上海交通大学硕士学位论文

电力系统病态潮流计算的高性能实现研究

硕士研究生：张道天

学　　号：1130319024

导　　师：严正教授

申请学位：工学硕士

学　　科：电气工程

所在单位：电子信息与电气工程学院

答辩日期：2016 年 1 月

授予学位单位：上海交通大学

Dissertation Submitted to Shanghai Jiao Tong University
for the Degree of Master

STUDY ON HIGH PERFORMANCE REALIZATION OF POWER SYSTEM ILL-CONDITIONED POWER FLOW CALCULATION

Candidate：	Zhang Daotian
Student ID:	1130319024
Supervisor：	Prof. Yan Zheng
Academic Degree Applied for：	Master of Engineering
Speciality：	Electrical Engineering
Affiliation：	School of Electronic Information and Electrical Engineering
Date of Defence：	Jan,2016
Degree-Conferring-Institution：	Shanghai Jiao Tong University

上海交通大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文《电力系统病态潮流计算的高性能实现研究》，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：张道天

日期：2016年 1月 18日

上海交通大学

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□，在___年解密后适用本授权书。

本学位论文属于

不保密☑。

（请在以上方框内打“√”）

学位论文作者签名：张道天　　指导教师签名：

日期：2016年1月18日　　日期：2016年1月18日

上海交通大学 硕士 学位论文答辩决议书

1130319024

姓 名	张道天	学 号	1130319024	所在学科	电气工程
指导教师	严正	答辩日期	2016-1-18	答辩地点	电信群楼1-334B
论文题目	电力系统病态潮流计算的高性能实现研究				

投票表决结果：3 /3 / 3 （同意票数/实到委员数/应到委员数） 答辩结论：☑通过 ☐未通过

评语和决议：

论文以电力系统病态潮流计算为研究对象，基于 LM 方法以及稀疏矩阵算法，研究了电力系统病态潮流计算的高性能实现，主要成果如下：

从提高病态潮流计算速度的角度出发，分析了 LM 方法求解病态潮流中不同的迭代步求取方案，通过比较两方案的注入元和计算时间，得出直接求解比扩展矩阵求解更为高效；采用改进的 Cholesky 分解算法，进一步提升病态潮流计算的计算速度，通过实际大规模系统的潮流算例，验证了算法的正确性和高效性。

该生在答辩过程中表述清晰，回答问题正确，表明作者已掌握本专业的基础理论和专门知识，具有独立从事科研的能力。经答辩委员会一致表决同意其通过工学硕士学位论文答辩，并建议授予工学硕士学位。

年 月 日

答辩委员会成员	职务	姓名	职称	单位	签名
	主席	王杰	教授	上海交通大学电子信息与电气工程学院(电气系)	
	委员	杨镜非	副教授	上海交通大学电子信息与电气工程学院(电气系)	
	委员	宋依群	副教授	上海交通大学电子信息与电气工程学院(电气系)	
	秘书	许少伦	高级工程师	上海交通大学	

电力系统病态潮流计算的高性能实现研究

摘 要

潮流计算作为电力系统运行控制中的基本分析工具，其结果有助调度人员了解系统的实际运行情况，同时也是稳定计算等后续分析的基础。重负荷、网络参数不合理等因素会导致雅可比矩阵病态，往往使得牛顿法、快速解耦法等常规方法无法收敛，需要采用改进的病态潮流算法。此外，大规模电力系统病态潮流计算的快速、高效实现也是工程应用中的一个重要内容。

本文主要从计算高效性的角度出发，着重研究大规模电力系统病态潮流计算的高性能实现。首先，对主流的病态潮流算法进行总结，通过对比分析选取自适应 LM 方法提升潮流计算的收敛性。随后，从注入元和计算时间两个角度对 LM 方法中迭代步求取的两种方案进行对比。

另一方面，本文通过引入高效的稀疏 Cholesky 分解技术，进一步提升病态潮流计算的计算速度。通过对广泛应用于各领域的超节点 Cholesky 分解算法进行研究，发现其应用于电力系统病态潮流计算时，由于矩阵极度稀疏的特点，难以发挥算法的加速优势。故选用向上看 Cholesky 分解算法，并进一步根据病态潮流计算的应用场景进行算法的改进，采用针对 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵的隐式 Cholesky 分解算法，相比于显式算法能够有效减少冗余计算量。最后，本文在以上研究的基础上设计了电力系统病态潮流计算的高性能实现方案，并完成了病态潮流计算平台的软件开发，为病态潮流问题提供了行之有效的计算工具。

本文利用实际大规模系统的潮流算例，从正确性和高效性角度对病态潮流高性能实现方案进行测试。测试结果表明，自适应 LM 方法能够正确求解良态潮流、病态有解潮流，并能求得病态无解系统的最

小二乘解；而采用隐式 Cholesky 分解算法，在矩阵分解部分可相对显式算法提升 20%的计算速度。对于大规模系统的潮流计算，所开发程序相比于 PSD-BPA 能够缩减近一半的计算时间。

关键词：病态潮流计算；自适应 LM 方法；稀疏技术；
超节点 Cholesky 分解；隐式 Cholesky 分解

STUDY ON HIGH PERFORMANCE REALIZATION OF POWER SYSTEM ILL-CONDITIONED POWER FLOW CALCULATION

ABSTRACT

As the fundamental of power system analysis, power flow calculation can help dispatchers understand the running status of power system. The result of power flow calculation is the basis for other stability calculations. Factors like heavy loads and unreasonable network parameters can cause an ill-conditioned Jacobian matrix, which leads to regular algorithms like Newton-Raphson and fast decoupled method not converging. Thus, algorithms need to be improved to solve the ill-conditioned power flow problems. In addition, the high performance realization of large-scale ill-conditioned power flow calculation is also an important aspect in application.

This thesis studies the high performance realization of power system ill-conditioned power flow calculation concentrating on computational efficiency. Firstly, several mainstream ill-conditioned power flow algorithms are studied and compared, and the self-adaptive LM method is chosen to improve the convergence of power flow calculation. Secondly, two sparse solution methods in LM iterations are compared in terms of fill-ins and computational efficiency.

Moreover, sparse Cholesky factorization is introduced to improve the speed of ill-conditioned power flow calculation. The widely-used supernodal Cholesky factorization is studied. It is found that the supernodal algorithm cannot exploit its advantages because of the extremely sparsity of the matrices, when the algorithm is applied to ill-condition power flow calculation. Thus, up-looking Cholesky is chosen, and the algorithm is improved according to the features of LM method. An implicit Cholesky factorization method for $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ is proposed to improve matrix factorization

speed in LM iterations. With the proposed method, redundant computation in regular methods can be reduced. Finally, the scheme of the high performance realization of power system ill-conditioned power flow calculation is proposed. And the ill-conditioned power flow calculation platform is developed to provide a useful tool for ill-conditioned power flow problem.

The correctness and efficiency of the proposed scheme is tested by large-scale cases. Test results show that self-adaptive LM method can obtain the correct solutions for well-conditioned power flow and feasible ill-conditioned power flow problems. And the method can obtain the least square solutions for infeasible ill-conditioned power flow problems. Test results also show that the implicit Cholesky factorization method can accelerate the speed of factorization by 20%, compared with explicit method. And the developed platform can reduce nearly half of the overall running time compared with PSD-BPA.

KEY WORDS: ill-conditioned power flow, self-adaptive LM method, sparse technology, supernodal Cholesky factorization, implicit Cholesky factorization

目 录

第一章 绪论

1.1 研究的背景及意义

电力系统作为国民经济的重要组成部分，其稳定运行对于经济的发展以及人民的生活质量有重要的影响。为了确保电力系统的安全稳定运行，适当的分析手段必不可少。而潮流计算作为电力系统分析的基石，能够帮助调度人员了解电力系统运行情况，其计算结果更是稳定计算等后续分析手段的基础。

潮流计算由电力系统各节点的给定功率值，求得对应的电压向量，本质而言是求解一组非线性方程组。以极坐标为例，该方程组可表示为[1]：

$$\begin{cases} P_i = V_i \sum_{j \in i} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij}) \\ Q_i = V_i \sum_{j \in i} V_j (G_{ij} \sin\theta_{ij} - B_{ij} \cos\theta_{ij}) \end{cases} (i = 1, 2, ..., n) \qquad (1\text{-}1)$$

式中，P_i、Q_i 分别为节点 i 的注入有功功率和无功功率，V_i 为节点 i 的电压幅值，G_{ij}、B_{ij} 分别为线路 ij 的电导和电纳，θ_{ij} 为节点 ij 间的电压相角差，n 为系统的节点规模。

对于如式(1-1)的潮流方程组的求解，最早采用的是基于导纳的高斯-塞德尔法[2]，该方法原理简单且计算占用的内存较少，然而算法的收敛性较差。为了提升潮流计算的收敛性，基于阻抗的高斯-塞德尔法被提出[3]，该方法采用系统的阻抗矩阵作为方程组求解的系数矩阵，提高了算法的收敛性，然而由于阻抗矩阵的稠密特性，大大增加了计算占用的内存量，限制了求解的系统规模。在 1967 年，由 William F. Tinney 等人提出采用牛顿-拉夫逊(Newton-Raphson, NR)法求解系统潮流[4]，该方法基于系统的导纳阵进行求解，解决了内存占用的问题。同时，该方法的收敛性也在收敛性与计算速度上优于基于阻抗的高斯-塞德尔法，使得大规模电力系统的潮流计算真正具备可实现性。随后，Brian Stott 等又针对电力系统的特性，提出了 PQ 分解法[5]。该方法在计算过程中，每次迭代的系数矩阵固定不变，故其计算速度相较牛顿法有所提高。时至今日，牛顿-拉夫逊法以及 PQ 分解法作为经典的潮流计算方法，凭借良好的计算特性，依旧是各种电力仿真软件中潮流计算模块所采用的基本工具。

然而近年来，随着我国电网的建设、特高压工程的推进以及“华东-华中-华北”的“三华”电网区域间互联，系统的规模越来越大，且电网的运行方式复杂

多变。在对系统采用常规牛顿法进行潮流计算时，时常会出现计算结果不收敛的问题，主要原因是系统的负荷过重接近极限水平，或者网络参数配置不合理，导致牛顿法计算过程中的雅克比矩阵奇异或接近奇异，从而使得迭代过程难以收敛。这样的情况下，系统潮流被称为病态潮流，否则即为良态潮流。病态潮流包括以下两种情况：

1）潮流方程本身无可行解，称为病态无解情况；

2）潮流方程本身有可行解，然而由于牛顿法迭代过程中雅克比矩阵接近奇异，使得计算无法收敛到该可行解，称为病态有解情况。

潮流计算采用常规牛顿-拉夫逊法不收敛时，得到的发散结果与真实情况相去甚远，无法为调度人员及规划人员提供任何参考。针对这种问题，现在一般的应对方法是根据经验试探性调整规划方式或数据，从而得到收敛的潮流解。这种方法工作量大，效率很低。因此，为了有效解决病态潮流的求解问题，亟需从原理上对潮流计算的算法进行进一步研究。

此外，由于电网规模的逐渐扩大，潮流计算所需的时间也逐渐增加。而与此同时，随着数值算法以及计算机技术的发展，为潮流计算的提速提供了可能。因此，如何在采用合适的算法后正确求解病态潮流的基础上，进一步依靠计算机软件实现电力系统病态潮流的高性能仿真，也是一个重要而具有实用意义的课题。

1.2 国内外研究现状

本文主要研究电力系统病态潮流计算的高性能实现，结合了电力系统病态潮流算法研究以及电力系统高性能仿真研究两个方面。下面分别对这两个方面的研究现状进行总结。

1.2.1 电力系统病态潮流计算研究现状

从采用数字计算机的潮流计算方法受到研究者关注开始，计算的收敛性问题就始终是学者们关注的重要问题。20 世纪 80 年代时，日本学者田村康夫就电力系统的多解问题进行了研究，研究发现随着负荷的增加，系统潮流解个数成对减少，直到无解[6][7]，从机理上解释了病态潮流出现的原因。

与此同时，有大量的学者针对病态潮流问题提出解决方案。其中，最优乘子法的发展最为广泛。1981 年，岩本伸一等人提出了最优乘子法[8]。它基于直角坐标系的潮流方程，通过引入二阶展开项，在当前迭代步方向搜索最优步长，改善

了雅克比矩阵病态时步长过大的问题。1994 年王宪荣等人提出了极坐标系下的最优乘子法[9]，不仅能够提升潮流计算的收敛性，也一定程度缓解了牛顿法对初值的敏感性，取得了比较好的结果。随后，Joseph E. Tate 等人对直角坐标下与极坐标下的最优乘子法进行了对比[10]，发现直角坐标下的最优乘子法其效率不如极坐标系下的最优乘子法。

近年来，有人将张量法引入到潮流计算中，在处理重负荷引起的病态潮流问题中取得了一定的成果。张量法最早由 Robert B. Schnabel 等人于 1984 年提出[11]，用于提升非线性方程组的收敛性。该方法于 1998 年推广至大规模稀疏系统[12]，为其应用于电力系统潮流计算提供了条件。文献[13]首次将张量法引入潮流计算中，提出了直角坐标下的张量法计算模型，通过算例验证了其对潮流计算鲁棒性的提升效果。随后，文献[14]将直角坐标下的张量法推广到极坐标，算例验证了其对于重负荷导致潮流病态问题具有比较好的提升收敛性效果。

此外，有人采用优化技术处理病态潮流问题。文献[15]将松弛量引入交直流潮流方程组中，并以松弛量平方和最小作为目标，建立了非线性规划模型，并通过内点法进行求解，通过算例验证方法对于潮流病态时的有效性。也有学者采用智能优化算法[16]，对于系统潮流病态有较强的处理能力。然而智能算法的计算量非常大，导致方法缺乏实用性。

Levenberg-Marquardt(LM)方法作为经典的非线性最小二乘算法，也被应用到病态潮流的求解中。文献[17]首先将 LM 方法引入潮流求解中，验证了算法对于提升潮流收敛性的效果。文献[18]采用自适应 LM 方法，通过引入自适应阻尼因子，提升算法的自调节能力，进一步提高算法的收敛性和收敛速度。

可以看出，国内外学者对于提升潮流病态时收敛性的问题进行了大量的研究，成果众多。而这些成果主要从数学机理出发，试图解决常规牛顿法在雅克比矩阵接近奇异时迭代不收敛的问题。然而，这些研究成果大多没有结合电力系统稀疏性的特点以及先进的计算机科学技术，探讨算法更高效的实现方法。故从计算速度上，这些求解病态潮流的算法还有进一步提速的空间。

1.2.2 电力系统高性能仿真研究现状

随着电力系统规模的增加以及模型的复杂化，仿真所需的时间随之增加。与此同时，在线监测又对计算的实时性提出了要求。为了使电力系统仿真计算过程更加快速、高效，也有大量的学者致力于电力系统高性能仿真的研究。

由于电力系统高度稀疏的特性，在电力系统仿真计算的过程中，往往计算的

核心问题会归结为对稀疏矩阵的处理。因此，为了提升电力系统仿真计算的速度，主流的思路是根据电力系统的稀疏特性，采用高效的稀疏技术对计算进行加速。文献[19]在静态安全分析中引入稀疏技术，在 N-1 安全分析中利用因子表路径树，减少了分析中对因子表的修改，并利用快速前代/回代，进一步减少计算时间。文献[20]在潮流计算中利用二维链表存储稀疏矩阵，并用 LU 扩展的方法来处理矩阵分解过程中产生的注入元，从而提升潮流计算的计算效率。文献[21]基于图论理论，分析了潮流计算中矩阵符号分解的方法，从而减少计算过程中的注入元，减少计算时间。文献[22]在电力系统状态估计中引入了稀疏技术，利用稀疏向量法以及符号因子化等方法，提升计算的效率。文献[23]在潮流计算中以消去树理论为基础，实现了 LU 分解中的快速前代，同时采用数组形式存储稀疏矩阵，减少了计算过程中的内存占用以及总的计算时间。

近年来，随着芯片制造技术遇到瓶颈，计算机中央处理器的核心频率增长受到限制。为了进一步提升芯片的性能，Intel 等芯片厂商开始采用多核心的方式来弥补单核心处理器主频的停滞不前。与此同时，NVIDIA 公司也针对图形处理器推出了统一计算设备架构，使得图形处理器在通用计算中有了用武之地。CPU、GPU 等硬件技术的发展，使得程序的并行化称为可能，大大提升了程序的执行效率。得益于多核、众核处理器技术，越来越多的学者将电力系统仿真的目光放在了仿真算法的并行化上。

文献[24]探讨了电力系统仿真中常用的 LU 分解的并行方案，并在 60 节点的小规模系统中验证了并行效果。文献[25]中将潮流计算的矩阵化为对角块加边的形式，并在此基础上对不同对角块进行并行计算，取得了较好的效果。文献[26]探讨了嵌套式的对角块加边形式，在文献[25]的基础上进一步提升了并行的效果。分块的形式为电力系统仿真的并行化提供了基本的手段。在此基础上，大量学者分别对电力系统潮流计算、暂态稳定计算和小干扰计算的并行化方案进行了研究。

在电力系统潮流计算的并行化研究方面，文献[27]通过将系统划分为多个子网后进行并行计算，双核 CPU 上加速比可达到 1.794。文献[28]则采用 GPU 对潮流计算进行并行化处理，对于系统规模为 300 的系统，加速比可达到 4.9。

在电力系统暂态稳定的并行化研究方面，文献[29]、[30]利用多核 CPU 并行技术对电力系统暂态稳定仿真进行加速。文献[31]采用众核 GPU 对基于广义最小残差法的暂态稳定仿真进行并行化设计，达到 3.3 的加速比。文献[32]同样采用众核 GPU 并行技术，对基于直接法的稀疏线性方程组求解进行了并行化处理，

并应用于暂态稳定仿真中。

在电力系统小干扰稳定的并行化研究方面，文献[33]利用 PC 机群对小干扰稳定并行中基于 Rayleigh 商迭代法的部分特征值求取进行并行化处理，实现仿真的加速。文献[34]则在小干扰稳定计算中采用 QR 方法求取全特征值，并利用 GPU 实现 QR 方法的并行化计算，取得了良好的加速效果。

可以看出，对于电力系统高性能仿真研究，通常需要根据不同的仿真计算以及具体算法的特性，有针对性地进行方案选择和设计。现阶段，对于病态潮流计算这一领域的高性能仿真方案，尚缺乏相关的研究成果。为了在正确求解病态潮流的同时保证仿真算法的高效性，本文将对病态潮流的高性能实现方案进行研究。

1.3 本文的主要研究内容

本文以电力系统病态潮流计算为研究对象，利用稀疏技术以及并行技术等手段，对以下内容进行研究。

（1）电力系统病态潮流算法分析比较。在了解电力系统病态潮流产生机理的基础上，对各种电力系统病态潮流算法的数学原理进行学习，并进一步分析其优缺点，根据比较的结果选取 LM 方法作为病态潮流求解方法。针对 LM 方法迭代步中不同的稀疏求解方案，通过大规模电力系统实例比较直接求解方案与扩展矩阵方案，并分析差异原因。

（2）稀疏矩阵分解算法适用性分析。学习基本的稀疏技术，并针对 LM 方法中的 Cholesky 分解步骤研究稀疏矩阵 Cholesky 分解算法。比较分析向上看 Cholesky 分解算法和超节点 Cholesky 分解算法的原理，并利用实际的系统数据分析其在电力系统领域的适用性，从而选取恰当的数值算法。

（3）电力系统病态潮流高性能实现方案构建。在基于 LM 方法的病态潮流计算步骤中，根据实际的应用情况，对核心求解步：矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 的 Cholesky 分解进行改良，采用隐式 Cholesky 分解方法，有效地提升算法的效率，缩短计算时间。在此基础上，构建电力系统病态潮流高性能实现方案，并基于此实现方案，完成电力系统病态潮流计算平台的软件开发，为电力系统病态潮流计算提供正确且高效的求解工具。

（4）算例测试与分析。通过实际系统潮流数据的仿真测试，验证所采用算法以及高性能实现方案的正确性，并进一步通过算法的计算时间比较，考察本文

所采用的隐式 Cholesky 方法在提升潮流计算速度中的作用。

本文共分为七章，其中第二至第六章为主体部分，分别为病态潮流算法比较分析，稀疏技术及稀疏分解算法研究，病态潮流的高性能实现设计，以及算例测试与分析。各章的内容概述如下：

第一章介绍了本论文的研究背景，阐述了研究电力系统病态潮流高性能实现的意义，同时对现阶段国内外的研究现状进行了归纳与总结。

第二章首先对主流的病态潮流算法进行了介绍，并从计算量和收敛性的角度对几个算法进行比较，选取特性较优的自适应 LM 方法作为病态潮流高性能实现的算法基础。随后，对 LM 方法中核心的迭代步求解进行研究，分析了两种不同的计算方案，通过实际的算例比较两方案的计算效率，选取了更为高效的直接分解方案作为迭代步的求解方案。

第三章从电力系统相关计算中矩阵稀疏的特性出发，对稀疏技术及算法进行了研究。首先，对稀疏技术的研究历史进行介绍，并介绍了稀疏技术中基础的稀疏存储以及稀疏三角分解。随后，针对 LM 方法求解病态潮流中的实际需要，研究了稀疏 Cholesky 分解的实现，在数值分解部分，对主流的向上看 Cholesky 分解即超节点 Cholesky 分解算法进行介绍。并通过实际系统数据的测试以及与其他领域矩阵的对比，对两种数值算法在电力系统领域的适用性进行分析。

第四章在病态潮流算法以及稀疏技术的基础上研究了电力系统病态潮流计算的高性能实现方案。首先，根据 LM 方法求解迭代步的计算公式特征，对向上看 Cholesky 分解流程进行改进，采用隐式 Cholesky 分解算法，减少冗余计算量，提升计算效率，并进一步在隐式分解算法的基础上设计病态潮流计算的高性能实现方案。另一方面，对病态潮流计算平台的开发进行了论述，从软件实现的角度验证了方案的可实现性，并为病态潮流计算问题提供高效、可靠的实用工具。

第五章对第四章提出的电力系统病态潮流高性能实现方案进行正确性与高效性测试。使用实际的大规模电力系统数据，测试自适应 LM 方法求解良态/病态潮流的正确性和隐式 Cholesky 分解算法的计算效率。

第六章总结全文的研究工作，并根据本文的内容，提出后续可以进一步研究的内容。

第二章 主流病态潮流算法比较分析

2.1 引言

现阶段，对于因重负荷、网络参数不合理等因素所导致的病态潮流问题，主流的解决方案有：最优乘子法、张量法、优化算法、LM 方法等。本章首先对这四种方法的数学原理进行介绍，通过比较不同方法的特性，选取恰当的病态潮流算法。随后，对病态潮流算法中核心解算步骤的实现方案进行研究，通过实际的系统数据测试方案的效果，并分析差异原因，选取合适的方案，作为后续高性能实现的基础。

为了表达的简洁性，下面将形如式(1-1)的潮流方程简记为

$$\boldsymbol{F}(\boldsymbol{x})=0 \tag{2-1}$$

式中 $\boldsymbol{x}$ 为电力系统的状态变量$[\boldsymbol{V},\boldsymbol{\theta}]^{\mathrm{T}}$。

2.2 病态潮流算法数学原理

2.2.1 最优乘子法

最优乘子法最早由岩本伸一等人于 1981 年提出[8]，其基于直角坐标系的潮流方程，一定程度改善了雅克比矩阵病态时步长过大的问题。随后在 1994 年，王宪荣等人将方法推广至极坐标系下[9]。最优乘子法的原理是将最优乘子引入到常规牛顿法求解方程(2-1)中，限制迭代的步长，从而防止雅克比矩阵接近奇异时修正量过大而产生的振荡发散问题。下面假设第 k 次迭代所得的修正步长为 $\Delta\boldsymbol{x}^{(k)}$，所引入的最优乘子为 $\mu^{(k)}$，那么最优乘子法的修正方程则可以表示为：

$$\boldsymbol{x}^{(k+1)}=\boldsymbol{x}^{(k)}+\mu^{(k)}\Delta\boldsymbol{x}^{(k)} \tag{2-2}$$

式中最优乘子 $\mu^{(k)}$的值满足沿该迭代步方向，修正后的 $\boldsymbol{x}$ 能够使潮流功率的偏差值最小，即 $\mu^{(k)}$满足：

$$\min\left\|\boldsymbol{F}(\boldsymbol{x}^{(k)}+\mu^{(k)}\Delta\boldsymbol{x}^{(k)})\right\|_2 \tag{2-3}$$

根据文献[8]，在直角坐标下，最优乘子可通过以下方法进行计算：

$$\begin{aligned}\boldsymbol{F}(\boldsymbol{x}+\mu\Delta\boldsymbol{x})&=\boldsymbol{F}(\boldsymbol{x})+\mu\boldsymbol{J}(x)\Delta\boldsymbol{x}-\mu^2\boldsymbol{F}(\Delta\boldsymbol{x})\\&=\boldsymbol{a}+\boldsymbol{b}\mu+\boldsymbol{c}\mu^2\end{aligned} \tag{2-4}$$

其中，$\boldsymbol{a}=\boldsymbol{F}(\boldsymbol{x})$，$\boldsymbol{b}=\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}$，$\boldsymbol{c}=-\boldsymbol{F}(\Delta\boldsymbol{x})$。将式(2-4)代入式(2-3)，则可表示为

$$\min\left\|\boldsymbol{a}+\boldsymbol{b}\mu+\boldsymbol{c}\mu^2\right\|_2=\min\frac{1}{2}\sum_{i=1}^{n}(a_i+b_i\mu+c_i\mu^2) \tag{2-5}$$

式(2-5)的最优解可通过目标函数对 μ 的偏导数为 0 进行求取，即

$$g_0+g_1\mu+g_2\mu^2+g_3\mu^3=0 \tag{2-6}$$

其中，

$$\begin{cases} g_0=\boldsymbol{a}^T\boldsymbol{b} \\ g_1=\left\|\boldsymbol{b}\right\|_2^2+2\boldsymbol{a}^T\boldsymbol{c} \\ g_2=3\boldsymbol{b}^T\boldsymbol{c} \\ g_3=2\left\|\boldsymbol{c}\right\|_2^2 \end{cases} \tag{2-7}$$

可以看出，式(2-6)是一个变量为 μ 的一元三次方程，可以采用 Cardan 公式求出该方程的解，即最优乘子 μ。

根据文献[9]，极坐标下的最优乘子法可表示为：

$$\boldsymbol{F}(\boldsymbol{x}+\mu\Delta\boldsymbol{x})=\boldsymbol{F}(\boldsymbol{x})+\mu\boldsymbol{J}(x)\Delta\boldsymbol{x}+\boldsymbol{H}(\mu\Delta\boldsymbol{x}) \tag{2-8}$$

其中，$\boldsymbol{H}(\boldsymbol{x})$为对潮流方程在新的迭代点处泰勒展开后的非线性总项。由于 $\Delta\boldsymbol{x}$ 首先通过牛顿法求得的，故

$$\boldsymbol{H}(\Delta\boldsymbol{x})=\boldsymbol{F}(\boldsymbol{x}+\Delta\boldsymbol{x}) \tag{2-9}$$

认为二阶项在 $\boldsymbol{H}(\mu\Delta\boldsymbol{x})$中占主导作用，则

$$\boldsymbol{H}(\mu\Delta\boldsymbol{x})\approx\mu^2\boldsymbol{H}(\Delta\boldsymbol{x})=\mu^2\boldsymbol{F}(\boldsymbol{x}+\Delta\boldsymbol{x}) \tag{2-10}$$

将式(2-10)带入式(2-8)，则可得到

$$\begin{aligned}\boldsymbol{F}(\boldsymbol{x}+\mu\Delta\boldsymbol{x})&=\boldsymbol{F}(\boldsymbol{x})-\mu\boldsymbol{F}(\boldsymbol{x})+\mu^2\boldsymbol{F}(\boldsymbol{x}+\Delta\boldsymbol{x})\\&=\boldsymbol{a}-\boldsymbol{a}\mu+\boldsymbol{c}\mu^2\end{aligned} \tag{2-11}$$

其中 $\boldsymbol{a}=\boldsymbol{F}(\boldsymbol{x})$， $\boldsymbol{c}=\boldsymbol{F}(\boldsymbol{x}+\Delta\boldsymbol{x})$。

同样地，通过目标函数对 μ 的偏导数为 0，可以求得最优乘子，即

$$g_0+g_1\mu+g_2\mu^2+g_3\mu^3=0 \tag{2-12}$$

其中

$$\begin{cases} g_0=-\left\|\boldsymbol{a}\right\|_2^2 \\ g_1=\left\|\boldsymbol{a}\right\|_2^2+2\boldsymbol{a}^T\boldsymbol{c} \\ g_2=-3\boldsymbol{a}^T\boldsymbol{c} \\ g_3=2\left\|\boldsymbol{c}\right\|_2^2 \end{cases} \tag{2-13}$$

与直角坐标下的最优乘子法类似，式(2-13)是一个变量为 μ 的一元三次方

程，可以采用 Cardan 公式求出该方程的解，即最优乘子 μ。

由此，最优乘子法的计算流程图如下：

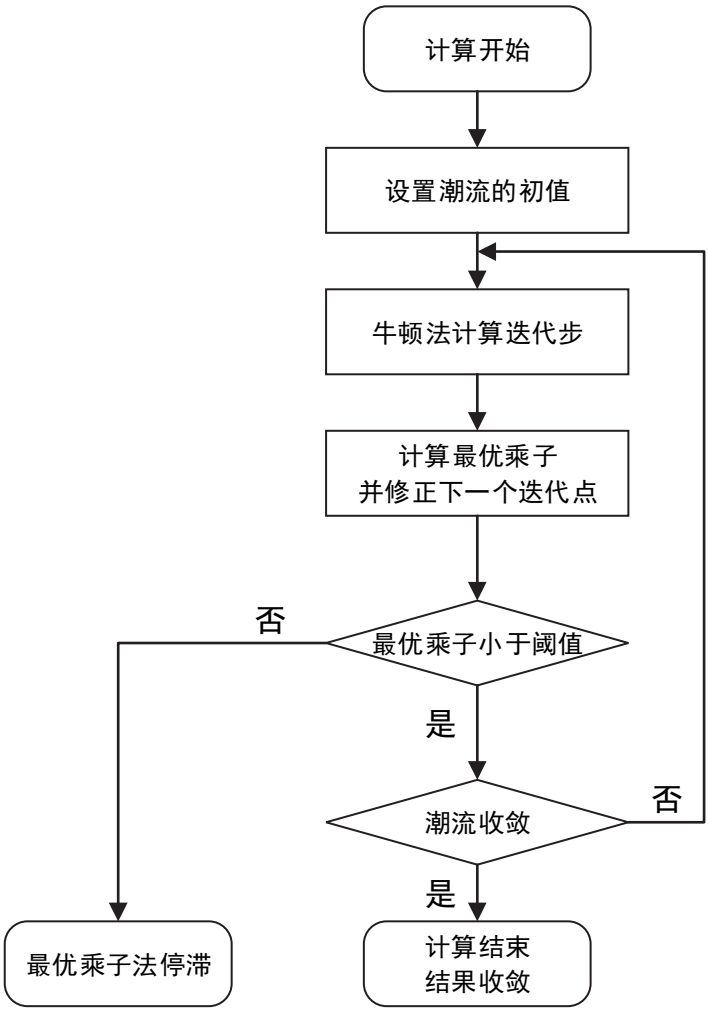


图 2-1 最优乘子法计算流程图

Fig. 2-1 Flow chart of the optimal multiplier method

最优乘子法具有以下特点：

1）当迭代点接近潮流解时，最优乘子 μ 接近 1。而当雅克比矩阵的条件数很大，接近奇异时，μ 接近 0，从而保证不会因为牛顿步过大而产生振荡；

2）算法能够在常规的基于牛顿-拉夫逊的潮流程序基础上进行修改，较为易于实现；

3）若当最优乘子法停滞的时候，潮流方程的功率偏差依旧比较大，那么所得到的近似解就不具备参考价值。

2.2.2 张量法

张量法是用于计算含二阶项非线性方程的算法，最早于 1984 年由 Robert B. Schnabel 等人提出[11]，并应用于计算雅克比矩阵为稠密的非线性方程组。该方法于 1998 年推广至大规模稀疏系统[12]，从而为其应用于电力系统潮流计算奠定了基础。在 2008 年，学者 R. S. Salgado 首次将基于插值的张量法引入潮流计算中，并提出了直接张量法。

