國立臺灣大學理學院數學系

碩士論文

Department of Mathematics

College of Science

National Taiwan University

Master Thesis

高效率更新非線性偏微分方程導出之線性系統序列的預處理器

Effective Preconditioner Updates for Sequences of Linear Systems Derived from Nonlinear Partial Differential Equations

郭彥祥

Yen-Hsiang Kuo

指導教授：王藹農教授

Advisor: Ainung Wang, Professor

中華民國一百零四年七月

July , 2015

謝　辭

兩年的碩士學習過程中，不同於大學教與學的模式，而是強調自主學習、獨立解決問題的能力，壓力也在無形中開始萌芽，非常感謝我的家人與俞廷儒精神上的支持與鼓勵．在學業上，不免會遇的問題，很幸運地身旁總是有同學的幫助，謝謝王柏川、吳志強、蕭煜修、林士閔、薛竣壬同學們的幫助，以及學長的教導，謝謝夏佑槐、朱哲銘、李其澔、許奕中、黃志銘、陳嘉宏學長們的教導。在研究的過程中，感謝王偉仲教授對我在數學程式上的訓練，提供許多數學軟體的最新資訊，感謝薛克民教授不厭其煩地讓我無時無刻地尋求幫助，每次都能提供我許多解決問題的提示，讓我依循線索找到解答．最後非常感謝王藹農教授每當我遇到問題時，教授會非常熱心地、耐心地、細心地教導我，直到完全確定我清楚理解才會安心的結束教導。

摘 要

隨著科技的進步，人們在許多領域（如物理學、地震學、氣體動力學、化學等）上處理著更精密且精確的問題，因此科學計算應該被高度重視。在科學計算中，有率效地解決一連串大型且稀疏的線性系統扮演了極為重要的角色。

在早期，人們使用直接法或迭代法單獨地解決一連串線性系統中的每一個問題，當線性系統的維度很大時，直接法將會非常悲慘。如果我們使用迭代法，強大的預處理器對於解決線性系統非常有幫助，但是要找尋或建造出全能的預處理器是非常困難且耗時的任務。現今，我們應用先前線性系統的資訊到目前線性系統或是其餘的線性系統達到節省時間的功效。

在文章中我們將會以一個二維度非線性對流 -擴散模型問題來當作我們的例子。我們會簡單的介紹有限差分方法, 牛頓 -拉弗森方法和線搜索法，而且透過以上的這些概念，我們將會創造出一連串的線性系統。

之後，我們會討論三種有趣的逼近更新分解預處理器的方法，數值結果告訴我們這三種方法是有幫助的，也就是在使用預處理器的迭代法時，相較於固定一連串線性系統中的第一個預處理器，這三種方法會得到比較少的迭代次數。因為這三種有趣的更新預處理器的方法基本上來說是很省時的、容易實行的，所以他們可以取代很耗時的重新計算預處理器。

最後，為了完成我們的工作，我們主要的參考文獻為 Jurjen Duintjer Tebbens 和 Miroslav Tuma 共同研究的 [7] 與 [8]，基本知識的準備我們參

考 John E Dennis Jr 和 Robert B Schnabel 的 [1]、Hans Petter Langtangen 的 [2]、Randall J LeVeque 的 [3] 和 Stephen J Wright 與 Jorge Nocedal 合力完成的 [10] 等著作。我們重新設計與安排 [7]，盡可能讓讀者容易了解 [7] 的內容與想法。

關鍵字：一連串線性系統、預處理迭代法、不完全分解、分解更新、高斯喬丹轉換、謝爾曼·莫里森公式

Abstract

With the advance of science and technology, people deal with problems more precisely and accurately in many fields like Physics, Seismology, Aerodynamics, Chemistry and so on and so forth. Therefore scientific computing should be highly concerned. Effective solving sequence of linear systems with large and sparse matrices plays a very important role in scientific computing.

In early times, people used direct method or iterative method to solve linear system one by one in the sequence. The direct method will be miserable if the dimension of the linear system is pretty much large. If we use iterative method to solve linear systems, a powerful preconditioner will be very helpful. But finding and constructing an almighty preconditioner will be a very difficult and time-consuming mission. Nowadays, we can use the information from the previous linear system to the current linear system or the other systems in order to save time.

In our article, we will take a two-dimensional nonlinear convection-diffusion model problem to be our example. We present a brief introduction of finite difference method, Newton-Raphson method and line search method. After applying these ideas, we will have a sequence of linear systems needed to be solve.

And then, we will discuss three interesting methods for approximate updates of factorized preconditioners for solving sequences of linear systems. Numerical experiments show that these three method are profitable, that is, they have fewer number of iterations of preconditioned iterative methods for solving sequent systems of a sequence than freezing the preconditioner from the first system of the sequence. Since the interesting updates mainly cost less and straightforward, they may substitute for recomputing preconditioners which may take lots of time.

To complete our work, we mainly consult [1], [2], [3], [7], [8] and [10]. And we also redesign and rearrange [7] in order to introduce everything as explicit as we can.

Keywords: Sequence of linear systems, Preconditioned Iterative method, Incomplete factorizations, Factorization updates, Gauss-Jordan transformations, Sherman–Morrison formula

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sequences of linear systems with large and sparse matrices turns up in a lot of applications such as kinematics, computational fluid dynamics, structural mechanics, numerical optimization as well as in solving problems derived from nonlinear partial differential equations. In more detail, we are going to face sequences of linear systems

$$A^{(i)}x = b^{(i)}, i = 1, \cdots , \tag{1.1}$$

where $A^{(i)} \in \mathbb{R}^{n\times n}$ are general nonsingular sparse matrices and $b^{(i)} \in \mathbb{R}^n$ are the corresponding right-hand sides in the above applications.

1.1 Literature Review

These sequences of linear systems can be derived from solving a system of nonlinear equations $F(x) = 0$ where $F : \mathbb{R}^n \to \mathbb{R}^n$. By using Newton or quasi-Newton method, we have a sequence of linear problems

$$J(x_i)(x_{i+1} - x_i) = -F(x_i), i = 1, \cdots , \tag{1.2}$$

where $J(x_i)$ is the Jacobian or approximation to Jacobian in the current iteration x_i.

The Jacobians $J(x_i)$ sometimes are expensive so we alternatively compute their approximations. It is well-known to use Broyden-Fletcher-Goldfarb-Shanno (BFGS) update which are dense matrices when we compute approximations. At the beginning, Schubert offered a method for updating approximation which makes matrices sparse [5]. Shanno presented a sparse version of BFGS update.[6]. Toint also proposed an interesting approximation updates which are not only sparse but also symmetric [9]. Lucia tendered a new quasi-Newton updating formula combined planning for fixed symmetric and idea from Schubert [4].

Efficient solving sequence of linear systems is a difficult task as we mentioned in abstract. We could compute mighty preconditioners $M^{(1)}, M^{(2)}, \cdots$ for each system respectively, but it would be pretty expensive in some cases. So there is a natural way to enhance the performance, that is to freeze the preconditioner which is only computed from the initial systems and reused for the other system matrices. But, in practice, freezing the preconditioner may not be fast enough for convergence. Hence we need distribute some information or computational effort from the current system among other systems in the subsequent linear system. This is what we will mainly introduce [7] and [8] in this thesis. We discuss three strategies in [7] for updating preconditioner which is factorized as $LDU \approx A$. The updated preconditioner will be used for solving the subsequence of linear systems. Numerical experiments show that the three kinds of updates are competitive with recomputing preconditioners in terms of having the similar convergence rates for subsequence of systems. Furthermore, it is more cheaper to form the updated preconditioners.

