Approximate Generalized Inverse Preconditioning Methods for Least Squares Problems

Xiaoke Cui

DOCTOR OF
PHILOSOPHY

Department of Informatics
School of Multidisciplinary Sciences
The Graduate University for Advanced Studies (Sokendai)

2009

September 2009

Thesis advisor
Ken Hayami

Author
Xiaoke Cui

Approximate Generalized Inverse Preconditioning Methods for Least Squares Problems

Abstract

A basic problem in science is to fit a model to observations subject to errors. It is clear that the more observations that are available the more accurate will it be possible to calculate the parameters in the model. This gives rise to the problem of "solving" an overdetermined linear or nonlinear system of equations. When enough observations are not available, it gives rise to underdetermined systems. Overdetermined systems together with underdetermined systems are called *least squares problems*. It can be shown that the solution which minimizes a weighted sum of the squares of the residual is optimal in a certain sense. These solutions are called *least squares solutions*.

Least squares problems are usually written in the form

$$\min_{x\in\mathbb{R}^n} \|b - Ax\|_2, \quad A \in \mathbb{R}^{m\times n}, \quad b \in \mathbb{R}^n, \tag{0.1}$$

where the norm $\|\cdot\|_2$ stands for 2-norm. When A is large and sparse, it is advantageous to apply iterative methods to the normal equations $A^T(Ax - b) = 0$ or $AA^Ty - b = 0$. Since the condition number of A^TA or AA^T is the square of that of A, when A is ill-conditioned, preconditioning for the iterative methods becomes necessary.

In this thesis, we consider constructing preconditioners for some Krylov subspace it-

erative methods to solve least squares problems more efficiently. We especially focused on one kind of preconditioners, in which preconditioners are the approximate generalized inverses of the coefficient matrices of the least squares problems. We proposed two different approaches for how to construct the approximate generalized inverses of the coefficient matrices: one is based on the *Minimal Residual* method with the steepest descend direction, and the other is based on the *Greville's Method* which is an old method developed for computing the generalized inverse based on the rank-one update. And for these two preconditioners, we also discuss how to apply them to least squares problems. Both theoretical issues and practical implementation issues about the preconditioning are discussed in this thesis. Our numerical tests showed that our methods performed competitively rank deficient ill-conditioned problems. As an example of problems from the real world, we apply our preconditioners to the linear programming problems, where many large-scale sparse least squares problems with rank deficient coefficient matrices arise. Our numerical tests showed that our methods showed more robustness than the Cholesky decomposition method.

Contents

Title Page i
Abstract iii
Table of Contents v
Acknowledgments vii
Dedication x

1 Introduction **1**

2 Generalized Inverse **5**

3 Methods for Solving Least Squares Problems **11**
3.1 Direct methods for solving least squares problems 11
3.2 Iterative methods for solving least squares problems 13
3.3 GMRES methods for least squares problems 17

4 Approximate Generalized Inverse Preconditioning Methods **24**
4.1 The approximate inverse preconditioning methods 24
4.2 The approximate generalized inverse preconditioning with minimizing Frobenius norm approach 28
4.2.1 Left Preconditioning 28
4.2.2 Right Preconditioning 33
4.3 Numerical Examples 37
4.4 Conclusion 42

5 Greville's Methods for Preconditioning Least Squares Problems **44**
5.1 Greville's method 44
5.2 Global Algorithm for General Matrices 48
5.3 Vector-wise Algorithm for General Matrices 54
5.4 Greville Preconditioning Method for Full Column Rank Matrices 58
5.5 Two Important Issues about Preconditioning the Least Squares Problems . . 65
5.5.1 Equivalence Condition 66

5.5.2 Breakdown Free Condition 70
5.6 Implementation Consideration 74
5.6.1 Detect Linear Dependence 74
5.6.2 Right-Preconditioning Case 77
5.7 Numerical Examples 79

6 Applications to Linear Programming Problems 86
6.1 Linear programming 87
6.2 Application of the Greville's method to linear programming problems . . . 95
6.2.1 When a feasible initial solution is available 95
6.2.2 When a feasible initial solution is not available 97
6.3 Numerical Examples 102
6.4 Conclusion 110

7 Conclusion 112

Bibliography 115

Acknowledgments

Completing this doctoral work is one of the most wonderful experiences to me. It is not only about writing a paper and presenting it, but also about learning, discussing, coding, and accepting suggestions to improve. While I was writing this thesis, I felt I was going over what knowledge I have learnt, what researches I have done, the good things and bad things happened to me in the past three years, and especially the people who showed up into my life. Without them, I would not be sitting here writing my thesis.

First I would like to thank my supervisor Professor Ken Hayami, or in Japanese, Hayami Sensei. He is the nicest advisor I could ever ask for. Since the first day I met him, I have been learning from him. I am always excited by his good ideas and the discussions between us. Hayami Sensei has an extremely wide range of knowledge, from theoretical fields to practical fields. Every time I had difficulties, he always inspired me with his good ideas, taught me the new knowledge that I need to solve my problems, and refer me to good papers and books. Hayami Sensei is good at asking questions. He always comes up with a lot of good questions in our seminars. Thanks to his questions, I have been stimulated and have improved a lot. I also want to thank Hayami Sensei for his generosity, because throughout my three years, I was supported by the MEXT scholarship that Hayami Sensei applied for me.

I also want to say thank you to Professor Yimin Wei. Professor Wei is my supervisor during my undergraduate and master course in Fudan University. He is a great advisor, a very interesting man, and a close friend to me as well. It was Professor Wei who helped me to make the decision to go for a doctoral degree. Without him, I would not have the opportunity to come to Japan to continue my study.

Our research group has always been very small. Dr. Jun-Feng Yin was a post-doc of Professor Hayami. He really helped me a lot when the first time I came to Japan. He showed me around, helped me to make new friends, took me to conferences, and a lot and a lot.... I cannot even remember how many times he helped me out. He is also a brilliant man, open-minded, full of good ideas, and willing to share. I also learnt a lot of coding skills from him. Mr. Keiichi Morikuni is another Ph. D student in our group. He just came one year ago. He is an intelligent and hard working guy. Thanks for teaching me Japanese.

On the foreign front, I must thank Professor Michael Eiermann and Professor Miroslav Tůma for inviting me to visit and the fascinating discussions and suggestions.

There are many others who have been helping me throughout these three years. Thank Ms. Miyuki Kobayashi and Ms. Yasuko Umebayashi, from International Affairs and Education Support Team, for taking good care of we foreign students. And our group's secretary Ms. Suzuki is surely the kindest person; she did a lot of document work for me. I would also like to thank the internship students who were in our group. Although they came and left pretty quickly, we had a lot of fun together.

My fascination with the mathematical world is undoubtedly due to the influence of my father, Zhenghua. He teaches mathematics himself and he is the one who encouraged me to choose the Mathematical department when I was in the university. Thank all my family members, my father Zhenghua, my mother Shuzhen and my sister Xiaoguang, who are always being there for me, supporting every decision I made. Although I am far from you all, I am missing you everyday.

Also my gratitude is devoted to Professor Shini'chi Satoh, Professor Takeaki Uno, Professor Takashi Tsuchiya and Professor Masaaki Sugihara, who are my committee members during the whole thesis evaluation process. Especially Professor Takashi Tsuchiya, he taught me a lot and passed me his test examples.

Dedicated to my father Zhenghua,
my mother Shuzhen,
and my sister Xiaoguang.

Chapter 1

Introduction

This thesis focuses on solving the least squares problem,

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2, \quad A \in \mathbb{R}^{m \times n}, \quad b \in \mathbb{R}^m, \tag{1.1}$$

where the rectangular matrix A can be full rank or rank deficient. The development of the basic modern numerical methods for solving linear least squares problems took place in the late sixties. The QR decomposition using Householder transformations was developed by Golub and published in 1965 [18]. The implicit QR algorithm for computing the singular value decomposition (SVD) was developed at about the same time by Kahan, Golub, and Wilkinson, and the final algorithm was published in 1970. These matrix decompositions have since been developed and generalized to a high level of sophistication. Great progress has been made in the last two decades for generalized and modified least squares problems and in direct and iterative methods for large sparse problems.

For solving least squares problem $\min_x \|Ax - b\|_2$ where A is large and sparse, it-

erative methods can be applied to the normal equation $A^T(Ax - b) = 0$. An important class of iterative methods is the Krylov subspace methods, which in step k seeks an approximation x_k which minimizes a quadratic error functional in the Krylov subspace $x_k \in x^{(0)} + \mathcal{K}_k(A^TA, s^{(0)})$, $s^{(0)} = A^T(b - Ax^{(0)})$. The implementation can either be based on a conjugate gradient algorithm or a Lanczos process. To improve the convergence of iterative methods they can be applied to the (right) preconditioned problem $\min_y \|(AS^{-1})y - b\|2$, where $Sx = y$. Here S is chosen so that AS^{-1} has a more favorable spectrum than A. Note that if $S = R$, the Cholesky factor of A^TA, then AS^{-1} is orthogonal and CGLS (CG applied to the normal equations) will converge in one iteration. Often S is taken to be an incomplete Cholesky factor of A^TA. i.e., $A^TA = \tilde{S}^T\tilde{S} - E$, where E is a defect matrix of small norm.

However, in general matrix A is not necessarily a full rank matrix, in which case, the incomplete Cholesky factorization does not exist. Moreover even when A is full rank, say full column rank, to compute the incomplete Cholesky decomposition, we need to form A^TA explicitly. Doing this costs a lot of computations and also sometimes A^TA has a very condition number because it is known that $\mathrm{cond}(A^TA) = \mathrm{cond}(A)^2$.

In order to avoid forming A^TA explicitly and to deal with the general rank deficient matrices, we focused on the Approximate Generalized Inverse Preconditioners, where the preconditioners are the approximations to the generalized inverse of A. This thesis consists of the following five parts:

- A brief introduction to the theories of generalized inverse (Chapter 2).
- Theories and methods for solving and preconditioning least squares problem (Chapter 3).

- Constructing approximate generalized inverse preconditioners by a steepest descent approach (Chapters 4).

- Another approach to construct approximate generalized inverse based on rank-one update and A^TA-orthogonalization (Chapter 5).

- Application to solving least squares problems arising in the Interior Point methods for linear programming problems.

The first two parts give preliminary knowledge. Since we consider preconditioning for rank deficient problems, we would like to have one chapter to give a simple description to the generalized inverse. Chapter 2 introduces the basic facts and theories of generalized inverse. Chapter 3 introduces the basic direct methods and iterative methods for solving least squares problems, and how and when preconditioning works for least squares problems. In the last part of Chapter 3, we introduce using GMRES to solve least squares problems. This part is greatly based on Prof. Hayami's paper [34]. We briefly list some theories so that we can use them in the following chapters.

Chapter 4 and Chapter 5 form the main body of the thesis. In Chapter 4, we used the *Minimal Residual* method to compute an approximate the generalized inverse of A. The Minimal Residual method was developed for square matrices. We applied it to rectangular matrices for the first time by choosing the steepest descent direction as the correction matrix, please also refer to [23]. In Chapter 5, we propose a method based on the computation of the generalized inverse of rank-one updated matrices, please also refer to [22, 24]. We also show that when matrix A is nonsingular, our method can be reduces to an A^TA-orthogonalization process. For these two approaches, we present numerical experiments to show their efficiency of accelerating the convergence of the Krylov subspace method.

In each step of the Interior Point Method for solving linear programming problems, the approximate solution is improved along a certain correction direction. The correction direction vector is obtained by solving a least squares problem. In Chapter 6, we present the applications of our preconditioners to the linear programming problems.

Chapter 2

Generalized Inverse

In this thesis, we considered solving least squares problems of the form

$$\min_x \|Ax - b\|_2, \quad A \in \mathbb{R}^{m\times n}, \ \ x \in \mathbb{R}^n, \ \ b \in \mathbb{R}^m. \tag{2.1}$$

When $m > n$ and A is full column rank, the problem is called over-determined problem, and when $m < n$ and A is full row rank, the problem is called under-determined problem. In this thesis, we work with general matrices A, i.e., $m \geq n$ or $m < n$ and A could be full column rank, full row rank, or rank deficient.

Since we work with a general matrix A in this thesis, before we address the main problems, in this chapter we summarize some facts about generalized inverses of a general matrix. This is not only done for introducing the reader into this subject, but also to emphasize common properties and differences between the regular inverse for nonsingular matrices and the generalized inverses for general rectangular matrices. The full theories and details for this topic can be found in the book by Campbell and Meyer [19], Ben-Israel

and Greville [5] and Wang, Wei and Qiao [50].

It is well known that every nonsingular matrix $A \in \mathbb{C}^{n\times n}$ has a unique matrix $X \in \mathbb{C}^{n\times n}$ satisfying

$$AX = \mathrm{I}, \quad XA = \mathrm{I} \tag{2.2}$$

where I is the identity matrix of order n. This X is called the inverse of A, and is denoted by $X = A^{-1}$. From equation 2.2, we can deduce some more equations,

$$AXA = A \tag{2.3}$$

$$XAX = X \tag{2.4}$$

$$(AX)^* = AX \tag{2.5}$$

$$(XA)^* = XA \tag{2.6}$$

$$AX = XA \tag{2.7}$$

$$A^{k+1}X = A^k \quad \text{for } k \geq 0. \tag{2.8}$$

Here M^* denotes the conjugate transpose of matrix M. Equations (2.3) – (2.6) are known as the Penrose conditions.

In the general case, where A maybe singular or rectangular, we can generalize the definition of the inverse of a nonsingular matrix to that of a generalized inverse. Define the generalized inverse X of A by requiring X to satisfy some equations of equations (2.3) – (2.8). Choosing a different combination of equations from (2.3) – (2.8) leads to a different definition of generalized inverse. The most popular generalized inverse of A is the Moore-Penrose inverse, which requires the generalized inverse X to satisfy equation (2.3) – (2.6). In this thesis, we mainly treat the Moore-Penrose inverse.

DEFINITION 2.0.1 *[14] Let $A \in \mathbb{C}^{m\times n}$. Then the matrix $X \in \mathbb{C}^{n\times m}$ satisfying the Penrose conditions (2.3) – (2.6) is called the Moore-Penrose inverse of A (abbreviated as the M-P inverse), and is denoted by $X = A^{\dagger}$. And X exists and is unique.*

For other definitions of generalized inverses and their applications, we refer to [50].

A nice algebraic property of the Moore-Penrose inverse is that it can be expressed explicitly by using the Singular Value Decomposition (SVD) of A.

THEOREM 2.0.1 *[29] For matrix $A \in \mathbb{C}^{m\times n}$ with rank r, let the singular value decomposition of A be*

$$A = U \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} V^T, \tag{2.9}$$

where U and V are unitary matrices, and Σ_r is a r by r diagonal matrix whose diagonal elements are all positive. Then the Moore-Penrose inverse of A is given by

$$A^{\dagger} = V \begin{bmatrix} \Sigma_r^{-1} & 0 \\ 0 & 0 \end{bmatrix} U^T. \tag{2.10}$$

The nonsingular system of linear equations

$$Ax = b \qquad (A \in \mathbb{C}^{n\times n},\ b \in \mathbb{C}^n) \tag{2.11}$$

has a unique solution

$$x = A^{-1}b. \tag{2.12}$$

In the general case,

$$Ax = b \qquad (A \in \mathbb{C}^{m\times n},\ b \in \mathbb{C}^m)\ , \tag{2.13}$$

where A may be singular or rectangular. The system of equations may not have a solution or may have many solutions. Hence, for this general case, we are interested in minimizing the residual

$$\min_{x \in \mathbb{C}^n} \|Ax - b\|_2 \, . \tag{2.14}$$

A vector $x^* \in \mathbb{C}^n$ which satisfies

$$\|Ax^* - b\|_2 = \min_{x \in \mathbb{C}^n} \|Ax - b\|_2 \tag{2.15}$$

is called a least squares solution. Among all the least squares solutions, the one with the minimum norm $\|x\|_2$ is called minimum-norm least squares solution. The following theorem shows the relation between the minimum-norm least squares solution and the Moore-Penrose inverse $A^\dagger$.

THEOREM 2.0.2 *Let $A \in \mathbb{C}^{m \times n}$ and $b \in \mathbb{C}^m$. Then $A^\dagger b$ is the minimum-norm least squares solution of (2.14).*

Consider another least squares problem

$$\min_{x \in \mathbb{C}^n} \|A^\dagger A x - A^\dagger b\|_2 \, . \tag{2.16}$$

It is obvious to see that the above least squares problem is consistent and $A^\dagger b$ is a least squares solution. For any solution y of (2.14), we have

$$y = A^\dagger b + z, \;\; z \in \mathcal{N}(A), \tag{2.17}$$

where $\mathcal{N}(A)$ stands for the null space of A. Note that

$$\|A^\dagger Ay - A^\dagger b\|_2 = \|A^\dagger A(A^\dagger b) - A^\dagger b\|_2 \,. \tag{2.18}$$

Hence, every least squares solution of (2.14) is also a least squares solution of (2.16). For any solution s of (2.16), we have

$$\begin{aligned} s &= (A^\dagger A)^\dagger A^\dagger b + t, \quad t \in \mathcal{N}(A^\dagger A) && (2.19)\\ &= A^\dagger A A^\dagger b + t && (2.20)\\ &= A^\dagger b + t. && (2.21) \end{aligned}$$

where $\mathcal{N}(A^\dagger A)$ is the null space of $A^\dagger A$. Note that $A^\dagger A$ is a projection, t has the form $t = (\mathrm{I} - A^\dagger A)u$, where $u \in \mathbb{R}^n$, which implies that $At = 0$. By

$$\begin{aligned} \|A(A^\dagger b + t) - b\|_2 &= \|AA^\dagger b - b\|_2 && (2.22)\\ &= \|(\mathrm{I} - AA^\dagger)b\|_2 && (2.23) \end{aligned}$$

we know that every least squares solution of problem (2.16) is also a least squares solution of problem (2.14). Moreover since $A^\dagger b$ is the minimum-norm least squares solution to (2.14), it is also the minimum-norm least squares solution to (2.16). Hence problem (2.14) and problem (2.16) have the same solutions, which means that these two problems are equivalent to each other, but the solution of problem (2.16) is more obvious.

However, usually we do not have $A^\dagger$, and computing $A^\dagger$ is almost as expensive as solving the problem (2.14) itself. Instead, we should consider finding a matrix $M \in \mathbb{C}^{n\times m}$

which is close to $A^{\dagger}$, i.e., an approximation to $A^{\dagger}$, and solving the problem

$$\min_{x \in \mathbb{C}^n} \|MAx - Mb\|_2, \tag{2.24}$$

in the hope of obtaining a good enough approximate solution to the original least squares problem (2.14). Problem (2.14) and problem (2.24) are not always equivalent to each other. The details will be introduced in chapter 3.

Chapter 3

Methods for Solving Least Squares Problems

3.1 Direct methods for solving least squares problems

The standard direct method for solving the least squares problem

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2 \tag{3.1}$$

is to use the QR decomposition: $A = QR$ where $Q \in \mathbb{R}^{m \times m}$ orthogonal matrix and $R \in \mathbb{R}^{m \times n}$ is an upper triangular matrix. Different methods can be used to compute the QR decomposition of A, such that Householder transformation method, Givens transformation method, and the modified Gram-Schmidt method [14]. Then, the original least squares

problem can be transformed into the following form

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \|b - Ax\|_2 &= \min_{x\in\mathbb{R}^n} \|Q^T b - Q^T Ax\|_2 \quad (3.2) \\ &= \min_{x\in\mathbb{R}^n} \|Q^T b - Rx\|_2. \quad (3.3) \end{aligned}$$

When A has full column rank, R is a nonsingular matrix, hence, to solve problem (2.14), we only need to solve a linear system

$$Rx = Q^T b \tag{3.4}$$

by back substitution. When A is not full column rank, $\mathrm{rank(A)} = \mathrm{r} < \min\{\mathrm{m}, \mathrm{n}\}$, the upper triangular matrix R has the form

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix}, \tag{3.5}$$

where R_{11} is an $r \times r$ nonsingular upper triangular matrix. Partitioning x and $Q^T b$ conformally, we obtain a linear system

$$R_{11}x_1 + R_{12}x_2 = (Q^T b)_1. \tag{3.6}$$

Since R_{11} is $r \times r$ nonsingular matrix, we can simply let $x_2 = 0$, solve

$$R_{11}x_1 = (Q^T b)_1, \tag{3.7}$$

then we obtain a least squares solution $\begin{bmatrix} x_1 \\ 0 \end{bmatrix}$. When A is large and sparse, techniques are used to save memory and computation time [14].