基于插值的张量法是通过插值法来近似计算潮流方程的二阶泰勒展开项，从而计算迭代步。首先对式(2-1)做泰勒展开，则

$$\boldsymbol{F}(\boldsymbol{x}+\Delta\boldsymbol{x})=\boldsymbol{F}(\boldsymbol{x})+\boldsymbol{J}(x)\Delta\boldsymbol{x}+\frac{1}{2}\Delta\boldsymbol{x}^{T}\boldsymbol{T}\Delta\boldsymbol{x} \tag{2-14}$$

其中 $\boldsymbol{J}(\boldsymbol{x})$为当前的雅克比矩阵，$\Delta\boldsymbol{x}$ 为迭代步，$\boldsymbol{T}$ 是 $\boldsymbol{F}(\boldsymbol{x})$的二阶导数。

令 $\boldsymbol{s}_k=\boldsymbol{x}^{(k-1)}-\boldsymbol{x}^{(k)}$为之前每次迭代的增量，那么矩阵 $\boldsymbol{T}$ 可以表示为

$$T=\sum_{k=1}^{p}\boldsymbol{s}_k\boldsymbol{a}_k\boldsymbol{s}_k^{T} \tag{2-15}$$

其中，$\boldsymbol{a}_k^{T}$ 是矩阵 $\boldsymbol{A}$ 的第 k 列，而 $\boldsymbol{A}=\boldsymbol{Z}\boldsymbol{M}^{-1}$。矩阵 $\boldsymbol{Z}$ 与 $\boldsymbol{M}$ 的定义可参见文献[14]。

将式(2-15)代入式(2-14)，可以得到

$$\boldsymbol{F}(\boldsymbol{x})+\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x}+\frac{1}{2}\sum_{k=1}^{p}\boldsymbol{a}_k(\Delta\boldsymbol{x}^{T}\boldsymbol{s}_k)^{2}=0 \tag{2-16}$$

如果雅克比矩阵 $\boldsymbol{J}(\boldsymbol{x})$非奇异，则对式(2-16)左右同时乘以 $\boldsymbol{s}_i^{T}\boldsymbol{J}(\boldsymbol{x})^{-1}$，并设 $\beta_i^{T}=\boldsymbol{s}_i^{T}\Delta\boldsymbol{x}$，则

$$\boldsymbol{s}_i^{T}\boldsymbol{J}(\boldsymbol{x})^{-1}\boldsymbol{F}(\boldsymbol{x})+\beta+\frac{1}{2}\boldsymbol{s}_i^{T}\boldsymbol{J}(\boldsymbol{x})^{-1}\boldsymbol{a}\beta^{2}=0 \tag{2-17}$$

将式(2-17)称为张量方程，若张量方程存在实数解，则迭代步则通过下式求得

$$\Delta\boldsymbol{x}=-\boldsymbol{J}(\boldsymbol{x})(\boldsymbol{F}(\boldsymbol{x})+\frac{1}{2}\boldsymbol{a}\boldsymbol{\beta}^{2}) \tag{2-18}$$

若张量方程不存在实数解，则文献[14]通过正交变换，求得迭代步为

$$\Delta\boldsymbol{x}=-\boldsymbol{J}(\boldsymbol{x})(\boldsymbol{F}(\boldsymbol{x})+\frac{1}{2}\boldsymbol{a}\boldsymbol{\beta}+\boldsymbol{J}(\boldsymbol{x})^{-T}\boldsymbol{S}\boldsymbol{W}^{-1}\boldsymbol{q}(\beta)) \tag{2-19}$$

其中 $\boldsymbol{W}$ 和 $\boldsymbol{q}(\beta)$的定义可参见文献[14]。

而直接张量法通过假设$\|\Delta\boldsymbol{x}_N\|_2 \gg \|\Delta\boldsymbol{x}_T\|_2$，近似计算张量法迭代步。令 $\Delta\boldsymbol{x}=\Delta\boldsymbol{x}_N+\Delta\boldsymbol{x}_T$，则代入式(2-14)后可以得到

$$\boldsymbol{F}(\boldsymbol{x})+\boldsymbol{J}(x)(\Delta\boldsymbol{x}_N+\Delta\boldsymbol{x}_T)+\frac{1}{2}(\Delta\boldsymbol{x}_N+\Delta\boldsymbol{x}_T)^T\boldsymbol{T}(\Delta\boldsymbol{x}_N+\Delta\boldsymbol{x}_T)=0 \qquad (2\text{-}20)$$

由于假设$\|\Delta\boldsymbol{x}_N\|_2 \gg \|\Delta\boldsymbol{x}_T\|_2$，式(2-20)可以近似求出

$$\Delta\boldsymbol{x}_T=\boldsymbol{J}(x)^{-1}(\frac{1}{2}\Delta\boldsymbol{x}_N{}^T\boldsymbol{T}\Delta\boldsymbol{x}_N) \qquad (2\text{-}21)$$

由此，张量法计算流程图如下：

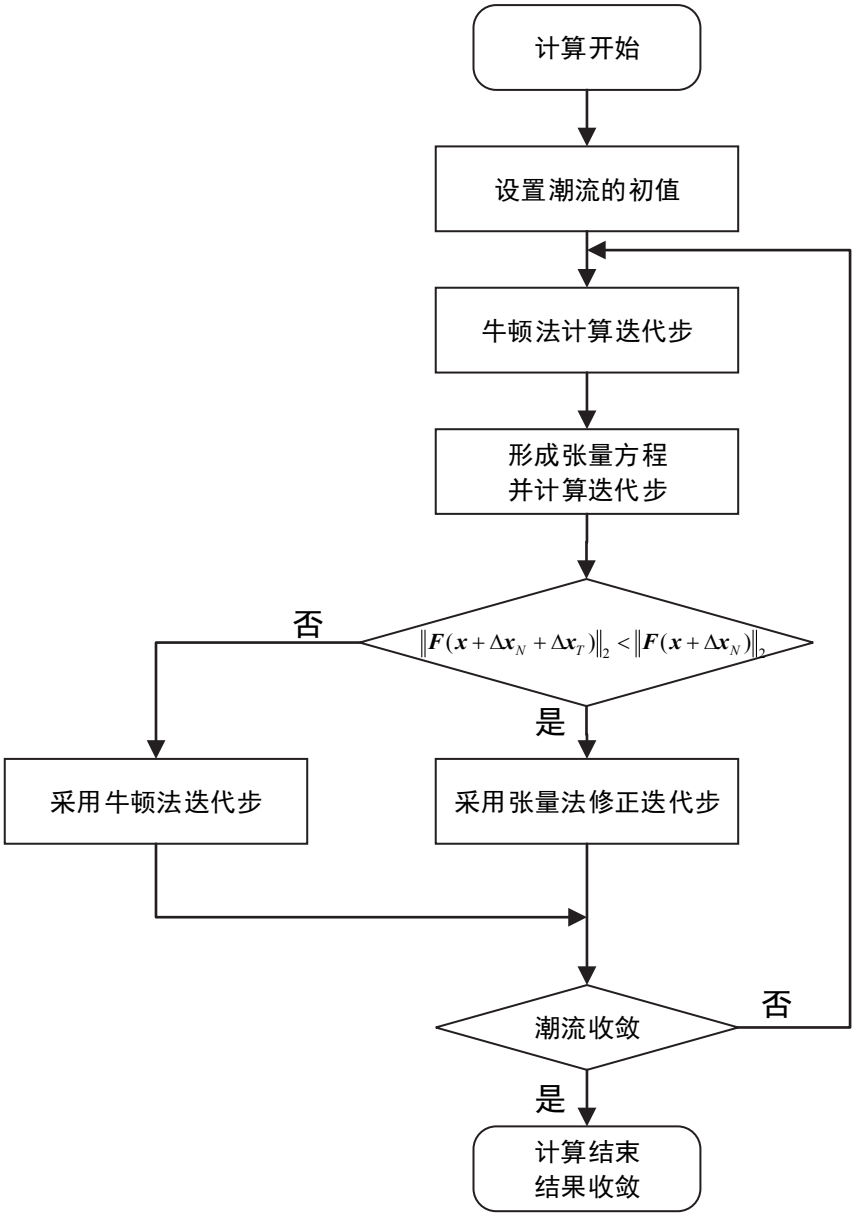


图 2-2 张量法的计算流程图

Fig. 2-2 Flow chart of the tensor method

张量法的特点是：算法的迭代步修正牛顿法的迭代步方向，收敛性优于最优

乘子法，而张量步的存在使计算不会振荡。

2.2.3 优化算法

通过优化算法求解病态潮流，本质上就是通过在潮流方程中引入松弛量，从而把潮流计算问题转换为求取松弛量平方和的极小值问题。若以极坐标下的潮流方程为基础，则规划问题可表示为：

$$\begin{aligned} &\min \sum \boldsymbol{\varepsilon}^T \boldsymbol{\varepsilon} \\ &s.t.\ \ \boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{\varepsilon} \end{aligned} \tag{2-22}$$

其中，$\boldsymbol{\varepsilon}$ 为功率方程的偏差值，$\boldsymbol{F}(\boldsymbol{x})$的值参照式(1-1)求出。分别对约束中的有功与无功引入拉格朗日乘子，则问题可转换为

$$\min f(\boldsymbol{x},\boldsymbol{\varepsilon}) = \sum_i \boldsymbol{\varepsilon}^T \boldsymbol{\varepsilon} + \sum_i (\mu_i^P(\Delta P_i - \varepsilon_i^P) + \mu_i^Q(\Delta Q_i - \varepsilon_i^Q)) \tag{2-23}$$

式(2-23)的极值条件为：

$$\begin{cases} \dfrac{\partial f}{\partial \mu_i^P} = \Delta P_i - \varepsilon_i^P = 0 \\ \dfrac{\partial f}{\partial \mu_i^Q} = \Delta Q_i - \varepsilon_i^Q = 0 \\ \dfrac{\partial f}{\partial \varepsilon_i^P} = 2\varepsilon_i^P - \mu_i^P = 0 \\ \dfrac{\partial f}{\partial \varepsilon_i^Q} = 2\varepsilon_i^Q - \mu_i^Q = 0 \\ \dfrac{\partial f}{\partial V_j} = \sum_i (\mu_i^P \dfrac{\partial \Delta P_i}{\partial V_j} + \mu_i^Q \dfrac{\partial \Delta Q_i}{\partial V_j}) = 0 \\ \dfrac{\partial f}{\partial \theta_j} = \sum_i (\mu_i^P \dfrac{\partial \Delta P_i}{\partial \theta_j} + \mu_i^Q \dfrac{\partial \Delta Q_i}{\partial \theta_i}) = 0 \end{cases} \tag{2-24}$$

通过简化，可得

$$\begin{cases} \sum_i \Delta P_i \dfrac{\partial \Delta P_i}{\partial \theta_j} + \sum_i \Delta Q_i \dfrac{\partial \Delta Q_i}{\partial \theta_j} = 0 \\ \sum_i \Delta P_i \dfrac{\partial \Delta P_i}{\partial V_j} + \sum_i \Delta Q_i \dfrac{\partial \Delta Q_i}{\partial V_j} = 0 \end{cases} \tag{2-25}$$

通过牛顿法，可以由式(2-25)求得迭代步

$$\Delta \boldsymbol{x} = -\left(\boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{J}(\boldsymbol{x}) + \boldsymbol{G}\right)^{-1} \boldsymbol{J}(\boldsymbol{x})^T \boldsymbol{F}(\boldsymbol{x}) \tag{2-26}$$

其中，$\boldsymbol{J}(\boldsymbol{x})$为雅克比矩阵，$\boldsymbol{G}$ 为与 $\boldsymbol{J}(\boldsymbol{x})$同阶的包含潮流方程海森矩阵信息的方

阵。***G*** 阵的求取需要一定的计算量。

由此，优化方法的计算流程图如下：

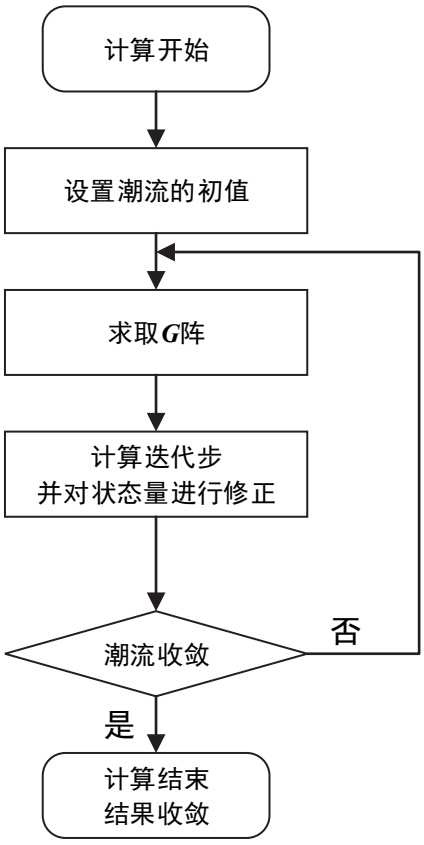


图 2-3 优化算法的计算流程图

Fig. 2-3 Flow chart of the optimal method

优化算法的特点是：

1）不考虑规划问题的凸性时，算法能够求得潮流的精确解或最小二乘解，然而修正矩阵的求取以及约束条件的增加会大幅度提升算法的计算量；

2）考虑规划问题的凸性时，则存在收敛性问题，变得更为复杂。

2.2.4 LM 方法

LM 方法最早由 Kenneth Levenberg 于 1944 年提出[35]，用于求解非线性最小二乘问题。在 1963 年，Donald W. Marquardt 重新发现了该方法[36]。由此，LM 方法成为一个求解非线性最小二乘问题的经典算法。文献[17]首先将 LM 方法引入电力系统病态潮流计算中，验证了算法对于提高潮流计算收敛性的作用。文献[18]则从理论角度分析了 LM 方法应用于电力系统潮流计算的收敛性特点，同时在 LM 方法中引入自适应因子，进一步提高了算法的收敛性。

对于潮流方程(2-1)，LM 方法考虑如下最小二乘问题：

$$\min \boldsymbol{f}(\boldsymbol{x}) = \frac{1}{2}\left\|\boldsymbol{F}(\boldsymbol{x})\right\|^2 = \frac{1}{2}\boldsymbol{F}(\boldsymbol{x})^T\boldsymbol{F}(\boldsymbol{x}) \tag{2-27}$$

若求得解 $\boldsymbol{x}^*$ 使得 $f(\boldsymbol{x}^*)=0$，则易知 $\boldsymbol{x}^*$ 为潮流方程(2-1)的精确潮流解。

对式(2-27)中的 $F(\boldsymbol{x})$ 在当前迭代步 $\boldsymbol{x}_k$ 作一阶泰勒展开，并忽略高次项，可得

$$\min \boldsymbol{f}(\boldsymbol{x}_{k+1}) = \frac{1}{2}\left\|\boldsymbol{F}(\boldsymbol{x}_k)+\boldsymbol{J}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k\right\|^2 \tag{2-28}$$

其中，$\boldsymbol{J}(\boldsymbol{x}_k)$为当前迭代步下的雅克比矩阵，$\Delta\boldsymbol{x}_k$ 为迭代步长 $\boldsymbol{x}_{k+1}-\boldsymbol{x}_k$。

根据文献[35]，式(2-28)的解可以表示为

$$\Delta\boldsymbol{x}_k = -(\boldsymbol{J}(x_k)^T\boldsymbol{J}(x_k)+\mu_k\boldsymbol{I})^{-1}\boldsymbol{J}(x_k)^T\boldsymbol{F}(x_k) \tag{2-29}$$

其中，$\mu_k\geq 0$ 为 LM 方法阻尼因子；$\boldsymbol{I}$ 为与 $\boldsymbol{J}(\boldsymbol{x}_k)$同规模的单位阵。

可以看出，与 2.2.3 节优化算法类似，LM 方法求取迭代步是在 $\boldsymbol{J}^T\boldsymbol{J}$ 矩阵的基础上加上一个修正矩阵，进而通过矩阵求逆与矩阵乘法计算 $\Delta\boldsymbol{x}_k$。然而与优化算法不同的是，LM 方法的修正矩阵为 $\mu\boldsymbol{I}$，求取简单，计算量小。

文献[37]证明了 LM 方法的迭代步长度随阻尼因子 μ 而单调递减，即阻尼因子的存在限制了原牛顿步的步长，且阻尼因子越大，限制作用越大，从而保证了算法在雅克比矩阵接近奇异时不会振荡。此外，LM 方法每次的迭代步均是下降步，即使得

$$\boldsymbol{f}(\boldsymbol{x}_k+\Delta\boldsymbol{x}) < \boldsymbol{f}(\boldsymbol{x}_k) \tag{2-30}$$

这些性质保证了 LM 方法在应对病态潮流问题中雅克比矩阵接近奇异的问题时具有良好的数值特性。

可以看出，LM 方法中核心的问题是阻尼因子 μ 的选取，当阻尼因子为 0 时，式(2-29)的迭代步求取就退化为牛顿-拉夫逊法的迭代步计算公式。而阻尼因子的选取方式，不止影响了迭代步长的大小以及收敛速度，更直接影响了算法的收敛性。

对于阻尼因子的选取，一般采用的方法是随着迭代过程的推进，阻尼因子会随之自行更新。文献[38]提出了一种基于信赖域算法[39]的更新方法，在每次计算完迭代步后，对该步长进行评估。首先定义该迭代步的实际逼近量 $Ared_k$ 和预测逼近量 $Pred_k$：

$$Ared_k = \left\|\boldsymbol{F}(\boldsymbol{x}_k)\right\|_2^2 - \left\|\boldsymbol{F}(\boldsymbol{x}_k+\Delta\boldsymbol{x}_k)\right\|_2^2 \tag{2-31}$$

$$Pred_k = \left\|\boldsymbol{F}(\boldsymbol{x}_k)\right\|_2^2 - \left\|\boldsymbol{F}(\boldsymbol{x}_k)+\boldsymbol{J}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k\right\|_2^2 \tag{2-32}$$

两者的比值定义为取舍指标 r_k，该指标的值用于决定步长的接受与否以及阻尼因子的更新方式：

$$r_k = \frac{Ared_k}{Pred_k} = \frac{\|\boldsymbol{F}(\boldsymbol{x}_k)\|_2^2 - \|\boldsymbol{F}(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k)\|_2^2}{\|\boldsymbol{F}(\boldsymbol{x}_k)\|_2^2 - \|\boldsymbol{F}(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k\|_2^2} \tag{2-33}$$

经典的 LM 方法采用以下过程更新阻尼因子：

第1步：设置初始阻尼因子μ_0，参数$\gamma_1, \gamma_2, \eta_1, \eta_2$以及收敛精度$\varepsilon$，满足

$\mu_0 > 0,\ 0 < \gamma_1 < 1 < \gamma_2,\ 0 \le \eta_1 \le \eta_2 \le 1,$

令$k = 0;$

第2步：若$\|\boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{F}(\boldsymbol{x}_k)\| \le \varepsilon$，计算收敛，退出迭代；否则进行第3步；

第3步：计算迭代步

$$\Delta\boldsymbol{x}_k = -(\boldsymbol{J}(x_k)^T\boldsymbol{J}(x_k) + \mu_k\boldsymbol{I})^{-1}\boldsymbol{J}(x_k)^T\boldsymbol{F}(x_k)$$

第4步：计算取舍指标

$$r_k = \frac{\|\boldsymbol{F}(\boldsymbol{x}_k)\|_2^2 - \|\boldsymbol{F}(\boldsymbol{x}_k + \Delta\boldsymbol{x}_k)\|_2^2}{\|\boldsymbol{F}(\boldsymbol{x}_k)\|_2^2 - \|\boldsymbol{F}(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k\|_2^2}$$

第5步：更新阻尼因子

若$r_k < \eta_1$，取$\mu_{k+1} = \gamma_2\mu_k;$

若$\eta_1 \le r_k \le \eta_2$，取$\mu_{k+1} = \mu_k;$

若$r_k > \eta_2$，取$\mu_{k+1} = \gamma_1\mu_k;$

第6步：更新迭代点

$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$，令$k = k + 1$，返回第2步

文献[40]证明，当选取阻尼因子为 $\mu_k = \|\boldsymbol{F}(\boldsymbol{x}_k)\|^2$ 时，算法有局部二次收敛特性。由此，文献[17]采用 $\mu_k = 0.001\|\boldsymbol{F}(\boldsymbol{x}_k)\|^2$ 的取值方式，由于功率偏差向量 $\boldsymbol{F}(\boldsymbol{x}_k)$ 在计算过程中会进行求解，该取值方式在计算阻尼因子时不需要过多的计算量。然而，考虑到阻尼因子 μ 正比于功率偏差向量的二范数平方，经过多次迭代后，迭代点接近精确解时，阻尼因子会迅速减小，趋近于 0。若此时阻尼因子的值小于机器的最小精度，则对于计算机程序而言，阻尼因子 μ=0，此时 LM 方法就退化为常规牛顿-拉夫逊方法，阻尼因子的存在失去意义。与此同时，当迭代刚开始时，功率偏差向量 $\boldsymbol{F}(\boldsymbol{x}_k)$的二范数较大，导致阻尼因子较大，过度限制步长，影响了收敛的速度。

为了对以上问题进行改进，2003 年我国学者范金燕提出了基于自适应阻尼因子的 LM 方法[41]，阻尼因子的形式为：

$$\mu_k = \alpha_k \left\| \boldsymbol{F}(\boldsymbol{x}_k) \right\|_2 \tag{2-34}$$

式中，阻尼因子的值正比于功率偏差向量的二范数，而系数则依据信赖域算法进行更新。自适应阻尼因子更新的过程如下：

第1步：设置初始阻尼因子系数α_0，参数$m, \eta_0, \eta_1, \eta_2$以及收敛精度$\varepsilon$，满足

$$\alpha_0 > m > 0,\ 0 < \gamma_1 < 1 < \gamma_2,\ 0 \le \eta_0 \le \eta_1 \le \eta_2 \le 1,$$

令$k = 0$;

第2步：若$\left\| \boldsymbol{J}(\boldsymbol{x}_k)\boldsymbol{F}(\boldsymbol{x}_k) \right\| \le \varepsilon$，计算收敛，退出迭代；否则进行第3步；

第3步：计算迭代步

$$\Delta \boldsymbol{x}_k = -(\boldsymbol{J}(x_k)^T \boldsymbol{J}(x_k) + \mu_k \boldsymbol{I})^{-1} \boldsymbol{J}(x_k)^T \boldsymbol{F}(x_k)$$

第4步：计算取舍指标

$$r_k = \frac{\left\| \boldsymbol{F}(\boldsymbol{x}_k) \right\|_2^2 - \left\| \boldsymbol{F}(\boldsymbol{x}_k + \Delta \boldsymbol{x}_k) \right\|_2^2}{\left\| \boldsymbol{F}(\boldsymbol{x}_k) \right\|_2^2 - \left\| \boldsymbol{F}(\boldsymbol{x}_k) + \boldsymbol{J}(\boldsymbol{x}_k)\Delta \boldsymbol{x}_k \right\|_2^2}$$

第5步：更新阻尼系数

若$r_k < \eta_1$，取$\alpha_{k+1} = 10\alpha_k$;

若$\eta_1 \le r_k \le \eta_2$，取$\alpha_{k+1} = \alpha_k$;

若$r_k > \eta_2$，取$\alpha_{k+1} = \max\{\frac{\alpha_k}{10}, m\}$;

第6步：更新阻尼因子

$$\mu_k = \alpha_k \left\| \boldsymbol{F}(\boldsymbol{x}_k) \right\|_2$$

第7步：更新迭代点

若$r_k > \eta_0, \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta \boldsymbol{x}_k$，接受迭代步

若$r_k \le \eta_0, \boldsymbol{x}_{k+1} = \boldsymbol{x}_k$，不接受迭代步

令$k = k + 1$，返回第2步

根据文献[18]，计算流程中的参数$m, \eta_0, \eta_1, \eta_2$可根据大量实验后进行选定，本文选取$m = 10^{-4}, \eta_0 = 10^{-4}, \eta_1 = 0.85, \eta_2 = 0.95$。

基于自适应阻尼因子的 LM 方法具有全局收敛至解的特性[42]，对于电力系统病态潮流计算而言，若系统有潮流解，则算法收敛至精确潮流解，否则，会收敛至最小二乘解，为运行人员提供一定的参考。

根据自适应阻尼因子的更新过程，基于自适应阻尼因子 LM 方法的迭代过程如下：

1）对于良态系统的求解，每步迭代的取舍指标 r_k 较大，使得阻尼因子迅速减小。数次迭代后，阻尼因子接近于 0，故迭代步近似于牛顿步，保证了迭代的快速收敛；

2）对于病态系统的求解，接近雅克比矩阵奇异的区域时，取舍指标 r_k 较小，使阻尼因子增大，限制迭代步长，防止振荡发生。同时，当取舍指标 r_k 过小时，该步迭代会被放弃，迭代步方向调整，直至取舍指标 r_k 到达合理范围为止，从而保证了算法的收敛性。

基于自适应阻尼因子的 LM 方法计算流程图如下：

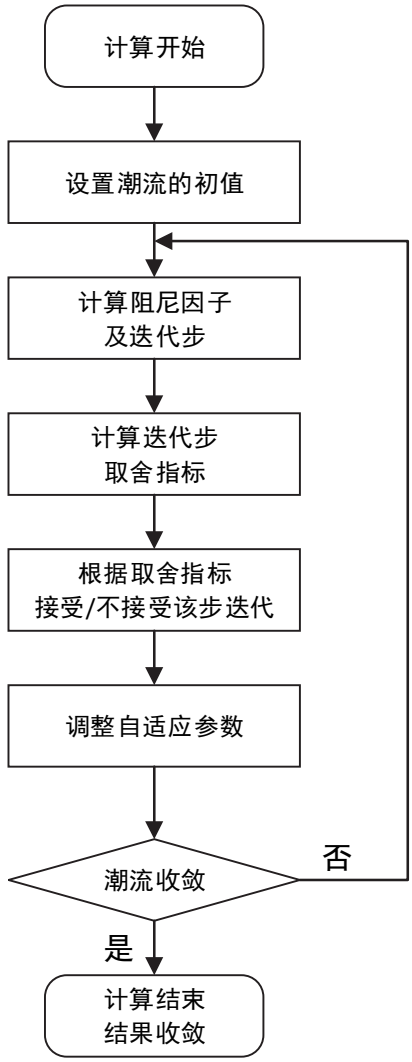


图 2-4 自适应 LM 方法的计算流程图

Fig. 2-4 flow chart of the self-adaptive LM method

采用自适应 LM 方法计算电力系统病态潮流，具有以下特点：

1）求取迭代步是在 $\boldsymbol{J}^T\boldsymbol{J}$ 矩阵的基础上加上一个修正矩阵，相比于常规牛顿

法而言，计算量增加不大；

2）阻尼因子的存在限制了迭代步长，防止常规牛顿发的振荡问题，同时阻尼因子的自适应调节保证了算法的全局收敛性。

2.2.5 算法比较

本章 2.2.1 至 2.2.4 节分别对四个主流的病态潮流算法进行了介绍，分别是最优乘子法、张量法、优化算法以及自适应 LM 方法。下面对这个四个算法进行比较：

（1）计算量：

最优乘子法在求解牛顿迭代步之后，还需要通过 Cardan 公式计算式(2-6)，从而求出最优乘子，并对迭代步长进行调整。相比于牛顿法，最优乘子法的计算量稍有增加。

张量法中，插值张量法首先需要通过(2-17)求得张量系数，再代入(2-18)中计算出迭代步，需要求解两次矩阵求逆问题，因此计算量大约为牛顿法的两倍；直接张量法与插值张量法类似，也需要先求解张量方程，然后求出迭代步，计算量也为牛顿法的两倍左右。

优化算法需要求解式(2-26)计算迭代步，单从计算过程而言，相比于牛顿法多了矩阵转置和几次矩阵乘法、加法。与此同时，矩阵 $\boldsymbol{G}$ 的求取也较为复杂，需要一定的计算量。此外，若优化问题的约束规模增加，计算的复杂性会进一步增大。总体而言，优化算法相对于牛顿法增加较多。

LM 算法需要通过式(2-29)计算迭代步，与优化算法类似，求解过程相比于牛顿法多出了矩阵转置、乘法、加法等计算量。同时，需要对阻尼因子进行更新。然而阻尼因子的更新主要是一系列逻辑判断，并不需要进行大规模的计算，故总体而言，这部分的计算量相比于迭代步的求解而言非常微小。因此，LM 方法的整体计算量相比于牛顿法而言增加也不多。

（2）收敛性：

最优乘子法通过对牛顿法步长进行修正，避免病态潮流时产生的振荡问题。然而，最优乘子法仅对步长的幅值进行修正，而不改变步长方向，因此一定程度上还是会存在方法对初值的敏感性。此外，当算法产生停滞的时候，所得到的结果也并非可以用于参考的最小二乘解。

张量法通过对迭代步求取方式的改进，对迭代步的方向进行了修正，因此收敛性好于只对步长幅值进行修正的最优乘子法。然而，当雅克比矩阵非常接近奇

异的情况下，该方法依旧会因为求不到牛顿方向，而导致其对方向的修正随之失效[14]。

优化算法在不考虑模型凸性的情况下，能够收敛至潮流的精确解或是最小二乘解，具有较好的收敛性。

LM 方法与优化算法类似，具有收敛至系统精确解或最小二乘解的特性，此外，当阻尼因子采用自适应因子时，方法具有全局收敛性。

下面通过表格对几种方法的特性进行总结：

表 2-1 四种病态潮流算法特性总结

Table 2-1 Summary of four ill-conditioned algorithms' features

算法名称	计算量	收敛特性
最优乘子法	相比 NR 法增加不大	一般可收敛至近似最小二乘解
张量法	大约 NR 法的两倍	雅克比矩阵非常接近奇异时仍会失效
优化算法	相比 NR 法增加较大	可收敛至最小二乘解
LM 方法	相比 NR 法增加不大	阻尼因子选取恰当时可全局收敛，可收敛至精确