We are going to introduce in Chapter 2 some backgrounds ,in [1], [2], [3] and [10],of constructing a sequence of linear systems from a two-dimensional nonlinear convection-diffusion model problem. For example, after discretizing the two-dimensional nonlinear convection-diffusion model problem by finite difference method, we need the concept of solving nonlinear systems by Newton–Raphson method with line search method. After these procedures, we will have a sequence of linear systems to solve. In Chapter 3, we present the idea of three interesting approaches in [7] for approximation updates of factorized preconditioners in theory and implemention in practice.

The numerical experiments will be offered and discussed in Chapter 4.

To complete our work, we mainly consult [1], [2], [3], [7], [8] and [10]. In this thesis, We will discuss the effective triangular preconditioner updates and the other two updates for sequences of linear systems derived from nonlinear partial differential equations as explicit as we can. Throughout the article, $\| \cdot \|$ denotes an unspecified, arbitrary norm.

Chapter 2

Preliminary

2.1 Newton-Raphson Method and Line Search Method

We first introduce Newton-Raphson method in [2] for one nonlinear equation $F(x) = 0$ in single scalar variable x. The systems of nonlinear equations can be extended by similar ways.

Assume x^k is an approximation of x. We hope x^k can be close enough to x, when k increases, so we have to improve the approximation. And we hope there is an idea that not only makes procedure easier to implement but also improves the approximation x^k. The idea is to construct a approximation of $F(x)$ close to x^k such that $F(x) \approx N(x; x^k)$ where $N(x; x^k) = 0$ is easier to solve. The solution of $N(x;x^k)=0$ is viewed as an improved approximation of x^{k+1} to the root x of $F(x)=0$. By using Taylor expansion, $N(x;x^k)$ is the linear part of a Taylor-series approximation of F at the point of $x=x^k$.

$$N(x;x^k)=F(x^k)+\frac{dF}{dx}(x^k)(x-x^k).$$

Next, we let x^{k+1} to be the solution of $N(x;x^k)=0$. In other words, we are going to find the solution of the equation $N(x^{k+1};x^k)=0$ with respect to x^{k+1} then we have

$$x^{k+1}=x^k-\frac{F(x^k)}{\frac{dF}{dx}(x^k)}.$$

This is the Newton-Raphson iteration scheme for solving $F(x)=0$. The Convergence rate for Newton-Raphson iteration scheme is quadratic, that is,

$$|x-x^{k+1}|\leq C|x-x^k|^2$$

For the systems of nonlinear equations $F(x)=0$, we also construct a approximating $F(x)$ by $N(x;x^k)$ near x^k which is also an approximation to x. And $N(x;x^k)$ similarly satisfies

$$N(x;x^k)=F(x^k)+J(x^k)(x-x^k).$$

where $J\equiv\nabla F$ is the Jacobian of F. If $F=(F_1,\cdots,F_n)^T$ and $x=(x_1,\cdots,x_n)^T$ then entry (i,j) in J is $\partial F_i/\partial x_j$. Similarly, we solve a linear system with J as coefficient matrix in order to find the next approximation

x^{k+1} from $N(x^{k+1}; x^k)$. By repeating this process, we have a sequence of x^i which is close to the solution of $F(x) = 0$.

The stopping criterions are

$$\|x^{k+1} - x^k\| \leq \epsilon_x \quad \text{or} \quad \|F(x^{k+1})\| \leq \epsilon_r$$

or

$$\frac{\|x^{k+1} - x^k\|}{\|x_k\|} \leq \epsilon_x \quad \text{or} \quad \frac{\|F(x^{k+1})\|}{\|F(x^0)\|} \leq \epsilon_r$$

Algorithm 2.1.0.1 Newton-Raphson Method

```
1: Given a initial $x^0$ for the solution of $F(x) = 0$
2: **while** termination criterion is NOT fullfilled **do**
3:     Solve $J(x^k)\delta x^{k+1} = -F(x^k)$ with respect to $\delta x^{k+1}$.
4:     Set $x^{k+1} = x^k + \delta x^{k+1}$
5: **end while**
```

The idea of line search algorithm in [10] is very simple : given a descent direction p_k, we find a step in that direction that yields an acceptable x_{k+1}, that is

Algorithm 2.1.0.2 Line Search Method

```
1: Calculate the descent direction $p_k$
2: Set $x_{k+1} \equiv x_k + \lambda_k p_k$ for some $\lambda_k > 0$ which makes $x_{k+1}$ an acceptable
   for next iteration.
```

A famous inexact line search condition is the Wolfe conditions satisfying

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k \tag{2.1}$$

for some $c_1 \in (0,1)$. The reduction in f should be proportional to both step length α_k and the directional derivative $\nabla f_k^T p_k$. Inequality (2.1) is called the *Armijo condition*. In practice, $c_1$is chosen to be equit small and ussually to be $c_1 = 10^{-4}$. Sometime, we want a more larger step, we have second requirement called a *curvature condition* is introduced

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f_k^T p_k \tag{2.2}$$

where $c_2 \in (c_1, 1)$.

Combine (2.1) with (2.2), we have *strong Wolfe conditions*.

$$f(x_k + \alpha_k p_k) \leq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k,$$

$$|\nabla f(x_k + \alpha_k p_k)^T p_k| \leq |c_2 \nabla f_k^T p_k|,$$

$$0 < c_1 < c_2 < 1$$

There are some other conditions, for example : Goldstein Condition,$\cdots$, etc..

When we use line search method, we may face some breakdowns which we will not discuss here. But we can use the backtracking line search method below to avoid some of the breakdowns.

Algorithm 2.1.0.3 Backtracking Line Search Method

```
1: Choose $\overline{\alpha} > 0,\ \rho,\ c \in (0,1)$.
2: Set $\alpha = \overline{\alpha}$.
3: while $f(x_k + \alpha_k p_k) \geq f(x_k) + c_1 \alpha_k \nabla f_k^T p_k$ do
4:     $\alpha = \rho\alpha$
5: end while
6: Terminate with $\alpha_k = \alpha$.
```

We can combine the Newton-Raphson method with line search method to make more efficiently and more stably.

2.2 Sherman–Morrison Formula

Suppose A is an invertible matrix and u, v are two column vectors. And we assume that $1 + v^T A^{-1} u \neq 0$. Then the Sherman–Morrison formula is that

$$(A + uv^T)^{-1} = A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}$$

where uv^T is the outer product of the vectors u and v.

Proof.

$$\begin{aligned}
(A + uv^T)\left(A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}\right) &= AA^{-1} + uv^TA^{-1} - \frac{AA^{-1}uv^TA^{-1} + uv^TA^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} \\
&= I + uv^TA^{-1} - \frac{uv^TA^{-1} + uv^TA^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u} \\
&= I + uv^TA^{-1} - \frac{u(1 + v^TA^{-1}u)v^TA^{-1}}{1 + v^TA^{-1}u} \\
&= I + uv^TA^{-1} - uv^TA^{-1} = I
\end{aligned}$$

Similarly,

$$\left(A^{-1} - \frac{A^{-1}uv^TA^{-1}}{1 + v^TA^{-1}u}\right)(A + uv^T) = I$$

□

Chapter 3

Preconditioner Update

In this thesis, we are going to present some strategies to update preconditioners from sequences of general, nonsymmetric systems that solved by preconditioned iterative method. We will also introduce these ways that updated preconditioner is as powerful as the original one in theory and construct a preconditioner cheap in practice. And this types of updates have particularly beneficial behavior in [8] under the following three kinds of circumstances : first, if preconditioner recomputation undergoes instability or updates have a more stable factorization; second, if the update is dominant, at the very least structurally or the update covers the important part of the difference matrix between the current and the other matrix; third, if recomputed preconditioners is somehow expensive (solving for a long time) in parallel computations, matrix-free environment.