3.2 Iterative methods for solving least squares problems

It is well known that solving a least squares problem is equivalent to solving its corresponding normal equation.

THEOREM 3.2.1 *[14] Denote the set of all solutions to the least squares problem*

$$\min_{x\in\mathbb{R}^n} \|Ax-b\|_2, \quad A\in\mathbb{R}^{m\times n},\ m\geq n, \quad b\in\mathbb{R}^m \tag{3.8}$$

by

$$\mathcal{S}=\{x\in\mathbb{R}^n | \|Ax-b\|_2=\min\}. \tag{3.9}$$

Then $x\in\mathcal{S}$ if and only if the following orthogonality condition holds:

$$A^T(b-Ax)=0. \tag{3.10}$$

Equation (3.10) can be rewritten as $A^TAx=A^Tb$, which is called the normal equation. When $m<n$, the minimum norm solution x of the least squares problem is given by the normal equation of the form $AA^Ty=b,\ x=A^Ty$.

In principle, any iterative method for symmetric positive definite or symmetric positive semi-definite linear systems can be applied to the normal equations. Among the iterative methods to solve least squares problems, the Conjugate Gradient Least Squares (CGLS) [14, 35] method is most commonly used.

ALGORITHM 3.2.1 *CGLS(CGNR)*

Let $x^{(0)}$ be an initial approximation, set

$$r^{(0)} = b - Ax^{(0)}, \;\; p^{(0)} = s^{(0)} = A^T r^{(0)}, \;\; \gamma_0 = \|s^{(0)}\|_2^2, \tag{3.11}$$

for $k = 0, 1, \ldots$ while $\gamma_k > tol$ compute

1. $q^{(k)} = Ap^{(k)}$,
2. $\alpha_k = \frac{\gamma_k}{\|q^{(k)}\|_2^2}$,
3. $x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$,
4. $r^{(k+1)} = r^{(k)} - \alpha_k q^{(k)}$,
5. $s^{(k+1)} = A^T r^{(k+1)}$,
6. $\gamma_{k+1} = \|s^{(k+1)}\|_2^2$,
7. $\beta_k = \frac{\gamma_{k+1}}{\gamma_k}$,
8. $p^{(k+1)} = s^{(k+1)} + \beta_k p^{(k)}$.

Let $\mathrm{rank}(A) = r$, and σ_1 and σ_r be the largest and smallest singular value of A, respectively. Then the condition number with respect to 2-norm of A is defined as

$$\kappa_2(A) = \|A\|_2 \|A^\dagger\|_2 = \frac{\sigma_1}{\sigma_r}, \tag{3.12}$$

which implies that the condition number of $A^T A$ is

$$\kappa_2(A^T A) = (\frac{\sigma_1}{\sigma_r})^2 = \kappa_2(A). \tag{3.13}$$

CGLS is equivalent to applying CG to the normal equation in corresponding to the least squares problem. It is a well known fact that the convergence of CGLS method depends on the spectrum of $A^T A$. When $\kappa_2(A)$ is large, $\kappa_2(A^T A)$ is even larger, CGLS converges

slowly. Björck, Elfving, and Strakos analyzed the lack of stability of CGLS [15].

Paige and Saunders [43] developed the LSQR algorithm based on the lanczos bidiagonalization algorithm. LSQR is mathematically equivalent to CGLS but converges somewhat more quickly when A is ill-conditioned. However, the achievable accuracy with CGLS and LSQR seem to be the same.

When CGLS is used to solve the problem, and A is ill-conditioned, preconditioning becomes necessary. The normal equation can be preconditioned symmetrically as

$$P^{-T}A^TAP^{-1}y = P^{-T}A^Tb, \;\; P^{-1}y = x. \tag{3.14}$$

Performing CG on equation (3.14), we obtain the following preconditioned CGLS method (PCGLS).

ALGORITHM 3.2.2 *PCGLS*

Let $x^{(0)}$ be an initial approximation, set

$$r^{(0)} = b - Ax^{(0)}, \;\; p^{(0)} = s^{(0)} = P^{-T}(A^Tr^{(0)}), \;\; \gamma_0 = \|s^{(0)}\|_2^2, \tag{3.15}$$

for $k = 0, 1, \ldots$ while $\gamma_k > tol$ compute

1. $t^{(k)} = P^{-1}p^{(k)}$,
2. $q^{(k)} = At^{(k)}$,
3. $\alpha_k = \frac{\gamma_k}{\|q^{(k)}\|_2^2}$,
4. $x^{(k+1)} = x^{(k)} + \alpha_k t^{(k)}$,
5. $r^{(k+1)} = r^{(k)} - \alpha_k q^{(k)}$,
6. $s^{(k+1)} = P^{-T}(A^Tr^{(k+1)})$,
7. $\gamma_{k+1} = \|s^{(k+1)}\|_2^2$,

8. $\beta_k = \frac{\gamma_{k+1}}{\gamma_k}$,
9. $p^{(k+1)} = s^{(k+1)} + \beta_k p^{(k)}$.

People came up with a lot of different ways to construct this preconditioner P for CGLS. An early idea for preconditioning the conjugate gradient method can be found in [28]. A major breakthough took place around the mid-1970s, with the introduction by Meijerink and van der Vorst of the incomplete Cholesky-Conjugate Gradient (ICCG) algorithm [38]. Incomplete factorization methods were introduced for the first time by Buleev in the then-Soviet Union in the late 1950s, and independently by Varga (see [17,37,40,48]; see also [41]). However, Meijerink and van der Vorst deserve credit for recognizing the potential of incomplete factorizations as preconditioners for the conjugate gradient method. Since then, a number of improvements and extensions have been made, including level-of-fills and drop tolerance-based incomplete factorizations, generalizations to block matrices, modified and stabilized variants.

Every incomplete factorization method is based on a certain factorization of A or A^TA when $m \geq n$ (AA^T when $m \leq n$) and some techniques to control the fill-ins. For example the incomplete Cholesky decomposition based on A^TA [38], incomplete QR decomposition based on A by using incomplete modified Gram-Schmidt method [36,45,51], using incomplete Givens orthogonalization [2,44], and Robust incomplete factorization (RIF) [13], which computes the incomplete Cholesky decomposition of A^TA without forming A^TA explicitly.

3.3 GMRES methods for least squares problems

CGLS and LSQR are efficient Krylov subspace methods [27] which construct the orthogonal Krylov subspace basis by a short term recurrence form. However, because of the short term recurrence, when A is ill-conditioned, CGLS and LSQR easily lose orthogonality due to rounding errors easily. Even with preconditioning, the convergence behavior may deteriorate for highly ill-conditioned problems due to rounding errors. The Generalized Minimal Residual (GMRES) method [47] is an efficient and robust Krylov subspace iterative method for solving systems of linear equations

$$Ax = b, \quad A \in \mathbb{R}^{n\times n}, \ x \in \mathbb{R}^n, \ b \in \mathbb{R}^n, \tag{3.16}$$

where A is nonsingular and nonsymmetric. Let the initial solution to the linear system (3.16) be x_0, and denote the initial residual $r_0 = b - Ax_0$. In the kth iteration, GMRES looks for the "best" approximate x_k solution in the affine subspace

$$x_0 + \mathcal{K}_k(A, r_0), \quad \mathcal{K}_k = \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}. \tag{3.17}$$

The "best" approximation is in the sense that we require x_k to minimize the residual norm,

$$\|b - Ax_k\|_2 = \min_{x\in x_0+\mathcal{K}_k(A,r_0)} \|b - Ax\|_2. \tag{3.18}$$

To find this "best" approximate solution x_k in $x_0 + \mathcal{K}_k(A, r_0)$, GMRES performs a modified Gram-Schmidt process to construct a sequence of orthogonal basis vectors $v_1, \ldots, v_k$, where v_1 is defined as $\frac{r_0}{\|r_0\|_2}$. Hence for any $x \in x_0 + \mathcal{K}_k(A, r_0)$, x can be written in the

form

$$x = x_0 + [v_1, \ldots, v_k]\, y, \quad y \in \mathbb{R}^k. \tag{3.19}$$

Note

$$\begin{aligned} \|r\|_2 &= \|b - Ax\|_2 && (3.20)\\ &= \|b - A(x_0 + [v_1, \ldots, v_k]\, y)\|_2 && (3.21)\\ &= \|b - Ax_0 - A\,[v_1, \ldots, v_k]\, y\|_2 && (3.22)\\ &= \|r_0 - A\,[v_1, \ldots, v_k]\, y\|_2. && (3.23) \end{aligned}$$

From the Modified Gram-Schmidt process, we obtain the Arnoldi decomposition

$$A\,[v_1, \ldots, v_k] = [v_1, \ldots, v_k]\, H_{k+1,k}, \tag{3.24}$$

where $H_{k+1,k}$ is a product of the Modified Gram-Schmidt process, and is a $(k+1) \times k$ upper Hessenberg matrix

$$H_{k+1,k} = \begin{bmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,k} \\ h_{2,1} & \ddots & \ddots & \vdots \\ & \ddots & \ddots & h_{k-1,k} \\ & & \ddots & h_{k,k} \\ 0 & \cdots & \cdots & h_{k+1,k} \end{bmatrix} = \begin{bmatrix} \bar{H}_{k,k} \\ [0, \cdots, 0, h_{k+1,k}] \end{bmatrix}. \tag{3.25}$$

Denote $V = [v_1, \ldots, v_k]$, so that V is a $n \times k$ orthogonal matrix. So that the Equation (3.23) can be rewritten as

$$\begin{aligned} \|r\|_2 &= \|r_0 - A\,[v_1, \ldots, v_k]\,y\|_2 && (3.26)\\ &= \|r_0 - V H_{k+1,k} y\|_2 && (3.27)\\ &= \|V^T r_0 - H_{k+1,k} y\|_2. && (3.28) \end{aligned}$$

Remember that the first column of V is $v_1 = \dfrac{r_0}{\|r_0\|_2}$. Hence, it is orthogonal to $v_2, \ldots, v_k$. Denote $[1, 0, \ldots, 0]^T$ of order $k+1$ as e_1, equation (3.28) can be transformed into the following form,

$$\|r\|_2 = \left\|\|r_0\|_2 e_1 - H_{k+1,k} y\right\|_2. \quad (3.29)$$

Using the Givens rotations to zero out the elements in the lower sub-diagonal of $H_{k+1,k}$, we obtain a matrix whose first k rows is a k by k upper triangular matrix R_k, and the last row is 0. Denote the product of all the Givens rotations as G. Then we have

$$\begin{aligned} \|r\|_2 &= \left\| \|r_0\|_2 G e_1 - \begin{bmatrix} R_k \\ 0 \end{bmatrix} y \right\|_2 && (3.30)\\ &= \left\| \begin{bmatrix} g - R_k y \\ 0 \end{bmatrix} \right\|_2, && (3.31) \end{aligned}$$

where g is the vector whose elements are the first k elements of $\|r_0\|_2 G e_1$. When A is a nonsingular matrix, R is nonsingular. Solving equation

$$R_k y = g, \quad (3.32)$$

we obtain y, and we can compute the k-approximation solution x_k.

However, the Krylov Subspace

$$\mathcal{K}_k(A, r_0) = \mathrm{span}\{\mathrm{r}_0, \mathrm{Ar}_0, \ldots, \mathrm{A}^{\mathrm{k}-1}\mathrm{r}_0\} \tag{3.33}$$

is only well defined when A is a square matrix. Hence GMRES cannot be applied to the least squares problems directly. To use GMRES to solve least squares problems, we need to find a preconditioner matrix $P \in \mathbb{R}^{n\times m}$, so that PA or AP is a square matrix, and then to solve the problem

$$\min_x \|PAx - Pb\|_2 \tag{3.34}$$

or

$$\min_y \|APy - b\|_2, \tag{3.35}$$

in the hope that we can obtain the solution to the original least squares problem (2.14). This idea was introduced in [49], and fully discussed in [34]. Since the original problem (2.14) may not be equivalent to the preconditioned problem (3.34) or (3.35), conditions must be imposed on P so that we can ensure that the solution we compute from (3.34) or (3.35) is also the solution to the original problem. In the following we will summarize some theoretical results. For the details we refer to [34].

First we consider preconditioning the least squares problem (2.14) from the right. In this case, we want to solve the problem (3.35).

THEOREM 3.3.1 *[34]* $\min_x \|b - Ax\|_2 = \min_z \|b - APx\|_2$ *holds for all* $b \in \mathbb{R}^n$ *if and only if* $\mathcal{R}(A) = \mathcal{R}(AP)$.

Hence, if the condition in the above theorem is satisfied, we can apply the GMRES method on AP, which is a square matrix. When AP is rank deficient, GMRES may break down before it finds a solution. Hence, we also need the following theorem to ensure that after preconditioning, the GMRES can find a solution to the problem (3.35) before break down happens. We first state a result by Brown and Walker [16].

THEOREM 3.3.2 *[16] Let $A \in \mathbb{R}^{n\times n}$, the GMRES method can give a solution to* $\min_{x\in\mathbb{R}^n} \|b - Ax\|_2$ *without break down for arbitrary $b \in \mathbb{R}^n$ and $x_0 \in \mathbb{R}^n$ if and only if $\mathcal{R}(A) = \mathcal{R}(A^T)$.*

Based on this Brown and Walker 's theorem, to make sure that GMRES does not break down before finding a solution, we need $\mathcal{R}(AP) = \mathcal{R}(P^T A^T)$. According to [34], we have the following theorem.

THEOREM 3.3.3 *[34]*

- *If $\mathcal{R}(A^T) = \mathcal{R}(P)$, then $\mathcal{R}(A) = \mathcal{R}(AP)$.*
- *If $\mathcal{R}(A^T) = \mathcal{R}(P)$, then $\mathcal{R}(AP) = \mathcal{R}(P^T A^T) \Longleftrightarrow \mathcal{R}(A) = \mathcal{R}(P^T)$.*

To sum up,

THEOREM 3.3.4 *[34] If $\mathcal{R}(A^T) = \mathcal{R}(P)$ holds, then the GMRES method determines a least squares solution of $\min_{x\in\mathbb{R}^n} \|b - Ax\|_2$ for all $b \in \mathbb{R}^m$ and $x_0 \in \mathbb{R}^n$ without breakdown if and only if $\mathcal{R}(A) = \mathcal{R}(P^T)$.*

For preconditioning problem (2.14) from the left, we have analogical results. We summarize the results from [34] in the following.

THEOREM 3.3.5

$$\|b - Ax^*\|_2 = \min_{x\in\mathbb{R}^n} \|b - Ax\|_2 \tag{3.36}$$

and

$$\|Pb - PAx^*\|_2 = \min_{x\in\mathbb{R}^n} \|Pb - PAx\|_2 \tag{3.37}$$

are equivalent for all $b \in \mathbb{R}^m$, if and only if $\mathcal{R}(A) = \mathcal{R}(P^T PA)$.

THEOREM 3.3.6 *[34]*

- *If $\mathcal{R}(A) = \mathcal{R}(P^T)$, then $\mathcal{R}(A) = \mathcal{R}(P^T PA)$.*
- *For all $b \in \mathbb{R}^m$, $PAx = Pb$ has a solution, which attains $\min_{x\in\mathbb{R}^n} \|b - Ax\|_2$ if and only if $\mathcal{R}(A) = \mathcal{R}(P)$.*
- *If $\mathcal{R}(A) = \mathcal{R}(P^T)$, then $\mathcal{R}(PA) = \mathcal{R}(A^T P^T) \Longleftrightarrow \mathcal{R}(A^T) = \mathcal{R}(P)$.*

To sum up, the following theorem gives the conditions to guarantee that the original least squares problem is equivalent to the right-preconditioned problem, and that GMRES method can determine a solution to the original problem before breakdown happens.

THEOREM 3.3.7 *[34] If $\mathcal{R}(A) = \mathcal{R}(P^T)$ holds, the GMRES method determines a least squares solution of $\min_{x\in\mathbb{R}^n} \|b - Ax\|_2$ for all $b \in \mathbb{R}^m$ and all $x_0 \in \mathbb{R}^n$ without breakdown if and only if $\mathcal{R}(A^T) = \mathcal{R}(P)$.*

Hence, no matter wether we precondition the original least squares problem (2.14) from the left or from the right,

$$\mathcal{R}(A) = \mathcal{R}(P^T), \qquad \mathcal{R}(A^T) = \mathcal{R}(P) \tag{3.38}$$

is a sufficient condition that the preconditioned least squares problem is equivalent to the original least squares problem and GMRES can solve the preconditioned problem and determine a least squares solution without breakdown. The preconditioner P for GMRES should be carefully chosen, because of the two image space conditions (3.38). Unlike the

preconditioners for CGLS, which is essentially preconditioning the normal equations of the least squares problem from the both sides, many preconditioners designed for CGLS cannot be applied to the GMRES method. The most straightforward choice for the GMRES method is to let $P = \alpha A^T$, where α is a nonzero scalar. The conditions $\mathcal{R}(A) = \mathcal{R}(P^T)$ and $\mathcal{R}(A^T) = \mathcal{R}(P)$ are obviously satisfied. When $\alpha = 1$, using this preconditioner is equivalent to solving the normal equations of the original least squares problem. For a full rank matrix A (full column rank or full row rank), a generalization to $P = \alpha A^T$ is to let $P = CA^T$ for full column rank matrix A and $P = AC^T$ for full row rank matrix A, where C is a nonsingular matrix with appropriate size [34].

Chapter 4

Approximate Generalized Inverse Preconditioning Methods

4.1 The approximate inverse preconditioning methods

In Chapter 3, we briefly introduced the incomplete factorization preconditioning methods. Notwithstanding their popularity, incomplete factorization preconditioning methods have their own limitations: potential instabilities, difficulty of parallelization. These two limitations motivated the development of other preconditioning methods. During the past two decades, a class of algebraic preconditioning techniques called *sparse approximate inverses (SAINV)* received considerable interest.

The incomplete factorization preconditioning methods, whether it is the incomplete Cholesky decomposition or the incomplete QR decomposition, all focus on constructing an approximation to the coefficient matrix A itself, or an incomplete factorization of A

itself. In contrast, the SAINV approach focuses on constructing an approximation to the inverse of A, or an incomplete factorization of the inverse of A.

Sparse Approximate Inverse preconditioners were first introduced in the early 1970s by Benson [6, 7]. The methods were originally developed for solving large sparse linear systems of the form,

$$Ax = b, \tag{4.1}$$

where $A \in \mathbb{R}^{n\times n}$ is a nonsingular matrix and $b \in \mathbb{R}^n$ is a right-hand-side vector. As is well known, the rate of convergence of iterative methods for solving (4.1) is strongly influenced by the spectral properties of A. It is therefore natural to transform the original system into an equivalent system with more favorable spectral properties by using its approximate inverse. There are different approaches to construct the approximate inverse of A [8, 46]. One popular way to accomplish this construction is to find a matrix M which minimizes the following Frobenius norm

$$\min_{M\in\mathcal{S}} \|\mathrm{I} - MA\|_F \quad \text{or} \quad \min_{M\in\mathcal{S}} \|\mathrm{I} - AM\|_F \tag{4.2}$$

over all $n \times n$ matrices with a certain sparsity pattern $\mathcal{S}$, where I is the $n \times n$ identity matrix. Hence, MA and AM are approximations to an identity matrix, which implies that M is an approximation to the inverse of A. This idea of constructing M by minimizing the Frobenius norm $\|\mathrm{I} - AM\|_F$ was first proposed by Benson in his master thesis [6]. See also Benson and Frederickson [7].

