2008

Accurate and Robust Preconditioning Techniques for Solving General Sparse Linear Systems

Eun-Joo Lee
University of Kentucky, elee3@csr.uky.edu

ABSTRACT OF DISSERTATION

Eun-Joo Lee

The Graduate School
University of Kentucky
2008

Accurate and Robust Preconditioning Techniques for Solving General Sparse Linear Systems

ABSTRACT OF DISSERTATION

A dissertation submitted in partial
fulfillment of the requirements for
the degree of Doctor of Philosophy
in the College of Engineering at the
University of Kentucky

By
Eun-Joo Lee
Lexington, Kentucky

Director: Dr. Jun Zhang, Professor of Computer Science
Lexington, Kentucky 2008

ABSTRACT OF DISSERTATION

Accurate and Robust Preconditioning Techniques for Solving General Sparse Linear Systems

Preconditioned Krylov subspace methods are generally regarded as one class of the most promising techniques for solving very large and sparse linear systems, but these methods may produce instability and/or inaccuracy problems for certain matrices. Inaccuracy problems are usually caused by the difficulty in determining parameters or threshold values in constructing preconditioners for some particular matrices. In addition, small or zero pivots in indefinite matrices may yield preconditioners that are unstable and inaccurate. The purpose of this study is mainly concerned with improving stability and accuracy of preconditioners.

Firstly, for the purpose of improving the accuracy of incomplete lower-upper (ILU) factorization, two preconditioning accuracy enhancement strategies were proposed. The strategies employ the elements that are dropped during ILU factorization and utilize them in different ways with separate algorithms. The first strategy (error compensation) applies the dropped elements to the lower and upper parts of the LU factorization to compute a new error compensated LU factorization. The other strategy (inner-outer iteration), which is a variant of the ILU factorization, embeds the dropped elements in the preconditioning iteration process.

Secondly, in order to increase the accuracy and stability of preconditioners in solving indefinite matrices, hybrid reordering and two-phase preconditioning strategies based on incomplete LU (ILU) factorization and a sparse approximate inverse (SAI) preconditioner, respectively are considered. These strategies attempt to replace the small or zero pivots with large values, ensuring that the resulting preconditioners are better conditioned. Specifically, the hybrid reordering strategies efficiently search for the entries of single element and/or the maximum absolute value to place the elements on the main diagonal of the original matrix, and the two-phase preconditioning strategy adopts the idea of a shifting method that adds a value to the diagonals of an indefinite matrix. The two-phase preconditioning strategy produces an inverse approximation of the shifted matrix by constructing a SAI preconditioner in each phase. The two inverse approximation matrices produced from each phase are then combined to be used as a preconditioner.

Thirdly, with the intention of enhancing the convergence performance of the preconditioned iterative solvers, two sparsity pattern selection algorithms for a factored sparse approximate inverse preconditioner are considered. The sparsity pattern is adaptively updated in the construction phase of a preconditioner by using combined information of the inverse and original triangular factors of the original matrix. In order to determine the sparsity pattern, the first algorithm uses the norm of the inverse factors multiplied by the largest absolute value of the original factors, and the second employs the norm of the inverse factors divided by the norm of the original factors.

KEYWORDS: Linear System, Preconditioning, Incomplete LU (ILU) Factorization, Indefinite Matrices, Sparse Approximate Inverse (SAI) Preconditioner

Author's signature: Eun-Joo Lee

Date: August 20, 2008

Accurate and Robust Preconditioning Techniques
for Solving General Sparse Linear Systems

By
Eun-Joo Lee

Director of Dissertation:	Dr. Jun Zhang
Director of Graduate Studies:	Dr. Andrew Klapper
Date:	August 20, 2008

DISSERTATION

Eun-Joo Lee

The Graduate School
University of Kentucky
2008

Accurate and Robust Preconditioning Techniques for Solving General Sparse Linear Systems

DISSERTATION

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the College of Engineering at the University of Kentucky

By
Eun-Joo Lee
Lexington, Kentucky

Director: Dr. Jun Zhang, Professor of Computer Science
Lexington, Kentucky 2008

ACKNOWLEDGMENTS

It is pleasure to thank many people who made this thesis possible. Without their helpful support, and encouragement, I would have never been able to complete this work.

First of all, I would like to express my gratitude to my Advisory Committee: Dr. Jun Zhang, Dr. Jerzy W. Jaromczyk, Dr. Ruigang Yang, Dr. Fuqian Yang, and Dr. Lawrence A. Harris for their helpful comments on my dissertation.

It is difficult to overstate my gratitude to my Ph.D. supervisor, Dr. Jun Zhang. With his enthusiasm, his inspiration, and his great efforts to explain things clearly and simply, he helped to make research in computer science fun and exciting for me. Throughout my thesis-writing, he provided encouragement, sound advice, good teaching, good company, and lots of good ideas.

Also, I am deeply grateful to Dr. Grzegorz W. Wasilkowski and Dr. Debby Keen, for their kind assistance with writing letters, giving wise advice, helping with various applications, and so on.

I am indebted to many student colleagues for providing a stimulating and fun environment in which to learn and grow. Especially, I would like to thank Dr. Ning Kang at UC San Diego, Dr. Chi Shen at Kentucky State University, Dr. Wensheng Shen at SUNY Brockport, and Dr. Shuting Xu at Virginia State University, Mr. Ning Cao, Mr. Dianwei Han, Mr. Xuwei Liang, Mr. Zhenmin Lin, Mr. Lian Liu, Ms. Jie Wang, Mr. Yin Wang, Mr. Changjiang Zhang, Mr. Qi Zhuang and other group members in our lab. Working together with all of you has been not only a unforgettable experience, but a great pleasure as well.

I wish to thank my entire extended family for providing a loving environment for me. My brother, my sisters, aunts, my parents-in-law were particularly supportive.

Specially, I would like to thank my husband, Kideog Jeong, for his love and support.

Lastly, and most importantly, I wish to thank my parents, Yong-Sub Lee and Myung-Ja Kim. They bore me, raised me, supported me, taught me, and loved me. To them I dedicate this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Files

Chapter 1 Introduction

Many computational science and engineering problems can be formulated in the form of partial differential equations (PDEs). The typical way to solve such equations is to discretize them, changing a continuous problem into a discrete problem. These discretizations generally result in linear systems with large and sparse coefficient matrices, which have to be solved efficiently.

For solving a linear system of the form:

$$A\,x = b, \tag{1.1}$$

either direct or iterative methods can be used. Here, A is a real (or complex) valued sparse coefficient matrix of order n, x is an unknown vector, and b is a known right-hand side vector. Direct methods, based on the factorization of the coefficient matrix A into easily invertible matrices, allow exact computations up to the machine accuracy. On account of their robustness, these methods are widely employed in many industrial areas where reliability is the primary concern. Direct methods, however, often become inefficient for solving large scale problems. Especially, the problems arising from the discretization of PDEs in three space dimensions sometimes lead to linear systems comprising of hundreds of millions or even billions of equations [8]. Generally speaking, direct methods are not useful to solve such large problems because (1) they have a computational complexity of $\mathcal{O}(n^3)$ on an $n \times n$ matrix and (2) a large amount of memory cost caused by the fill-ins, nonzero elements into original zero element locations, during the Gaussian elimination.

Iterative methods should be utilized for solving large-scale problems by reason of their computational efficiency. These methods generate a sequence of approximate solutions to the system that (hopefully) converges to the exact solution. In addition, iterative methods require fewer storage and fewer operations than direct methods in solving problems because iterative methods avoid some of the costs associated with fill-in in matrix factors. However, iterative methods do not have the same reliability of direct methods, and the convergence of iterative methods is related to the condition number of a coefficient matrix. This results that iterative methods often fail to attain convergence in a reasonable amount of time or even diverge in solving ill-conditioned problems.

Preconditioning a matrix is useful to accelerate the convergence of iterative methods. Preconditioning is a way to transform the original linear system (1.1) into another system of the form:

$$M^{-1}Ax = M^{-1}b, \tag{1.2}$$

where M is a nonsingular matrix with the same order of A. Here, the system (1.2) is preconditioned from left, and the matrix M is called a (left) preconditioner. The preconditioned system (1.2) has the same solution of the original linear system (1.1), and $M^{-1}A$ generally attempts to provide a smaller condition number than A.

It has been widely noticed that preconditioning is the most crucial part in the development of efficient solvers for challenging problems in scientific computation [8, 57]. Moreover, the importance of preconditioning has been more weighted as the size of the problems to be solved has become larger and larger. As a result, research in matrix preconditioning has received more attention than both direct methods and iterative methods in past few years, and lots of efforts have been made in developing effective preconditioners. This dissertation mainly focuses on preconditioning techniques to improve the performance and reliability of iterative methods in solving general sparse matrices.

1.1 Motivation

Preconditioned Krylov subspace methods, which are based on projection processes and are one of the most important iterative methods for solving large linear systems, are generally regarded as one class of the most promising techniques [9, 17, 19, 41] for solving very large and sparse linear systems. In general, incomplete lower-upper (ILU) and sparse approximate inverse (SAI) preconditioning techniques have been widely used and have been successful in solving many large and sparse matrices. Such techniques, however, may commonly encounter two major issues. Firstly, their accuracy is often insufficient to yield an adequate rate of convergence due to the difficulty in determining parameters or threshold values in constructing preconditioners. In addition, they may confront difficulty when a matrix to be solved is indefinite, that is, when the matrix has both positive and negative eigenvalues. This is because small or zero pivots in indefinite matrices may produce unstable and inaccurate preconditioners.

In response to these issues associated with preconditioning techniques, we have concerned ourselves with improving accuracy and stability of a preconditioner in solving large and sparse matrices.

Insufficient Accuracy of Incomplete LU (ILU) Factorization

The convergence of iterative methods can be accelerated by using incomplete LU (ILU) preconditioners associated with an ILU factorization. ILU preconditioners usually perform quite well in solving symmetric and asymmetric matrices. Most ILU preconditioners, however, have a common issue that the accuracy of ILU factorization may be insufficient to yield an adequate rate of convergence because of the dropped elements during the factorization.

In general, more accurate ILU factorizations are often more efficient as well as more reliable [54, 55]. In this way, we consider two strategies to improve the accuracy of some ILU preconditioners. ILU(0) (incomplete LU factorization with zero fill-in) is one of the simplest ways to define a preconditioner having the form of an incomplete

LU factorization

$$A = LU + E, \tag{1.3}$$

where the L and U matrices have the same nonzero structures as those of the lower and upper triangular parts of A, respectively, and E is the error or residual matrix of the factorization. According to Saad [55], the elements of the error matrix E are exactly those that are dropped during the ILU factorization. In standard ILU factorizations, the dropped elements are discarded. In order to improve the accuracy of the computed ILU(0) preconditioner, we collect the dropped elements during the ILU factorization process by paying little computation and some storage space that can be ignorable in modern computing environments.

ILU(0) preconditioning is rather easy to implement and computationally inexpensive compared to other ILU preconditioners, but it may require many iterations to converge due to its coarse approximation of the original matrix A [54, 55]. It has been noticed that ILU factorization generally yield more accurate factorizations than ILU(0) when numerical dropping strategies are adopted in the processes of ILU factorization. The reason is that the dropping strategies try to minimize E in Equation (1.3) in some sense [19].

Accordingly, ILUT (a dual threshold incomplete LU factorization) [54] can be considered as an alternative of ILU(0). ILUT approach relies on numerical values for dropping elements, and it has a limit on the number of allowed fill-in elements. Also, this approach develops ILU factorization preconditioners with improved accuracy if the dropping tolerance and the fill-in parameters are chosen properly. On the other hand, ILUT with large dropping tolerance and small fill-in does not constitute a reliable approach, and generally it gives low accuracy ILU factorizations [54, 55].

In response to the accuracy issue associated with ILU factorizations, we consider the fact that the size of the error matrix E directly affects the convergence rate of the preconditioned iterative methods [32]. Furthermore, the quality of a preconditioning step is directly related to the size of both $(LU)^{-1}$ and E, that is, a high quality preconditioner must have an error matrix that is small in size [68]. The new preconditioning accuracy enhancement strategies proposed in Chapter 3 were inspired by scrutinizing these facts and exploiting the error values dropped in the factorization to improve the accuracy of the incomplete LU factorizations.

The main idea of the error compensation strategy is to use the error matrix before starting the preconditioned iteration process. After acquiring the LU factorization with a preconditioner, we add the error matrix E to the L and U matrices to obtain an error compensated factorization $\widetilde{L}\widetilde{U}$. The accuracy of using $\widetilde{L}\widetilde{U}$ is increased significantly in some cases, compared with that of using LU directly, such that $||\widetilde{E}(= A - \widetilde{L}\widetilde{U})|| < ||E(= A - LU)||$.

The second preconditioning enhancement strategy (an inner-outer iteration process) uses the error matrix during the preconditioned iteration process. It can be considered as a variant of the LU solver with an error embedded LU approach.

Limitations of Preconditioning Techniques for Indefinite Matrices

Indefinite matrices, which have positive and negative eigenvalues, arise frequently from finite element discretizations of coupled partial differential equations in computational fluid dynamics and other applications. In solving such indefinite matrices, however, there are at least two reasons that make preconditioning techniques difficult [19]. The first reason can be due to small or zero pivots in an indefinite matrix that may lead to unstable and inaccurate factorizations [44]. In addition, small pivots are usually related to small or zero entries on the diagonal of the matrix, so the indefinite matrix with zero diagonal entries may have higher chances of encountering zero pivots if the matrix is also nonsymmetric [54, 68]. Secondly, unstable triangular solutions can be resulted when $||L^{-1}||$ and $||U^{-1}||$ are extremely large while the off-diagonal elements of L and U are reasonably bounded. Such problems are usually caused by very small pivots, but they may sometimes happen without a small pivot [19, 34, 68].

As mentioned above that small pivots are often the origin of cumbersome to computing ILU factorization on an indefinite matrix, a better performance can be expected if small pivots would be replaced with some large values in the matrix. Starting from this idea, we proposed reordering strategies that reorder the rows of the matrix to increase the diagonal dominance rate of the rows.

Many researchers [10, 27, 28, 30, 31] have focused on permuting large entries onto the diagonal of indefinite matrix in the situations of either using direct or iterative solvers. For example, Duff and Koster [30, 31] exploited bipartite matching algorithms and scaling techniques for computing permutations of a sparse matrix. Their reordering and scaling algorithms are effective to the performance of various methods for solving sparse matrices. In experimental studies using Duff and Koster's [31] algorithms, Benzi *et al.* [10] considered matching algorithms such as maximum transversal, maximum product transversal, and the bottleneck transversal with symmetric permutations. However, their preprocessing step in matching is still complicated with several symmetric permutations.

In order to solve indefinite matrices efficiently, we present two diagonal reordering algorithms that are computationally inexpensive but efficient in the aspect of reordering performance for indefinite matrices. To that end, we propose new hybrid reordering strategies that are composed with a combination of two different diagonal reorderings and a nondecreasing degree algorithm presented in Chapter 4.

Sparse Aapproximate Inverses and Indefinite Matrices

Of the many types of preconditioners, sparse approximate inverse (SAI) preconditioners, of which the preconditioning process is just a (sparse) matrix-vector product operation, have recently become popular in solving many application problems due to their potential employment in parallel implementations [9, 16, 19, 18, 37, 38, 41].

And for parallelizable preconditioners, several versions of sparse approximate inverse (SAI) techniques have been developed [13, 16, 20, 21, 37, 41, 69] in the past few years. Unlike ILU-type preconditioners, which need triangular solution procedures at every preconditioning step, SAI-type preconditioners possess a high degree of parallelism in the preconditioner application phase. Thus, SAI-type preconditioners would be an interesting alternative to ILU-type preconditioners [6, 7] that can be run on high performance distributed memory parallel computers.

Sparse inverse preconditioning techniques may succeed in solving certain problems where the ILU factorizations are difficult to handle [18]. It has been noticed that factored sparse approximate inverse (FAPINV), which is a sparse approximate inverse that has a factored form, tends to perform better in convergence rate for the same amount of nonzeros in the preconditioner and also requires less computational cost than a non-factored form does. However, the resulting sparse approximate inverse can still break down for solving indefinite matrices due to zero or small pivots [9, 69].

A shifting strategy, which adds a value to the diagonals of an indefinite matrix, can help the resulting preconditioner better conditioned [9, 51, 64], but determining the value to be added for small pivots is usually critical to the performance of the resulting preconditioner [44]. In other words, selecting a large replacing value may result in a factorization that is stable but less accurate. On the other hand, selecting a small replacing value may result in a factorization that is accurate but unstable. Such a tradeoff of the shifting strategy has been well studied in [68].

As part of our continuous efforts on solving indefinite matrices, we propose to adopt the idea of a shifting strategy [51] to replace small or zero elements on the diagonal of the original matrix, and to reinforce with a two-phase preconditioning process to deal with the tradeoff between stability and accuracy of the resulting preconditioner in Chapter 5. More specifically, the first phase of the construction process employs the shifting strategy to the original matrix so that a shifted matrix can be well conditioned, and then an approximate inverse, M_1, of the shifted matrix is obtained by utilizing FAPINV. In the second phase of the construction process, a temporary matrix, a product of M_1 and the shifted matrix, is considered to acquire a better approximate inverse of the original matrix than M_1. Applying the shifting strategy again to the temporary matrix produces a second shifted matrix, and FAPINV computes a second approximate inverse matrix, M_2, of the second shifted matrix. Then the resulting sparse approximate inverse, M, has the form of $M = M_2M_1$, where M_1 and M_2 are computed in each phase.

Sparse Approximate Inverses and Sparsity Patterns

Computational efficiency and potential parallelism of sparse approximate inverse (SAI)-type preconditioners leads us to choose SAI preconditioners as an alternative to the conventional ILU preconditioners. It has been noticed that the performance of SAI preconditioners depends on their sparsity patterns so that the search for an optimal sparsity pattern to construct the SAI preconditioners should be taken with extreme care [62, 69].

In determining the sparsity pattern of SAI, there exists two main classes of methods called static and dynamic strategies. A static sparsity pattern strategy prescribes the sparsity pattern before constructing a preconditioner, and the pattern remains unchanged until finishing the construction. Although this class of sparsity pattern selection strategy is usually efficient in terms of computational cost due to the use of a prescribed (fixed) pattern through the construction process, for general sparse matrices, the prescribed sparsity pattern is often inadequate for robustness [69]. On the contrary, a dynamic sparsity pattern strategy adjusts the pattern using some rules in the construction phase. Such a strategy usually computes more accurate and more robust preconditioners than a static strategy [62]. Thus, a dynamic sparsity pattern strategy may be used to substitute for a static sparsity pattern strategy in solving difficult matrices.

In recent years, a few dynamic pattern strategies [15, 41] for SAI preconditioners have been developed. For example, Bollhöfer [14] showed that the norms of the inverse triangular factors have direct influence on the dropping strategy in computing a new ILU decomposition. Based on this insight, Bollhöfer [15] proposed an algorithm that manages the process of dropped entries with small absolute values by using the row norm of any row of the inverse factors, but the algorithm has a limitation on solving some ill-conditioned problems.

In Chapter 6, we introduce two enhanced algorithms, which extend the algorithm [15] mentioned above, using combined information of the norm of the inverse factors and either the largest absolute value of the original factors or the norm of the original factors. Due to our enhancement strategy, the presented algorithms may produce better results in solving ill-conditioned problems.

1.2 Organization

This dissertation is composed of seven chapters, notations are introduced as the need arises, and conclusions are given in each chapter separately. The remainder of this dissertation is organized as follows:

- Chapter 2 provides some background on research areas relevant to the rest of the dissertation. Readers with prior knowledge in iterative methods and preconditioning techniques may want to skim this chapter.

- In Chapter 3, several preconditioning enhancement strategies for improving inaccurate preconditioners generated by the incomplete LU factorizations of sparse matrices are presented. The strategies employ the elements that are dropped during the incomplete LU factorization and utilize them in different ways with separate algorithms. Experimental results of enhancement strategies are presented to show accuracy improvement of the incomplete LU factorization in case that the initial factorizations are found to be inaccurate.

- In Chapter 4, we present hybrid strategies for indefinite matrices and new diagonal reorderings that are in conjunction with a symmetric nondecreasing degree algorithm. With the reordered matrices, we have achieved a considerably improved stability of incomplete LU factorizations.

- In Chapter 5, a two-phase preconditioning strategy based on FAPINV is proposed for solving sparse indefinite matrices. The presented strategy improves the accuracy and the stability of the preconditioner in solving indefinite sparse matrices.

- In Chapter 6, we propose two sparsity pattern selection algorithms for factored approximate inverse preconditioners in solving general sparse matrices. The sparsity pattern is adaptively updated in the construction phase by using combined information of the inverse and original triangular factors of the original matrix.

- Chapter 7 summarizes conclusions of this dissertation and outlook for possible future research directions.

Chapter 2 Background

In solving a linear system $Ax = b$, direct methods or iterative methods can be considered. Direct methods are usually used in areas where reliability is the primary concern on account of their delivery of exact solutions. The most widely known direct method is Gaussian elimination, which computes a matrix factorization,

$$A = LU,$$

where L and U are lower and upper triangular, respectively. In order to solve a linear system $Ax = b$, the computed factorization $LUx = b$ can be utilized with forward elimination ($Ly = b$) and backward substitution ($Ux = y$). Gaussian elimination on an $n \times n$ matrix has a computational complexity of $\mathcal{O}(n^3)$ and storage complexity of $\mathcal{O}(n^2)$. However, direct methods become inefficient in terms of operation count and memory requirement as the size of the linear systems increased.

For solving such large and sparse linear systems, iterative methods, which generate a sequence of approximate solutions to the systems, are generally employed due to their more favorable computational efficiency than direct methods. In fact, iterative methods, when implemented properly, may require less storage and fewer computing operations than direct methods (See [55] for more details).

This chapter is to provide background knowledge on research areas relevant to the rest of the dissertation. The most relevant topics are the convergence rate of iterative methods and preconditioning techniques that are discussed in Section 2.1. Of several types of preconditioners in solving large and sparse linear systems, we introduce incomplete LU (ILU) preconditioner (in Section 2.2) and sparse approximate inverse (SAI) preconditioner (in Section 2.3) that will be frequently referred later in the dissertation.

2.1 Preconditioned Iterative Methods

Iterative methods are usually employed in solving large and sparse linear systems because of their computational efficiency. However, they do not have the reliability of direct methods and often fail in some applications to attain convergence in a reasonable amount of time [8, 55]. In order to improve the reliability and convergence rate of iterative methods, preconditioning, which transforms an original linear system into another equivalent system that is easier to solve, is required.

The Convergence of Iterative Methods

Generally speaking, there are two types of iterative methods, stationary iterative methods and Krylov subspace (nonstationary) methods. Stationary iterative methods are old-fashioned and simple to drive, implement, and analyze, but usually not as effective as Krylov subspace methods. Examples of stationary iterative methods are the Jacobi method and the Gauss-Seidel method. Krylov subspace methods are relatively new and can be highly effective, but these methods are harder to drive, implement, and analyze. Prototypes of Krylov subspace methods are the conjugate gradient method (CG), the generalized minimal residual method (GMRES), and the biconjugate gradient method (BiCG) [5].

It has been noticed that the convergence rate of iterative methods are strongly related to the distribution of eigenvalues of the coefficient matrix of a linear system $Ax = b$ [1, 2, 8, 23, 25, 55]. In this subsection, we present a general structure and the convergence rate of iterative methods.

Stationary iterative methods define a sequence of iterates of the form:

$$x_{k+1} = Gx_k + f, \quad k = 0, 1, \cdots, \tag{2.1}$$

where G is a certain iteration matrix, f is a fixed vector, and x_0 is an initial guess. If the iteration converges, then the limit is the solution of the original system $Ax =$

b [55].

In order to determine the convergence rate of iterative methods, we begin with the definition of spectrum and spectral radius.

Definition 2.1.1. *The spectrum of a matrix A, denoted by $\sigma(A)$, is the set of all eigenvalues of A. The spectral radius is the maximum size of the eigenvalues of A and denoted by*

$$\rho(A) = max_{1 \leq i \leq k}(|\lambda_i|),$$

where $\lambda_1, \cdots, \lambda_k$ are the eigenvalues of A.

Theorem 2.1.2 [55] shows that the convergence rate of an iterative method strongly depends on the spectrum of the coefficient matrix A.

Theorem 2.1.2. *Let G be a square nonsingular matrix such that $\rho(G) < 1$. Then $I - G$ is nonsingular and the iteration (2.1) converges for any f and x_0. Conversely, if the iteration (2.1) converges for any f and x_0, then $\rho(G) < 1$.*

From the results of Theorem 2.1.2, we note that the spectrum value of the iteration matrix G is less than 1 is a necessary and sufficient condition for the iteration to converge.

