如果稀疏模式s的前k行和前k列的乘积LU等于A，则在下一次迭代中，L的前k列和U的前k行不会发生变化。

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：在稀疏模式下，算法3的第k次迭代得到的和收敛于不完全的ILU(k,)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。

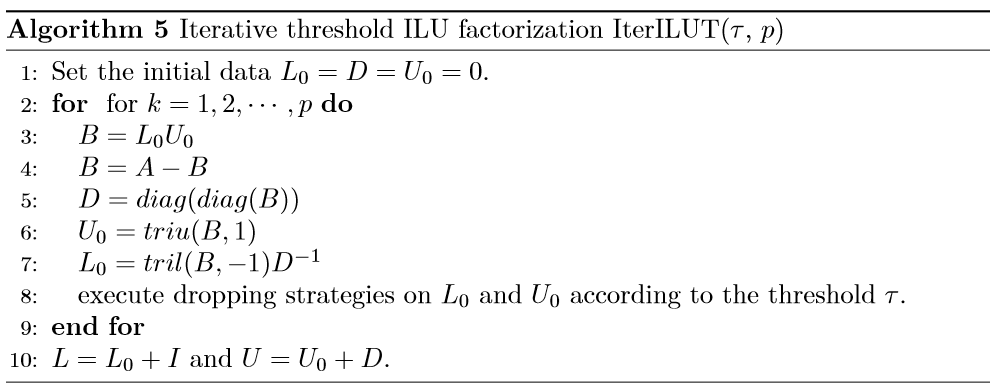
通常，随着p的增加空间也在增加。当固定了p，解三角系统的花费的时间和空间也应该被考虑在其中，我们在matlab中写一个简短的程序，粘贴在附录A中。

3.2 基于阈值

我们提出另一个删除原则，仅仅基于L0和U0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L0的行中删除比值小的元素。

（2.）在U的每列中找到最大的绝对值T，在相应的U0的列中删除比值小的元素。

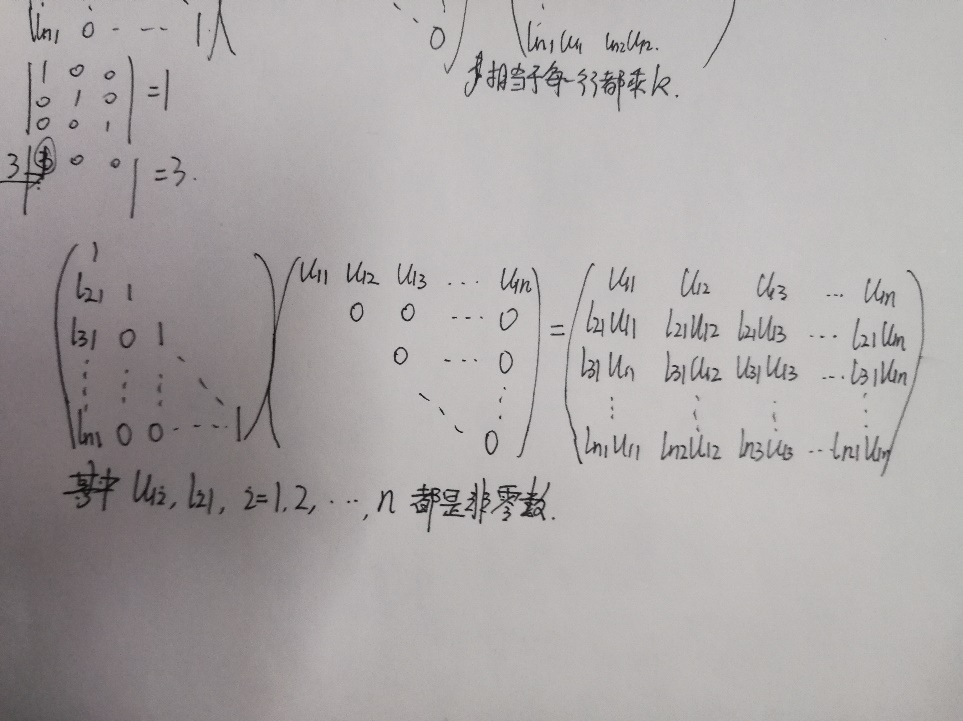
阈值T是为了L0和U0统一设置的。我们将算法5中的算法称为IterILU(T,p)，其中p是迭代次数。

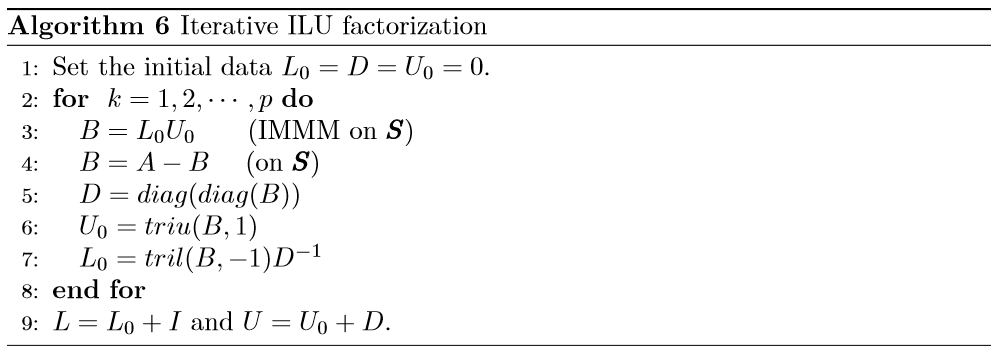
基于阈值的ILU应用没有基于模式的应用广泛。因为当矩阵元素的值集中于一个值的周围，这个方法是不合适的。而且值T的选择是一个难点，选的太大，矩阵的重要部分可能被删除，选的太小，可能在减少空间存储方面是没有意义的。而且这个阈值应该根据算法的想要达到准确性和因素的空间存储来设计。当然了，基于位置、数字和元素的大小的删除策略可以被混合使用。

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L0的第一列和U0的第一行是满向量时，L0和U0的乘积会产生一个满矩阵，见下图：