3.1 Theoretical Analysis

In [7], for the purpose of simplifying the notation, we denote two linear systems of dimension n by $Ax = b$ and $A^*x^* = b^*$. Denote by B the difference matrix $A - A^*$ and by M the preconditioner of A, respectively. There are two some information we can get about the quality of the preconditioner M from a norm of the following matrix.

$$A - M \tag{3.1}$$

If we study preconditioning from the left or right, we can get message from the two norms of matrices, individually.

$$I - M^{-1}A \tag{3.2}$$

or

$$I - AM^{-1} \tag{3.3}$$

We call information from the norm of the matrix (3.1) *accuracy* of the preconditioner M with respect to A. Message from the norms of the matrices (3.2) and (3.3) are said to be *stability* of M (with respect to A). We let M^* be to a updated preconditioner for A^* whose accuracy and stability are close to the accuracy and stability of M for A. In our thesis, we will consider the norm of the matrix (3.1) due to its simplicity.

Easily, we get

$$\|A - M\| = \|A^* - (M - B)\|$$

Hence the norm of the difference matrix $A^* - M^*$ with $M^* \equiv M - B$ is the same as the norm of the difference matrix $A - M$. We will call M^* the *ideal* updated preconditioner with respect to A^*. Actually, there may be other preconditioners that are ideal with the norm of $A^* - M^*$.For example, we can consider $M^* = M - C$ for some matrices $M \neq B$ with

$$\|A - M\| = \|A^* - M^*\| = \|A^* - M + C\|$$

Since B is usually directly available, we will focus on $M^* = M - B$.

If we use preconditioned iterative method to solve the sequences of linear systems, we will suffer from multiplying vectors with inverse of the preconditioner M^* in each iteration of the linear solver. In some special cases, the difference matrix B can make $(M - B)^{-1}$ obtain from M^{-1} with low costs. For instance, if B has small rank, $M^* = M - B$ will be directly inverted by using Sherman-Morrison formula. But in general cases, the ideal preconditioner M^* can not be accepted in practice since the multiplications of vectors with $(M - B)^{-1}$ is very expensive. Instead, we will choose cheaper approximations of $(M - B)^{-1}$.

In this thesis, we may assume that M is in the form of a triangular decomposition, that is, $M = LDU \approx A$, where L and U have unit main diagonal.

We typically suppose that the approximate updates of factorized preconditioner that we are going to discuss have strong diagonals. This assumption is in order to do not have a breakdown when we have a simple incomplete factorization. For instance, if the system matrix is an H-matrix, ILU(0) and AINV preconditioners are proved to be breakdown-free. We can extend the breakdown-free property by some modifications which change the decomposition and make the diagonal stronger, e.g., preliminary shift or global modification of the decomposition. In the following we presume that matrices are in the form that factors L and U more or less approximate the identity matrix.

If $M - B$ is invertible, we may use a product of more factors which are easier to invert to approximate its inverse such as a product of inverses of triangular matrices and an inverse of a difference of matrices where a diagonal matrix is used instead of M take the place of $(M - B)^{-1}$.

$$(M - B)^{-1} = U^{-1}(D - L^{-1}BU^{-1})^{-1}L^{-1} \approx U^{-1}(D - B)^{-1}L^{-1}, \qquad (3.4)$$

where we have $D - B$ is nonsingular. Now we believe $\overline{D - B}$ is a invertible approximation of $D - B$ which can be inverted inexpensively. Then we can define a precondtioner M^* via the last explication (3.4) as

$$M^* = L(\overline{D - B})U \qquad (3.5)$$

In the symmetric case, that is, $L = U$. We have $M^* = L(\overline{D - B})L^T$. If we choose $\overline{D-B}$ approximately, we can still preserve the symmetry. Here we mainly discuss in the nonsymmetric case. We can also get further simple update. For instance, we can approximate as

$$(M-B)^{-1} = (DU - L^{-1}B)^{-1}L^{-1} \approx (DU-B)^{-1}L^{-1} \tag{3.6}$$

where $DU-B$ is invertible. If $\overline{DU-B}$ is an easily invertible and nonsingular approximation of $DU-B$, then we define M^* by

$$M^* = L(\overline{DU-B}) \tag{3.7}$$

It appears to be much easier to cope with two factors than three factors comparing to (3.5). An analogue of (3.6) is approximation through

$$(M-B)^{-1} = U^{-1}(LD - BU^{-1})^{-1} \approx U^{-1}(LD-B)^{-1} \tag{3.8}$$

And the analogue of (3.7) is

$$M^* = (\overline{LD-B})U \tag{3.9}$$

We will adaptively take approximation between (3.6) and (3.8) in our discussion (we explain this later on) but we only express theoretical results for the case (3.6).

The first thing we are interested in is whether the update (3.7) has the potential to be much more powerful than the frozen preconditioner $M = LDU$ for A^*. We display the relation of quantity between updated and frozen preconditioner in the following by using the simple lemma below.

Lemma 3.1. *Let $\|A-LDU\| = \varepsilon\|A\| < \|B\|$. Then the preconditioner from* (3.7) *satisfies*

$$\begin{aligned}\|A^* - M^*\| &\le \frac{\|L(DU - \overline{DU-B}) - B\| + \varepsilon\|A\|}{\|B\| - \varepsilon\|A\|} \cdot \|A^* - LDU\| \\ &\le \frac{\|L\|\|DU - B - \overline{DU-B}\| + \|L-I\|\|B\| + \varepsilon\|A\|}{\|B\| - \varepsilon\|A\|} \cdot \|A^* - LDU\|\end{aligned}$$

Proof.

Easily, We will have

$$\begin{aligned}\|A^* - M^*\| &= \|A - B - L(\overline{DU-B})\| = \|(A - LDU) + L(DU - \overline{DU-B}) - B\| \\ &\le (\varepsilon\|A\| + \|L(DU - \overline{DU-B}) - B\|)\frac{\|B\| - \varepsilon\|A\|}{\|B\| - \varepsilon\|A\|} \\ &\le (\varepsilon\|A\| + \|L(DU - \overline{DU-B}) - B\|)\frac{\|(A - LDU) - B\|}{\|B\| - \varepsilon\|A\|} \\ &= \|A^* - LDU\|\frac{\|L(DU - \overline{DU-B}) - B\| + \varepsilon\|A\|}{\|B\| - \varepsilon\|A\|} \\ &= \|A^* - LDU\|\frac{\|L(DU - B - \overline{DU-B}) + (L-I)B\| + \varepsilon\|A\|}{\|B\| - \varepsilon\|A\|} \\ &\le \|A^* - LDU\|\frac{\|L\|\|DU - B - \overline{DU-B}\| + \|L-I\|\|B\| + \varepsilon\|A\|}{\|B\| - \varepsilon\|A\|}\end{aligned}$$

□

By Lemma 3.1, if the $\overline{DU-B}$ is short distance away from $DU-B$ and $\|L-I\|$ is likely to be small then the multipliers of $\|A^* - LDU\|$ will be smaller than one. We take into consideration that preconditioner modifications to improve the diagonal dominance in practice. When we get a potent preconditioner $M = LDU$, the assumption $\|A - LDU\| = \varepsilon\|A\| < \|B\|$ will be satisfied.

Lemma 3.1 declares two important informations for us. First, we can get the relation of quantity between updated and frozen preconditioner. Second, when $\varepsilon\|A\|$ is small enough, $\overline{DU - B}$ is a approximation which is good enough and L is close enough to a diagonal factor, we will have an accurate preconditioner that **may** be as powerful as a recomputed preconditioner. Image that we have $\|A^* - M^R\| = \delta = \|A - M\|$, where M^R is the a recomputed preconditioner with respect to A^*, so we have $\|A^* - M^R\| \geq \delta$ in general. But Lemma 3.1 do not exclude $\|A^* - M^R\| < \delta$ at all. Hence M^* is potential to be as great as the preconditioner which is recomputed. The update (3.7) has a higher convergence rate than a recomputed preconditioner in some cases.