根据表 2-1，可以看出四种方法中，最优乘子法与 LM 法的计算量增加较小，而张量法与优化算法增加较大；而从收敛性角度，优化算法与 LM 方法的收敛性相比最优乘子法与张量法更优。因此，综合计算量与收敛特性两个方面，可以看出 LM 方法都具有较为突出的优点。与此同时，LM 方法的计算流程相对于牛顿-拉夫逊法改动不大，在传统的潮流计算程序上，仅需要进行一定的修改，即可实现基于 LM 方法的潮流计算程序。

综合以上特性，本文在以上四种算法中选取 LM 方法作为病态潮流算法，同时采用自适应阻尼因子，进一步提高算法的收敛特性。

2.3 LM 方法中病态潮流迭代步求解方案

根据 2.2 节对主流病态潮流的研究，选取 LM 方法作为本文病态潮流高性能实现的基础。在 LM 方法的计算流程中，式(2-29)的稀疏线性方程组求解是核心的步骤，不同于牛顿法，解算的过程中需要对矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 进行求逆，下面对该步分解的实现方案进行研究。

2.3.1 直接求解方案

对于式(2-29)的求解，最为直观的求解方案如下：

1）根据当前迭代点的信息，计算系统的雅克比矩阵 $\boldsymbol{J}$；

2）通过矩阵转置、矩阵相乘以及矩阵相加，求得矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$；

3）考虑到矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 的对称正定特性，采用稀疏 Cholesky 分解对其进行因子分解，随后利用稀疏前代/回代，求得迭代步长。

该求解方案直接基于式(2-29)的计算流程，本文称之为直接求解方案。

2.3.2 扩展矩阵求解方案

有学者认为，对于形如 $\boldsymbol{A}\boldsymbol{A}^T$ 或 $\boldsymbol{A}^T\boldsymbol{A}$ 的矩阵，若采用直接求解的方案进行因子分解，会在矩阵乘法这一步骤上花费大量的时间，因此影响计算效率[43]。文献[44]在研究内点法求解电力系统规划问题时，遇到了求解类似形式非线性方程组的问题，文中将该形式的矩阵分解问题转换形式，变为对一个 2×2 阶块状扩展矩阵进行因子分解的问题，从而避免对于 $\boldsymbol{A}\boldsymbol{A}^T$ 的计算。

类似地，文献[18]在研究自适应 LM 方法应用于病态潮流计算时，认为系数矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 稀疏度相比于雅克比矩阵 $\boldsymbol{J}$ 降低了很多，影响到计算效率。此外，其稀疏结构与导纳矩阵相异，因此提出采用扩展矩阵的方式对迭代步进行求解。

首先，将式(2-29)进行改写

$$-(\boldsymbol{J}(x_k)^T\boldsymbol{J}(x_k)+\mu_k\boldsymbol{I})\Delta\boldsymbol{x}_k=\boldsymbol{J}(x_k)^T\boldsymbol{F}(x_k) \tag{2-35}$$

引入一个中间变量 $\boldsymbol{y}$，令

$$\boldsymbol{y}=\boldsymbol{J}(x_k)\Delta\boldsymbol{x}_k/\beta_k \tag{2-36}$$

其中，系数 $\beta_k=\sqrt{\mu_k}$ 。

将式(2-36)代入式(2-35)中，可得

$$\beta_k\Delta\boldsymbol{x}_k+\boldsymbol{J}(x_k)^T\boldsymbol{y}=-\boldsymbol{J}(x_k)^T\boldsymbol{F}(x_k)/\beta_k \tag{2-37}$$

可将式(2-36)与式(2-37)联立，

$$\begin{cases}\boldsymbol{J}(x_k)\Delta\boldsymbol{x}_k-\beta_k\boldsymbol{y}=0\\ \beta_k\Delta\boldsymbol{x}_k+\boldsymbol{J}(x_k)^T\boldsymbol{y}=-\boldsymbol{J}(x_k)^T\boldsymbol{F}(x_k)/\beta_k\end{cases} \tag{2-38}$$

式(2-38)可改写成矩阵的形式

$$\begin{bmatrix}\beta_k\boldsymbol{I} & \boldsymbol{J}(\boldsymbol{x}_k)^{\mathrm{T}}\\ \boldsymbol{J}(\boldsymbol{x}_k) & -\beta_k\boldsymbol{I}\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{x}_k\\ \boldsymbol{y}\end{bmatrix}=\begin{bmatrix}\boldsymbol{F}_1(\boldsymbol{x}_k)\\ 0\end{bmatrix} \tag{2-39}$$

其中，$\boldsymbol{F}_1(\boldsymbol{x}_k)=-\boldsymbol{J}^{\mathrm{T}}(\boldsymbol{x}_k)\boldsymbol{J}(\boldsymbol{x}_k)/\beta_k$。

于是，原问题就从对矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 的分解问题转换为了对形如 $\begin{bmatrix} \beta_k \boldsymbol{I} & \boldsymbol{J}(\boldsymbol{x}_k)^{\mathrm{T}} \\ \boldsymbol{J}(\boldsymbol{x}_k) & -\beta_k \boldsymbol{I} \end{bmatrix}$ 的 2×2 阶块状扩展矩阵分解问题，通过求解式(2-39)的线性方程组，可以求出迭代步 $\Delta \boldsymbol{x}_k$。

根据文献[18]，采用式(2-39)的形式对迭代步进行求解，相较于直接求解方案更具优势。其原因在于：

1） 线性方程组的系数矩阵 $\begin{bmatrix} \beta_k \boldsymbol{I} & \boldsymbol{J}(\boldsymbol{x}_k)^{\mathrm{T}} \\ \boldsymbol{J}(\boldsymbol{x}_k) & -\beta_k \boldsymbol{I} \end{bmatrix}$ 与系统的导纳矩阵同构，因此，常规潮流程序的节点重排序、稀疏存储结构等内容可以完全兼容于该方法中，降低了程序开发的复杂度；

2）系数矩阵的稀疏程度相较于雅克比矩阵而言下降不大，此外，矩阵具有对称特性，可以采用稀疏 LDL 分解，计算量增加也不明显。

2.3.3 方案验证与比较

为了实际验证直接分解方案与扩展矩阵方案间的优劣，本文从因子分解时间和注入元数量这两个指标出发，对以上两个方案的效果进行测试。本文采用的测试环境见 5.1 节叙述，测试程序采用 Matlab R2014a 进行执行与计时。算例所采用的系统潮流数据的基本信息如表 2-2 所示，其中，实际的电网数据来源自华东电网历年数据。

表 2-2 算例测试系统基本概况
Table 2-2 Basic information of test systems

系统名称	节点数	支路数	数据来源
IEEE162 节点	162	280	IEEE162
IEEE300 节点	300	409	IEEE300
华东 3647 系统	3647	4239	华东电网 2004 年
华东 4171 系统	4171	4860	华东电网 2005 年
华东 4769 系统	4769	5597	华东电网 2006 年
华东 5473 系统	5473	6642	华东电网 2009 年
华东 7872 系统	7872	9830	华东电网 2014 年夏低
华东 8241 系统	8241	10280	华东电网 2015 年夏低

对于表 2-2 中的系统数据，采用 LM 方法的计算流程，执行第一步迭代时的迭代步求解步骤。对于两个方案具体的稀疏矩阵，首先采用近似最小度排序

(approximate minimum degree, AMD)进行节点重排序，减少注入元数量，随后，分别采用稀疏 LDL 分解算法进行因子分解。统计两种方案中式(2-29)与(2-39)对应稀疏矩阵通过因子分解前后非零元的个数以及矩阵分解时间。此算例中，为了测试程序的易开发性，阻尼因子取值为 $\mu_k = 0.001\left\|\boldsymbol{F}(\boldsymbol{x}_k)\right\|^2$。矩阵分解时间与非零元个数的结果分别如表 2-3 及表 2-4 所示

表 2-3 两种方案的矩阵分解时间统计

Table 2-3 Run time of matrix factorization of two methods

测试系统	直接求解时间(s)			扩展矩阵求解时间(s)		
	矩阵形成	因子分解	总时间	矩阵形成	因子分解	总时间
IEEE162	0.00094	0.00093	0.00187	0.00071	0.00358	0.00429
IEEE300	0.00106	0.00076	0.00182	0.00080	0.00316	0.00396
华东 3647	0.00753	0.00545	0.01298	0.00430	0.02366	0.02796
华东 4171	0.00976	0.00601	0.01577	0.00500	0.03288	0.03788
华东 4769	0.01029	0.00662	0.01691	0.00539	0.02631	0.03170
华东 5473	0.01379	0.00827	0.02206	0.00746	0.02931	0.03677
华东 7872	0.01915	0.01069	0.02984	0.01076	0.04885	0.05961
华东 8241	0.02055	0.01149	0.03203	0.01167	0.03782	0.04949

表 2-4 两种方案的矩阵分解前后非零元个数

Table 2-4 Number of nonzero entries before/after factorization of two methods

测试系统	直接求解		扩展矩阵求解	
	分解前/后非零元个数	注入元个数	分解前/后非零元个数	注入元个数
IEEE162 节点	7675/18483	10808	6156/39404	33248
IEEE300 节点	9736/14948	5212	8532/36092	27560
华东 3647 系统	103828/156778	52950	94148/503556	409408
华东 4171 系统	131075/184457	53382	120956/638942	517986
华东 4769 系统	137184/203304	66120	124558/642524	517966
华东 5473 系统	182011/245801	63790	157552/724532	566980
华东 7872 系统	271824/325616	53792	240504/998696	758192
华东 8241 系统	285680/346710	61030	251772/927416	675644

表 2-3 中，矩阵形成的时间分别代表了直接分解方案中 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵的形成时间以及扩展矩阵方案中扩展矩阵的形成时间，因子分解时间则代表两者的 LDL

分解时间。表 2-4 则分别统计了两个方案分解前后矩阵的非零元个数，以及分解前后的非零元个数之差，即注入元个数。

由表 2-3，在矩阵形成部分，直接分解方案相比于扩展矩阵方案更为耗时，前者大约为后者的 1.5 倍~2 倍左右，即通过矩阵乘法形成 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 的过程比形成扩展矩阵需要更多的计算量，这一点与文献[18]的判断相符合。而另一方面，在因子分解的耗时部分，扩展矩阵方案的耗时却大大高于直接分解方案，即对于扩展矩阵的稀疏 LDL 分解耗时大约为对矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 的四倍左右。此外可以看出，在整个矩阵分解过程中，矩阵形成过程的耗时与因子分解时间基本为同一数量级，这就导致了总体分解时间上，直接分解方案要优于扩展矩阵方案。

两者差异的原因可参见表 2-4，可以看出，对同一系统而言，分解前，两方案的 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵与扩展矩阵非零元个数相近，而直接分解方案的 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵大约多出 10%-20%左右。因子分解后，扩展矩阵对应的矩阵非零元个数大幅增加，而直接分解方案矩阵的非零元个数大约只有扩展矩阵方案的三分之一左右。两相比较，扩展矩阵方案的注入元个数大幅高于直接求解方案，而且其个数随着矩阵规模的增加有进一步上升的趋势。正是大量的注入元导致了扩展矩阵方案在因子分解步骤中耗费了大量的计算时间。

进一步，若以华东 8241 系统为例，观察两个方案在因子分解前后具体的稀疏矩阵结构，如图 2-5 所示。

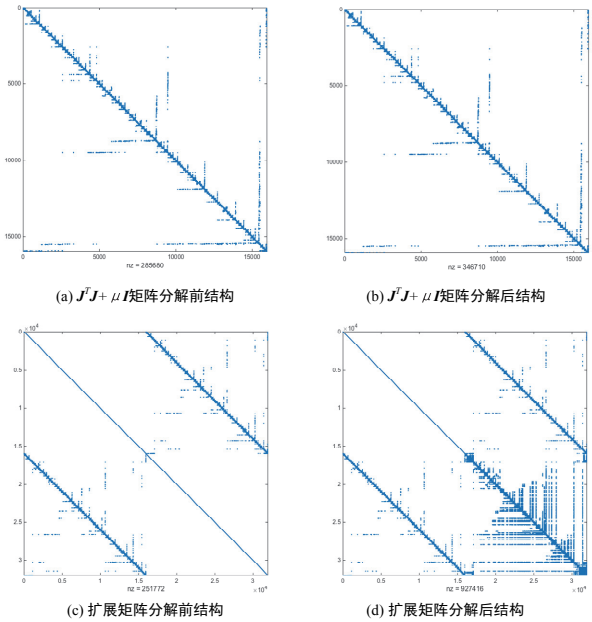


图 2-5 两方案分解前后矩阵非零结构

Fig. 2-5 Nonzero pattern of matrices before/after factorization of two methods

图 2-5 中，空白部分为零元，蓝色部分代表非零元。由图中可以看出，$\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵在分解过程中产生的注入元明显少于扩展矩阵，而扩展矩阵分解后所产生的注入元主要集中于矩阵的右下分块部分，从而导致分解后矩阵非零元个数大幅增多，进一步影响因子分解的计算速度。对于其他规模的系统，同样可以得到类似的结论。

由此，可以看出对于电力系统潮流数据，在采用 LM 方法求解的迭代步求解中，直接分解与扩展矩阵两个方案里，直接分解方案实际上具有更高的计算效率。而许多学者所采用的扩展矩阵方案，虽然其矩阵形式与导纳阵同构，使得程序开发更为简便，但由于矩阵分解过程中产生的注入元过多，导致其整体分解时间近乎于直接分解方案的两倍。

考虑到本文的目标是尽可能高效、快速地实现病态潮流计算，从计算速度的角度出发，本文将选取更为高速有效的直接分解方案。

2.4 本章小结

本章对最优乘子法、张量法、优化算法和 LM 方法这四种主流的病态潮流算法从数学原理、收敛性、计算量等角度进行了介绍，并对四者进行了比较，发现 LM 方法无论从收敛性还是计算量两方面都具有较好的特性，故选取其作为本文病态潮流高性能实现的算法基础。此外，对 LM 方法的阻尼因子选取进行了进一步研究，发现采用自适应因子可以使算法具有全局收敛的特性。

另一方面，本章对 LM 方法中核心的迭代步求解进行了研究，分析了两种不同的计算方案，通过实际的算例验证了普遍认为更优的扩展矩阵方案在分解过程中会产生大量注入元，导致计算效率较低。从计算高速性的角度考虑，选取了较为高效的直接分解方案作为迭代步的求解方案。

第三章 稀疏 Cholesky 分解算法研究

3.1 引言

由于电力系统中节点的连接特性，在电力系统相关仿真中，涉及到矩阵运算的相应矩阵往往具有高度稀疏的特性。以导纳阵为例，根据经验，矩阵的稀疏度大约为(3~4)/n，其中 n 为系统规模[45]。当系统规模大于 500 时，矩阵的稀疏度就已小于 1%，而现代电力系统的规模远大于此，稀疏度往往不足 1‰。随着计算系统规模的增大，稀疏度还会进一步下降。

考虑到电力系统极度稀疏的特性，通常会采用稀疏技术，减少内存占用并提高计算速度。本章对 LM 方法计算病态潮流中将会采用的稀疏技术进行介绍，并针对 LM 方法中迭代步求解所涉及的稀疏矩阵分解算法进行研究，为后续的高性能实现提供理论基础。

3.2 稀疏技术概述

3.2.1 稀疏直接法与迭代法

对于一个矩阵，若非零元仅占很小的百分比，则称其为稀疏矩阵。在实际的数值计算和工程应用中，所涉及的矩阵往往是稀疏的，因此，稀疏技术在诸如规划问题、结构分析、电力系统、微分方程数值解法、遗传学理论等等众多领域有大量的研究与应用[46]。

稀疏技术中一个最基础也是最常面临的问题就是以矩阵的形式求解线性方程组，即

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{3-1}$$

其中，矩阵 $\boldsymbol{A}$ 为 $n \times n$ 阶的稀疏方阵，$\boldsymbol{x}$、$\boldsymbol{b}$ 分别为 n 维向量，且 $\boldsymbol{x}$ 为所需求得的解向量。

早在 1823 年，德国数学家 Johann C. F. Gauss 在其给学生 Gerling 的信中首先提出了采用迭代法求解如式(3-1)的问题，即高斯-塞德尔法[47]。1845 年，德国数学家 Carl G. J. Jacobi 提出了另一种迭代求解方法，即雅克比迭代法[48]。这些研究主要是从数学原理角度阐述求解大规模线性方程组的方法，同时，也一定程度涉及了一些利用矩阵稀疏特性的方法。随后一段时间中，对于稀疏技术的研究

也主要集中于迭代法上，美国学者 Richard S. Varga 于 1962 年对当时迭代法的主要成果进行了总结，并深入分析探讨了迭代法求解系数矩阵高度稀疏的线性方程组问题[49]。

与此同时，学者们也开始意识到可以利用矩阵的稀疏特性在直接法求解中获得更好的计算特性。1957 年，Harry M. Markowitz 就基于稀疏矩阵的特点，将高斯消元法应用于线性规划中[50]。而在电力系统领域，1963 年，N. Sato 与 William F. Tinney 首先探讨了稀疏技术在电力系统导纳矩阵中的应用[51]。在 1970 年，E. C. Ogbuobiri 与 Tinney 等人又进一步对电力系统计算中高斯消元法的稀疏求解进行了研究[52]。而从 20 世纪 70 年代至 90 年代，以 I. S. Duff[53]、Joseph W. H. Liu[54]、John R. Jilbert[55]等为代表的数值计算领域研究者们对稀疏技术在直接法求解线性系统中的应用做出了巨大的贡献。而在本世纪，美国学者 Timothy A. Davis 又先后提出 AMD 排序算法[57]、KLU 稀疏分解算法[58]等算法，使得稀疏技术进一步得到完善。同时，他使用 C 语言开发了 SuiteSparse 程序包，其中包含了 CSparse[59]、UMFPACK[60]、CHOLMOD[61]等稀疏解算器，具有极高的计算效率，源代码应用于 MATLAB 等商业软件中。

如上所述，对于式(3-1)的求解，通常有迭代法和直接法两类方法。其中，迭代法实现简单，特别是易于并行化处理，然而算法的收敛性较差，并且收敛速度很慢。此外，广义最小残差法（Generalized Minimal Residual, GMRES）等迭代算法需要非常复杂的预处理算法，其执行时间甚至可能超过迭代法自身的计算时间，而且执行效果也因问题而异。

相比之下，直接法则因其计算结果的鲁棒性、计算过程的可预测性以及计算速度的高效性称为工程应用中的主流选择。基于直接法的稀疏系统求解算法往往涉及图论、组合数学等知识，程序实现难度较高，但所开发出的稀疏解算器往往具有很好的通用性，能够实现高效、正确的矩阵运算。

基于以上原因，本章将主要对直接法相关的稀疏技术进行研究。

3.2.2 稀疏存储技术

对于稀疏矩阵，首要的问题便是以何种形式存储这个矩阵。以一个一万阶的矩阵为例，若采用满阵的形式，以双精度开辟空间存储该矩阵，大约需要占用 800MB 的内存空间，而由于矩阵的稀疏特性，其中大部分空间储存的是零元素，内存的有效利用率可能不足 1%。与此同时，以满阵形式存储稀疏矩阵，还会进一步使得矩阵计算的过程中，大量的运算是针对零元素所进行的无用计算，

极大地影响运算效率。

一般而言，常见的稀疏存储结构有：三元组存储、列压缩存储（Compressed Sparse Column，CSC）、行压缩存储(Compressed Sparse Row，CSR)等，下面逐个进行介绍：

（1）三元组存储

对于稀疏矩阵中的每一个非零元，有描述其位置的行号、列号坐标，以及自身的元素值。一个最为直观的稀疏存储方式，就是使用三个数组，分别存储稀疏矩阵中每一个非零元的行号、列号以及非零元值。

以下面一个稀疏矩阵为例：

$$\begin{bmatrix} 7 & 11 & 20 & \\ & 13 & & 8 \\ & 6 & 15 & 19 \\ & 9 & & 10 \\ 18 & & 3 & 6 \\ & 24 & 27 & 17 \end{bmatrix} \tag{3-2}$$

若采用三元组形式存储矩阵(3-2)，其形式如表 3-1。

表 3-1 稀疏矩阵的三元组形式存储

Table 3-1 The triplet form of sparse matrix storage

序号	0	1	2	3	4	5	6	7
ROW	0	0	0	1	1	2	2	2
COL	0	2	4	1	4	1	3	5
VAL	7	11	20	13	8	6	15	19
序号	8	9	10	11	12	13	14	15
ROW	3	3	4	4	4	5	5	5
COL	1	5	0	3	4	1	3	5
VAL	9	10	18	3	6	24	27	17

如表 3-1，其中数组 ROW 存储行号，COL 存储列号，VAL 存储具体数值。值得一提的是，三元组形式本身对有序性没有要求，即表 3-1 中的元素序号可以进行随意的调换。另外，若矩阵的对角元都有值，还可以专门开辟一个 DIAG 数组存储对角元，而只用三元数组存储非对角的非零元。

采用三元组形式存储的最大优势是其直观性，此外，由于其元素的存储本身可以是无序的，添加或者删去一个元素都较为简便。然而，采用该形式存储矩阵

会导致元素检索非常缓慢。即便存储的元素已经按照行号、列号进行了排序，获取数据依旧需要进行大量的搜索。

假设现在需要获得矩阵(3-2)中第 4 行第 3 列元素，则首先需要从 ROW 行进行遍历，寻找第一个 ROW 值为 4 的元素。根据表 3-1，需要从元素 0 检索至元素 10。随后，需要从元素 10 开始，寻找 COL 值为 3 的元素，即从元素 10 检索至元素 11，最终获得该第 4 行第 3 列元素的值 3。可以看出，这样一次对小规模矩阵简单的检索就需要进行大量的逻辑判断。

此外，若是存储矩阵的元素本身是无序的，对元素值的获取更是需要对整个数组进行遍历后才可以完成。这样的效率对于稀疏矩阵计算的高效性要求而言是无法容忍的。

（2）列压缩存储

为了解决三元组形式存储所存在的检索效率低下问题，列压缩存储 CSC 格式通过改变 COL 数组所存储的值，提高在列维度上的检索效率。CSC 格式需要元素的存储按照列的顺序进行，每一列中，元素按照行号的从大到小顺序排序。而 CSC 格式中的 COL 数组不再存储元素的列号，而是该列第一个非零元的元素序号。依旧以矩阵(3-2)为例，矩阵的 CSC 存储形式如表 3-2。

表 3-2 稀疏矩阵的列压缩存储

Table 3-2 The CSC form of sparse matrix storage

序号	0	1	2	3	4	5	6	7
ROW	0	4	1	2	3	5	0	2
VAL	7	18	13	6	9	24	11	15
COL	0	2	6	7	10	13	16	
序号	8	9	10	11	12	13	14	15
ROW	4	5	0	1	4	2	3	5
VAL	3	27	20	8	6	19	10	17

由表 3-2，COL 数组存储了第 0 至第 5 列首个非零元的序号，而 COL[6]则是最大序号+1，标示了总的非零元个数，起到了封底的作用。而数组 ROW 与 VAL 存储的内容与三元组形式类似，但必须保证同一列的元素依照 ROW 值从小到大的顺序排列，便于检索。

依旧以需要获得矩阵(3-2)中第 4 行第 3 列元素为例，观察 CSC 格式存储的定位效率。首先，由于知道列号为 3，可根据 COL[3]的值，即序号 7，直接知道第 3 列的第一个非零元为 7 号元素。而 COL[4]的值为 10，即第 3 列最后一个元

素序号为 9，第 10 号元素开始即为第 4 列的元素。随后，从 7 号开始至 9 号结束，搜索 ROW 值为 4 的元素，发现 ROW[8]为 4，获得其元素值，即 VAL[8]。

可以看出，相比于三元组形式而言，CSC 格式在列上的检索速度大为提升。且所占用的存储空间进一步减少。

（3）行压缩存储

与列压缩存储格式相对应，也可采用行压缩 CSR 格式进行存储。其方法就是将元素的存储按照行的顺序进行，每一行中，元素按照列号的从大到小顺序排序。而 CSR 格式中的 ROW 数组不再存储元素的行号，而是该行第一个非零元的元素序号。以矩阵(3-2)为例，矩阵的 CSR 存储形式如表 3-3。

表 3-3 稀疏矩阵的行压缩存储

Table 3-3 The CSR form of sparse matrix storage

序号	0	1	2	3	4	5	6	7
COL	0	2	4	1	4	1	3	5
VAL	7	11	20	13	8	6	15	19
ROW	0	3	5	8	10	13	16	
序号	8	9	10	11	12	13	14	15
COL	1	5	0	3	4	1	3	5
VAL	9	10	18	3	6	24	27	17

由表 3-3，ROW 数组存储了第 0 至第 5 行首个非零元的序号，与此同此同时，COL 与 VAL 的存储满足有序性的要求。而对于 CSR 格式的定位与 CSC 格式非常相似，区别仅为前者首先根据要定位的行号从 ROW 数组获取该行的第一个非零元序号。故该格式的定位过程此处就不再赘述。

可以看出，不论是 CSC 格式或是 CSR 格式，相较于三元组格式都能在保证存储稀疏性的前提下，大幅度提升元素定位的效率。虽然两者依旧不能完全去除定位过程中的逻辑判断，实际的效率已经可以令人满意。

对于实际的稀疏矩阵计算，采用行压缩或者列压缩的形式一般皆可，而在成熟的稀疏解算器中，采用 CSC 与 CSR 格式的程序库都可以见到。需要注意的是，在算法实现的过程中，算法流程需要严格参照存储格式，即若采用 CSC 格式，为了保证效率，算法中元素的提取就应按列进行。本文通过对主流稀疏解算器的分析，选取当前应用更为广泛的 CSC 格式，作为稀疏矩阵存储的格式。

3.2.3 稀疏三角形方程组求解

求解稀疏三角方程组即为求解形如

$$\boldsymbol{Lx}=\boldsymbol{b} \tag{3-3}$$

式中：$\boldsymbol{L}$ 为 n 阶下三角方阵。

在稀疏矩阵求解中，式(3-3)的求解是一个非常重要的核心过程。不论是 Cholesky 分解或是 LU 分解都需要频繁进行这样的稀疏三角方程组求解，而稀疏线性方程组求解的回代过程，也恰恰正是一个形如式(3-3)的稀疏三角方程组求解过程。

将矩阵 $\boldsymbol{L}$ 分解为 2×2 的矩阵块形式，即：

$$\begin{bmatrix} l_{11} & 0 \\ \boldsymbol{l}_{21} & \boldsymbol{L}_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ \boldsymbol{x}_2 \end{bmatrix}=\begin{bmatrix} b_1 \\ \boldsymbol{b}_2 \end{bmatrix} \tag{3-4}$$

其中：$\boldsymbol{L}_{22}$ 是矩阵 $\boldsymbol{L}$ 的 $n-1$ 阶子阵，$\boldsymbol{l}_{21}$、$\boldsymbol{x}_2$ 和 $\boldsymbol{b}_2$ 是长度为 $n-1$ 的向量，l_{11}、x_1 和 b_1 为标量。

由矩阵乘法可得：

$$\begin{cases} l_{11}x_1=b_1 \\ \boldsymbol{l}_{21}x_1+\boldsymbol{L}_{22}\boldsymbol{x}_2=\boldsymbol{b}_2 \end{cases} \tag{3-5}$$

由式(3-5)可得：

$$\begin{cases} x_1=b_1/l_{11} \\ \boldsymbol{L}_{22}\boldsymbol{x}_2=\boldsymbol{b}_2-\boldsymbol{l}_{21}x_1 \end{cases} \tag{3-6}$$

即 x_1 可求得，而 $\boldsymbol{x}_2$ 可通过解 $n-1$ 阶线性方程组求得。通过迭代可将解 $\boldsymbol{x}$ 求出。

若右端项稠密，求解稀疏三角方程组算法的伪代码如下：

```
x = b
for j = 0 to n − 1 do
    x_j = x_j / l_jj
    for each i > j for which l_jj ≠ 0 do
        x_i = x_i − l_ij x_j
```

以上算法针对右端项稠密的情况。若右端项稀疏，右端为 0 对应列可能出现冗余操作，浪费计算时间。为了避免此情况的发生，可对算法进行修改。为了便于讨论，下面假设矩阵 $\boldsymbol{L}$ 具有单位对角元，则算法的伪代码如下：

```
x = b
for j = 0 to n−1 do
    if x_j ≠ 0
    for each i > j for which l_jj ≠ 0 do
        x_i = x_i − l_ij x_j
```

以上算法可避免对 $x_j = 0$ 对应列的冗余操作，其时间复杂度为 $O(n + |b| + f)$，其中 n 为矩阵阶数，$|b|$为右端项的非零元个数，而 f 为浮点数操作次数。通常，$|b| < f$，而对大规模高稀疏度的矩阵 $f < n$，即时间复杂度的数量级相当于 $O(n)$。

若已知所需处理的列号，定为集合 $\chi = \{j \mid x_j \neq 0\}$，则算法的伪代码可做如下改写：

```
x = b
for each j ∈ χ do
    for each i > j for which l_jj ≠ 0 do
        x_i = x_i − l_ij x_j
```

若集合 χ 已知，则算法的时间复杂度可降为 $O(|b| + f)$。

集合 χ 的求取即判断 x_i 是否为零，可由以下两条准则进行判断：

$$\begin{aligned} &1.\ b_i \neq 0 \Rightarrow x_i \neq 0 \\ &2.\ x_j \neq 0 \wedge \exists i(l_{ij} \neq 0) \Rightarrow x_i \neq 0 \end{aligned} \tag{3-7}$$

式(3-7)的两条准则可以通过图 3-1 表示：

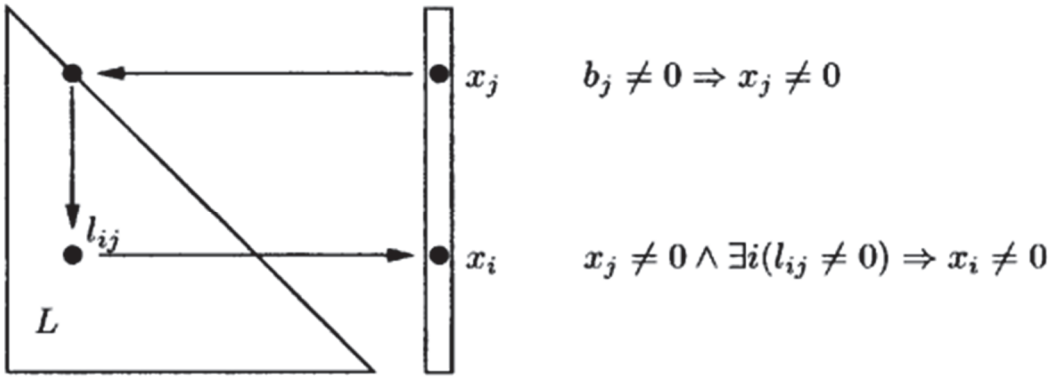


图 3-1 集合 χ 的求取

Fig. 3-1 The criteria of getting set χ

若将矩阵 $\boldsymbol{L}$ 看做有向图 $G_L = (V, E)$，点集 $V = \{1 \ldots n\}$，边集 $E = \{(j, i) \mid l_{ij} \neq 0\}$，则依照(3-7)两条准则求取集合 χ 的过程可看做是对图 G_L 的遍

历问题，用函数 reach 表示该过程。该函数的伪代码如下：

```
function χ = reach(L, B)
    假定所有顶点都未被标记
        for each i for which b_i ≠ 0 do
            if 顶点i未被标记
                dfs(i)
```