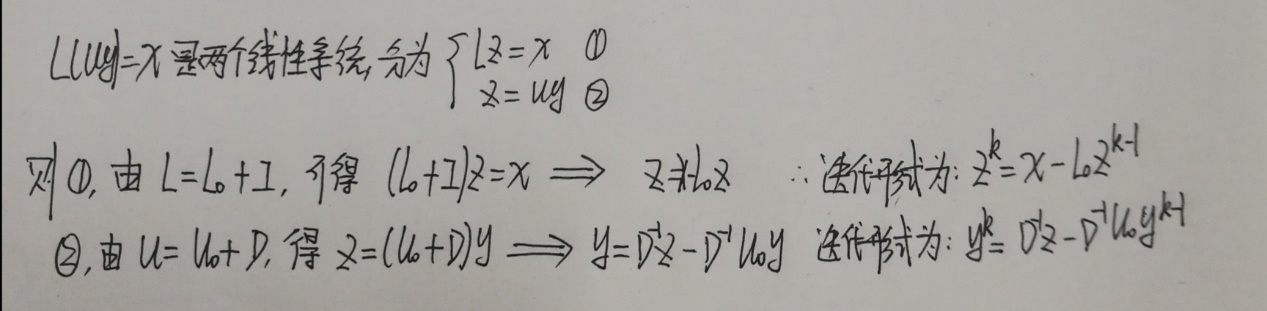
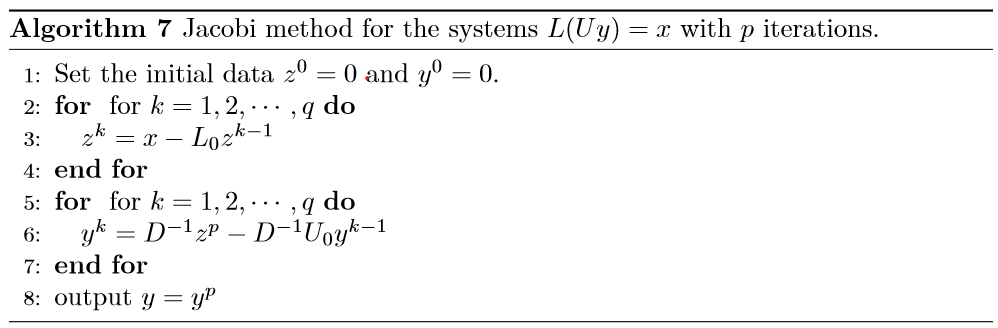


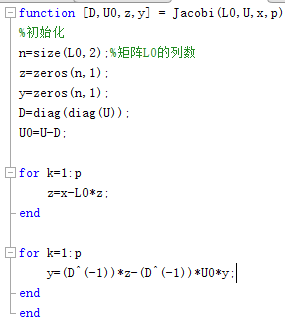
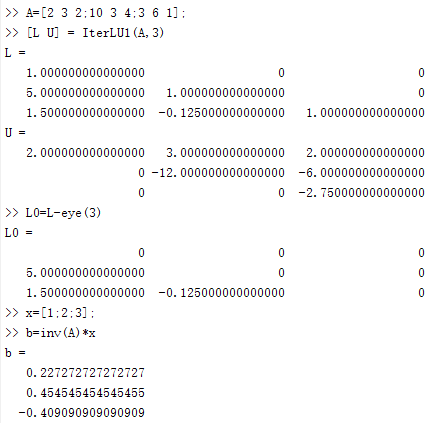
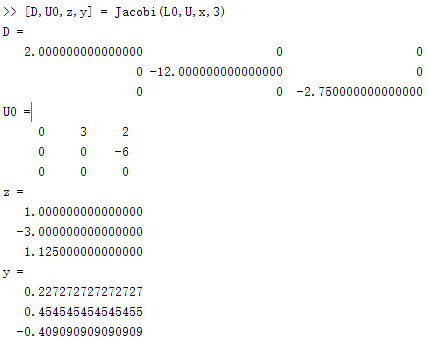
重新排序的方法在某些情况下可以减少填充的数字，可能也可以提高准确性。另一种方法是仅计算在矩阵的乘法过程中被给定的模式，我们称此为IMMM。算法6用这个方法实现，数学上与算法3等价，但是理论上比3更简单。

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。

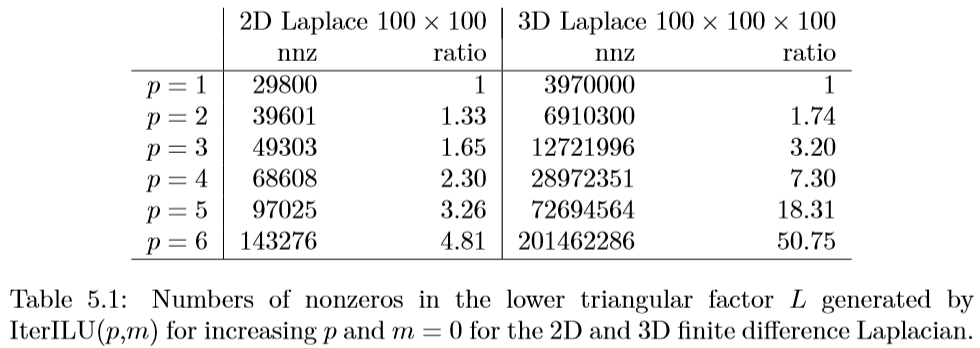
 (4.1)

算法7用了这种方法，特殊的情况是L0和U0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。

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。

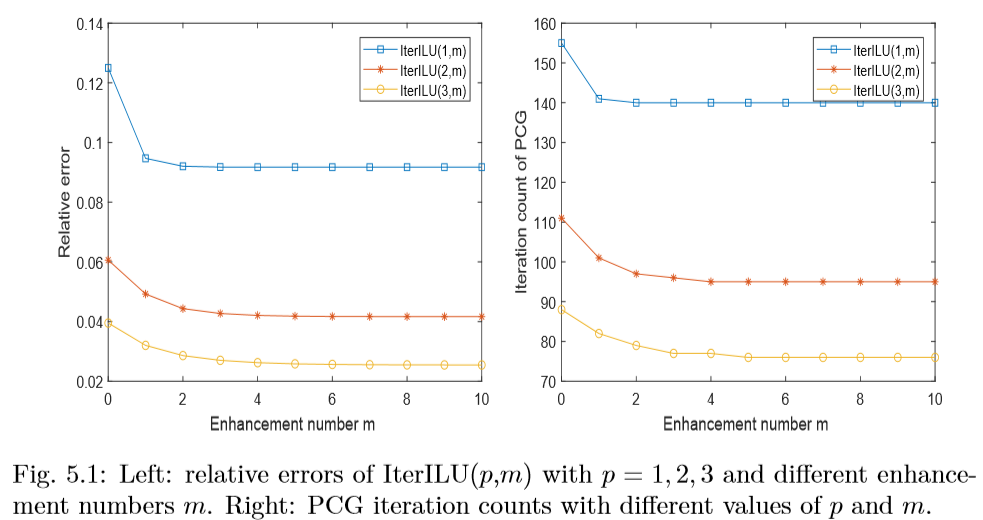
5.1  IterILU(p,m)

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，，相应的方阵有，行。

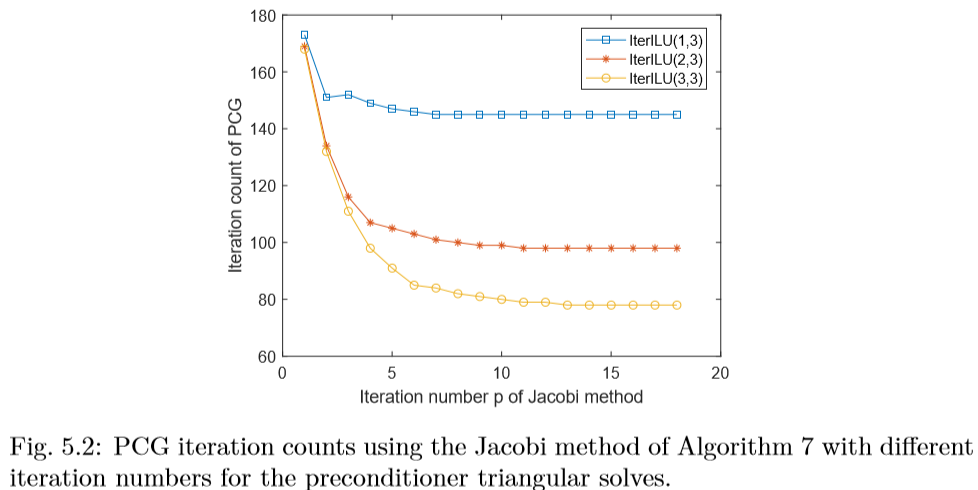
首先，我们测试由IterILU(p,m)产生的因子的填充的性质。表格5.1展示了不同的p值的L的非零元个数和当p=1时这些数字和非零数的比值。非零元的数字个数随着p的增加而增加，且因为三维的拉普拉斯在每一行或者每一列中的非零元数比二维的多，所以三维的比值增长得更快。为了保证因子的内存在控制之内，我们限制我们的测试在p=1,2,3时。