Lemma 3.1 gives a relation with accuracy according to (3.1). Now we want to introduce a theorem that is related to (3.3) or (3.2). And the theorem express that, under some particular assumptions, the quality of updates **may** be better than recomputed preconditioners if the approximation $\overline{DU - B}$ is suitably chosen. For the purpose of simplification, the scaled updated approximate factor $D^{-1}(\overline{DU - B})$ will be denoted by $\overline{U - D^{-1}B}$ and $U^{-1}(\overline{U - D^{-1}B})$ will be denoted by $I - \overline{U^{-1}D^{-1}B}$.

Theorem 3.2. *Assume that $LDU + E = A$. for some error matrix E and let $\|\overline{U^{-1}D^{-1}B}\|_2 \leq 1/c < 1$ where $\|\cdot\|_2$ denotes the Euclidean norm. Further* *assume that the singular values* σ_i *of*

$$M^* - A^* = L(\overline{DU - B}) - A^* = (I - L)B + L(\overline{DU - B} - (DU + L^{-1}E - B))$$

satisfy

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_t \geq \delta \geq \sigma_{t+1} \geq \cdots \geq \sigma_n$$

for some integer $t, t \ll n$, *and some small* $\delta > 0$. *Let* $(\overline{DU - B})$ *have nonzero main diagonal, and* $D = diag(d_1, \cdots, d_n)$. *Then there exist matrices* F *and* $\triangle$ *such that the stability of* M^* *with respect to* A^* *satisfies*

$$I - (M^*)^{-1}A^* = I - (\overline{DU - B})^{-1}L^{-1}A^* = \triangle + F, \qquad (3.10)$$

with rank($\triangle$)$\leq t$ *and*

$$\|F\|_2 \leq \frac{c}{c-1} \max_i \frac{\delta}{|d_i|} \|L^{-1}\|_2 \|U^{-1}\|_2.$$

Proof.

We have

$$L(\overline{DU - B}) - A^* = L(DU + L^{-1}E - B + \overline{DU - B} - (DU + L^{-1}E - B)) - A^*$$

$$= (I - L)B + L(\overline{DU - B} - (DU + L^{-1}E - B)).$$

By assumption, the singular value decomposition of the latter matrix can be written as

$$(I-L)B + L(\overline{DU-B} - (DU + L^{-1}E - B)) = W\Sigma V^T =$$

$$Wdiag(\sigma_1, \cdots, \sigma_t, 0, \cdots, 0)V^T + Wdiag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)V^T$$

$$\equiv \triangle_1 + F_1, \qquad \text{with rank}(\triangle_1) \leq t$$

and

$$\|F_1\|_2 = \|Wdiag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)V^T\|_2 \tag{3.11}$$

$$= \|diag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)V^T\|_2 \tag{3.12}$$

$$= \|diag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)\|_2 \tag{3.13}$$

$$\leq \delta \tag{3.14}$$

The second equality (3.12) is caused by

$$\|Wdiag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)V^T\|_2^2$$

$$= (Wdiag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)V^T)^T(Wdiag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)V^T)$$

$$= Vdiag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)W^TWdiag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)V^T$$

$$= Vdiag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)diag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)V^T$$

$$= \|diag(0, \cdots, 0, \sigma_{t+1}, \cdots, \sigma_n)V^T\|_2^2$$

And the third equality (3.13) is satisfied by

$$\begin{aligned}
&\|diag(0,\cdots,0,\sigma_{t+1},\cdots,\sigma_n)V^T\|_2\\
&= \max_{\|x\|_2=1} \|diag(0,\cdots,0,\sigma_{t+1},\cdots,\sigma_n)V^T x\|_2\\
&= \max_{\|y\|_2=1} \|diag(0,\cdots,0,\sigma_{t+1},\cdots,\sigma_n)y\|_2\\
&= \|diag(0,\cdots,0,\sigma_{t+1},\cdots,\sigma_n)\|_2\\
&\leq \delta
\end{aligned}$$

Hence

$$L(\overline{DU-B}) - A^* = \triangle_1 + F_1$$

then

$$I-(M^*)^{-1}A^* = I-(\overline{DU-B})^{-1}L^{-1}A^* = (\overline{DU-B})^{-1}L^{-1}\triangle_1 + (\overline{DU-B})^{-1}L^{-1}F_1$$

By setting

$$F \equiv (\overline{DU-B})^{-1}L^{-1}F_1, \quad \triangle \equiv (\overline{DU-B})^{-1}L^{-1}\triangle_1$$

we get (3.10),where $rank(\triangle) \leq t$. The matrix F can be bounded by

$$\|F\|_2 \leq \|L^{-1}\|_2 \|(D(\overline{U-D^{-1}B}))^{-1}\|_2 \delta$$

hence

$$\begin{aligned}
\|F\|_2 &\leq \max_i \frac{\delta}{|d_i|}\|L^{-1}\|_2\|(\overline{U-D^{-1}B})^{-1}\|_2\\
&\leq \max_i \frac{\delta}{|d_i|}\|L^{-1}\|_2\|U^{-1}\|_2\|(I-\overline{U^{-1}D^{-1}B})^{-1}\|_2
\end{aligned}$$

By assumption, $\|\overline{U^{-1}D^{-1}B}\|_2 \leq 1/c < 1$, and consequently

$$\begin{aligned}
\|F\|_2 &\leq \max_i \frac{\delta}{|d_i|}\|L^{-1}\|_2\|U^{-1}\|_2(1-\|\overline{U^{-1}D^{-1}B}\|_2)^{-1}\\
&\leq \frac{c}{c-1}\max_i \frac{\delta}{|d_i|}\|L^{-1}\|_2\|U^{-1}\|_2.
\end{aligned}$$

□

If the matrix F in (3.10) is a zero matrix then the preconditioned system is a rank t update and Krylov subspace methods converges theoretically in at most $t+1$ iterations.

3.2 Practical Manipulation

In [7], in this section, we are going to discuss approximations $\overline{DU-B}$ of $DU-B$ which can be efficiently computed and make the preconditioners that are cheap to put in use. More precisely, updated preconditioners are easily invertible matrices. We will offer three strategies which are classified into *triangular updates* or *unstructured updates* according to the structure of the preconditioner. Likewise, we only present for the case (3.7) but all means can be analogously formulated for $(\overline{LD-B})U$ corresponding to (3.9).

3.2.1 Triangular Update

A strategy which not only is obvious and effective but also preserves the triangular structure which we consider entries from is to set

$$\overline{DU-B} \equiv triu(DU-B) \tag{3.15}$$

,where *triu* denotes the sparsified upper triangular part which also includes the main diagonal part, herefore the updated preconditioner will be

$$M^* = L(DU - triu(B)) \tag{3.16}$$

Due to the sparsity pattern of (3.15) is triangle, we will call this update *triangular updates.* We know that M^* can be acquired totally for free because we just take only one triangular sweep with the triangular part of B, if we store U and B separately. If the sparsity patterns of *triu*(B) and U are close enough then it nearly costs us nothing in practice.

And the analogue of (3.16) is

$$M^* = (LD - tril(B))U \tag{3.17}$$

These two updates (3.16) and (3.17) are powerful in many problems when one triangular part of B is clearly dominates the other. we will show this result in our experiments. However, they only take into account one triangular part of the difference matrix B, the information from the other part of B will be lost. This will lead to weak convergence in some applications.

3.2.2 Unstructured Update

In this section, for the purpose of avoiding the information from each triangular part of the difference matirx B, we will propose two strategies to get the same idea of updated preconditioner which is easily to get inverse matrix. But these two strategies give rise to matrix which is generally not triangular.