The rate of convergence of Krylov subspace methods is also strongly related to the distribution of the eigenvalues of A [8, 36, 55]. For example, the convergence rate of the conjugate gradient (CG) method depends on the distribution of the eigenvalue of A, that is, for a symmetric positive definite matrix A, if the matrix has a smaller spectral condition number and/or eigenvalues clustered around 1, the CG method usually results in a rapid convergence. This can be seen from the following result [36].

Theorem 2.1.3. *Suppose $A \in \mathcal{R}^{n \times n}$ is symmetric positive definite and $b \in \mathcal{R}^n$. If CG method produces iterates $\{x_k\}$ and $\kappa = \lambda_n/\lambda_1$, where λ_n and λ_1 are the largest* *and smallest eigenvalues of the matrix A, respectively. Then*

$$||x_k - x||_A \leq 2||x - x_0||_A(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1})^k.$$

Here, the norm $||\cdot||_A$ is the operator norm with respect to the symmetric positive definite matrix A. For two vectors ($x, y \in \mathcal{R}^n$), the inner product with respect to A is defined as

$$(x, y)_A = y^T Ax.$$

The operator norm with respect to A is defined as $||x||_A = \sqrt{(x,x)_A}$.

In general, Krylov subspace methods often produce rapid convergence when a matrix A has a clustered spectrum (away from 0) [1, 8, 55].

Preconditioned Iterations

Although computational efficiency of iterative methods induces their wide usage in solving large and sparse linear systems, they potentially have an issue of slow convergence for linear systems that arise from many applications including fluid dynamics and electronic device simulation [55]. In these applications, preconditioning plays an important role for the success of iterative methods because it can improve both the efficiency and robustness of iterative methods [8, 36, 55].

Preconditioning transforms an original linear system $Ax = b$ into another system, which has the same solution of the original system, with more favorable eigenspectrum properties for an iterative solver. Generally speaking, preconditioning aims at improving spectral properties of the coefficient matrix. The transformed matrix is called a *preconditioned matrix* that expectantly will have a smaller spectral condition number and/or eigenvalues clustered around 1 [8].

There are three ways to apply preconditioning techniques. If M is a nonsingular matrix and approximates A, then the linear system of the form:

$$M^{-1}Ax = M^{-1}b \tag{2.2}$$

has same solution as $Ax = b$. Here, M is a *preconditioner*, and Equation (2.2) is preconditioned from the left. Also, the preconditioner can be applied to the right:

$$AM^{-1}u = b, \quad x = M^{-1}u.$$

In addition, split preconditioning of the form:

$$M_L^{-1}AM_R^{-1}u = b, \quad x = M_R^{-1}u,$$

is also possible when the preconditioner is available in the factored form $M = M_L M_R$, where M_L and M_R are typically triangular matrices [55].

In order for a preconditioner to be efficient, the construction cost of the preconditioning matrix M should be low, and the computation cost of the inverse of M or the solution with the matrix M should be inexpensive. That is, a good preconditioner, M, should be easy to solve with and also cheap to construct and apply [8, 55]. In the next two sections, we continue discussing two of the most widely used preconditioning techniques, the incomplete factorization methods and sparse approximate inverses.

The accuracy and stability of preconditioners

The conditioning of a matrix affects the accuracy of the computed solution. A matrix is said to be ill-conditioned if a small change in the data causes a large change in the solution, otherwise it is well-conditioned. Ill-conditioning or well- conditioning of a matrix problem is generally measured by means of a number called the condition number. The condition number, κ, is

$$\kappa = ||A||||A^{-1}||.$$

If the condition number of a preconditioner, M^{-1}, is large, computations with the matrix M^{-1} is likely to be inaccurate. The reason is that with a large condition number, even a small error in the right-hand side of a linear equation may cause a large error in the solution of the equation. In that case, the preconditioned matrix,

$M^{-1}A$, might be worse conditioned than the original matrix A, and also the preconditioned GMRES method might need more iterations to converge or not converge at all. However, convergence might happen even if the actual residual norm is still large when the initial residual norm is large. Such convergence is called a false convergence.

In order to determine a false convergence from a true convergence, we use some statistical information to estimate condition numbers of the preconditioned matrices. A statistic "condest", suggested by Chow and Saad [19], is utilized to estimate the condition number by calculating $||(LU)^{-1}e||_{\infty}$, where e is the vector of all ones. This statistics indicates a relation between the unstable triangular solutions and the poorly conditioned L and U factors.

2.2 Incomplete Factorization Preconditioners

One of the widely used preconditioners is based on incomplete factorization of the coefficient matrix A. Incompleteness can be generated during the factorization process, that is, certain fill-in elements, which are nonzero in the factorization at positions where the original matrix had a zero, have been discarded. This type of preconditioner produces a decomposition of the form:

$$A = LU - R,$$

where L and U are lower and upper triangular matrices, respectively, and R is the residual or error of the decomposition. Here, the preconditioner is given in a factored form $M = LU$, and the efficacy of the preconditioner depends on how well M^{-1} approximates A^{-1}.

Now, we describe the ILU(0) factorization, the simplest form of incomplete LU (ILU) preconditioners and ILUT that is more accurate and the most widely used ILU preconditioner.

ILU(0)

Incomplete LU factorization takes a set P of the matrix entry positions to be precisely the nonzero pattern of the matrix A and keeps all positions outside this set equal to zero during the factorization. The set P is generally chosen to enclose all nonzero positions of A, and a position is called a *fill-in* position when the position is zero in A but is not zero in an exact factorization.

ILU(0) factorization is an incomplete LU factorization with no fill-in position. That is, ILU(0) factorization takes a set $P = NZ(A)$ and entails a preconditioner $M = LU$, where L and U have the same nonzero structure as the lower and upper parts of A, respectively, and $NZ(A)$ is the nonzero positions in A. An example of the ILU(0) factorization for a five-point matrix is shown in Figure 2.1 [55]. The procedure of the ILU(0) factorization is described in Algorithm 2.2.1 [55].

Algorithm 2.2.1. *The ILU(0) algorithm.*

```
1.  For $i = 2, \ldots, n$
2.      For $k = 1, \ldots, i-1$
3.        If $(i,k) \in NZ(A)$, then
4.            Compute $a_{ik} = a_{ik}/a_{kk}$
5.        For $j = k+1, \ldots, n$
6.           If $(i,j) \in NZ(A)$, then
7.               Compute $a_{ij} = a_{ij} - a_{ik}a_{kj}$
```

ILU(0) factorization is rather easy to implement and inexpensive to compute, and it is effective for certain matrices from low-order discretizations of scalar elliptic PDEs and diagonally dominant matrices. On the other hand, ILU(0) factorization often produces a rough approximation to the original matrix. For difficult problems, the accuracy of the ILU(0) factorization may be insufficient to yield an adequate rate

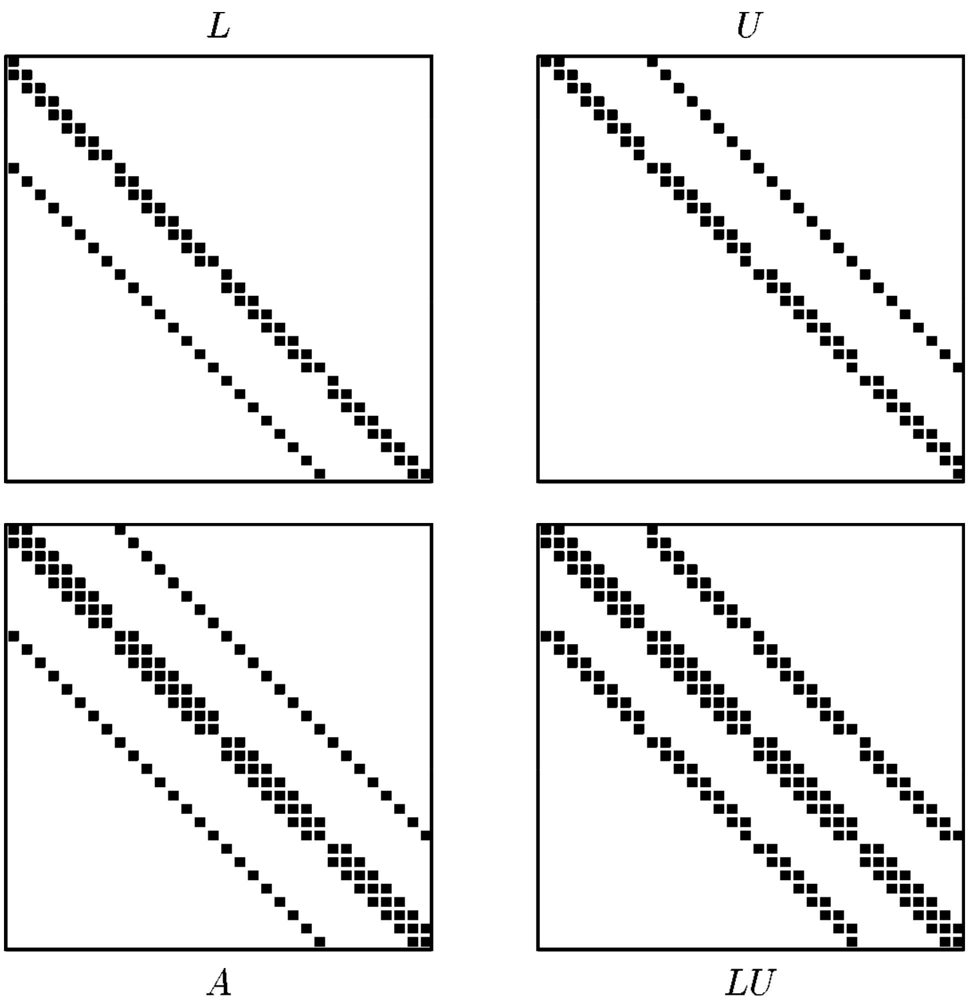


Figure 2.1: The ILU(0) factorization for a five-point matrix with five nonzero diagonals.

of convergence [8, 54, 55]. In order to obtain better approximation, several alternative incomplete factorizations have been developed by allowing more fill-in in L and U.

ILUT: A Dual Threshold ILU Factorization

In the process of incomplete factorization, dropped elements are usually determined by the structure of the matrix A rather than the numerical values of the matrix. This often derives some difficulties in solving for realistic problems. In order to remedy this, a few alternative methods are available based on dropping elements in the Gaussian elimination process according to their magnitude instead of their locations. With these techniques, ILUT (Incomplete LU (ILU) factorization with a dual threshold) factorization, which selects the dropped elements dynamically, is one of widely employed alternative methods. This preconditioner utilizes two-step dropping strategy to control fill-in during its construction phase.

The dropping strategy in ILUT works as the follows: (1) The threshold in L and U is set by tol. Any element whose magnitude is less than tol (relative to the absolute value of the diagonal element of the current row) is dropped. (2) Keeping only the largest $lfil$ elements in the ith row of L and the largest $lfil$ element in the ith row of U (excluding the diagonal elements) [54, 55]. The ILUT algorithm is presented in Algorithm 2.2.2 [55]. Note that a_{i*} denotes the ith row of A.

Algorithm 2.2.2. THE ILUT ALGORITHM.

```
1.  For $i = 2, \ldots, n$
2.      $w = a_{i*}$
3.      For $k = 1, \ldots, i-1$
4.          If $w_k \neq 0$, then
5.              $w_k = w_k / a_{kk}$
6.              Apply a dropping rule to $w_k$
7.              If $w_k \neq 0$, then
8.                  $w = w - w_k * u_{k*}$
9.      Apply a dropping rule to row $w$
10.     For $j = 1, \ldots, i-1$
11.         $l_{i,j} = w_j$
12.     For $j = i, \ldots, n$
13.         $u_{i,j} = w_j$
14.     $w = 0$
```

ILUT preconditioner has been one of efficient and robust preconditioners for iterative solvers, and the construction cost of ILUT is not expensive [55, 70]. In addition, the ILUT preconditioner is efficient when the two dropping parameters tol and $lfil$ are chosen carefully, and the total storage of L and U is bounded by $2n * lfil$ [55, 70]. In fact, $lfil$ can be viewed as a parameter that helps control memory usage, while tol helps reduce computational cost.

2.3 Sparse Approximate Inverses

Of the many types of preconditioners, recently sparse approximate inverse (SAI) preconditioners, of which the preconditioning process is just a (sparse) matrix-vector product operation, have become popular in solving many application problems due to their potential in parallel implementations [9, 16, 19, 18, 37, 38, 41]. Unlike incomplete LU (ILU)-type preconditioners, which need triangular solution procedures that is difficult to perform on parallel computers, SAI-type preconditioners possess high degree of parallelism in the preconditioner application phase. Thus, SAI-type preconditioners are interesting alternative of ILU-type preconditioners on high performance distributed memory parallel computers [6, 7].

For parallelizable preconditioners, several versions of sparse approximate inverse (SAI) techniques have been developed [13, 16, 20, 21, 37, 41, 69] in the past few years. In general, there can be three categories to construct SAI techniques; sparse approximate inverses based on Frobenius norm minimization [20, 21, 22, 37, 41], sparse approximate inverses computed from an ILU factorization [55], and factored sparse approximate inverses [13, 45]. Each of these categories contains its own advantages and limitations in the aspects of construction cost, application cost, robustness, and efficiency [12, 20, 37, 59].

Sparse Approximate Inverses based on Frobenius Norm Minimization

Sparse approximate inverses [20, 21, 22, 37, 41] based on Frobenius norm minimization is the most popular and simple approach for finding approximate inverses. The basic idea of this approach is to compute a sparse matrix $M \approx A^{-1}$ that minimizes the Frobenius norm of the residual matrix $I - AM$:

$$F(M) = ||I - AM||_F^2, \tag{2.3}$$

where $||\cdot||_F$ denotes the Frobenius norm[1] of a matrix. The minimization problem (2.3) can be decoupled into the sum of the squares of the 2-norms of the individual columns of the residual matrix as:

$$||I - AM||_F^2 = \sum_{j=1}^{n} ||e_j - Am_j||_2^2,$$

where e_j and m_j denote the jth columns of the identity matrix and of the matrix M, respectively. It follows that the computation of M is equivalent to minimizing the individual functions

$$f_j(m) = ||e_j - Am_j||_2^2,$$

for $j = 1, 2, \cdots, n$. In other words, each column of M can be computed independently so that the computation of M can be carried out in parallel. However, the construction of the sparse approximate inverses based on norm minimization is sometimes very expensive, and parallel implementations are necessary for them to be practically useful [4, 37].

Sparse Approximate Inverses Computed from an ILU Factorization

According to Benzi and Tuma [12], sparse approximate inverses computed from an ILU factorization has a two-phase process, factorization phase and inverting phase. In other words, an incomplete LU (ILU) factorization $A \approx \tilde{L}\tilde{U}$ is computed firstly by

[1]Frobenius norm is defined for a matrix $A = (a_{ij})_{n\times n}$ as $||A||_F = \sqrt{\sum_{i,j=1}^{n} {a_{ij}}^2}$.

using standard techniques, and then incomplete factors $\tilde{L}$ and $\tilde{U}$ are approximately inverted. This approach can produce the computation of a preconditioner with two phases of incompleteness. As a result, these methods may be difficult to use in practice [12, 69].

Factored Sparse Approximate Inverses

In this subsection, we discuss factored sparse approximate inverses based on the incomplete inverse factorizations of A^{-1} [12, 69]. Factored sparse approximate inverse can be constructed by computing sparse approximations $\tilde{L} \approx L$ and $\tilde{U} \approx U$ when A^{-1} can be factorized $A^{-1} = LDU$, where L is unit lower triangular, D is diagonal, and U is unit upper triangular. The factored approximate inverse is then

$$M = \tilde{L}\tilde{D}\tilde{U}^T \approx A^{-1},$$

where $\tilde{D}$ is a nonsingular diagonal matrix, $\tilde{D} \approx D$. The weakness of this approach is the lack of parallelism in the construction phase, but factored sparse approximate inverses have some advantages over the two approaches previously introduced. The advantages are that (1) factored sparse approximate inverses often result in better convergence rates with the same storage requirement, and (2) factored forms are less expensive to compute [12, 69].

Several researchers [11, 69] considered algorithms to improve the parallelism of factored approximate inverses so that one can use the advantages of the factored approximate inverses over the other sparse approximate approaches. Zhang [69] proposed an factored inverse algorithm that is derived from a matrix decomposition method for inverting nonsymmetric matrices. In this dissertation, we utilized the factored sparse inverses based on an algorithm from [69] that possesses greater inherent parallelism. The factored inverse algorithm [69] is presented in Algorithm 2.3.1, and the algorithm returns $M = LDU \approx A^{-1}$.

Algorithm 2.3.1. Factored Inverse Algorithm for Sparse Matrices [69].

```
1.  For $j = n \to 1$ with step (-1)
2.      For $i = j+1, \to n$
3.          $w_i = A_{ji} + \sum_{k=i+1}^{n} A_{jk} * L_{ki}$
4.      For $i = j+1, \to n$
5.          $w_i = w_i * D_{ii}$
6.          $v_i = -w_i$
7.      For $i = j+1, \to n$
8.          For $k = i+1, \to n$
9.              $v_k = v_k - w_k * U_{ik}$
10.     For $i = j+1, \to n$
11.             $U_{ji} = v_i$
12.     $D_{jj} = 1/(A_{jj} + \sum_{k=j+1}^{n} U_{jk} * A_{kj})$
13.     For $i = j+1, n$
14.         $w_i^{(j)} = A_{ij} + \sum_{k=i+1}^{n} U_{ik} * A_{kj}$
15.     For $i = j+1, n$
16.         $w_i = w_i * D_{ii}$
17.         $v_i = -w_i$
18.     For $i = j+1, \to n$
19.         For $k = i+1, \to n$
20.             $v_k = v_k - w_k * L_{ki}$
21.     For $i = j+1, \to n$
22.         $L_{ij} = v_i$
```

Note that $L = [L_1, L_2, \cdots, L_n]$ denotes a lower triangular matrix of order n, L_{ij} represents an (i, j) element in L, and L_i is a column vector of length n with $L_{ii} = 1, i =$

$1, 2, \cdots, n$. Similarly, $U = [U_1, U_2, \cdots, U_n]$ represents an upper triangular matrix of order n, U_{ij} is an (i, j) element in U, and U_i denotes row vectors of length n with $U_{ii} = 1, i = 1, 2, \cdots, n$. A diagonal matrix is denoted by $D = \text{diag}[D_{11}, D_{22}, \cdots, D_{nn}]$.

Chapter 3 Incomplete LU Preconditioning Enhancement Strategies

In this chapter, we discuss the inaccuracy problems associated with incomplete LU (ILU) factorization. In order to improve inaccurate preconditioners produced by ILU factorizations, two preconditioning enhancement strategies are presented. The strategies employ the elements that are dropped during the ILU factorization and utilize them in different ways by separate algorithms. The first strategy (error compensation) applies the dropped elements to the lower and upper triangular parts of the ILU factorization to compute a new error compensated LU factorization. The other strategy (inner-outer iteration), which is a variant of the ILU factorization, embeds the dropped elements in its iteration process.

This chapter is organized as follows: In Section 3.2, preconditioning enhancement strategies arising from ILU preconditioning and their analyses are presented. The experimental results are in Section 3.3, and the concluding remarks are discussed in Section 3.4.

3.1 Introduction

Incomplete LU (ILU) factorization, which is constructed from the incomplete lower-upper factorization, is one of the most popularly used preconditioners. However, a common problem with such ILU preconditioners is that their accuracy may be insufficient to yield an adequate rate of convergence. This problem is usually caused by the difficulty in determining even a small number of parameters or threshold values for the ILU factorizations for some particular matrices. In addition, there still remains as a tradeoff problem between the factorization accuracy and the costs of the computation and memory associated with the factorization.

In the form of an ILU factorization by incorporating the error or residual matrix,

E, we have

$$A = LU + E, \tag{3.1}$$

where L and U are the lower and upper parts of A, respectively. The elements of the error matrix E are just discarded in standard ILU factorizations. It has been known that the size of the error matrix E directly affects the convergence rate of the preconditioned iterative methods [32, 55]. Moreover, the quality of a preconditioning step is directly related to the size of both $(LU)^{-1}$ and E, that is, a high quality preconditioner must have an error matrix that is small in size [31, 68].

Based on the understanding of these facts, we propose two new preconditioning accuracy enhancement strategies that exploit the error values dropped in the factorization to improve the accuracy of ILU factorizations. This then leads us to the idea of the error compensation strategy that utilizes the error matrix before the start of the preconditioned iteration process. After acquiring an ILU factorization with a preconditioner, we add the error matrix E to L and U to obtain an error compensated factorization $\widetilde{L}\widetilde{U}$. The accuracy of using $\widetilde{L}\widetilde{U}$ is increased significantly in some cases, compared with that of using LU directly, in the sense that $||\widetilde{E}(= A - \widetilde{L}\widetilde{U})|| < ||E(= A - LU)||$.

The other preconditioning accuracy enhancement strategy uses the error matrix during the preconditioned iteration process. This can be considered as a variant of the LU solver with an error embedded LU approach and results in an inner-outer iteration process. Numerical results of the inner-outer iteration strategies are provided to show that this algorithm requires fewer outer iterations to converge. In other words, a few inner iteration at each preconditioning step reduces the number of outer iterations of the preconditioned iterative solver. Note that in order to collect the elements during the ILU factorization process, additional computation cost and some memory spaces are required to keep these elements. This, however, can be negligible as low cost storages become available.

3.2 Preconditioning Accuracy Enhancement Strategies

ILU factorizations are made incomplete by dropping carefully chosen nonzero elements to make the factorizations more economical to store, compute, and solve with it. Each nonzero element that is dropped contributes to the "error" in the factorization [19]. In this section, we assume that ILU factorizations in question are stable, and propose an error compensation algorithm and an inner-outer iteration algorithm to improve the accuracy of the (inaccurate) ILU factorizations.

Matrix Error Compensation Strategy

We propose an algorithm to improve the preconditioning accuracy by reducing the size of the error matrix E and $(LU)^{-1}E$ in Equation (3.1). By switching the matrix A and LU, Equation (3.1) can be written as:

$$E = A - LU. \tag{3.2}$$

Then the error matrix E can be separated into two parts, corresponding to the nonzero structures of L and U, in the following way:

$$E = E_l + E_u,$$

where E_l and E_u are the strictly lower and upper triangular parts of E, respectively. In order to acquire error compensated $\widetilde{L}$ and $\widetilde{U}$, the separated error matrices, E_l and E_u, are now combined with the original lower part (L) and upper part (U) of the preconditioned matrix:

$$\widetilde{L} = L + E_l, \qquad \widetilde{U} = U + E_u.$$

These new lower triangular part $\widetilde{L}$ and upper triangular part $\widetilde{U}$ are then used as the preconditioner instead of the original L and U. In other words, $\widetilde{L}$ and $\widetilde{U}$ are used in the preconditioning steps, i.e., we solve $(\widetilde{L}\widetilde{U})e = r$ instead of $(LU)e = r$ at each preconditioning step. Here e and r are the error and residual vectors of the current approximate solution. In the experimental results presented in Section 3.3, we will see that replacing LU solver with the new $\widetilde{L}\widetilde{U}$ results in a more accurate preconditioner. In many cases, the size of the error matrix with the new $\widetilde{L}\widetilde{U}$ is smaller than that with the original LU.

In order to provide a better picture of the algorithm, an ILU factorization in which the error compensated L and U improves the accuracy of the original factorization is given in the following: The ILU(0) factors L and U of a given matrix

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix}$$

are

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & 0 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1.5 & 0 \\ 0 & 0 & 1.5 \end{pmatrix},$$

from which we can compute the error matrix $E = A - LU$ as

$$E = A - LU = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -0.5 \\ 0 & -0.5 & 0 \end{pmatrix}.$$

The error matrix E is then split into two parts, E_l and E_u. We add E_l and E_u to the original incomplete lower and upper triangular parts of A. This results in a new incomplete LU factors

$$\widetilde{L} = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & -0.5 & 1 \end{pmatrix}, \qquad \widetilde{U} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1.5 & -0.5 \\ 0 & 0 & 1.5 \end{pmatrix}.$$

From the above A, $\widetilde{L}$, and $\widetilde{U}$, we can obtain an new error matrix $\widetilde{E}$ as:

$$\widetilde{E} = A - \widetilde{L}\widetilde{U} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0.25 & -0.25 \end{pmatrix}.$$

Table 3.1 presents the comparisons of the sizes of the error matrices in the example given above. The sizes of the preconditioned error matrix for the error compensated preconditioner and the original preconditioned matrix are measured in the 2-norm and the Frobenius norm. The case using error compensated preconditioned matrix reduces the error norm by 36% on average. The error compensation algorithm can

Table 3.1: Comparison of the size of preconditioned error matrices E and $\widetilde{E}$.