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：

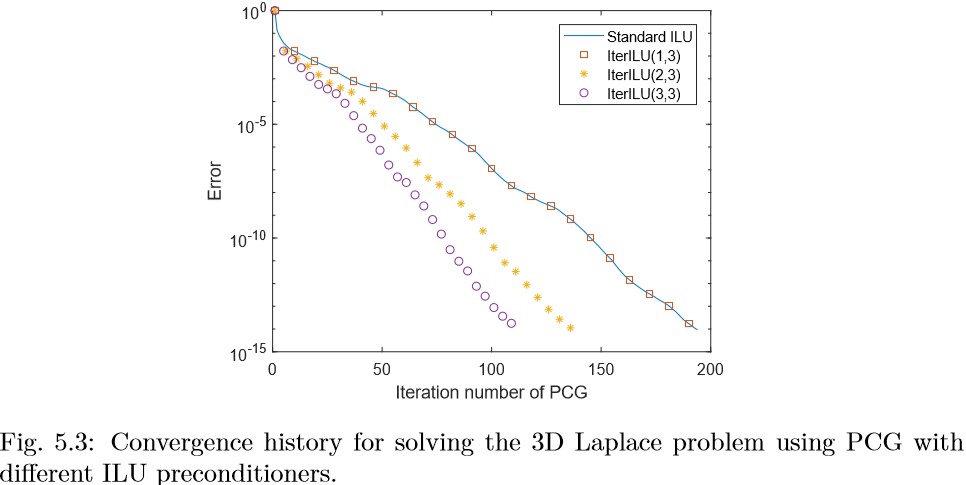


图5.1左侧展示随着m的增加，相对误差也增加，但是在几次迭代m之后相对误差会达到一个稳定状态。然后，我们利用迭代数m后的右手侧的值为随机值的PCG方法中的IterILU(p,k)因子作为预处理器。图5.1右侧展示，在几次迭代之后，PCG迭代数达到一个稳定的状态。从图5.1我们可以得出，对于相对误差和迭代数，三次迭代之后就可以达到稳定值。

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的雅克比迭代方法来求解两个三角系统。图5.2展示了PCG和雅克比迭代的关系。

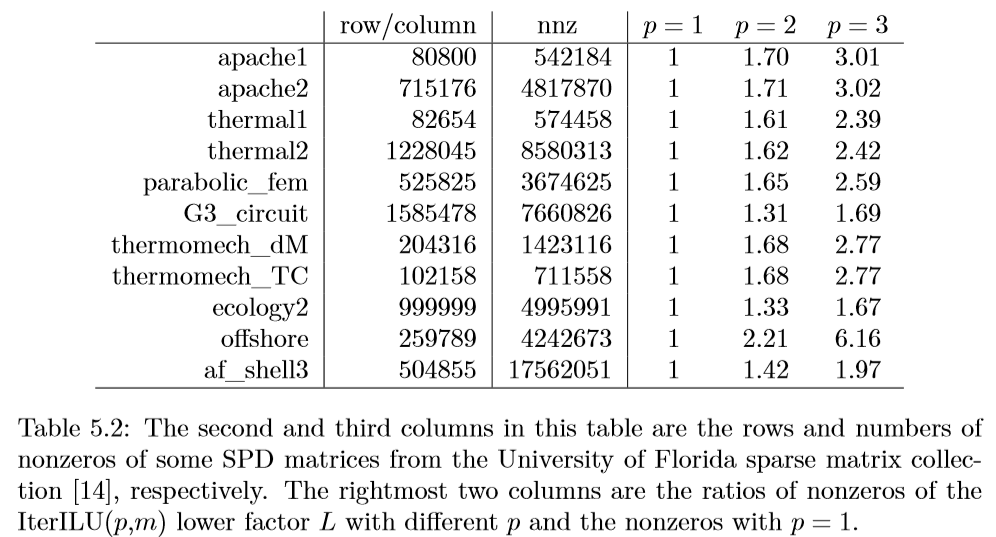
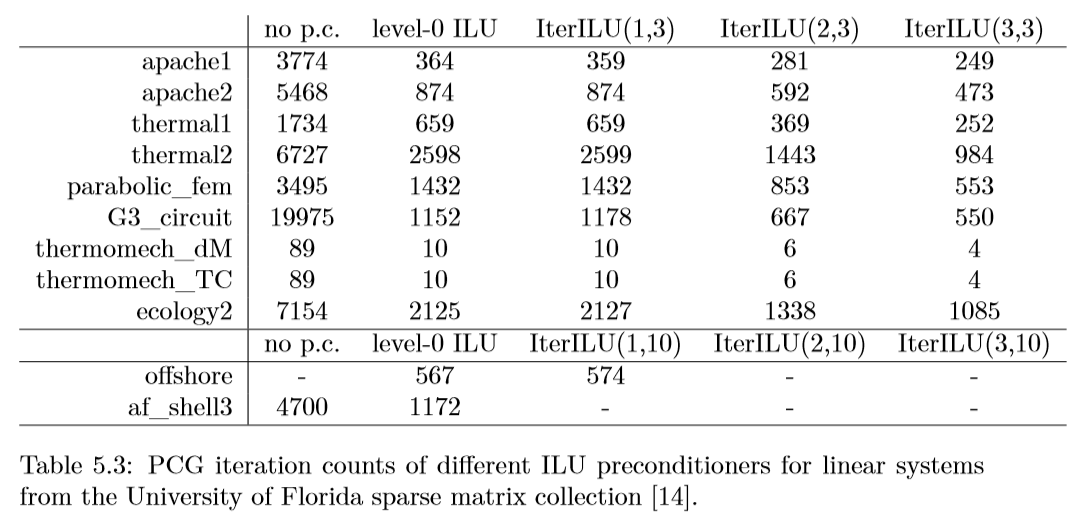
从结果中，我们得出带有IterILU(p,m)预处理器的PCG迭代数在(p,m)=(1,3)时6次迭代后达到稳定，(p,m)=(2,3)时8次迭代，(p,m)=(3,3)时12次迭代。因此，在雅可比方法中为迭代数q设置一致的值是不合适的。如果q太小，对于某些问题，预处理器的精确度是不够的，但是如果它太大，可能会浪费计算量。为了减轻不同数字的影响，在余下的实验中我们对于求解三角系统使用精确的求解器。