Denote the matrices diag($\overline{DU-B}$) by $\tilde{D}$, and $\tilde{D}^{-1}(\tilde{D}-\overline{DU-B})$ by $\tilde{B}$, respectively. We have zero diagonal on the main diagonal of $\tilde{B}$ and

$$\overline{DU-B} = \tilde{D}(I-\tilde{B}) \tag{3.18}$$

For the first strategy, we are motivated by Sherman-Morisson formula. For example, when $\tilde{B} = \beta e_i e_j^T$ for some $1 \leq i, j \leq n, i \neq j$,and remember that we assume $\overline{DU-B}$ is nonsingular, so is $I-\tilde{B}$. Then ,by Sherman-Morisson formula, we have

$$(I-\tilde{B})^{-1} = I + \frac{\beta e_i e_j^T}{(1-e_j^T e_i)} = I + \beta e_i e_j^T = I + \tilde{B} \tag{3.19}$$

From (3.19), we know the inverse of $(I-\tilde{B})$ is the identity matrix which is modified by an off-diagonal entry β at the position (i,j), so it costs nothing and is fill-in free. And we know $(I-\tilde{B})$ is a particular Gauss-Jordan transformation. According to this motivation, we are going to find approximations $\overline{DU-B}$ of $DU-B$ such that the scaled matrix $I-\tilde{B}$ can be transformed by a product of Gauss-Jordan transformations

$$(I-e_{i_1}\tilde{b}_{i_1*})(I-e_{i_2}\tilde{b}_{i_2*})\cdots(I-e_{i_K}\tilde{b}_{i_K*}), \quad K \leq n-1 \tag{3.20}$$

where $\tilde{B} = (\tilde{b})_{ij}$. We denote by $row(i)$ the sparsity structure of a row i of $\tilde{B}$ (with zero diagonal). It is very cheap for multiplication of $(I - \tilde{B})^{-1}v$ with a given vector v, as we can see from Observation 3.3.

Observation 3.3. *The number of operations for multiplying a vector by a matrix of the form* (3.20) *or its inverse is at most* $2\sum_{j=1}^{K} |row(i_j)|$.

Proof.

The number of element in each row of the matrix which can be written as (3.20) is $|row(i_j)|$ elements. Every element in the i_j-th row needs to multiply a element in the given vector so there will be $\sum_{j=1}^{K} |row(i_j)|$ operations. And the number of operations for summing up the products is at most $\sum_{j=1}^{K} |row(i_j)|$. Hence the total number of operations is $2\sum_{j=1}^{K} |row(i_j)|$.

□

We know that (3.16) is special case of (3.20) because of the well-known fact that unit upper triangular matrix $I - \tilde{B}$ from (3.18) can be trivially written as the product $R_{n-1} \cdots R_1$ of $n - 1$ elementary triagular matrices ,where $R_i = I - e_i \widetilde{b_{i*}}$ for $i = 1, \cdots, n - 1$. In the following theorem, we discuss a necessary and sufficient condition for the existence of a decomposition of $I - \tilde{B}$ of the form (3.20).

Theorem 3.4. *Let* $I - \tilde{B} = I - \sum_{j_l:l=1,\cdots,K} e_{j_l}\tilde{b}_{j_l*}$. *Then*

$$I - \tilde{B} = (I - e_{i_1}\tilde{b}_{i_1*})(I - e_{i_2}\tilde{b}_{i_2*})\cdots(I - e_{i_K}\tilde{b}_{i_K*}) \tag{3.21}$$

if and only if

$$i_l \notin \bigcup_{k=1}^{l-1} row(i_k) \;\; for \;\; 2 \leq l \leq K \tag{3.22}$$

for all $i_1, \cdots, i_K$ *such that* $\{j_1, \cdots, j_K\} = \{i_1, \cdots, i_K\}$.

Proof.

The equivalence of (3.21) and (3.22) follows from the orthogonality of the unit vector e_{i_l} with respect to all $\tilde{b}_{i_k*}$ for $k < l$, $1 \leq l \leq K$. □

Now, we are going to introduce the first strategy for finding the approximation $\overline{DU - B}$ with $I - \tilde{B}$ satisfying (3.21).In algorithm (3.2.2.1), we first initialize a **candidate rows** $\mathcal{R}$ by $\{1, \cdots, n\}$. In each step, after choosing a row i, we update the candidate rows $\mathcal{R}$ by removing all the rows $j \in \mathcal{R}$ for which $\tilde{b}_{ij} \neq 0$. At the end of the algorithm (3.2.2.1), we will have a sequence of row indices $i_1, \cdots i_K$, where $K \leq n - 1$, which are made from the algorithm.

Algorithm 3.2.2.1 Find matrix $\overline{DU - B}$ such that it, scaled by its diagonal, can be written in the form (3.21)

```
1: Set $\mathcal{R} = \{1, \cdots, n\}, K = 0$
2: for $k = 1, \cdots, n$ do
3:     Set $row(k) = \{i | i \neq k \wedge |(DU - B)_{ki}| \neq 0\}$
4:     Set $p_k = \sum_{j \in row(k)} |(DU - B)_{ki}|$
5: end for
6: while $\mathcal{R} \neq 0$ do
7:     Choose a $row$ $i \in \mathcal{R}$ maximizing $p_i - \sum_{j \in \mathcal{R} \cap row(i)} p_j$
8:     Set $K = K + 1, i_k = i, \mathcal{R} = \mathcal{R} \backslash \{row(i_K) \cup i\}$
9: end while
```

We can use the row indices $i_1, \cdots i_K$ determined from the algorithm to construct the approximation in (3.18) with $I - \tilde{B}$ which can be written as the product (3.20). The heuristic criterions in step (7) and (8) not only are the point in finding the row of $DU - B$ with largest entries but also spur the choice of a row to remove the candidate rows $\mathcal{R}$ with small entries. To equalize the two heuristics,we may perform a weighting parameter ω when running the algorithm and substitute step (7) with the new step (3.23).

$$7' \qquad \textit{Choose a row } i \in \mathcal{R} \textit{ maximizing } p_i - \omega \cdot \sum_{j \in \mathcal{R} \cap row(i)} p_j \qquad (3.23)$$

It may be happen that, if there are fewer nonzero entries in the hunted rows, we may have more factors of (3.21). Hence we may introduce a drop-tolerance *tol* into step (3) and replace step (3) with

$$3' \qquad Set \quad row(k) = \{i | i \neq k \wedge |(DU - B)_{ki}| > tol\} \tag{3.24}$$

Other than tolerance dropping, we can also sparsify the structure of the matrix $DU - B$ had a foothold on the given mask in order to enhance the efficientibility of the strategy. Sparsification stimulates the the chosen candidate rows $\mathcal{R}$ covering as much rows as possible by Gauss-Jordan transformation as well as gives rise to less expensive matvecs (matrix-vector computations) with the inverse of (3.20).

The second strategy, we introduce below, for finding the approximation $\overline{DU - B}$ based on Gauss-Jordan transformation will be more systematic and beautiful, is characterized by bipartite graph model.We define the bipartite graph of $(DU - B)$ as $G(DU - B) = (R, C, E)$, where $R = \{1, \cdots, n\}$, $C = \{1', \cdots, n'\}$ and $E = \{(i, j') | (DU - B)_{ij'} \neq 0\}$. The following Theorem (3.5) give us a first glance through this idea.

Theorem 3.5. *Assume that $T = (V_T, E_T)$ is a spanning forest of $G(DU - B)$ such that $\{(i, i') | 1 \leq i \leq n\} \subseteq E_T$ where V_T is the vertices of T and E_T is the edge of T. We define entries of the matrix $\overline{DU - B} \in \mathbb{R}^{n \times n}$ by*

$$\overline{(DU - B)}_{ij} = \begin{cases} (DU - B)_{ij} & \text{if } (i, j') \in E_T \\ 0 & \text{otherwise} \end{cases}$$

scaled by its diagonal entries as in (3.18)*. Then* $\overline{DU - B}$ *can be expressed as a product of the form* (3.20)*.*

Proof.