Norm	E	$\widetilde{E}$	$(LU)^{-1}E$	$(\widetilde{L}\widetilde{U})^{-1}\widetilde{E}$
2-norm	0.5	0.3536	0.4082	0.2940
Frobenius norm	0.7071	0.3536	0.5270	0.2940

have four different implementations. The above description assumes that both the L and U factors are compensated. If none of them is compensated, we have the standard ILU preconditioner. If either one of them, but not both factors, is compensated, we have two forms of partial error compensations.

An Inner-Outer Iteration Process

In this subsection, we describe an error embedded variant of ILU preconditioning process to compensate dropped elements during the ILU factorization. In solving a linear equation with a standard preconditioned iterative solver, the preconditioning step is an LU solution process with an ILU factorization type preconditioner, i.e., to solve $(LU)e = r$ for the residual vector r of the current approximate solution and to get an approximate correction vector e. For notational convenience, we denote the current approximate solution as $\tilde{x}$. The current residual vector is $\tilde{r} = b - A\tilde{x}$. The ideal preconditioning step is to solve $A\tilde{e} = \tilde{r}$ and to correct the approximate solution

$\tilde{x}$ to get the exact solution as $(\tilde{x} + \tilde{e})$. Since solving $A\tilde{e} = \tilde{r}$ is as expensive as solving the original system $Ax = b$ and is thus impractical in a preconditioning step, we can again use an iteration procedure to approximately solve $A\tilde{e} = \tilde{r}$ to achieve a good preconditioning effect. If the error matrix is available, we can split the matrix A as in Equation (3.1), so, for an idealized preconditioning step, we have

$$(LU + E)e = r, \tag{3.3}$$

or

$$LUe + Ee = r. \tag{3.4}$$

As a stationary iteration process, Equations (3.3) and (3.4) can be rewritten by:

$$LUe^{(k+1)} = r - Ee^{(k)},$$

and

$$e^{(k+1)} = (LU)^{-1}(r - Ee^{(k)}) \tag{3.5}$$

respectively. This iteration process starts with $e^{(0)} = 0$, and $LUe^{(1)} = r$ is just the standard one step ILU preconditioning procedure. Better preconditioning effect can be achieved by performing a few iterations at each preconditioning step. This will constitute an inner-outer iteration scheme and is well-known for the preconditioned Krylov method, as each outer Krylov iteration step is preconditioned by a series of inner stationary iteration steps.

The necessary and sufficient condition to converge for the inner iteration process, given by Equation (3.5), is that the spectral radius of the iteration matrix $(LU)^{-1}E$ is smaller than 1. This is, of course, not automatically guaranteed. However, we found that the inner iteration process usually converges, when the ILU factorization of the original matrix A is stable and the LU factor is not an extremely poor approximation to the original matrix A. We point out that the inner-outer iteration strategy is not a new one, but here we study it in a systematic way.

Analysis

In order to validate the presented algorithms, the comparison of the size of the error matrices of a five-point matrix is given in Table 3.2. A five-point matrix (of a finite difference discretization of a Poisson equation), A, of size $n = 400$ corresponding to an $n_x \times n_y = 20 \times 20$ mesh is used. The sizes of the error matrix, E, from the original preconditioner and the new error matrix, $\widetilde{E}$, from the error compensated preconditioner are measured in the 2-norm and the Frobenius norm. For the case

Table 3.2: Comparison of the size of preconditioned error matrices E and $\widetilde{E}$.

Norm	E	$\widetilde{E}$	$(LU)^{-1}E$	$(\widetilde{L}\widetilde{U})^{-1}\widetilde{E}$	$(\widetilde{L}U)^{-1}\widetilde{E}$	$(L\widetilde{U})^{-1}\widetilde{E}$
2-norm	0.5788	0.3582	0.9276	0.8889	0.9109	0.9106
Frobenius norm	7.7958	3.2058	3.6129	2.4812	3.1488	3.1447
Spectral radius			0.9276	0.8885	0.9098	0.9098

using the error compensated algorithm, the norms (2-norm and Frobenius norm) in columns 2 and 3 in Table 3.2 show the fact that the norms of the new error matrix are smaller than the norms of the original error matrix, and also the norms (2-norm and Frobenius) in columns from 4 to 7 show that all the norms (2-norm and Frobenius) of the corresponding matrices in columns from 5 to 7 are smaller than the norms in column 4. In addition to that, from the values in the last row of Table 3.2, we can see that the inner iteration process converges, since the spectral radii of the $(LU)^{-1}E$ and $(\widetilde{L}\widetilde{U})^{-1}\widetilde{E}$ satisfy the necessary and sufficient condition for convergence.

3.3 Experimental Results

We present numerical results from the experiments by solving sparse matrices with our algorithms and some preconditioners. The preconditioned iterative solver we employed was GMRES(20), with ILU type preconditioners, ILU(0) and ILUT [54]. For all linear systems, the right-hand side was generated by assuming that the solution is a vector of all ones. The initial guess was a zero vector. We set the iteration to be terminated when the l_2-norm of the initial residual is reduced by at least seven orders of magnitude, or when the number of iterations reaches 200. The computations were carried out in double precision arithmetics on a 64-bit Sun-Blade-100 workstation with a 500 MHz UltraSPARC III CPU and 1 GB of RAM. The set of sparse matrices

Table 3.3: Description of the test matrices.

Matrix	Description	n	nnz	$cond$
ADD32	Computer component design 32-bit adder	4960	23884	2.1E+02
BFW398A	Bounded Finline Dielectric Waveguide	398	3678	7.6E+03
CDDE1	Model 2-D Convection Diffusion Operator p1=1, p2=2, p3=30	961	4681	4.1E+03
CDDE3	Model 2-D Convection Diffusion Operator p1=1, p2=2, p3=80	961	4681	1.0E+04
HOR 131	Flow network problem	434	4710	1.3E+05
JPWH991	Circuit physics modeling	991	6027	7.3E+02
LOP163	LOPSI Stochastic Test Matrix	163	935	3.4E+07
MHD3200B	Alfven Spectra in Magneto hydrodynamics	3200	18316	2.0E+13
MHD416B	Alfven Spectra in Magneto hydrodynamics	416	2312	5.1E+09
MHD4800B	Alfven Spectra in Magneto hydrodynamics	4800	27520	1.0E+14
ORSIRR1	Oil reservoir simulation - generated problems	1030	6858	1.0E+02
ORSIRR2	Oil reservoir simulation - generated problems	886	5970	1.7E+05
ORSREG1	Oil reservoir simulation - generated problems	2205	14133	1.0E+02
PDE225	Partial Differential Equation	225	1065	1.0E+02
PDE2961	Partial Differential Equation	2961	14585	9.5E+02
PDE900	Partial Differential Equation	900	4380	2.9E+02
PORES2	Reservoir modeling Reservoir simulation	1224	9613	3.3E+08
PORES3	Reservoir modeling Reservoir simulation	532	3474	6.6E+05
SAYLR1	Saylor's petroleum engineering/reservoir simulation matrices	238	1128	1.6E+09
SAYLR4	Saylor's petroleum engineering/reservoir simulation matrices	3564	22316	1.0E+02
SHERMAN1	Oil reservoir simulation challenge matrices	1000	3750	2.3E+04
SHERMAN4	Oil reservoir simulation challenge matrices	1104	3786	7.2E+03
SHERMAN5	Oil reservoir simulation challenge matrices	3312	20793	3.9E+05
WATT1	Petroleum engineering	1856	11360	5.4E+09

that we used for our experiments are from the Harwell-Boeing Sparse Matrix Test Collection [29]. A brief description of each matrix of the experiment set is given in Table 3.3. In the table, the order, the number of nonzero entries, and the condition number of the matrix are denoted by n, nnz, and $cond$, respectively. All tested matrices do not have zero diagonals and the ILU factorizations are stable.

Preconditioning with Error Compensation Strategies

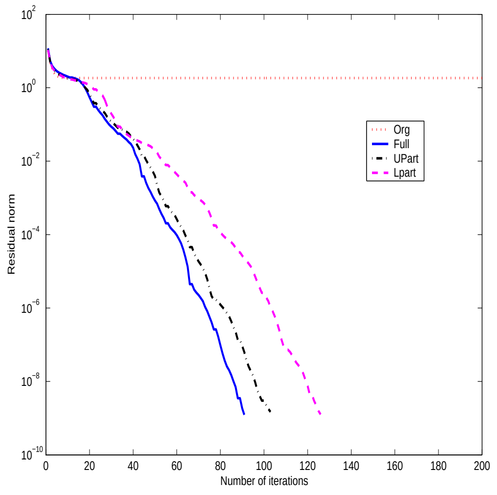
Numerical experiments of testing the error compensation algorithm with different implementations are presented. ILU(0) and ILUT preconditioners are used. In ILUT, we chose the dropping tolerance and fill-in parameter to be 0.1 and 5, respectively, for all test problems.

Table 3.4 reports the number of preconditioned GMRES (PGMRES) iterations with and without different error compensation strategies. The data in the columns marked "No" are those of the standard preconditioning strategy, i.e., no error compensation strategy was used. The columns marked "Full" indicate that both the L and the U parts were compensated. Similarly, the columns marked "Upart" means that only the U part was compensated, and the columns marked "Lpart" means that only the L part was compensated. The value "-1" in the table indicates the failure of convergence within the maximum number (200) of allowed iterations.

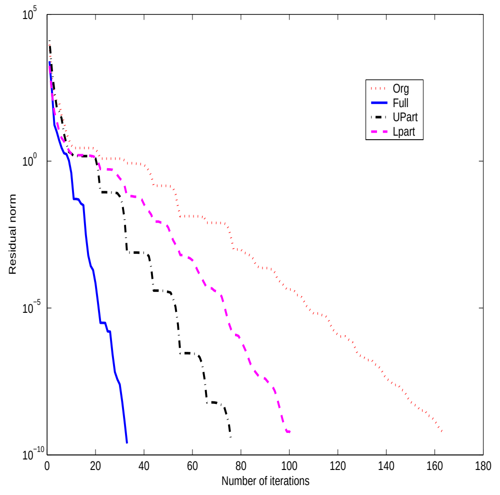
Table 3.4: Comparison of the number of PGMRES iterations with and without different error compensation strategies.

	ILU(0)				ILUT			
Matrix	No	Full	Upart	Lpart	No	Full	Upart	Lpart
ADD32	81	39	40	40	16	15	-1	15
BFW398A	-1	150	-1	196	-1	-1	-1	-1
HOR131	-1	183	-1	-1	-1	-1	-1	-1
JPWH991	29	20	24	24	32	21	25	24
LOP163	-1	88	98	176	18	19	18	19
MHD3200B	5	4	4	4	164	33	76	101
MHD416B	4	4	4	4	137	32	63	77
MHD4800B	4	3	4	3	148	33	76	107
PDE225	30	16	21	22	16	11	15	13
PDE2961	-1	91	103	126	85	66	74	77
PDE900	61	46	53	55	33	22	30	27
SHERMAN1	116	62	76	75	107	87	-1	94
SHERMAN4	143	107	124	128	136	101	117	116
SHERMAN5	101	73	77	98	-1	145	-1	169
WATT1	178	114	115	95	108	85	94	121

Each column of Table 3.4 shows the number of PGMRES iterations. The number of (preconditioned) iterations, especially for solving the matrices ADD32, BFW398A, HOR131, LOP163, PDE2961, SHERMAN1, SHERMAN5, and WATT1, are greatly reduced with the error compensation strategies in the ILU(0) preconditioner, compared with the standard preconditioner ILU(0). In addition, the number of iterations for solving the matrices MHD and SHERMAN5 are reduced with the error compensation strategies in ILUT, compared with the standard preconditioner ILUT.



(a) PDE2961 Matrix with ILU(0)



(b) MHD3200B Matrix with ILUT

Figure 3.1: The convergence curves for the PDE2961 and MHD3200B matrices.

We can see that for most of the test matrices, the use of the error compensation strategies reduces the number of PGMRES iterations. Furthermore, the full error compensation strategy is shown to be more robust than both the L part and the U part partial error compensation strategies. For the two partial error compensation strategies, their performance seems to be comparable. In order to provide better understanding of this, we present Figure 3.1 that shows the PGMRES convergence behavior for PDE2961 and MHD3200B matrices. This represents comparisons of the number of PGMRES iterations of PDE2961 and MHD3200B matrices with ILU(0) and ILUT, respectively, along with and without the error compensation strategies.

Table 3.5 presents the total CPU processing time which includes the precondi-

Table 3.5: Comparison of CPU time in seconds with the different error compensation strategies.

	ILU(0)				ILUT			
Matrix	No	Full	Upart	Lpart	No	Full	Upart	Lpart
ADD32	1.1E+00	1.0E+00	9.9E-01	9.8E-01	1.2E+00	1.2E+00	N/A	1.1E+00
BFW398A	N/A	1.9E-01	N/A	1.9E-01	N/A	N/A	N/A	N/A
HOR131	N/A	2.1E-01	N/A	N/A	N/A	N/A	N/A	N/A
JPWH991	7.0E-02	8.9E-02	7.9E-02	7.9E-02	1.1E-01	1.5E-01	1.3E-01	1.3E-01
LOP163	N/A	2.9E-02	3.9E-02	5.0E-02	6.0E-02	2.9E-02	4.9E-02	5.0E-02
MHD3200B	1.8E-01	2.0E-01	2.2E-01	2.1E-01	8.8E-01	4.0E-01	5.9E-01	6.9E-01
PDE225	1.9E-02	2.9E-02	1.9E-02	3.9E-02	1.9E-02	2.9E-02	1.9E-02	4.9E-02
PDE2961	N/A	5.6E-01	5.8E-01	7.1E-01	1.2E+00	1.4E+00	1.3E+00	1.4E+00
SHERMAN1	1.5E-01	1.1E-01	1.1E-01	1.2E-01	2.4E-01	1.9E-01	N/A	2.1E-01
WATT1	5.1E-01	4.9E-01	4.3E-01	3.9E-01	7.7E-01	7.7E-01	6.9E-01	7.4E-01

tioner construction time with different error compensation strategies. The listed results in Table 3.5 are concerned with the results listed in Table 3.4. "N/A" indicates the failure of computation since the number of GMRES iteration reached the maximum number (200). The total CPU time for solving the matrices (ADD32, BFW398A, HOR131, LOP163, MHD3200B, PDE2961, SHERMAN1, and WATT1) is reduced by using the error compensation strategies with ILU(0)/ILUT. This implies that as the number of overall iterations are greatly decreased, the CPU time is also reduced even though we need extra processing time for implementing the error compensation strategies.

Preconditioning with Inner-Outer Iteration Strategies

In this subsection, we present numerical experiments of the inner-outer iteration strategy with different settings of the number of inner iterations. Table 3.6 reports the number of preconditioned GMRES (PGMRES) iterations with respect to each setting. The data in the columns marked "1-Its" are those of the standard preconditioning strategy, i.e., no additional iteration beyond the standard preconditioning procedure was used, whereas each column marked with from "2-Its" to "4-Its" indicates inner iteration steps of 2 to 4, respectively, that are performed at each preconditioning step. Also, the value "-1" in the table denotes the failure of convergence within the maximum number (200) of allowed iterations.

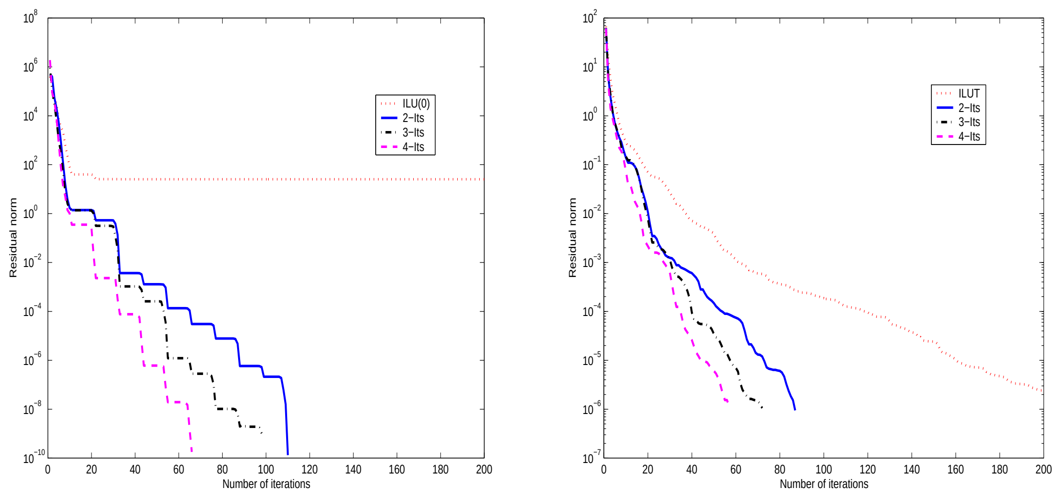
Table 3.6: Comparison of the number of PGMRES iterations with different number of inner iterations.

	ILU(0)				ILUT			
Matrix	1-Its	2-Its	3-Its	4-Its	1-Its	2-Its	3-Its	4-Its
ADD32	81	38	43	27	16	9	7	6
BFW398A	-1	117	116	50	-1	-1	-1	-1
JPWH991	29	15	13	10	32	18	13	11
LOP163	-1	110	98	66	18	8	7	5
ORSIRR1	41	22	20	16	50	25	30	17
PDE225	30	16	10	8	16	8	7	5
PDE900	61	31	18	15	33	16	11	9
SHERMAN1	116	52	57	32	107	50	47	30
SHERMAN4	143	65	46	33	136	60	36	26
SHERMAN5	101	54	39	29	-1	87	72	57
WATT1	178	66	53	36	108	49	39	31

Table 3.6 shows that the inner-outer iteration strategies reduce the number of PGMRES iterations for most of the tested matrices. Especially, ILU(0) with the inner-outer iteration strategies is shown to be effective for solving ADD32, BFW398A, LOP163, PDE225, PDE900, SHERMAN, and WATT1 matrices. ILUT with different number of inner-outer iteration strategies works well on the PDE and SHERMAN matrices.

Figure 3.2 shows that the number of PGMRES iterations is decreased as the number of inner iterations is increased for solving the LOP163 and SHERMAN5 matrices. In the figure, "ILU(0)" and "ILUT" denote the standard ILU(0) and ILUT preconditioner, respectively. Also, "2-Its," "3-Its," and "4-Its" represent the preconditioners with inner iteration steps of 2, 3, and 4, respectively.

In Table 3.7, we compare the total CPU processing time in seconds with and without the inner-outer iteration strategies. In general, larger number of inner iterations reduces the number of outer iterations needed for convergence. However, the reduction of the number of outer iterations does not always lead to the reduction of



(a) The LOP163 matrix with ILU(0) (b) The SHERMAN5 matrix with ILUT

Figure 3.2: The convergence curves for the LOP163 and SHERMAN5 matrices.

Table 3.7: Comparison of CPU time in seconds of ILU(0) and ILUT with different number of inner iterations.

	ILU(0)				ILUT			
Matrix	1-Its	2-Its	3-Its	4-Its	1-Its	2-Its	3-Its	4-Its
ADD32	1.1E+00	1.1E+00	1.4E+00	1.2E+00	1.2E+00	1.2E+00	1.5E+00	1.3E+00
BFW398A	N/A	1.7E-01	2.4E-01	1.6E-01	N/A	N/A	N/A	N/A
JPWH991	7.0E-02	7.0E-02	8.9E-02	7.0E-02	1.1E-01	1.1E-01	1.3E-01	1.2E-01
LOP163	N/A	4.9E-02	5.9E-02	3.9E-02	6.0E-02	4.9E-02	7.0E-02	3.9E-02
ORSIRR1	9.0E-02	1.3E-01	1.1E-01	1.2E-01	1.4E-01	1.7E-01	1.7E-01	1.7E-01
PDE225	1.9E-02	1.9E-02	1.9E-02	1.9E-02	1.9E-02	2.9E-02	2.9E-02	2.9E-02
SHERMAN1	1.5E-01	1.0E-01	1.3E-01	1.2E-01	2.4E-01	1.7E-01	2.1E-01	1.8E-01
WATT1	5.1E-01	3.9E-01	3.7E-01	3.6E-01	7.7E-01	5.9E-01	5.9E-01	5.8E-01

the total CPU time, as shown in Table 3.7. This is because that the cost of employing more inner iterations sometimes outweighs the savings obtained in the reduction of the number of outer iterations, as each outer iteration becomes more expensive to conduct. For the purpose of maintaining robustness, however, two or three inner iterations may be used in each outer iteration.

Comprehensive Results: Comparisons between Error Compensation and Inner-Outer Iteration Strategies

In this section, we show the comparison of the number of PGMRES iterations with different enhanced preconditioning strategies. The inner-outer iteration strategy is implemented with two inner iterations in this comparison.

Table 3.8: Comparison of the number of PGMRES iterations with and without different error compensation strategies.

	ILU(0)					ILUT				
Matrix	No	Full	2-Its	Both	BothL	No	Full	2-Its	Both	BothL
BFW398A	-1	150	117	81	88	-1	-1	-1	-1	-1
CDDE1	-1	102	94	55	77	164	102	66	44	54
CDDE3	-1	-1	-1	185	-1	-1	-1	196	198	175
ORSIRR1	41	40	22	22	22	50	41	25	22	25
ORSIRR2	40	41	22	22	22	51	43	25	24	25
ORSREG1	51	50	22	24	25	52	52	26	22	25
PDE2961	-1	-1	-1	-1	-1	85	66	51	31	47
PDE900	61	46	31	22	27	33	22	16	16	15
PORES2	117	148	-1	-1	82	-1	-1	-1	-1	-1
PORES3	108	64	52	44	54	121	127	88	53	82
SAYLR1	33	33	16	16	15	75	-1	24	31	24
SAYLR4	-1	-1	185	163	-1	-1	-1	-1	187	-1
SHERMAN1	116	62	52	43	52	107	87	50	51	50
SHERMAN4	143	107	65	43	55	136	101	60	43	37
SHERMAN5	101	73	54	44	50	-1	145	87	76	79
WATT1	178	114	66	40	69	108	85	49	37	49

Table 3.8 shows the comparison of the number of PGMRES iterations with different enhanced preconditioning strategies. The data in the columns marked "Both" and "BothL"mean that applying "Full" and "Lpart" followed by "2-Its". The data in the Table 3.8 indicate that most of the proposed strategies are of benefit to reduce the number of PGMRES iterations, and show that "Both" (the error compensation strategy with the L and U parts followed by the inner-outer iteration strategy) has best results for almost all tested matrices; 12 (8) matrices have best results with the "Both," and 3 (2) matrices with the "2-Its" (inner-outer iteration strategy), in the context of ILU(0) (ILUT). Also, "BothL" (the error compensation strategy with the

L part only followed by the inner-outer iteration strategy) is best to 4 (5) matrices with ILU(0) (ILUT), respectively.

Specifically, Table 3.8 indicates that most of the proposed strategies are of benefit to reduce the number of PGMRES iterations. In the context of ILU(0), 13 matrices have best results with the inner-outer iteration strategy, and 8 matrices with the error compensation strategies. However, the error compensation strategy with the L part only is also best for 4 matrices in the sense of the least number of GMRES iterations only. In the context of ILUT, 13 matrices have best results with the inner-outer iteration strategy, and 5 matrices with the error compensation strategy. As a result, the error compensation strategy followed by the two inner-outer iteration strategy outperformed the other error compensation strategies based on this experiment.

Table 3.9: Comparison of CPU time in seconds with the different error compensation strategies in ILU(0) preconditioning.