However, it is very difficult to choose a suitable sparsity pattern for M. Hence, several authors developed adaptive methods which start from a simple initial nonzero pattern and

gradually refine it until $\|I - MA\|_F < \epsilon$ is achieved, where ϵ is a threshold [11, 21, 30, 33]. The most successful of these methods is the one proposed by Grote and Huckle [33], which is called SPAI. Unfortunately, the setup time for adaptive SPAI is often high [3,4,11]. Thus, Chow and Saad developed the *Minimal Residual (MR)* [20] method so that no nonzero pattern needs to be prescribed in advance. For the left preconditioning case, the algorithm can be written in the following form.

ALGORITHM 4.1.1 MR Algorithm [46]

1. Set $M_0 = \alpha_0 A^T$, $\alpha_0 = \frac{\|A\|_F^2}{\|A^T A\|_F^2}$
2. For $k = 1, 2, \ldots,$ until convergence, Do
3. Compute $R_{k-1} = I - M_{k-1}A$, and G_{k-1}
4. Compute $\alpha_k = \text{trace}(R_{k-1}^T G_{k-1} A)/\|G_{k-1}A\|_F^2$
5. Compute $M_k := M_{k-1} + \alpha_k G_{k-1}$
6. Apply numerical droppings to M_k
7. End Do

In the above algorithm, α_0 is chosen to minimize $\|I - \alpha A^T A\|_F$, and $\alpha_k,\ k \geq 1$ are chosen to minimize

$$\|I - (M_{k-1} + \alpha G_{k-1})A\|_F. \tag{4.3}$$

One choice for G_k is the residual matrix $R_k = I - M_k A$, and another popular choice is $G_k = (I - M_k A)A^T$, which is the direction of steepest descent.

There is another way to minimize (4.2). Instead of minimizing globally as a function

of matrix M, it can be minimized column by column (or row by row), as follows

$$\begin{array}{cc} \min \|\mathrm{I} - MA\|_F & \min \|\mathrm{I} - AM\|_F \\ \Updownarrow & \Updownarrow \\ \min \|e_i - m_i A\|_2,\ i = 1, \ldots, n & \min \|e^i - Am^i\|_2,\ i = 1, \ldots, m, \end{array}$$

where e_i and m_i are rows of the identity matrix I and M, e^i and m^i are columns of the identity matrix I and M, respectively. The advantage of performing the minimization column by column or row by row is that it can be easily parallelized. For the left preconditioning, the row-oriented algorithm [46] is as followings.

ALGORITHM 4.1.2

1. Set $M_0 = \alpha_0 A^T$, $\alpha_0 = \frac{\|A\|_F^2}{\|A^T A\|_F^2}$
2. For $j = 1, \ldots, n$ Do
3. Define $m_j = e_j M$
4. For $k = 1, \ldots, n_k$ Do
5. $r_j = e_j - m_j A$
6. $\alpha_j = \frac{<r_j A, r_j>}{\|r_j A\|_2^2}$
7. $m_j = m_j + \alpha_j r_j$
8. Apply numerical dropping to m_j
9. End Do
10. End Do

4.2 The approximate generalized inverse preconditioning with minimizing Frobenius norm approach

In this thesis, we consider applying Saad's MR algorithm to the least squares problems

$$\min_{x\in\mathbb{R}^n} \|b - Ax\|_2 \tag{4.4}$$

where $A \in \mathcal{R}^{m\times n}$, $\mathrm{rank(A)} = \mathrm{r} \leq \min\{\mathrm{m}, \mathrm{n}\}$, $b \in \mathcal{R}^m$, which to the authors' knowledge, is new. Thus, we aim to construct a preconditioner $M \in \mathbb{R}^{n\times m}$ which minimizes

$$\|I - MA\|_F \quad \text{or} \quad \|I - AM\|_F \,, \tag{4.5}$$

where I is an $n \times n$ or $m \times m$ identity matrix, respectively. Since now matrices A and M_k are rectangular, we cannot choose the correction matrix G_k as $R_k = I - M_k A$. Hence, we let $G_k = (I - M_k A)A^T$. We will also give mathematical justifications for applying the method to least-squares problems.

4.2.1 Left Preconditioning

Consider solving the least squares problem (4.4) by transforming it into a left preconditioned form,

$$\min_{x\in\mathbb{R}^n} \|Mb - MAx\|_2, \tag{4.6}$$

where $A \in \mathbb{R}^{m\times n}$, $M \in \mathbb{R}^{n\times m}$, and b is a right-hand-side vector $b \in \mathbb{R}^m$.

For preconditioning, one important issue is whether the solution of the preconditioned

problem is the solution of the original problem. For square nonsingular linear systems, the condition for this equivalence is that the preconditioner M should be nonsingular. Since we are dealing with rectangular problems, we need some other conditions to ensure that the preconditioned problem (4.6) is equivalent to the original least squares problem (4.4). By Theorem 3.3.5 we know that in order that the preconditioned problem (4.6) is equivalent to the original problem (4.4), the matrix M of Algorithm 4.1.1 should satisfy the condition $\mathcal{R}(A) = \mathcal{R}(M^T MA)$.

In order to analyze this condition, we rewrite Algorithm 4.1.1 for left preconditioning on the rectangular matrix A as follows.

ALGORITHM 4.2.1

1. Set $M_0 = \alpha_0 A^T$, $\alpha_0 = \frac{\|A\|_F^2}{\|A^T A\|_F^2}$
2. For $k = 1, 2, \ldots,$ until maximum step is reached, Do
3. Compute $R_{k-1} = I - M_{k-1}A$
4. Compute $G_{k-1} = R_{k-1}A^T$
5. Compute $\alpha_k = \|G_{k-1}\|_F^2 / \|G_{k-1}A\|_F^2$
6. Compute $M_k := M_{k-1} + \alpha_k G_{k-1}$
7. Apply numerical dropping to M_k
8. End Do

In the above Algorithm 4.2.1, $M_0 = \alpha_0 A^T$, where α_0 minimizes $\|I - \alpha A^T A\|_F$ over all real scalar α. Hence, we have

$$\begin{aligned} M_1 &= M_0 + \alpha_1 G_0 && (4.7)\\ &= M_0 + \alpha_1 (I - M_0 A) A^T && (4.8)\\ &= (\alpha_0 + \alpha_1 - \alpha_0 \alpha_1 A^T A) A^T && (4.9)\\ &= p_1(A^T A) A^T, && (4.10) \end{aligned}$$

where $p_k(\cdot)$ is a polynomial of degree k. Similarly, if we assume $M_{k-1} = p_{k-1}(A^TA)A^T$, then for M_k, we have

$$\begin{aligned} M_k &= M_{k-1} + \alpha_k G_{k-1} && (4.11)\\ &= M_{k-1} + \alpha_k (I - M_{k-1}A)A^T && (4.12)\\ &= M_{k-1}(I - \alpha_k AA^T) + \alpha_k A^T && (4.13)\\ &= (p_{k-1}(A^TA)(I - \alpha_k A^TA) + \alpha_k)A^T && (4.14)\\ &\equiv p_k(A^TA)A^T. && (4.15) \end{aligned}$$

Combining all the above argument, we have the following.

THEOREM 4.2.1 *If no numerical droppings are performed, M_k in Algorithm 4.2.1 can be expressed as $M_k = p_k(A^TA)A^T$, where $p_k(\cdot)$ is a polynomial of degree k, p_0 is the scalar α_0 defined in Algorithm 4.2.1.*

By expressing M_k in the form $M_k = p_k(A^TA)A^T$, we can easily deduce the condition for the equivalence between the preconditioned problem (4.6) and the original problem (4.4) as follows.

THEOREM 4.2.2 *If no numerical droppings are performed, the preconditioned problem (4.6) is equivalent to the original problem (4.4) if and only if $p_k(\sigma_i^2) \neq 0$ for all singular values $\sigma_i > 0$ of A.*

PROOF. By Theorem 3.3.5, we only need to prove $\mathcal{R}(A) = \mathcal{R}(M_k^T M_k A)$. Since

$$\begin{aligned} \mathcal{R}(M_k^T M_k A) &= \mathcal{R}(A p_k(A^TA) p_k(A^TA) A^T)\\ &\subseteq \mathcal{R}(A), \end{aligned}$$

$\mathcal{R}(A) = \mathcal{R}(M_k^T M_k A)$ is equivalent to

$$\text{rank}(A) = \text{rank}(M_k^T M_k A). \tag{4.16}$$

Assume that the SVD of A is $A = U\Sigma V^T$, where U is an $m \times m$ orthogonal matrix, V is an $n \times n$ orthogonal matrix, and $\Sigma = \text{diag}\{\sigma_1, \ldots, \sigma_r, 0, \ldots, 0\}_{m\times n}$, $\sigma_i > 0, i = 1, \ldots, r$. Then,

$$\begin{aligned} M_k^T M_k A &= A p_k(A^T A) p_k(A^T A) A^T A \\ &= U\Sigma p_k^2(\Sigma^T\Sigma)\Sigma^T\Sigma V^T \\ &= U\text{diag}\{\sigma_1^3 p_k^2(\sigma_1^2), \ldots, \sigma_r^3 p_k^2(\sigma_r^2), 0, \ldots, 0\}_{m\times n} V^T. \end{aligned}$$

Hence, (4.16) is equivalent to $p_k(\sigma_i^2) \neq 0$ for $\sigma_i > 0,\ i = 1, \ldots, r$. □

It is difficult to prove that $p_k(\sigma_i^2) \neq 0$ for $\sigma > 0, i = 1, \ldots, r$. However, we can assume that it holds generically, i.e., the probability of $p_k(\sigma_i^2) = 0$ for any $1 \leq i \leq r$ is zero. Also in our numerical experiments, we never observed $p_k(\sigma_i^2) = 0$ to happen.

Besides the above equivalence theorem, we are also concerned whether any breakdown may occur when we solve the preconditioned problem (4.6) using Krylov subspace methods. Using Brown and Walker's theorem and Theorem 3.3.2, we have the following.

THEOREM 4.2.3 *For M_k in Algorithm 4.2.1, $M_k A$ is symmetric, so that the GMRES method determines a least-squares solution of the preconditioned problem $\min\limits_{x\in\mathcal{R}^n} \|M_k b - M_k A x\|_2$ without breakdown for arbitrary $b \in R^m$ and initial guess $x_0 \in R^n$.*

PROOF. The proof follows directly from Theorem 4.2.1 and Lemma 3.3.2. Since

$M_k = p_k(A^TA)A^T$, M_kA is symmetric, which implies $\mathcal{N}(M_kA) = \mathcal{N}((M_kA)^T)$. □

From Theorem 4.2.2 and Theorem 4.2.6, the GMRES method can be used to solve the preconditioned least squares problem (4.6) with the preconditioner M_k from Algorithm 4.2.1, to obtain a least squares solution to the original least squares problem (4.4) without breakdown. Moreover, since M_kA is symmetric, the MINRES method [42], which is equivalent to the GMRES method for symmetric matrices and uses short recurrences, can be used instead to save computation time and memory.

REMARK 1 *In Theorem 4.2.1 we assume that there is no numerical droppings performed so that we have Theorem 4.2.2 and Theorem 4.2.6. When the numerical dropping strategy is used, M_k cannot be written in the polynomial form $p_k(A^TA)A^T$ as we show in the following.*

In Algorithm 4.2.1, $M_0 = \alpha_0 A^T$, where α_0 minimizes $\|I - \alpha A^TA\|_F$ over all real scalars α. When A is sparse, we do not need to do numerical droppings for M_0. Denote the dropped part in the process of computing M_i as E_i, Hence, we have

$$\begin{aligned}
M_1 &= M_0 + \alpha_1(I - M_0A)A^T - E_1 \\
&= p_1(A^TA)A^T - E_1 \\
M_2 &= M_1 + \alpha_2(I - M_1A)A^T - E_2 \\
&= p_1(A^TA)A^T - E_1 + \alpha_2(I - p_1(A^TA)A^TA - E_1A)A^T - E_2 \\
&= p_1(A^TA)A^T + \alpha_2(I - p_1(A^TA)A^TA)A^T - E_1 - \alpha_2E_1AA^T - E_2 \\
&= p_2(A^TA)A^T - E_1 - E_2 - \alpha_2E_1AA^T \\
&\vdots
\end{aligned}$$

where $p_k(\cdot)$ is a polynomial of degree k.

According to the above discussion, we cannot ensure the equivalence and the breakdown free theorems when numerical droppings are used. However, in our numerical experiments, when the dropping threshold is not too large, we did not encounter breakdown of GMRES.

The row-oriented Algorithm 4.1.2 can also be modified to be applied to rectangular matrices.

ALGORITHM 4.2.2

1. Set $M_0 = \alpha_0 A^T$, $\alpha_0 = \frac{\|A\|_F^2}{\|A^T A\|_F^2}$
2. For $j = 1, \ldots, n$ Do
3. Define $m_j = e_j M$
4. For $i = 1, \ldots, n_i$ Do
5. $r_j = e_j - m_j A$
6. $g_j = r_j A^T$
7. $\alpha_j = \frac{\|g_j\|_2^2}{\|r_j A^T A\|_2^2}$
8. $m_j = m_j + \alpha_j g_j$
9. Apply numerical dropping to m_j
10. End Do
11. End Do

However, it is difficult to show equivalence theorems for this row-oriented method.

4.2.2 Right Preconditioning

So far we have discussed left preconditioning. For over-determined problems, i.e. $A \in \mathbb{R}^{m \times n}$, $m > n$, left preconditioning is more favorable, since the size of the preconditioned

matrix $M_k A$ is $n \times n$. However, if we are considering an under-determined problem, i.e. $A \in \mathbb{R}^{m \times n}$, $m < n$, right preconditioning is more suitable. Results analogous to the left preconditioning hold for the right preconditioning case.

When we precondition the original least squares problem (4.4) from the right-hand-side, we have,

$$\min_{y \in \mathbb{R}^m} \|b - AMy\|_2. \tag{4.17}$$

We rewrite Algorithm 4.2.1 for right preconditioning as follows.

ALGORITHM 4.2.3

1. Set $M_0 = \alpha_0 A^T$, $\alpha_0 = \frac{\|A\|_F^2}{\|AA^T\|_F^2}$
2. For $k = 1, 2, \ldots,$ until convergence, Do
3. Compute $R_{k-1} = I - AM_{k-1}$
4. Compute $G_{k-1} = A^T R_{k-1}$
5. Compute $\alpha_k = \|G_{k-1}\|_F^2 / \|AG_{k-1}\|_F^2$
6. Compute $M_k := M_{k-1} + \alpha_k G_{k-1}$
7. Apply numerical dropping to M_k
8. End Do

Similar to the left preconditioning case, M_k from Algorithm 4.2.3 also has a polynomial form.

THEOREM 4.2.4 *M_k in Algorithm 4.2.3 can be expressed as $M_k = A^T p_k(AA^T)$, where $p_k(\cdot)$ is a polynomial of degree k, p_0 is a scalar α_0 defined in Algorithm 4.2.3.*

PROOF. According to Algorithm 4.2.3,

$$\begin{aligned} M_0 &= \alpha_0 A^T \\ &\equiv A^T p_0(AA^T) \\ M_1 &= M_0 + \alpha_1 G_0 \\ &= \alpha_0 A^T + \alpha_1 A^T (I - \alpha_0 AA^T) \\ &= A^T(\alpha_0 + \alpha_1 - \alpha_0\alpha_1 AA^T) \\ &\equiv A^T p_1(A^T A). \end{aligned}$$

Thus assume M_{k-1} can be expressed as $M_{k-1} = A^T p_{k-1}(AA^T)$. Then,

$$\begin{aligned} M_k &= M_{k-1} + \alpha_k G_{k-1} \\ &= A^T p_{k-1}(AA^T) + \alpha_k A^T (I - AA^T p_{k-1}(AA^T)) \\ &= A^T((I - \alpha_k AA^T) p_{k-1}(AA^T) + \alpha_k) \\ &\equiv A^T p_k(AA^T). \quad \square \end{aligned}$$

Combining this Theorem 4.2.4 and Theorem 3.3.1, we get the following equivalence theorem for right preconditioning.

THEOREM 4.2.5 *The preconditioned problem (4.17) is equivalent to the original problem (4.4) if and only if* $p_k(\sigma_i^2) \neq 0$ *for all singular values* $\sigma_i > 0$ *of* A.

PROOF. By Lemma 3.3.1, we only need to prove $\mathcal{R}(A) = \mathcal{R}(AM_k)$. Since

$$\begin{aligned} \mathcal{R}(AM_k) &= \mathcal{R}(AA^T p_k(AA^T)) \\ &\subseteq \mathcal{R}(A), \end{aligned}$$

$\mathcal{R}(A) = \mathcal{R}(AM_k)$ is equivalent to

$$\text{rank}(A) = \text{rank}(AM_k). \tag{4.18}$$

Let the SVD of A be $A = U\Sigma V^T$, where U is an $m \times m$ orthogonal matrix, V is an $n \times n$ orthogonal matrix, and $\Sigma = \text{diag}\{\sigma_1, \ldots, \sigma_r, 0, \ldots, 0\}_{m\times n}$, $\sigma_i > 0,\ i = 1, \ldots, r$. Then,

$$\begin{aligned} AM_k &= AA^T p_k(AA^T) \\ &= U\Sigma\Sigma^T p_k(\Sigma\Sigma^T)U^T \\ &= U\text{diag}\{\sigma_1^2 p_k(\sigma_1^2), \ldots, \sigma_r^2 p_k(\sigma_r^2), 0, \ldots, 0\}_{m\times m}U^T. \end{aligned}$$

Now, it is easy to see that the equation (4.18) holds if and only if $p_k(\sigma_i^2) \neq 0$ for $\sigma_i > 0, i = 1, \ldots, r$. □

Again, we may expect that $p_k(\sigma_i^2) \neq 0$ for $\sigma_i > 0, i = 1, \ldots, r$ holds generically.

Combining Theorem 3.3.2 and Theorem 4.2.4, we also obtain a breakdown free theorem for the right preconditioning case.

THEOREM 4.2.6 *For M_k in Algorithm 4.2.3, AM_k is symmetric, so that the GMRES method determines a least squares solution of the preconditioned problem* $\min_{x\in\mathcal{R}^m} \|b - AM_k x\|_2$

without breakdown for arbitrary $b \in R^m$ *and initial guess* $x_0 \in R^n$.

PROOF. The proof follows directly from Theorem 4.2.4 and Lemma 3.3.2. Since $M_k = A^T p_k(AA^T)$, AM_k is symmetric, which implies $\mathcal{N}(AM_k) = \mathcal{N}((AM_k)^T)$. □

4.3 Numerical Examples

In this section, we show some examples to test our algorithm. We compare our preconditioning method with well known methods, i.e. CGLS and the diagonally scaled CGLS method [14]. Table 4.1 provides some basic information about the test matrices. In the table, m is the number of the rows, n is the number of columns, nnz is the total number of nonzeros, $density$ is the density of the nonzeros in the matrices, $cond$ is the condition number of the matrices. The first matrix illc1850 was taken from the Matrix Market [26], and the matrix sprandn8L and sprand8S are random matrices generated by the MATLAB command: sprandn.