我们使用带有不同参数的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。

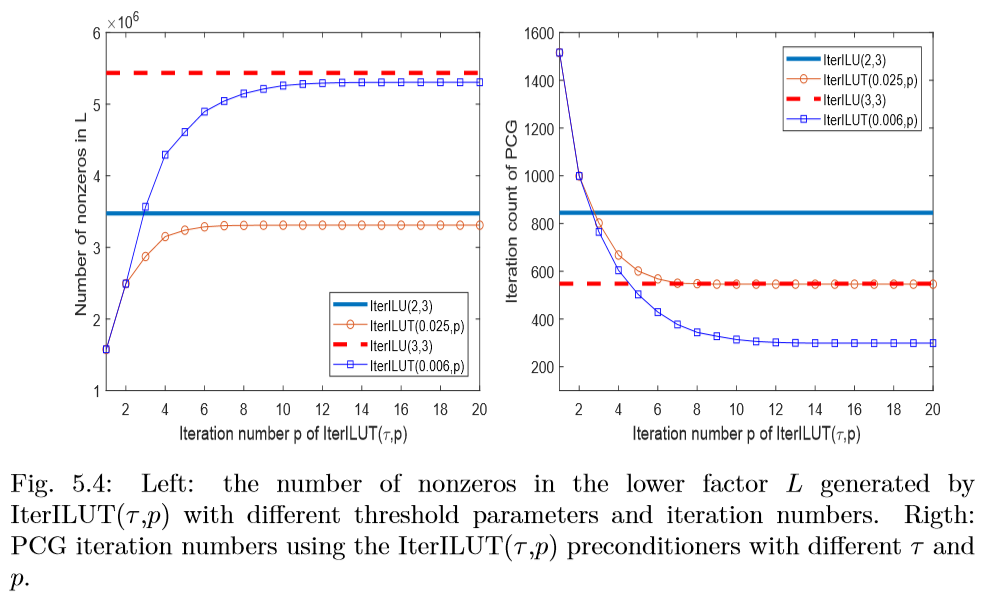
图5.3 ，IterILU(1,3)和标准的ILU产生了同样的结果，这意味着它们对于这个问题有着相似的影响。不同之处是IterILU(1,3)能更轻易的实现而且具有良好的并行性，IterILU(2,3)和IterILU(3,3)以允许更多的元素在他们的三角因子中为代价，比level-0更快地达到PCG收敛。

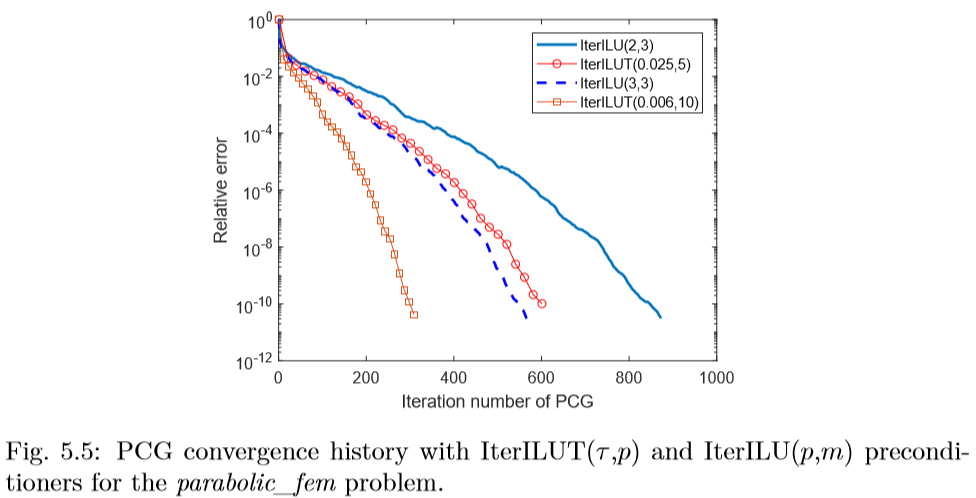
接下来，我们对佛罗里达大学稀疏矩阵集合中的一些对称矩阵做同样的研究。

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

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。

5.2 IterILUT(,k)

我们测试在算法5中的阈值分解IterILUT(,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=0.025和=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IterILU(2,m)和IterILU(3,m)相比有相似的非零元数。

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。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=0.025时，p=5之后趋于平稳，PCG迭代数会在=0.006时， p=10之后趋于平稳。然后我们为每个选择这些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。

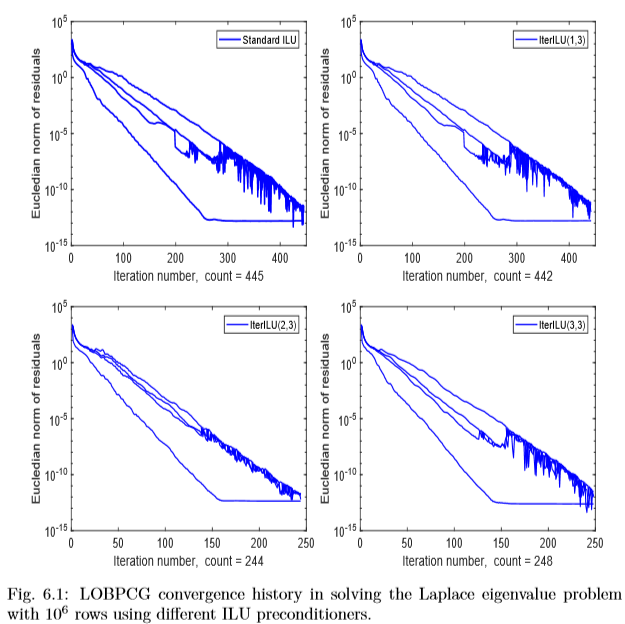
这些结果表明尽管非零数是相似的，对于这个问题，IterILUT(,k)比IterILU(p,m)表现得更好。

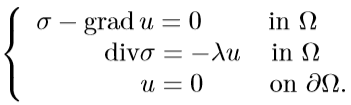
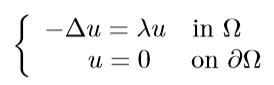
6 数值实验2：特征值问题

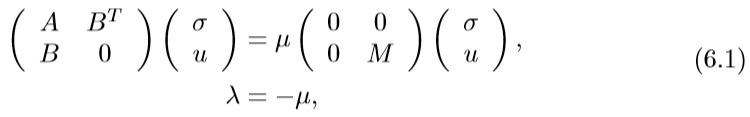
在这部分我们用算法4的IterILU(p,m)来求解特征值问题。我们使用局部最优快预处理器共轭梯度法来计算谱的部分。除了预处理器，初始值，停止标准，同时计算的特征值和随机误差，这些都会影响特征值求解的收敛行为。当特征函数收敛退化或者甚至失败，可以用不同的初始值和参数重新开始。

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，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。