	ILU(0)				
Matrix	No	Full	2-Its	Both	BothL
BFW398A	N/A	1.9E-01	1.7E-01	1.9E-01	1.6E-01
CDDE1	N/A	1.6E-01	2.1E-01	1.5E-01	1.8E-01
CDDE3	N/A	N/A	N/A	3.4E-01	N/A
ORSIRR1	9.0E-02	1.3E-01	1.3E-01	1.4E-01	1.5E-01
ORSIRR2	8.9E-02	1.1E-01	9.9E-02	1.2E-01	1.1E-01
ORSREG1	2.6E-01	3.9E-01	2.7E-01	3.3E-01	3.1E-01
PDE2961	N/A	N/A	N/A	N/A	N/A
PDE900	9.0E-02	1.0E-01	8.9E-02	9.0E-02	8.9E-02
PORES2	2.7E-01	4.5E-01	N/A	N/A	3.9E-01
PORES3	7.9E-02	7.9E-02	9.0E-02	1.0E-01	1.0E-01
SAYLR1	2.9E-02	1.9E-02	1.9E-02	2.9E-02	1.9E-02
SAYLR4	N/A	N/A	2.1E+00	2.4E+00	N/A
SHERMAN1	1.5E-01	1.1E-01	1.0E-01	1.1E-01	1.2E-01
SHERMAN4	1.6E-01	1.6E-01	1.3E-01	1.2E-01	1.4E-01
SHERMAN5	6.9E-01	7.4E-01	7.3E-01	7.7E-01	7.6E-01
WATT1	5.1E-01	4.9E-01	3.9E-01	3.4E-01	4.5E-01

Table 3.9 and Table 3.10 present the total CPU processing time which includes the preconditioner construction time with different error compensation strategies. The listed results in Table 3.9 and Table 3.10 are concerned with the results listed in Table 3.8. “N/A” indicates the failure of computation since the number of GMRES

Table 3.10: Comparison of CPU time in seconds with the different error compensation strategies in ILUT preconditioning.

	ILUT				
Matrix	No	Full	2-Its	Both	BothL
BFW398A	N/A	N/A	N/A	N/A	N/A
CDDE1	3.6E-01	2.7E-01	3.2E-02	2.4E-01	2.8E-01
CDDE3	N/A	N/A	5.9E-01	6.8E-01	6.4E-01
ORSIRR1	1.4E-01	1.9E-01	1.7E-01	2.0E-01	2.0E-01
ORSIRR2	1.3E-01	1.7E-01	1.4E-01	1.7E-01	1.6E-01
ORSREG1	3.9E-01	5.8E-01	3.9E-01	4.7E-01	4.4E-01
PDE2961	1.2E+00	1.3E+00	1.4E+00	1.6E+00	1.6E+00
PDE900	1.2E-01	1.4E-01	1.3E-01	1.4E-01	1.4E-01
PORES2	N/A	N/A	N/A	N/A	N/A
PORES3	1.4E-01	1.6E-01	1.6E-01	1.6E-01	1.8E-01
SAYLR1	3.9E-02	N/A	2.9E-02	3.9E-02	2.9E-02
SAYLR4	N/A	N/A	N/A	4.2E+00	N/A
SHERMAN1	2.4E-01	1.9E-01	1.7E-01	1.9E-01	1.2E-01
SHERMAN4	2.9E-01	2.8E-01	2.2E-01	2.0E-01	2.0E-01
SHERMAN5	N/A	1.6E+00	1.3E+00	1.5E+00	1.4E+00
WATT1	7.7E-01	7.7E-01	5.9E-01	5.6E-01	6.9E-01

iterations reached the maximum number(200). This implies that as the number of overall iterations are greatly decreased, and the CPU time is also reduced even though we need extra processing time for implementing the error compensation strategies.

3.4 Conclusions and Remarks

We proposed preconditioning accuracy enhancement strategies to augment ILU (incomplete LU) factorizations in case that the initial factorizations are found to be inaccurate. The strategies recompense the dropped elements during the incomplete factorization before/during the preconditioned iteration processes. Results of extensive numerical experiments demonstrated that the proposed strategies are quite promising and effective in helping reduce the number of PGMRES iterations, and a noticeable improvement in enhancing the accuracy of ILU factorizations. Furthermore, the total computation time for solving sparse linear system was reduced in many cases even though extra processing time was used for the proposed strategies.

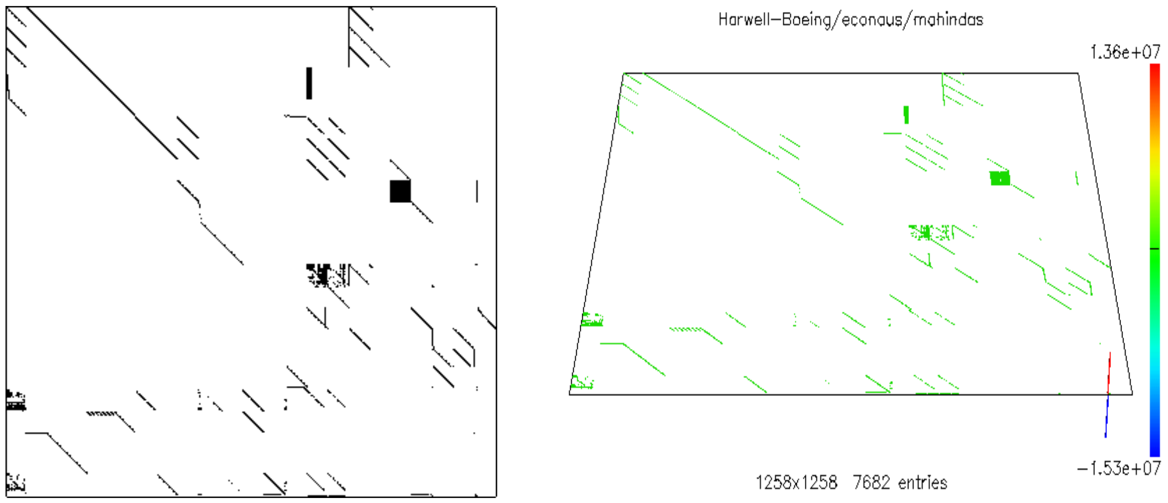
Chapter 4 Hybrid Reordering Strategies

Incomplete LU factorization preconditioning techniques often cause preconditioners to be unstable and/or inaccurate due to very small pivots encountered during the factorization in solving indefinite matrices. In this chapter, we concern ourselves with improving the stability and accuracy of preconditioners in solving indefinite matrices using two diagonal reordering algorithms that replace small pivots with large values on the main diagonal by searching for entries of a single nonzero element and/or maximum absolute value.

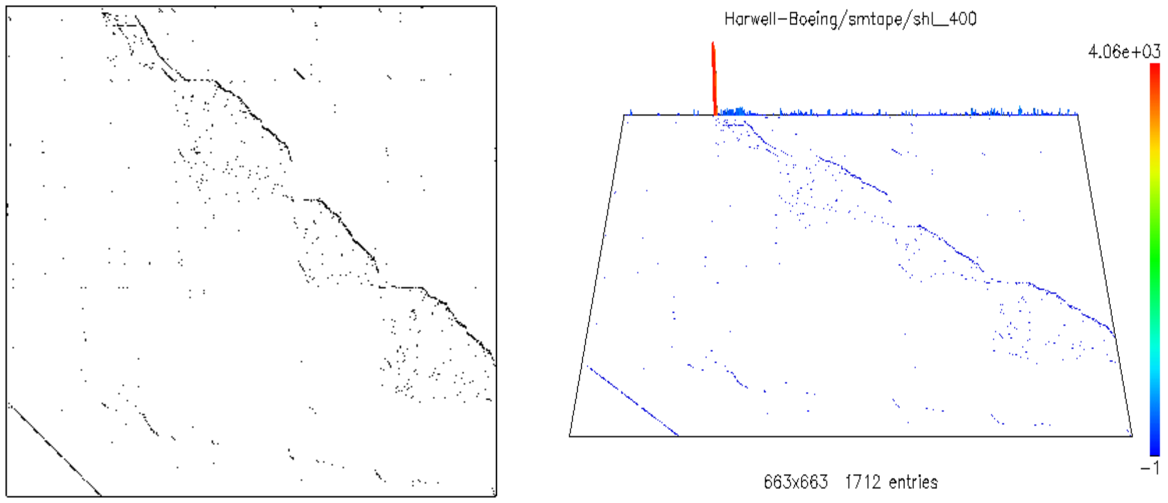
4.1 Introduction

Incomplete LU (ILU) preconditioning techniques have been successful for solving many symmetric and nonsymmetric matrices, however, they may encounter difficulty when the matrix to be solved is indefinite, i.e., when the matrix has both positive and negative eigenvalues (See Figure 4.1 for the examples of indefinite matrices). If matrix A has positive and negative eigenvalues, $M^{-1}A$ attempts to move negative eigenvalues to the positive side and to be clustered around 1. But an unsuccessful preconditioning may move some of the eigenvalues to be close to 0, which causes $M^{-1}A$ to have a very large condition number.

According to Chow and Saad [19], there are at least two reasons which make ILU factorization approaches difficult in solving indefinite matrices. The first reason can be due to small or zero pivots in an indefinite matrix, which may lead to unstable and inaccurate factorizations [44]. In addition, small pivots are usually related to small or zero entries on the main diagonal of the matrix, so an indefinite matrix with zero diagonal entries may have higher chances of encountering zero pivots if the matrix is also nonsymmetric [54, 68]. Secondly, unstable triangular solutions can



(a) MAHINDAS Matrix



(b) SHL_400 Matrix

Figure 4.1: The MAHINDAS and SHL_400 matrices [50].

happen when $||L^{-1}||$ and $||U^{-1}||$ are extremely large while the off-diagonal elements of L and U are reasonably bounded. Such problems are usually caused by very small pivots [19, 34, 68].

Since small pivots are often the origin of stability problems in computing ILU factorization on an indefinite matrix, we can expect a better performance if small pivots could be replaced with some large values on the diagonal of the matrix. Based on the idea mentioned above, several researchers [10, 27, 28, 30, 31] focused on permuting large entries onto the main diagonal of an indefinite matrix before using direct or iterative solvers to solve the matrix. For example, Duff and Koster [30, 31] exploited bipartite matching algorithms and scaling techniques for computing permutations of a sparse matrix. Their reordering and scaling algorithms have an effect on the performance of various solution methods for solving sparse matrices. In the experimental studies with the Duff and Koster's [31] algorithms, Benzi *et al.* [10] considered matching algorithms such as maximum transversal, maximum product transversal, and the bottleneck transversal with symmetric permutations. Their preprocessing step still requires complicated matching algorithms with several symmetric permutations.

As a part of the continuous efforts to develop approaches to replace small pivots with large values on the main diagonal, we present two diagonal reordering algorithms that are computationally inexpensive but efficient in the aspect of reordering performance for indefinite matrices, and also propose new hybrid reordering strategies that are composed of a combination of the two proposed diagonal reorderings and the nondecreasing degree algorithm [68] described in Section 4.2.

The main idea of the diagonal reordering algorithms lies on searching for entries of (1) single nonzero element and/or (2) the maximum absolute value, to be placed on the main diagonal for computing a nonsymmetric permutation. SINGLE-ELEMENT-REORDERING described in Algorithm 4.2.2 examines the rows and the columns which have only one nonzero element for the permutation so that the main diagonal of the permuted matrix has more nonzero entries than the original one. On the other hand, MAXIMUM-VALUE-REORDERING described in Algorithm 4.2.3 searches for the entries, which have the maximum absolute value for each column for the permutation, to maximize the absolute value on the diagonal entries for each column.

In order to augment the effectiveness of the nonsymmetric diagonal reorderings, we apply a symmetric nondecreasing degree algorithm to reduce the amount of fill-in during the ILU factorization. This algorithm is used to reorder the rows of the matrix in a decreasing degree fashion; i.e., the rows with smaller degrees are listed first and those with larger degrees are listed last [68]. It will be defined later.

In Section 4.3, we present the numerical results to demonstrate the performance of each reordering strategy. A set of sparse indefinite matrices were used in the experiments. We first performed each reordering strategy as a preprocessing step. In order to solve the reordered matrices, we employed several ILU type preconditioners and a preconditioned iterative solver. The comparison factors derived from solving the reordered matrices were nonzero entries on the main diagonal, the number of iterations, the amount of fill-in in the preconditioners, and a preconditioner conditioning estimate parameter. As a result, we can see that the convergence cost of the preconditioned Krylov subspace methods on solving the reordered indefinite matrices is markedly reduced by using our reordering strategies.

4.2 Hybrid Reordering Strategy

Let A be an indefinite matrix, and $P = \{(i,j) \mid i = j,\ 1 \le i,j \le n\}$ be the set of pairs for a permutation, where n is the order of the matrix A, and (i,j) means to interchange rows i and j of the matrix A.

Algorithm 4.2.1. VERIFY-PERMUTATION*(A, i, k, d).*

```
1. $r \leftarrow$ FALSE
2. $\widehat{k} \leftarrow$ COUNT-NONZEROS($A, i, d$)
3. $c \leftarrow$ CHECK-PAIR($P, i, d$)
4. If ( ($k = \widehat{k}$) $\wedge$ ($c$ = TRUE) )
5.     $r \leftarrow$ TRUE
6. Return $r$
```

We describe the VERIFY-PERMUTATION(A, i, k, d) function in Algorithm 4.2.1 to determine a permutation, where d is the direction (`ROW` or `COL`; i.e., row-wise or column-wise), i is the index for the direction d, and k is the number of nonzero elements to be compared. In line 2, the COUNT-NONZERO(A, i, d) function counts the number of nonzero elements in the i-th of the direction d (row or column). In line 3, the CHECK-PAIR(P, i, d) function determines whether the pair (i, d) is used for the permutation set P. At last, line 4 verifies whether the given position on the matrix A can be a member of the set P.

Single-Element-Reordering

Algorithm 4.2.2. SINGLE-ELEMENT-REORDERING.

```
1. For $i = 1, ..., n$
2.    If (VERIFY-PERMUTATION(A, i, 1, ROW) = TRUE )
3.       $j \leftarrow$ Find the column index of the nonzero element in row $i$.
4.       Replace $(i, i)$ with $(i, j)$ in $P$
5. For $i = 1, ..., n$
6.    If (VERIFY-PERMUTATION(A, i, 1, COL) = TRUE)
7.       $j \leftarrow$ Find the row index of the nonzero element in column $i$.
8.       Replace $(j, j)$ with $(j, i)$ in $P$.
```

Algorithm 4.2.2 describes the SINGLE-ELEMENT-REORDERING algorithm that finds rows and columns which have only one nonzero element for the permutation. The main role of this algorithm is to increase the number of nonzero elements on the diagonal. The number of elements in a row or in a column can be computed very inexpensively if we use the compressed sparse row (or column) format. Since both rows and columns are needed, the most efficient implementation needs the matrix to be stored in both formats. We can temporarily use the storage space allocated for the preconditioner to be used in a later phase to store an additional copy of the matrix in another format. With the availability of both rows and columns, the SINGLE-ELEMENT-REORDERING algorithm costs $O(n)$ nonnumerical operations. It is computationally more efficient than the maximum transversal algorithm [10, 30, 31], which is $O(n * q)$, where q is the number of nonzero entries of the matrix.

Maximum-Value-Reordering

Algorithm 4.2.3. MAXIMUM-VALUE-REORDERING.

```
1. $\alpha \leftarrow 0$
2. For $i = 1, ..., n$
3.     $\widehat{k} \leftarrow$ COUNT-NONZEROS$(A, i, d)$
4.     If $(\alpha < \widehat{k}\ )\ \alpha \leftarrow \widehat{k}$
5. For $k = 2, ..., \alpha$
6.     For $i = 1, ..., n$
7.         If (VERIFY-PERMUTATION$(A,\ i,\ k,$ COL$)=$TRUE )
8.             $j \leftarrow$ Find the row index of the maximum value element in column $i$.
9.             Replace $(j, j)$ with $(j, i)$ in $P$.
```

As described in Algorithm 4.2.3, the MAXIMUM-VALUE-REORDERING algorithm searches for an element which has the maximum absolute value in each column for a permutation to place that entry on the diagonal of each column of the matrix. In lines 1-4, we find the maximum number α of nonzero elements in the columns. In line 8, the algorithm finds the maximum absolute value in the columns takes $O(n)$ time. Thus, it is obvious that the time complexity of MAXIMUM-VALUE-REORDERING is $O(n^2 * \alpha) = O(n * q)$, where α is the maximum number of nonzero elements in a column.

Duff and Koster [30, 31], and Benzi *et al.* [10] consider the maximum product transversal which the product of the absolute value of the entries on the diagonal is maximized with a complexity $O(n * q * \log n)$. Since the matrices under our consideration are very sparse, α is smaller than $\log n$ and $n \ll q$. The MAXIMUM-VALUE-

Reordering algorithm is also more efficient in terms of time complexity than the maximum product transversal [10, 31].

Nondecreasing Degree Algorithm

We denote by $\mathrm{d}eg(v_i)$ the degree of the node (row) v_i, which equals the number of nonzero elements of the i-th row minus one, i.e., $\mathrm{d}eg(v_i) = Nz(A_i) - 1 = Nz(v_i) - 1$, where $Nz(A_i)$ represents the number of nonzero elements in row i of a matrix A. To be more precise, we reorder the nodes in a decreasing degree fashion; i.e., the nodes with smaller degrees are listed first and those with larger degrees are listed last. The purpose of the nondecreasing degree reordering strategy is to reduce the fill-in during the ILU factorization [67].

4.3 Experimental Results

We present numerical experiments of the reordering strategies in this section. The preconditioned iterative solver, GMRES(20), was employed with several ILU type preconditioners, ILU(0), MILU(0), ILUT, and ILUTP. For all linear systems, the right-hand side was generated by assuming that the solution is a vector of all ones. The initial guess was a zero vector. The iteration was set to terminate when the l_2-norm of the initial residual was reduced by at least seven orders of magnitude, or when the number of iterations reached 500.

Description of Test Matrices

The test sparse matrices for experiments were unsymmetric matrices from the Harwell-Boeing sparse matrix test collection [29] and the sparse matrix collection at the University of Florida [24]. A short description of the matrices is given in Table 4.1.

In all tables with numerical results, SER, MVR, and SMR represent Single-Element-Reordering, Maximum-Value-Reordering, and SER followed by

Table 4.1: Description of the test matrices.

Name	Description
MAHINDAS	Economic model of Victoria, Australia, 1880 data
OLM1000	The OLMstead model represents the flow of a layer of viscoelastic fluid
OLM5000	The OLMstead model represents the flow of a layer of viscoelastic fluid
SHL_0	Simplex method basis matrix
SHL_200	Simplex method basis matrix
SHL_400	Simplex method basis matrix
FPGA_DCOP_03	Circuit simulation matrices
FPGA_DCOP_07	Circuit simulation matrices
FPGA_TRANS_01	Circuit simulation matrices
FPGA_TRANS_02	Circuit simulation matrices
INIT_ADDER1	Circuit simulation matrices
SHERMANACa	Matrices from Kai Shen, UCSB
SHERMANACd	Matrices from Kai Shen, UCSB
ADDER_DCOP_15	Circuit simulation matrices
MEG4	Primarily chemical process simulation matrices

MVR algorithm, respectively.

Comparisons of Reordering Strategies

The comparisons of nonzero entries on the diagonal and the number of GMRES iterations with the reordering strategies are presented. For simplicity, n, nnz, org, and $nnzD(org)$ denote the dimension of the matrix, the number of nonzero entries in the original matrix, the original matrix, and the number of nonzero entries on the diagonal of the original matrix, respectively.

Table 4.2 reports that in most cases, the reordering strategies place more nonzero entries on the diagonal than those of the original matrix. For example, the matrices SHL_0, SHL_200, and SHL_400 have 5, 6, and 3 nonzero diagonal entries ($nnzD$) in original matrices, and $nnzD$ of the each matrix was increased 406, 360, and 353 with SER, and 663, 645, and 641 nonzero with SMR, respectively. For the SHL_0 matrix, all diagonal entries become nonzero with SMR. We note that $nnzD$ of some matrices are reduced with MVR and SMR, but such losses are seen as minor because the number of GMRES iterations are decreased compared to those of original matrix. In

Table 4.2: Comparison of nonzero entries on diagonal.

Name	n	nnz	$nnzD(org)$	$nnzD$(SER)	$nnzD$(MVR)	$nnzD$(SMR)
MAHINDAS	1258	7682	106	519	907	1152
OLM1000	1000	3996	1000	1000	352	1000
OLM5000	5000	19996	5000	5000	366	5000
SHL_0	663	1687	5	406	352	663
SHL_200	663	1726	6	360	366	645
SHL_400	663	1712	3	353	371	641
FPGA_TRANS_01	1220	7382	1154	1220	1158	1220
FPGA_TRANS_02	1220	7382	1154	1220	1158	1220
SHERMANACa	3432	25220	3432	3432	3408	3432
SHERMANACd	6136	53329	6136	6136	6110	6130
ADDER_DCOP_15	1813	11246	1801	1813	1786	1797
MEG4	5860	46842	5806	5860	5806	5860

the case of "SHERMANACd," the number of nonzero diagonal entries of the matrix is reduced from 6136 to 6130 with SMR reordering, but the failure of convergence result of the matrix is changed to convergence with 83 GMRES iterations (See Table 4.3).

Table 4.3: Comparison of the number of iterations.

Name	τ	p	*precond*	org	ND	SER-ND	MVR-ND	SMR-ND
MAHINDAS	10^{-3}	15	ILUTP	12(U)	12	12	11	6
OLM1000	10^{-2}	5	ILUT	-1	-1	-1	-7	19
OLM5000	10^{-2}	5	ILUT	-1	-1	-1	-7	19
SHL_0			ILU(0)	-7	-7	-7	-7	7
SHL_0	10^{-4}	5	ILUT	-3	-3	-1	-7	2
SHL_200	10^{-5}	10	ILUT	-3	-3	-3	-1	31
SHL_400	10^{-3}	10	ILUT	-3	-3	-3	-1	53
FPGA_TRANS_01			ILU(0)	-7	-7	105	-7	105
FPGA_TRANS_02			ILU(0)	-7	-7	105	-7	105
SHERMANACa	10^{-3}	5	ILUT	-3	-3	-3	-1	9
SHERMANACd	10^{-6}	15	ILUT	-3	-3	-3	-1	83
ADDER_DCOP_15			ILU(0)	-7	-7	299	-7	-7
MEG4			ILU(0)	-7	-7	7	-7	7

In Table 4.3, we present comparison results of the number of preconditioned GMRES iterations with the reordering strategies. We have tested SER, and SMR followed by a symmetric nondecreasing degree (ND) permutation, denoted "SER-ND" and "SMR-ND," respectively, to reduce fill-in in the preconditioner. The "τ" and "p" columns in Table 4.3 are the two dropping parameters used in the ILUT factorization, and the "*precond*" column specifies the particular preconditioners. The value "-1" and "-3" indicate the failure of convergence within the maximum number (500) of allowed iterations and with the breakdown of the GMRES solver, respectively, and "-7" indicates that the preconditioner was not constructed due to zeros on the main diagonal (zero pivot). The symbol "U" denotes failure due to unstable preconditioning result, which only happened with the original "MAHINDAS" matrix, and the computed results are not reliable although the preconditioned solver stopped at 12 iterations. Overall, SMR seems to be the most robust since it helped the ILU preconditioners solve all but one matrix used in the experiments.

Table 4.4: Comparison of the number of nonzero entries.

Name	precond	*org*	ND	SER-ND	SMR	SMR-ND
MAHINDAS	ILUTP	20234	9374	9087	25864	9520
OLM1000	ILUT	2000	2000	2000	1997	1997
OLM5000	ILUT	9997	9997	9997	9997	9997
SHL_0	ILUT	3946	3647	2857	1898	1409
SHL_200	ILUT	5839	5707	5399	2568	1970
SHL_400	ILUT	5885	5353	5120	2589	1967
FPGA_DCOP_03	ILUT	9174	4837	4189	8483	4189
FPGA_DCOP_07	ILUT	9339	4713	4106	8514	4106
INIT_ADDER1	ILUTP	22679	8841	8833	27586	9110
SHERMANACa	ILUT	42094	43395	43395	29163	23277
SHERMANACd	ILUT	191446	165574	362596	231520	84770

As shown in Table 4.4, we compare the number of nonzero entries in the preconditioners with SER and SMR followed by the ND algorithm with the same dropping tolerance parameters as listed in Table 4.3. We can see that the nondecreasing degree strategy reduces the number of fill-in entries during the ILUT and ILUTP factorizations. It is interesting to notice that for the most matrices including the SHL matrices and the SHERMAN matrices, the SMR-ND strategy have the minimum or close to the minimum amount of nonzero entries.

Computational Analysis

According to Greenbaum and Saad [39, 55], the convergence behavior of the Krylov subspace methods depends on the distribution of the eigenvalues and on the condition number of the coefficient matrix. Based on these theoretical facts, we analyze the reordering algorithms by presenting some numerical and graphical results.