Table 4.1: Test Matrices

	Origin	m	n	nnz	$density$	$cond$
illc1850	Matrix Market	1850	712	8636	0.007	10^3
sprandn8L	random Matrix	10000	1000	487816	0.0488	10^8
sprandn8S	random Matrix	2000	500	48788	0.0488	10^8

All computations were run on an IBM T60 laptop computer, where the CPU is 2.0 GHz and the memory is 1 GB, and the programming language MATLAB in Windowxp.

For the first matrix, we use a random right hand side vector, which is generated by MATLAB, and the problem is inconsistent. The initial guess was set to $x_0 = 0$. The conver-

gence criterion we used for this problem is

$$\|A^T(b - Ax_k)\|_2 < 10^{-8}\|A^T b\|_2.$$

The time to compute $\|A^T(b - Ax_k)\|_2$ was neglected in all the iteration times. The numerical results are given in Table 2. In Table 2, no numerical dropping was performed, i.e. no nonzero elements were neglected in M_k. Hence, $M_k A$ is symmetric, and we can use the MINRES method instead of the GMRES method to solve the preconditioned problems (4.6).

Table 4.2: Results for illc1850

Method	k	ITS	Pre. T	Its. T	Tot. T
Global Pre(k) + GMRES	0	700	1.6e-2	4.25	4.27
	1	661	9.40e-2	4.59	4.69
	2	573	2.81e-1	3.86	4.14
	3	516	6.10e-1	3.50	4.11
	4	476	8.75e-1	3.04	3.92
	5	445	1.14	2.82	3.96
	10	354	2.44	1.94	4.38
Global Pre(k) + MINRES	0	1904	1.6e-2	1.84e-1	2.00e-1
	1	1317	9.40e-2	9.40e-2	$*$1.88e-1
	2	1066	2.81e-1	6.24e-2	3.43e-1
	3	802	6.10e-1	3.42e-2	6.44e-1
	4	796	8.75e-1	3.76e-2	9.13e-1
	5	663	1.14	3.12e-2	1.17
Row-oriented(k) + GMRES	1	658	7.5e-1	4.54	5.29
	2	553	1.35	3.65	5.00
	3	485	2.03	3.15	5.17
	4	449	2.21	2.78	4.99
	5	418	2.29	2.44	4.73
	10	333	2.72	1.75	4.48
CGLS		2083	0.00	3.26e-1	3.26e-1
Diag-CGLS		2081	5.31e-3	3.66e-1	3.72e-1

In Table 4.2, we choose different k in Algorithm 4.2.1 to precondition GMRES and MINRES. In the table, 'ITS' is the number of iterations for the algorithm to reach conver-

gence, 'Pre.T' is the preconditioning time, 'Its.T' is the iteration time, and 'Tol.T' is the total time, in seconds, respectively. The asterisk $*$ indicates the shortest total time in the table. According to the table, we observe that as k increases, the number of iterations decrease significantly for both MINRES and GMRES. All the preconditioners M_k can achieve better iteration numbers than CGLS and diagonal scaled CGLS. The MINRES preconditioned by Algorithm 4.2.1 with $k = 1$ was the fastest in computation time. Comparing MINRES and GMRES, the GMRES required less number of iterations, but the iteration time was longer than MINRES. The reason is that GMRES is more robust against rounding error, but at the cost of the more expensive Gram-Schmidt process. Comparing the global preconditioner and the row-oriented preconditioner, the row-oriented preconditioner required less iterations. However, it requires more preconditioning time than global preconditioners do. When $k = 0$, the global preconditioner and row-oriented preconditioner give the same M_0, thus we did not list the result of $k = 0$ for row-oriented preconditioner.

In Table 3 , we gave results for an over-determined problem sprandn8L, which is larger, much more ill-conditioned and denser. We chose b as $A[1, \ldots, 1]^T$ as the right hand side vector, so that the problem is consistent. The convergence criterion was chosen as

$$\|b - Ax_k\|_2 < 10^{-6}\|b\|_2.$$

Since this problem is larger than the first problem, the numerical dropping strategy was used, which implies that M_kA is not symmetric and the MINRES method cannot be employed. For the global preconditioner we dropped the (i, j) element in $G = (I-MA)A^T$

when

$$|G(i,j)| < \tau_{\text{global}}\|G\|_1$$

holds. For the row-oriented preconditioner, we dropped the i th element in $g_j = (e_j - m_j A)A^T$ when

$$|g_j(i)| < \tau_{\text{row-oriented}}\|g_j\|_2$$

holds. The notations are the same as the ones used in Algorithm 4.2.2 The numerical results are given in Table 4.3. Compared to the global preconditioner, the row-oriented preconditioner usually gives a denser M_k. Thus, the preconditioning time is much longer than that of the global preconditioner. In the table, we set τ_{global} to 10^{-5}, and $\tau_{\text{row-oriented}}$ to 10^{-4}, since they are nearly optimal and give M_k with approximately the same density.

Table 4.3: Results for sprandn8L

Method	k	ITS	Pre. T	Its. T	Tot. T
	0	840	4.80e-1	1.17e+1	$*$1.22e+1
Global Pre(k)	1	847	7.18	1.19e+1	1.91e+1
+ GMRES	2	857	1.53e+1	1.23e+1	2.75e+1
$\tau_{\text{global}} = 10^{-5}$	3	865	2.36e+1	1.25e+1	3.61e+1
Row-oriented(k)	1	890	2.83e+1	1.34e+1	4.17e+1
+ GMRES	2	882	4.09e+1	1.42e+1	5.51e+1
$\tau_{\text{row-oriented}} = 10^{-4}$	3	876	4.85e+1	1.44e+1	6.28e+1
CGLS		50000+	0.00	2.54e+2	2.54e+2
Diag-CGLS		21548	1.60e-2	1.10e+2	1.10e+2

Table 4.3 shows that increasing k does not necessarily result in improved convergence for the global method with droppings. On the other hand, for the row-oriented preconditioner the convergence improves as k is increased. However, the numbers of iterations are more than the global preconditioner. For this extremely ill-conditioned matrix A,

the convergence criterion is stricter than $\|A^T(b - Ax_k)\|_2 < 10^{-8}\|A^Tb\|_2$. The criterion $\|b - Ax_k\|_2 < 10^{-6}\|b\|_2$ is approximately equivalent to $\|A^T(b - Ax_k)\|_2 < 10^{-12}\|A^Tb\|_2$, and the convergence behavior becomes somewhat irregular when a very accurate solution is required.

From Table 4.2 to Table 4.3, we observe that the proposed preconditioning is time-consuming compared to the iteration time especially for large k. Improvement in the iteration time cannot compensate the expense for preconditioning. However, when dealing with multi-right-hand-side problems, the CPU time spent on preconditioning pays off. In Table 4, we solved a multiple right-hand-side problem. The coefficient matrix A is the matrix sprand8S of Table 1. The right-hand-side vectors were given by A times a series of random vectors which were also generated by MATLAB.

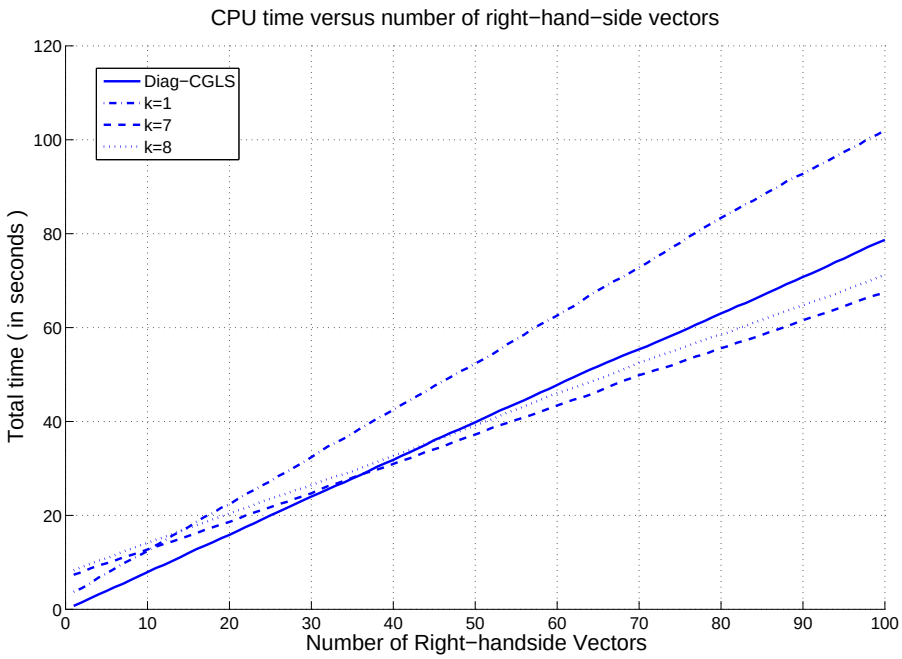
Table 4.4: Results for multiple right-hand-side problem, (sprandn8S)

num of subproblems	k=0	k=1	k=2	k=6	k=7	diag-CGLS
1	1.453	2.798	3.734	6.704	7.357	*7.190e-1
25	4.014e+1	2.946e+1	2.725e+1	2.312e+1	2.174e+1	*1.998e+1
50	7.969e+1	5.670e+1	5.232e+1	4.057e+1	*3.720e+1	3.981e+1
75	1.189e+2	8.448e+1	7.803e+1	5.802e+1	*5.262e+1	5.903e+1
100	1.583e+2	1.128e+2	1.020e+2	7.460e+1	*6.738e+1	7.867e+1

In Table 4.4, we give the total computation time in seconds for solving the least squares problems with different number of right-hand-side vectors b. The global left preconditioning with the MINRES method was used. We can observe that as the number of the right-hand-side vectors increases, the preconditioners become more and more competitive. This is also shown in Figure 4.1.

From Figure 4.1, we see that as k increases, the preconditioning time becomes larger, but the iteration time per problem decreases. When k keeps increasing to 8, the iteration

Figure 4.1: Multiple Right-hand-side Problem



time per problem starts to increase. Hence, there is an optimal k which minimizes the total CPU time. For this problem, the optimal k is 7.

4.4 Conclusion

We applied the approximate inverse preconditioner to least squares problem. Based on the preconditioner M_k from the MR algorithm, we gave equivalence theorems and breakdown free theorems for both the left preconditioning case and the right preconditioning case.

Numerical experiments showed that with the above preconditioning, the MINRES method achieves a faster convergence for solving least squares problems, although the preconditioning is time-consuming. However, for multiple right-hand-side problems, the CPU time

spent on preconditioning pays off.

Chapter 5

Greville's Methods for Preconditioning Least Squares Problems

5.1 Greville's method

In Chapter 4 we proposed using Minimal Residual method to construct the approximate generalized inverse of A and using it as a preconditioner. It has some obvious disadvantages.

- The theorems for the equivalence between the original least squares problem and the preconditioned problem could not be guaranteed when numerical droppings are performed.
- If no numerical droppings are performed, the preconditioner is storage-demanding.
- If numerical droppings are performed, not only the equivalence is not guaranteed, but also the preconditioned coefficient matrix is not symmetric, which implies that

the GMRES method has to be applied to solve the preconditioned rather than the MINRES method.

In this chapter we propose another preconditioning method for least squares problems. The method can not only construct an approximate Moore-Penrose inverse of A, but also can give an incomplete factorization for the Moore-Penrose inverse of A. First we will introduce what the Greville's method is.

Given a rectangular matrix $A \in \mathbb{R}^{m\times n}$, $\mathrm{rank(A)} = \mathrm{r} \leq \min\{\mathrm{m},\mathrm{n}\}$. Assume the Moore-Penrose inverse of A is known, we are interested in how to compute the Moore-Penrose inverse of

$$A + cd^T, \;\; c \in \mathbb{R}^m, \;\; d \in \mathbb{R}^n, \tag{5.1}$$

which is a rank-one update of A. In [19], the following six logical possibilities are considered

1. $c \notin \mathcal{R}(A)$, $d \notin \mathcal{R}(A^T)$ and $1 + d^T A^\dagger c$ arbitrary,
2. $c \in \mathcal{R}(A)$, $d \notin \mathcal{R}(A^T)$ and $1 + d^T A^\dagger c = 0$,
3. $c \in \mathcal{R}(A)$, d arbitrary and $1 + d^T A^\dagger c \neq 0$,
4. $c \notin \mathcal{R}(A)$, $d \in \mathcal{R}(A^T)$ and $1 + d^T A^\dagger c = 0$,
5. c arbitrary, $d \in \mathcal{R}(A^T)$ and $1 + d^T A^\dagger c \neq 0$,
6. $c \in \mathcal{R}(A)$, $d \in \mathcal{R}(A^T)$ and $1 + d^T A^\dagger c = 0$.

For each possibility, an expression for the Moore-Penrose inverse of the rank one update of A is given by the following theorem.

THEOREM 5.1.1 *[19] For $A \in \mathbb{R}^{m\times n}$, $c \in \mathbb{R}^m$, $d \in \mathbb{R}^n$, let $k = A^\dagger c$, $h = d^T A^\dagger$, $u = (I - AA^\dagger)c$, $v = d^T(I - A^\dagger A)$, and $\beta = 1 + d^T A^\dagger c$. Notice that,*

$$c \in \mathcal{R}(A) \quad \Leftrightarrow \quad u = 0 \tag{5.2}$$

$$d \in \mathcal{R}(A^T) \quad \Leftrightarrow \quad v = 0. \tag{5.3}$$

Then, the generalized inverse of $A + cd^T$ is given as follows.

1. *If $u \neq 0$ and $v \neq 0$, then $(A + cd^T)^\dagger = A^\dagger - ku^\dagger - v^\dagger h + \beta v^\dagger u^\dagger$.*

2. *If $u = 0$ and $v \neq 0$, and $\beta = 0$, then $(A + cd^T)^\dagger = A^\dagger - kk^\dagger A^\dagger - v^\dagger h$.*

3. *If $u = 0$ and $\beta \neq 0$, then $(A + cd^T)^\dagger = A^\dagger + \frac{1}{\bar\beta} v^T k^T A^\dagger - \frac{\bar\beta}{\sigma_1} p_1 q_1^T$, where $p_1 = -\left(\frac{\|k\|_2^2}{\beta} v^T + k\right)$, $q_1^T = -\left(\frac{\|v\|_2^2}{\beta} k^T A^\dagger + h\right)$.*

4. *If $u \neq 0$, $v = 0$ and $\beta = 0$, then $(A + cd^T)^\dagger = A^\dagger - A^\dagger h^\dagger h - ku^\dagger$.*

5. *If $v = 0$ and $\beta \neq 0$, then $(A + cd^T)^\dagger = A^\dagger + \frac{1}{\bar\beta} A^\dagger h^T u^T - \frac{\bar\beta}{\sigma_2} p_2 q_2^T$, where $p_2 = -\left(\frac{\|u\|_2}{\beta} A^\dagger h^T + k\right)$, $q_2^T = -\left(\frac{\|h\|_2^2}{\beta} u^T + h\right)$, and $\sigma_2 = \|h\|_2^2 \|u\|_2^2 + |\beta|^2$.*

6. *If $u = 0$, $v = 0$ and $\beta = 0$, then $(A + cd^T)^\dagger = A^\dagger - kk^\dagger A^\dagger - A^\dagger h^\dagger h + (k^\dagger A^\dagger h^\dagger) kh$.*

To utilize the above theorem, we first write A into a column summation form. Let

$$A = \sum_{i=1}^{n} a_i e_i^T, \tag{5.4}$$

where a_i is the ith column of A, and e_i is a n dimensional vector whose elements are all zero except the ith element is one. Further define a sequence of matrices $A_i = [a_1, \ldots, a_i, 0, \ldots, 0]$, $i = 1, \ldots, n$, so that A_i has the same size as A does and the first i columns are the same as

A's, only the last $n-i$ columns are zero. Hence we have

$$A_i = \sum_{k=1}^{i} a_k e_k^T, \;\; i = 1, \ldots, n, \tag{5.5}$$

and if we denote $A_0 = 0_{m\times n}$, then

$$A_i = A_{i-1} + a_i e_i^T, \;\; i = 1, \ldots, n. \tag{5.6}$$

Thus every A_i, $i = 1, \ldots, n$ is a rank-one update of A_{i-1}. Noticing that $A_0^\dagger = 0_{n\times m}$, we can utilize Theorem 5.1.1 to compute the Moore-Penrose inverse of A step by step and have $A^\dagger = A_n^\dagger$ in the end.

In Theorem 5.1.1, substituting c with a_i and d with e_i, we can rewrite Equation 5.2 as following,

$$a_i \notin \mathcal{R}(A_{i-1}) \Leftrightarrow u = (I - A_{i-1}A_{i-1}^\dagger)a_i \neq 0. \tag{5.7}$$

Equation (5.3) becomes

$$v = e_i^T(I - A_{i-1}^\dagger A_{i-1})a_i \neq 0, \;\; i = 1, \ldots, n \tag{5.8}$$

The β in Theorem 5.1.1 is nonzero for any $i = 1, 2, \ldots n$.

$$\beta = 1 + e_i^T A_{i-1}^\dagger a_i = 1. \tag{5.9}$$

Hence, we can use Case 1 in Theorem 5.1.1 for column $a_i \notin \mathcal{R}(A_{i-1})$ and Case 3 in Theorem 5.1.1 for column $a_i \in \mathcal{R}(A_{i-1})$.

Then from Theorem 5.1.1, denoting $A_0 = 0_{m\times n}$, we obtain a method to compute $A_i^\dagger$ based on $A_{i-1}^\dagger$ as

$$A_i^\dagger = \begin{cases} A_{i-1}^\dagger + (e_i - A_{i-1}^\dagger a_i)((I - A_{i-1}A_{i-1}^\dagger)a_i)^\dagger & \text{if} \quad a_i \notin \mathcal{R}(A_{i-1}) \\ A_{i-1}^\dagger + \frac{1}{\sigma_i}(e_i - A_{i-1}^\dagger a_i)(A_{i-1}^\dagger a_i)^T A_{i-1}^\dagger & \text{if} \quad a_i \in \mathcal{R}(A_{i-1}) \end{cases}, \tag{5.10}$$

where $\sigma_i = 1 + \|k_i\|_2^2$. This method was proposed by Greville in the 1960s [31].

Notice that $A_i A_i^\dagger$ is an orthogonal projection on to $\mathcal{R}(A_{i-1})$, hence,

$$a_i \notin \mathcal{R}(A_{i-1}) \quad \Leftrightarrow \quad (I - A_{i-1}A_{i-1}^\dagger)a_i = u_i \neq 0. \tag{5.11}$$

To decide whether $a_i \in \mathcal{R}(A_{i-1})$, we only need to see if u_i is a zero vector or not.

5.2 Global Algorithm for General Matrices

In this section, we will construct our preconditioning algorithm according to the Greville's method of section 5.1. First of all, we notice that the different part between case $a_i \notin \mathcal{R}(A_{i-1})$ and case $a_i \in \mathcal{R}(A_{i-1})$ in Equation (5.10) lies in the second term. We first make some denotations

$$k_i = A_{i-1}^\dagger a_i, \tag{5.12}$$

$$u_i = a_i - A_{i-1}k_i = (I - A_{i-1}A_{i-1}^\dagger)a_i, \tag{5.13}$$

$$\sigma_i = 1 + \|k_i\|_2^2. \tag{5.14}$$

If we define f_i and v_i as

$$f_i = \begin{cases} \|u_i\|_2^2 & \text{if} \quad a_i \notin \mathcal{R}(A_{i-1}) \\ \sigma_i & \text{if} \quad a_i \in \mathcal{R}(A_{i-1}) \end{cases}, \tag{5.15}$$

$$v_i = \begin{cases} u_i & \text{if} \quad a_i \notin \mathcal{R}(A_{i-1}) \\ (A_{i-1}^{\dagger})^T k_i & \text{if} \quad a_i \in \mathcal{R}(A_{i-1}) \end{cases}, \tag{5.16}$$

we can express $A_i^{\dagger}$ in a unified form for any matrices as

$$A_i^{\dagger} = A_{i-1}^{\dagger} + \frac{1}{f_i}(e_i - k_i)v_i^T, \quad i = 1, \ldots, n, \tag{5.17}$$

and we have

$$A^{\dagger} = \sum_{i=1}^{n} \frac{1}{f_i}(e_i - k_i)v_i^T. \tag{5.18}$$

If we denote

$$K = [k_1, \ldots, k_n] \in \mathbb{R}^{n \times n}, \tag{5.19}$$

$$V = [v_1, \ldots, v_n] \in \mathbb{R}^{m \times n}, \tag{5.20}$$

$$F = \begin{bmatrix} f_1 & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & f_n \end{bmatrix} \in \mathbb{R}^{n \times n}, \tag{5.21}$$

the above summation equation can be written into a matrix product form, hence, we obtain a matrix factorization of $A^{\dagger}$ as follows.