There are two kind of T, connected or unconnected. If T is unconnected then T is a spanning forest. Then the spanning forest T will be pieces of connected parts. Each connected part associates a block diagonal submatrix of $\overline{DU - B}$, and submatrices corresponding to individual connected parts can be mutually multiplied in any order. And it lead the same result and causes no fill-in. So we can only take T is connected into consideration without loss of generality.

In the following of the proof, we will present how to find the indices of the Gauss-Jordan transformations from the left to the right for T is connected.

Due to $\{(i, i')|1 \leq i \leq n\} \subseteq E_T$ and T is connected, T contains at most $n - 1$ edges (i, j') with $i \neq j$. We can find a free row vertex $i \in R$ in T which is in T incident only to the edge (i, j') such that there is an edge $(k, j') \in E_T$ for some k. Set $i_1 = i$. Then remove from T the vertices $i \in R$, $j' \in C$ and all edges incident to them. Again, there exists a free row vertex in the updated tree T. We reduplicate choice of free row vertices and update T in this way, we will get the sequence of indices $i_1, \cdots, i_{n-1}$ for $I - \tilde{B} = (I - e_{i_1}\tilde{b}_{i_1 *})(I - e_{i_2}\tilde{b}_{i_2 *}) \cdots (I - e_{i_{n-1}}\tilde{b}_{i_{n-1} *})$ which is scaled the diagonal entries from $\overline{DU-B}$. $\square$

We marshal theorem (3.5) to the algorithm (3.2.2.2) which could find an approximation $\overline{DU-B}$ of $DU-B$. At the end, $\overline{DU-B}$ would be written as a product of Gauss-Jordan transformations.

Algorithm 3.2.2.2 Approximate $DU-B$ such that (3.21) is satisfied based on a bipartite graph of $DU-B$

```
1: Find spanning forest $T = (V_T, E_T)$ of $G(DU-B)$ of maximum weight with edge weights $w_{ij} = |(DU-B)_{ij}|$ for $(i, j') \in E_T$ such that $\{(i, i')|1 \leq i \leq n\} \subseteq E_T$.
2: Find the entries of $\tilde{B}$ (and corresponding entries of $\overline{DU-B}$) as well as a feasible ordering of Gauss-Jordan factors for $i_1, \cdots, i_{n-1}$ in (3.20) with Theorem (3.5).
3: For each $k = 2, \cdots, n$ add to $\overline{DU-B}$ all entries $(DU-B)_{i_k l}$ of $DU-B$ such that $l \in \{i_1, \cdots, i_{k-1}\}$.
```

The last step of algorithm (3.2.2.2) will put much more nonzero entries than the $2n-1$ entries given by the weighted spanning forest into $\overline{DU-B}$. This is an allowed procedure which still can be written as a product of Gauss-Jordan transformations because of Theorem (3.4). The computational complexity of weighted minimum spanning forest for the Kruskal algorithm (But actually, here, we are applying weighted maximum spanning forest) is $O(m \log m)$ and for the Prim algorithm is $O(n+m \log m)$, where m is the number of edges in the graph G. Note that we begin with the partial spanning tree with the set of edges $\{(i,i')|1 \leq i \leq n\}$ in the Kruskal algorithm. The Kruskal algorithm in some cases will be time-consuming but this algorithm provided us powerful updates. In order to save time, we can do the same things in Algorithm (3.2.2.1). We can sparsify $DU - B$ by omitting entries smaller than a pre-given drop tolerance *tol*. This idea decreases not only the number of edges m but also the complexity.

We know that, from Lemma (3.1), the quality of the preconditioner $DU - B$ will play an important role in the power of the preconditioner $M^* = L(\overline{DU - B})$. For the purpose of using the most effective type of update, we switch between the equality (3.16) and the equality (3.17) adaptively in our experiments. The criteria for switch is in terms of the weighting of both triangular parts of B. We will use the the triangular part of B which has the bigger weight.

Despite of the truth that the updated preconditioner loses some information on the system matrix, it is a preconditioner that is sometimes better than the recomputed preconditioner. For instance, as we have shown theoretically in Lemma (3.1) and Theorem (3.2), the updated preconditioner has the potential to be better that preconditioner which is recomputed. It occasionally occurred that updated preconditioner is related to previous decomposition which is more diagonally dominant than a recomputed preconditioner. The updated precondtioner can inherit the property of stable and may also stabilizes the less stable factors.

The next chapter will be devoted to some numerical experiments. We will compare all the ideas that we introduced in chapter 3.

Chapter 4

Numerical Result

To finish our work, we mainly consult [1], [2], [3], [7], [8] and [10]. In this thesis, we only use our own Matlab code to complete our experiments comparing to the code , in [7], which is a combination of Matlab and Fortran. We also optimize our Matlab code so that the numerical results have excellent behavior in some cases and are also reliable. The Matlab is a new version 8.0 and the computer is nearly high standard as compared with [7]. However, the updates are still fantastic for these facilities in some cases.

In this chapter, we will present numerical results for our experiments with preconditioned krylov subspace methods for solving a sequence of linear systems and compare the strategies that we introduce in chapter (3) with recomputed preconditioners and frozen preconditioners. We also give the numerical results with many kinds of incomplete LU-decompositions to show the circumstances that we will meet.

All of the tests were performed by Matlab version 8.0. The codes were run on a computer with Xeon E5160, 3.0GHz processor, 2 cores, 4M L2 cache, 32GB RAM memory. BiCGSTAB iterative method with left preconditioning was used to be our accelerator. Iterations were stopped when the Euclidean norm of residual was decreased by ten orders of magnitude.

Our test problem is a two-dimensional nonlinear convection-diffusion model problem. It has the form

$$-\Delta u + Ru(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}) = 2000x(1-x)y(1-y)$$

on the unit square, discretized by 5-point finite differences with natural ordering on a uniform 70×70 grid. The initial approximation is the discretization of $u_0(x, y) = 0$. And we choose $R = 50$. Then we have a sequence of matrices with 24220 nonzeros each. In the following tables, the number of nonzero is denoted by nnz and the average number of nonzero of preconditioner is represented by psize.

The update techniques are, in particular, beneficial when it is expensive to recompute preconditioners. We begin with ILU (incomplete LU decomposition) with threshold 0.1. This kind of ILU is very expensive in Matlab, i.e.it takes lots of time to compute ILU(0.1). Note that, in Matlab, loop is much slower that array operation but there are some operations that we can not use array operations due to data dependency so we are much more concerned to the numbers of BiCGSTAB iterations in our experiments.

In table (4.1), we present the numbers of BiCGSTAB iterations which is needed to solve individual linear systems for several preconditioning strategies we introduced in Chapter 3, and time including solving the sequence of linear systems and constructing the preconditioners. In the second row, the 'Recomp' represents that the preconditioner ILU(0.1) will be recomputed for each matrix separately. The method 'Freeze' displays that the preconditioner will be computed only once at the first linear system and reused for the other linear systems. 'Triangular' denotes the triangular updated we introduced in in Chapter 3. Algorithm 3.2.2.1 and Algorithm 3.2.2.2 in Chapter 3 are just denoted by the 'Algorithm 3.2.2.1' and 'Algorithm 3.2.2.2'. We use Kruskal algorithm to construct our spanning forest for'Algorithm 3.2.2.2'.