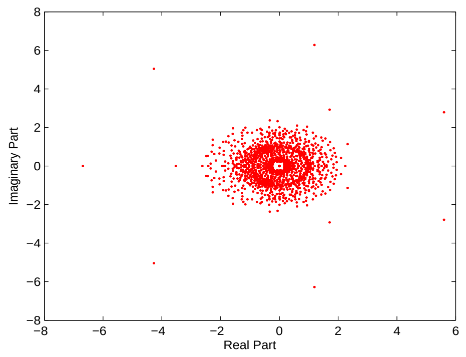
For a given (diagonalizable) matrix A, it can be decomposed as $A = X\Sigma X^{-1}$, where $\Sigma = diag\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$ is the diagonal matrix of the eigenvalues and X is the matrix of the eigenvectors of the matrix A. The condition number of the eigenvector matrix X measures the normality of the matrix A, and it can be used to explain the convergence of the Krylov subspace methods heuristically [39, 55].

Table 4.5: The largest and smallest eigenvalues and condition numbers of the MAHINDAS matrix.

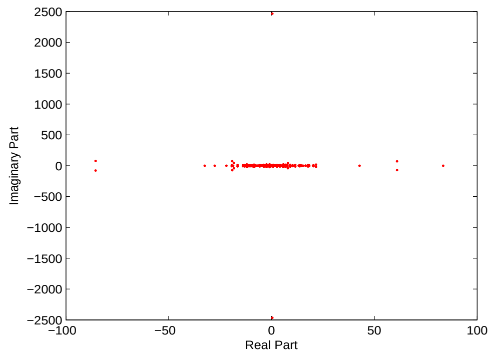
Matrix	$\lambda_{largest}$	$\lambda_{smallest}$	cond(X)
A	-6.68E+00	-2.90E-03 + 9.60E-03i	1.08E+07
ILUT(A)	4.16E-01 + 2.47E+03i	N/A	6.42E+07
SMR-ND followed by ILUT	2.52E+06	1.95E-05	1.09E+06

Table 4.5 reports the largest and the smallest eigenvalues in magnitude of the original and the preconditioned matrices, and the condition number of the eigenvector matrices in the MAHINDAS matrix. Under “N/A”, we represent that the the smallest eigenvalue was not computed. As shown in the table and Figure 4.2 (a), the condition number of the eigenvector matrix of the original matrix is relatively large, and the MAHINDAS matrix is highly indefinite.

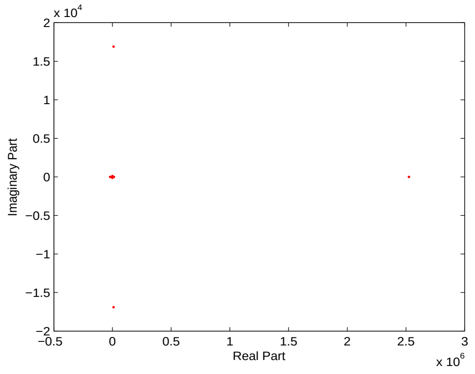
The ILUT preconditioner makes the spectrum more spread around the origin and indefinite than those of the original matrix in real part (See Figure 4.2 (b)). This makes the preconditioned matrix have slightly larger condition number of the eigenvector matrix and causes a failed convergence of the GMRES method on the ILUT preconditioned matrix. On the other hand, in Figure 4.2 (c), the eigenvalues ap-



(a) Eigenvalues of the MAHINDAS matrix



(b) Eigenvalues of the MAHINDAS matrix with ILUT



(c) Eigenvalues of the MAHINDAS matrix with SMR-ND followed by ILUT

Figure 4.2: Plots of the eigenvalues of the MAHINDAS matrix. (These subfigures are in different scale.)

plied by the SMR-ND reordering followed by the ILUT preconditioner are shifted to the right-hand side of the origin in real part (positive definite). Also, the condition number of the eigenvector matrix is decreased. These explain the good convergence behavior of the GMRES method on this reordering followed by the ILUT preconditioned matrix.

Next, we use some statistical information to estimate condition numbers of the preconditioned matrices. In order to estimate the condition number, a statistic "condest", suggested by Chow and Saad [19], is utilized by calculating $||(LU)^{-1}e||_\infty$, where e is the vector of all ones. This statistics indicates a relation between the unstable triangular solutions and the poorly conditioned L and U factors.

Table 4.6: Comparison with condest.

Name	precond	*org*	ND	SER-ND	SMR	SMR-ND
MAHINDAS	ILUTP	Unstable	$4.30E+3$	$1.61E+5$	$1.03E+6$	$5.54E+5$
OLM1000	ILUT	$1.15E+2$	$1.12E+2$	$1.15E+2$	1.88	1.98
OLM5000	ILUT	$1.01E+3$	$1.01E+3$	$1.0E+3$	1.79	1.81
SHL_0	ILU(0)	N/A	N/A	N/A	$4.71E+3$	$4.51E+3$
SHL_0	ILUT	Infinity	Infinity	Unstable	$5.44E+3$	$5.44E+3$
SHL_200	ILUT	Infinity	1.00	Unstable	$3.21E+13$	$9.73E+10$
SHL_400	ILUT	Infinity	Infinity	Unstable	$3.65E+11$	$7.19E+14$
FPGA_DCOP_03	ILUT	$1.49E+12$	$1.50E+12$	$1.50E+12$	$1.50E+12$	$1.50E+12$
FPGA_DCOP_07	ILUT	$1.50E+12$	$1.50E+12$	$1.50E+12$	$1.50E+12$	$1.50E+12$
FPGA_TRANS_01	ILU(0)	N/A	N/A	$1.92E+1$	$4.89E+2$	$1.92E+1$
FPGA_TRANS_02	ILU(0)	N/A	N/A	$1.92E+1$	$5.28E+2$	$1.92E+1$
INIT_ADDER1	ILUTP	$7.62E+8$	$1.23E+8$	$1.23E+8$	$1.50E+8$	$1.23E+8$
SHERMANACa	ILUT	Unstable	Infinity	Infinity	$2.07E+4$	$2.07E+4$
SHERMANACd	ILUT	$2.83E+3$	$2.83E+3$	$2.83E+3$	$3.29E+19$	$4.67E+11$
ADDER_DCOP_15	ILU(0)	N/A	N/A	$5.00E+11$	N/A	N/A
MEG4	ILU(0)	N/A	N/A	$1.97E+5$	$1.97E+5$	$1.97E+5$

The results are listed in Table 4.6, which are concerned with the results listed in Table 4.3. Under "N/A", we report that the preconditioner was not defined, due to zeros on the diagonal (zero pivot). Also, "Infinity" and "Unstable" represent the unstable triangular solvers. Here, we use "Unstable" if condest $> 10^{15}$. The statistics of the "condest" shows that many matrices greatly benefit from the proposed SMR algorithm in this paper.

4.4 Concluding Remarks

We have proposed hybrid reordering strategies for an indefinite matrix A in ILU preconditioning techniques. The strategies are combinations of the diagonal reorderings to construct stable preconditioner and nondecreasing degree reordering to reduce fill-in numbers. The diagonal reorderings were used to make a matrix increase the number of the nonzero elements on the diagonal and/or maximize the absolute value on the diagonal entries. It is also worth mentioning that the strategies run efficiently in low computational time cost. For the performance aspect, we showed various experimental results from the proposed reordering strategies. The results demonstrated were quite promising and effective in helping construct ILU type preconditioners for solving indefinite sparse matrices. In addition to that, the convergence cost of the preconditioned Krylov subspace methods on solving the reordered indefinite matrices was greatly decreased.

Chapter 5 A Two-Phase Preconditioning Strategy

In this chapter, as a part of the continuous effort on solving indefinite matrices, a two-phase preconditioning strategy based on a factored sparse approximate inverse is proposed. This strategy adopted the idea of a shifting method, which adds a number to the diagonals of an indefinite matrix, to make the resulting preconditioner well conditioned. Moreover, the two-phase preconditioning process reinforces the tradeoff between stability and accuracy of the resulting preconditioner obtained from the shifting method.

5.1 Introduction

In recent years, a few preconditioning techniques in the form of sparse approximate inverse have been developed [9, 16, 18, 20, 37, 38, 41]. Such techniques have some advantages over the conventional ILU factorizations. Specifically, the process of applying the sparse approximate inverse preconditioning techniques can be performed by the matrix-vector operations, in which the operations are relatively easier to parallelize than the triangular solutions associated with the ILU factorizations. The sparse inverse preconditioning techniques may succeed in solving certain problems where the ILU factorizations are difficult to handle [20]. In addition to that, factored sparse approximate inverse (FAPINV) in which a sparse approximate inverse has a factored form, tends to perform better in convergence rate for the same amount of nonzeros and also requires less computational cost than a non-factored form does. However, the resulting approximate inverse could still break down for solving indefinite matrices due to zero or small pivots [9, 69].

In solving indefinite matrices, small pivots are also often the origin of stability problems in computing SAI preconditioners. Therefore, several researchers have focused on designing methods that replace the small pivots of indefinite matrices with some large values. In [8, 51, 64], a shifting strategy that adds a value to the diagonals of an indefinite matrix is proposed to make the resulting preconditioner better conditioned. It has been noticed that determining the value to be added for small pivots is usually critical to the performance of the resulting preconditioner [44]. In other words, selecting a large replacing value may result in a factorization that is stable but less accurate. On the other hand, selecting a small replacing value may result in a factorization that is accurate but unstable. Such a tradeoff of the shifting strategy has been well studied in [68].

We propose to adopt the idea of a shifting strategy [51] to replace small or zero elements on the main diagonal of the original matrix, and to reinforce with a two-phase preconditioning process to deal with the tradeoff between stability and accuracy of the resulting preconditioner. More specifically, the first phase of the process employs the shifting strategy to the original matrix so that a shifted matrix can be well conditioned, and then an approximate inverse, M_1, of the shifted matrix is obtained by utilizing FAPINV. In the second phase of the process, a temporary matrix which is a product of M_1 and the shifted matrix, is considered to acquire a better approximate inverse of the original matrix than M_1. Applying the shifting strategy again to the temporary matrix produces a second shifted matrix, and FAPINV computes a second inverse approximation, M_2, of the second shifted matrix. The resulting sparse approximate inverse, M, has the form of $M = M_2M_1$, where M_1 and M_2 are computed in each phase, respectively.

This chapter is organized as follows. In Section 5.2, a function for determining the shifting parameter α, and a two-phase preconditioning of shifted matrices for computing sparse approximate inverses are proposed. In Section 5.3, numerical results are presented to demonstrate advantages and preconditioning performance of the proposed preconditioner over the standard FAPINV preconditioner. Concluding remarks are in Section 5.4.

5.2 Stabilized FAPINV (SFAPINV) Preconditioner

We now introduce the stabilized factored approximate inverse (SFAPINV) algorithm for solving indefinite matrices. The SFAPINV preconditioner is computed in a two-phase preconditioning of two shifted matrices. Each phase generates an approximation of the inverse of a shifted matrix. These two approximations have factored forms of $M_1 = L_1 D_1 U_1 \approx {A_1}^{-1}$ and $M_2 = L_2 D_2 U_2 \approx {A_2}^{-1}$, where L_i is a lower triangular matrix, D_i is a diagonal matrix, and U_i is an upper triangular matrix, $i = 1, 2$. Finally, the resulting preconditioner becomes a form of $M = (L_2 D_2 U_2)(L_1 D_1 U_1)$.

Determining the Shifting Parameter

Indefinite matrices usually have many small or zero pivots that can be the reason of breakdown in constructing sparse approximate inverse as well as ILU-type preconditioners. In order to prevent the resulting sparse approximate inverse from being unstable, we employ a shifting strategy which factors a shifted matrix $A' = A + \alpha I$, where α is a scalar so that $A + \alpha I$ is well conditioned (e.g., diagonally dominant). As was mentioned in the introduction, the choice of α is significant for good performance for such strategies. For example, if a matrix A is ill-conditioned, the inverse of $A + \alpha I$ could be quite different from A^{-1} for a large value of α [11, 9, 69]. Indeed, α should be large enough to ensure the existence of the sparse approximate inverse factorization, but also small enough so that $A + \alpha I$ is close to A.

We present an efficient algorithm FIND-ALPHA(A, α) in Algorithm 5.2.1, to determine a proper value for the shifting parameter α, rather than a brute-force approach which is computationally prohibitive.

Algorithm 5.2.1. FIND-ALPHA*(A, α).*

```
1.  For $i = 1, n$
2.      $c(i) = 0$
3.  $\alpha = 0$
4.  For $i = 1, n$
5.      $temp = 0$
6.      For $j = 1, n$
7.          If $(j \neq i)$, then $c(i) = c(i) + |a_{ij}|$
8.          else $temp = |a_{ii}|$
9.      If $(c(i) \leq temp)$, then $c(i) = temp$
10. For $i = 1, n$
11.     If $(\alpha \leq c(i))$, then $\alpha = c(i)$
12. Return $\alpha$
```

Note that n refers to the order of the original matrix A, a_{ij} denotes a nonzero element in row i and column j, and $c(i)$ represents an accumulator of the off-diagonals in row i, where $i, j = 1, \cdots, n$. The function starts with initializing $c(i)$ for all rows and the shifting parameter α. In lines 5–12, $c(i)$ is determined by selecting a larger value between the summation of the absolute value of the off-diagonals in row i and the diagonal of row i. In lines 13–16, the shifting value α is decided by choosing the largest value among $c(i)$s. At the end, the function returns the largest accumulator as the computed shifting parameter that will make all rows diagonally dominant.

Two-Phase FAPINV Preconditioner (SFAPINV)

Given a sparse approximate inverse M_1 computed by using FAPINV to the original matrix A, we assume that M_1 is inefficient to solve the preconditioned linear system

$$M_1 A x = M_1 b. \tag{5.1}$$

Another sparse approximate inverse M_2 for the preconditioned linear system (5.1) could be considered to acquire a closer inverse of A than M_1. A product of the two preconditioners, M_2M_1, is utilized as a sparse approximate inverse of A, and M_2M_1 becomes more accurate to the inverse of A than M_1 does. In fact, the product matrix M_2M_1 may hold more information than a single matrix M can. If M_2M_1 is not successful to solve the preconditioned system, another approximate inverses may be considered, and this procedure can be continued for a few times to obtain a good preconditioner (See [63] for details).

In the case that the original matrix is indefinite, a combined preconditioner M_2M_1, computed directly from A, however, may be unstable because of small or zero pivots. Furthermore, the shifting strategy makes the original matrix diagonally dominant, but the inverse of $A + \alpha I$ could be quite different from A^{-1} if A is ill-conditioned [11, 9, 69]. Thus, we combine the two-phase preconditioning with the shifting strategy, called STABILIZED FAPINV PRECONDITIONER (SFAPINV) in Algorithm 5.2.2, to improve the accuracy and stability of the sparse approximate inverse factorization. Here, FAPINV [69] is applied as a local preconditioner in each phase.

Algorithm 5.2.2. STABILIZED FAPINV PRECONDITIONER

```
1. Call FIND-ALPHA($A, \alpha_1$)
2. Construct a shifted matrix $A_1 = A + \alpha_1 I$
3. Call FAPINV($A_1, \tau_1, M_1$)
4. Compute a temporary matrix $W = M_1 A$
5. Drop small entries of $W$ with respect to $\tau$
6. Call FIND-ALPHA($W, \alpha_2$)
7. Construct a shifted matrix $A_2 = W + \alpha_2 I$
8. Call FAPINV($A_2, \tau_2, M_2$)
9. Return $M_2 M_1$
```

Note that τ_1, τ_2, and τ represent different dropping tolerances. The algorithm first determines the shifting parameter α_1 from the FIND-ALPHA function, and constructs a shifted matrix A_1 which becomes diagonally dominant. In line 3, FAPINV(A_1, τ_1, M_1) applies the FAPINV preconditioner to the matrix A_1 with the dropping tolerance τ_1, and returns an approximate matrix M_1 which has a factored form of $M_1 = L_1D_1U_1$ of the inverse of A_1. Then with the factored approximate inverse M_1, the preconditioned system (5.1) can be written as

$$(L_1D_1U_1)Ax = (L_1D_1U_1)b. \tag{5.2}$$

The preconditioned system (5.2), however, may need many iterations to converge because the factorization $L_1D_1U_1$ could be inaccurate if the shifting parameter α_1 is too large. For this potential problem in inaccuracy, in line 4, a temporary matrix $W = M_1A$ is considered to further precondition the system (5.2). In line 5, a dropping threshold parameter, τ, is applied to control the sparsity rate of W, where W is usually denser than A except the diagonal entries. By doing this, the diagonal entries of the matrix W are not dropped regardless of their magnitude. In lines 6-8, the shifting strategy is re-employed to obtain a more accurate and stabilized preconditioner, and the second FAPINV(A_2, τ_2, M_2) factorization computes an approximation M_2 which has a factored form $M_2 = L_2D_2U_2$ of the inverse of A_2. In line 9, the algorithm returns a combined approximation, M_2M_1, of the inverse of A. As a result, the final preconditioned system becomes

$$(L_2D_2U_2)(L_1D_1U_1)Ax = (L_2D_2U_2)(L_1D_1U_1)b. \tag{5.3}$$

Now, we point out some important features behind our strategy.

- It has been well studied that for a given matrix, finding a suitable shifting parameter of the existing shifting strategies may be complicated [64]. More specifically, too small a diagonal shift will not have the desired effect of stabilization, but too large a diagonal shift will result in an inaccurate preconditioner. In this regard, the FIND-ALPHA(A, α) function provides an explicit and efficient way in determining the shifting parameters compared to a commonly used brute-force approach.

- It is easy to see that the resulting preconditioner M_2M_1 is nonsingular. This claim can be justified by the following two facts. The first is that M_1 and M_2 are produced by the FAPINV preconditioner in a factored form which is nonsingular as long as each factor is nonsingular. The second is that the two factored sparse approximate inverses are not singular [69]. The singularity of each factor can be guaranteed by making sure that its diagonal is not zero.

- In general, the second shifting parameter α_2 requires to be much smaller than the first parameter α_1. It can be observed that a matrix which has more zeros on its diagonal needs a larger shifting parameter to stabilize the original matrix. In fact, $W = M_1A$ tends to be closer to I, or more diagonally dominant than A_1 does. Here, we propose two possible settings in choosing the two shifting parameters α_1 and α_2. For the case that a matrix is ill-conditioned and has a low percentage of zeros on its diagonal, the setting with $\alpha_1 =$ FIND-ALPHA(A, α_1) and $\alpha_2 =$ FIND-ALPHA(W, α_2) can be a proper choice. On the other hand, if a matrix has a large number of zeros on the diagonal of the matrix, the setting with $\alpha_1 =$ FIND-ALPHA(A, α_1) and $\alpha_2 = 0$ is recommended.

- The resulting preconditioner $M_2M_1 = (L_2D_2U_2)(L_1D_1U_1)$ can be not only stable but also accurate on solving some highly indefinite matrices. This claim will be supported by experiments with the two-phase preconditioning constructed from the two shifted matrices. In short, the shifting method first enhances the stability of the factorization [51], and then the two-phase preconditioning improves the accuracy of the preconditioner.

5.3 Numerical Experiments

We present numerical experiments of the SFAPINV (for stabilized factored approximate inverse) preconditioner on solving indefinite and unsymmetric matrices. The description of the test matrices is given in Table 5.1. The test matrices[1] were solved as they were, that is, no scalings or permutations were applied. The SFAPINV preconditioner was used as a right preconditioner for experiments, but it can be used as a left preconditioner as well since there was not much difference in preconditioning effect. The preconditioned iterative solver employed was GMRES(50). For all linear systems, the right-hand side was generated by assuming that the solution is a vector of all ones. The initial guess was a zero vector. The iteration was terminated when the l_2-norm of the initial residual was reduced by at least eight orders of magnitude, or when the number of iterations reached 500. The programs of our approach were coded in standard Fortran 77 programming language in double precision with 64-bit arithmetic. The computations were carried out on a Sun-Blade-100 workstation with a 500 MHz UltraSPARC III CPU and 1 GB of RAM.

In all tables with numerical results, "iter," "comp," "solu," "cond," "α_1," and "α_2" denote the number of GMRES iterations, the CPU time in seconds for computing the preconditioner, the CPU time for the solution phase (both GMRES and preconditioner), the condest[2], the first and second shifting parameters, respectively. "τ_1," "τ_2," and "$\tau = \tau_1 * \tau_2$" are the dropping tolerances. The value "-1" and "-3" indicate the failure of convergence within the maximum number of the allowed iterations (500) and with the GMRES solver breakdown, respectively.

[1]All of these matrices are available online from the Matrix Market of the National Institute of Standards and Technology at http://math.mist.gov/matrixMarket.

[2]A statistic, condest, is introduced by Chow and Saad [19] to measure the stability of triangular solutions.

Matrix	Description	n	nnz	$nnzdiag$	$condition$
FIDAP007	Matrices generated by the FIDAP Package	1633	46570	1633	1.93E+11
FIDAP014	Matrices generated by the FIDAP Package	3251	65747	2351	2.62E+16
FIDAP033	Matrices generated by the FIDAP Package	1733	20315	1733	1.20E+13
GRE_512	Simulation of computer systems	512	1976	296	3.8E+02
IMPCOL_B	Chemical engineering plant models Cavett's process	59	271	17	2.7E+05
NNC1374	Nuclear reactor models	1374	8588	870	1.0E+02
NNC261	Nuclear reactor models	261	1500	150	1.2E+15
NNC666	Nuclear reactor models	666	4044	410	1.8E+11
PSMIGR_2	Inter-county migration US Inter-county migration 1965-1970	3140	540022	0	1.0E+02
RBS480_A	Forward Kinematics for the Stewart platform of Robotics	480	17088	42	1.3E+05
RBS480_B	Forward Kinematics for the Stewart platform of Robotics	480	17088	43	1.6E+05
RW136	Markov Chain Transition Matrix	136	479	0	1.4E+05
WEST0067	Chemical engineering plant models	67	294	2	3.0E+02

Table 5.1: Description of the test matrices ; n, nnz, $nnzdiag$, and $condition$ denotes the order, the number of nonzero entries, the number of nonzero entries on the main diagonal, and the condition number of a matrix, respectively.

Matrix	n	$nnzdiag$	$condition$	$DD-row$	α_1	α_2
FIDAP007	1633	1633	1.93E+11	0.0245	Yes	Yes
FIDAP014	3251	2351	2.62E+16	0.2125	Yes	Yes
FIDAP033	1733	1733	1.20E+13	0.0848	Yes	Yes
GRE_512	512	296	3.8E+02	0.1328	Yes	No
IMPCOL_B	59	17	2.7E+05	0.1694	Yes	No
NNC1374	1374	870	1.0E+02	0.	Yes	No
NNC261	261	150	1.2E+15	0.	Yes	No
NNC666	666	410	1.8E+11	0.	Yes	No
PSMIGR_2	3140	0	1.0E+02	0.	Yes	No
RBS480_A	480	42	1.3E+05	0.	Yes	No
RBS480_B	480	43	1.6E+05	0.	Yes	No
RW136	136	0	1.4E+05	0.	Yes	No

Table 5.2: The relationship between the matrix properties and the shifting factors.

Shifting Parameters and Dropping Tolerances

We present results arisen from different settings of the shifting parameters and the dropping tolerances. In order to obtain a concrete convergence rate, two different settings of the shifting parameters are used in constructing the preconditioner. Note that the first and the second settings are denoted as $\alpha_1 = \textsc{Find-Alpha}(A, \alpha_1)$ and $\alpha_2 = \textsc{Find-Alpha}(W, \alpha_2)$, and $\alpha_1 = \textsc{Find-Alpha}(A, \alpha_1)$ and $\alpha_2 = 0$, respectively.

We recommend that α_1 and α_2 be chosen to improve the stability of the preconditioner, but for some matrices, α_2 be set to zero to enhance the accuracy. Table 5.2 informs a guideline on choosing the proper setting of the shifting parameters for each matrix to achieve good convergence. Determining the shifting parameters is related with the statistics of each matrix, such as the condition number, diagonally dominant row rate (DD-row), and the number of nonzeros on diagonal. Specifically, when DD-row of a matrix is zero, the second setting to the parameters ($\alpha_2 = 0$) could be chosen due to the need for increased accuracy in the second phase of the preconditioning. As we can see in Table 5.2, for the matrices which have small condition numbers and a large number of zeros on their diagonals, we select the second setting to the parameters ($\alpha_2 = 0$). On the contrary, when DD-row of a matrix is not zero, the two parameters with the first setting ($\alpha_2 \neq 0$) are usually selected to improve stability in the second phase of the preconditioning. In this case, the matrices usually have large condition numbers. For example, The FIDAP matrices, FIDAP007, FIDAP014, and FIDAP033, with the first setting have large condition numbers in the range of [1.93E+11, 2.62E+16] and low DD-rate of [0.0245, 0.2125].