THEOREM 5.2.1 *Let* $A \in \mathbb{R}^{m \times n}$ *and* $\text{rank}(A) \leq \min\{m, n\}$. *Using the above notations,*

the Moore-Penrose inverse of A has the following factorization

$$A^{\dagger} = (I - K)F^{-1}V^{T}. \tag{5.22}$$

Here I is the identity matrix of order n, K is a strict upper triangular matrix, F is a diagonal matrix, whose diagonal elements are all positive.

If A is full column rank, then

$$V = A(I - K) \tag{5.23}$$

$$A^{\dagger} = (I - K)F^{-1}(I - K)^{T}A^{T}. \tag{5.24}$$

PROOF. Denote $\bar{A}_i = [a_1, \ldots, a_i]$, then since

$$k_i = A_{i-1}^{\dagger}a_i \tag{5.25}$$

$$= [a_1, \ldots, a_{i-1}, 0, \ldots, 0]^{\dagger}a_i \tag{5.26}$$

$$= \left[\bar{A}_{i-1}, 0, \ldots, 0\right]^{\dagger} a_i \tag{5.27}$$

$$= \begin{bmatrix} \bar{A}_{i-1}^{\dagger} \\ 0 \end{bmatrix} a_i \tag{5.28}$$

$$= \begin{bmatrix} k_{i,1} \\ \vdots \\ k_{i,i-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \tag{5.29}$$

$K = [k_1, \ldots, k_n]$ is a strictly upper triangular matrix.

Since $u_i = 0 \Leftrightarrow a_i \in \mathcal{R}(A_{i-1})$ and

$$f_i = \begin{cases} \|u_i\|_2^2 & \text{if} \quad a_i \notin \mathcal{R}(A_{i-1}) \\ \sigma_i & \text{if} \quad a_i \in \mathcal{R}(A_{i-1}) \end{cases}. \tag{5.30}$$

Thus $f_i(i = 1, \ldots, n)$, are always positive, which implies that F is a diagonal matrix with positive diagonal elements.

If A is a full rank matrix, we have

$$\begin{aligned} V &= [u_1, \ldots, u_n] && (5.31)\\ &= \left[(I - A_0 A_0^\dagger)a_1, \ \ldots, \ (I - A_{n-1}A_{n-1}^\dagger)a_n\right] && (5.32)\\ &= [a_1 - A_0 k_1, \ \ldots, \ a_n - A_{n-1}k_n] && (5.33)\\ &= A - [A_0 k_1, \ \ldots, \ A_{n-1}k_n] && (5.34)\\ &= A - [A_1 k_1, \ \ldots, \ A_n k_n] && (5.35)\\ &= A(I - K). && (5.36) \end{aligned}$$

The second from the bottom equality follows from the fact that K is a strictly upper triangular matrix. Now, when $A^\dagger$ is full rank, can be decomposed as follows,

$$A^\dagger = (I - K)F^{-1}V^T = (I - K)F^{-1}(I - K)^T A^T. \qquad \square \tag{5.37}$$

REMARK 2 *According to Theorem 5.2.1, when A is full column rank, in exact arithmetic*

the factorization of $A^\dagger$ in Theorem 5.2.1 can be rewritten as

$$A^\dagger = (I - K)F^{-1}(I - K)^T A^T. \tag{5.38}$$

Hence, we obtain that,

$$(A^T A)^{-1} = (I - K)F^{-1}(I - K)^T. \tag{5.39}$$

And if we define $(I - K)^{-T} = L$, the above equation equals

$$A^T A = LFL^T, \tag{5.40}$$

which is a LDL^T decomposition of $A^T A$.

Based on Greville's method, we obtain a simple algorithm. We only want to construct a sparse approximation to the Moore-Penrose inverse of A, hence, we perform some numerical droppings in the middle of the algorithm to maintain the sparsity of the preconditioner. We call the following algorithm the *Global* Greville Preconditioning algorithm, since it forms or updates the whole matrix at a time rather than column by column.

ALGORITHM 5.2.1 *Global Greville Preconditioning algorithm*

1. *set* $M_0 = 0$
2. *for* $i = 1 : n$
3. $k_i = M_{i-1}a_i$
4. *perform numerical droppings on* k_i
5. $u_i = a_i - A_{i-1}k_i$
6. *if* a_i *is recognized as* $\notin \mathcal{R}(A_{i-1})$
7. $f_i = \|u_i\|_2^2$

8. $v_i = u_i$

9. *else*

10. $f_i = 1 + \|k_i\|_2^2$

11. $v_i = M_{i-1}^T k_i$

12. *end if*

13. $M_i = M_{i-1} + \frac{1}{f_i}(e_i - k_i)v_i^T$

14. *end for*

15. *Get* $M_n \approx A^\dagger$.

REMARK 3 *In Line 6, we use "*a_i *is recognized as* $\notin \mathcal{R}(A_{i-1})$*" as an if condition. From the previous discussion, we know that* $\|u_i\|_2 \neq 0 \Leftrightarrow a_i \notin \mathcal{R}(A_{i-1})$*. However, because of the rounding error and the droppings we perform,* $\|u_i\|_2 \neq 0$ *or* $\|u_i\|_2$ *being small is not reliable enough to be used to judge* $a_i \in \mathcal{R}(A_{i-1})$ *or not. We will come back to this issue in Subsection 5.6.1.*

REMARK 4 *In Algorithm 5.2.1, actually, we do not need to store* k_i*,* v_i*,* f_i*,* $i = 1, \ldots, n$*, because we form the* $M_i^\dagger$ *explicitly.*

REMARK 5 *In Algorithm 5.2.1, we need to perform numerical droppings on* k_i*, however, doing this cannot control the sparsity in* M_i *directly. When* k_i *is sparse, to update* M_{i-1}*, we only need to update the rows which correspond to the nonzero elements in* k_i*. Hence, the rank-one update will be cheaper.*

we also need to update M_i *in every step, but actually we do not need to update the whole matrix, since only the first* $i-1$ *rows of* M_{i-1} *could be nonzero. Hence, to compute* M_i*, we need to update the first* $i-1$ *rows of* M_{i-1}*, and then add one new nonzero row to be the* i*th row of* M_i*.*

THEOREM 5.2.2 *Let $A \in \mathbb{R}^{m\times n}$ and* rank$(A) \leq \min\{m, n\}$. *Use the notations in Algorithm 5.2.1 and let $K = [k_1, \ldots, k_n]$, $V = [v_1, \ldots, v_n]$ and $F = \mathrm{diag}\{f_1, \ldots, f_n\}$, then the matrix M_n constructed by Algorithm 5.2.1 has the following factorization*

$$M_n = (I - K)F^{-1}V^T. \tag{5.41}$$

Here I is the identity matrix of order n, K is a strict upper triangular matrix, F is a diagonal matrix, whose diagonal elements are all positive.

If A is full column rank, then

$$V = A(I - K) \tag{5.42}$$

$$M_n = (I - K)F^{-1}(I - K)^T A^T. \tag{5.43}$$

REMARK 6 *Comparing Theorem 5.2.1 and the above theorem, we can see that $A^\dagger$ and the M_n from the Algorithm 5.2.1 have the same factorization. The only difference is that $M_n \approx A^\dagger$ because of the numerical droppings. And this M_n can be used as an approximate generalized inverse preconditioner.*

REMARK 7 *We can also perform numerical droppings to M_i after it is updated. In this case the structure of M_i will be ruined, which means that M_n could not be written into the factorization form in the above theorem.*

5.3 Vector-wise Algorithm for General Matrices

If we want to construct the matrix K, F and V without forming M_i explicitly, we can use a vector-wise version of Algorithm 5.2.1. In Algorithm 5.2.1, the column vectors of K are constructed one column at a step, then we compute a v_i vector based on k_i, and

the computation of the ith diagonal element of F is based on v_i. We can see that all the definition of other vectors are based on k_i. Hence, it is possible to rewrite Algorithm 5.2.1 into a vector-wise form.

Since u_i can be computed from $a_i - A_{i-1}k_i$, which does not refer to M_{i-1} explicitly, to vectorize Algorithm 5.2.1, we only need to form k_i and $v_i = M_{i-1}^T k_i$ when linear dependence happens, without using M_{i-1} explicitly.

Consider the numerical droppings are not used. Since we already know that

$$\begin{aligned} A^\dagger &= (I-K)F^{-1}V^T. \\ &= (I - \begin{bmatrix} k_1 & \dots & k_n \end{bmatrix}) \begin{bmatrix} f_1^{-1} & & \\ & \ddots & \\ & & f_n^{-1} \end{bmatrix} \begin{bmatrix} v_1^T \\ \vdots \\ v_n^T \end{bmatrix} \\ &= \sum_{i=1}^{n} (e_i - k_i)\frac{1}{f_i}v_i^T, \end{aligned}$$

for any integer p, it is easy to see that

$$A_p^\dagger = \sum_{i=1}^{p} (e_i - k_i)\frac{1}{f_i}v_i^T. \tag{5.44}$$

Therefore, when a_i is in the range space of A_{i-1}, we have

$$\begin{aligned} v_i &= (A_{i-1}^{\dagger})^T k_i && (5.45)\\ &= (\sum_{p=1}^{i-1}(e_p - k_p)\frac{1}{f_p}v_p^T)^T k_i && (5.46)\\ &= \sum_{p=1}^{i-1}\frac{1}{f_p}v_p(e_p - k_p)^T k_i. && (5.47) \end{aligned}$$

Also for k_i

$$\begin{aligned} k_i &= A_{i-1}^{\dagger}a_i && (5.48)\\ &= \sum_{p=1}^{i-1}(e_p - k_p)\frac{1}{f_p}v_p^T a_i && (5.49)\\ &= \sum_{p=1}^{i-2}(e_p - k_p)\frac{1}{f_p}v_p^T a_i + (e_{i-1} - k_{i-1})\frac{1}{f_{i-1}}v_{i-1}^T a_i && (5.50)\\ &= A_{i-2}^{\dagger}a_i + (e_{i-1} - k_{i-1})\frac{1}{f_{i-1}}v_{i-1}^T a_i. && (5.51) \end{aligned}$$

To make this more clear, from the last column of K, the requirement relationship can be shown as

$$\begin{array}{cccc} & & k_n = A_{n-1}^{\dagger}a_n & \\ & \nearrow & & \nwarrow \\ & A_{n-2}^{\dagger}a_n & & k_{n-1} = A_{n-2}^{\dagger}a_{n-1} \\ \nearrow & & \nwarrow & \nearrow & & \nwarrow \\ A_{n-3}^{\dagger}a_n & k_{n-2} = A_{n-3}^{\dagger}a_{n-2} & & A_{n-3}^{\dagger}a_{n-1} & k_{n-2} = A_{n-3}^{\dagger}a_{n-2} \\ & & \cdots\cdots & & \end{array}$$

In other words, we need to compute every $A_i^{\dagger}a_k,\ k = i+1, \ldots, n$. Denote $A_i^{\dagger}a_j,\ j > i$ as

$k_{i,j}$. In this sense, $k_i = k_{i-1,i}$. In the algorithm, $k_{i,j},\ j > i$ will be stored in the jth column of K, if $j = i+1$, $k_{i,j} = k_j$, and it will not be changed any more. If $j > i+1$, $k_{i,j}$ will be updated to $k_{i+1,j}$ and still stored in the same position.

Based on the above discussion, and add the numerical dropping strategy, we can write the following algorithm. In the algorithm we omit the first subscript of $k_{i,j}$, since all the vectors $k_{i,j},\ i = 1, \ldots, n$ are stored in the jth column of K, they are actually one vector in different iterations.

ALGORITHM 5.3.1 *Vector-wise Greville Preconditioning Algorithm*

1. *set* $K = 0_{n\times n}$
2. *for* $i = 1 : n$
3. $\quad u = a_i - A_{i-1}k_i$
4. $\quad$ *if* a_i *is recognized as* $\notin \mathcal{R}(A_{i-1})$
5. $\quad\quad f_i = \|u\|_2^2$
6. $\quad\quad v_i = u$
7. $\quad$ *else*
8. $\quad\quad f_i = \|k_i\|_2^2 + 1$
9. $\quad\quad v_i = (A_{i-1}^{\dagger})^T k_i = \sum_{p=1}^{i-1} \frac{1}{f_p} v_p (e_p - k_p)^T k_i$
10. $\quad$ *end if*
11. $\quad$ *for* $j = i+1, \ldots, n$
12. $\quad\quad k_j = k_j + \frac{v_i^T a_j}{f_i}(e_i - k_i)$
13. $\quad\quad$ *perform numerical droppings on* k_j
14. $\quad$ *end for*
15. *end for*
16. $K = [k_1, \ldots, k_n]$, $F = \mathrm{Diag}\{f_1, \ldots, f_n\}$, $V = [v_1, \ldots, v_n]$.

REMARK 8 *For the full column rank case, we already showed that $V = A(I-K)$. Hence, we do not need to store matrix V in this case. However, it does not mean that for the general case, we need to store the whole matrix V. In fact, we only need to store the vectors of V which correspond to the columns a_i recognized to be in the range space of A_{i-1}. Hence, if the rank deficiency is small, the extra storage compared to the full rank case is small.*

5.4 Greville Preconditioning Method for Full Column Rank Matrices

In this section, we especially take a look at the full column rank case. When A is full column rank, both Algorithm 5.3.1 can be simplified as follows.

ALGORITHM 5.4.1 *Vector-wise Greville Preconditioning Algorithm for Full Column Rank Matrices*

1. *set* $K = 0_{n\times n}$
2. *for* $i = 1 : n$
3. $u_i = a_i - A_{i-1}k_i$
4. $f_i = \|u_i\|_2^2$
5. *for* $j = i+1, \ldots, n$
6. $k_j = k_j + \frac{u_i^T a_j}{f_i}(e_i - k_i)$
7. *perform numerical droppings on* k_j
8. *end for*
9. *end for*
10. $K = [k_1, \ldots, k_n]$, $F = \mathrm{Diag}\{f_1, \ldots, f_n\}$.

In Algorithm 5.4.1,

$$\begin{aligned} u &= a_i - A_{i-1}k_i \\ &= [a_1, \ldots, a_i, 0, \ldots, 0] \begin{bmatrix} -k_{i,1} \\ \vdots \\ -k_{i,i-1} \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ &= A_i(e_i - k_i) \\ &= A(e_i - k_i). \end{aligned}$$

If we denote $e_i - k_i$ as z_i, then $u_i = Az_i$.

The Line 6 in the Algorithm 5.4.1, can also be rewritten as

$$\begin{aligned} k_j &= k_j + \frac{u_i^T a_j}{\|u_i\|_2^2}(e_i - k_i) \\ e_j - k_j &= e_j - k_j - \frac{u_i^T a_j}{\|u_i\|_2^2}(e_i - k_i) \\ z_j &= z_j - \frac{u_i^T a_j}{\|u_i\|_2^2} z_i. \end{aligned}$$

Denote $d_i = \|u_i\|_2^2$, $\theta = \dfrac{u_i^T a_j}{d_i}$ and $Z = [z_1, \ldots, z_n]$. Then combining all the new notations, we can rewrite the algorithm as follows.

ALGORITHM 5.4.2

1. *set* $Z = I_{n\times n}$

2. *for* $i = 1 : n$
3. $u_i = A_i z_i$
4. $d_i = (u_i, u_i)$
5. *for* $j = i + 1, \ldots, n$
6. $\theta = \frac{(u_i, a_j)}{d_i}$
7. $z_j = z_j - \theta z_i$
8. *perform numerical droppings on* z_j.
9. *end for*
10. *end for*
11. $Z = [z_1, \ldots, z_n]$, $D = \mathrm{Diag}\{d_1, \ldots, d_n\}$.

REMARK 9 *Since* $z_i = e_i - k_i$, *in the beginning of this algorithm we have* $Z = I - K$, *where* I *is an identity matrix of order* n. *Denoting* $D = \mathrm{Diag}\{d_1, \ldots, d_n\}$, *when there is no numerical droppings and in exact arithmetic, the factorization of* $A^\dagger$ *in Theorem 5.2.1 can be rewritten as*

$$A^\dagger = ZD^{-1}Z^T A^T \tag{5.52}$$

When numerical droppings is performed, we can construct a matrix M *in the factorization form*

$$M = ZD^{-1}Z^T A^T \approx A^\dagger. \tag{5.53}$$

In [13] Benzi and Tůma proposed a technique, robust incomplete factorization (RIF) , for constructing robust preconditioners for the CGLS method applied to the solution of large and sparse least squares problems. The algorithm computes an incomplete LDL^T factorization of the normal equations matrix without the need to form the normal matrix itself. The RIF idea was also introduced in [9,12], where Benzi and Tůma applied a similar idea to construct a preconditioner for CG method to solve symmetric positive definite linear systems. Their ideas are based on a $A^T A$-Orthogonalization when A is a nonsymmetric

nonsingular matrix or A-Orthogonalization when A is a symmetric positive definite matrix.

We describe the A-Orthogonalization procedure as follows. Let a_i^T denote the ith row of A. Also, let e_i denote the ith unit basis vector. The basic A-orthogonalization procedure, which orthogonalizes the unit basis vectors with respect to the inner product defined as $(x, y)_A = y^T Ax$, can be written as follows.

ALGORITHM 5.4.3 *[9]*

1. *Let* $z_0 = e_i$, *(*$1 \leq i \leq n$*)*
2. *For* $i = 1, 2, \ldots, n$
3. *For* $j = i, i+1, \ldots, n$
4. $p_j^{(i-1)} = a_i^T z_j^{(i-1)}$
5. *End For*
6. *If* $i = n$ *go to* (11)
7. *For* $j = i+1, \ldots, n$
8. $z_j^{(i)} = z_j^{(i-1)} - (\frac{p_j^{(i-1)}}{p_i^{(i-1)}}) z_i^{(i-1)}$
9. *End For*
10. *End For*
11. *Let* $z_i = z_i^{(i-1)}$, $p_i = p_i^{(i-1)}$ *for* $1 \leq i \leq n$. *Return* $Z = [z_1, \ldots, z_n]$ *and* $D = \mathrm{Diag}\{p_1, \ldots, p_n\}$.