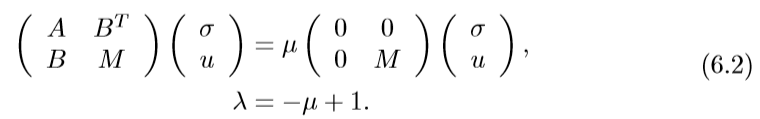
6.1 混合形式的一个特征值问题

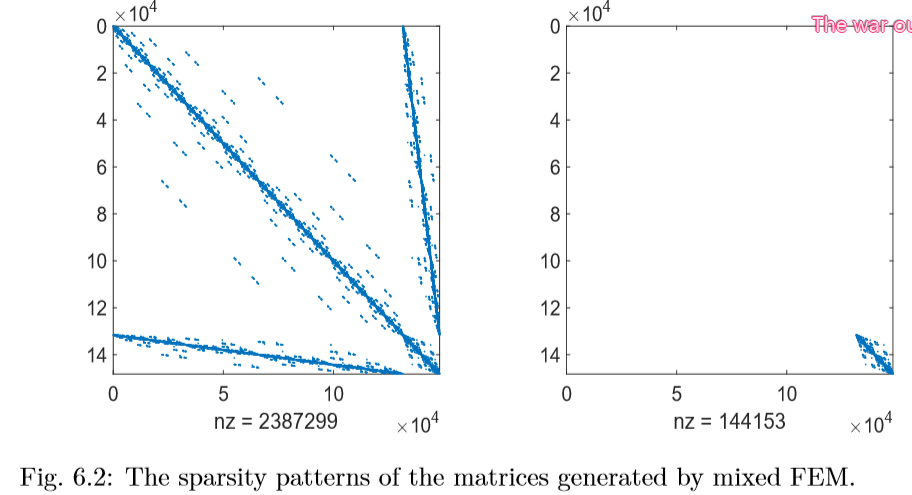
在这个小部分，我们考虑一个普遍的由混合有限维分解的产生拉普拉斯特征值问题。我们将拉普拉斯特征值问题重新写为一阶系统：

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]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：

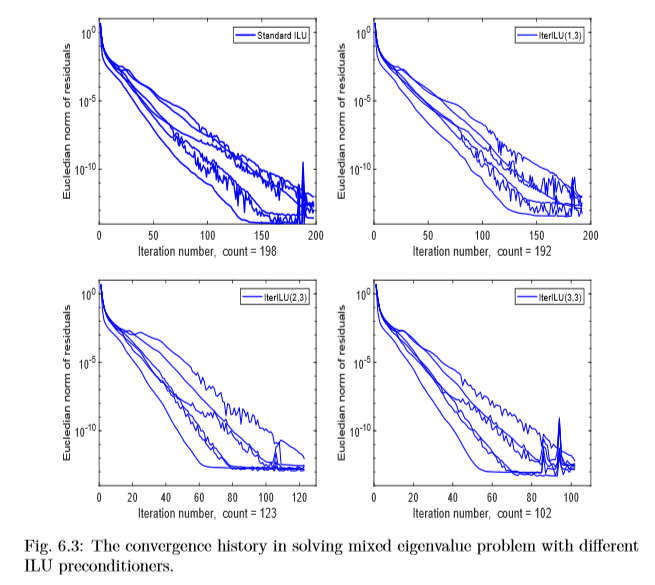
A,M都是埃尔米特矩阵。

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：

有限的特征值通过特征对，，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：

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。

在这个测试例子中，块矩阵被网格数为128\*128,行数为148225行的平方域生成。我们设置为停止准则，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。

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的两个删除原则，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）