The choice of the first setting is usually acceptable for the FIDAP matrices. Tables 5.3 and 5.4 show that the FIDAP matrices with the first setting increases the accuracy of the preconditioning, and as a result of that the number of GMRES iterations is decreased noticeably. The FIDAP014 matrix converges in around 167 iterations with the first setting (See cases 1 and 4 in Figure 5.1), but with the second, it converges in 436 (460) iterations (See cases 2 and 3 in Figure 5.1). In addition, the preconditioner for solving the FIDAP033 matrix cannot converge in 500 iterations with the second setting while it converges with the first setting. In the second setting, the number of GMRES iterations may be related with the second dropping

Shifting factors		Dropping tolerances			Convergence results			
α_1	α_2	τ_1	τ_2	τ	comp	solu	iter	cond
1.14E+07	5.00E-01	1.0E-01	1.0E-02	1.0E-03	2.30E+00	2.28E+00	167	1.76E-07
1.14E+07	5.00E-01	1.0E-01	1.0E-03	1.0E-04	2.44E+00	2.53E+00	167	1.76E-07
1.14E+07	5.00E-01	1.0E-01	1.0E-04	1.0E-05	2.50E+00	2.81E+00	167	1.76E-07
1.14E+07	5.00E-01	1.0E-02	1.0E-01	1.0E-03	2.32E+00	2.32E+00	180	1.76E-07
1.14E+07	4.95E-01	1.0E-02	1.0E-02	1.0E-04	3.20E+00	2.34E+00	163	1.78E-07
1.14E+07	4.95E-01	1.0E-02	1.0E-03	1.0E-05	3.40E+00	2.55E+00	163	1.78E-07
1.14E+07	4.95E-01	1.0E-03	1.0E-01	1.0E-04	6.10E+00	2.47E+00	169	1.78E-07
1.14E+07	4.95E-01	1.0E-03	1.0E-02	1.0E-05	6.10E+00	2.50E+00	162	1.78E-07
1.14E+07	4.95E-01	1.0E-04	1.0E-01	1.0E-05	1.07E+01	2.72E+00	169	1.78E-07
1.14E+07	0	1.0E-03	1.0E-01	1.0E-04	6.17E+00	7.31E+00	460	8.80E+00
1.14E+07	0	1.0E-01	1.0E-03	1.0E-04	2.75E+00	9.09E+00	436	8.80E+00

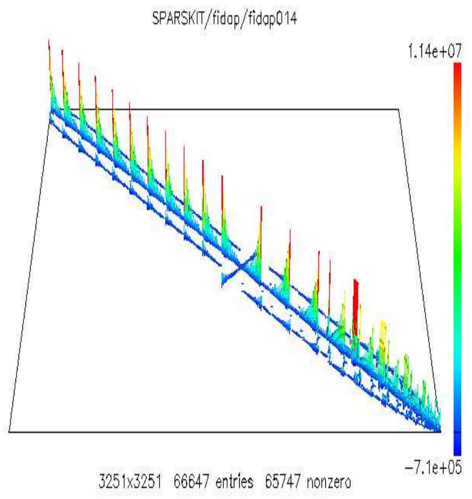
Table 5.3: Test results for solving the FIDAP014 matrix with different values of the shifting factors and the dropping parameters.

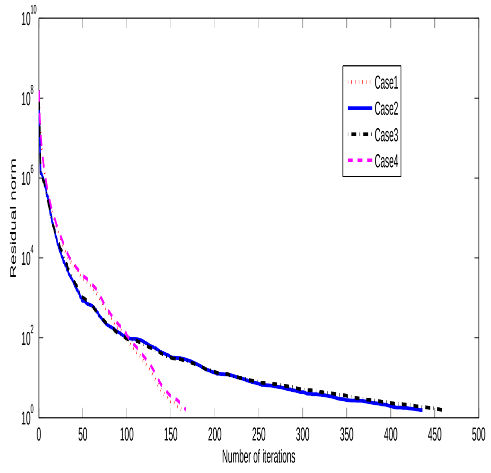
Shifting factors		Dropping tolerances			Convergence results			
α_1	α_2	τ_1	τ_2	τ	comp	solu	iter	cond
1.21E+11	6.07E-01	1.0E-01	1.0E-02	1.0E-03	6.69E-01	1.55E+00	289	1.37E-11
1.21E+11	6.09E-01	1.0E-01	1.0E-03	1.0E-04	7.00E-01	1.87E+00	300	1.37E-11
1.21E+11	6.10E-01	1.0E-01	1.0E-04	1.0E-05	7.09E-01	1.96E+00	300	1.37E-11
1.21E+11	0.75E+00	1.0E-02	1.0E-01	1.0E-03	6.59E-01	1.67E+00	320	1.09E-11
1.21E+11	5.93E-01	1.0E-02	1.0E-03	1.0E-05	6.69E-01	1.60E+00	296	1.49E-11
1.21E+11	5.94E-01	1.0E-02	1.0E-03	1.0E-05	7.20E+00	1.89E+00	300	1.49E-11
1.21E+11	5.91E-01	1.0E-03	1.0E-01	1.0E-04	1.57E+00	1.78E+00	288	1.39E-11
1.21E+11	5.92E-01	1.0E-03	1.0E-02	1.0E-05	1.56E+00	1.89E+00	297	1.50E-11
1.21E+11	5.92E-01	1.0E-04	1.0E-01	1.0E-05	1.65E+00	1.79E+00	288	1.39E-11

Table 5.4: Test results of SFAPINV for solving the FIDAP033 matrix with different values of the shifting factors and the dropping parameters.

Shifting factors		Dropping tolerances			Convergence results			
α_1	α_2	τ_1	τ_2	τ	comp	solu	iter	cond
1.00E+00	0	1.0E-03	1.0E-02	1.0E-05	1.17E+01	3.69E-01	12	2.44E+00
1.00E+00	0	1.0E-02	1.0E-02	1.0E-04	8.17E+00	3.29E-01	13	1.79E+00
1.00E+00	0	1.95E-03	1.0E-02	5.35E-03	1.05E+01	4.40E-01	15	4.19E+00
1.00E+00	0	1.0E-02	1.0E-01	1.0E-03	9.69E+00	7.09E+00	289	1.03E+02
1.00E+00	0	1.0E-02	1.0E-03	1.0E-05	8.08E+00	3.00E-01	12	1.52E+00
1.00E+00	0	1.0E-04	1.0E-01	1.0E-05	1.31E+01	1.07E+01	353	3.09E+02

Table 5.5: Test results of SFAPINV for solving the GRE_512 matrix with different values of the dropping parameters.

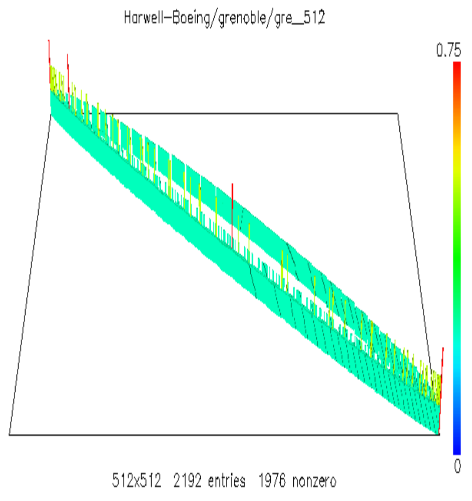


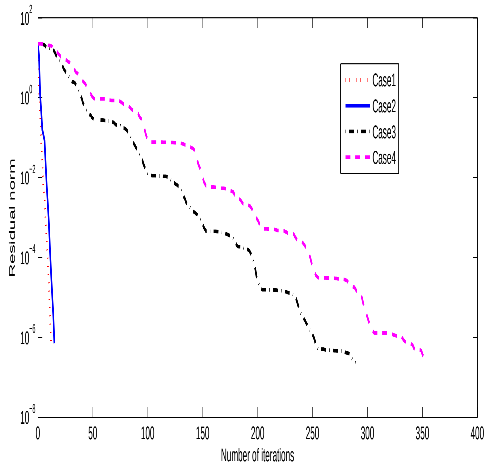


(a) The FIDAP 014 matrix [50] (b) The convergence curves of the FIDAP 014 matrix

Figure 5.1: The FIDAP 014 matrix [50] and the convergence curves with different values of the shifting factors and the dropping parameters.

tolerance. For example, Table 5.5 indicates that the GRE_512 matrix (See Figure 5.2) converges only with the second setting ($\alpha_2 = 0$). The number of iterations vary from 12 to 353 when the dropping tolerances are in the range of [1.0E-05, 1.0E-01]. More





(a) GRE_512 matrix [50] (b) The convergence curves of GRE_512

Figure 5.2: The GRE_512 matrix [50] and the convergence curves with different values of the dropping parameters.

specifically, when $\tau_2 = 0.1$, the number of iterations are 289 and 353 (See cases 3 and

4 in Figure 5.1) while when $\tau_2 \neq 0.1$, the iterations are from 12 to 15 (See cases 1 and 2 in Figure 5.1). We found that for the matrix GRE_512, the number of iterations become large when the second dropping tolerance sets at 0.1.

Comparison between SFAPINV and FAPINV

The comparisons of the SFAPINV (for stabilized factored approximate inverse) preconditioner with the FAPINV (for factored approximate inverse) [69] preconditioner are presented in Table 5.6. Under "N/A", we report that the preconditioner was not constructed, due to zeros on the diagonal (zero pivot). In each testing, the dropping tolerances were carefully chosen to keep the memory cost (sparsity ratio) of these two preconditioners comparable. Table 5.7 is a list of the parameters used in both the SFAPINV and FAPINV preconditioners for the test matrices.

	SFAPINV				FAPINV			
Matrix	comp	solu	iter	cond	comp	solu	iter	cond
FIDAP007	2.56E+00	1.56E+00	153	8.96E-10	1.79E+01	6.78E+01	-1	3.71E+03
FIDAP014	6.10E+00	2.50E+00	162	1.78E-07	9.64E+01	3.30E+02	-1	7.17E+09
FIDAP033	1.57E+00	1.78E+00	288	1.39E-11	8.09E+00	4.59E+01	-1	6.83E+01
GRE_512	1.17E+01	3.69E-01	12	2.44E+00	N/A	N/A	-3	N/A
IMPCOL_B	9.99E-03	9.99E-03	27	7.57E+03	N/A	N/A	-3	N/A
PSMIGR_2	1.92E+03	1.69E+01	19	1.49E+03	N/A	N/A	-3	N/A
NNC1372	3.61E+00	2.63E+00	46	9.57E+04	3.71E+00	3.02E+01	-1	6.59E+35
NNC261	7.00E-02	8.99E-02	44	5.99E+04	5.00E-02	1.05E+00	-1	2.93E+22
NNC666	5.60E-01	2.99E-01	20	1.06.E+05	4.39E-01	6.62E+00	-1	1.23E+12
RBS480_A	5.41E+00	4.20E-01	22	2.93E+01	N/A	N/A	-3	N/A
RBS480_B	5.40E+00	3.19E-01	16	1.34E+00	4.26E+00	8.09E+00	-1	1.22E+03
RW136	1.60E-02	7.99E-03	20	2.24E+10	3.99E-03	1.24E-01	-1	2.69E+15
WEST0067	2.99E-02	0.00E+00	5	1.14E+01	N/A	N/A	-3	N/A

Table 5.6: Comparisons of SFAPINV and FAPINV.

As shown in Table 5.6, in most cases, the SFAPINV preconditioner performed better than the FAPINV preconditioner. The test matrices that were not solved by the FAPINV preconditioner may be difficult to solve by the ILU preconditioners, such as ILU(0), ILUT [54]. The data in the table also demonstrates that the construction cost of the preconditioners is quite inexpensive. For the case of the FIDAP014 matri-

	SFAPINV					FAPINV
	Shifting factors		Dropping tolerances			Dropping tolerance
Matrix	α_1	α_2	τ_1	τ_2	τ	τ
FIDAP007	2.00E+09	5.64E-01	1.0E-03	1.0E-02	1.0E-05	1.0E-03
FIDAP014	1.14E+07	4.95E-01	1.0E-03	1.0E-02	1.0E-05	1.0E-03
FIDAP033	1.21E+11	5.91E-01	1.0E-03	1.0E-01	1.0E-04	1.0E-03
GRE_512	1.00E+00	0	1.0E-03	1.0E-02	1.0E-05	1.0E-03
IMPCOL_B	1.27E+01	0	1.0E-04	1.0E-01	1.0E-05	1.0E-04
NNC1374	3.56E+03	0	1.0E-01	1.0E-04	1.0E-05	1.0E-01
NNC261	3.56E+03	0	1.0E-01	1.0E-04	1.0E-05	1.0E-01
NNC666	3.56E+03	0	1.0E-01	1.0E-04	1.0E-05	1.0E-01
PSMIGR_2	1.00E+00	0	3.18E-04	1.0E-02	2.79E-05	3.18E-04
RBS480_A	5.79E+03	0	1.0E-03	1.0E-02	1.0E-05	1.0E-03
RBS480_B	6.16E+03	0	1.0E-03	1.0E-02	1.0E-05	1.0E-03
RW136	1.57E+00	0	1.0E-03	1.0E-02	1.0E-05	1.0E-03
WEST0067	6.14E+00	0	1.0E-03	1.0E-02	1.0E-05	1.0E-03

Table 5.7: Parameters used in SFAPINV and FAPINV.

ces, the construction cost of the SFAPINV preconditioner was 16 times cheaper than that of the FAPINV preconditioner. In each phase, an sparse matrix (approximation) could be computed with low memory cost. As a result, the total computational cost of these sparse matrices becomes cheaper compared with computing a single sparse matrix with a denser sparsity pattern.

We note that the SFAPINV preconditioner does not converge if the first shifting parameter α_1 is set to zero (result is not shown in this paper). Thus, a nonzero value for the first shifting parameter is necessary for the two-phase preconditioning of the shifting method to achieve good performance.

Computational Analysis

Again, in order to analyze the convergence behavior of the Krylov subspace methods, we measure the distribution of the eigenvalues and the condition number of the eigenvector matrix and the coefficient matrix [39, 55].

As we can see in Table 5.8 and Figure 5.3 (a), the condition number of the original matrix A (FIDAP014) is relatively large, and the eigenvalues of the original matrix are distributed near the origin and far away from zero. The FAPINV preconditioned

Table 5.8: The largest and smallest eigenvalues and conditioning information of the FIDAP014 matrix.

Matrix	$\lambda_{largest}$	$\lambda_{smallest}$	cond(Matrix)	cond(X)
A	1.27E+07	-8.55E-10	2.16E+16	1.00E+00
FAPINV(A)	1.02E+46	-6.37E-12 + 1.87E-11i	1.96E+57	1.36E+21
SFAPINV(A)	0.5261	-1.52E-16	3.468E+15	5.81E+03

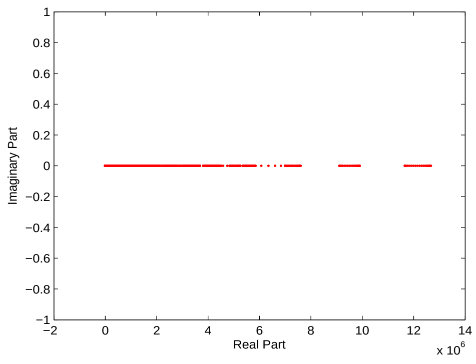
matrix (FAPINV(A)) made the spectrum more spread (See Figure 5.3 (b)), and the condition number of the eigenvector matrix and preconditioned matrix became larger than the original matrix. As a result, on the original matrix and the FAPINV preconditioned matrix, the convergences of the GMRES method are failed.

It has been known that eigenvalues tightly clustered around a single point (away from the origin) provide fast convergence, while widely spread eigenvalues, especially around the origin, cause the convergence to be very slow [39, 55].

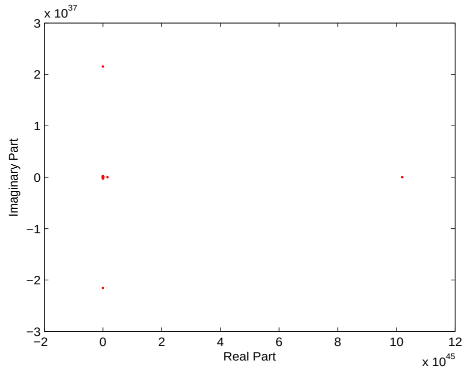
Unlike the FAPINV preconditioned matrix, the eigenvalues of the SFAPINV preconditioned matrix (SFAPINV(A)) became more clustered as clearly shown in Figure 5.3 (c). In addition, the largest and the smallest eigenvalue in magnitude of the SFAPINV(A) are 0.5261 and -1.52E-16, and the condition numbers of the eigenvector matrix of SFAPINV(A) is decreased. These improvements support a good convergence behavior of the GMRES method on the SFAPINV preconditioned matrix.

5.4 Concluding Remarks

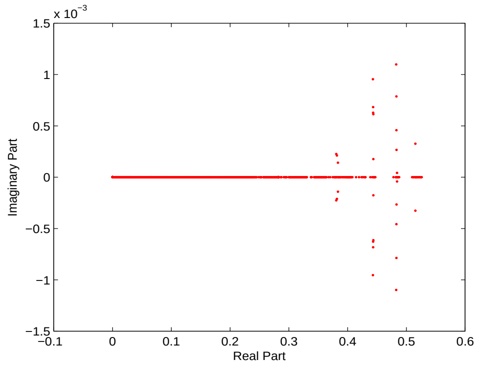
We proposed an algorithm called SFAPINV (for stabilized factored approximate inverse) for solving highly indefinite matrices. SFAPINV consists of two preconditioning phases to two shifted matrices. Each phase is a factorization of a shifted matrix of the form $A + \alpha I$. The shifting method is employed to prevent unstable factorization due to the small or zero pivots of the original matrix, and the two-phase preconditioning is utilized to improve the accuracy of the preconditioner. FAPINV [69] has been utilized as a local preconditioner in the SFAPINV preconditioning because of



(a) Eigenvalues of the FIDAP014 matrix



(b) Eigenvalues of the FIDAP matrix with FAPINV



(c) Eigenvalues of the FIDAP014 matrix with SFAPINV

Figure 5.3: Plots of the eigenvalues of the FIDAP014 matrix. (These subfigures are in different scale.)

its low construction cost and robustness.

Numerical experiments demonstrate the stability and accuracy of the SFAPINV preconditioner. Moreover, for the NNC1372 and FIDAP matrices, the construction cost of the SFAPINV preconditioner may be cheaper than that of a single sparse matrix in the FAPINV preconditioner although the SFAPINV preconditioner is composed of two sparse matrices, M_1 and M_2. In fact, the memory cost of each sparse matrix is small and each of the sparse approximate inverse matrices could be computed cheaply. According to [63], computing a series of two matrices where each matrix needs small memory cost may be cheaper than that of a matrix that needs high memory cost. Thus, the SFAPINV preconditioner could be robust and fast on solving indefinite matrices.

We remark that the concept of the two-phase preconditioning of shifted matrices may be applied to other preconditioning techniques. For example, one can construct a hybrid two-phase preconditioning by using a different preconditioner in each phase. Also, the presented algorithm can be extended to a parallel version because FAPINV naturally possesses of great inherent parallelism.

Chapter 6 Factored Approximate Inverse Preconditioners with Dynamic Sparsity Patterns

The search for an optimal sparsity pattern in constructing sparse approximate inverse (SAI) preconditioners should be taken with extreme care because the performance of SAI preconditioners depends on the sparsity pattern [69]. In this chapter, we present two dynamic sparsity pattern selection algorithms for FAPINV (factored approximate inverse) preconditioners in solving general sparse matrices. The sparsity pattern is adaptively updated in the construction phase by using combined information of the inverse and original triangular factors of the original matrix.

6.1 Introduction

The computational efficiency and potential parallelism of sparse approximate inverse (SAI) leads us to choose SAI preconditioners as an alternative to the conventional ILU preconditioners. It has been well known that the performance of SAI preconditioners depends on their sparsity patterns [63, 69]. Thus, obtaining an optimal sparsity pattern in constructing an SAI preconditioner can be the most important part. Generally speaking, there exist two main classes of methods called static and dynamic strategies in determining the sparsity pattern of SAI. A static sparsity pattern strategy prescribes the sparsity pattern before constructing a preconditioner, and the pattern remains unchanged until finishing the construction. The use of a prescribed (fixed) pattern through the construction process assumes that this class of sparsity pattern selection strategy is usually efficient in terms of computational cost. But, for general sparse matrices, the prescribed sparsity pattern is often inadequate for robustness of SAI [69]. On the contrary, a dynamic sparsity pattern strategy adjusts the pattern using some rules in the construction phase. Such a strategy usually computes more accurate and robust preconditioners than a static strategy [63]. Thus, a dynamic sparsity pattern strategy may be used to substitute for a static sparsity pattern strategy in solving difficult matrices.

In recent years, a few dynamic pattern strategies [15, 41] for SAI preconditioners have been developed. For example, Bollhöfer [14] showed that the norms of the inverse triangular factors have direct influence on the dropping strategy in computing a new ILU decomposition. Based on this insight, Bollhöfer [15] proposed an algorithm that manages the process of dropped entries with small absolute values by using the row norm of any row of the inverse factors, but the algorithm has a limitation on solving some ill-conditioned problems.

As a part of our continuous efforts in determining dynamic sparsity pattern, we introduce two enhanced algorithms, which extend the algorithm [15] mentioned above, using combined information of the norm of the inverse factors and either the largest absolute value of the original factors or the norm of the original factors. Here, a factored approximate inverse (FAPINV) [69], which is a sparse approximate inverse with a factored form, is utilized as an SAI.

The remainder of this chapter is organized as follows. Section 6.2 describes the FAPINV algorithm used to characterize and justify our algorithms for dynamic sparsity pattern in Section 6.3. In Section 6.4, numerical results are presented to demonstrate the performance of the proposed algorithms over a static pattern based FAPINV. Concluding remarks are in Section 6.5.

6.2 Factored Sparse Approximate Inverses

In this section, we first review a general framework of a factored inverse and an incomplete factored inverse algorithms, and this framework will be utilized in Section 6.3 to justify our sparsity pattern algorithms.

Zhang [69] proposed an algorithm of a factored inverse for sparse matrices, based on the Luo's algorithm [47], in the form,

$$A^{-1} = LDU, \tag{6.1}$$

for a nonsingular matrix A of order n, where $L = [L_1, L_2, \cdots, L_n]$ is a lower triangular matrix of order n, and L_i is a column vector of length n with $L_{i,i} = 1, i = 1, 2, \cdots, n$. Similarly, $U = [U_1, U_2, \cdots, U_n]^T$ is an upper triangular matrix with a row vector U_i and $U_{i,i} = 1, i = 1, 2, \cdots, n$, and $D = \mathrm{d}iag[D_{1,1}, D_{2,2}, \cdots, D_{n,n}]$ denotes a diagonal matrix. The factored inverse algorithm that computes the inverse matrix, A^{-1}, of Equation (6.1) is presented in Algorithm 6.2.1.

Algorithm 6.2.1. FACTORED INVERSE ALGORITHM FOR DENSE MATRICES [47].

```
1.  For $j = n \to 1$ with step (-1)
2.      For $i = j+1, n$
3.          $w_i^{(j)} = a_{j,i} + \sum_{k=i+1}^{n} a_{j,k} * L_{k,i}$
4.      For $i = j+1, n$
5.          $U_{j,i} = -w_i^{(j)} * D_{i,i} - \sum_{k=i+1}^{n} w_k^{(j)} * D_{k,k} * U_{k,i}$
6.      $D_{j,j} = 1/(a_{j,j} + \sum_{k=j+1}^{n} U_{j,k} * a_{k,j})$
7.      For $i = j+1, n$
8.          $z_i^{(j)} = a_{i,j} + \sum_{k=i+1}^{n} U_{i,k} * a_{k,j}$
9.      For $i = j+1, n$
10.         $L_{i,j} = -z_i^{(j)} * D_{i,i} - \sum_{k=i+1}^{n} z_k^{(j)} * D_{k,k} * L_{i,k}$
```

Note that n refers to the order of the original matrix A, and a_{ij} denotes a nonzero element in row i and column j, where $i, j = 1, \cdots, n$. The upper inverse factor U and the lower inverse factor L are computed in lines 2–7 and lines 9–14, respectively, and the diagonal inverse is constructed in line 8. Although Algorithm 6.2.1 is written to compute the inverse of a dense matrix, it can be easily modified to compute the inverse of a sparse matrix [69].