The $\mathrm{A^TA}$-orthogonalization procedure is similar to the above algorithm. The only difference lies in the Line 4. In Algorithm 5.4.3, line 4 can be rewritten into the following

inner product form

$$\begin{aligned} p_j^{(i-1)} &= a_i^T z_j^{(i-1)} && (5.54)\\ &= (Ae_i)^T z_j^{(i-)1} && (5.55)\\ &= e_i^T A z_j^{(i-1)} && (5.56)\\ &= (e_i, z_j^{(i-1)})_A. && (5.57) \end{aligned}$$

If we change the definition of $p_j^{(i-1)}$ to the following, can easily obtain $\mathrm{A^TA}$-orthogonalization.

$$\begin{aligned} p_j^{(i-1)} &= a_i^T A z_j^{(i-1)} && (5.58)\\ &= (Ae_i)^T A z_j^{(i-1)} && (5.59)\\ &= e_i^T A^T A z_j^{(i-1)} && (5.60)\\ &= (e_i, z_j^{(i-1)})_{A^TA}. && (5.61) \end{aligned}$$

Now if we look back our Algorithm 5.4.2, the definition of θ can be rewritten as follows,

$$\begin{aligned} \theta &= \frac{u_i^T a_j}{d_i} && (5.62)\\ &= \frac{(Az_i)^T (Ae_j)}{(Az_i)^T (Az_i)} && (5.63)\\ &= \frac{(e_j, z_i)_{A^TA}}{(z_i, z_i)_{A^TA}}. && (5.64) \end{aligned}$$

Hence, in Algorithm 5.4.2, from Line 5 to Line 9, it can be viewed as an $\mathrm{A^TA}$-orthogonalization procedure when the numerical droppings are not performed. When the droppings are performed, it is an incomplete $\mathrm{A^TA}$-orthogonalization procedure.

And also notice that, for the above A-orthogonalization procedure or the $\mathrm{A^TA}$-orthogonalization procedure or our Algorithm 5.4.2, in the ith step, all the vectors $z_j,\ j = i+1, \ldots, n$ are updated, and the vectors $z_j,\ j = 1, \ldots i$ are not changed and also remain the same in the following steps. This is called the *right-looking* process. For each of these algorithms, according to [10], a *left-looking* variant also exists, which is sometimes advantageous. The left-looking version of Algorithm 5.4.2 can be written as follows.

ALGORITHM 5.4.4 *Vector-wise Greville Preconditioning Algorithm*

1. *Set* $Z = I_{n\times n}$, $u = a_1$, $f_1 = \|a_1\|_2^2$
2. *For* $i = 2, \ldots, n$
3. *For* $j = 1, \ldots, i-1$
4. $\theta = a_i^T u_j$
5. $z_i = z_i - \frac{\theta}{f_j} z_j$
6. *End for*
7. *perform numerical droppings on* z_i
8. $u_i = Az_i$
9. $f_i = \|u_i\|_2^2$
10. *End For*
11. $Z = [z_1, \ldots, z_n]$, $F = \mathrm{Diag}\{f_1, \ldots, f_n\}$.

From the above algorithm, it is easy to see that this algorithm coincides a *Gram–Schmidt process* with respect to the inner product $(x, y)_{A^TA} = xA^TAy$. We can change it to a *Modified Gram-Schmidt* process to obtain a more stable version of this $\mathrm{A^TA}$-orthogonalization procedure.

ALGORITHM 5.4.5 *Vector-wise Greville Preconditioning Algorithm*

1. *Set* $Z = I_{n\times n}$, $u = a_1$, $f_1 = \|a_1\|_2^2$
2. *For* $i = 2, \ldots, n$
3. *For* $j = 1, \ldots, i-1$

4. $\theta = (Az_i)^T u_j$
5. $z_i = z_i - \frac{\theta}{f_j} z_j$
6. *End for*
7. *perform numerical droppings on* z_i
8. $u_i = Az_i$
9. $f_i = \|u_i\|_2^2$
10. *End For*
11. $Z = [z_1, \ldots, z_n]$, $F = \mathrm{Diag}\{f_1, \ldots, f_n\}$.

And in the same way, we can also rewrite our Algorithm 5.3.1 into left-looking version.

ALGORITHM 5.4.6 *Vector-wise Greville Preconditioning Algorithm*

1. *set* $K = 0_{n\times n}$, $v_1 = a_1$, $f_1 = \|a_1\|_2^2$
2. *for* $i = 2 : n$
3. *for* $j = 1 : i-1$
4. $\theta = a_i^T v_j$
5. $k_i = k_i + \frac{\theta}{f_j}(e_j - k_j)$
6. *end for*
7. *perform numerical droppings on* k_i
8. $u = a_i - A_{i-1}k_i$
9. *if* a_i *is recognized as* $\notin \mathcal{R}(A_{i-1})$
10. $f_i = \|u\|_2^2$
11. $v_i = u$
12. *else*
13. $f_i = \|k_i\|_2^2 + 1$
14. $v_i = (A_{i-1}^{\dagger})^T k_i = \sum_{p=1}^{i-1} \frac{1}{f_p} v_p (e_p - k_p)^T k_i$
15. *end if*
16. *end for*
17. $K = [k_1, \ldots, k_n]$, $F = \mathrm{Diag}\{f_1, \ldots, f_n\}$, $V = [v_1, \ldots, v_n]$.

5.5 Two Important Issues about Preconditioning the Least Squares Problems

When we precondition a nonsingular linear system

$$Ax = b, \;\; A \in \mathbb{R}^{n \times n}, \;\; b \in \mathbb{R}^n, \tag{5.65}$$

instead of solving the linear system itself, we solve

$$MAx = Mb \tag{5.66}$$

or

$$AMy = b, \;\; x = My. \tag{5.67}$$

As long as preconditioner $P \in \mathbb{R}^{n \times n}$ is nonsingular, we can ensure we obtain the solution to the original solution.

In the case of least squares problems this is not true, even when the coefficient matrix is a full rank matrix. In this section, we will discuss to ensure that we obtain the least squares solution to the original problem 2.1 by solving the preconditioned problem, what conditions should the preconditioner M satisfy. And after transforming the original problem to the preconditioned problem we want to use some Krylov subspace methods to solve it. We will also discuss that when the Krylov subspace solver can determine a solution to the preconditioned problem before it breaks down.

5.5.1 Equivalence Condition

Consider solving the least squares problem (2.1) by transforming it into the left preconditioned form,

$$\min_{x\in\mathbb{R}^n} \|Mb - MAx\|_2, \tag{5.68}$$

where $A \in \mathbb{R}^{m\times n}$, $M \in \mathbb{R}^{n\times m}$, and b is a right-hand-side vector $b \in \mathbb{R}^m$.

Since we are dealing with general rectangular matrices, we need some other conditions to ensure that the preconditioned problem (5.68) is equivalent to the original least squares problem (2.1). These conditions have been introduced in Chapter 3. For the left preconditioning, by Theorem 3.3.5, we need $\mathcal{R}(A) = \mathcal{R}(M^T MA)$.

If we perform Algorithm 5.2.1 or Algorithm 5.3.1 completely and exactly, we will finally have the exact Moore-Penrose inverse of A, i.e. $M = A^\dagger$. By the properties of $A^\dagger$ [50], it is easy to know that Theorem **??** is satisfied. However, we need to performe some numerical droppings to control the sparsity of the preconditioner M. Assume the dropping threshold is τ, we drop the elements in k_i which are smaller than τ in Algorithm 5.2.1 or Algorithm 5.3.1. Because of the droppings, the norm of u_i may not be an accurate way to detect if $a_i \in \mathcal{R}(A_{i-1})$ or $a_i \notin \mathcal{R}(A_{i-1})$. We will come back to how to detect the linear dependence later. After droppings, we have

$$A^\dagger \approx M = (I - K)F^{-1}V^T. \tag{5.69}$$

To analyze the equivalence between the original problem (2.14) and the preconditioned problem (5.68), where M is from any of our algorithms, we first consider the simple case,

in which A is a full column rank matrix. After numerical droppings, we have,

$$A^{\dagger} \approx M = (I - K)F^{-1}U^T, \tag{5.70}$$

where U is

$$U = A(I - K). \tag{5.71}$$

Notice that K is a strictly upper triangular matrix and F is a diagonal matrix with positive elements. Hence, we can denote

$$M = CA^T, \tag{5.72}$$

where C is an nonsingular matrix. According to the discussion in [34], $M = CA^T$ satisfies Theorem 3.3.5, hence, we have the following result.

THEOREM 5.5.1 *If $A \in \mathbb{R}^{m\times n}$, and A is full column rank, by Algorithm 5.2.1 or Algorithm 5.3.1 with numerical droppings, we can construct a preconditioner M. With this preconditioner M, the preconditioned least squares problem and the original least squares problem are equivalent and GMRES can determine a least squares solution to the preconditioned problem before breakdown happens.*

For the general case, we still have K and F nonsingular. However, the expression for V is not straightforward. To simplify the problem, we need to make the following assumptions.

ASSUMPTION 5.5.1

- *There is no zero column in A.*
- *Our algorithm can detect all the linear independence correctly.*

The assumption of no zero columns is very general, since if there is a zero column in the coefficient matrix A, we only need to omit the corresponding element in the variable vector x. For the other assumption of detecting all the linear independence, the most simple case is that no matter A is full column rank or not we take it as a full column rank matrix. Doing so may cause breakdown for the preconditioning algorithm.

The definition of v_i can be rewritten as follows.

$$v_i = \begin{cases} a_i - Ak_i \in \mathcal{R}(A_i) \notin \mathcal{R}(A_{i-1}) & \text{if} \quad a_i \notin \mathcal{R}(A_{i-1}) \\ (A_{i-1}^{\dagger})^T k_i \in \mathcal{R}(A_{i-1}) = \mathcal{R}(A_i) & \text{if} \quad a_i \in \mathcal{R}(A_{i-1}) \end{cases} \tag{5.73}$$

When Assumption 5.5.1 is satisfied , we have

$$\text{span}\{v_1, v_2, \ldots, v_i\} = \text{span}\{a_1, a_2, \ldots, a_i\}. \tag{5.74}$$

On the other hand, note the 12-th line of Algorithm 5.2.1

$$M_i = M_{i-1} + \frac{1}{f_i}(e_i - k_i)v_i^T, \tag{5.75}$$

which implies that every row of M_i is a linear combination of the vectors $v_k^T,\ 1 \leq k \leq i$, i.e.,

$$\mathcal{R}(M_i^T) = \text{span}\{v_1, \ldots, v_i\}. \tag{5.76}$$

Based on the above discussions, we obtain the following theorem.

THEOREM 5.5.2 *Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$. If Assumption 5.5.1 holds, then we have the following relationships, where M is the approximate Moore-Penrose inverse constructed*

by Algorithm 5.2.1.

$$\mathcal{R}(M^T) = \mathcal{R}(V) = \mathcal{R}(A) \tag{5.77}$$

REMARK 10 *In Assumption 5.5.1, we assume that our algorithms can detect all the linear independence in the columns of A. Hence, we allow such mistakes that a linearly dependent column is recognized as a linearly independent column. An extreme case is that we recognize all the columns of A as linearly independent, i.e., we take A as a full column rank matrix. In this sense, our assumption can always be satisfied.*

Hence, we proved that for any matrix $A \in \mathbb{R}^{m\times n}$, $\mathcal{R}(A) = \mathcal{R}(M^T)$. We have the following theorem.

THEOREM 5.5.3 *If Assumption 5.5.1 holds, then for all $b \in \mathbb{R}^m$, the preconditioned least squares problem (5.68), where M is constructed by Algorithm 5.2.1, is equivalent to the original least squares problem (2.14).*

PROOF. If Assumption 5.5.1 holds, we have

$$\mathcal{R}(A) = \mathcal{R}(M^T). \tag{5.78}$$

Then there exists a nonsingular matrix $C \in \mathbb{R}^{n\times n}$ such that $A = M^T C$. Hence,

$$\begin{aligned}
\mathcal{R}(M^T MA) &= \mathcal{R}(M^T MM^T C) && (5.79)\\
&= \mathcal{R}(M^T MM^T) && (5.80)\\
&= \mathcal{R}(M^T M) && (5.81)\\
&= \mathcal{R}(M^T) && (5.82)\\
&= \mathcal{R}(A). && (5.83)
\end{aligned}$$

In the above equalities we used the relationship $\mathcal{R}(MM^T) = \mathcal{R}(M)$.

By Theorem 3.3.5 we complete the proof. □

THEOREM 5.5.4 *For all $b \in \mathbb{R}^m$, M is constructed by Algorithm 5.2.1, if Assumption 5.5.1 holds and M is used as a left preconditioner, the least squares problem (5.68) is equivalent to the original least squares problem (2.1).*

5.5.2 Breakdown Free Condition

In this subsection we assume without losing generality that the first r columns of A are linearly independent. Hence,

$$\mathcal{R}(A) = \text{span}\{a_1, a_2, \ldots, a_r\}, \tag{5.84}$$

where $\text{rank}(A) = r$, and $a_i, (i = 1, 2, \ldots, r)$ is the ith column of A. The reason is that we can incorporate a column pivoting in Algorithm 5.2.1 easily. With Assumption 5.5.1, every time when a linear dependence is detected, we can pivot the current column to the end of the matrix A, and after we have the least squares solution to the pivoted A, we can permute the solution to get the solution to the original problem.

Then we have,

$$a_i \in \mathcal{R}(A_r), \quad i = r+1, r+2, \ldots, n. \tag{5.85}$$

In this case, after performing Algorithm 5.2.1 with numerical dropping, matrix V can be written in the form

$$V = [u_1, u_2, \ldots, u_r, v_{r+1}, v_{r+2}, \ldots, v_n]. \tag{5.86}$$

If we denote $[u_1, u_2, \ldots, u_r]$ as U_r, then

$$U_r = A(I - K)I_r, \quad I_r = \begin{bmatrix} I_{r\times r} \\ 0 \end{bmatrix}. \tag{5.87}$$

From Theorem 5.5.2,

$$\begin{aligned} \mathcal{R}(V) &= \mathcal{R}(A) && (5.88)\\ \mathcal{R}(V) &= \text{span}\{U_r\} && (5.89)\\ &\Longrightarrow \text{span}\{v_{r+1}, v_{r+2}, \ldots, v_n\} \subseteq \text{span}\{U_r\}. && (5.90) \end{aligned}$$

Therefore there exists a $r \times (n-r)$ matrix H such that

$$\begin{aligned} [v_{r+1}, v_{r+2}, \ldots, v_n] &= U_r H && (5.91)\\ &= A(I - K)I_r H, && (5.92) \end{aligned}$$

H could be full rank or rank deficient. Then the whole V is given by

$$\begin{aligned} V &= [u_1, u_2, \ldots, u_r, v_{r+1}, v_{r+2}, \ldots, v_n] && (5.93)\\ &= [U_r, U_r H] && (5.94)\\ &= U_r \begin{bmatrix} I_{r\times r} & H \end{bmatrix} && (5.95)\\ &= A(I - K) \begin{bmatrix} I_{r\times r} \\ 0 \end{bmatrix} \begin{bmatrix} I_{r\times r} & H \end{bmatrix} && (5.96)\\ &= A(I - K) \begin{bmatrix} I_{r\times r} & H \\ 0 & 0 \end{bmatrix}. && (5.97) \end{aligned}$$

Hence,

$$\begin{aligned} M &= (I-K)F^{-1}V^T && (5.98)\\ &= (I-K)F^{-1}\left(A(I-K)\begin{bmatrix} I_{r\times r} & H \\ 0 & 0 \end{bmatrix}\right)^T && (5.99)\\ &= (I-K)F^{-1}\begin{bmatrix} I_{r\times r} & 0 \\ H^T & 0 \end{bmatrix}(I-K)^T A^T. && (5.100) \end{aligned}$$

From the above equation, we can also see the difference between the full column rank case and the rank deficient case lies in

$$\begin{bmatrix} I_{r\times r} & 0 \\ H_{r\times n-r} & 0 \end{bmatrix}, \tag{5.101}$$

which should be an identity matrix when A is full column rank.

If there is no numerical dropping, M will be the Moore-Penrose inverse of A in the form of the following,

$$A^\dagger = (I-\tilde{K})\tilde{F}^{-1}\begin{bmatrix} I_{r\times r} & 0 \\ \tilde{H}^T & 0 \end{bmatrix}(I-\tilde{K})^T A^T. \tag{5.102}$$

Comparing Equation (5.100) and Equation (5.102), it is easy to see that $\mathcal{R}(M) = \mathcal{R}(A^\dagger)$. Note that $\mathcal{R}(A^\dagger) = \mathcal{R}(A^T)$, we can have the following theorem.

THEOREM 5.5.5 *Let $A \in \mathbb{R}^{m\times n}$, and* rank$(A) = r$. *If Assumption 5.5.1 holds for Algorithm 5.2.1. Then the following relationships hold, where M denotes the approximate*

Moore-Penrose inverse constructed by Algorithm 5.2.1

$$\mathcal{R}(M) = \mathcal{R}(A^{\dagger}) \tag{5.103}$$

$$= \mathcal{R}(A^{T}). \tag{5.104}$$

Based on Theorem 5.5.2 and Theorem 5.5.5, according to Theorem 3.3.7 we have the following theorem which ensures that the GMRES method can determine a solution to the preconditioned problem $MAx = Mb$ before breakdown happens for any $b \in \mathbb{R}^m$.

THEOREM 5.5.6 *Let $A \in \mathbb{R}^{m\times n}$, and* rank$(A) = r$*. Assume that all the linear independence is detected by Algorithm 5.2.1 and that the preconditioner M is computed using Algorithms 5.2.1. Then, for all $b \in \mathbb{R}^m$, preconditioned GMRES determines a least squares solution of*

$$\min_{x\in\mathbb{R}^n} \|MAx - Mb\|_2 \tag{5.105}$$

before breakdown and this solution attains $\min_{x\in\mathbb{R}^n}\|b - Ax\|_2$.

REMARK 11 *Using the result from Theorem 5.5.5, there exists a nonsingular matrix T, such that $A = M^T T$. Hence,*

$$\mathcal{R}(MA) = \mathcal{R}(MM^T T) \tag{5.106}$$

$$= \mathcal{R}(MM^T) \tag{5.107}$$

$$= \mathcal{R}(M). \tag{5.108}$$

Hence, no matter if the original problem consistent, the preconditioned problem (5.68) is always a consistent problem.

5.6 Implementation Consideration

5.6.1 Detect Linear Dependence

In Algorithm 5.2.1 and Algorithm 5.3.1, one important issue is how to judge the condition "if a_i is recognized as $\notin \mathcal{R}(A_{i-1})$". Simply speaking, we can set up a tolerance τ in advance, and switch to "else" when $\|u_i\|_2 < \tau$. However, is this good enough to help us detect the which column of A is linearly independent and which is linearly dependent when we perform numerical droppings? To address this issue, we first take a look at the RIF preconditioning algorithm.

The RIF preconditioner was developed for full rank matrices. However, numerical experiments showed it also works for rank deficient matrices. For this phenomenon, our similar Algorithm 5.4.1 can give a better insight into the RIF preconditioning algorithm. Since our Algorithm 5.4.1 and RIF are both based on the $\mathrm{A^TA}$-orthogonalization procedure. A breakdown of Algorithm 5.4.1 or RIF is actually a breakdown to the $\mathrm{A^TA}$-orthogonalization procedure. The only possibility for the $\mathrm{A^TA}$-orthogonalization procedure to breakdown is when the f_i in the denominator becomes zero, which implies that u_i is a zero vector. From our algorithm, we know that

$$\begin{aligned} u_i &= a_i - A_{i-1}k_i && (5.109)\\ &= a_i - A_{i-1}A_{i-1}^{\dagger}a_i && (5.110)\\ &= (I - A_{i-1}A_{i-1}^{\dagger})a_i. && (5.111) \end{aligned}$$

It is clear that u_i is the projection of a_i onto $\mathcal{R}(A_{i-1})^{\perp}$. Hence in exact arithmetic $u_i = 0$ if

and only if $a_i \in \mathcal{R}(A_{i-1})$. Our algorithm has an alternative when $a_i \in \mathcal{R}(A_{i-1})$ happens, i.e. when $u = 0$, our algorithm will turn into "else" case. However, this is not always necessary because of the numerical droppings. With numerical droppings, the u_i is actually,

$$\begin{aligned} u_i &= a_i - A_{i-1}k_i && (5.112)\\ &= a_i - A_{i-1}M_{i-1}a_i && (5.113)\\ &\approx a_i - A_{i-1}A_{i-1}^{\dagger}a_i && (5.114)\\ &\neq 0. && (5.115) \end{aligned}$$

Hence, even though $a_i \in \mathcal{R}(A_{i-1})$, since u_i will not be the exact projection of a_i onto $\mathcal{R}(A_{i-1})^{\perp}$, the RIF preconditioning algorithm will not necessarily breakdown when linear dependence happens.