其中，*dfs*(*i*)表示对顶点 *i* 做深度优先搜索，是一个递归函数，其伪代码为：

```
function dfs(j)
    标记节点j
        for each i for which l_ij ≠ 0 do
            if 顶点i未被标记
                dfs(i)
        将j压入χ栈内
```

整个 reach 过程的时间复杂度是 O(|b| + f)。故调用 reach 后，再根据集合 χ 进行稀疏三角形方程组的求解，可将时间复杂度降低至 O(|b| + f)。

下面以一个 8 阶稀疏三角方程组为例，简单叙述 reach 的具体计算过程。示例的三角方程组如图 3-2。

$$L \quad x = \quad b$$

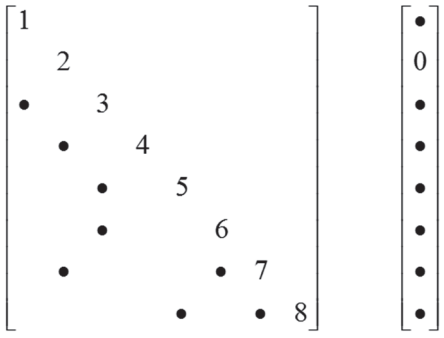


图 3-2 稀疏三角方程组示例

Fig. 3-2 The example of sparse triangular system

对于如图 3-2 稀疏三角矩阵的 reach 过程，首先从右端项判断开始。右端项 *b* 的第 1 行元素不为零，故对矩阵 ***L*** 的第 1 列进行深度优先搜索，即开始执行 *dfs*(1)。从矩阵 ***L*** 的第 1 列向下搜索，发现(1, 3)为非零元，于是执行 *dfs*(3)；从第 3 列向下搜索，发现(3, 5)为非零元，于是执行 *dfs*(5)；从第 5 列向下搜索，发现(5, 8)为非零元，执行 *dfs*(8)；第 8 列无其他非零元，于是 *dfs*(8)结束。至此，过程可参考图 3-3(a)。

dfs(8)结束后，递归程序返回 *dfs*(5)，而第 5 列无其他非零元，*dfs*(5)结束；

递归返回 *dfs*(3)，在第 3 列继续向下搜索，发现(3, 6)为非零元，执行 *dfs*(6)；在第 6 列向下搜索，发现(6, 7)为非零元，执行 *dfs*(7)；第 7 列的(7, 8)，但节点 8 已压入 χ 的栈中，故 *dfs*(7)结束。至此的过程可见图 3-3(b)。

dfs(7)结束后，递归程序先后返回 *dfs*(6)、*dfs*(3)与 *dfs*(1)，最终 *dfs*(1)结束。随后，继续判断右端项，因 *b* 的第 2 行为 0，而节点 3 已压入栈中，故跳过直接执行 *dfs*(4)，矩阵 ***L*** 的第 4 列无其他非零元素，*dfs*(4)直接结束，如图 3-3(c)。随后，由于元素 5~8 都已压入栈内，整个 reach 过程结束，如图 3-3(d)。

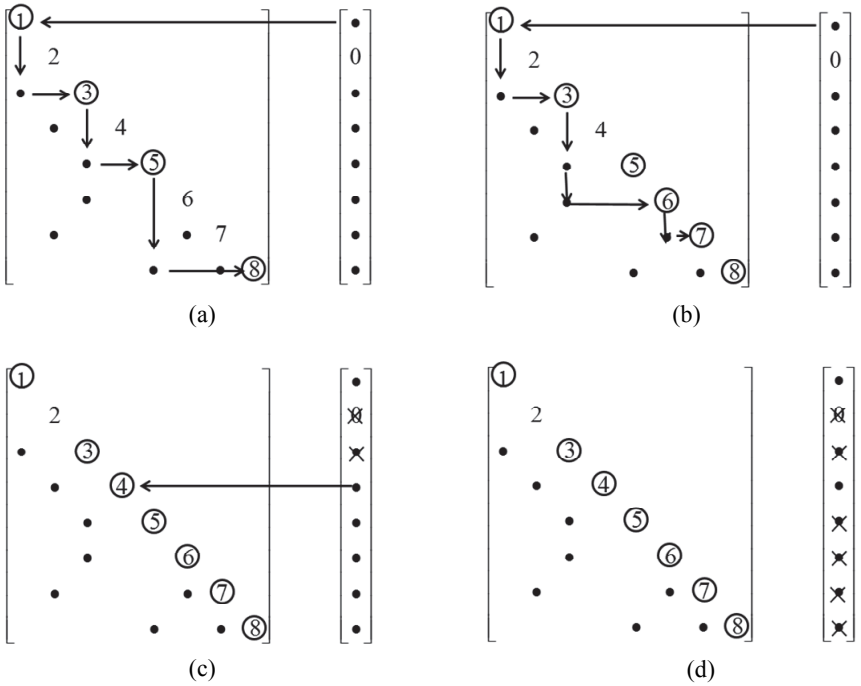


图 3-3 稀疏三角方程的 reach 过程

Fig. 3-3 The reach process of sparse triangular system

在该示例，递归算法中深度优先搜索的调用过程如下：

```
dfs(1){
   dfs(3){
      dfs(5)
         {dfs(8)}
      dfs(6)
         {dfs(7)}
   }
}
dfs(4)
```

通过对该矩阵的 reach 操作，所得到的集合 χ 为{4, 1, 3, 6, 7, 5, 8}。可根据该

集合，仅对矩阵 $\boldsymbol{L}$ 中有价值的列进行运算，从而降低整个稀疏三角方程组求解的计算复杂度。

3.3 稀疏分解算法研究

3.3.1 稀疏 Cholesky 分解概述

在 LM 方法求解电力系统病态潮流中，涉及稀疏矩阵计算的是迭代步的求解，即

$$-(\boldsymbol{J}(x_k)^T\boldsymbol{J}(x_k)+\mu_k\boldsymbol{I})\Delta\boldsymbol{x}_k=\boldsymbol{J}(x_k)^T\boldsymbol{F}(x_k) \tag{3-8}$$

根据本文 2.3 节的论述，采用直接求解方案对式(3-8)进行计算，其中，线性方程组的系数矩阵为 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$。由于 LM 方法中的阻尼因子 μ 是一个正数，矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 是一个对称正定矩阵，故可以采用 Cholesky 分解算法实现对式(3-8)的求解。

一般地，若有线性方程组形如

$$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b} \tag{3-9}$$

其中，$\boldsymbol{A}$ 是对称正定矩阵。

对于式(3-9)，使用 Cholesky 分解算法进行求解，即首先将矩阵 A 分解为如下形式：

$$\boldsymbol{A}=\boldsymbol{L}\boldsymbol{L}^T \tag{3-10}$$

其中矩阵 $\boldsymbol{L}$ 为下三角矩阵。

将式(3-10)代入(3-9)可得

$$\boldsymbol{L}\boldsymbol{L}^T\boldsymbol{x}=\boldsymbol{b} \tag{3-11}$$

对于式(3-11)，令

$$\boldsymbol{L}^T\boldsymbol{x}=\boldsymbol{y} \tag{3-12}$$

将式(3-12)代入(3-11)，可得

$$\boldsymbol{L}\boldsymbol{y}=\boldsymbol{b} \tag{3-13}$$

由于矩阵 $\boldsymbol{L}$ 已经通过式(3-10)矩阵分解求得，向量 $\boldsymbol{y}$ 可以通过式(3-13)的一个简单的矩阵求逆而求得。随后，由于向量 $\boldsymbol{y}$ 已经求出，根据式(3-12)，可以再通过一次矩阵求逆求出解向量 $\boldsymbol{x}$。

以上是 Cholesky 分解的数学原理，而在具体的稀疏实现中，一般分为四步：节点预排序、符号分解、数值分解、前代回代。

（1）节点预排序

根据高斯消元法的经验，若在不对矩阵进行任何处理的前提下直接求解式(3-9)，矩阵 $\boldsymbol{A}$ 在计算过程中通常会产生大量的注入元，影响计算效率。通过恰当的节点预排序，可以大幅减少注入元。使用节点排序，使得注入元达到最小的问题本身是一个 NP 完全问题，因此一般都会采用近似的算法来达到次优的效果。相关的排序算法最早由 William W. Tinney 于 1967 年提出[63]，包括 Tinney-1、Tinney-2、Tinney-3 三种算法。随后，相关研究又受到了大量学者的关注。在 1996 年，Patrick R. Amestoy 和 Timothy A. Davis 提出了 AMD 排序算法[64]，通过大量的测试证明了算法良好的效果。

考虑到节点预排序的算法已经比较成熟，本文不再对相关算法进行赘述。在相关的矩阵分解前，本文均采用 AMD 算法进行排序，从而减小注入元，提升计算速度。

（2）符号分解

在具体的矩阵分解之前，考虑到矩阵的稀疏特性，需要开辟合适大小的空间用于存储最终的分解结果，即矩阵 $\boldsymbol{L}$。由于在分解过程中会产生一定的注入元，将矩阵 $\boldsymbol{L}$ 与矩阵 $\boldsymbol{A}$ 开辟同样地大小显然是不够的。通常希望开辟出足够存放矩阵 $\boldsymbol{L}$ 的空间，同时保证开辟的空间又不会远大于实际的矩阵 $\boldsymbol{L}$ 大小，导致内存的浪费。就需要在分解前能够根据矩阵 $\boldsymbol{A}$ 的非零结构对矩阵 $\boldsymbol{L}$ 的非零结构进行判断，预估出矩阵 $\boldsymbol{L}$ 中的大致非零元个数，从而开辟出合适的空间，为后续的实际分解做准备。

这样一个根据矩阵 $\boldsymbol{A}$ 推测矩阵 $\boldsymbol{L}$ 稀疏结构的过程就是符号分解。值得一提的是，符号分解的过程仅依赖于矩阵 $\boldsymbol{A}$ 的稀疏结构，而与具体数值无关。在潮流计算的过程中，雅克比矩阵 $\boldsymbol{J}$ 中非零元的值会随着迭代的过程而发生变化，但矩阵本身的非零结构通常不会变化。因此一般而言，整个潮流计算的过程中，符号分解仅需要执行一次即可。

（3）数值分解

数值分解是在符号分解的基础上，求解出实际的矩阵 $\boldsymbol{L}$ 的非零元数值，即完成式(3-10)的求解。对于整个矩阵分解过程而言，数值分解往往是计算的主体，其耗时占据了整个过程的绝大部分。Cholesky 的稀疏数值分解算法一般可分为向上看 Cholesky 分解和向左看（超节点）Cholesky 分解。

（4）前代回代

当通过数值分解过程求得矩阵 $\boldsymbol{L}$ 后，可先后通过式(3-13)和式(3-12)，最终求

得稀疏线性方程组的解 $\boldsymbol{x}$，其中，求解式(3-13)的过程称为前代过程，而求解式(3-12)的过程即为回代过程。可以看出，由于矩阵 $\boldsymbol{L}$ 是稀疏下三角矩阵，前代与回代计算实际上就是两次求解线性三角方程组的过程，而其求解方法已经在本文 3.2.3 节进行了阐述。

下面对稀疏 Cholesky 分解的符号分解和数值分解算法进行具体的研究。

3.3.2 稀疏 Cholesky 符号分解

稀疏 Cholesky 符号分解的目的是通过矩阵 $\boldsymbol{A}$ 得到矩阵 $\boldsymbol{L}$ 的稀疏结构，从而为具体的数值分解开辟合适的存储空间。由于稀疏矩阵采用 CSC 形式存储，符号分解需要对分解后矩阵 $\boldsymbol{L}$ 每一列的非零元个数进行统计。

符号分解的过程与矩阵的图之间有较大的关系，因此首先对相关概念进行介绍。

（1）消去树（elimination tree）

一个对称的稀疏矩阵 $\boldsymbol{A}$ 可以与一个无向图 G 一一对应，其中，图 G 的每一个节点对应矩阵 $\boldsymbol{A}$ 的一行或一列；而矩阵 $\boldsymbol{A}$ 中的 i 行 j 列元素为非零元，或者说 $a_{ij}\neq 0$ 则对应图 G 中的 ij 节点间有边相连。

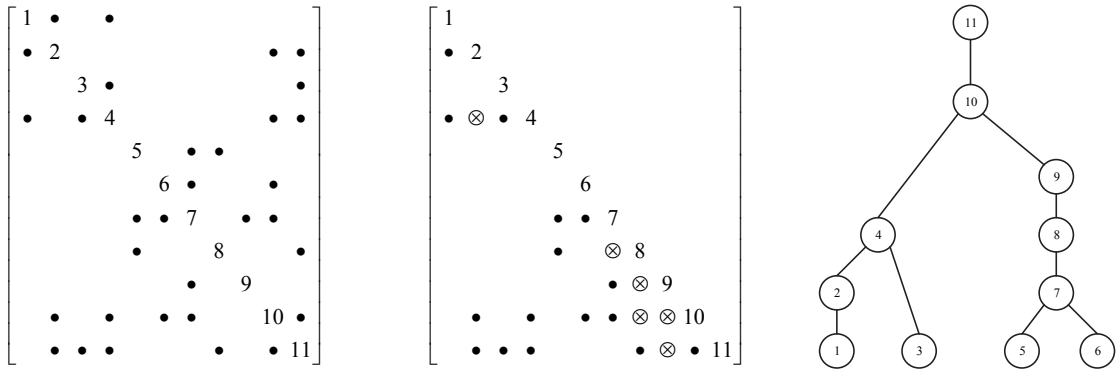
对于矩阵 $\boldsymbol{A}$ 在 Cholesky 分解后的结果 $\boldsymbol{L}$，令 $G_{\boldsymbol{L}+\boldsymbol{L}^T}$ 为 $\boldsymbol{L}+\boldsymbol{L}^T$ 的无向图，称之为矩阵 $\boldsymbol{A}$ 的注入图。因为注入图是根据矩阵 $\boldsymbol{L}+\boldsymbol{L}^T$ 而得，能够反映出注入元的情况。而矩阵 $\boldsymbol{A}$ 的消去树则是对注入图 $G_{\boldsymbol{L}+\boldsymbol{L}^T}$ 进行剪裁后的结果。下面首先对 Cholesky 分解中非零元位置的规律给出两条定理：

定理 3.1[59] 对于 Cholesky 分解 $\boldsymbol{L}\boldsymbol{L}^T=\boldsymbol{A}$，同时不考虑数值分解过程中的非零元消除，则 $a_{ij}\neq 0\Rightarrow l_{ij}\neq 0$。即若 a_{ij} 不为零，则 l_{ij} 也必不为零。

定理 3.2[65] 对于 Cholesky 分解 $\boldsymbol{L}\boldsymbol{L}^T=\boldsymbol{A}$，同时不考虑数值分解过程中的非零元消除，则 $i<j<k\wedge l_{ji}\neq 0\wedge l_{ki}\neq 0\Rightarrow l_{kj}\neq 0$。即若 l_{ji} 和 l_{ki} 都不为零，而 $i<j<k$，则 l_{kj} 也不为零。

定理 3.1 与定理 3.2 描述了矩阵 $\boldsymbol{L}$ 的非零元规律，特别地，根据定理 3.2，若原来 a_{kj} 为零，则新生成的非零元 l_{kj} 即为产生的注入元。从图的角度进行考虑，若 $l_{ji}\neq 0\wedge l_{ki}\neq 0$，即注入图 $G_{\boldsymbol{L}+\boldsymbol{L}^T}$ 中有 ij 边与 ik 边，则必有 jk 边。而从路径上来看，从节点 i 至节点 k 存在 ij-jk 这样一条路径，则原来的 ik 边存在与否对于图的深度优先搜索就没有意义，可以剪裁掉。这样，通过对 $G_{\boldsymbol{L}+\boldsymbol{L}^T}$ 中无用的边进行剪裁后，所得的图即为消去树。在消去树中，节点 i 的父节点为节点 j，则在矩阵 $\boldsymbol{L}$

中，第 i 列的第一个下对角非零元为 l_{kj}。矩阵 $\boldsymbol{A}$，其 Cholesky 分解矩阵 $\boldsymbol{L}$，以及矩阵 $\boldsymbol{A}$ 的消去树示例可参照图 3-4。



(a) 矩阵$\boldsymbol{A}$　　(b) 分解后的矩阵$\boldsymbol{L}$　　(c) 矩阵$\boldsymbol{A}$的消去树

图 3-4 消去树示例

Fig. 3-4 Example of elimination tree

在图 3-4 中，•表示了非零元，⊗表示分解中产生的注入元。将矩阵的消去树记为$\mathcal{T}$。

（2）消去行子树(elimination row subtree)

进一步定义矩阵 $\boldsymbol{L}$ 的第 k 列非零结构为 $Struct(\boldsymbol{L}_{*k})$，由以下定理所得到：

定理 3.3[66] 矩阵 $\boldsymbol{L}$ 第 k 列非零结构 $Struct(\boldsymbol{L}_{*k})$可通过以下方式求得

$$Struct(\boldsymbol{L}_{*k}) = Reach_{G_{k-1}}(Struct(\boldsymbol{A}_{*k})) = Reach_{\mathcal{T}_{k-1}}(Struct(\boldsymbol{A}_{*k})) \tag{3-14}$$

其中，G_{k-1} 与 $\mathcal{T}_{k-1}$ 分别代表矩阵 $\boldsymbol{A}$ 前 k-1 行 k-1 列的注入图以及对应的消去树，Reach 函数定义可参见本文 3.2.3 节，$Struct(\boldsymbol{A}_{*k})$为矩阵 $\boldsymbol{A}$ 的第 k 列下三角部分非零结构。

定理 3.3 定义了 $Struct(\boldsymbol{L}_{*k})$的求取方式，而消去树$\mathcal{T}$的 k 行子树 $\mathcal{T}^k$，就是由所有$\mathcal{L}_k$中所包含节点所组成的消去树$\mathcal{T}$的子树。以图 3-4 中的矩阵 $\boldsymbol{A}$ 以及消去树$\mathcal{T}$为例，消去树的所有行子树如图 3-5 所示。

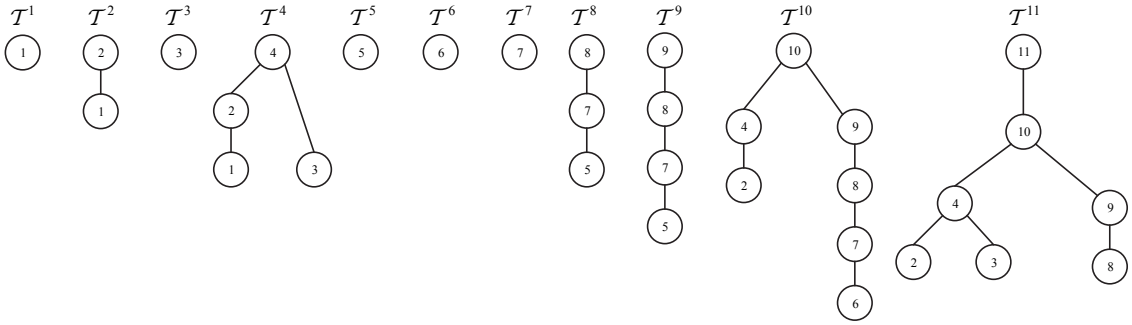


图 3-5 消去树子树示例

Fig. 3-5 Example of elimination row subtrees

（3）骨架矩阵(skeleton matrix)

对于矩阵 $\boldsymbol{A}$，定义其骨架矩阵 $\hat{\boldsymbol{A}}$，若节点 j 是消去树 i 行子树 $\mathcal{T}^i$ 的叶节点，则认为骨架矩阵 $\hat{\boldsymbol{A}}$ 的 i 行 j 列元素 $\hat{a}_{ij}=a_{ij}$，否则 $\hat{a}_{ij}$ 为零。骨架矩阵 $\hat{\boldsymbol{A}}$ 经 Cholesky 分解后的非零结构与原始矩阵 $\boldsymbol{A}$ 相同。同时，若 $\hat{a}_{ij}\neq 0$，则 $a_{ij}\neq 0$；而反之则未必正确。

以图 3-4 的矩阵 $\boldsymbol{A}$ 为例，并根据其消去行子树图 3-5，可得其骨架矩阵的非零结构如图 3-6。

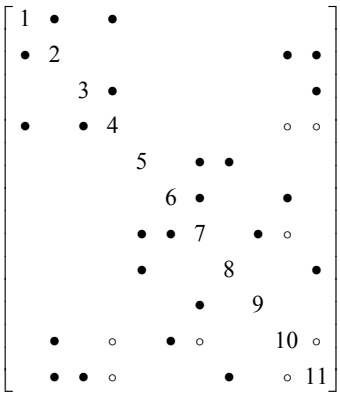


图 3-6 骨架矩阵示例

Fig. 3-6 Example of skeleton matrix

图 3-6 中，空心符号◦表示该元素在矩阵 $\boldsymbol{A}$ 中为非零元，但在矩阵 $\hat{\boldsymbol{A}}$ 中是零元。

稀疏 Cholesky 符号分解的主要目的是获得矩阵每一列的非零元个数，而列的非零元结构 $Struct(\boldsymbol{L}_{*k})$可由以下定理而迭代求出：

定理 3.4[67] 若将 $Struct(\boldsymbol{L}_{*j})$记为矩阵 $\boldsymbol{L}$ 第 j 列的非零结构，$Struct(\boldsymbol{A}_{*j})$为矩阵 $\boldsymbol{A}$ 的第 j 列下三角非零结构，则

$$Struct(\boldsymbol{L}_{*j})=Struct(\boldsymbol{A}_{*j})\cup\{j\}\cup\left(\bigcup_{j\text{是}s\text{的父节点}}Struct(\boldsymbol{L}_{*s})\setminus\{s\}\right) \tag{3-15}$$

定理 3.4 表明，矩阵 $\boldsymbol{L}$ 第 j 列的非零结构，是矩阵 $\boldsymbol{A}$ 的第 j 列非零结构，加上节点 j，并与 j 在消去树$\mathcal{T}$中所有子节点 s 对应列非零结构除去自身后的并集。该定理可转化为以下求取公式：

$$c_j=\left|Struct(\hat{\boldsymbol{A}}_{*j})\right|-e_j-o_j+\sum_{j\text{是}s\text{的父节点}}Struct(\boldsymbol{L}_{*s}) \tag{3-16}$$

其中，c_j 表示矩阵 $\boldsymbol{L}$ 第 j 列的非零元个数，$Struct(\hat{\boldsymbol{A}}_{*j})$ 表示骨架矩阵 $\hat{\boldsymbol{A}}$ 第 j 列下三角非零结构，e_j 表示节点 j 在消去树$\mathcal{T}$中的子节点个数，o_j 表示所有 j 的子节点非

零结构之间的重叠数。

以图 3-4 中矩阵 $\boldsymbol{L}$ 的第 4 列为例，其非零结构为{4, 10, 11}，而节点 4 在消去树中有两个子节点，节点 2 和节点 3，而$\mathcal{L}_2$为{2, 4, 10, 11}，$\mathcal{L}_3$为{3, 4, 11}，根据公式(3-16)，e_4为节点 4 的子节点个数 2；o_4为$\mathcal{L}_2$和$\mathcal{L}_3$的重叠节点{4, 11}个数，即 2；而 c_2和 c_3分别为 4 和 3，故 c_4为 0-2-2+4+3=3。

关于重叠数 o_j的计算，可通过消去行子树中每个节点计算其对重叠数的贡献。令节点 j 是 i 行子树 $\mathcal{T}^i$ 的一个节点，也即 $i \in Struct(\boldsymbol{L}_{*j})$，则该节点有以下两种情况：

1）若 j 是 $\mathcal{T}^i$ 的叶节点，则根据骨架矩阵的定义，$\hat{a}_{ij} \neq 0$。那么由于节点 i 不出现在节点 j 的任何子节点的非零结构中，第 i 行就不对重叠数 o_j作出贡献。

2）若 j 不是 $\mathcal{T}^i$ 的叶节点，即 d_{ij}为 j 在 $\mathcal{T}^i$ 中的子节点个数。由于节点 i 会出现在这 d_{ij}个子节点的非零结构中，故第 i 行对 o_j的贡献值为 d_{ij}-1。

通过求解出所有的重叠数 o_j，即可求出所有的列非零元个数。该过程可通过以下引理抽象化表示出来：

引理 3.1[68] 矩阵 $\boldsymbol{A}$ 经过 Cholesky 分解结果 $\boldsymbol{L}$ 的第 j 列非零元个数 c_j 可通过以下公式进行计算

$$c_j = \sum_i \sum_{s=j\cup\{j\text{的子节点}\}} w_i(s) \tag{3-17}$$

其中，i 为矩阵 $\boldsymbol{A}$ 的所有行号，s 为 j 自身与 j 在消去树中所有子节点的并集，$w_i(s)$为矩阵 $\boldsymbol{A}$ 第 i 行元素对 $\boldsymbol{L}$ 第 s 列非零元个数的贡献；$w_i(s)$可以通过矩阵 $\boldsymbol{A}$ 的非零结构求得

引理 3.2[68]

$$w_i(s) = \begin{cases} 1-d_{is} & i<s\text{且}a_{is}\neq 0 \\ -1 & i\text{是s的子节点} \\ 0 & \text{其他} \end{cases} \tag{3-18}$$

其中，d_{is}为节点 s 在矩阵 $\boldsymbol{A}$ 第 i 行的子节点个数，a_{is}为矩阵 $\boldsymbol{A}$ 第 i 行 s 列的元素。

通过对原始矩阵 $\boldsymbol{A}$ 的非零元进行遍历，根据(3-18)可求得 $w_i(s)$，随后根据式(3-17)即可求出矩阵 $\boldsymbol{L}$ 所有列的非零元个数。由此，即可完成对于矩阵 $\boldsymbol{A}$ 的稀疏 Cholesky 符号分解过程。

3.3.3 向上看 Cholesky 分解

通过符号分解的过程，可以获得 Cholesky 分解后矩阵 $\boldsymbol{L}$ 每一列的非零元个数，随后即可开始进行实际的数值分解过程，求取 $\boldsymbol{L}$ 中每一个非零元的具体数值。

针对稀疏矩阵的 Cholesky 数值分解算法中，向上看（left-looking）Cholesky 分解算法是一个非常经典的算法。该算法的数学原理最早由 Joseph W. H. Liu 提出[69]，而 Timothy A. Davis 于 2005 年使用精简的代码对其进行了实现[70]。

向上看 Cholesky 分解在每一步迭代中求取矩阵 $\boldsymbol{L}$ 的一行。若假设 $\boldsymbol{L}$ 的前 k-1 行均已求得，要求取第 k 行元素，将式(3-10)的前 k 行 k 列改写成 2×2 块状形式

$$\begin{bmatrix} \boldsymbol{L}_{11} & \\ \boldsymbol{l}_{12}^T & l_{22} \end{bmatrix}\begin{bmatrix} \boldsymbol{L}_{11}^T & \boldsymbol{l}_{12} \\ & l_{22} \end{bmatrix}=\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{a}_{12} \\ \boldsymbol{a}_{12}^T & a_{22} \end{bmatrix} \tag{3-19}$$

其中 $\boldsymbol{L}_{11}$ 和 $\boldsymbol{A}_{11}$ 分别表示 $\boldsymbol{L}$ 和 $\boldsymbol{A}$ 前 $(k-1)\times(k-1)$ 阶子阵，$\boldsymbol{l}_{12}$ 和 $\boldsymbol{a}_{12}$ 分别表示 $\boldsymbol{L}^T$ 和 $\boldsymbol{A}$ 第 k 列第 1 行到第 $k-1$ 行元素组成的列向量，l_{22} 和 a_{22} 分别表示 $\boldsymbol{L}$ 和 $\boldsymbol{A}$ 第 k 行第 k 列的元素。

由式(3-19)，易得

$$\begin{cases} \boldsymbol{L}_{11}\boldsymbol{l}_{12}=\boldsymbol{a}_{12} & ① \\ l_{22}=\sqrt{a_{22}-\boldsymbol{l}_{12}^T\boldsymbol{l}_{12}} & ② \end{cases} \tag{3-20}$$

其中，由式(3-20)-①可以通过一次下三角求解解得 $\boldsymbol{l}_{12}$，通过式(3-20)-②可以求得 l_{22}，从而完成矩阵 $\boldsymbol{L}$ 第 k 行元素的求解。

根据式(3-20)，从 k=1 至 n 迭代求解，即可求出矩阵 $\boldsymbol{L}$ 的 1 至 n 行数值分解结果，从而完成整个数值分解过程。该向上看 Cholesky 分解算法可用以下伪代码实现：

for $k=1$ to n **do**
    $b(1:k)=A(k,1:k)$
    $L(k,1:k-1)=L(1:k-1,1:k-1)\backslash b(1:k-1)$
    $L(k,k)=\sqrt{b(k)-L(k,1:k-1)*L(k,1:k-1)}$
end for

3.3.4 超节点 Cholesky 分解

超节点 Cholesky 分解算法基于向左看（left-looking）Cholesky 分解算法，下面首先介绍向左看 Cholesky 分解的原理进行介绍，再研究超节点的概念以及超

节点 Cholesky 分解算法。

（1）向左看 Cholesky 分解

不同于向上看 Cholesky 分解的按行计算矩阵 $\boldsymbol{L}$，向左看 Cholesky 分解算法从左至右逐列计算矩阵 $\boldsymbol{L}$ 的数值。若假设 $\boldsymbol{L}$ 的前 k-1 列均已求得，要求取第 k 列元素，将式(3-10)的改写成 3×3 块状形式

$$\begin{bmatrix} \boldsymbol{L}_{11} & & \\ \boldsymbol{l}_{12}^T & l_{22} & \\ \boldsymbol{L}_{31} & \boldsymbol{l}_{32} & \boldsymbol{L}_{33} \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_{11}^T & \boldsymbol{l}_{12} & \boldsymbol{L}_{31}^T \\ & l_{22} & \boldsymbol{l}_{32}^T \\ & & \boldsymbol{L}_{33}^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{a}_{12} & \boldsymbol{A}_{31}^T \\ \boldsymbol{a}_{12}^T & a_{22} & \boldsymbol{a}_{32}^T \\ \boldsymbol{A}_{31} & \boldsymbol{a}_{32} & \boldsymbol{A}_{33} \end{bmatrix} \tag{3-21}$$

其中 $\boldsymbol{L}_{11}$ 和 $\boldsymbol{A}_{11}$ 分别表示 $\boldsymbol{L}$ 和 $\boldsymbol{A}$ 前 $(k-1)\times(k-1)$ 阶子阵，$\boldsymbol{l}_{12}$ 和 $\boldsymbol{a}_{12}$ 分别表示 $\boldsymbol{L}^T$ 和 $\boldsymbol{A}$ 第 k 列第 1 行到第 $k-1$ 行元素组成的列向量，l_{22} 和 a_{22} 分别表示 $\boldsymbol{L}$ 和 $\boldsymbol{A}$ 第 k 行第 k 列的元素，$\boldsymbol{L}_{31}$ 和 $\boldsymbol{A}_{31}$ 分别表示 $\boldsymbol{L}$ 和 $\boldsymbol{A}$ 第 $k+1$ 至第 n 行，以及第 1 列至第 $k-1$ 列构成的子阵，$\boldsymbol{l}_{32}$ 和 $\boldsymbol{a}_{32}$ 分别表示 $\boldsymbol{L}$ 和 $\boldsymbol{A}$ 第 k 列第 $k+1$ 至第 n 行元素组成的列向量，$\boldsymbol{L}_{33}$ 和 $\boldsymbol{A}_{33}$ 分别表示 $\boldsymbol{L}$ 和 $\boldsymbol{A}$ 后 $(n-k)\times(n-k)$ 阶子阵。