For a sparse matrix, we can see that the L and U matrices computed by Algorithm 6.2.1 can be too dense, and the computational cost might be high [69]. Thus, an incomplete inverse method that drops elements with small absolute values of the inverse is usually employed to reduce the cost and maintain the sparsity of the inverse factors. For example, Zhang [69] introduced an incomplete FAPINV algorithm that applies a static dropping strategy in two parts (four places) of the computational process to improve Algorithm 6.2.1. Specifically, for a given dropping tolerance $\tau > 0$, the conditions $|w_i^{(j)}| \leq \tau$ and $|z_i^{(j)}| \leq \tau$ determine the loop involving lines 2–4 and 9–11 to be skipped, respectively. After processing lines 5–7 and 12–14, $U_{j,i}$ and $L_{i,j}$ are dropped if their absolute values are smaller than or equal to τ.

Definition 6.2.1. *A matrix A is said to be an M-matrix if it satisfies the following four properties:*

1. $a_{i,i} > 0$ *for* $i = 1, \cdots, n$.

2. $a_{i,j} \leq 0$ *for* $i \neq j$, $i, j = 1, \cdots, n$.

3. A *is nonsingular.*

4. $A^{-1} \geq 0$.

According to Maijerink and van der Vorst [52], a conventional incomplete LU factorization can be executed in an exact factorization, and the computed pivots are strictly positive when A is a nonsingular M-matrix. This holds true in FAPINV associated with the inverse computed from Algorithm 6.2.1, and in the following, Proposition 6.2.2 establishes a proof that is similar to Proposition 3.1 in [11] where no breakdown in the incomplete process occurs if A is an M-matrix.

Proposition 6.2.2. *Let A be an M-matrix, and $A^{-1} = LDU$ be an inverse matrix of A obtained by Algorithm 6.2.1, where $L, D,$ and U are lower, diagonal, and upper inverse factors of A, respectively. Then*

$$U_{j,i} \geq 0, \quad L_{i,j} \geq 0 \quad \textit{and} \quad D_{j,j} > 0 \tag{6.2}$$

for $1 \leq i, j \leq n$.

Proof. We will prove the inequalities in (6.2) by using induction. For $j = n$, the inequalities are obviously true. In fact,

$$L_{n,n} = U_{n,n} = 1 \text{ and } D_{n,n} = 1/a_{n,n} \geq {a_{n,n}}^{-1} > 0.$$

Now, the following inductive assumptions are considered for $j \leq n-1$ and $j+2 \leq i \leq n$.

$$U_{j+1,i} = -w_i^{(j+1)} * D_{i,i} - \sum_{k=j+2}^{n} w_k^{(j+1)} * D_{k,k} * U_{k,i} \geq 0, \tag{6.3}$$

$$L_{i,j+1} = -z_i^{(j+1)} * D_{i,i} - \sum_{k=j+2}^{n} z_k^{(j+1)} * D_{k,k} * L_{i,k} \geq 0, \quad \text{and} \tag{6.4}$$

$$D_{j+1,j+1} = 1/(a_{j+1,j+1} \; + \sum_{k=j+2}^{n} U_{j+1,k} * a_{k,j+1}) > 0. \tag{6.5}$$

From the inductive assumptions (6.3) and (6.4), we get $U_{i,k} \geq 0$ and $L_{k,i} \geq 0$ for $i+1 \leq k \leq n$. Also, $a_{j,i}$ and $a_{i,j}$ are nonpositive since A is an M-matrix. Hence, $w_i^{(j)}$ and $z_i^{(j)}$ can be expressed by

$$w_i^{(j)} = a_{j,i} + \sum_{k=i+1}^{n} a_{j,k} * L_{k,i} \leq 0 \;\text{ and }\; z_i^{(j)} = a_{i,j} + \sum_{k=i+1}^{n} U_{i,k} * a_{k,j} \leq 0, \tag{6.6}$$

where $j+1 < i < n$. Using the updating formula given in Algorithm 6.2.1 for $U_{j,i}, L_{i,j},$ and $D_{j,j}$, we have

$$U_{j,i} = -w_i^{(j)} * D_{i,i} - \sum_{k=j+1}^{i-1} w_k^{(j)} * D_{k,k} * U_{k,i} \geq 0,$$

$$L_{i,j} = -z_i^{(j)} * D_{i,i} - \sum_{k=j+1}^{i-1} z_k^{(j)} * D_{k,k} * L_{k,i} \geq 0,$$

and

$$D_{j,j} = 1/(a_{j,j} + \sum_{k=j+1}^{n} U_{j,k} * a_{k,j}) > 0.$$

Therefore, no component of $U_{j,i}$ and $L_{i,j}$ can become negative, and $D_{j,j}$ remains positive. ■

Incompleteness arises from FAPINV due to the dropping of nonzero fill-ins and causes the algorithm to produce smaller or equal absolute entries than those of the inverse factorization from Algorithm 6.2.1. In accordance with such facts and the result of Proposition 6.2.2, we can see that FAPINV also generates a nonnegative factored approximate inverse when A is an M-matrix.

Proposition 6.2.3. *Let A be an M-matrix, and $A^{-1} = LDU$ be an inverse matrix of A obtained by Algorithm 6.2.1, where $L, D,$ and U are lower, diagonal, and upper inverse factors of A, respectively. If $G = \tilde{L}\tilde{D}\tilde{U}$ is a factored approximate inverse computed by FAPINV [69], where $\tilde{L}, \tilde{D},$ and $\tilde{U}$ are lower, diagonal, and upper inverse factors, respectively. Then*

$$D_A^{-1} \leq G \leq A^{-1}, \tag{6.7}$$

where D_A is the diagonal part of A.

Proof. Since the elements of $\tilde{L}$, and $\tilde{U}$ are obtained from L and U by dropping elements if the absolute value of the elements are smaller than a dropping tolerance τ, then

$$\tilde{L} \leq L, \quad \tilde{D} \leq D, \text{ and } \tilde{U} \leq U.$$

Hence, we can easily get $G \leq A^{-1}$.

From Proposition 6.2.2, we know that

$$D_{j,j} > 0, \quad a_{j,j} > 0, \quad U_{j,k} \geq 0, \text{ and } a_{k,j} \leq 0.$$

It follows that

$$0 < a_{j,j} + \sum_{k=j+1}^{n} U_{j,k} * a_{k,j} \leq a_{jj}.$$

Therefore,

$$D_{j,j} \geq 1/a_{j,j}.$$

■

As the results of Proposition 6.2.2 and Proposition 6.2.3, we have found that both the exact factored inverse computed by Algorithm 6.2.1 and FAPINV [69] construct nonnegative inverses when A is an M-matrix.

6.3 Dynamic Sparsity Pattern based Factored Approximate Inverse Preconditioners

We now introduce two algorithms that determine dynamic sparsity patterns for FAPINV preconditioners on solving general sparse matrices. In determining the sparsity pattern, we exploit information of the inverse and original factors by the following two supporting reasons that (1) the entries of FAPINV are computed by the original and previously computed inverse triangular factors [69], and (2) the norm of the inverse factor is strongly related with the dropping tolerance of FAPINV [14]. From that point of view, (1) our first algorithm, Norm-Largest-Dynamic sparsity pattern in Algorithm 6.3.1, computes FAPINV with the dynamic sparsity pattern using the norm of the inverse factors multiplied by the largest absolute value of the original factors, and (2) the second, Norm-Norm-Dynamic sparsity pattern Algorithm 6.3.2, employs the norm of the inverse factors divided by the norm of the original factors.

Algorithm 6.3.1. FAPINV WITH NORM-LARGEST-DYNAMIC SPARSITY PATTERN.

```
1.  Find the largest absolute values, $Large_L$ and $Large_U$,
        of the lower and upper parts of $A$
2.  For $j = n, 1$ with step (-1)
3.      $\zeta_U(j) = \zeta_L(j) = 0$
4.      For $i = j+1, n$
5.          Compute $U_{j,i}$ by using FAPINV [69]
6.          If $(|U_{j,i}| > \zeta_U(j))$, then $\zeta_U(j) = |U_{j,i}|$
7.      $\eta_U = \zeta_U(j) * Large_U$
8.      If $(\eta_U > 1)$, then $\tau = \tau/\eta_U$
9.      For $i = j+1, n$
10.         If $(|U_{j,i}| < \tau)$, then $U_{j,i} = 0$
11.     $D_{j,j} = 1/(a_{j,j} + \sum_{k=j+1}^{n} U_{j,k} * a_{k,j})$
12.     For $i = j+1, n$
13.         Compute $L_{i,j}$ by using FAPINV [69]
14.         If $(|L_{i,j}| > \zeta_L(j))$, then $\zeta_L(j) = |L_{i,j}|$
15.     $\eta_L = \zeta_L(j) * Large_L$
16.     If $(\eta_L > 1)$, then $\tau = \tau/\eta_L$
17.     For $i = j+1, n$
18.         If $(|L_{i,j}| < \tau)$, then $L_{i,j} = 0$
```

Note that $Large_L$ and $Large_U$ refer to the largest absolute values of the lower and upper triangular factors of the original matrix, respectively. The $\zeta_U(j)$ and $\zeta_L(j)$ in lines 6 and 15 denote the largest absolute values of the columns U_j^T and L_j, respectively. The upper inverse factor U and the lower inverse factor L are computed in lines 4–12 and lines 14–22, respectively, and the diagonal inverse D is constructed in line 13. In lines 5 and 15, $U_{j,i}$ and $L_{i,j}$ can be obtained by using the FAPINV algorithm [69]. In lines 9 and 19, the dropping tolerance τ is determined by η_U and η_L, and the value of τ is updated for each j. Finally, in lines 10–12 and 20–22, if the absolute value of an element is smaller than the tolerance τ, the element is then dropped.

Algorithm 6.3.2. FAPINV WITH NORM-NORM-DYNAMIC SPARSITY PATTERN.

```
1.  For $i = 1, n$
2.      $U_\eta(i)$ = the largest absolute value in the row $i$ of the upper triangular factor of $A$.
3.      $L_\eta(i)$ = the largest absolute value in the row $i$ of the lower triangular factor of $A$.
5.  For $j = n, 1$ with step (-1)
6.      $\zeta_U(j) = \zeta_L(j) = 0$
7.      For $i = j+1, n$
8.          Compute $U_{j,i}$ by using FAPINV [69]
9.          If ($|U_{j,i}| > \zeta_U(j)$), then $\zeta_U(j) = |U_{j,i}|$
11.     $\eta_U = \zeta_U(j)/U_\eta(j)$
12.     If ($\eta_U > 1$), then $\tau = \tau/\eta_U$
13.     For $i = j+1, n$
14.         If ($U_{j,i} < \tau$), then $U_{j,i} = 0$
16.     $D_{j,j} = 1/(a_{j,j} + \sum_{k=j+1}^{n} U_{j,k} * a_{k,j})$
17.     For $i = j+1, n$
18.         Compute $L_{i,j}$ by using FAPINV [69]
19.         If ($|L_{i,j}| > \zeta_L(j)$), then $\zeta_L(j) = |L_{i,j}|$
21.     $\eta_L = \zeta_L(j)/L_\eta(j)$
22.     If ($\eta_L > 1$), then $\tau = \tau/\eta_L$
23.     For $i = j+1, n$
24.         If ($L_{i,j} < \tau$), then $L_{i,j} = 0$
```

As similar to Algorithm 6.3.1, the upper inverse factor U and the lower inverse factor L are computed in lines 7–15 and lines 17–25, respectively, and the diagonal inverse D is constructed in line 16.

Let G_1 and G_2 be two FAPINV preconditioners of a matrix A. If a given dropping tolerance of G_1 is greater than or equal to that of G_2, then

$$L_{1i,j} \leq L_{2i,j}, \quad U_{1j,i} \leq U_{2j,i}, \quad \text{and} \quad D_{1j,j} \leq D_{2j,j},$$

where $G_1 = L_1D_1U_1$ and $G_2 = L_2D_2U_2$. Also, from Proposition 6.2.3, we know that G_1 and G_2 are nonnegative matrices if A is an M-matrix. Thus, the following proposition is straightforward.

Proposition 6.3.1. *Let A be an M-matrix and $A^{-1} = LDU$ be the inverse matrix of A. If $G_1 = L_1D_1U_1$ is obtained by FAPINV [69] with a static dropping tolerance τ_1, $G_2 = L_2D_2U_2$ is from Algorithm 6.3.1 or Algorithm 6.3.2 with a dynamic dropping tolerance τ_2, and $\tau_1 \geq \tau_2$. Then*

$$D_A^{-1} \leq G_1 \leq G_2 \leq A^{-1}, \tag{6.8}$$

where D_A is the diagonal part of A.

Proof. Based on the Algorithm 6.3.1 and Algorithm 6.3.2, τ_2 is determined by τ_1. Specifically, in computing L and U, $\tau_2 = \tau_1/\eta_L$ and $\tau_2 = \tau_1/\eta_U$ where $\eta_L > 1$ and $\eta_U > 1$, respectively. Because of $\tau_1 \geq \tau_2$ and by Proposition 6.2.2,

$$0 \leq L_{1i,j} \leq L_{2i,j}, \quad 0 \leq U_{1j,i} \leq U_{2j,i}, \quad \text{and} \quad 0 < D_{1j,j} \leq D_{2j,j}.$$

It follows that $G_1 \leq G_2$. ∎

As we can see in Proposition 6.3.1, the accuracy of FAPINV depends on the value of the dropping tolerance τ. This implies that the FAPINV preconditioners with the dynamic sparsity patterns become more accurate than the FAPINV with a static sparsity pattern. We present the comparisons between a static and dynamic pattern for the FAPINV preconditioners in Table 6.4.

6.4 Numerical Experiments

We present numerical experiments of FAPINV with the proposed selection strategies of dynamic sparsity patterns on solving a few general sparse matrices. The descriptions of the test matrices are given in Table 6.4. The matrices[1] were solved as they were, that is, no scalings or permutations were applied.

The FAPINV (for factored approximate inverse) [69] preconditioner was used as an approximate inverse right preconditioner in the experiments. The preconditioned iterative solver employed was GMRES(50). For all linear systems, the right-hand side was generated by assuming that the solution is a vector of all ones. The initial guess was a zero vector. The iteration was terminated when the l_2-norm of the initial residual was reduced by at least eight orders of magnitude, or when the number of iterations reached 500. The programs of our approaches were coded in standard Fortran 77 programming language in double precision with 64-bit arithmetic, and the computations were carried out on a Sun-Blade-100 workstation with a 500 MHz UltraSPARC III CPU and 1 GB of RAM.

In all tables with numerical results, STATIC denotes an FAPINV [69] preconditioner with a static sparsity pattern. NLD and NND represent FAPINV preconditioners with Norm-Largest-Dynamic and Norm-Norm-Dynamic sparsity patterns described in Algorithm 6.3.1 and Algorithm 6.3.2, respectively; The "iter" refers to the number of GMRES iterations, the "time" represents the CPU time in seconds for computing the preconditioner and for the solution phase, and "spar" denotes the sparsity ratio that is the ratio of the number of nonzero elements of the preconditioner to that of the original matrix; The dropping tolerance "τ" is utilized as both a dropping tolerance of STATIC and initial dropping tolerances of NLD and NND; The value "-1" indicates the failure of convergence within the maximum number of

[1]All of these matrices are available on-line from the Matrix Market of the National Institute of Standards and Technology at http://math.mist.gov/matrixMarket.

Table 6.1: Description of the test matrices ; n, nnz, $nnzdiag$, and *condition* denotes the order, the number of nonzero entries, the number of nonzero entries on the main diagonal, and the condition number of a matrix, respectively. Under "N/A", we report that the condition number was not available from Matrix Market [50].

Matrix	Description	n	nnz	$nnzdiag$	*condition*
CAVITY03	Driven Cavity Problems	317	7311	243	3.30E+06
CAVITY04	Driven Cavity Problems	317	5923	243	1.40E+07
CAVITY06	Driven Cavity Problems	1182	32747	883	N/A
CAVITY08	Driven Cavity Problems	1182	32747	883	N/A
CAVITY11	Driven Cavity Problems	2597	76367	1923	N/A
CAVITY13	Driven Cavity Problems	2597	76367	1923	N/A
CAVITY15	Driven Cavity Problems	2597	76367	1923	N/A
E05R0300	Driven cavity driven cavity, 5x5 elements, Re= 300	236	5856	162	1.30E+06
E05R0400	Driven cavity driven cavity, 5x5 elements, Re= 400	236	5846	162	2.40E+06
E05R0500	Driven cavity driven cavity, 30x30 elements, Re=500	9661	306002	6962	1.31E+11
FIDAP006	Matrices generated by the FIDAP Package	1651	49479	1180	3.45E+21
FIDAP020	Matrices generated by the FIDAP Package	2203	69579	1603	5.89E+08
FIDAP021	Matrices generated by the FIDAP Package	656	18962	476	9.10E+08
FIDAP025	Matrices generated by the FIDAP Package	848	24261	608	7.90E+07
FIDAP026	Matrices generated by the FIDAP Package	2163	93749	1706	4.66E+18
FIDAPM02	Matrices generated by the FIDAP Package	537	19145	441	1.40E+05
LNS 131	Fluid flow modeling	131	536	112	1.50E+15
NNC261	Nuclear reactor models	261	1500	150	1.20E+15
NNC666	Nuclear reactor models	666	4032	410	1.80E+11

allowed iterations (500).

Comparison of Preconditioners with Static and Dynamic Pattern

Table 6.4 shows the comparisons between a static and dynamic sparsity pattern for the FAPINV preconditioners with two different settings of the dropping tolerance, $\tau = 0.1$ and $\tau = 0.01$.

Table 6.2: Comparisons of the number of preconditioned GMRES iterations with different sparsity pattern based FAPINV preconditioners.

	$\tau = 0.1$			$\tau = 0.01$					
	STATIC	NLD	NND	STATIC		NLD		NND	
Matrix	iter	iter	iter	iter	spar	iter	spar	iter	spar
CAVITY03	-1	37	41	-1	8.32	19	9.16	36	8.95
CAVITY04	-1	25	22	-1	6.19	10	7.35	23	8.35
CAVITY06	-1	-1	22	-1	7.07	-1	15.55	15	30.96
CAVITY08	-1	-1	40	-1	18.02	46	25.71	22	31.48
CAVITY11	-1	-1	31	-1	9.10	-1	25.49	22	65.57
CAVITY13	-1	-1	58	-1	21.57	-1	43.55	33	65.63
CAVITY15	-1	-1	92	-1	44.40	-1	52.39	40	67.95
E05R0300	-1	-1	33	197	5.26	28	7.19	20	8.22
E05R0400	-1	-1	46	-1	6.37	18	8.13	23	8.56
E05R0500	-1	100	47	-1	6.98	15	8.72	23	9.03
FIDAP006	-1	94	295	-1	21.72	9	35.71	37	43.52
FIDAP020	-1	-1	14	-1	18.08	-1	27.85	7	62.47
FIDAP021	-1	-1	27	-1	8.31	-1	13.57	19	20.28
FIDAP025	-1	-1	7	-1	16.21	-1	19.83	4	28.82
FIDAP026	-1	452	220	-1	9.033	191	21.83	124	34.28
FIDAPM02	-1	-1	36	-1	6.91	42	10.77	14	13.13
LNS 131	-1	-1	5	-1	2.65	3	5.12	4	9.89
NNC261	-1	15	36	-1	21.23	28	28.52	30	27.06
NNC666	-1	17	44	-1	46.16	17	60.53	28	57.29

As seen in the table, STATIC failed in solving any of the test matrices with the two values of τ except E05R0300 that was solved in 197 iterations when $\tau = 0.01$, whereas NLD and NND solved the matrix in 28 and 20 iterations, respectively. NND solved all the test matrices, while NLD solved 7 and 12 of the test matrices when $\tau = 0.1$ and $\tau = 0.01$, respectively.

In terms of space complexity, NLD and NND demanded more memory space than STATIC on solving the test matrices, but it would seem relatively minor compared to the number of iterations with convergence. For example, NND and NLD solved the CAVITY03, CAVITY04, E05R0500, and NNC666 matrices in 36, 10, 15, and 28 iterations with 1.075, 1.187, 1.249, and 1.241 times more memory storages than STATIC with no convergence, respectively.

Comparison with Strategies of Dynamic Sparsity Pattern

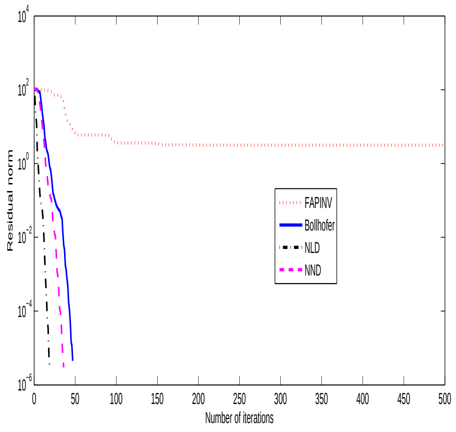
We compare our algorithms, NLD and NND, with the one by Bollhöfer [15] in Table 6.4. Under "N/A," we report that the value is not available due to no convergence,

Table 6.3: Comparisons with different dynamic sparsity pattern strategies.

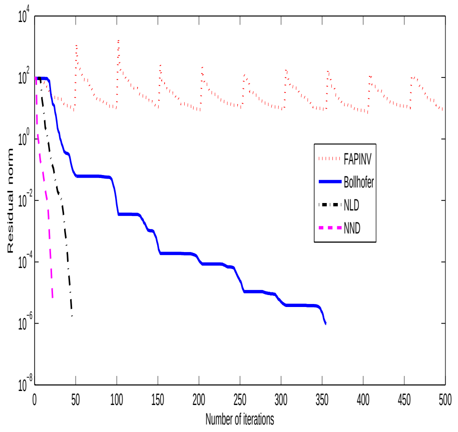
	$\tau = 0.01$								
	NND			Bollhöfer's			NLD		
Matrix	iter	time	spar	iter	time	spar	iter	time	spar
CAVITY03	36	0.53	8.95	47	0.57	8.63	19	0.45	9.16
CAVITY04	23	0.43	8.35	28	0.38	7.24	10	0.32	7.35
CAVITY06	15	16.85	30.96	-1	N/A	14.72	-1	N/A	15.55
CAVITY08	22	17.79	31.48	355	28.73	21.89	46	15.09	25.71
CAVITY11	22	137.27	65.57	-1	N/A	21.21	-1	N/A	25.49
CAVITY13	33	141.67	65.63	-1	N/A	35.02	-1	N/A	43.55
CAVITY15	40	149.86	67.95	-1	N/A	44.92	-1	N/A	52.39
E05R0300	20	0.28	8.22	33	0.26	6.85	28	0.26	7.19
E05R0400	23	0.30	8.56	42	0.33	7.63	18	0.27	8.13
E05R0500	23	0.32	9.03	42	0.35	8.36	15	0.28	8.72
FIDAP006	37	24.17	43.52	46	25.87	43.68	9	15.55	35.71
FIDAP020	7	56.33	62.47	-1	N/A	28.95	-1	N/A	27.85
FIDAP021	19	3.14	20.28	-1	N/A	13.75	-1	N/A	13.57
FIDAP025	4	5.44	28.82	404	19.71	21.76	-1	N/A	19.83
FIDAP026	124	89.99	34.28	-1	N/A	17.84	191	67.39	21.83
FIDAPM02	14	1.73	13.13	46	1.98	11.05	42	1.85	10.77
LNS 131	4	0.01	9.89	-1	N/A	4.91	3	0.01	5.12
NNC261	30	0.15	27.06	27	0.14	27.03	28	0.16	28.52
NNC666	28	1.17	57.29	18	0.99	57.56	17	1.05	60.53

and "Bollhöfer's" refers to the preconditioner with a dynamic sparsity pattern proposed in [15].

Based on the numerical results in Table 6.4, we reported the comparisons in three aspects of accuracy, efficiency, and memory storage. In terms of accuracy, NND performed better than Bollhöfer's in most cases, and NLD was comparable to Bollhöfer's. For example, NND solved all of the test matrices, while Bollhöfer's and NLD solved 11 and 12 of the test matrices, respectively. Specifically, the number of iterations of NND is much smaller than that of Bollhöfer's and NLD (See Figure 6.1), and NND generally used less CPU time in solving the test matrices than Bollhöfer's.



(a) The CAVITY03 matrix



(b) The CAVITY08 matrix

Figure 6.1: The convergence curves of the CAVITY matrices.