The RIF algorithm does not take the rank deficient columns or nearly rank deficient columns into consideration. Hence, if we can capture the rank deficient columns, we might be able to have a better preconditioner. Assume the M we compute from any of our three algorithms can be viewed as an approximation to $A^{\dagger}$ with error matrix $E \in \mathbb{R}^{n\times m}$,

$$M = A^{\dagger} + E. \tag{5.116}$$

First note a theoretical result about the perturbation lower bound of the generalize inverse.

THEOREM 5.6.1 *[52] If $rank(A+E) \neq rank(A)$, then*

$$\|(A+E)^{\dagger} - A^{\dagger}\|_2 \geq \frac{1}{\|E\|_2}. \tag{5.117}$$

By Theorem 5.6.1, if the rank of $M = A^\dagger + E$ from our algorithm is not equal to the rank of $A^\dagger$, (or A, since they have the same rank), by the above theorem, we have,

$$\|M^\dagger - (A^\dagger)^\dagger\|_2 \geq \frac{1}{\|E\|_2} \tag{5.118}$$

$$\Rightarrow \|M^\dagger - A\|_2 \geq \frac{1}{\|E\|_2}. \tag{5.119}$$

The above inequality says that, if we denote $M^\dagger = A + \Delta A$, then $\|\Delta A\|_2 \geq \frac{1}{\|E\|_2}$. Hence, when $\|E\|_2$ is small, which means M is a good approximation to $A^\dagger$, M can be an exact generalized inverse of another matrix which is far from A, and the smaller the $\|E\|_2$ is, the further $M^\dagger$ from A is. In this sense, if the rank of M is not the same as that of A, M may not be a good preconditioner.

Thus, it is important to maintain the rank of M to be the same of $\mathrm{rank}(A)$. Hence, when we perform our algorithm, we need to sparsify the preconditioner M, but at the same time we also want to capture the rank deficient columns as many as possible, and maintain the rank of M. To achieve this, apparently, it is very import to decide how to judge when the exact value $u_i = \|(I - A_{i-1}A_{i-1}^\dagger)a_i\|_2$ is close to zero or not based on the computed value $\tilde{u}_i = \|(I - A_{i-1}M_{i-1})a_i\|_2$.

Taking a closer look at $\tilde{u}_i$, we have

$$\tilde{u}_i = a_i - A_{i-1}M_{i-1}a_i \tag{5.120}$$

$$= a_i - A_{i-1}(A_{i-1}^\dagger + E_1)a_i \tag{5.121}$$

$$= (a_i - A_{i-1}A_{i-1}^\dagger a_i) - A_{i-1}E_1 a_i \tag{5.122}$$

$$= u_i - A_{i-1}E_1 a_i. \tag{5.123}$$

When $a_i \in \mathcal{R}(A_{i-1})$, $u_i = a_i - A_{i-1}A_{i-1}^{\dagger}a_i = 0$. Then,

$$\tilde{u}_i = -A_{i-1}E_1 a_i \tag{5.124}$$

$$\|\tilde{u}_i\|_2 \leq \|A_{i-1}\|_F \|E_1\|_F \|a_i\|_2, \tag{5.125}$$

If we require E_1 to be small, we can use a tolerance τ_s. If

$$\|\tilde{u}_i\|_2 \leq \tau_s \|A_{i-1}\|_F \|a_i\|_2, \tag{5.126}$$

we suppose we detect a column a_i which is in the range space of A_{i-1}. From now on, we call τ_s the *switching tolerance*.

5.6.2 Right-Preconditioning Case

So far we assume $A \in \mathbb{R}^{m\times n}$, and discussed the left-preconditioning. When $m \geq n$, it is better to perform a left-preconditioning since the size of the preconditioned problem will be smaller. When $m \leq n$, a right-preconditioned problem can be described as follows,

$$\min_{y\in\mathbb{R}^m} \|b - ABy\|_2. \tag{5.127}$$

In this subsection we will show that all the results for left-preconditioning can be extended to the right-preconditioning case.

When $m \leq n$, Theorem 5.5.2 and Theorem 5.5.5 still hold, since in the proof of these two theorems we did not refer to the fact that $m \geq n$. Then, note Theorem 3.3.1, it is easy to obtain similar conclusions for right-preconditioning.

THEOREM 5.6.2 *Let $A \in \mathbb{R}^{m\times n}$. If Assumption 5.5.1 holds, then for any $b \in \mathbb{R}^m$ we*

have $\min_{x\in\mathbb{R}^n} \|b - Ax\|_2 = \min_{y\in\mathbb{R}^m} \|b - AMy\|_2$*, where* M *is a preconditioner constructed by Algorithm 5.2.1.*

PROOF. From Theorem 5.5.5, we know that $\mathcal{R}(M) = \mathcal{R}(A^T)$, which implies that there exists a nonsingular matrix $C \in \mathbb{R}^{m\times m}$ such that $M = A^T C$. Hence,

$$\begin{aligned} \mathcal{R}(AM) &= \mathcal{R}(AA^T C) && (5.128)\\ &= \mathcal{R}(AA^T) && (5.129)\\ &= \mathcal{R}(A). && (5.130) \end{aligned}$$

Using Theorem 3.3.1 we complete the proof. □

THEOREM 5.6.3 *Let* $A \in \mathbb{R}^{m\times n}$*. If Assumption 5.5.1 holds, then GMRES determines a solution to the right preconditioned problem*

$$\min_{y\in\mathbb{R}^m} \|b - AMy\|_2 \tag{5.131}$$

before breakdown happens.

PROOF. The proof is directly from Theorem 3.3.7. □

THEOREM 5.6.4 *Let* $A \in \mathbb{R}^{m\times n}$*. If Assumption 5.5.1 holds, then for any* $b \in \mathbb{R}^m$*, GMRES determines a least squares solution of*

$$\min_{y\in\mathbb{R}^m} \|b - AMy\|_2 \tag{5.132}$$

before breakdown and this solution attains $\min_{x\in\mathbb{R}^n} \|b - Ax\|_2$*, where* M *is computed by Algorithm 5.2.1.*

We would like to remark that it is preferable to perform Algorithm 5.2.1 and Algorithm 5.3.1 to A^T rather than A when $m < n$, based on the following three reasons. By doing so, we construct $\hat{M}$, an approximate generalized inverse of A^T. Then, we can use $\hat{M}^T$ as the preconditioner to the original least squares problem.

1. In Algorithm 5.2.1 and Algorithm 5.3.1, the approximate generalized inverse is constructed row by row. Hence, we perform a loop which goes through all the columns of A once. When $m \geq n$, this loop is relatively short. However, when $m < n$, this loop could become very long, and the preconditioning will be more time-consuming.

2. Another reason is that, linear dependence will always happen in this case even though matrix A is full row rank. If $m << n$, then when we perform the precondition algorithm on A, a lot of linear dependence needs to be detected. This fact makes it more difficult to capture the rank deficiency of A, and may result in a bad preconditioner.

3. Even though our algorithms can detect the linear dependence accurately, if we look at the algorithms, for a certain column a_i of A, it is more expensive to deal with than when a_i is independent of the previous columns.

5.7 Numerical Examples

In this section, we first use matrices from the Florida University Sparse Matrices Collection to test our algorithms, where zero rows are omitted and if the original matrix is under-determined we use its transpose. All computations were run on a Dell Precision 690, where the CPU is 3 GHz and the memory is 16 GB, and the programming language and compiling environment was GNU C/C++ 3.4.3 in Redhat Linux. Detailed information is given in the following table.

Table 5.1: Information on the matrix

Name	m	n	rank	density(%)	cond
80bau3b	11934	2262	2262	0.09	567.23
bnl2	4486	2324	2324	0.14	7.77×10^3
capri	482	271	271	1.45	8.09×10^3
d2q06c	5831	2171	2171	0.26	1.33×10^5
fit1p	1677	627	627	0.94	6.85×10^3
fit2p	13525	3000	3000	0.12	4.69×10^3
maros	1966	846	846	0.61	1.95×10^6
perold	1560	625	625	0.65	5.13×10^5
pilot	4860	1441	1441	0.63	2.66×10^3
pilot_we	2928	722	722	0.44	4.28×10^5
scfxm1	600	330	330	1.38	2.42×10^4
scfxm2	1200	660	660	0.69	2.42×10^4
scfxm3	1800	990	990	0.51	1.39×10^3
share1b	253	117	117	3.98	1.05×10^5
stocfor2	3045	2157	2157	0.14	2.84×10^4
vtp_base	346	198	198	1.53	3.55×10^7
beaflw	500	492	460	21.71	1.52×10^9
beause	505	492	459	17.93	6.74×10^8
Maragal_2	536	260	171	3.13	308.95
landmark	71952	2673	2671	0.60	1.02×10^8
lp_cycle	3371	1890	1875	0.33	1.46×10^7
Pd_rhs	5804	4371	4368	0.02	3.36×10^8

In Table 5.1, when the original matrix is under-determined, we used its transpose. Table 5.1 is divided into two parts. The matrices in the first half are full column rank matrices, the matrices in the second half are the rank deficient matrices. The condition number of A is given by $\frac{\sigma_1(A)}{\sigma_r(A)}$, where r is the rank. We construct the preconditioner M and perform the BA-GMRES [34] which is given below.

ALGORITHM 5.7.1 *BA- GMRES*

1. *Choose* x_0
2. $\tilde{r}_0 = B(b - Ax_0)$
3. $v_1 = \tilde{r}_0 / \|\tilde{r}_0\|_2$
4. *for* $i = 1, 2, \ldots, k$
5. $\quad w_i = BAv_i$

6. *for* $j = 1, 2, \ldots, i$
7. $h_{j,i} = (w_i, v_j)$
8. $w_i = w_i - h_{j,i} v_j$
9. *end for*
10. $h_{i+1,i} = \|w_i\|_2$
11. $v_{i+1} = w_i / h_{i+1,i}$
12. *Find* $y_i \in \mathbb{R}^i$ *which minimizes* $\|\tilde{r}_i\|_2 = \|\|\tilde{r}_0\|_2 e_i - \bar{H}_i y\|_2$
13. $x_i = x_0 + [v_1, \ldots, v_i] y_i$
14. $r_i = b - A x_i$
15. *if* $\|A^T r_i\|_2 < \varepsilon$ *stop*
16. *end for*
17. $x_0 = x_k$
18. *Go to* 2.

The BA-GMRES is a method that solving least squares problems with GMRES by preconditioning the original problem from the left with a suitable preconditioner B.

In the following example, the right hand side vectors b are random vectors generated by Matlab, so that all the least squares problems are inconsistent. In this section, we use

$$\|u_i\|_2 \leq \tau_s \|A_{i-1}\|_F \|a_i\|_2, \tag{5.133}$$

the criterion to judge if we need to switch to the "else" case. When the switching tolerance is zero, it implies that we are constructing RIF-like preconditioners. Our dropping rule is to drop the i-th element of k_j, i.e., $k_{j,i}$ when the following inequality holds.

$$|k_{j,i}| \|a_i\|_2 < \tau_d, \tag{5.134}$$

where τ_d is the dropping tolerance. The stopping rule for GMRES is

$$\|A^T(b - Ax)\|_2 \leq 10^{-8} \cdot \|A^T b\|_2. \tag{5.135}$$

First we show how the switching tolerance τ_s works. Take matrix lp_cycle for an example, we know that the rank deficient columns are,

$$\begin{array}{ccccc} 182 & 184 & 216 & 237 & 253 \\ 717 & 754 & 961 & 1221 & 1239 \\ 1260 & 1261 & 1278 & 1640 & 1859, \end{array} \tag{5.136}$$

15 columns in all. As we know that the rank deficient columns are not unique, the above columns we list are the columns which are linearly dependent on their previous columns. In the following example, we can see that our preconditioning algorithm can detect most of them precisely.

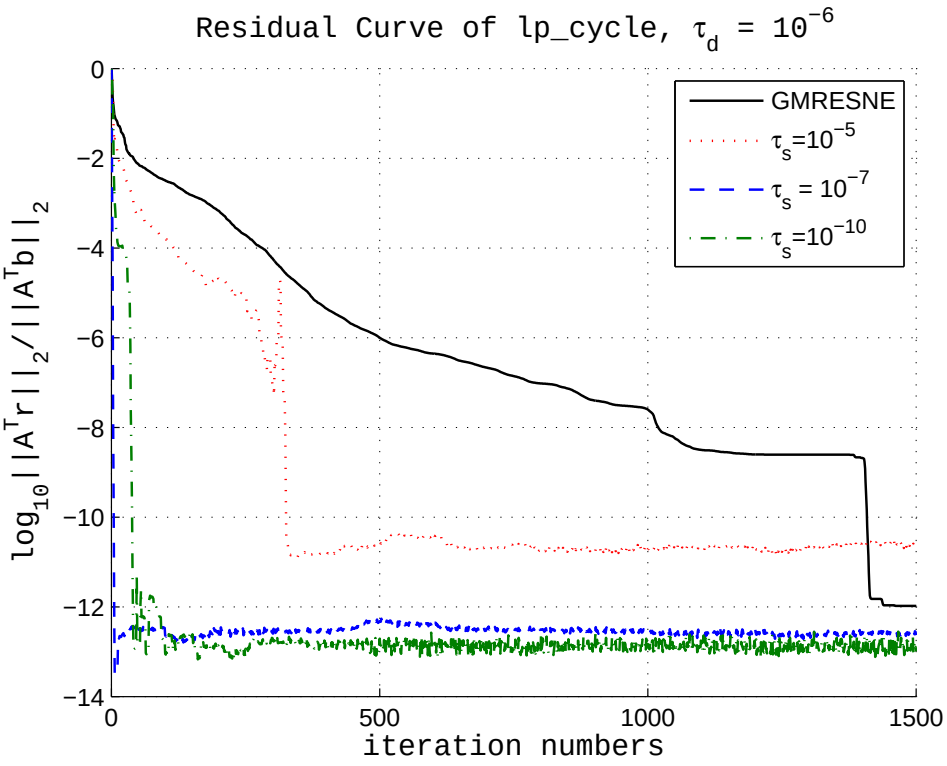
Table 5.2: Numerical Results

τ_d	τ_s	deficiency detected	ITS	Pre. T	Its. T	Tot. T
1.e-6	1.e-10	$-1239, -1261, -1278$	37	3.33	0.99	4.32
1.e-6	1.e-7	detect all exactly	4	3.5	0.10	3.6
1.e-6	1.e-5	detected 92 col	324	7.45	12.14	19.59

In the above table, we fix the dropping tolernace τ_d to 10^{-6} and test different switching tolerances. The column "deficiency detected" gives the linearly dependent columns detected by our algorithm. For example -1239 means the 1239th column, which is a rank deficient column is missed by our algorithm. Form the table we can see, when $\tau_s = 10^{-10}$, our preconditioning algorithm detected 12 rank deficient columns except $1239, 1261, 1278$ and did not detect wrong rank deficient columns. When $\tau_s = 10^{-7}$, our preconditioning algorithm found exactly 15 rank deficient columns, and all of them are correct. When $\tau_s = 10^{-5}$, 92 columns are recognized as linearly dependent columns, obviously many linearly independent columns are recognized as linearly dependent columns by mistake. For other columns, "ITS" means iteration numbers, "Pre. T" means preconditioning time, "Its.

T" means iteration time, and "Tot. T" means total CPU time. From this table we can see, when then rank deficiency is detected correctly, the convergence can be accelerated. When wrong rank deficiency is detected, the convergence can be slowed down.

Figure 5.1: Convergence Curve for lp_cycle



In Figure 5.1, We can see that, when $\tau_s = 10^{-10}$ or $\tau_s = 10^{-7}$, no wrong linearly dependent columns are detected, hence, Assumption 5.5.1 is satisfied. By Theorem 5.5.6, the preconditioned problem is equivalent to the original problem, and GMRES can solve a solution to the preconditioned problem. In Figure 5.1, the residual curves for $\tau_s = 10^{-10}$ and $\tau_s = 10^{-7}$ decrease to 10^{-12}, which shows that the solution to the preconditioned problem is the solution to the original least squares problem. When $\tau_s = 10^{-5}$, too many rank deficient columns are detected, Assumption 5.5.1 is not satisfied, hence, the residual curve for $\tau_s = 10^{-5}$ only deceases to 10^{-10} level and maintains. This phenomenon shows

that when $\tau_s = 10^{-5}$, the preconditioned problem is not equivalent to the original problem. To sum up, Figure 5.1 illustrates that Assumption 5.5.1 is necessary to obtain a solution to the original problem, however, when the assumption is not satisfied, a good enough approximate solution may still be achieved.

Table 5.3: Numerical results of full rank matrices

matrix	τ_d	Detected	ITS	Pre. T	Its. T	Tot. T	GMRESNE	D-CGLS
80bau3b	0.1	0	17	1.58	0.02	1.60	82, 0.31	56, *0.09
bnl2	$6.e-2$	0	64	0.84	0.42	1.26	651, 16.64	657, *0.64
d2q06c	0.002	0	32	3.63	0.76	4.39	858, 27.03	2743, *4.05
capri	0.01	0	17	0.02	0.01	*0.03	215, 0.22	496, 0.06
fit1p	0.0001	0	7	0.92	0.03	0.95	31, *0.02	5986, 2.54
fit2p	$1.e-5$	0	5	89.81	0.9	90.71	31, *0.08	100000+, 251.2
maros	$1.e-6$	0	3	1.22	0.02	*1.24	448, 2.85	13714, 6.71
perold	0.01	0	17	0.17	0.04	*0.21	400, 1.68	1058, 0.36
pilot	0.05	0	63	1.49	0.44	1.93	385, 3.81	822, 1.24
pilot_we	0.1	0	41	0.17	0.09	*0.26	429, 2.24	736, 0.38
scfxm1	0.001	0	20	0.04	0.02	*0.06	184, 0.20	761, 0.11
scfxm2	$5.e-5$	0	5	0.18	0.02	*0.20	283, 0.89	1332, 0.42
scfxm3	$5.e-5$	0	6	0.34	0.03	*0.37	346, 1.99	1583, 0.72
stocfor2	$5.e-5$	0	25	8.30	0.66	8.66	330, 3.69	2147, 1.59
share1b	0.001	0	6	0.00	0.00	*0.00	117, 0.04	547, 0.03
vtp_base	$2.e-6$	0	5	0.03	0.00	*0.03	119, 0.05	3667, 0.29

In Table 5.3 we list our numerical results of full rank problems. We simply set τ_s to 0.0 so that we did not detect any linearly dependent columns. From Table 5.3 we conclude that our preconditioner performs competitively when matrices are ill-conditioned. In Table 5.4, we list the results of rank deficient problems. We compare GMRES method with our preconditioner to GMRES with RIF preconditoiner, which is listed in column "RIF-GMRES".

Table 5.4: Numerical results of rank deficient matrices

matrix	τ_d	τ_s	Detected	ITS	Pre. T	Its. T	Tot. T	RIF-GMRES	GMRESNE	D-CGLS
beaflw	1.e-5	1.e-10	28	66	0.96	0.24	*1.20	†	450, 2.00	†
beause	1.e-6	1.e-10	35	22	0.90	0.08	*0.98	†	447, 1.92	†
landmark	1.e-1	1.e-8	2	100	34.5	7.42	41.92	239, 38.43 (10.0)	†	252, *8.77
lp_cycle	1.e-4	1.e-6	17	43	2.24	0.84	*3.08	†	1020, 32.85	6799, 6.74
Maragal_2	1.e-4	1.e-3	38	36	0.11	0.04	*0.15	†	169, *0.15	11224, 1.94
Pd_rhs	1.e-4	1.e-8	3	4	0.93	0.00	0.93	†	780, 44.47	242, *0.29

†: GMRES did not converge in 2500 steps or D-CGLS did not converge in 100000 steps.