对式(3-21)进行矩阵乘法，选取矩阵 $\boldsymbol{L}$ 的第二、三行与 $\boldsymbol{L}^T$ 第二列相乘结果，可以得到

$$\begin{cases} \boldsymbol{l}_{12}^T\boldsymbol{l}_{12} + l_{22}^2 = a_{22} \\ \boldsymbol{L}_{31}\boldsymbol{l}_{12} + l_{22}\boldsymbol{l}_{32} = \boldsymbol{a}_{32} \end{cases} \tag{3-22}$$

对式(3-22)进行进一步变换，即可求出矩阵 $\boldsymbol{L}$ 第 k 列的数值

$$\begin{cases} l_{22} = \sqrt{a_{22} - \boldsymbol{l}_{12}^T\boldsymbol{l}_{12}} & ① \\ \boldsymbol{l}_{32} = (\boldsymbol{a}_{32} - \boldsymbol{L}_{31}\boldsymbol{l}_{12}) / l_{22} & ② \end{cases} \tag{3-23}$$

向左看 Cholesky 分解算法即基于式(3-23)进行每一列的求解。其中，算法进一步对②式进行改写

$$\boldsymbol{l}_{32} = (\boldsymbol{a}_{32} - \sum_{j=1}^{k-1} \boldsymbol{L}_{31(*\mathrm{j})}\boldsymbol{l}_{12(\mathrm{j})}) / l_{22} \tag{3-24}$$

其中，$\boldsymbol{L}_{31(*\mathrm{j})}$ 代表矩阵 $\boldsymbol{L}_{31}$ 的第 j 列，而 $\boldsymbol{l}_{12(\mathrm{j})}$ 代表向量 $\boldsymbol{l}_{12}$ 的第 j 行元素。

采用式(3-24)计算 Cholesky 分解，是将前 k-1 列数值以逐列的形式对当前第 k 列数值进行更新，从而完成计算。考虑到矩阵的稀疏特性，式(3-24)中求和项的 j 列应选取 $\boldsymbol{l}_{12(\mathrm{j})}$ 不为零的项。该算法可通过以下伪代码进行实现：

for $k=1$ to n **do**
  $a(k:n)=A(k:n,k)$
  for j **such that** $L(k,j)\neq 0$ **do**
    $a(k:n)=a(k:n)-L(k:n,j)*L(k,j)$
  end for
  $L(k,k)=\sqrt{a(k)}$
  $L(k+1:n,k)=a(k+1:n)/L(k,k)$
end for

（2）超节点 Cholesky 分解

由于向左看 Cholesky 分解的基本计算都是以列为单位进行的，可认为该算法为列-列级的 Cholesky 分解。而在此基础上，许多学者们进一步提出了超节点（supernodal）Cholesky 分解算法。超节点的概念最早于 1987 年由 C. Cleveland Ashcraft 等人提出[71]，随后，Esmond G. Ng 等人于 1991 年提出超点-列级的分解算法[72]，并于 1993 年提出超点-超点级的分解算法[73]。为了研究超节点 Cholesky 分解的算法原理，需要首先对超节点的概念进行了解。

超节点是指在矩阵 $\boldsymbol{L}$ 中，除去对角部分外，其余部分的稀疏结构完全相同的连续列。这几个连续的列，或者说这几列对应的节点，即可组成一个大的超节点。更为正式地说，如果 $Struct(\boldsymbol{L}_{*k})=Struct(\boldsymbol{L}_{*k+1})\cup\{k+1\}\ (j\le k\le j+t-1)$，则这些连续的列$\{j, j+1, \ldots, j+t-1\}$组成一个超节点。关于超节点的示例，可见图 3-7。

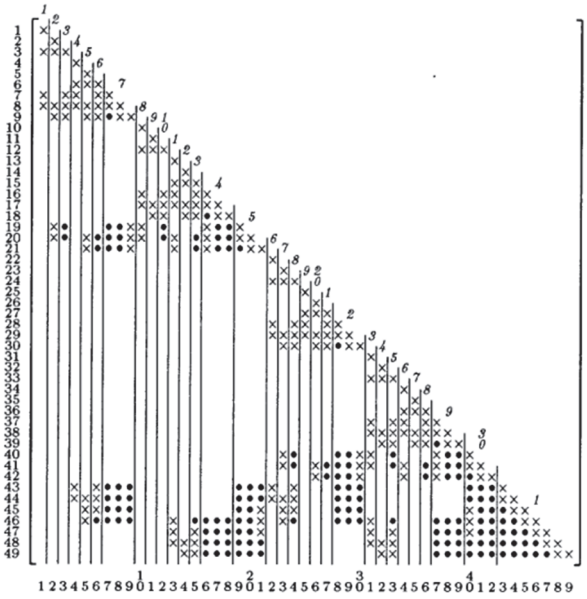


图 3-7 超节点示例
Fig. 3-7 Example of supernodes

在图 3-7 中，×表示原始矩阵 $\boldsymbol{A}$ 的非零元，而•表示分解过程中产生的注入元，超节点之间用竖线间隔开。可以看出，矩阵共可分为 31 个超节点，例如矩阵第 7 至第 9 列即组成一个超节点。

对矩阵的超节点进行分析后，即可以超节点为单位，进行 Cholesky 数值分解。其中，超节点-列级的 Cholesky 分解原理依旧基于式(3-23)，不同于向左看 Cholesky 分解采用式(3-24)以列为单位计算式(3-23)-②，超节点-列级的 Cholesky 分解直接以超节点为一个单位进行式②的计算。

$$\boldsymbol{l}_{32}=(\boldsymbol{a}_{32}-\sum_{J=1}^{N}\boldsymbol{L}_{31(*J)}\boldsymbol{l}_{12(J)})/l_{22} \tag{3-25}$$

其中，N 为所有超节点的总数，J 代表其中的一个超节点，$\boldsymbol{L}_{31(*J)}$ 为矩阵 $\boldsymbol{L}_{31}$ 中超节点 J 的对应列，$\boldsymbol{l}_{12(J)}$ 为向量 $\boldsymbol{l}_{12}$ 中超节点 J 的对应行。

该算法可以通过以下伪代码实现

for $K=1$ to N **do**
  for $k\in K$ **do**
    $a(k:n)=A(k:n,k)$
    for J **such that** $Struct(L(k,J))\neq 0$ **do**
      $j_1=\min\{J\}$
      $j_2=\min\{J+1\}$
      $j=j_1:j_2-1$
      $a(k:n)=a(k:n)-L(k:n,j)*L(k,j)$
    end for
    $L(k,k)=\sqrt{a(k)}$
    $L(k+1:n,k)=a(k+1:n)/L(k,k)$
  end for
end for

其中，j_1 和 j_2 分别为超节点 J 和下一个超节点 J+1 的首列号。

可以看出，超节点-列级的 Cholesky 分解过程与向左看 Cholesky 分解的过程非常类似，主要的区别体现在前者在更新 k 列时不再以列为单位，而是以超节点中的多个列为单位进行，因此便包含了矩阵-向量操作。

超节点-超节点级的 Cholesky 分解则更进一步，不再是一次求解 $\boldsymbol{L}$ 的一列，而是求解一个超节点对应的多个列。再一次将式(3-10)的改写成 3×3 块状形式

$$\begin{bmatrix} \boldsymbol{L}_{11} & & \\ \boldsymbol{L}_{12}^T & \boldsymbol{L}_{22} & \\ \boldsymbol{L}_{31} & \boldsymbol{L}_{32} & \boldsymbol{L}_{33} \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_{11}^T & \boldsymbol{L}_{12} & \boldsymbol{L}_{31}^T \\ & \boldsymbol{L}_{22} & \boldsymbol{L}_{32}^T \\ & & \boldsymbol{L}_{33}^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \boldsymbol{A}_{31}^T \\ \boldsymbol{A}_{12}^T & \boldsymbol{A}_{22} & \boldsymbol{A}_{32}^T \\ \boldsymbol{A}_{31} & \boldsymbol{A}_{32} & \boldsymbol{A}_{33} \end{bmatrix} \tag{3-26}$$

与式(3-21)不同的是，其中 $\boldsymbol{L}_{12}$、$\boldsymbol{A}_{12}$ 等不再是矩阵第 k 行/列的对应元素，而是超节点 K 对应行/列所形成的矩阵块。

对式(3-26)进行矩阵乘法，可以得到

$$\begin{cases} \boldsymbol{L}_{22}^T \boldsymbol{L}_{22} = \boldsymbol{A}_{22} - \boldsymbol{L}_{12}^T \boldsymbol{L}_{12} \\ \boldsymbol{L}_{22} \boldsymbol{L}_{32} = \boldsymbol{A}_{32} - \boldsymbol{L}_{31} \boldsymbol{L}_{12} \end{cases} \tag{3-27}$$

进一步地，式(3-27)可以进行如下求解

$$\begin{cases} \boldsymbol{L}_{22} = chol(\boldsymbol{A}_{22} - \boldsymbol{L}_{12}^T \boldsymbol{L}_{12})^T \\ \boldsymbol{L}_{32} = (\boldsymbol{A}_{32} - \boldsymbol{L}_{31} \boldsymbol{L}_{12}) / \boldsymbol{L}_{22} \end{cases} \tag{3-28}$$

其中，$\boldsymbol{L}_{22}$ 通过 Cholesky 分解求得，而 $\boldsymbol{L}_{32}$ 通过求解矩阵求逆可以求得。该算法可通过以下伪代码进行表示：

for $K = 1$ to N **do**
  $k_1 = \min\{J\}$
  $k_2 = \min\{K+1\}$
  $k = k_1 : k_2 - 1$
  $L(k, k_1 : n) = A(k, k_1 : n)$
  for $J < K$ **do**
    $j_1 = \min\{J\}$
    $j_2 = \min\{J+1\}$
    $j = j_1 : j_2 - 1$
    $L(k,k) = L(k,k) - L(\mathrm{k}, j) * L(\mathrm{k}, j)^T$
    $L(k_2 : n, k) = L(k_2 : n, k) - L(k_2 : n, j) * L(k_2 : n, j)^T$
  end for
  $L(k,k) = chol(L(k,k))^T$
  $L(k_2 : n, k) = L(k_2 : n, k) / L(k,k)$
end for

可以看出超节点-超节点级 Cholesky 分解中，主要计算都变为矩阵-矩阵操作。由于超节点的特性，同一个超节点中对角比分必然是稠密的。由此，该算法可分为以下几个部分

1）对角部分更新，即 $L(k,k) = L(k,k) - L(\mathrm{k}, j) * L(\mathrm{k}, j)^T$，该部分计算为稠密

矩阵运算；

2）下三角部分更新，即 $L(k_2:n,k)=L(k_2:n,k)-L(k_2:n,j)*L(k_2:n,j)^T$，该部分为稀疏矩阵运算；

3）对角部分 Cholesky 分解，即 $L(k,k)=chol(L(k,k))^T$，是一个稠密的 Cholesky 分解运算；

4）矩阵求逆，即 $L(k_2:n,k)=L(k_2:n,k)/L(k,k)$，是对稠密三角矩阵 $L(k,k)$ 的求逆运算。

不论是超节点-列级 Cholesky 分解还是超节点-超节点 Cholesky 分解，通过超节点的引入，都可以将原本稀疏求解问题一定程度上转化为稠密运算。对于稠密矩阵的运算，可以通过基础线性代数子程序（Basic linear algebra subprograms, BLAS）进行加速。BLAS 是一系列线性代数运算的底层程序[74]，包括 1 级向量-向量操作、2 级矩阵-向量操作和 3 级矩阵-矩阵操作。BLAS 针对稠密矩阵而设计，计算效率很高。对于稀疏矩阵运算，若在恰当位置引入 BLAS，可极大地起到提速效果，且 BLAS 运算的层次越高，由于其访问存储器次数更少，导致其计算效率就越高，提速效果也更明显。

对于超节点-列级 Cholesky 分解，有稠密的矩阵-向量操作，可使用 2 级 BLAS 进行加速。而对于超节点-超节点级 Cholesky 分解，有稠密的矩阵-矩阵操作，可以使用 3 级 BLAS 进行加速。因而，一般采用超节点-超节点级的 Cholesky 分解，从而达到更高的加速效果。本文之后所提到的超节点 Cholesky 分解算法，也专指超节点-超节点级的 Cholesky 分解。

3.3.5 稀疏分解算法的适用性分析

对于稀疏 Cholesky 数值分解，主流的向上看 Cholesky 分解与超节点 Cholesky 分解有各自不同的计算特性。考虑到两个数值分解算法原理间的差异性，下面结合电力系统这一实际应用场景进行适用性分析。

对于 LM 方法计算系统病态潮流中求解迭代步的式(2-29)，对于稀疏线性方程组的稀疏矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 采用 AMD 算法进行节点预排序后，用 3.3.2 节的符号分解算法统计每列的非零元个数，随后，分别采用向上看 Cholesky 分解和超节点 Cholesky 分解算法，测试在第一步迭代时两个分解算法各自的计算效率。测试中，阻尼因子取值为 $\mu_k=0.001\left\|\boldsymbol{F}(\boldsymbol{x}_k)\right\|^2$。

程序的硬件测试环境参见 5.1 节的叙述，向上看 Cholesky 分解算法与超节点

Cholesky 分解算法的实现均基于 Timothy A. Davis 教授开发的 SuiteSparse 软件包，编译环境为 Visual Studio 2010。测试的系统信息见表 3-4。

表 3-4 算例测试系统基本概况
Table 3-4 Basic information of test systems

系统名称	节点数	支路数	雅克比矩阵阶数	数据来源
IEEE9 节点	9	9	14	IEEE9
IEEE14 节点	14	20	22	IEEE14
IEEE30 节点	30	41	53	IEEE30
IEEE39 节点	39	46	67	IEEE39
IEEE57 节点	57	78	106	IEEE57
IEEE300 节点	162	280	530	IEEE300
华东 3647 系统	3647	4239	7014	华东电网 2004 年
华东 4171 系统	4171	4860	7999	华东电网 2005 年
华东 4769 系统	4769	5597	9128	华东电网 2006 年
华东 2806 系统	2806	3369	5276	华东电网 2007 年
华东 5473 系统	5473	6642	10491	华东电网 2009 年
华东 7872 系统	7872	9830	15222	华东电网 2014 年夏低
华东 8003 系统	8003	9986	15465	华东电网 2014 年冬高
华东 8241 系统	8241	10280	16049	华东电网 2015 年夏低
华东 8386 系统	8386	10443	16261	华东电网 2015 年冬高

对表 3-4 中的系统进行测试，分别记录向上看 Cholesky 分解与超节点 Cholesky 分解两个算法的数值分解时间，结果如表 3-5。

表 3-5 向上看与超节点 Cholesky 数值分解时间
Table 3-5 Running time of up-looking and supernodal Cholesky factorization

系统名称	向上看算法 数值分解时间(s)	超节点算法 数值分解时间(s)	向上看比 超节点快(%)
IEEE9 节点	0.000019	0.000024	18.69%
IEEE14 节点	0.000031	0.000037	17.25%
IEEE30 节点	0.000068	0.000085	19.68%
IEEE39 节点	0.000085	0.000100	15.42%
IEEE57 节点	0.000140	0.000175	20.15%
IEEE300 系统	0.000808	0.000950	14.93%
华东 2806 系统	0.002811	0.003730	24.63%

续表 3-5 向上看与超节点 Cholesky 数值分解时间

系统名称	向上看算法 数值分解时间(s)	超节点算法 数值分解时间(s)	向上看比 超节点快(%)
华东 3647 系统	0.004294	0.005202	17.45%
华东 4171 系统	0.004536	0.005521	17.85%
华东 4769 系统	0.005248	0.006488	19.11%
华东 5473 系统	0.006053	0.007524	19.56%
华东 7872 系统	0.008060	0.010205	21.01%
华东 8003 系统	0.008366	0.010579	20.92%
华东 8241 系统	0.008777	0.011089	20.85%
华东 8386 系统	0.008915	0.011239	20.68%

将表 3-5 中的数值分解时间，根据系统的节点规模绘制曲线，其变化规律如图 3-8。

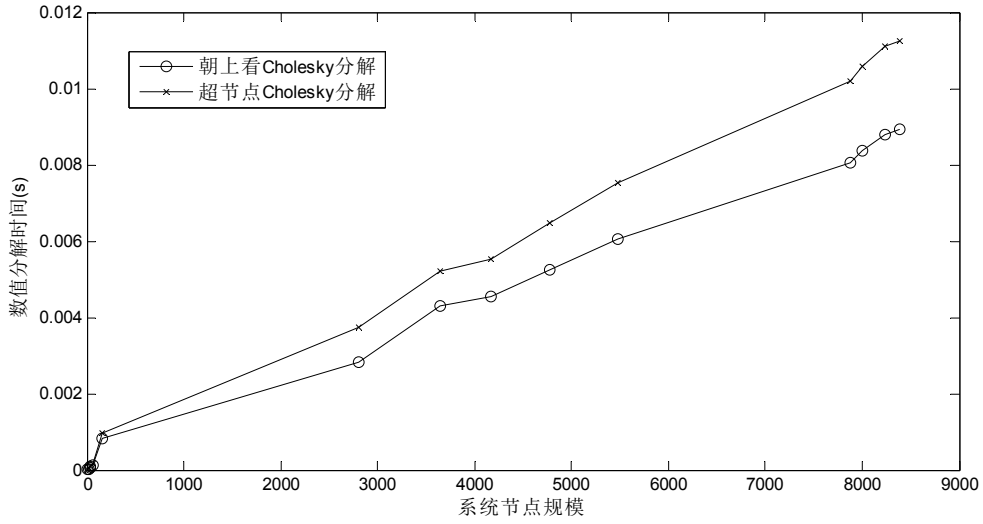


图 3-8 数值分解时间随系统规模变化图

Fig. 3-8 Running time of numerical factorization with respect to system scale

结合表 3-5 与图 3-8，发现两算法的数值分解时间整体都随系统规模增大而线性增大。而通过对两个方法分解时间的比较，可以发现超节点 Cholesky 分解虽然算法较为复杂，且引入 BLAS 进一步对算法进行了加速，在对电力系统矩阵进行分解时，其计算速度反而不如原理更为简单的向上看 Cholesky 分解。对于同一矩阵，向上看算法的计算时间大约比超节点算法少 20%左右。

为了更全面地测试两个算法的计算性能，本文进一步从佛罗里达大学稀疏矩阵库[75]中选取几个来自于其他领域的对称正定矩阵，对这些矩阵分别采用向上看

和超节点 Cholesky 分解算法，测试两者计算速度的差异。所选取矩阵的基本信息见表 3-6。

表 3-6 测试矩阵基本概况
Table 3-6 Basic information of test matrices

矩阵名称	矩阵规模	数据来源
bcsstk16	4884	结构问题
bcsstk17	10974	结构问题
ex3	1821	流体动力学
ex13	2568	流体动力学
ex15	6867	流体动力学
Pres_Poisson	14822	流体动力学
plat1919	1919	2D/3D 问题
fv1	9604	2D/3D 问题
fv2	9801	2D/3D 问题

对表 3-6 中系统的矩阵进行测试，统计两个算法的计算时间，见表 3-7。

表 3-7 向上看与超节点 Cholesky 数值分解时间
Table 3-7 Running time of up-looking and supernodal Cholesky factorization

矩阵名称	向上看算法 数值分解时间(s)	超节点算法 数值分解时间(s)	向上看比 超节点慢(%)
bcsstk16	0.184289	0.080331	129.41%
bcsstk17	0.169422	0.076105	122.62%
ex3	0.004287	0.003007	42.57%
ex13	0.004859	0.003854	26.08%
ex15	0.013874	0.010566	31.31%
Pres_Poisson	0.606681	0.250797	141.90%
plat1919	0.004683	0.003472	34.87%
fv1	0.023807	0.015697	51.66%
fv2	0.026352	0.016123	63.45%

由表 3-7 的测试结果，发现与表 3-5 的结果相反，对这些矩阵进行分解时，超节点 Cholesky 分解算法的计算效率更高。在最极端情况下，向上看算法的计算时间能达到超节点算法的近 2.5 倍。而一般而言，超节点 Cholesky 算法相较向上看 Cholesky 算法是更为主流高效的数值分解算法[59]。

根据以上测试可以发现，超节点 Cholesky 分解算法虽然在大部分应用场合具有更好的计算特性，在用于分解电力系统潮流计算中的 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵时，却无

法发挥出其应有的计算效率，不如向上看 Cholesky 分解。下面对其原因进行分析：

（1）超节点规模

超节点 Cholesky 分解的核心原理是利用超节点间的运算将原本的稀疏矩阵运算变为稠密矩阵运算，并利用 3 级 BLAS 进行加速，若超节点的规模越大，则稠密矩阵运算的规模也就越大，加速效果就比较明显。因此，超节点 Cholesky 分解过程中超节点的规模与算法的效率是有密切联系的。下面对表 3-4 与表 3-6 中涉及的矩阵统计超节点的规模，见表 3-8。

表 3-8 超节点算法中超节点的规模统计

Table 3-8 The scales of supernodes in supernodal Cholesky

矩阵/系统名称	矩阵阶数	超节点个数	最大超节点规模	平均超节点规模
IEEE9 节点	14	1	14	14
IEEE14 节点	22	2	15	11
IEEE30 节点	53	7	20	7.5714
IEEE39 节点	67	7	20	9.5714
IEEE57 节点	106	11	28	9.6364
IEEE300 节点	530	65	45	8.1538
华东 3647 系统	7014	787	46	8.9123
华东 4171 系统	7999	874	46	9.1522
华东 4769 系统	9128	1010	46	9.0376
华东 2806 系统	5276	639	46	8.2567
华东 5473 系统	10491	1214	42	8.6417
华东 7872 系统	15222	1800	44	8.4567
华东 8003 系统	15465	1826	46	8.4693
华东 8241 系统	16049	1921	46	8.3545
华东 8386 系统	16261	1936	46	8.3993
bcsstk16	4884	338	456	14.4497
bcsstk17	10974	1228	330	8.9365
ex3	1821	184	88	9.8967
ex13	2568	247	63	10.3968
ex15	6867	680	100	10.0985
Pres_Poisson	14822	830	500	17.8578
plat1919	1919	207	88	9.2705

续表 3-8 超节点算法中超节点的规模统计

矩阵/系统名称	矩阵阶数	超节点个数	最大超节点规模	平均超节点规模
fv1	9604	1168	176	8.2226
fv2	9801	1197	154	8.1880

由表 3-8 可以看出，对于电力系统矩阵与其余矩阵，平均的超节点规模基本相近，除了极个别矩阵外，大部分的超节点平均规模都不超过 10。两者的主要体现在最大的超节点规模上。电力系统矩阵的最大超节点规模较小，最大的也不超过 50,；而其他领域相关矩阵的超节点规模最小也有 63，最大可以达到 500，同时根据表 3-7 的测试结果，可以看出超节点算法相对于向上看算法的优势基本随最大超节点规模的上升而上升。一个大的超节点保证了矩阵在分解的过程中，超节点 Cholesky 分解中 3 级 BLAS 运算可以达到足够的提速效果，从而抵消前期的算法准备耗时。

对于 LM 方法求解病态潮流中的矩阵，由于其自身极度稀疏的特点，导致最大超节点的规模太小，BLAS 运算对缓存的利用效果比较低，反而导致超节点算法的计算效率不及向左看 Cholesky 分解算法。

（2）方法选取指标

对于超节点算法与向上看算法的取舍问题，文献[61]提出了一种取舍指标，该指标定义为

$$\alpha = flo / |\boldsymbol{L}| \tag{3-29}$$

其中，flo 代表浮点数运算（floating-point operation）的总数，而$|\boldsymbol{L}|$代表矩阵 $\boldsymbol{L}$ 的非零元个数。在符号分解的过程中，这两个数值会被统计出来。若该比值能够粗略代表在数值分解的过程中缓存的重复利用效果，若比值越高，则超节点 Cholesky 分解中 BLAS 操作的加速效果就越好。

再次对表 3-4 与表 3-6 中涉及的矩阵统计进行统计，结果如表 3-9。

表 3-9 超节点算法中取舍指标统计

Table 3-9 The ratio of floating-point operation and |$\boldsymbol{L}$| in supernodal Cholesky

矩阵/系统名称	浮点数运算总数	非零元个数	指标值
IEEE9 节点	664	88	7.545455
IEEE14 节点	1823	185	9.854054
IEEE30 节点	6833	553	12.35624
IEEE39 节点	7178	656	10.94207
IEEE57 节点	18981	1337	14.19671

续表 3-9 超节点算法中取舍指标统计

矩阵/系统名称	浮点数运算总数	非零元个数	指标值
IEEE300 节点	135867	7635	17.79528
华东 3647 系统	1287511	84075	15.31384
华东 4171 系统	1533125	97053	15.79678
华东 4769 系统	1608967	108809	14.78708
华东 2806 系统	732957	55395	13.23146
华东 5473 系统	1973331	129589	15.22761
华东 7872 系统	2360036	171306	13.77673
华东 8003 系统	2493653	175901	14.17646
华东 8241 系统	2603838	183214	14.212
华东 8386 系统	2633321	185463	14.19863
bcsstk16	186418497	812183	229.5277
bcsstk17	157345295	1043601	150.7715
ex3	2428708	58346	41.62596
ex13	3273772	81264	40.28564
ex15	11037724	227362	48.54692
Pres_Poisson	617372099	2507325	246.2274
plat1919	3035360	67262	45.12741
fv1	20460996	301652	67.8298
fv2	21893300	310926	70.41322

由表 3-9 可以看出，对于该指标，电力系统矩阵与其他矩阵的指标值差异性很大。对于电力系统矩阵，该比值最大不超过 20，而对于其他矩阵，最大则可达到近 250。

文献[61]中通过大量的矩阵测试，给出了该取舍指标阈值的经验数据，认为当该比值超过 40 时，超节点 Cholesky 算法能体现出较大的优势，其计算速度高于向上看 Cholesky 分解。否则，向上看 Cholesky 分解则具有更快的计算速度。由此，电力系统的矩阵其取舍指标远低于阈值，故而超节点 Cholesky 分解算法并不能在数值分解过程中展现出其高效性。

可以看出，不论是从最大超节点规模或是浮点数运算总数/非零元个数的取舍指标角度出发，相较于流体动力学等领域的矩阵，电力系统的矩阵由于其极度稀疏的特性，导致了整体的超节点规模不大，因此在超节点 Cholesky 分解的过程中，3 级 BLAS 操作不能很好地利用缓存进行运算，影响了加速的效果。

因此，对于 LM 方法求解电力系统病态潮流中出现的对称正定矩阵，采用向上看 Cholesky 分解算法进行数值分解能够获得更高的计算效率。相较之下，超节点 Cholesky 分解算法则不太适用于该问题的求解。本文选取向上看 Cholesky 分解算法作为高性能实现中数值分解算法的基础。

3.4 本章小结

本章从电力系统相关计算中矩阵稀疏的特性出发，对稀疏技术及算法进行了研究。首先，对稀疏技术的研究历史进行介绍，并分析了求解稀疏线性方程组的直接法与迭代法各自特点，选取直接法作为本章的主要研究对象。随后，对基础的稀疏存储技术以及稀疏三角方程组求解算法进行了介绍。

针对 LM 方法求解病态潮流中稀疏线性方程组求解的系数矩阵，由于矩阵的对称正定特性，需采用 Cholesky 分解算法进行相应计算。因此，本章对稀疏 Cholesky 分解算法中的符号分解算法以及两种主流的数值分解算法进行介绍。并通过实际系统数据的测试以及与其他领域矩阵的对比，得出了超节点 Cholesky 分解算法并不适用于电力系统病态潮流计算的结论，故选用向上看 Cholesky 分解算法作为后续高性能实现的数值分解算法基础。

第四章 病态潮流的高性能实现方案研究

4.1 引言

随着现代电力系统的建设，所研究的系统规模越来越大，对潮流计算的程序速度也提出了更高的要求。本章在第二、第三章的基础上，立足于潮流计算的计算效率，依据 LM 方法求解病态潮流中迭代步求解的计算特点，有针对性地研究具体的数值分解算法，完成病态潮流迭代步的稀疏高性能实现，并进一步完成整个病态潮流高性能实现方案的设计。在此基础上，本章对病态潮流计算平台的开发进行论述，完成了方案的程序实现，从理论与程序两方面对病态潮流计算的高性能实现进行完整的论述。

4.2 病态潮流迭代步的稀疏高性能实现

4.2.1 显式 Cholesky 分解

本文第三章中对正定矩阵的 Cholesky 分解算法进行了研究，并考虑到电力系统相关矩阵的特性，选取了向上看 Cholesky 分解作为数值分解的算法。

在实际的 LM 方法求解过程中，每步的迭代步计算公式为

$$(\boldsymbol{J}(x_k)^T \boldsymbol{J}(x_k) + \mu_k \boldsymbol{I})\Delta \boldsymbol{x}_k = -\boldsymbol{J}(x_k)^T \boldsymbol{F}(x_k) \tag{4-1}$$

其中矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 为正定矩阵，令 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}=\boldsymbol{B}$，则对其进行 Cholesky 分解可以表示为：

$$\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}} = \boldsymbol{B} = \boldsymbol{J}^{\mathrm{T}}\boldsymbol{J} + \mu\boldsymbol{I} \tag{4-2}$$

其中，矩阵矩阵 $\boldsymbol{L}$ 为下三角矩阵。

根据 3.3.1 节的论述，对于式(4-2)进行 Cholesky 分解，包括符号分解与数值分解两步。其中，符号分解是在 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 稀疏结构的基础上形成矩阵分解结果 $\boldsymbol{L}$ 的稀疏结构，从而为后续数值分解步骤做准备。由于 $\mu\boldsymbol{I}$ 项的存在并不影响矩阵的非零结构，对 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 的符号分解即等同于对 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ 阵的符号分解。数值分解则在符号分解的基础上，根据 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 阵的具体数值进一步求解出对应 $\boldsymbol{L}$ 阵的数值结果。

在 LM 方法的每步迭代中，雅克比矩阵 $\boldsymbol{J}$ 及阻尼因子 $\boldsymbol{\mu}$ 均已知。而 Cholesky

分解对象 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 需要通过矩阵的乘法与加法才可获得。一般在使用主流稀疏解算库或科学计算软件对算法进行程序实现时，会采用显式 Cholesky 分解方法进行符号分解和数值分解。

显式符号分解：由于符号分解仅涉及矩阵的稀疏结构，一般采用显式 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ 阵的符号分解会先通过模拟矩阵相乘显式求出 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ 阵的结构，再根据本文 3.3.2 节所述对称正定矩阵的稀疏符号分解算法生成 $\boldsymbol{L}$ 的稀疏结构。文献[22]在研究电力系统状态估计中遇到了类似矩阵 $\boldsymbol{G}=\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ 的符号分解问题，该文章在处理该问题时采用节点邻接矩阵的方法，从而避免模拟矩阵相乘导致的效率低下，但是仍然没有完全回避显式生成 $\boldsymbol{G}=\boldsymbol{H}^{\mathrm{T}}\boldsymbol{H}$ 稀疏结构这一步骤。