Also, with respect to the storage space, NLD and NND might need slightly more memory storage than Bollhöfer's, but NND still performed far better in convergence than Bollhöfer's. For example, NLD using 1.174 times more memory storage solved the CAVITY08 matrix in 46 iterations while Bollhöfer's solved in 355 iterations.

Computational Analysis

In this subsection, we analyze the convergence behavior of the Krylov subspace methods with respect to the distribution of the eigenvalues and the condition number of the eigenvector matrix.

Table 6.4: Largest and smallest eigenvalues and conditioning information of the LNS_131 matrix.

Matrix	$\lambda_{largest}$	$\lambda_{smallest}$	cond(X)
A	-1.19E+05	-1.23E-04	1.67E+17
FAPINV(A)	-8.50E+67	-2.29E-56	2.53E+30
NLD(A)	-1.55E+22	8.29E-13	7.82E+07
NND(A)	-1.52E+22	-1.43E-12	2.22E+08

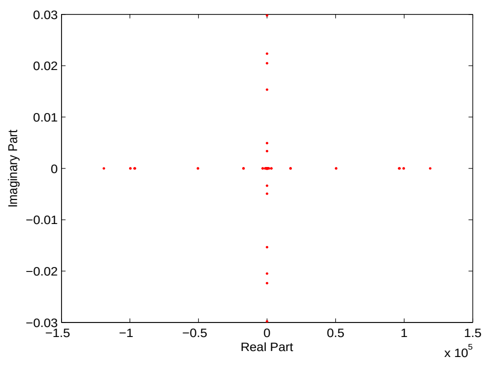
Table 6.4 and Figure 6.2 (a) report that the condition number of the eigenvector matrix of the original matrix A (LNS_131) is relatively large, and the eigenvalues of the original matrix are spread on the positive and negative parts. The FAPINV

preconditioned matrix (FAPINV(A)) shifted spectrum to the left hand side of the origin (See Figure 6.2 (b)), but the condition number of the eigenvector matrix is considerably large. As a result, the convergences of the GMRES method are failed on the original matrix and FAPINV(A).

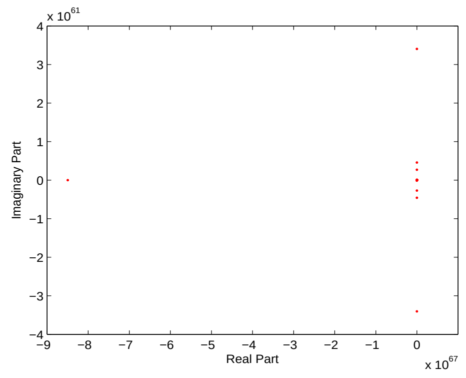
On the other hand, the NLD preconditioned matrix (NLD(A)) and the NND preconditioned matrix (NND(A)) performed differently in the aspect of the condition number of the eigenvector matrices. The condition numbers of the eigenvector matrices of NLD(A) and NND(A) are significantly decreased as shown in Table 6.4. In addition, the distribution of the eigenvalues of LNS_131 preconditioned by NLD and NND shifted the spectrum to the left hand side of the origin (See Figure 6.2 (c) and (d)). These strongly support a good convergence behavior of the GMRES method on these NLD and NND preconditioned matrices although the condition number of the NLD and NND preconditioned matrices are bit increased.

6.5 Concluding Remarks

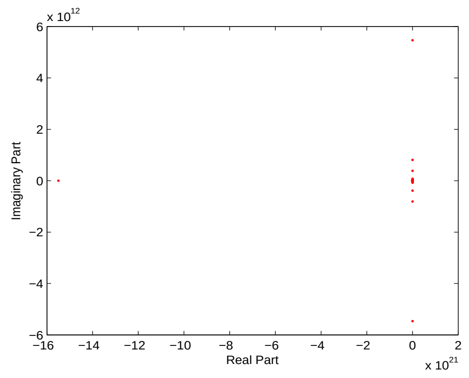
We proposed two algorithms that determine the dynamic sparsity patterns for the FAPINV preconditioners for solving general sparse matrices. In the computation phase, the dropping tolerance has been adaptively determined by the norm of the inverse factors and either the norm of the original factors or the largest value of the original factors. Numerical experiments showed that FAPINV with the proposed dynamic sparsity pattern generates more accurate and robust preconditioner than FAPINV with not only a static sparsity pattern but also other dynamic sparsity pattern (Bollhöfer's) preconditioners do.



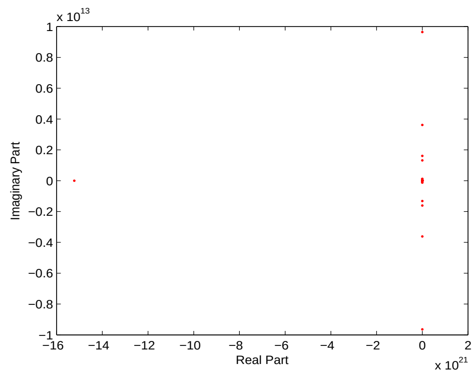
(a) Eigenvalues of the LNS_131 matrix



(b) Eigenvalues of the LNS_131 matrix with FAPINV



(c) Eigenvalues of the LNS_131 matrix with NLD



(d) Eigenvalues of the LNS_131 matrix with NND

Figure 6.2: Plots of the eigenvalues of the LNS_131 matrix. (These subfigures are in different scale.)

Chapter 7 Conclusions and Future Work

In solving large and sparse linear systems, which are usually generated from many computational science and engineering problems, preconditioned Krylov subspace methods are generally considered. This dissertation has been mainly focused on preconditioning techniques to improve the performance and reliability of iterative methods in solving linear systems. In this chapter, we summarize the specific contributions of this dissertation and present opportunities for future work.

7.1 Accomplishments

Preconditioned Krylov subspace methods are generally regarded as one class of the most promising techniques [9, 17, 19, 41] for solving very large and sparse linear systems. In general, incomplete lower-upper (ILU) triangular factorizations and sparse approximate inverse (SAI) preconditioning techniques have attracted much attention as a preconditioner because they have been successful in solving many symmetric and non-symmetric matrices. These techniques, however, may yield two common problems. Firstly, their accuracy may be insufficient to yield an adequate rate of convergence due to the difficulty in determining parameters or threshold values in constructing preconditioners. In addition, they may encounter difficulty when the matrix to be solved is indefinite, that is, when the matrix has both positive and negative eigenvalues. Specifically, small or zero pivots in indefinite matrices may produce unstable and inaccurate preconditioners. In response to these problems associated with preconditioning techniques, we have concerned with improving accuracy and stability of a preconditioner in solving large and sparse matrices.

Incomplete LU (ILU) Preconditioning Enhancement Strategies

Several preconditioning enhancement strategies for improving inaccurate preconditioners produced by the incomplete LU factorizations of sparse matrices were presented. The strategies employ the elements that are dropped during the incomplete LU factorization and utilize them in different ways by separate algorithms. The first strategy (error compensation) applies the dropped elements to the lower and upper parts of the LU factorization to compute a new error compensated LU factorization. Another strategy (inner-outer iteration), which is a variant of the incomplete LU factorization, embeds the dropped elements in its iteration process. Experimental results showed that the presented enhancement strategies improve the accuracy of the incomplete LU factorization when the initial factorizations found to be inaccurate. Furthermore, our numerical experimental results showed that in many cases, the convergence cost of the preconditioned Krylov subspace methods is reduced on solving the original sparse matrices with the proposed strategies.

Hybrid Reordering Strategies

Incomplete LU factorization techniques often have difficulty on indefinite sparse matrices. We presented hybrid strategies for such matrices and new diagonal reorderings that are in conjunction with a symmetric nondecreasing degree algorithm. We first use the diagonal reorderings to efficiently search for entries of single element and/or the maximum absolute value to be placed on the diagonal for computing a nonsymmetric permutation. In addition, a nondecreasing degree algorithm is applied to reduce the fill-in during the ILU factorization. With the reordered matrices, we achieved a considerably improved performance of incomplete LU factorizations in stability. Consequently, the convergence cost of preconditioned Krylov subspace methods was reduced on solving the reordered indefinite matrices.

Two-Phase Preconditioning Strategy

As a part of the continuous effort on solving indefinite matrices, a two-phase preconditioning strategy based on a factored sparse approximate inverse has been proposed. In each phase, the strategy first makes the original matrix diagonally dominant to enhance the stability by a shifting method and constructs an inverse approximation of the shifted matrix by utilizing a factored sparse approximate inverse preconditioner. The two inverse approximation matrices produced from each phase are then combined to be used as a preconditioner. Experimental results showed that the presented strategy improves the accuracy and the stability of the preconditioner on solving indefinite sparse matrices. Furthermore, the strategy ensures that convergence rate of the preconditioned iterations of the two-phase preconditioning strategy is much better than that of the standard sparse approximate inverse ones for solving indefinite matrices.

Factored Approximate Inverse Preconditioners with Dynamic Sparsity Patterns

We proposed two sparsity pattern selection algorithms for factored approximate inverse preconditioners to enhance the performance of the preconditioned iterative solvers. The sparsity pattern is adaptively updated in the construction phase by using combined information of the inverse and original triangular factors of the original matrix. In order to determine the sparsity pattern, our first algorithm uses the norm of the inverse factors multiplied by the largest absolute value of the original factors, and the second employs the norm of the inverse factors divided by the norm of the original factors. Experimental results showed that these algorithms improve the accuracy and robustness of the preconditioners on solving general sparse matrices.

7.2 Future Work

Our research area fits well with other fields and disciplines as we have significant collaborative experience with researchers from computer vision areas [40]. Thus, our future work will be applying the preconditioning methods to real-world applications in order to achieve an improved performance in computational time as well as accuracy.

Preconditioner Recommendation System

My first plan for future work is to develop a preconditioner recommendation system for indefinite matrices based on the features of matrices [65, 66]. This idea is inspired by the fact that sparse matrices arising from discretized partial differential equation problems possess certain different features represented by the sizes and the locations of their nonzero entries. Thus, if it can be determined that the performance of preconditioned Krylov subspace methods is related to such matrix features, it will be possible to predict the performance of these preconditioned methods to solve other sparse matrices that may have the same or similar features.

Application Areas

Along with the development of a preconditioner recommendation system, my future work will also be concentrated on applying these strategies presented above with existing preconditioners and solvers. More specifically, the new decomposition techniques, aiming at solving irregularly structured sparse linear systems, will also be used for potential application areas. Computational fluid dynamics, large scale computer modeling and simulations are two examples in which large scale linear systems are routinely solved, and our preconditioning techniques can be extended to accommodate the requirement of the applications.

Bibliography

[1] K. Atkinson *An Introduction to Numerical Analysis.* John Wiley&Sons, New York, 1988.

[2] O. Axelsson *A survey of preconditioned iterative methods for linear systems of algebraic equations.* BIT, **2**5 (1985), 166–187.

[3] O. Axelsson *Iterative Solution Methods.* Cambridge Univ. Press, Cambridge, 1994.

[4] S.T. Barnard, L.M.Bernado, and H.D. Simon. *An MPI implementation of the SPAI preconditioner on the T3E.* Technical Report LBNL–40794, Ernest Orlando Lawrance Berkeley National Laboratory, Berkely, CA, 1997.

[5] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eikhout, V. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, Philadelphia, 1994.

[6] M. W. Benson and P. O. Frederickson. *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems.* Utilitas Math., **2**2 (1982), 127–140.

[7] M. W. Benson, N. Krettmann, and M. Wright. *Parallel algorithms for the solution of certain large sparse linear systems.* Int. J. Comput. Math., **16** (1984), 245–260.

[8] M. Benzi. *Preconditioning techniques for large linear systems: a survey.* J. Comput. Phys., 182:418–477 (2002).

[9] M. Benzi and D. Bertaccini, *Approximate inverse preconditioning for shifted linear systems*, BIT, **4**3 (2003), 231–244.

[10] M. Benzi, J.C. Haws, and M. Tuma. *Preconditioning highly indefinite and nonsymmetric matrices.* SIAM J. Sci. Comput., 22:1333–1353 (2000).

[11] M. Benzi, C.D. Meyer, and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., **1**7-5 (1996), 1135–1149.

[12] M. Benzi and M. Tuma, *A comparative study of sparse approximate inverse preconditioners*, Applied Numerical Meth., **3**0-2-3 (1999), 305–340.

[13] M. Benzi and M. Tuma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., **1**9-3 (1998), 968–994.

[14] M. Bollhöfer, *A robust ILU with pivoting based on monitoring the growth of the inverse factors*, Linear Algebra and its Applications, **3**38 (2001), 201–218.

[15] M. Bollhöfer, *A robust and efficient ILU that incorporates the growth of the inverse triangular factors*, SIAM J. Sci. Comput., **2**5-1 (2003), 86–103.

[16] T.F. Chan, W.P. Tang, and W.L. Wan, *Wavelet sparse approximate inverse preconditioners*, BIT, **3**7-3 (1997), 644–660.

[17] T.F. Chan, and H.A. van der Vorst, *Approximate and incomplete factorizations*, Parallel Numerical Algorithms, ICASE/LaRC Interdisciplinary Series in Science and Engineering **4** (1997), 91–118. (http://www.math.uu.nl/people/vorst/publ.html#publ94)

[18] E. Chow and Y. Saad, *Approximate inverse preconditioners for general sparse matrices*, Technical Report UMSI **9**4/101 (1994), Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN.

[19] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., **8**6 (1997), 387–414.

[20] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., **1**9 (1998), 995–1023.

[21] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM J. Sci. Comput., **1**8 (1997), 1657–1675.

[22] J. D. F. Cosgrove, J. C. Diaz, and A. Griewank. *Approximate inverse preconditionings for sparse linear systems.* Internat. J. Comput. Math., **4**4 (1992), 91–110.

[23] B.S. Datta *Numerical linear algebra and applications.* Brroks/Cole Publishing Company, Pacific Grove, CA, 1995.

[24] T.A. Davis. *University of Florida sparse matrix collection.* http://www.cise.ufl.edu/research/sparse/matrices, 1997.

[25] J. W. Demmel. *Applied Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[26] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers.* SIAM, Philadelphia, 1998.

[27] I.S. Duff. *Algorithm 575: Permutations for zero-free diagonal.* ACM Trans. Math. Software, 7:387–390 (1981).

[28] I.S. Duff, G.A. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices.* Claredon Press, New York, 1986.

[29] I.S. Duff, R.G. Grimes, and J.G. Lewis. *Users' Guide for the Harwell-Boeing Sparse Matrix Collection (Release I).* Technical Report RAL-92-086, Rutherford Appleton Laboratory, Oxfordshire, England, 1992.

[30] I.S. Duff, and J. Koster. *The design and use of algorithms for permuting large entries to the diagonal of sparse matrices.* SIAM J. Matrix Anal. Appl., 20:889-901 (1999).

[31] I.S. Duff, and J. Koster. *On Algorithms for Permuting Large Entries to the Diagonal of a Sparse Matrix.* SIAM J. Matrix Anal. Appl., 22:973-996 (2001).

[32] I.S. Duff, and G.A. Meurant, *The effect of reordering on preconditioned conjugate gradients.* BIT, 29:635-657 (1989).

[33] A.C.N. van Duin, *Scalable parallel preconditioning with the sparse approximate inverse of triangular matrices*, SIAM J. Matrix Anal. Appl. **2**0 (1999), 987–1006.

[34] H.C. Elman, *A stability analysis of incomplete LU factorization*, Math. Comput., **47** (1986), 191–217.

[35] G.H. Golub and H.A. var der Vorst, *Closer to the solution:iterative solvers*, In I. S. Duff and G. A. Watson, editors, *The state of the Art in Numerical Analysis*, Clarendon Press, Oxford, pages 63–92, 1997.

[36] G.H. Golub and C.F. Van Loan, *Matirx computations*, Johns Hopkins, Baltimore and London, 1996.

[37] N.I.M. Gould and J.A. Scott, *Sparse approximate-inverse preconditioners using norm-minimization techniques*, SIAM J. Sci. Comput., **1**9-2 (1998), 605–625.

[38] G.A. Gravvanis, *An approximate inverse matrix technique for arrowhead matrices*, Intern. J. Computer Math., **7**0 (1998), 35–45.

[39] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.

[40] E. Grossmann, E.-J. Lee, P. Hislop, D. Nister and H. Stewenius, *Are two rotational flows sufficient to calibrate a smooth non-parametric sensor?*, Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, **1** (2006), 1222–1229.

[41] M.J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., **1**8 (1997), 838–853.

[42] N. Kang, J. Zhang, and E. S. Carlson. Parallel simulation of anisotropic diffusion with human brain DT-MRI data. *Computers and Structures*, 82(28):2389–2399, 2004.

[43] S. Karaa, J. Zhang, and C. C. Douglas. Preconditioned multigrid simulation of an axisymmetric laminar diffusion flame, *Math. Comput. Model.*, 38:269–279, 2003.

[44] D.S. Kershaw, *On the problem of unstable pivots in the incomplete LU-conjugate gradient method*, J. Comput. Phys. **3**8 (1980), 114–123.

[45] L. Y. Kolotina and A.Y. Yeremin. *Factored sparse approximate inverse preconditioning I:theory.* SIAM J. Matrix Anal. Appl., **1**4 (1993), 45–58.

[46] E.-J. Lee, and J. Zhang, *Hybrid reordering strategies for ILU preconditioning of indefinite sparse matrices*, J. Appl. Math. Comput., 22:307–316 (2006).

[47] J.-G. Luo, *An incomplete inverse as a preconditioner for the conjugate gradient method*, Computer Math. Applic., **2**5-2 (1993), 73–79.

[48] J.-G. Luo, *A new class of decomposition for inverting asymmetric and indefinite matrices*, Computer Math. Applic., **2**5-4 (1993), 95–104.

[49] J.-G. Luo, *A sparsity for decomposing a symmetric matrix*, Computer Math. Applic., **2**5-5 (1993), 83–90.

[50] The Matrix Market of the National Institute of Standards and Technology, *http://math.mist.gov/matrixMarket.*

[51] T.A. Manteuffel, *An incomplete factorization techniques for positive definite linear systems*, Math. Comput., **3**4 (1980), 473–497.

[52] J.A. Meijerink and H.A. Van Der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is asymmetric M-matrix*, Math. Comp., **3**1 (1977), 148–162.

[53] J. Meixner. *Network theory and its relation to the theory of linear systems.* Antennas and Propagation, IEEE Transactions, **7**-5 (1959), 435–439.

[54] Y. Saad, *ILUT: a dual threshold incomplete LU factorization*, Numer. Linear Algebra Appl. **1**4 (1994), 387–402.

[55] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS, New York, 1996.

[56] Y. Saad and M. Schultz, *Conjugate gradient-like algorithms for solving nonsymmetric linear systems.* Mathematics of Computation, **4**4 (1985), 417–424.

[57] Y. Saad and H. A. van der Vorst. *Iterative Solution of Linear Systems in the 20-th Century.* Journal of Computational and Applied Mathematics, **1**23-1-2 (2000), 1–33.

[58] H.D. Simon. *Incomplete LU preconditioners for conjugate-gradient-type iterative methods.* in Proceedings of the 1985 SPE Reservoir Simulation Symposium, Richardson, TX, Society of Petroleum Engineers, pp. 387–396 (1985).

[59] W.-P. Tang. *Towards an effective sparse approximate inverse preconditioner.* SIAM J. Matrix Anal. Appl., **2**0-4 (1999), 970–986.

[60] L. N. Trefethen and D. Bau. *Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[61] R.S. Varga. *Matrix Iterative Analysis.* Springer Verlag, Berlin, New York, 2000.

[62] K. Wang, S. Kim, and J. Zhang, *A comparative study on dynamic and static sparsity patterns in parallel sparse approximate inverse preconditioning*, Journal of Mathematical Modeling and Algorithms, **3**-2 (2003), 203–215.

[63] K. Wang and J. Zhang, *MSP: A class of parallel multistep successive sparse approximate inverse preconditioning strategies*, SIAM J. Sci. Comput., **2**4-4 (2003), 1141–1156.

[64] L. Wang and J. Zhang, *A two step combined stable preconditioning strategy for incomplete LU factorization of CFD matrices*, Appl. Math. Comput., **1**44-1 (2003), 75–87.

[65] S. Xu, E.-J. Lee, and J. Zhang, *An interim analysis report on preconditioners and matrices*, Technical Report No.388-03, Department of Computer Science, University of Kentucky, KY, 2003.

[66] S. Xu, E.-J. Lee, and J. Zhang, *Designing and building an intelligent preconditioner recommendation system (a progress report)*, In Abstracts of the 2003 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications, Napa, CA, 2003.

[67] J. Zhang. *Preconditioned Krylov subspace methods for solving nonsymmetric from CFD applications.* Comput. Methods Appl. Mech. Engrg., 189:825–840 (2000).

[68] J. Zhang, *A multilevel dual reordering strategy for robust incomplete LU factorization of indefinite matrices*, SAIM J. Matrix Anal. Appl., **2**2-3 (2001), 925–947.

[69] J. Zhang, *A sparse approximate inverse technique for parallel preconditioning of general sparse matrices*, Appl. Math. Comput., **1**30-1 (2002), 63–85.

[70] J. Zhang, A. Pardhanani, G. Carey *Performance of adaptive dual-dropping ILUT preconditioners in semiconductor dopant diffusion simulation*, International J. Numerical Modelling: ElectronicNetworks, Devices and Fields, **1**5-2 (2002), 147–167.

Vita

Personal Data:

Name: Eun-Joo Lee
Date of Birth: 10/04/1966
Place of Birth: Gwangju, South Korea

Educational Background:

- Aug. 1997: Ph.D.
  Department of Mathematics, Chonnam National University, Gwangju, South Korea.
- Feb. 1991: M.S.
  Department of Mathematics, Chonnam National University, Gwangju, South Korea.
- Feb. 1989: B.S.
  Department of Mathematics, Chonnam National University, Gwangju, South Korea.

Professional Experience:

- Jan. 2004 - Present: Teaching Assistant,
  Department of Computer Science, University of Kentucky, Lexington, KY, USA.
- Jan. 2003 - Dec. 2003: Research Assistant,
  Laboratory for High Performance Scientific Computing and Computer Simulation, Department of Computer Science, University of Kentucky, Lexington, KY, USA.
- Sep. 1997 - Feb. 1999: Lecturer,
  Department of Mathematics, Sunchon National University, Sunchon, South Korea.
- Mar. 1994 - Feb. 1999: Lecturer,
  Department of Mathematics, Chonnam National University, Gwangju, South Korea.
- Sep. 1991 - Aug. 1992: Administrative Support Associate,
  Department of Mathematics, Mokpo National University, Mokpo, South Korea.

- Mar. 1989 – Feb. 1991: Teaching Assistant,
  Department of Mathematics, Chonnam National University, Gwangju, South Korea.

Awards:

- Student support from Graduate School Fellowship, University of Kentucky, Mar. 2008.
- Outstanding Teaching Assistant Award 2007, Department of Computer Science, University of Kentucky, May 2007.
- Student support from Graduate School Fellowship, University of Kentucky, Aug. 2006.
- Fellowship Award to attend The Seventh DOE ACTS Workshop, Lawrence Berkeley National Lab, Aug. 2006.
- Student support from Graduate School Fellowship, University of Kentucky, Mar. 2006.
- Student support from The Eighth Cooper Mountain Conference on Iterative Methods, Mar. 2006.
- Student support from Graduate School Fellowship, University of Kentucky, May 2005.

Publications:

- E.-J. Lee and J. Zhang, *A Two-phase Preconditioning Strategy of Sparse Approximate Inverse for Indefinite Matrices*, Computers and Mathematics with Applications (2008), in review.
- E.-J. Lee and J. Zhang, *Factored approximate inverse preconditioners with dynamic sparsity patterns*, Journal of Computational and Applied mathematics (2008), in review.
- E.-J. Lee and J. Zhang, *A Two-phase Preconditioning Strategy of Sparse Approximate Inverse for Indefinite Matrices*, Computers and Mathematics with Applications (2008), in review.
- E. Grossmann, E.-J. Lee, P. Hislop, D. Nister and H. Stewenius, *Are two rotational flows sufficient to calibrate a smooth non-parametric sensor?*, Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, **1** (2006), 1222–1229.

- E.-J. Lee and J. Zhang, *Hybrid reordering strategies for ILU preconditioning of indefinite sparse matrices*, Journal of Applied Mathematics and Computing, **2**2-2 (2006), 307–316.

- S. Xu, E.-J. Lee, and J. Zhang, *An interim analysis report on preconditioners and matrices*, Technical Report No.388-03, Department of Computer Science, University of Kentucky, KY, 2003.