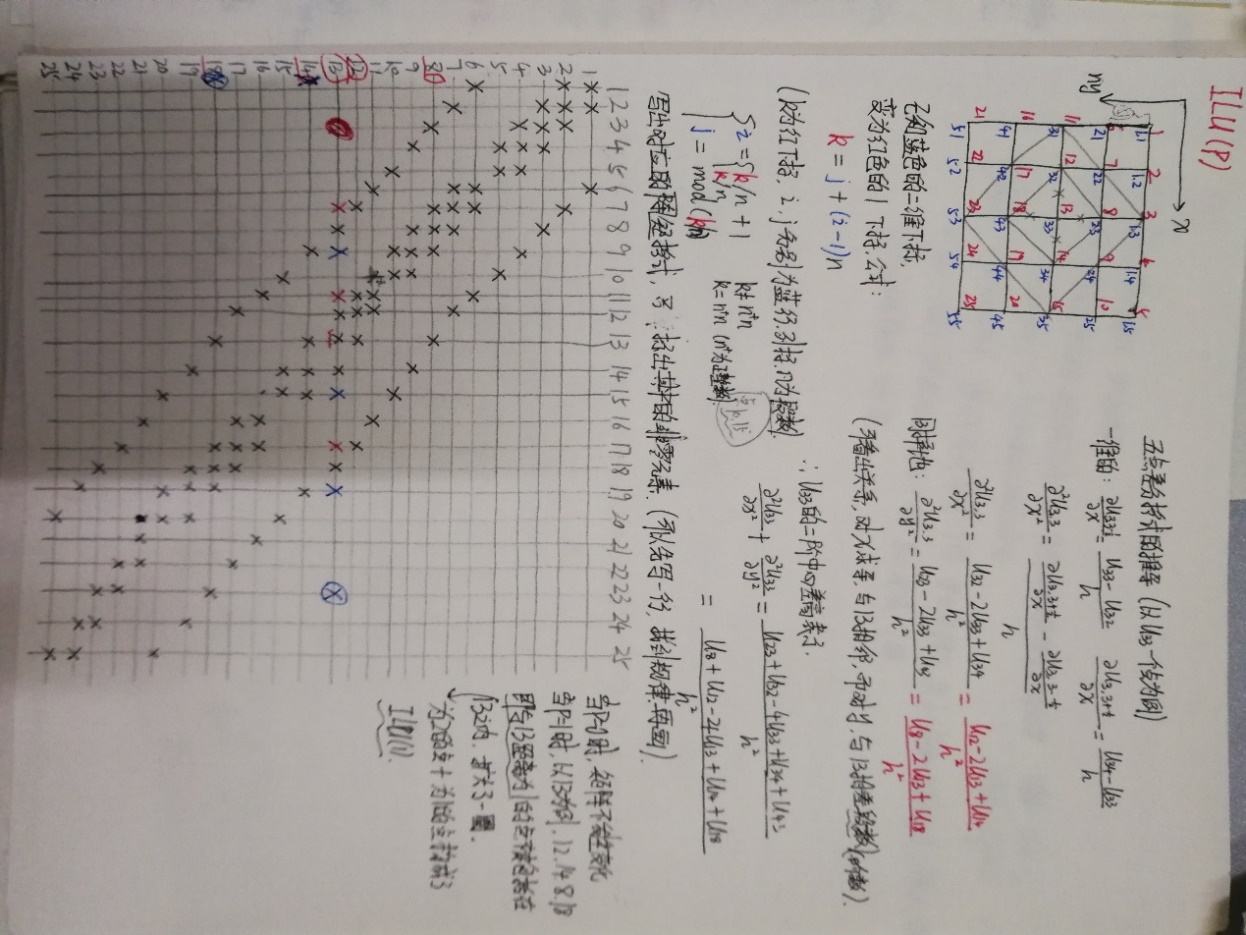
1. ILU(0)指的是矩阵所对应的五点差分格式的矩阵形式。

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





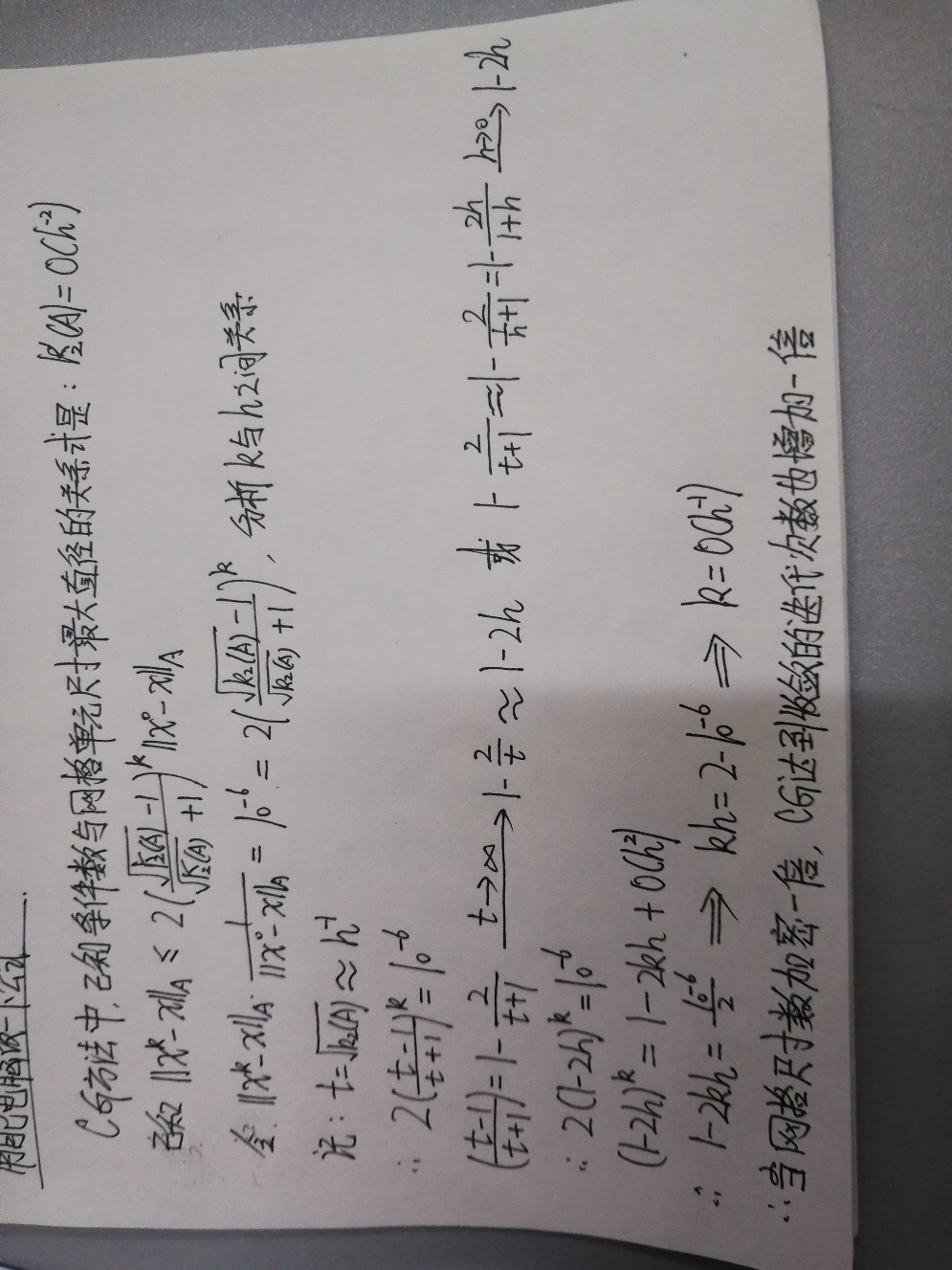


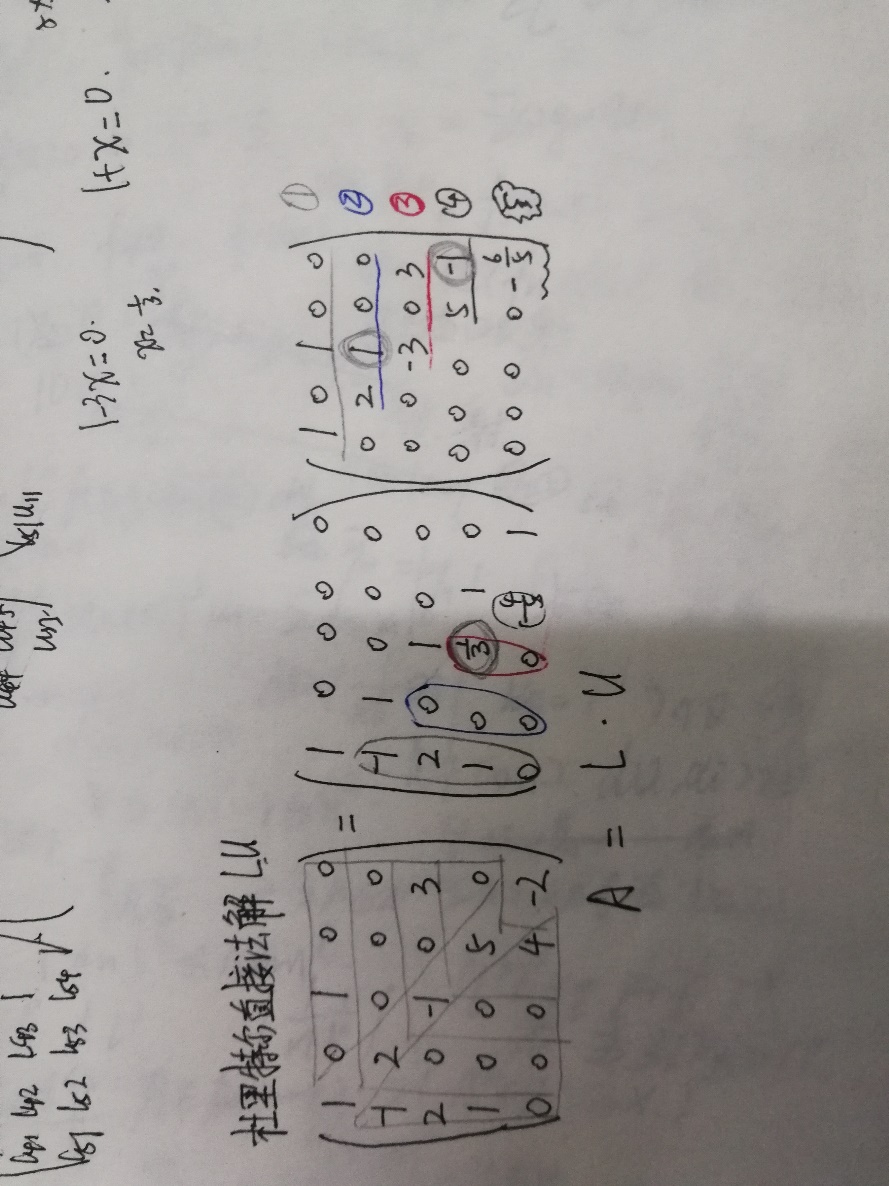
 

由和向量乘的最小非零元素个数决定。

附录3：迭代次数k与网格单元尺寸最大直径的关系

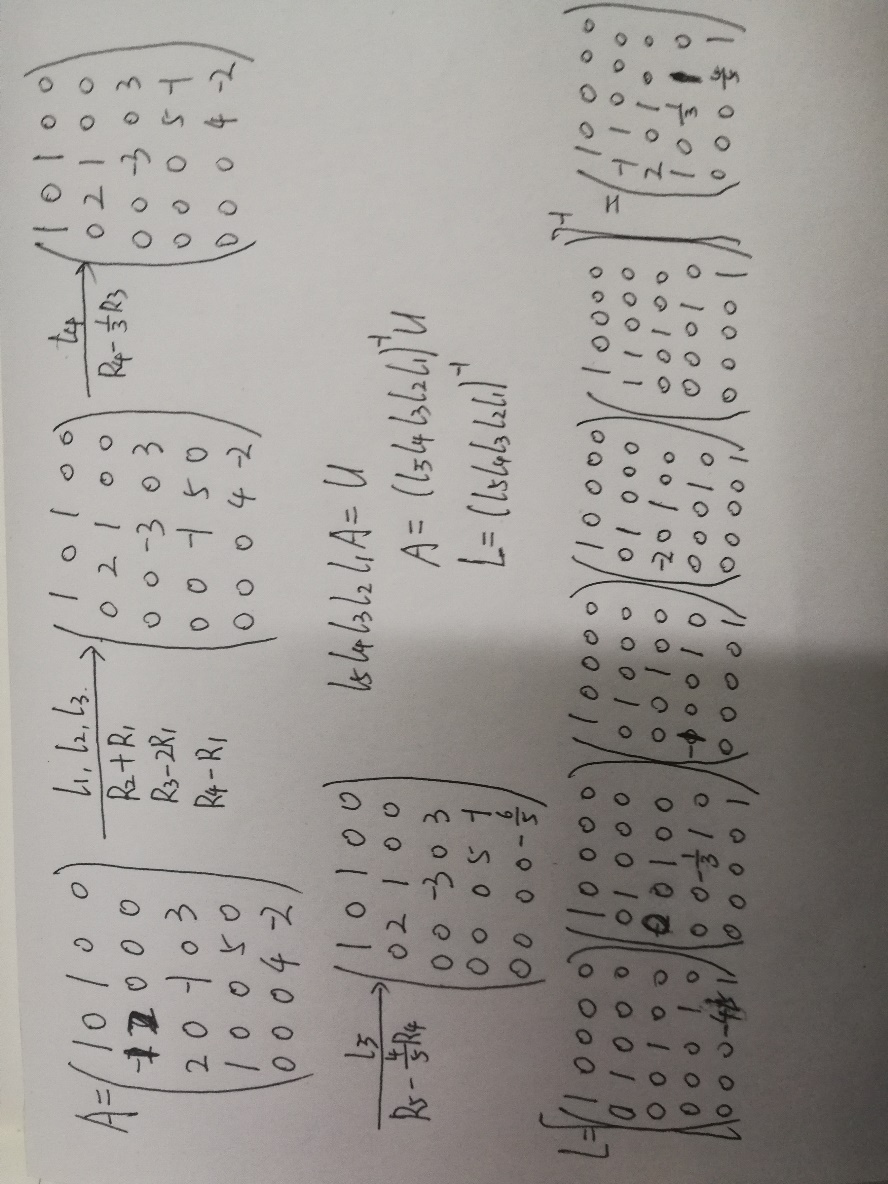
有限元、有限差分的核心思想与定积分类似，都是分割、求和、取极限。



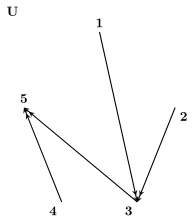
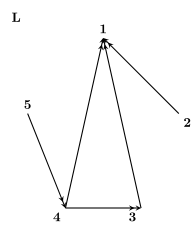
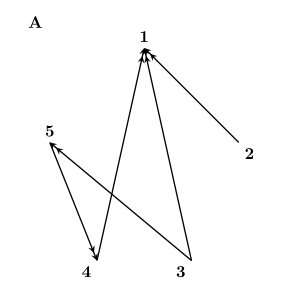