显式数值分解：一般进行显式 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 阵数值分解时，需要先通过矩阵乘法求出 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$，再与 $\mu\boldsymbol{I}$ 矩阵相加，从而求出 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 全矩阵的具体数值。随后，再采用数值分解算法，进一步求出 $\boldsymbol{L}$ 阵的具体数值结果。在该过程中，显式生成 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 全矩阵的计算步骤会产生一定的冗余计算量，并因此增加数值分解步骤的计算时间。

4.2.2 隐式 Cholesky 分解

由于显式 Cholesky 分解中会产生冗余计算量，本文将会采用 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵隐式 Cholesky 分解方法，从而减少传统方法中产生的冗余计算量。隐式 Cholesky 分解方法的主要特点如下：

隐式符号分解：在不通过模拟矩阵相乘求取矩阵 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ 稀疏结构的前提下，采用其他方法隐式生成分解结果 $\boldsymbol{L}$ 阵的稀疏非零元结构，从而用于数值分解步骤时的稀疏存储。

隐式数值分解：根据 $\boldsymbol{B}$ 阵的结构特点以及 Cholesky 分解的算法特点，只针对数值分解过程中必要应用到的数值进行求解，并进一步结合 Cholesky 分解的算法流程，在不显式进行矩阵相加的前提下，完成对 $\boldsymbol{L}$ 阵中非零元的数值进行求取。

（1）$\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ 矩阵的隐式符号分解

对于矩阵 $\boldsymbol{J}^{T}\boldsymbol{J}$，可以构造一个规模不大于原矩阵 $\boldsymbol{J}$，但 Cholesky 分解后非零结构与 $\boldsymbol{J}^{T}\boldsymbol{J}$ 阵完全一致的星矩阵 $\boldsymbol{J}^{*}$[76]。根据星矩阵 $\boldsymbol{J}^{*}$的符号分解结果，可在不显式生成 $\boldsymbol{J}^{T}\boldsymbol{J}$ 阵稀疏结构的前提下，形成 $\boldsymbol{L}$ 阵的稀疏结构。

若将矩阵 $\boldsymbol{J}$ 按列表示为$[\boldsymbol{J}_{1*}\ \boldsymbol{J}_{2*}\ \ldots\boldsymbol{J}_{i*}\ \ldots\boldsymbol{J}_{n*}]^{T}$，其中，$\boldsymbol{J}_{i*}$表示 $\boldsymbol{J}$ 阵第 i 行的行向量，则

$$\boldsymbol{J}^T\boldsymbol{J}=\begin{bmatrix}\boldsymbol{J}_{1*}{}^T & \boldsymbol{J}_{2*}{}^T & \cdots \boldsymbol{J}_{n*}{}^T\end{bmatrix}\begin{bmatrix}\boldsymbol{J}_{1*}\\ \boldsymbol{J}_{2*}\\ \vdots\\ \boldsymbol{J}_{n*}\end{bmatrix}=\sum_{i=1}^{n}\boldsymbol{J}_{i*}{}^T\boldsymbol{J}_{i*} \tag{4-3}$$

其中$\sum_{i=1}^{n}\boldsymbol{J}_{i*}{}^T\boldsymbol{J}_{i*}$中的每一项，即代表矩阵$\boldsymbol{J}^T$的第 i 列与矩阵 $\boldsymbol{J}$ 的第 i 行相乘所得结果。

令 $Struct(\boldsymbol{J}_{i*})=\{j|\ J_{ij}\neq 0\}$，表示 $\boldsymbol{J}$ 阵第 i 行非零结构，则 $\boldsymbol{J}^T$阵的 i 列与 $\boldsymbol{J}$ 阵的 i 行相乘所得的结果就是一个结构为 $Struct(\boldsymbol{J}_{i*})\times Struct(\boldsymbol{J}_{i*})$的稠密块，对应式(4-3)中$\sum_{i=1}^{n}\boldsymbol{J}_{i*}{}^T\boldsymbol{J}_{i*}$的一项，图 4-1 以一个 6 阶矩阵作为示例，其中，符号“•”代表非零元。

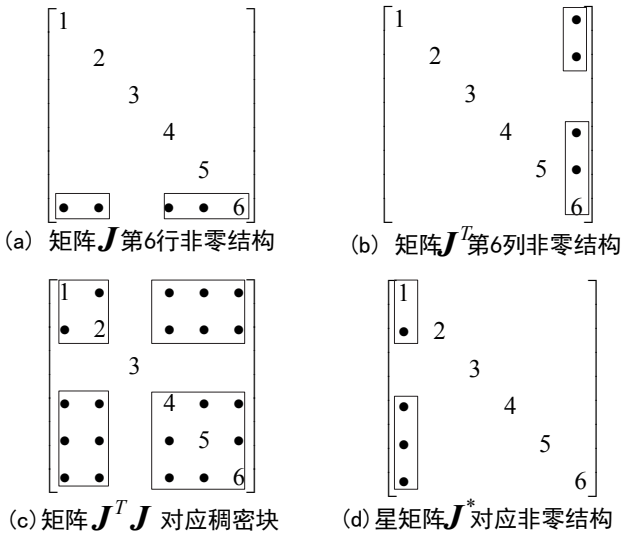


图 4-1 矩阵 $\boldsymbol{J}^T\boldsymbol{J}$ 及对应星矩阵非零结构示例图

Fig. 4-1 Nonzero pattern of matrix $\boldsymbol{J}^T\boldsymbol{J}$ and its corresponding star matrix

由上图，矩阵 $\boldsymbol{J}^T$ 的第 6 列与矩阵 $\boldsymbol{J}$ 的第 6 行相乘得到的是一个如图 4-1(c)的稠密块，然而实际上，稠密块中仅部分非零元对 Cholesky 分解后的非零结构有影响。令 $k=\min Struct(\boldsymbol{J}_{i*})$，即 k 为 $\boldsymbol{J}$ 阵第 i 行最小的非零列号，对于 $Struct(\boldsymbol{J}_{i*})\times Struct(\boldsymbol{J}_{i*})$稠密块中的元素$(\boldsymbol{J}^T\boldsymbol{J})_{ab}$，其中，$a,b\in Struct(\boldsymbol{J}_{i*})$，若 $a>k$ 且 $b>k$，则该元素对 Cholesky 分解结果的非零结构无影响，并有：

$$(a\le k)\vee(b\le k)\Rightarrow \boldsymbol{J}^*\text{的}a\text{行}b\text{列元素非零}$$

因而，星矩阵 $\boldsymbol{J}^*$的非零结构可表示为

$$Struct(\boldsymbol{J}^*_{*k}) = \bigcup\{Struct(\boldsymbol{J}^T_{*i}) \mid \boldsymbol{J}^T_{*i}\text{中非零元的最小行号为}k\} \tag{4-4}$$

其中，$Struct(\boldsymbol{J}^*_{*k})$和 $Struct(\boldsymbol{J}^T_{*k})$分别表示 $\boldsymbol{J}^*$和 $\boldsymbol{J}^T$ 第 k 列非零结构，而 $\boldsymbol{J}^T_{*i}$ 为 $\boldsymbol{J}^T$ 的第 i 列。

根据式(4-4)，星矩阵 $\boldsymbol{J}^*$的列非零结构是根据矩阵 $\boldsymbol{J}^T$ 的列非零结构生成的，若 $\boldsymbol{J}^T$ 第 i 列的第一个非零元行号为 k，即 $\boldsymbol{J}^T_{ki}$ 为非零元，则星矩阵 $\boldsymbol{J}^*$的第 k 列非零元结构与 $\boldsymbol{J}^T$ 第 i 列的非零结构一致，保证了星矩阵 $\boldsymbol{J}^*$的非零元总个数最多与原始矩阵的非零元个数相同。且易看出，由于式(4-4)中存在求并的过程可能存在相同非零元的重叠，导致星矩阵 $\boldsymbol{J}^*$的非零元个数一般会比原始矩阵更少。

可以证明，星矩阵 $\boldsymbol{J}^*$Cholesky 分解后的非零结构与矩阵 $\boldsymbol{J}^T\boldsymbol{J}$ 是一致的。

首先选取式式(4-3)中的一项，令所选的 $\boldsymbol{J}$ 阵行号 $\boldsymbol{J}^T$ 阵列号为 i，并将矩阵 $\boldsymbol{J}$ 第 i 行的非零结构 $Struct(\boldsymbol{J}_{i*})$记为集合 S，即有

$$S = \{j \mid \boldsymbol{J}_{ij} \neq 0\} \tag{4-5}$$

则根据之前描述，$\boldsymbol{J}_{i*}{}^T\boldsymbol{J}_{i*}$相乘所得的结果就是一个结构为 $S\times S$ 的稠密块。设 k 为 $\min S$，则根据式(4-4)，星矩阵 $\boldsymbol{J}^*$与该项对应，在 k 列上具有与 $\boldsymbol{J}^T$ 矩阵第 i 列相同的稀疏结构，即 $Struct(\boldsymbol{J}^*_{*k})= S$。

对于星矩阵 $\boldsymbol{J}^*$第 k 列的非零结构 $Struct(\boldsymbol{J}^*_{*k})$，设 $a,b\in S$，且有 $b>a>k$。则有 $\boldsymbol{J}^*_{kk}$、$\boldsymbol{J}^*_{ak}$、$\boldsymbol{J}^*_{bk}$ 均为非零元，且 $k< a<b$。进一步将星矩阵 Cholesky 分解后的下三角矩阵记为 $\boldsymbol{L}^*$，则根据定理 3.1，有

$$\boldsymbol{J}^*_{kk}, \boldsymbol{J}^*_{ak}, \boldsymbol{J}^*_{bk} \neq 0 \Rightarrow \boldsymbol{L}^*_{kk}, \boldsymbol{L}^*_{ak}, \boldsymbol{L}^*_{bk} \neq 0 \tag{4-6}$$

即星矩阵非零元位置对应的 $\boldsymbol{L}^*$矩阵元素也为非零元。进一步，根据定理 3.2，可以得出

$$k < a < b \wedge \boldsymbol{L}^*_{ak} \neq 0 \wedge \boldsymbol{L}^*_{bk} \neq 0 \Rightarrow \boldsymbol{L}^*_{ba} \tag{4-7}$$

即星矩阵在 Cholesky 分解的过程中，会在 $\boldsymbol{L}^*_{ba}$ 位置产生注入元。

故对于任意的 $a>k$，在 Cholesky 分解过程中，会在 a 列生成非零元，该列的非零结构为 $\{\forall b \mid b \in S \wedge b > a\}$。由此可以得出经过 Cholesky 分解后，$\boldsymbol{L}^*$的整体非零结构为

$$\boldsymbol{L}^*_{ab} \neq 0 \ \ (\forall a, b \in S \wedge b \geq a) \tag{4-8}$$

该结构就是 $S\times S$ 所得稠密块的下三角部分。由于 Cholesky 分解的符号分解过程仅考虑矩阵的下三角部分，星矩阵分解后的 $\boldsymbol{L}^*$非零结构与 $\boldsymbol{J}^T\boldsymbol{J}$ 完全一致。考虑到 Cholesky 符号分解过程只应用矩阵的稀疏结构，可以使用星矩阵 $\boldsymbol{J}^*$替代原

始的 $\boldsymbol{J}^T\boldsymbol{J}$ 进行符号分解。

由此，根据式(4-4)，对原矩阵的每行进行遍历，即可构造出的星矩阵 $\boldsymbol{J}^*$非零元规模不大于原矩阵 $\boldsymbol{J}$，并具有和 $\boldsymbol{J}^T\boldsymbol{J}$ 相同的 Cholesky 分解 $\boldsymbol{L}$ 阵结构[76]。对 $\boldsymbol{J}^*$ 进行稀疏 Cholesky 符号分解，就能够在不求解矩阵 $\boldsymbol{J}^T\boldsymbol{J}$ 非零结构的前提下，隐式完成对 $\boldsymbol{J}^T\boldsymbol{J}$ 的符号分解。

相较于显式生成 $\boldsymbol{J}^T\boldsymbol{J}$ 阵的稀疏结构，使用隐式实现符号分解的方法由于 $\boldsymbol{J}^*$阵的非零元规模更小且避免了矩阵乘法计算，计算中的时间、空间开销都低于显式方法。

（2）$\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵的隐式数值分解

对于式(4-2)中 $\boldsymbol{L}$ 阵的数值求解，采用 3.3.3 节所述向上看 Cholesky 分解算法，每步迭代求解 $\boldsymbol{L}$ 的一行。假设 $\boldsymbol{L}$ 前 k-1 行已求得，第 k 行元素待求，将式(4-2)的前 k 行 k 列改写成 2×2 块状形式

$$\begin{bmatrix} \boldsymbol{L}_{11} & \\ \boldsymbol{l}_{12}^T & l_{22} \end{bmatrix}\begin{bmatrix} \boldsymbol{L}_{11}^T & \boldsymbol{l}_{12} \\ & l_{22} \end{bmatrix}=\begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{b}_{12} \\ \boldsymbol{b}_{12}^T & b_{22} \end{bmatrix} \tag{4-9}$$

其中，$\boldsymbol{L}_{11}$ 和 $\boldsymbol{B}_{11}$ 分别为 $\boldsymbol{L}$ 和 $\boldsymbol{B}$ 前(k-1×k-1)阶子阵，$\boldsymbol{l}_{12}$ 和 $\boldsymbol{b}_{12}$ 分别为 $\boldsymbol{L}^T$ 和 $\boldsymbol{B}$ 第 k 列第 1 行到第 k–1 行元素组成的列向量，l_{22} 和 b_{22} 分别为 $\boldsymbol{L}$ 和 $\boldsymbol{B}$ 第 k 行第 k 列的元素。

由式(4-9)易得

$$\begin{cases} \boldsymbol{L}_{11}\boldsymbol{l}_{12}=\boldsymbol{b}_{12} & ① \\ l_{22}=\sqrt{b_{22}-\boldsymbol{l}_{12}^T\boldsymbol{l}_{12}} & ② \end{cases} \tag{4-10}$$

其中通过式(4-10)-①可通过一次前代运算解得 $\boldsymbol{l}_{12}$，通过式(4-10)-②可求得 l_{22}，从而完成矩阵 $\boldsymbol{L}$ 第 k 行元素的求解。

由式(4-10)，在 $\boldsymbol{L}$ 阵第 k 步数值分解时，需要用到 $\boldsymbol{b}_{12}$ 及 b_{22} 的数值，即 $\boldsymbol{B}$ 阵第 k 列的前 k 行数值。同时，考虑到 $\boldsymbol{B}=\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 为对称矩阵，故仅 $\boldsymbol{B}$ 阵的上三角部分是数值分解中必要的，显式方法中对 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 的全矩阵求取会产生接近一半的冗余计算量。

首先忽略 $\mu\boldsymbol{I}$ 项，即令 $\boldsymbol{B}=\boldsymbol{J}^T\boldsymbol{J}$，则矩阵乘法可按列表示为

$$\boldsymbol{B}_{*k}=[\boldsymbol{J}^T{}_{*1}\ldots\boldsymbol{J}^T{}_{*n}]\begin{bmatrix} J_{1k} \\ \vdots \\ J_{nk} \end{bmatrix}=\sum_{i=1}^{n}J_{ik}\cdot\boldsymbol{J}^T{}_{*k} \tag{4-11}$$

其中，$\boldsymbol{B}_{*k}$ 为 $\boldsymbol{B}$ 阵第 k 列，$\boldsymbol{J}_{ik}$ 为 $\boldsymbol{J}$ 阵的 i 行 k 列元素。

由于数值分解中仅需要 $\boldsymbol{B}$ 阵上三角元素的数值，故在式(4-11)求解中，只需要对 $\boldsymbol{B}$ 阵的前 k 行进行求解，从而避免求取 $\boldsymbol{J}^T\boldsymbol{J}$ 全矩阵时产生的冗余计算量。此外，这里求得的 $\boldsymbol{B}_{*k}$ 相当于式(4-10)-①的右端项，由于符号分解过程中已求出 $\boldsymbol{L}$ 阵的稀疏结构，采用稠密一维数组的形式存储右端项既不会影响式(4-10)-①快速前代的稀疏实现[23]，同时也能避免了采用稀疏矩阵形式存储时所需要考虑的非零结构预测问题及其产生的额外计算量[77]，并进一步提升了快速前代中右端项的索引速度。

式(4-11)前 k 行元素的求取可用以下伪代码实现：

```
function getRHS(k)
JT = tranpose(J)
for each i 满足J(i,k)为非零元 do
    for each j 满足JT(j,i)为非零元 do
        if j ≤ k
            b(j) = b(j) + J(i,k) × JT(j,i)
        end if
    end for
end for
```

其中，数组 b 存储式(4-10)中计算所需的 $\boldsymbol{b}_{12}$ 及 b_{22}。

在此基础上再考虑 $\mu\boldsymbol{I}$ 项的影响，其作用是在矩阵 $\boldsymbol{J}^T\boldsymbol{J}$ 的对角元加上数值 μ。根据式(4-10)的 Cholesky 分解公式，只有式(4-10)-②的计算中用到了所分解矩阵的对角元 b_{22} 数值，故对其进行以下修改，即可实现对矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 的 Cholesky 分解：

$$\begin{cases} \boldsymbol{L}_{11}\boldsymbol{l}_{12} = \boldsymbol{b}_{12} & ① \\ l_{22} = \sqrt{b_{22} + \mu - \boldsymbol{l}_{12}^T\boldsymbol{l}_{12}} & ② \end{cases} \tag{4-12}$$

其中，$\boldsymbol{b}_{12}$ 和 b_{22} 仍表示 $\boldsymbol{J}^T\boldsymbol{J}$ 阵中的元素数值。由式(4-12)-②，即可避免显式进行 $\boldsymbol{J}^T\boldsymbol{J}$ 与 $\mu\boldsymbol{I}$ 的矩阵加法，从而消除稀疏矩阵相加时寻找对角元所需的额外定位索引时间。

结合式(4-11)与式(4-12)，整个 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵的隐式数值分解过程可以在不求解 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 全矩阵数值的前提下，计算出矩阵的 Cholesky 数值分解结果，避免显式方法中的冗余计算量。

对于 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵的隐式 Cholesky 数值分解可以采用以下伪代码进行实现：

for $k=1$ to n **do**
  $b(1:k)=\text{getRHS}(k)$
  $x(1:k)=b(1:k)$
  for each $j \in Struct(\boldsymbol{L}_{k*})$ **do**
    $x(j)=x(j)/L(j,j)$
    $x(k)=x(k)-x(j)*x(j)$
    for each $i>j$ 满足$L(i,j)$为非零元 **do**
      $x(i)=x(i)-L(i,j)*x(j)$
    end for
  end for
  $x(k)=\sqrt{x(k)+\mu}$
  for each $j \in Struct(\boldsymbol{L}_{k*})$ **do**
    $L(k,j)=x(k)$
  end for
end for

（3）$\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵的隐式 Cholesky 分解流程图

综合符号分解和数值分解两部分，本文所述 $\boldsymbol{J}^{T}\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵隐式 Cholesky 分解方法的整体计算流程如图 4-2。

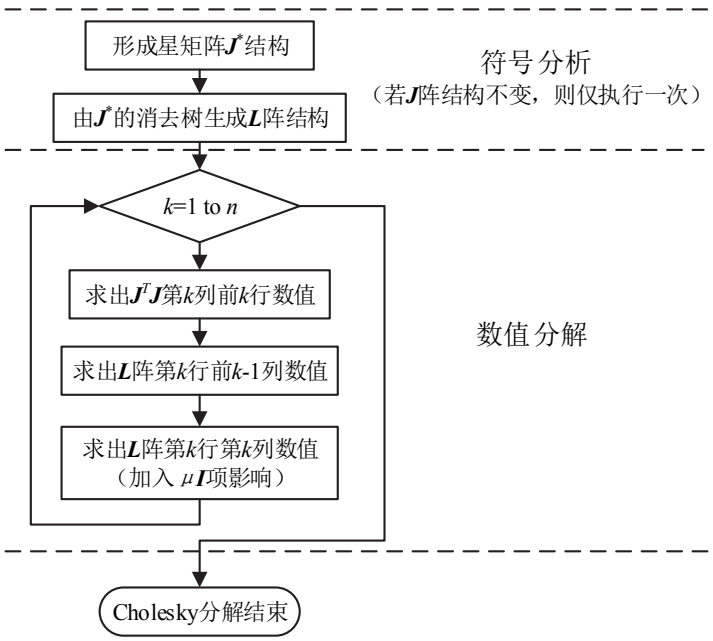


图 4-2 $\boldsymbol{J}^{T}\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵隐式 Cholesky 分解流程图
Fig. 4-2 Flow chart of implicit Cholesky factorization for $J^{T}J+\mu I$

4.3 病态潮流整体高性能实现方案

4.3.1 关键环节的高效实现

根据本文第二章、第三章的内容，结合 4.2 节对隐式 Cholesky 分解的研究，可以对病态潮流计算高性能实现的方案进行整体的设计。方案包括病态潮流算法、迭代步求解方案、稀疏算法选取以及具体的数值分解算法。这几个环节在前文中已经分别进行了叙述，下面再对这些内容进行总结，对方案中的各个环节进行选取。

（1）病态潮流算法

根据本文 2.2 节的论述，病态潮流算法的备选方案有最优乘子法、张量法、优化算法、LM 方法等。通过对这四个主流方法的比较，发现 LM 方法相较于其他方法有计算量不大且收敛性好的特点。结合自适应阻尼因子的引入，可以进一步提升算法的收敛速度，解决常规牛顿类算法计算系统潮流时的不收敛问题。

因此，对于病态潮流算法，本文选取 LM 方法进行求解，并选用自适应阻尼因子提升收敛速度。

（2）迭代步求解方案

根据本文 2.3 节的论述，在 LM 方法中迭代步的求解有两个方案：直接求解方案和扩展矩阵方案。前者思路比较直接，后者则通过形式的变换，使矩阵与系统导纳阵同构。通过对两个方案实际进行测试，发现扩展矩阵方案在计算的过程中会引入大量的非零元，很大程度上影响计算效率。

因此，对于 LM 方法中的迭代步求解，本文选取直接求解方案。

（3）稀疏算法

根据本文 3.3 节的论述，直接方案求解迭代步中，需要对线性方程组进行求解。考虑到电力系统的稀疏特性，采用稀疏技术进行处理。在数据存储方面，采用 CSC，即按列压缩的形式存储稀疏矩阵。而在具体的方程组求解算法上，由于系数矩阵的对称正定特性，采用稀疏 Cholesky 分解算法，包括符号分解和数值分解两步骤。数值分解算法包括向上看 Cholesky 分解与超节点 Cholesky 分解两种主流的算法。通过实际的算例测试以及分析，发现超节点算法虽然引入了 3 级 BLAS 进行算法加速，然而电力系统的矩阵特性导致计算过程中的超节点规模过小，BLAS 操作利用缓存进行加速的效果较差，计算效果不及向上看 Cholesky 分解。

因此，对于稀疏技术部分，本文选取 CSC 格式存储矩阵，采用 AMD 算法

进行节点预排序，减少注入元，并对正定矩阵进行 Cholesky 分解，同时选取向上看的数值分解算法。

（4）对矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 的分解算法

根据本文 4.2 节论述，在实际求解迭代步时，需要对矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 进行 Cholesky 分解，其中 $\boldsymbol{J}$ 与 μ 已知。若采用显式 Cholesky 分解算法，先求出 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 的全矩阵再进行矩阵分解，存在一定的冗余计算量。因此，本文在向上看 Cholesky 分解算法的基础上，进一步提出隐式 Cholesky 符号分解与数值分解，在不求取全矩阵的前提下，完成矩阵 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 的 Cholesky 分解过程。相较于显式算法，隐式算法能够避免冗余计算量，进一步提升计算效率。

因此，对于具体的 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵分解算法，采用隐式 Cholesky 分解算法。

4.3.2 整体方案流程

综合病态潮流高性能实现方案中的各个环节，即采用 LM 方法计算电力系统潮流，在迭代步求解中采用直接求解的方式，针对电力系统的稀疏特性应用 CSC 格式存储以及 AMD 预排序算法，在对 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵分解时采用隐式 Cholesky 分解算法。由此，可得整个方案的流程图如图 4-3。

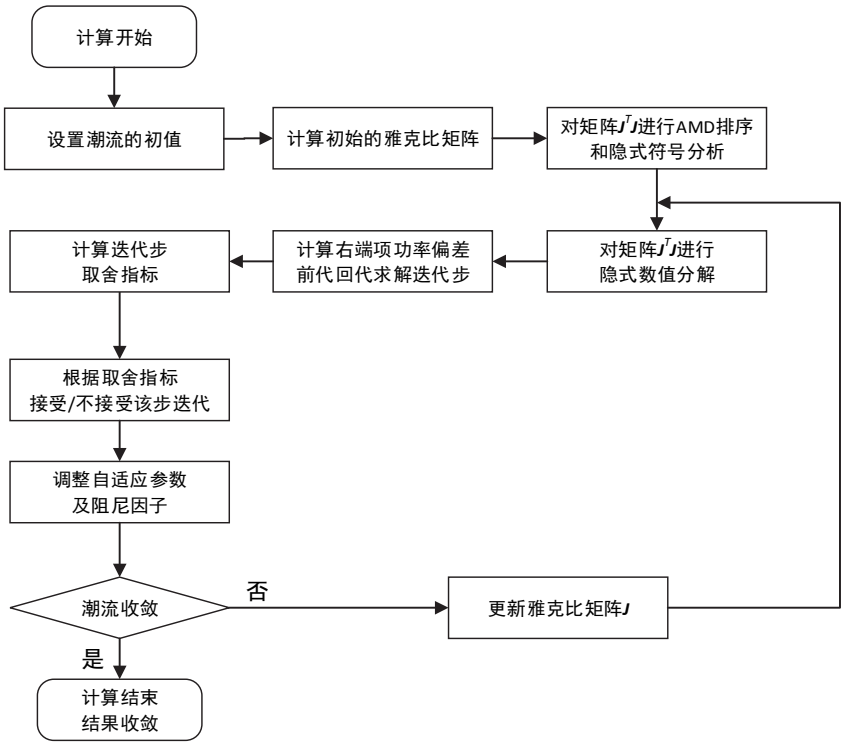


图 4-3 病态潮流计算高性能实现方案流程图

Fig. 4-3 Flow chart of the ill-conditioned power flow calculation with high performance

由图 4-3，可以看出病态潮流计算的高性能实现方案整体基于 LM 方法的流程，主要的高性能实现部分体现在迭代步求解的 Cholesky 隐式分解上。该方案通过自适应 LM 方法提升潮流计算收敛性，并利用稀疏技术和隐式 Cholesky 分解算法提高计算速度。

4.4 病态潮流计算平台实现

4.4.1 开发目的及软件平台特点

为了对电力系统病态潮流计算问题进行正确并且高速地计算，同时也是为了验证本文所述病态潮流计算高性能实现方案的有效性，需要在所研究的理论算法基础上，开发一套行之有效的病态潮流计算平台，从而为 4.3 节所述的方案提供程序化实现。

基于本文第二章的 LM 方法理论，以及第三章的稀疏技术和 3.2 节的数值算法，采用混合的程序开发模式，所使用语言包括标准 C、面向对象的 C++以及 C#，充分利用了几种语言和相应程序库各自的有点，使用编译环境为 Visual Studio2010。其中 C 语言负责完成底层的数值计算功能模块，C++语言主要负责 LM 方法计算潮流的主体流程内核模块，而 C#则提供了一个易于调用的可视化界面。各模块之间通过相互调用组成病态潮流计算平台，为病态潮流求解提供了可视化平台。

在程序设计上，电力系统病态潮流计算平台包括以下特点：

1）数据通用性：考虑到电力系统潮流计算所需要的数据量较大，若采用自定义的数据格式，通常会出现数据转换方面的难题。为了避免数据格式对用户造成的使用不便，本计算平台采用通用的 PSD-BPA 潮流数据格式，使得用户无需通过格式转换，直接使用本软件对原有的潮流数据进行计算。而在输出结果方面，本程序也与 BPA 的潮流输出结果保持格式上的一致，从而保证了结果的易读性。

2）计算高效性：本计算平台的主体方法依托 4.3 节所述的病态潮流计算高性能实现方案，故保证计算的正确高效是本软件最重要的目标。在开发软件时，通过将软件的模块化划分，分别采用不同语言进行开发。在最为耗时的数值计算部分，采用底层 C 语言，保证程序的高效性，并结合隐式 Cholesky 分解算法，提升计算效率。而在 LM 方法流程方面，结合电力系统的建模特性，采用面向对象的 C++语言，通过类的设计完成建模。

3）操作易用性：为了进一步使本计算平台更为简洁易用，采用 C#语言开发了用户界面，通过调用计算内核完成潮流计算工作。对于用户而言，只需要在界面上通过简单的操作，对算法进行选择、配置，并选择所需的潮流原始数据，即可进行潮流计算工作，而不需要对命令行等操作有任何了解。

4.4.2 软件功能

本潮流计算平台的主要功能即为正确、快速地实现电力系统潮流计算，在实际的软件开发过程中，计算平台的功能不只包含了本文 4.3 节所述的病态潮流计算方案，也包括常规的良态潮流计算功能。

1）良态潮流计算功能：考虑到传统的牛顿-拉夫逊法以及 PQ 分解法在传统的潮流计算中占有非常重要的地位，其计算效率以及结果的正确性为大家所公认，本潮流计算平台保留了传统的 NR 法即 PQ 分解法计算模块，用户通过选择算法，可以采用这两种算法进行常规的良态潮流计算。

2）病态潮流计算功能：在保证传统良态潮流算法的同时，本计算平台也融入了本文所述的病态潮流计算方案，通过自适应 LM 方法完成病态潮流计算。并基于隐式 Cholesky 算法，在数值计算的程序部分完成相应的代码编写，提升数值分解计算效率，加快程序执行速度。

对于用户而言，可以通过用户界面选择不同的计算功能，且对于每一种计算功能，都可以进行相应的参数设置，从而使程序达到预期的计算效果。

整个潮流计算平台的流程与功能见图 4-4。

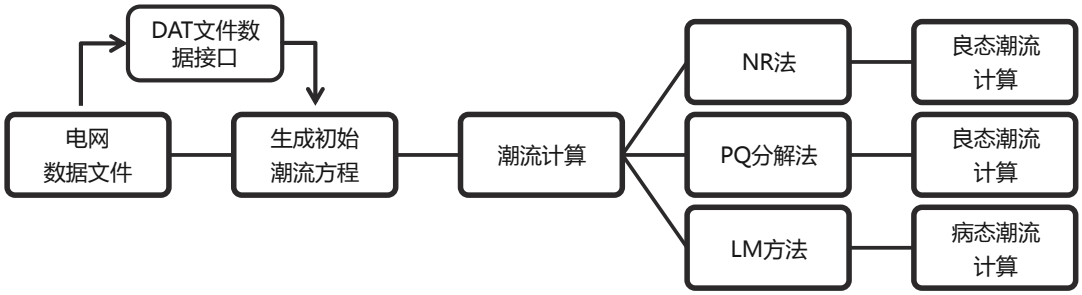


图 4-4 病态潮流计算平台功能流程图

Fig. 4-4 Flow chart of functionalities of the ill-conditioned power flow calculation platform