In table (4.1), we can see that ILU(0.1)-decomposition gives the test problem a very powerful preconditioner but computing ILU(0.1)-decomposition costs a lot. Thought freezing the ILU(0.1)-decomposition can reduce the computational time and bring much higher number of BiCGSTAB iterations, the overall time to solve these matrices is still shorter. Triangular updates has a wonderful performance in this table. Note that we choose the triangular part adaptively based on the magnitudes of the difference matrix

B. While the numbers of iterations for triangular updates are all but as low as the iteration number of recomputation, and most important of all, we save some significant time comparing to recomputed preconditioner. The iteration numbers for Algorithm 3.2.2.1 are a little higher than triangular updates but obviously lower that frozen preconditioners. Iteration counts for Algorithm 3.2.2.2 are comparable to triangular updates.

Table 4.1.

Numbers of iteration for individual linear systems and

totally elapsed time with ILU(0.1)

ILU(0.1), psize ≈ 24000					
Matrix	Recomp	Freeze	Triangular	Algorithm 3.2.2.1	Algorithm 3.2.2.2
$A^{(0)}$	42	42	42	42	42
$A^{(1)}$	34	45	36	38	39
$A^{(2)}$	27	51	34	34	37
$A^{(3)}$	27	60	35	36	36
$A^{(4)}$	26	77	34	37	37
$A^{(5)}$	22	82	31	36	37
$A^{(6)}$	22	83	31	35	35
$A^{(7)}$	19	66	27	29	28
Elapsed time	1.273 s	1.021 s	0.732 s	9.812 s	26.104 s

As we can see in Table 4.1, the elapsed times for 'Algorithm 3.2.2.1' and 'Algorithm 3.2.2.2' are really pessimistic. There are two explanation for this phenomenon. First, as we have already said, some procedures in Algorithm 3.2.2.1 and Algorithm 3.2.2.2 are data dependency so we have no choice but to use loop in our code. For example, the loop from step(6) to step(9) in the Algorithm 3.2.2.1. And the loop in Matlab costs much more time. Second, the unstructured updates from Algorithm 3.2.2.1 and Algorithm 3.2.2.2 may not be triangular forms which can not be competitive with the highly optimized operations for back-substitution and forward-substitution in Matlab. Hence the elapsed time for Algorithm 3.2.2.1 and Algorithm 3.2.2.2 are somehow not comparable to the other strategies. So, in Table 4.1, we focus on the numbers of iterations of Algorithm 3.2.2.1 and Algorithm 3.2.2.2 than the elapsed times of them.

Using the drop tolerance has an influence on the number of nonzeros and the computational time as well in Algorithm 3.2.2.1 and Algorithm 3.2.2.2. In Algorithm 3.2.2.1, We can not overestimate or underestimate the drop tolerance. If we underestimate this parameter, the updated preconditioner will be pretty sparse because we have only a small number of factors of the form (3.16). When we overestimate it then the updated preconditioner will be very sparse as well since small number of nonzero entries are covered by Gauss-Jordan transformations. For Algorithm 3.2.2.2 the case is more simple, we set the *tol* to choose the $2n - 1$ largest entries and as few other entries as necessary to construct the spanning forest.

We have optimized the *tol* in Algorithm 3.2.2.2 between one orders of meg-nitude and choose $tol = 1$ in order to get the smaller numbers of iterations, leading to an overall time of 32.075 seconds. We do not choose a perfect *tol* in Algorithm 3.2.2.1 but choose the same value of $tol = 1$ because this value produces a scaled system matrix to keep reasonable number of nonzeros in a row in Matlab which makes the smaller number of factors due to the most of the whole elapsed time is to find the indices $i_1, \cdots, i_K$ which takes much more time.

Apart from the sensitivity of dropping-tolerance *tol*, the omega ω is not easy to be affected in this test problem. In Figure 4.1, we show the total number of BiCGSTAB iterations for solving the eight linear systems for the values of ω. We can see that the overall number of iterations do not change a lot from $\omega = 0.6$ to 2. For the values smaller than 0.6, criterion (7) of Algorithm 3.2.2.1 overemphasize the weight of the chosen candidate row which lead to a bad approximation of $DU - B$. In all our experiments, we use $\omega = 2$ for Algorithm 3.2.2.1.

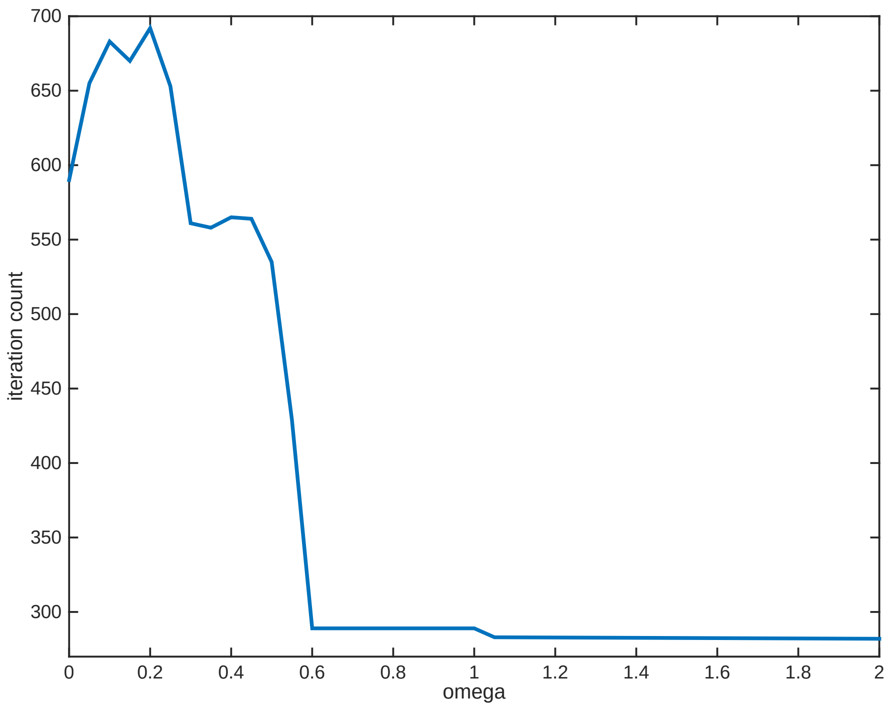

Figure 4.1. Iteration counts for Algorithm 3.2.2.1 in dependency of ω

In Table 4.2 and Table 4.3, we present the accuracies $\|A^{(i)} - M^*\|_F$ and $\|DU - B - \overline{DU - B}\|_F$, respectively, where $\|\cdot\|_F$ represents the Frobenius norm. $\|A^{(i)} - M^*\|_F$ tells how far between the linear systems and the preconditioners for each strategies. $\|DU - B - \overline{DU - B}\|_F$ gives the information about the difference of $DU - B$ and our approximations $\overline{DU - B}$(equality of $\overline{DU - B}$). For this test problem, we do not meet any problem of stability of the preconditioner. From these two Table, we can see that the values correspond to the numbers of BiCGSTAB iterations.

Table 4.2.

Accuracies of $\|A^{(i)} - M^*\|_F$

ILU(0.1), psize $\approx$ 24220					
Matrix	Recomp	Freeze	Triangular	Algorithm 3.2.2.1	Algorithm 3.2.2.2
$A^{(0)}$	28.51	28.51	28.51	28.51	28.51
$A^{(1)}$	27.51	37.20	37.51	78.66	78.41
$A^{(2)}$	26.58	44.00	43.09	79.81	79.57
$A^{(3)}$	26.61	52.70	49.88	81.15	80.91
$A^{(4)}$	28.45	62.91	57.78	82.73	82.49
$A^{(5)}$	28.17	66.71	60.74	83.35	83.11
$A^{(6)}$	28.21	66.57	60.64	83.33	83.09
$A^{(7)}$	28.21	66.57	60.64	83.33	83.09

Table 4.3.