附录4：传统的LU分解法与此论文的迭代法的比较

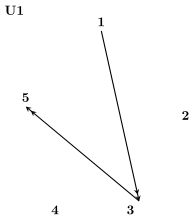
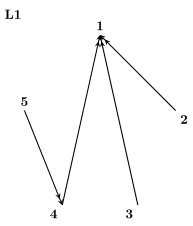
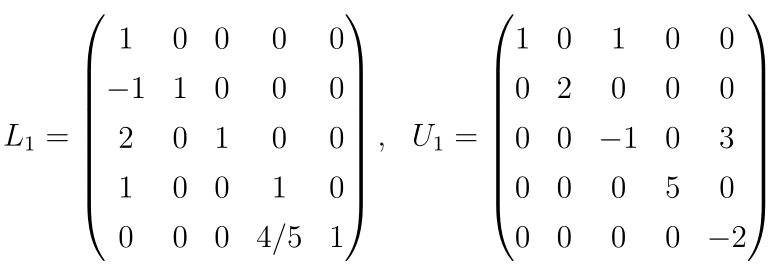
1.杜立特尔直接法求解L和U

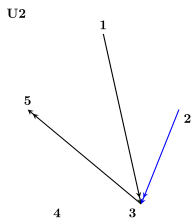
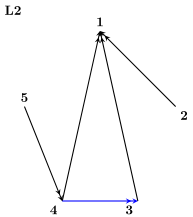
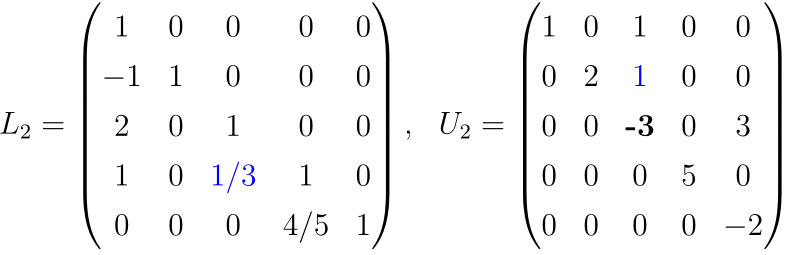
2.高斯消元法求解 L和U

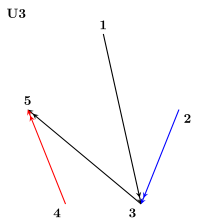
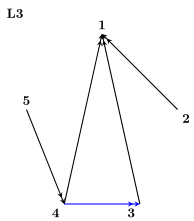
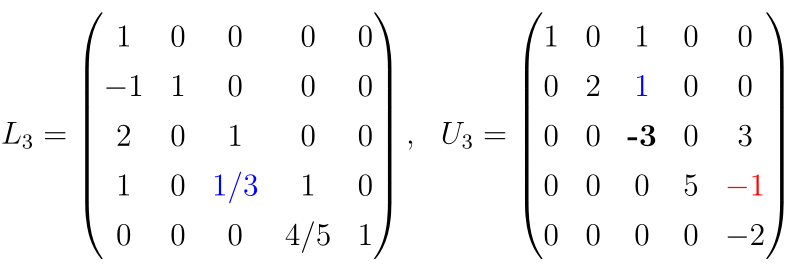