4.4.3 程序架构

本软件平台采用模块化方式进行程序开发，整个计算平台包括：可对 BPA 数据文件进行无缝读写的 BPA 数据接口模块、可完成潮流计算中相关稀疏矩阵

计算功能的稀疏矩阵计算模块、对潮流系统进行初始化建模的系统建模模块、用于潮流主体流程运算的潮流计算模块、对最终的潮流结果进行输出的潮流结果输出模块，整个软件平台的架构如图 4-5。

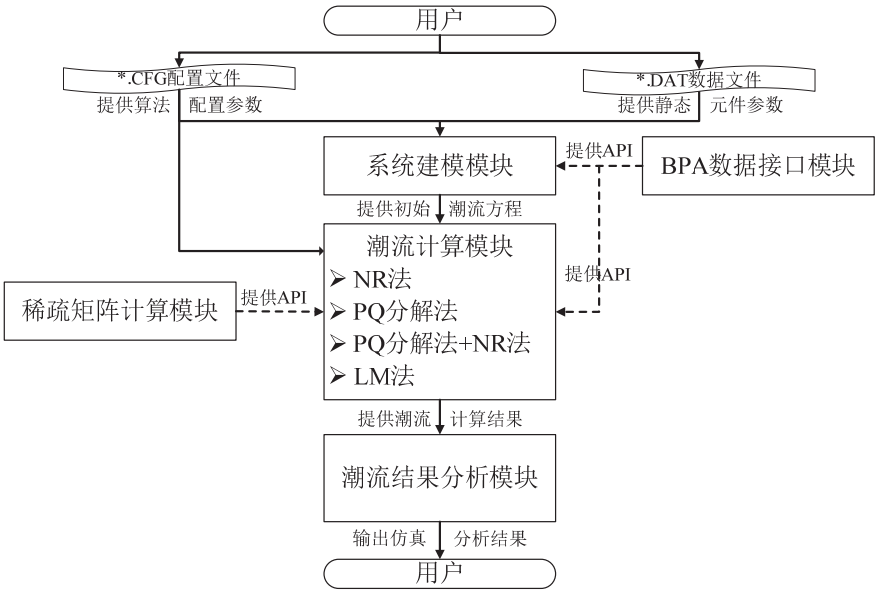


图 4-5 病态潮流计算平台程序架构

Fig. 4-5 Architecture of the ill-conditioned power flow calculation platform

图 4-5 中，各模块的功能如下：

1）BPA 数据接口模块：考虑到国内大部分电力相关企业都会采用 BPA 格式存放数据，为此采用 C++开发了该模块，以 BPA 的数据格式读取电网的潮流数据，并以动态链接库的形式提供 API 供潮流计算模块以及系统建模模块读取数据；

2）稀疏矩阵计算模块：由于潮流计算中核心的数值计算离不开稀疏技术，该模块采用 C 语言开发，实现程序中与稀疏矩阵有关的存储以及运算，包括基本的矩阵转置、矩阵加法、矩阵乘法、矩阵-向量乘法等，以及稀疏 LU 分解、稀疏隐式 Cholesky、稀疏前代回代等。并以动态链接库的形式为潮流计算模块提供 API；

3）系统建模模块：该模块根据 BPA 接口模块所得的电网原始数据，将各元件参数读入系统中，建立包括节点、支路、变压器等元件模型，进一步形成初始的潮流方程；

4）潮流计算模块：该模块为主体的计算模块，用于控制潮流算法的计算流程，利用 BPA 数据接口模块和稀疏矩阵计算模块来实现电力系统潮流计算。其主要功能包括自适应 LM 算法、牛顿-拉夫逊算法、PQ 分解法等。用户可通过选

取相应算法，调用模块中对应的计算功能；

5）潮流结果输出模块：该模块以潮流计算模块所得的计算结果为基础，通过 BPA 数据接口模块生成对应的输出结果，并将计算结果、收敛结果等输出至文本文件与显示器中，使用户直观地了解程序的运行结果。

而在主体的系统建模模块与潮流计算模块，程序采用面向对象的 C++语言进行开发，通过类的定义来完成电力系统各元件的建模以及潮流程序的流程控制。这两个模块类的定义以及与其他模块间的交互见图 4-6。

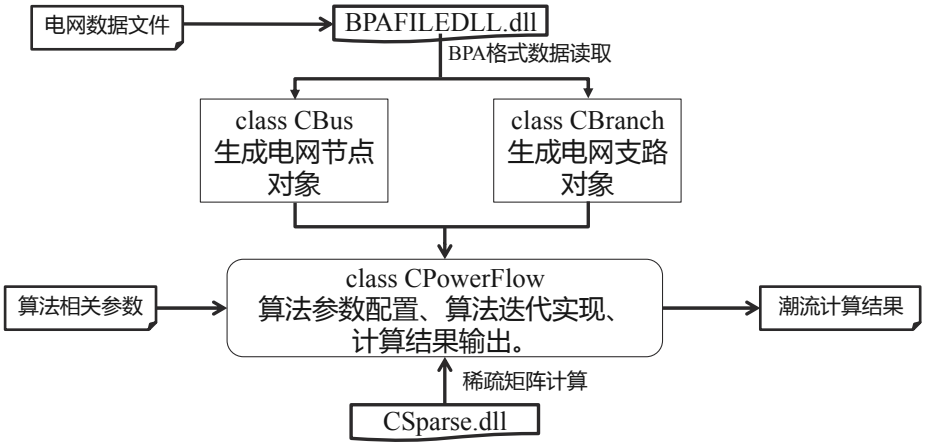


图 4-6 病态潮流计算平台中的类定义

Fig. 4-6 Class definition of the ill-conditioned power flow calculation platform

4.4.4 软件界面及使用

大规模电力系统潮流计算软件可通过双击可执行程序“PowerFlow.exe”进入软件界面，见图 4-7。

图 4-7 软件界面-算法及参数选择

Fig. 4-7 Software UI- choosing algorithm and parameters

图 4-7 中，可点击潮流算法选择中的“NR 法”、“PQ 分解法”以及“LM 法”选择合适的算法。在下方是算法的参数设置，包括极坐标/直角坐标选择（PQ 分解法是 BX 型/XB 型选择）、收敛精度最大迭代次数等，对于 LM 方法，还有一些算法自身的参数。点击“恢复默认参数”按钮，可以将参数改动重置回默认状态。点击文件上方的“打开文件”按钮，即可出现对话框，选择潮流原始数据，如图 4-8。

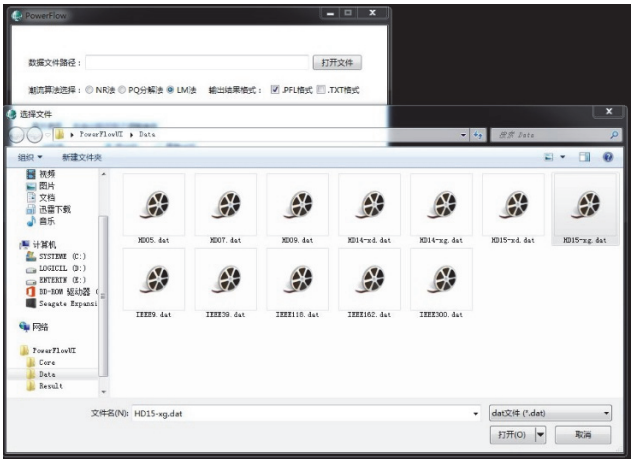


图 4-8 软件界面-潮流数据选取

Fig. 4-8 Software UI- choosing power flow data

图 4-8 中，可以选择相应的潮流原始数据进行计算。需要注意的是，本软件支持的潮流数据格式为 BPA 的文件格式。若所选的数据不符合格式要求，在潮流计算运行的过程中，会对该错误进行提示。选择好潮流数据后，点击“打开”按钮，即可回到图 4-7 的界面。在确认选择好算法及参数后，点击“开始计算”，程序就会自行调用潮流计算的内核程序，执行潮流计算的相应流程。计算内核程序的执行情况如图 4-9。

图 4-9 软件界面-计算内核程序

Fig. 4-9 Software UI- calculation core program

图 4-9 即为调用计算内核时出现的窗口。在具体的迭代过程中，该窗口会显示相应的数据，如计算时间，功率偏差等。由于该内核采用 C++开发，显示的界面为命令行窗口。对于不在意程序执行细节的用户，可以忽略该窗口。计算结束后，窗口会自行退出。计算完成后，原软件界面会跳出提醒，如图 4-10。

图 4-10 软件界面-计算结束

Fig. 4-10 Software UI- end of calculation

图 4-10 中，出现提示，向用户告知程序执行的结果。在上图中，结果为“收敛至精确解”。对于 NR 法及 PQ 分解法，可能出现在限定的迭代次数上限内，潮流不收敛的情况，此时提示会显示“潮流未收敛至要求”。对于 LM 方法，可能出现潮流无精确解的情况，此时，LM 方法会计算至潮流最小二乘解，程序提示“收敛至潮流最小二乘解”。计算完成后，可在 Result 文件夹打开相应的计算结果文件。如图 4-11。

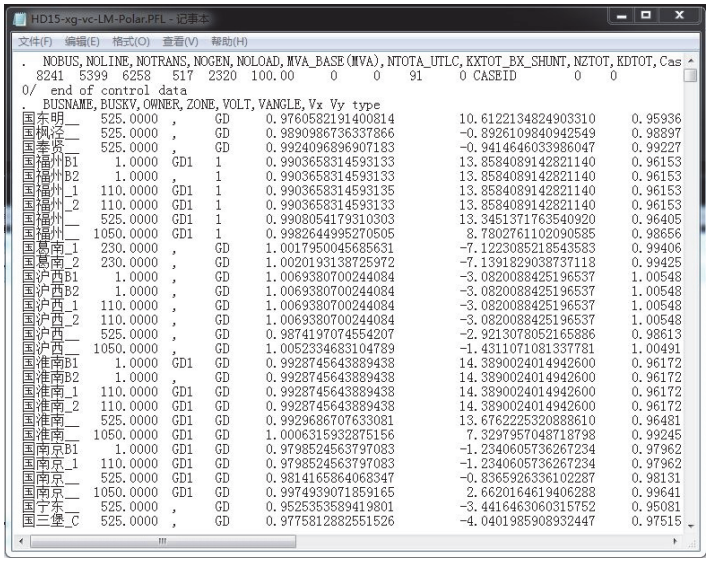


图 4-11 软件界面-潮流计算结果

Fig. 4-11 Software UI- result of power flow calculation

图 4-11 即为潮流计算的结果，文件格式为.pfl，与 BPA 的结果文件格式一致，且结果输出的规范也与 BPA 完全相同，保证了计算结果的通用性和可读性。

4.5 本章小结

本章在病态潮流算法以及稀疏技术的基础上研究了电力系统病态潮流计算的高性能实现方案。首先，根据 LM 方法求解迭代步的计算公式特征，对向上看 Cholesky 分解流程进行改进，采用隐式 Cholesky 分解算法，相较于常规的显式算法能够避免冗余计算量，提升计算效率。随后，结合第二章、第三章的研究结果，对高性能实现方案的各个环节进行方法的选取，从而完成整个方案的完整设计。

另一方面，为了实际检验方案的有效性，需要依据相关算法，开发出一套相应的潮流计算平台。本章进一步对该软件平台的开发进行了论述，通过采用 C 语言、C++以及 C#多语言混合开发方式，并结合常规的 NR 法、PQ 分解法以及应用于病态潮流计算的 LM 方法，设计出一套具有多功能的，能够高效、正确求解出良态/病态潮流的潮流计算平台。

第五章 算例测试与分析

5.1 测试环境与原始数据

对于第四章提出的电力系统病态潮流高性能实现方案，需要从正确性、高效性两方面进行全面的测试。考虑到第二章、第三章已分别对迭代步求解的直接方案/扩展矩阵方案以及稀疏分解算法的向上看算法/超节点算法进行了测试和分析，本章采用 4.4 节所述的潮流计算平台，重点测试针对 $\boldsymbol{J}^T\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵的隐式 Cholesky 分解算法应用于 LM 方法病态潮流计算中的计算正确性以及对整体方案的效率提升情况。

本文采用的测试环境为 Intel Core i5 3.0GHz 四核 CPU、8GB 内存，操作系统为 Windows 7。C++语言程序使用 Visual Studio 2010 编译，使用 Matlab R2012a 执行 Matlab 程序作为计算效率比较，使用 PSD-BPA v4.2 作为计算精度与效率比较。

本章算例所使用的系统潮流数据如表 5-1 所示。其中，实际的大规模系统数据取自华东电网。

表 5-1 算例测试系统基本概况
Table 5-1 Basic information of test systems

系统名称	节点数	支路数	数据来源
IEEE9 节点	9	9	IEEE9
IEEE14 节点	14	20	IEEE14
IEEE30 节点	30	41	IEEE30
IEEE39 节点	39	46	IEEE39
IEEE57 节点	57	78	IEEE57
IEEE118 节点	118	179	IEEE118
IEEE162 节点	162	280	IEEE162
IEEE300 节点	300	409	IEEE300
华东 3647 系统	3647	4239	华东电网 2004 年
华东 4171 系统	4171	4860	华东电网 2005 年
华东 4769 系统	4769	5597	华东电网 2006 年
华东 5473 系统	5473	6642	华东电网 2009 年

续表 5-1 算例测试系统基本概况

系统名称	节点数	支路数	数据来源
华东 7872 系统	7872	9830	华东电网 2014 年夏高
华东 8003 系统	8003	9986	华东电网 2014 年冬高
华东 8241 系统	8241	10280	华东电网 2015 年夏高
华东 8396 系统	8386	10443	华东电网 2015 年冬高

5.2 潮流计算正确性测试

5.2.1 良态潮流的计算精度测试

使用基于隐式 Cholesky 分解的 LM 方法潮流计算程序，对表 5-1 中除华东 8241 系统外的算例系统进行测试，将所得计算结果与 BPA 进行比对，记录下每个算例结果与 BPA 结果的最大幅值偏差以及相角偏差，结果如表 5-2 所示。

表 5-2 LM 方法良态潮流计算结果精度

Table 5-2 Precision of well-conditioned power flow result

测试系统	迭代次数	幅值偏差(p.u.)	相角偏差(°)
IEEE9 节点	4	7.03E-08	7.65E-07
IEEE14 节点	4	8.46E-08	7.56E-07
IEEE30 节点	5	3.98E-07	5.24E-06
IEEE39 节点	5	5.81E-07	7.53E-06
IEEE57 节点	5	3.49E-07	8.92E-06
IEEE118 节点	9	1.48E-07	1.33E-05
IEEE162 节点	9	1.32E-06	4.40E-05
IEEE300 节点	7	7.78E-06	3.56E-04
华东 3647 系统	9	3.85E-04	1.53E-03
华东 4171 系统	9	1.13E-04	2.30E-03
华东 4769 系统	9	2.11E-04	1.83E-03
华东 5473 系统	16	2.18E-04	1.26E-03
华东 7872 系统	15	2.94E-04	3.49E-03
华东 8003 系统	9	2.35E-04	4.82E-03
华东 8396 系统	11	2.64E-04	1.75E-03

由表 5-2 可以看出，采用本文所述方法进行良态潮流计算，其计算结果与

BPA 相比，电压幅值偏差均小于 10^{-3}p.u.，相角偏差均小于 10^{-2}° 。考虑到 BPA 计算时选取计算精度取为 5.0×10^{-3}，可以认为该数量级的偏差是由 BPA 的精度所导致。因此，认为该方法的良态潮流计算结果正确。

5.2.2 病态有解潮流的计算精度测试

对于华东 8241 节点系统，若采用 NR 法或 PQ 分解法进行求解时，迭代过程中出现雅克比矩阵奇异而不收敛的情况。具体的迭代情况以及与 BPA 结果的比对数据如表 5-3 所示。

表 5-3 LM 方法良态潮流计算结果精度

Table 5-3 Precision of well-conditioned power flow result

采用算法	迭代次数	幅值偏差(p.u.)	相角偏差(°)
极坐标 NR 法	13	2.24	190.74
直角坐标 NR 法	20	不收敛	不收敛
BX 型 PQ 分解法	17	迭代发散	迭代发散
XB 型 PQ 分解法	15	迭代发散	迭代发散

根据表 5-3，可以看出采用直角坐标下的 NR 法，该系统的潮流在规定的次数内未收敛到设定精度，而采用极坐标下的 NR 法，虽然收敛到了一个解，但并非是正确的潮流解。进一步，若采用 PQ 分解法，不论是 BX 型或 XB 型，均在十余次迭代后潮流发散。可以看出，常规的 NR 法及 PQ 分解法在求解该系统时已经失效。

然而，该系统仍然有潮流解，是个病态有解的系统。BPA 计算程序由于先采用几次 PQ 分解法进行迭代，能够提高算法的收敛性，故可求得正确的潮流解。而采用本文所述的基于 LM 方法的病态潮流算法同样可以求得收敛的解，将两者结果进行比对，其最大的电压幅值偏差与相角偏差如表 5-4 所示。

表 5-4 LM 方法病态潮流计算结果精度

Table 5-4 Precision of ill-conditioned power flow result

测试系统	迭代次数	幅值偏差(p.u.)	相角偏差(°)
华东 8241 节点	9	2.68E-03	3.74E-02

由表 5-4 的数据，采用 LM 方法所得的解与 BPA 所得结果的幅值偏差、相角偏差分别在 10^{-3} 和 10^{-2} 数量级，可以认为采用本文所述方法求解华东 8241 节点系统的结果与 BPA 结果一致。

5.2.3 病态无解潮流的最小二乘解测试

对于华东 8241 节点系统，若进一步将系统中上海区域节点的有功与无功负荷均等比例扩大 1.1 倍，系统会由病态有解系统转变为病态无解系统。此时系统不再存在真实的潮流解，常规 NR 法、PQ 分解法与 BPA 潮流程序计算该系统均无法收敛。对于 BPA 程序而言，若迭代不收敛，程序不会输出任何结果。对于使用者而言，仅知道潮流无法收敛，却无法进一步获取任何信息分析不收敛的原因。而采用 4.4 节所述计算平台中的 NR 法与 PQ 分解法模块，随能够得到一个输出的结果，但结果的功率偏差一般非常大，不具备任何参考价值。

而采用 LM 方法对该系统进行病态潮流计算，经过了 142 次迭代后，潮流计算可以收敛至具有 10^{-2} 精度的最小二乘解。整个迭代过程中注入功率偏差的变化如图 5-1。

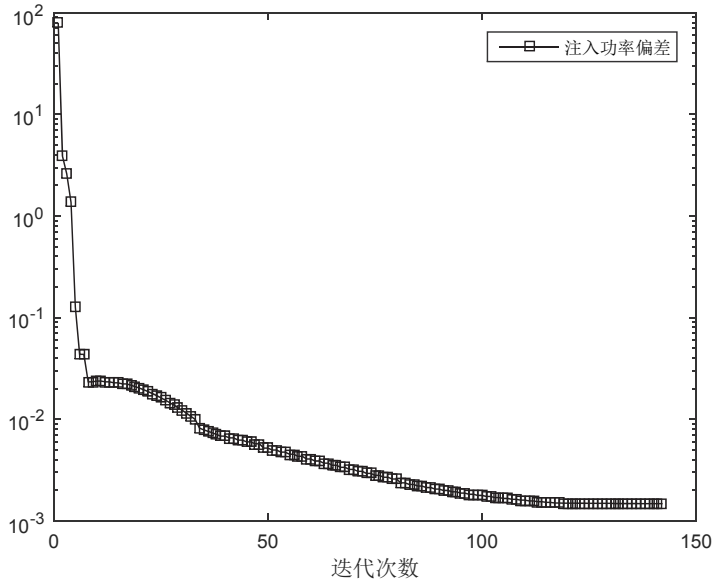


图 5-1 注入功率偏差随迭代次数的变化

Fig. 5-1 Variation of power mismatches to iterations

由图 5-1 可知，在经过约 30 次迭代后，注入功率偏差的数量级就不发生变化，保持在 10^{-2} 级别并持续减小，当进行到第 142 次迭代时 $\left\|\boldsymbol{J}^{T}\boldsymbol{F}\right\|=4.2\times10^{-6}$ 满足 LM 方法所设定的收敛条件。此时求得系统的最小二乘解，可作为近似潮流解为后续分析及调整提供依据。

对于 LM 方法计算所得的最小二乘解，可以将其电压幅值绘制成曲线，如图 5-2。

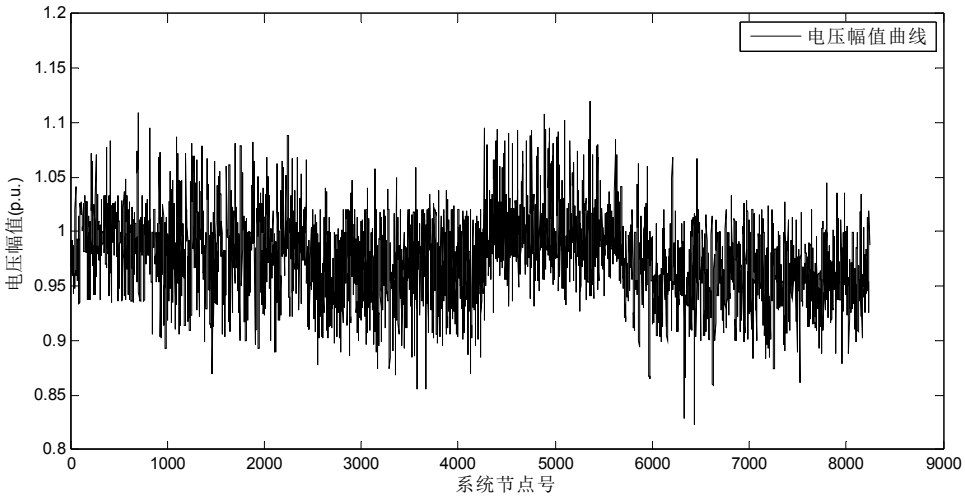


图 5-2 上海区域负荷 1.1 倍时最小二乘解的电压幅值曲线

Fig. 5-2 The voltage amplitude of least square solution with Shanghai's load enlarged to 1.1 times

由图 5-2 可以看出，整个系统的电压幅值最低不到 0.81，最高超过 1.1。结合最小二乘解的数据可以发现共有 591 个节点的电压幅值低于 0.9，最低的数个节点分别为：浙翠屏__37、浙翠屏 B2、浙翠屏 B1、浙翠屏__115、浙屯山__、闽金榜 11 等。而电压幅值高于 1.1 的节点共有 2 个，分别是苏任庄_1 和苏任庄_2。

进一步，可以采用 LM 方法判断系统的极限点，并利用所得最小二乘解进行分析。首先，对华东 8241 节点系统中上海区域所有节点的负荷进行等比提升，使用 LM 方法程序进行求解时，迭代次数随负荷倍数变化的情况如图 5-3。

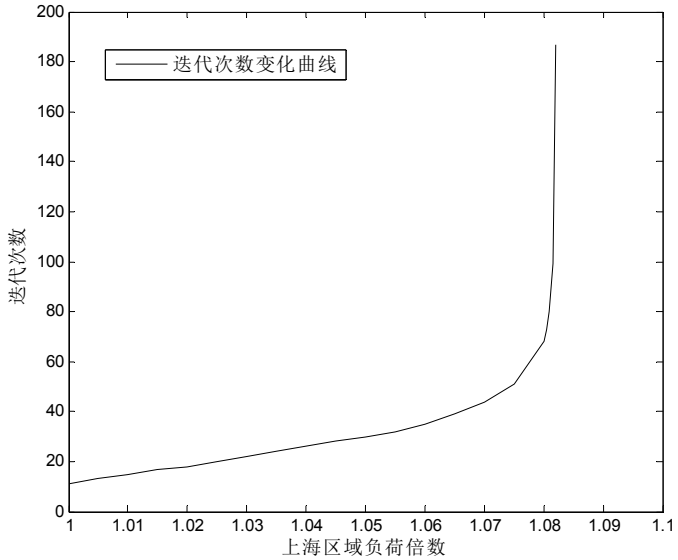


图 5-3 LM 方法迭代次数随负荷倍数变化曲线

Fig. 5-3 The variation of iterations of LM method with respect to load multiplier

由图 5-3，可以看出随着负荷倍数的上升，采用 LM 方法的潮流计算迭代次数逐步上升，当倍数约为 1.082 时，迭代次数发生了突变，由 99 次上升为 187 次，且所得结果不再是精确潮流解，而是最小二乘解。因此，可以认为系统的极限点大约发生在负荷倍数为 1.082 倍处。

当上海区域负荷倍数为 1.082 倍时，潮流的最小二乘解电压幅值如图 5-4。

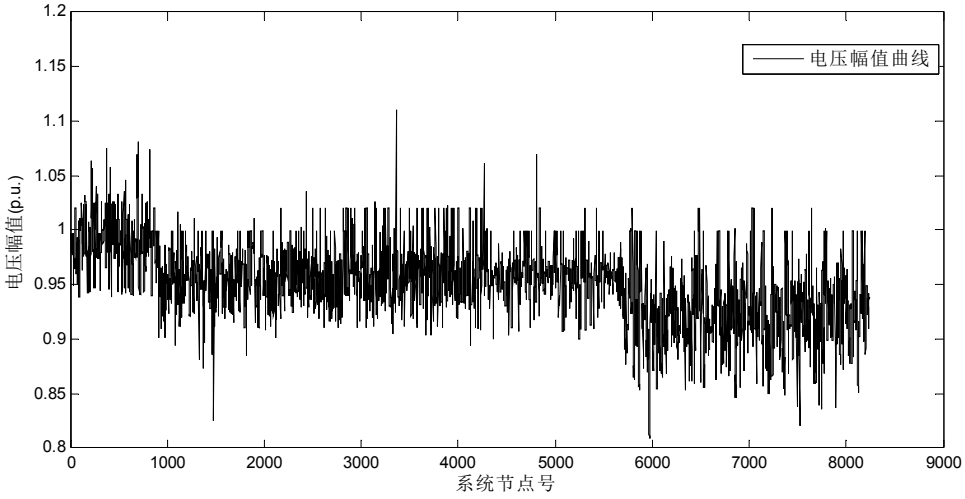


图 5-4 上海区域负荷 1.082 倍时最小二乘解的电压幅值曲线

Fig. 5-4 The voltage amplitude of least square solution with Shanghai's load enlarged to 1.082 times

根据最小二乘解的数据，系统中电压最低点为浙翠屏__37，若对该节点进行无功电压补偿，则节点电压幅值随补偿容量变化的曲线图如图 5-5 所示。

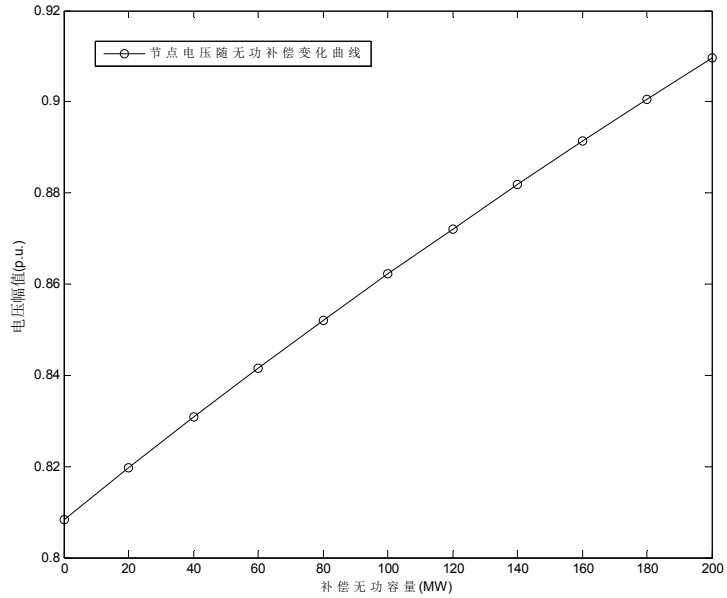


图 5-5 节点电压随无功补偿容量变化曲线

Fig. 5-5 The variation of node voltage with respect to reactive power compensation

由图 5-5 可以看出，随着补偿无功容量的增加，节点浙翠屏__37 的电压幅值也随之增加，体现出了无功补偿的作用。因而，采用 LM 方法所求得的最小二乘解对于临界方式下的稳定分析具有实际的参考价值。

5.3 潮流计算效率测试

5.3.1 稀疏矩阵分解计算效率测试

选取 LM 方法第一步迭代时各系统的系数矩阵 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$，分别采用显式 Cholesky 分解方法与本文所述隐式 Cholesky 分解方法，测试各自因子分解的时间，包括符号分解与数值分解两部分。测试所得数据如表 5-5 所示。

表 5-5 稀疏矩阵分解时间

Table 5-5 Run time of sparse matrix factorization

测试系统	隐式分解		显式分解	
	符号分解时间(s)	数值分解时间(s)	符号分解时间(s)	数值分解时间(s)
IEEE9 节点	0.000004	0.000008	0.000008	0.000010
IEEE14 节点	0.000006	0.000012	0.000014	0.000016
IEEE30 节点	0.000021	0.000064	0.000058	0.000079
IEEE39 节点	0.000026	0.000068	0.000069	0.000086
IEEE57 节点	0.000040	0.000075	0.000118	0.000142
IEEE118 节点	0.000067	0.000157	0.000213	0.000282
IEEE162 节点	0.000110	0.000756	0.000355	0.000900
IEEE300 节点	0.000175	0.000594	0.000437	0.000746
华东 3647 系统	0.001911	0.005731	0.005107	0.006956
华东 4171 系统	0.001972	0.006679	0.005762	0.008073
华东 4769 系统	0.002441	0.007998	0.006158	0.009093
华东 5473 系统	0.002659	0.008733	0.007046	0.010903
华东 7872 系统	0.003604	0.012033	0.009526	0.014919
华东 8003 系统	0.003660	0.012435	0.009716	0.015122
华东 8241 系统	0.003784	0.012772	0.010437	0.015906
华东 8386 系统	0.003850	0.012997	0.010669	0.015976

根据表 5-5 的数据，首先对显式 Cholesky 分解与隐式 Cholesky 分解各自的符号分解数据进行对比分析，两者的符号分解时间随仿真系统规模的变化曲线如图 5-6。

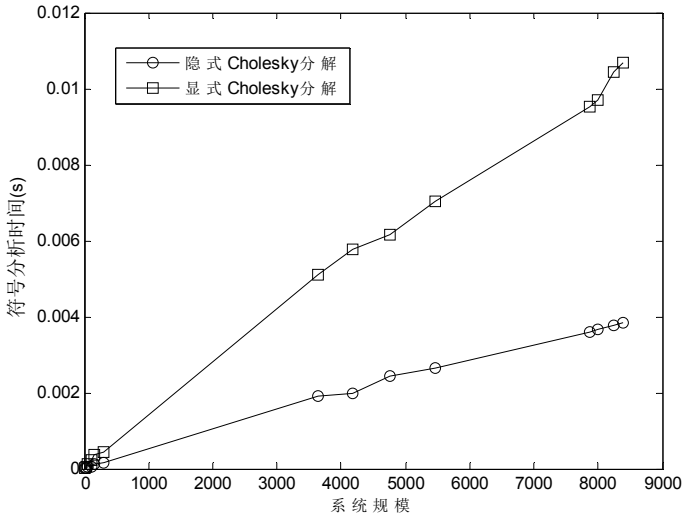


图 5-6 符号分解时间对比

Fig. 5-6 Comparison on running time of symbolic analysis

由图 5-6 可见，在符号分解的耗时部分，对于同一系统，隐式分解方法的运算时间远低于显式方法，一般仅为显式的三分之一左右。根据 5.2.2 节的论述，隐式 Cholesky 符号分解相较于显式方法更为快速地原因是其构建了一个规模不大于原矩阵的星矩阵 $\boldsymbol{J}^*$，避免了通过模拟矩阵相乘获得 $\boldsymbol{J}^T\boldsymbol{J}$ 阵非零结构的过程。为了进一步对隐式符号分解算法耗时低的原因进行分析，可统计隐式符号分解步骤中星矩阵 $\boldsymbol{J}^*$的非零元个数，并与显式方法中的 $\boldsymbol{J}^T\boldsymbol{J}$ 阵和原矩阵 $\boldsymbol{J}$ 进行对比。测试所得结果如表 5-6 所示。