Accuracies of $\|DU - B - \overline{DU - B}\|_F$

ILU(0.1), psize ≈ 24220			
Matrix	Triangular	Algorithm 3.2.2.1	Algorithm 3.2.2.2
$A^{(1)}$	16.93	23.90	26.61
$A^{(2)}$	23.78	33.52	35.50
$A^{(3)}$	31.49	44.32	45.84
$A^{(4)}$	39.92	56.08	57.29
$A^{(5)}$	42.96	60.31	61.44
$A^{(6)}$	42.84	60.16	61.29
$A^{(7)}$	42.84	60.16	61.29

In Table 4.1, we can see that triangular updates gives more powerful and efficient preconditioner than the other preconditioners. But this is not clear in Table 4.4. We use ILU-factorization with drop tolerance 0.005 as an initial preconditioner in Table 4.4. The tolerance *tol* in Algorithm 3.2.2.1 and Algorithm 3.2.2.2 are still 1. Although the elapsed time of Algorithm 3.2.2.1 and Algorithm 3.2.2.2 are much longer than Triangular updates in Matlab, Algorithm 3.2.2.1 will be quicker than Triangular updates in Fortran(cf. [7]). There are two explanations to these issues. First, it is caused by the loop in Matlab. We have already told that the loop comparing to vector operations in Matlab may be expensive. And The computational complexity of Algorithm 3.2.2.1 and Algorithm 3.2.2.2 are not very high because it is just linear in the number of matrix nonzeros so Algorithm 3.2.2.1 may be faster than Triangular updates in Fortran(cf. [7]) or C. Second reason is that the numbers of BiCGSTAB iterations is much smaller that the Triangular updates. This is caused by the truth that Triangular updates take only one part of information and the updates from Algorithm 3.2.2.1 will take larger entries which has more important information. In the following, we will describe the second reason in more detail. We randomly choose a difference matrix from the middle of the sequence $B = A^{(0)} - A^{(4)}$. For other choice of matrices $A^{(i)}$, we have similar phenomenon. Though the nonzeros are evenly distributed over both triangular parts of $LD - B$, $\|stril(LD - B)\|_F \approx 142.95$ and

$\|striu(DU - B)\|_F \approx 39.39$, where *stril*$(\cdot)$ and *striu*$(\cdot)$, represent the strict lower and upper triangular part, respectively. According to the two values, the lower triangular part dominates the upper part. But there may be some important entries in upper part and they are neglected by the Triangular updates. On the contrary, unstructured updates consider both triangular parts. This is reflected by, in the Frobenius norms,$\|stril(\overline{LD - B})\|_F \approx 142.95$ and $\|striu(\overline{LD - B})\|_F \approx 39.39$ for the approximation $\overline{LD - B}$ from Algorithm 3.2.2.1 ,and $\|stril(\overline{LD - B})\|_F \approx 142.43$ and $\|striu(\overline{LD - B})\|_F \approx 39.39$ for the approximation $\overline{LD - B}$ from Algorithm 3.2.2.2. Since the step 3 of Algorithm 3.2.2.2 inserts more nonzero entries, the values of $\|striu(\overline{LD - B})\|_F$ and $\|stril(\overline{LD - B})\|_F$ will be bigger in some other cases. In our experiments, this phenomenon is not obvious since we are more conerned the numbers of iterations which leads both the unstructure updates contain more nonzero entries. Also note that the number of nonzeros of triangular updates is more or less 61500 nevertheless unstructured updates have smaller number of nonzeros, around 61000, which may let Algorithm 3.2.2.1 and Algorithm 3.2.2.2 more cheaper. This gap will increase when *tol* is overestimated or underestimated as we have said before.

Algorithm 3.2.2.1 needs more elapsed time to implement in Matlab but it will be faster than Triangular method when it is performed in Fortran.(cf. [7])with this kind of preconditioner.

Table 4.4.

Numbers of iteration for individual linear systems and

totally elapsed time with ILU(0.005)

ILU(0.005), psize $\approx$ 61486				
Matrix	Freeze	Triangular	Algorithm 3.2.2.1	Algorithm 3.2.2.2
$A^{(0)}$	17	17	17	17
$A^{(1)}$	46	28	29	29
$A^{(2)}$	66	33	32	32
$A^{(3)}$	89	54	34	34
$A^{(4)}$	128	118	34	38
$A^{(5)}$	136	136	34	37
$A^{(6)}$	144	135	32	34
$A^{(7)}$	136	125	31	30
Elapsed time	1.493 s	1.317 s	9.485 s	32.075 s

There is another situation we will meet. when the recomputed preconditioners is straightforward and cheaper then most of the updates may not be efficient. In Table 4.5, we use ILU-decomposition with level one (i.e. ILU(1)) as the initial preconditioner. The number of nonzeros of the initial preconditioner is approximately 33000. Elapsed time to compute the ILU(1) is very small, the most of overall time is taken on solving with BiCGSTAB. Note that the iteration counts for Triangular is also a little worse than recomputation. We have the same results for ILU-decomposition with level zero (i.e. ILU(0)) in Table 4.6 as well.

Table 4.5.

Numbers of iteration for individual linear systems and

totally elapsed time with ILU(1)

ILU(1), psize $\approx$ 33742			
Matrix	Recomp	Freeze	Triangular
$A^{(0)}$	28	28	28
$A^{(1)}$	22	44	29
$A^{(2)}$	17	52	27
$A^{(3)}$	16	69	33
$A^{(4)}$	14	92	45
$A^{(5)}$	11	105	48
$A^{(6)}$	12	92	44
$A^{(7)}$	9	94	42
Elapsed time	0.295 s	0.813 s	0.507 s

Table 4.6.

Numbers of iteration for individual linear systems and

totally elapsed time with ILU(0)

ILU(0), psize ≈ 24220			
Matrix	Recomp	Freeze	Triangular
$A^{(0)}$	41	41	41
$A^{(1)}$	34	45	36
$A^{(2)}$	27	50	34
$A^{(3)}$	25	60	35
$A^{(4)}$	22	76	34
$A^{(5)}$	21	79	31
$A^{(6)}$	19	77	31
$A^{(7)}$	16	68	27
Elapsed time	0.345 s	0.694 s	0.506 s

References

[1] John E Dennis Jr and Robert B Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*, volume 16. Siam, 1996.

[2] Hans Petter Langtangen. *Computational partial differential equations: numerical methods and diffpack programming*. Springer Berlin, 1999.

[3] Randall J LeVeque. Finite difference methods for differential equations. *Draft version for use in AMath*, 585(6), 1998.

[4] Angelo Lucia. An explicit quasi-newton update for sparse optimization calculations. *MATHEMATICS of computation*, 40(161):317–322, 1983.

[5] LK Schubert. Modification of a quasi-newton method for nonlinear equations with a sparse jacobian. *Mathematics of Computation*, 24(109):27–30, 1970.

[6] DF Shanno. On variable-metric methods for sparse hessians. *Mathematics of Computation*, 34(150):499–514, 1980.

[7] Jurjen Duintjer Tebbens and Miroslav Tuma. Efficient preconditioning of sequences of nonsymmetric linear systems. *SIAM Journal on Scientific Computing*, 29(5):1918–1941, 2007.

[8] Jurjen Duintjer Tebbens and Miroslav Tuma. Improving triangular preconditioner updates for nonsymmetric linear systems. In *Large-scale scientific computing*, pages 737–744. Springer, 2008.

[9] Ph L Toint. On sparse and symmetric matrix updating subject to a linear equation. *Mathematics of Computation*, 31(140):954–961, 1977.

[10] Stephen J Wright and Jorge Nocedal. *Numerical optimization*, volume 2. Springer New York, 1999.