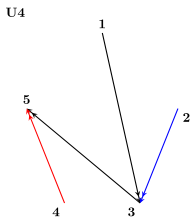
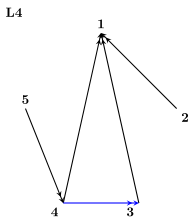
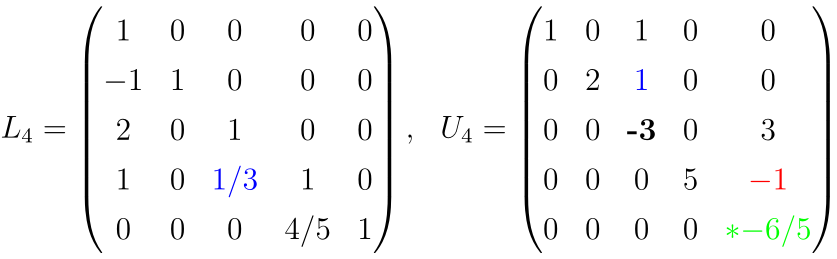
3.本论文的迭代法求解L和U

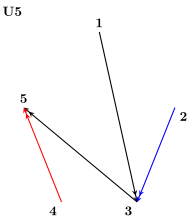
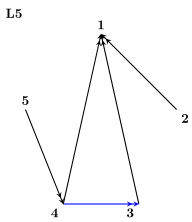
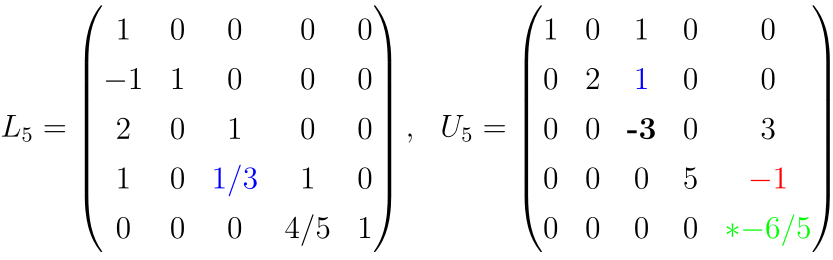


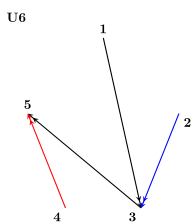
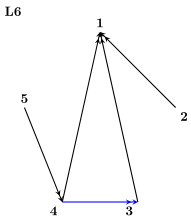
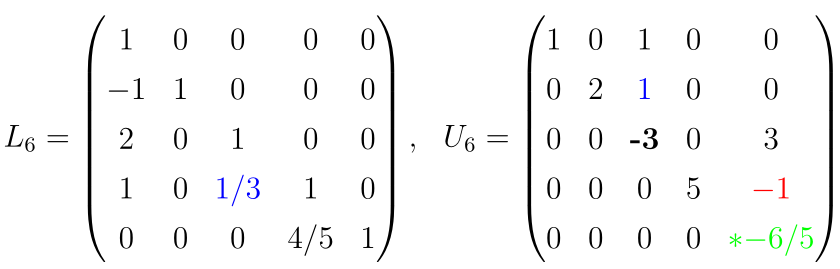


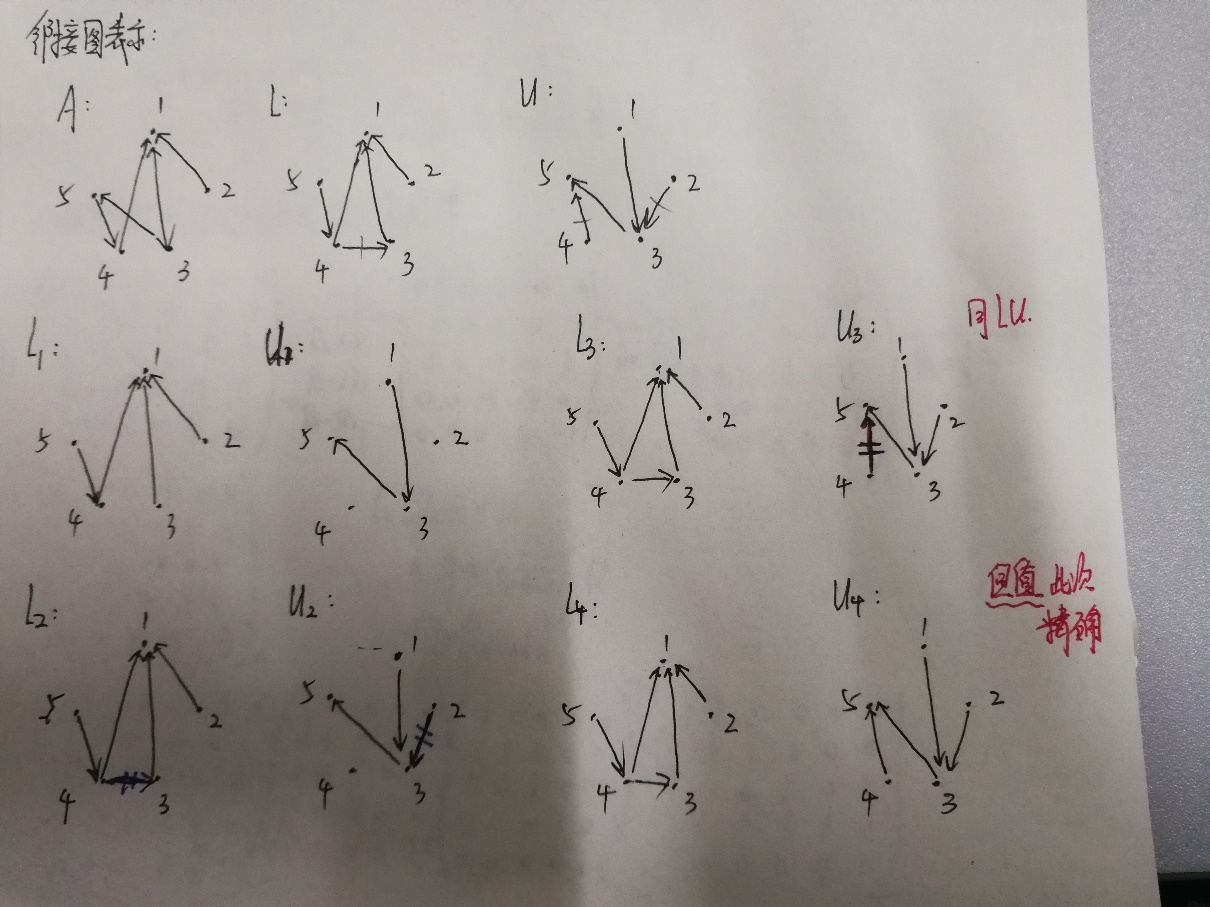












L1：

2

5

4

3

2

5

4

3