表 5-6 符号分解过程中矩阵非零元比较

Table 5-6 Comparison of matrices' number of nonzero entries during symbolic analysis

测试系统	原矩阵 $\boldsymbol{J}$ 非零元个数	星矩阵 $\boldsymbol{J}^*$非零元个数	$\boldsymbol{J}^T\boldsymbol{J}$ 阵非零元个数
IEEE9 节点	82	30	146
IEEE14 节点	146	51	308
IEEE30 节点	333	141	829
IEEE39 节点	445	191	949
IEEE57 节点	718	331	1596
IEEE118 节点	1051	464	2791
IEEE162 节点	2767	1080	7675
IEEE300 节点	3736	1548	9736
华东 3647 系统	46540	18654	103828
华东 4171 系统	53163	21262	131075

续表 5-6 符号分解过程中矩阵非零元比较

测试系统	原矩阵 $\boldsymbol{J}$ 非零元个数	星矩阵 $\boldsymbol{J}^*$非零元个数	$\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ 阵非零元个数
华东 4769 系统	60982	24225	137184
华东 5473 系统	71831	28400	182011
华东 7872 系统	106470	40255	271824
华东 8003 系统	108113	40904	278113
华东 8241 系统	111576	42058	285680
华东 8386 系统	113509	42785	292577

由表 5-6 可见，同一系统下雅可比矩阵 $\boldsymbol{J}$、隐式过程中生成的星矩阵 $\boldsymbol{J}^*$与显式过程中生成的 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ 阵三者的非零元个数，符合本文所述 $\mathrm{nnz}(\boldsymbol{J}^*)\leq\mathrm{nnz}(\boldsymbol{J})\leq\mathrm{nnz}(\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J})$的关系。且可以看出，隐式符号分解中所形成 $\boldsymbol{J}^*$阵的非零元个数仅为显式形成 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ 阵的 15%左右。由于 $\boldsymbol{J}^*$阵的非零元个数较少，隐式符号分解在形成消去树及 $\boldsymbol{L}$ 阵结构时可以避免对冗余非零元的访问，获得更高的计算速度。此外随着矩阵规模的扩大，采用隐式符号分解可以在计算更高速的同时节约大量的存储空间开销。

而对于数值分解部分，根据表 5-5 的数据，显式方法与隐式方法两者的数值分解时间随仿真系统规模的变化曲线如图 5-7。

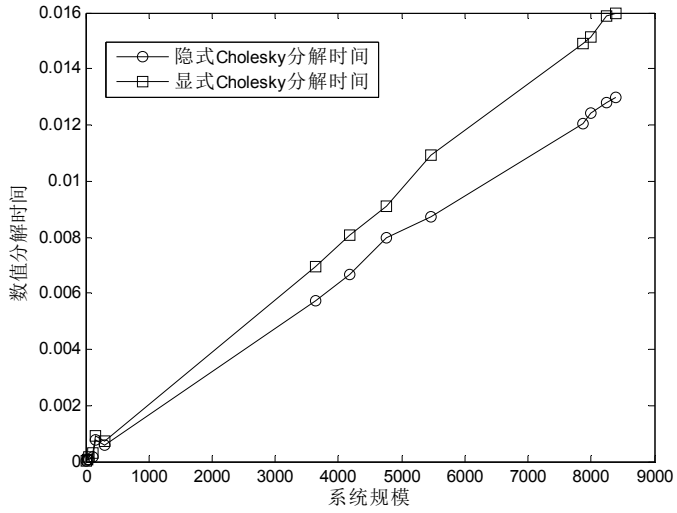


图 5-7 数值分解时间对比

Fig. 5-7 Comparison on running time of numerical factorization

由图 5-7，相比于显式分解方法，隐式数值分解能够提速 20%~25%。这部分计算时间的节省是由于隐式数值分解避免了计算 $\boldsymbol{B}$ 阵下三角数值造成的冗余计算量和矩阵相加时寻找对角元所需的额外定位索引时间。并且随着矩阵规模的扩大以及潮流计算中迭代次数的增多，相比于显式方法，隐式数值分解所能节省的绝

对时间会进一步增加。

5.3.2 潮流计算整体效率测试

使用基于隐式 Cholesky 分解的 LM 方法潮流计算程序测试表 5-1 中系统潮流的整体计算时间，并与显式求解迭代步的 LM 方法程序以及 BPA 的计算时间进行对比。测试结果如表 5-7 所示。

表 5-7 潮流计算整体时间对比

Table 5-7 Comparison on running time of power flow calculation

测试系统	采用隐式 Cholesky 算法的 LM 方法计算时间(s)	采用显式 Cholesky 算法的 LM 方法计算时间(s)	BPA 计算时间(s)
IEEE9 节点	0.01344	0.01738	0.03668
IEEE14 节点	0.01675	0.02092	0.03816
IEEE30 节点	0.01776	0.02332	0.04258
IEEE39 节点	0.02369	0.02862	0.04852
IEEE57 节点	0.02347	0.02869	0.04742
IEEE118 节点	0.02317	0.02766	0.05342
IEEE162 节点	0.04660	0.05757	0.09433
IEEE300 节点	0.04187	0.04762	0.09774
华东 3647 系统	0.15463	0.18320	0.38989
华东 4171 系统	0.16920	0.19799	0.46837
华东 4769 系统	0.17410	0.20474	0.50990
华东 5473 系统	0.42277	0.46614	0.57377
华东 7872 系统	0.53209	0.61135	0.84389
华东 8003 系统	0.54904	0.62662	0.92260
华东 8241 系统	0.52203	0.57711	0.86496
华东 8386 系统	0.56294	0.63291	0.96275

此外，对潮流计算过程中的迭代次数进行统计，结果如表 5-8 所示。

表 5-8 潮流计算迭代次数对比

Table 5-8 Comparison on iterations of power flow calculation

测试系统	LM 方法迭代次数	BPA 迭代次数
IEEE9 节点	4	5
IEEE14 节点	4	5
IEEE30 节点	5	5
IEEE39 节点	5	5
IEEE57 节点	5	5

续表 5-8 潮流计算迭代次数对比

测试系统	LM 方法迭代次数	BPA 迭代次数
IEEE118 节点	5	5
IEEE162 节点	9	6
IEEE300 节点	7	6
华东 3647 系统	9	5
华东 4171 系统	9	6
华东 4769 系统	9	5
华东 5473 系统	16	5
华东 7872 系统	15	6
华东 8003 系统	16	7
华东 8241 系统	9	6
华东 8386 系统	11	7

根据表 5-7 的计算数据，可以绘制采用隐式 Cholesky 分解的 LM 方法、采用显式 Cholesky 分解的 LM 方法以及 BPA 程序计算同一系统的计算时间随系统规模的变化趋势，如图 5-8。

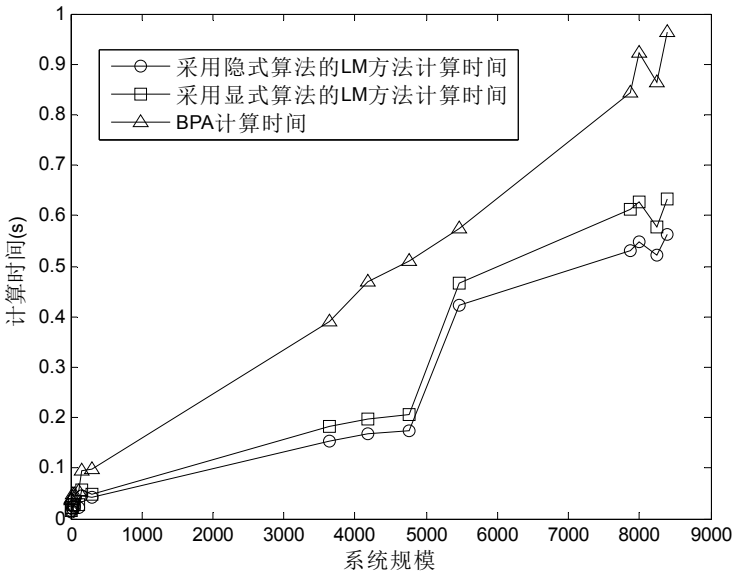


图 5-8 潮流计算整体计算时间

Fig. 5-8 Total run time of power flow calculation

根据表 5-7 数据与图 5-8 的变化趋势，并结合表 5-8 的迭代次数数据，可以对整体计算时间做出如下分析：

1）整体而言，不论是采用隐式/显式分解的 LM 方法，或是 BPA 程序，计算系统潮流的总体时间都随系统规模的增大而增大。另一方面，潮流计算的时间又

与迭代次数密切相关。如华东 8003 系统与华东 8241 系统，两系统的规模差距不大，而采用 LM 方法的迭代次数分别为 16 次与 9 次，导致用 LM 方法计算华东 8241 系统的时间反而少于比计算规模更小的华东 8003 系统；

2）采用本文所述的基于隐式 Cholesky 分解的 LM 方法进行潮流计算，相对于显式求解迭代步的 LM 方法程序能够提速 15%~20%，证明了隐式 Cholesky 分解方法应用在 LM 方法求解中的提速效果；

3）对比基于隐式 Cholesky 分解的 LM 方法与 BPA 的计算时间，对于规模较小的 IEEE 系统，由于两者的迭代次数相近，前者的计算时间仅为后者的三分之一左右。而对于规模较大的华东电网系统，虽然阻尼因子 μ 限制了迭代步的长度，导致 LM 方法的迭代次数多于采用 NR 法的 BPA 程序，但由于核心稀疏求解部分的高效性，基于隐式 Cholesky 分解的 LM 方法潮流计算程序相比 BPA 程序依旧能够缩短近一半的计算时间。对于系统规模过 8000 节点的实际系统，潮流计算总时间大约为 0.5 s 左右，具有良好的计算效率与实用价值。

5.4 本章小结

本章以标准 IEEE 系统以及华东电网数年实际潮流数据作为算例，测试本文所采用的电力系统病态潮流高性能实现方案的正确性与高效性。对于良态系统和病态有解系统，通过计算结果与 BPA 的比对验证基于 LM 方法病态潮流计算的结果正确性。随后，通过等比例扩大负荷的方式认为构造病态无解系统，对 LM 方法所得最小二乘解进行分析，证明该最小二乘解对于实际分析具有参考价值。最后，在高效性的测试方面，本章对计算过程中的矩阵分解时间和整体时间进行统计，通过方法间的计算时间比较，验证了隐式 Cholesky 分解的高效性，并证明该方法对于整体病态潮流计算的效率提升有较好的贡献。

第六章 总结与展望

6.1 主要研究工作

电力系统潮流计算中，因雅克比矩阵奇异导致常规牛顿法求解不收敛的病态潮流问题受到了大量学者的关注。本文以病态潮流计算的高性能实现为研究对象，首先总结了主流的病态潮流求解算法，并进一步对算法中迭代步求解的不同方案进行对比分析。考虑到电力系统网络拓扑所造成的矩阵稀疏特性，本文对相应的稀疏存储、稀疏分解算法等稀疏技术进行研究，并根据电力系统的矩阵特征进行适应性分析。最后，本文从电力系统病态潮流求解的特性出发，对数值分解算法进行改进，采用 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵的隐式 Cholesky 分解算法，进一步提升计算效率，并在此基础上完成病态潮流计算高性能实现的方案设计以及相应的病态潮流计算平台开发。通过以上研究工作以及相关的算例仿真，得到以下结论：

（1）在主流的电力系统病态潮流求解算法中，自适应 LM 方法从计算量和收敛性两方面都具有较好的特性，可以有效解决电力系统潮流计算的收敛性问题。在自适应 LM 方法求解病态潮流的计算过程中，主流认为采用扩展矩阵求解迭代步的方案能够减少计算量。然而通过实际测试，发现扩展矩阵方案会在矩阵分解过程中引入大量的注入元，减缓计算速度，其实际应用表现不及直接求解迭代步的方案。

（2）电力系统的网络连接特性导致了潮流计算过程中矩阵的极度稀疏特性，通过引入稀疏技术能够极大地提升计算效率。在自适应 LM 方法求解病态潮流的计算过程中，主要应用到稀疏 Cholesky 分解算法，其主流实现包括向上看 Cholesky 分解与超节点 Cholesky 分解算法。其中，后者通过将多列稀疏结构相同的列组成稠密块，引入了高效 3 级 BLAS 操作，加快计算的进行。通过实际测试，发现超节点 Cholesky 分解算法在应用于流体动力学等领域的矩阵时具有明显的优势。但对于电力系统病态潮流计算中产生的矩阵，由于矩阵的极度稀疏特性，导致算法难以形成大规模的超节点，影响了 3 级 BLAS 操作的加速效果，使得超节点 Cholesky 分解算法的应用效果不及经典的向上看 Cholesky 分解算法。

（3）在自适应 LM 方法求解病态潮流的计算过程中，需要对 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 矩阵进行 Cholesky 分解。常规的显式分解方法不可避免地需要通过矩阵乘法与加法计算 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}+\mu\boldsymbol{I}$ 全矩阵，产生了冗余计算量。本文采用隐式 Cholesky 分解的方法，在

符号分解部分形成一个规模不大于原矩阵，且符号分解结果与 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ 一致的星矩阵，避免了计算 $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ 稀疏结构的步骤；在数值分解部分仅对分解中需要用到的数值进行计算，并利用算法特点避免了稀疏矩阵加法运算，减少了冗余的计算量。

（4）通过实际算例分析，采用自适应 LM 方法的病态潮流计算能够正确求解良态系统潮流以及病态有解潮流的精确潮流解。对于病态无解系统，采用 LM 方法可求得系统的最小二乘解，具有实际的参考价值。而通过将隐式 Cholesky 分解引入病态潮流的计算中，可以在保证正确性的前提下较好地提升计算速度，为病态潮流计算的高性能实现提供有力工具。

6.2 未来工作展望

本文就如何对电力系统的病态潮流进行高性能实现进行了研究，取得了一定进展。同时也要承认本文的内容还有进一步研究的空间，后续可以从以下几个方面展开更深入的研究：

（1）病态求解潮流算法方面，现阶段 LM 方法在求解病态无解潮流问题时存在迭代次数较多的问题，可进一步对 LM 方法中阻尼因子的选取和更新方式进行研究，从算法的原理上提升收敛性和收敛速度，减少计算的迭代次数；

（2）稀疏分解算法方面，考虑到超节点 Cholesky 算法在其他领域的良好应用效果以及其基于 BLAS 的并行可行性，可以在其他电力系统仿真相关的问题上探讨其应用可行性，为提升电力系统仿真计算速度提供帮助；

（3）实际应用方面，本文的病态潮流计算基于交流系统，考虑到现在特高压直流线路的建设，可以将自适应 LM 方法、隐式 Cholesky 分解等引入交直流潮流计算中，并开发相应的计算软件，从而提升交直流潮流计算的收敛性和计算速度。

参 考 文 献

[1] 王锡凡. 现代电力系统分析[M]. 北京: 科学出版社, 2003.

[2] Ward J B, Hale H W. Digital computer applications solution of power flow problems, Power Apparatus and Systems, Part III. Transactions of the American Institute of Electrical Engineers, 1956, 75(3): 398-404.

[3] Brown H E, Carter G K, Happ H H, et al. Power flow solution by impedance matrix iterative method[J]. IEEE Transactions on Power Apparatus and Systems, 1963, 82(65): 1-10.

[4] Tinney W F, Hart C E. Power flow solution by Newton's method[J]. IEEE Transactions on Power Apparatus and Systems, 1967 (11): 1449-1460.

[5] Stott B, Alsa O. Fast decoupled load flow[J]. IEEE Transactions on Power Apparatus and Systems, 1974 (3): 859-869.

[6] Tamura Y, Nakanishi Y, Iwamoto Y. On the multiple solution structure, singular point and existence condition in the load flow multiple solutions[C]. IEEE PES Winter Meeting, 1980.

[7] Tamura Y, Mori H, Iwamoto S. Relationship between voltage instability and multiple load flow solutions in electric power systems[J]. IEEE Transactions on Power Apparatus and Systems, 1983 (5): 1115-1123.

[8] Iwamoto S, Tamura Y. A load flow calculation method for ill-conditioned power systems[J]. IEEE Transactions on Power Apparatus and Systems, 1981(4): 1736-1743.

[9] 王宪荣，包丽明. 极坐标系准最优乘子病态潮流解法研究[J]. 中国电机工程学报, 1994, 14(1): 40-45.

[10] Tate J E, Overbye T J. A comparison of the optimal multiplier in polar and rectangular coordinates[J]. IEEE Transactions on Power Systems, 2005, 20(4): 1667-1674.

[11] Schnabel R B, Frank P D. Tensor methods for nonlinear equations[J]. SIAM Journal on Numerical Analysis, 1984, 21(5): 815-843.

[12] Bouaricha A, Schnabel R B. Tensor methods for large sparse systems of nonlinear equations[J]. Mathematical programming, 1998, 82(3): 377-400.

[13] Salgado R S, Zeitune A F. Power flow solutions through tensor methods[J]. Generation, Transmission and Distribution, IET, 2009, 3(5): 413-424.

[14] 林济铿，吴鹏，袁龙，等. 基于张量法的电力系统潮流计算[J]. 中国电机工程学报, 2011, 31(34): 113-119.

[15] 覃智君，侯云鹤，吴复立. 大规模交直流系统潮流计算的实用化模型[J]. 中国电机工程学报, 2011, 31(10): 95-101.

[16] 鞠平，张翠. 电力系统潮流计算的模拟进化方法[J]. 河海大学学报: 自然科学版, 1998, 26(5): 22-27.

[17] Lagace P J. Power flow methods for improving convergence[C]//IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society. Montreal: IEEE, 2012: 1387-1392.

[18] 严正，范翔，赵文恺，等. 自适应 Levenberg-Marquardt 方法提高潮流计算收敛性[J]. 中国电机工程学报, 2015, 35(8): 1909-1918.

[19] 何洋，洪潮，陈昆薇. 稀疏向量技术在静态安全分析中的应用[J]. 中国电机工程学报, 2003, 23(1): 41-44.

[20] 朱凌志，安宁. 基于二维链表的稀疏矩阵在潮流计算中的应用[J]. 电网技术, 2005, 29(8): 51-55.

[21] 罗日成，李卫国. 电力系统潮流计算的符号分解方法[J] . 电网技术, 2005, 29(10): 25-29.

[22] 徐得超，李亚楼，吴中习. 稀疏技术在电力系统状态估计中的应用[J]. 电网技术, 2007, 31(8): 32-36.

[23] 徐得超，李亚楼，郭剑，等. 消去树理论及其在潮流计算中的应用[J]. 电网技术, 2007, 31(22): 12-16.

[24] Berry T, Daniels A R, Dunn R W. Parallel processing of sparse power system equations[J]. IEE Proceedings - Generation, Transmission and Distribution, 1994, 141(1): 68-74.

[25] 苏新民，毛承雄，陆继明. 对角块加边模型的并行潮流计算[J]. 电网技术, 2002, 26(1): 22-25.

[26] 赵文恺，房鑫炎，严正. 电力系统并行计算的嵌套分块对角加边形式划分算法[J]. 电机工程学报, 2010, 30(25): 66-73.

[27] 李传栋，房大中，杨金刚，等. 大规模电网并行潮流算法[J]. 电网技术, 2008, 32(7): 34-39.

[28] 夏俊峰，杨帆，李静，等. 基于 GPU 的电力系统并行潮流计算的实现[J]. 电力系统保护与控制, 2010, 38(18): 100-103.

[29] 韩晓言，韩祯祥. 电力系统暂态稳定分析的内在并行算法研究[J]. 中国电机工程学报, 1997, 17(3): 145-148.

[30] 周挺辉，严正，唐聪，等. 基于多核处理器技术的暂态稳定并行算法[J]. 电力

系统自动化, 2013, 37(8): 70-75.

[31] 唐聪，严正，周挺辉，等. 基于图形处理器的广义最小残差迭代法在电力系统暂态仿真中的应用[J]. 电网技术, 2013, 37(5): 1365-1371.

[32] 周挺辉，赵文恺，严正，等. 基于图形处理器的电力系统稀疏线性方程组求解方法[J]. 电力系统自动化, 2015, 39(2): 74-80.

[33] 李芳，郭剑，吴中习，等. 基于 PC 机群的电力系统小干扰稳定分布式并行算法[J]. 中国电机工程学报, 2007, 27(31): 7-13.

[34] 张逸飞，严正，赵文恺，等. 基于 GPU 的分块约化算法在小干扰稳定分析中的应用[J]. 电力系统自动化, 2015, 39(22): 90-97.

[35] Levenberg K. A method for the solution of certain nonlinear problems in least squares[J]. The Quarterly of Applied Mathematics, 1944, 2(17), 2: 164-168.

[36] Marquardt D W. An algorithm for least-squares estimation of nonlinear inequalities[J]. SIAM Journal of the Society for Industrial and Applied Mathematics, 1963, 11(2): 431–441.

[37] 范翔. 应用 Levenberg-Marquardt 方法提高电力系统大规模潮流计算收敛性研究[D]. 上海交通大学, 2014.

[38] Moré J J. The Levenberg-Marquardt algorithm: implementation and theory[J]. Numerical Analysis, 1978, 630: 105-116.

[39] Byrd R H, Shultz G A. A trust region algorithm for nonlinearly constrained optimization[J] . Siam Journal on Numerical Analysis, 1987, 24(5): 1152-1170.

[40] Yamashita N, Fukushima M. On the rate of convergence of the Levenberg-Marquardt method[J]. Computing Supplementa, 2001, 15: 239-249.

[41] Fan J. A modified Levenberg-Marquardt algorithm for singular system of nonlinear equations[J]. Journal of computational mathematics-international edition, 2003, 21(5): 625-636.

[42] Fan J, Pan J. A note on the Levenberg–Marquardt parameter[J]. Applied Mathematics and Computation, 2009, 207(2): 351-359.

[43] 刘明波，王晓村. 内点法在求解电力系统优化问题中的应用综述[J]. 电网技术, 1999, 23(8): 61-64.

[44] Wei H, Sasaki H, Yokoyama R. An application of interior point quadratic programming algorithm to power system optimization problems[J]. IEEE Transactions on Power Systems, 1996, 11(1): 98-104.

[45] 张逵. 电力系统大型稀疏矩阵的快速解法[J]. 电网技术, 1981, (01).

[46] 梯华森. 稀疏矩阵[M]. 科学出版社, 1981.

[47] Gauss C F. Letter to Gerling, 26 December 1823[J]. Werke, 9: 278-281.
[48] Jacobi C G J, Ueber eine neue Auflösungsart der bei der Methode der kleinsten Quadrate vorkommenden linearen Gleichungen[J]. Astronomische Nachrichten, 1845, 22(20): 297-306.
[49] Varga R S. Matrix iterative analysis[M]. Prentice-Hall, 1962.
[50] Markowitz H M. The Elimination form of the Inverse and its Application to Linear Programming[J]. Management Science, 1957, 3(3): 255-269.
[51] Sato N, Tinney W F. Techniques for exploiting the sparsity or the network admittance matrix[J]. IEEE Transactions on Power Apparatus and Systems, 1963, 82(69): 944-950.
[52] Ogbuobiri E C, Tinney W F, Walker J W. Sparsity-directed decomposition for gaussian elimination on matrices[J]. IEEE Transactions on Power Apparatus and Systems, 1970, 89(1): 141 - 150.
[53] Duff I S. On the number of nonzeros added when Gaussian elimination is performed on sparse random matrices[J]. Mathematics of Computation, 1974, 28(125): 219-230.
[54] Liu J W H. The multifrontal method for sparse matrix solution: theory and practice[J]. SIAM Review, 1992, 34(1): 82-109.
[55] Gilbert J R, Peierls T. Sparse partial pivoting in time proportional to arithmetic operations[J]. SIAM Journal on Scientific and Statistical Computing, 1988, 9(5): 862-874.
[56] Davis T A. A parallel algorithm for sparse unsymmetric LU factorization[D]. University of Illinois at Urbana Champaign, 1989.
[57] Amestoy P R, Davis T A, Duff I S. Algorithm 837: AMD, an approximate minimum degree ordering algorithm[J]. ACM Transactions on Mathematical Software, 2004, 30(3): 381-388.
[58] Davis T A. Algorithm 907: KLU, A Direct sparse solver for circuit simulation problems[J]. ACM Transactions on Mathematical Software, 2010, 37(3): 523-530.
[59] Davis T A. Direct methods for sparse linear systems[M]. SIAM, 2006.
[60] Davis T A. Algorithm 832: UMFPACK V4.3---an unsymmetric-pattern multifrontal method[J]. ACM Transactions on Mathematical Software, 2004, 30(2): 196-199.
[61] Chen Y, Davis T A, Hager W W, et al. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate[J]. ACM Transactions on Mathematical Software, 2008, 35(3): 1-14.
[62] Yannakakis M. Computing the minimum fill-in is NP-Complete[J]. SIAM Journal on Algebraic and Discrete Methods, 1981, 2(1): 77-79.

[63] Tinney W F, Walker J W. Direct solutions of sparse network equations by optimally ordered triangular factorization[J]. Proceedings of the IEEE, 1967, 55(11): 1801-1809.

[64] Amestoy P R, Davis T A, Duff I S. An approximate minimum degree ordering algorithm[J]. SIAM Journal on Matrix Analysis and Applications, 1996, 17(4): 886-905.

[65] Parter S. The use of linear graphs in Gauss elimination[J]. SIAM Review, 1961, 2(2): 119-130.

[66] Liu J W. A compact row storage scheme for Cholesky factors using elimination trees[J]. ACM Transactions on Mathematical Software, 1986, 12(2): 127-148.

[67] Liu J W. Computer solution of large sparse positive definite systems[M]. Prentice-Hall, 1981.

[68] Gilbert J R, Ng E G, Peyton B W. An efficient algorithm to compute row and column counts for sparse Cholesky factorization[J]. SIAM Journal on Matrix Analysis and Applications, 1994, 15(4): 1075-1091.

[69] Liu J W H. A generalized envelope method for sparse factorization by rows[J]. ACM Transactions on Mathematical Software, 1991, 17(1): 112-129.

[70] Davis T A. Algorithm: 849 A concise sparse Cholesky factorization package[J]. ACM Transactions on Mathematical Software, 2005, 31(4): 587-591.

[71] Ashcraft C C, Grimes R G, Lewis J G, et al. Progress in sparse matrix methods for large linear systems on vector supercomputers[J]. International Journal of High Performance Computing Applications, 1987, 1(4): 10-30.

[72] Ng E G, Peyton B W. A supernodal Cholesky factorization algorithm for shared-memory multiprocessors[J]. SIAM Journal on Scientific Computing, 1991, 14(4): 761-769.

[73] Ng E G, Peyton B W. Block sparse Cholesky algorithms on advanced uniprocessor computers[J]. SIAM Journal on Scientific and Statistical Computing, 1993, 14(5): 1034-1056.

[74] Lawson C L, Hanson R J, Kincaid D R, et al. Basic linear algebra subprograms for fortran usage[J]. ACM Transactions on Mathematical Software, 1979, 5(3): 308-323.

[75] Davis T A, Hu Y. The university of Florida sparse matrix collection[J]. Acm Transactions on Mathematical Software, 2011, 38(1): 734-747.

[76] Gilbert J R, Li X S, Ng E G, et al. Computing row and column counts for sparse QR and LU factorization[J]. BIT Numerical Mathematics, 2001, 41(4): 693-710.

[77] Gilbert J R. Predicting structure in sparse matrix computations[J]. SIAM Journal on Matrix Analysis and Applications, 1994, 15(1): 62-79.

致　谢

本论文的研究工作是在导师严正教授的悉心指导下完成的，在研究生两年半的时间里，严正老师不仅以一个导师的身份对我的研究工作进行指导，以其渊博的学识、开阔的眼界以及敏锐的视角指引我在正确的方向上开展研究，更是通过身体力行向我展示了如何以严谨的态度以及一丝不苟的精神对待科研、对待工作。在研究生的学习过程中，我不仅扩充了专业知识，掌握了基本的科研能力，更是深受导师耳濡目染，感受到了一个专业学者的生活态度。相信导师的谆谆教诲必会让我受益终生，在此也向严正老师表达我最为诚挚的感谢！

与此同时，也要感谢课题组的宋依群老师和冯冬涵老师，在我的研究生生活中给予我许多教导、关心和帮助。此外，感谢本科及研究生期间所有教导过我的老师们，感谢你们的辛勤付出！

感谢实验室的韩冬师兄、赵文恺师兄、李磊师兄和周挺辉师兄，在我的研究过程，尤其是在初期探索阶段中提供了莫大的帮助，很荣幸能作为你们的师弟，站在你们的肩膀上开展进一步研究。感谢张逸飞同学，作为实验室的伙伴和朋友，在同一个领域内一起完成项目工作、探讨研究内容，很幸运能并肩战斗。感谢周全、李牧青、韩笑，作为同级的实验室战友，一同分享研究心得以及生活中的苦和乐。也感谢实验室的徐潇源师兄、唐聪师兄、范翔师兄、周云师兄、张良师兄、张娜娜师姐，为我树立了良好的榜样，以及马洪艳、曹佳、李亦言、刘扬、孔祥瑞、赵小波、孙弢、郑琨琪等等的实验室兄弟姐妹们为我带来的欢乐和温暖。

衷心感谢我的父母，在我的求学生涯中时刻关注着我，鼓励着我。大爱如山，唯有用更辛勤的努力来报答你们。感谢大学中的好友们，给予我支持和鼓舞。

最后，再一次感谢所有关心、帮助过我的人们！

攻读硕士学位期间已发表或录用的论文

[1] 张道天，严正，韩冬，等. 采用灰色聚类方法的智能变电站技术先进性评价[J]. 电网技术, 2014, 07: 1724-1730. (EI)

[2] 张道天，严正，徐潇源，等. 采用隐式 Cholesky 分解的大规模病态潮流计算[J]. 电网技术, 已录用. (EI)

[3] Han Dong, Yan Zheng, Zhang Daotian, et al. Assessing the impact of advanced technologies on utilization improvement of substations[J]. Journal of Electrical Engineering & Technology, 2015, 10(5): 709-717. (SCIE)

攻读硕士学位期间申请的发明专利

[1] 张道天，严正，赵文恺，等. 大规模电力系统病态潮流分析系统[P]. 上海: CN104732459A, 2015-06-24 公开.

上海交通大学

学位论文原创性声明

本人郑重声明：所呈交的学位论文《电力系统病态潮流计算的高性能实现研究》，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：张道天

日期：2016年 1月 18日

上海交通大学

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

保密□，在___年解密后适用本授权书。

不保密☑。

（请在以上方框内打“√”）

学位论文作者签名：张道天　　指导教师签名：

日期：2016年1月18日　　日期：2016年1月18日