From Table 5.4 we see that our preconditioning algorithm detected some linearly dependent columns so that achieved faster convergence. On the other hand, RIF preconditioning algorithm broke down at the preconditioning stage or RIF-preconditioned GMERS converges slowly.

From Table 5.3 and 5.4 we conclude that, our preconditioner performs competitively for ill-conditioned problem, and more robust with rank deficient problems.

Chapter 6

Applications to Linear Programming Problems

A linear programming problem involves the optimization of a linear function subject to linear constraints on the variables. Although linear functions are simple functions, they arise frequently in economics, production planning, networks, scheduling and other applications. The simplex method is the most widely used method for linear programming. It was developed in the 1940's at the same time as linear programming models was introduced. Due to its efficiency, the simplex method has no competitors until recent years. With the development of interior-point methods, the simplex method has had a serious challenge.

Interior-point methods are the most significant development in linear optimization since the development of the simplex method. The methods have good theoretical efficiency and good practical performance. Interior-point methods require to solve a least squares problem in each step. The closer the iteration solutions get to the optimal solution, the more ill-conditioned the least squares problems become. Direct methods are usually used to

solve these least squares problems. In this thesis, we solve the least squares problem by the Krylov subspace methods with our preconditioner.

In this chapter, we first give an introduction to the basics of linear programming and the interior-point methods. For details, we refer to [32, 54]. And we also present some numerical results to show that our preconditioners are also suitable for this kind of problems.

6.1 Linear programming

There are many different ways to represent a linear programming problem. Sometimes one form is more convenient than the other. The standard way to describe a linear programming problem is

$$\begin{aligned} \min \quad & C^T x \\ s.t. \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{6.1}$$

where A is an $m \times n$ matrix called *constraint matrix*, vectors x and c are both n dimensional vectors, $b \in \mathbb{R}^m$. if x satisfies the constraints $Ax = b$, $x \geq 0$, we call it a feasible point. the set of all feasible points is the feasible set. All the linear programming problems can be written as standard form. The fundamental properties of a linear programming problem are

- a vector of real variables, whose optimal values are found by solving the problem;
- a linear objective function;
- linear constraints, both inequalities and equalities.

Associated with any linear program is another linear program called the *dual*, which consists of the same data objects arranged in a different way. The dual for (6.1) is

$$\begin{array}{ll} \max & b^T\lambda \\ s.t. & A^T\lambda + s = c \\ & s \geq 0, \end{array} \tag{6.2}$$

where λ is a vector in $\mathbb{R}^m$ and s is a vector in $\mathbb{R}^n$. We call components of λ the dual variables, while s is the vector of dual slacks.

The linear programming problem (6.1) is called the *primal*, to distinguish it from problem (6.2). The two problems together are referred to as the *primal-dual pair*. Define the primal-dual *feasible set* $\mathcal{F}$ and *strictly feasible set* $\mathcal{F}^o$,

$$\mathcal{F} = \{(x, \lambda, s) | Ax = b, A^T\lambda + s = c, (x, s) \geq 0\} \tag{6.3}$$

$$\mathcal{F}^o = \{(x, \lambda, s) | Ax = b, A^T\lambda + s = c, (x, s) > 0\}. \tag{6.4}$$

Elements in $\mathcal{F}$ are called feasible solutions, and the ones in $\mathcal{F}^o$ are called strictly feasible solutions. Assume $(\bar{x}, \bar{\lambda}, \bar{s})$ is a feasible solution, hence, $\bar{x}$, $\bar{\lambda}$ and $\bar{s}$ satisfy the following equations, and inequalities,

$$A\bar{x} = b \tag{6.5}$$

$$A^T\bar{\lambda} + \bar{s} = c \tag{6.6}$$

$$\bar{x} \geq 0, \quad \bar{s} \geq 0. \tag{6.7}$$

From the above relations, we can easily see that for any feasible solution we have $c^T\bar{x} \geq b^T\bar{\lambda}$.

$$\begin{aligned} c^T\bar{x} &= (A^T\bar{\lambda} + \bar{s})^T\bar{x} && (6.8)\\ &= \bar{\lambda}^T A\bar{x} + \bar{s}^T\bar{x} && (6.9)\\ &= \bar{\lambda}^T b + \bar{s}^T\bar{x} && (6.10)\\ &\geq \bar{\lambda}^T b. && (6.11) \end{aligned}$$

We call

$$c^T\bar{x} - \bar{\lambda}^T b = \bar{s}^T\bar{x} \tag{6.12}$$

the *duality gap*. The idea of the primal-dual method is to find a sequence of strictly feasible solutions $(\bar{x}_k, \bar{\lambda}_k, \bar{s}_k) \in \mathcal{F}^o$ so that the duality gap decreases gradually. Once the duality gap reaches zero, then the solution would be optimal. Note that when the duality gap is zero when the strictly feasible solution is $(\bar{x}, \bar{\lambda}, \bar{s})$, then $\bar{s}^T\bar{x}$ is zero too. Combing the fact that a feasible solution is optimal to the primal problem (6.1) and the dual problem (6.2) if and only if it satisfies the complementary slackness conditions

$$x_j s_j = 0, \quad j = 1, \ldots, n, \tag{6.13}$$

we obtain the following conclusion.

LEMMA 6.1.1 *The vector pair (x^*, λ^*, s^*) is the optimal solution to problem (6.2) and*

problem (6.1) if and only if it satisfies the following conditions,

$$\begin{aligned} A^T\lambda + s &= c \\ Ax &= b \\ x_i s_i &= 0, \quad i = 1, 2, \ldots, n \\ x \geq 0, \;\; s &\geq 0\,. \end{aligned} \tag{6.14}$$

The Lemma 6.1.1 can also be concluded from KKT conditions [54], hence, here we call the above conditions *KKT Conditions*.

The idea of the primal-dual method is to move through a sequence of strictly feasible primal and dual solutions that go closer and closer to zero. Specifically, at each iteration we attempt to find vectors $x(\mu), \lambda(\mu), s(\mu)$ satisfying, for some $\mu > 0$,

$$\begin{aligned} A^T\lambda + s &= c \\ Ax &= b \\ x_i s_i &= \mu, \quad i = 1, 2, \ldots, n \\ x \geq 0, \;\; s &\geq 0\,. \end{aligned} \tag{6.15}$$

The difference from KKT conditions is the third equation. Instead $x_i s_i = 0,\ i = 1, 2, \ldots, n$, we have $x_i s_i = \mu,\ i = 1, 2, \ldots, n$, where μ is a positive number and is reduced step by step until convergence is achieved. We choose μ to be a sequence of positive number so that we can guarantee that $x > 0$ and $s > 0$, i.e.,(x, λ, s) is a strictly feasible solution. In this case, the duality gap is

$$x^T s = n\mu. \tag{6.16}$$

Hence, when we reduce μ to zero gradually, we also reduce the duality gap, when the duality gap is small enough, we say we find an optimal solution within some tolerance. Let

$X = \mathrm{diag}(x)$, $S = \mathrm{diag}(s)$, and vector $e = [1, 1, \ldots, 1]^T$ be a n dimensional vector, the third equation can be simplified as

$$XSe = \mu e. \tag{6.17}$$

From the above discussion, it is easy to see that in the primal-dual algorithm, we need to solve equations

$$\begin{array}{rcl} A^T\lambda + s & = & c \\ Ax & = & b \\ XSe & = & \mu e \\ x \geq 0, \;\; s & \geq & 0\,. \end{array} \tag{6.18}$$

in every step. And solving these equations is the main cost of the primal-dual algorithm.

Note that the third in Equations (6.18) is nonlinear. We use Newton's Method to solve it. Assume we have a solution (x, λ, s) satisfying Equations (6.18), and then we reduce μ to $\bar{\mu} < \mu$, we need to find a new solution $(x + \Delta x, \lambda + \Delta\lambda, s + \Delta s)$ which satisfies the Equations (6.18) with $\bar{\mu}$. Hence, we want to solve

$$\begin{array}{rcl} A^T(\lambda + \Delta\lambda) + (s + \Delta s) & = & c \\ A(x + \Delta x) & = & b \\ (X + \Delta X)(S + \Delta S)e & = & \bar{\mu} e \\ (x + \Delta x) \geq 0, \;\; (s + \Delta s) & \geq & 0\,, \end{array} \tag{6.19}$$

which is equivalent to

$$\begin{aligned} A^T\Delta\lambda + \Delta s &= 0 \\ A\Delta x &= 0 \\ (X+\Delta X)(S+\Delta S)e &= \bar{\mu}e \\ \Delta x \geq 0, \;\; \Delta s &\geq 0\,, \end{aligned} \tag{6.20}$$

where $\Delta X = \mathrm{diag}(\Delta x)$, $\Delta S = \mathrm{diag}(s)$. The third equation $(X+\Delta X)(S+\Delta S)e = \bar{\mu}e$ can be written as

$$X\Delta s + S\Delta x + \Delta X\Delta S e = \bar{\mu}e - XSe. \tag{6.21}$$

When Δx and Δs are small, $\Delta X\Delta S e$ is much smaller, for this reason, we can omit this second order term. To sum up, to obtain the new solution $(x+\Delta x, \lambda+\Delta\lambda, s+\Delta s)$, we need to solve the following linear system,

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -XSe + \bar{\mu}e \end{bmatrix}. \tag{6.22}$$

Once the above linear system is solved, we set $(x+\Delta x, \lambda+\Delta\lambda, s+\Delta s)$ as the new solution and reduce $\bar{\mu}$ to look for the next solution.

However, in practical the new solution may fail to satisfy the conditions $x+\Delta x > 0$, $s+\Delta s$. To ensure every solution is a strictly feasible solution, we choose a scalar α and

set the new solution as

$$x(\alpha, \mu) = x + \alpha \Delta x \tag{6.23}$$

$$\lambda(\alpha, \mu) = \lambda + \alpha \Delta \lambda \tag{6.24}$$

$$s(\alpha, \mu) = s + \alpha \Delta s, \tag{6.25}$$

where α is chosen to ensure that $x(\alpha, \mu) > 0$ and $s(\alpha, \mu) > 0$. In summary, we can obtain the following algorithm.

ALGORITHM 6.1.1 *Primal-Dual Framework [54]*

1. *Given* $(x_0, \lambda_0, s_0) \in \mathcal{F}^o$.
2. *for* $k = 1, \ldots$ *until convergence*
3. *solve*
$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta \lambda^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma^k \mu^k e \end{bmatrix}, \tag{6.26}$$
where $\sigma^k \in [0, 1]$ *and* $\mu^k = (x^k)^T s^k / n$.
4. *set* $(x^{k+1}, \lambda^{k+1}, s^{k+1}) \leftarrow (x^k, \lambda^k, s^k) + \alpha_k (\Delta x^k, \Delta \lambda^k, \Delta s^k)$, *where* α_k *is chosen so that* $(x^{k+1}, s^{k+1}) > 0$. $\mu^{k+1} = (s^{k+1})^T x^{k+1}$.
5. *end for*

In Algorithm 6.1.1, we assume that we have a strictly feasible solution in the beginning. However, in practice, this is not very common. If we have a solution (x, λ, s), where $x > 0$ and $s > 0$, but one of

$$Ax = b \quad \text{and} \quad A^T \lambda + s = c \tag{6.27}$$

or both are not satisfied, Algorithm 6.1.1 is not suitable. In this case, we want to find a new solution satisfying

$$\begin{aligned} A^T(\lambda + \Delta\lambda) + (s + \Delta s) &= c \\ A(x + \Delta x) &= b \\ (X + \Delta X)(S + \Delta S)e &= \bar{\mu}e \\ (x + \Delta x) \geq 0, \ \ (s + \Delta s) &\geq 0\ . \end{aligned} \tag{6.28}$$

Move $A^T\lambda + s$ and Ax to the right hand side of the equations, and omit term $\Delta X \Delta S e$, we obtain an equivalent linear system

$$\begin{aligned} A^T\Delta\lambda + \Delta s &= r_D \\ A\Delta x &= r_P \\ X\Delta s + S\Delta x &= \bar{\mu}e - XSe \\ (x + \Delta x) \geq 0, \ \ (s + \Delta s) &\geq 0\ , \end{aligned} \tag{6.29}$$

where $r_D = c - A^T\lambda$ is the residual of the dual problem and $r_P = b - Ax$ is the residual of the primal problem. To sum up, for the case that an feasible solution is not available, we have the following algorithm.

ALGORITHM 6.1.2 *Primal-Dual Framework [54]*

1. *Given* (x_0, λ_0, s_0).
2. *for* $k = 1, \ldots$ *until convergence*
3. *solve*

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta\lambda^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} r_D \\ r_P \\ -X^kS^ke + \sigma^k\mu^ke \end{bmatrix}, \tag{6.30}$$

where $\sigma^k \in [0, 1]$ *and* $\mu^k = (x^k)^T s^k/n$.

4. *set* $(x^{k+1}, \lambda^{k+1}, s^{k+1}) \leftarrow (x^k, \lambda^k, s^k) + \alpha_k(\Delta x^k, \Delta\lambda^k, \Delta s^k)$*, where* α_k *is chosen so that* $(x^{k+1}, s^{k+1}) > 0$*.* $\mu^{k+1} = (s^{k+1})^T x^{k+1}$.

5. *end for*

6.2 Application of the Greville's method to linear programming problems

6.2.1 When a feasible initial solution is available

In this section, we introduce how to apply our preconditioners to linear programming. First we consider the case that we have a strictly feasible initial solution. In every step of Algorithm 6.1.1, we solve

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta\lambda^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -X^k S^k e + \sigma^k \mu^k e \end{bmatrix}, \tag{6.31}$$

where $\sigma^k \in [0,1]$ and $\mu^k = (x^k)^T s^k/n$. Since in the algorithm we ensure (x^k, λ^k, s^k) is a strictly feasible solution, S^{-1} and X^{-1} exist. Denote $S^{-1/2}X^{1/2}$ by D.

$$
\begin{aligned}
X\Delta s + S\Delta x &= \sigma^k \mu^k e - XSe && (6.32)\\
S^{-1}X\Delta s + \Delta x &= \sigma^k \mu^k S^{-1} e - S^{-1}XSe && (6.33)\\
AD^2\Delta s + A\Delta x &= \sigma^k \mu^k AS^{-1} e - AD^2 Se && (6.34)\\
AD^2\Delta s &= \sigma^k \mu^k AS^{-1} e - AD^2 Se && (6.35)\\
AD^2(-A^T\Delta\lambda^k) &= \sigma^k \mu^k AS^{-1} e - AD^2 Se && (6.36)\\
AD^2A^T\Delta\lambda^k &= AD^2Se - \sigma^k\mu^k AD^2X^{-1}e && (6.37)\\
(AD)(AD)^T\Delta\lambda^k &= ADD(S - \sigma^k\mu^k X^{-1})e. && (6.38)
\end{aligned}
$$

If we can solve $\Delta\lambda^k$ from Equation (6.38), we can compute the other two vectors Δx^k and Δs^k as follows,

$$
\begin{aligned}
\Delta s^k &= -A^T\Delta\lambda && (6.39)\\
\Delta x^k &= -X(S^k)^{-1}\Delta s^k - X^k e + \sigma^k\mu^k(S^k)^{-1}e. && (6.40)
\end{aligned}
$$

For Equation (6.38), if we define

$$
\begin{aligned}
C &= (AD)^T && (6.41)\\
f &= D(S - \sigma^k\mu^k X^{-1})e, && (6.42)
\end{aligned}
$$

Equation (6.38) can be written as a normal equation form

$$C^TC\Delta\lambda^k = C^Tf. \qquad (6.43)$$

Note that the above normal equation is equivalent to the least squares problem

$$\min_{\Delta\lambda^k \in \mathbb{R}^m} \|C\Delta\lambda^k - f\|_2, \tag{6.44}$$

hence, we can apply our preconditions from Chapter 5 to the above least squares problem.

6.2.2 When a feasible initial solution is not available

In the Subsection 6.2.1 we introduced how to apply preconditioners to the interior-point method when a feasible initial solution is available. However, there are many case that a feasible initial solution is not available. In this case, from the discussion of Section 6.1, we need to solve

$$\begin{bmatrix} 0 & A^T & I \\ A & 0 & 0 \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta\lambda^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} r_D^k \\ r_P^k \\ -X^k S^k e + \sigma^k \mu^k e \end{bmatrix} \tag{6.45}$$

in every step of Algorithm 6.1.2. In the algorithm we keep x^k and s^k positive so that $D = S^{-1/2}X^{1/2}$ is well defined.

$$\begin{aligned}
X^k\Delta s + S^k\Delta x &= \mu^k e - X^k S^k e && (6.46)\\
(S^k)^{-1}X^k\Delta s + \Delta x &= \sigma^k\mu^k(S^k)^{-1}e - (S^k)^{-1}X^k S^k e && (6.47)\\
A(D^k)^2\Delta s + A\Delta x &= \sigma^k\mu^k A(S^k)^{-1}e - AD^2 S^k e && (6.48)\\
A(D^k)^2\Delta s + r_P^k &= \sigma^k\mu^k A(S^k)^{-1}e - A(D^k)^2 S^k e && (6.49)\\
A(D^k)^2(r_D^k - A^T\Delta\lambda^k) + r_P^k &= \sigma^k\mu^k AS^{-1}e - A(D^k)^2 S^k e && (6.50)\\
(AD^k)(AD^k)^T\Delta\lambda^k &= r_P^k + AD^kD^k(r_D^k + S^k e - \sigma^k\mu^k(X^k)^{-1}e). && (6.51)
\end{aligned}$$

After solve $\Delta\lambda^k$ from Equation (6.51), we can compute the other two vectors Δx^k and Δs^k as follows,

$$\Delta s^k = r_D^k - A^T\Delta\lambda \tag{6.52}$$

$$\Delta x^k = -X^k(S^k)^{-1}\Delta s^k - X^k e + \sigma^k\mu^k(S^k)^{-1}e. \tag{6.53}$$

For Equation (6.51), if we define

$$C = (A(D^k))^T \tag{6.54}$$

$$f = (D^k)((S^k) - \sigma^k\mu^k(X^k)^{-1})e, \tag{6.55}$$

Equation (6.38) can be written as a normal equation form

$$C^TC\Delta\lambda^k = r_D^k + C^Tf. \tag{6.56}$$

Hence, the above equation is not in a normal equation form because of r_D^k is generally nonzero, which means that we cannot apply our preconditioners directly to the corresponding least squares problem.

We can overcome this difficulty by reformulating the linear programming problem in a special way. The following method is described in [39]. Consider a linear programming problem in the standard form, its primal is (6.1), where A is an $m \times n$ matrix, and its dual is (6.2). We introduce another linear programming problem by augmenting this linear

programming problem. We first define some new quantities.

$$
\begin{aligned}
L &= \lceil \log \left(\begin{array}{c} \text{largest absolute value of the determinant} \\ \text{of any square submatrix of } A \end{array} + 1 \right) \rceil \\
&\quad + \lceil \log(1 + \max_j |c_j|) \rceil + \lceil \log(1 + \max_i |b_i|) \rceil + \lceil \log(m+n) \rceil \qquad (6.57) \\
\alpha &= 2^{4L} \qquad (6.58) \\
\beta &= 2^{2L} \qquad (6.59) \\
\tilde{m} &= m + 1 \qquad (6.60) \\
\tilde{n} &= n + 2 \qquad (6.61) \\
K_b &= \alpha\beta(n+1) - \beta c^T e \qquad (6.62) \\
K_c &= \alpha\beta, \qquad (6.63)
\end{aligned}
$$

where L is called the size of problem (6.1), $\lceil \dots \rceil$ is defined as, for a scalar $w \in \mathbb{R}$

$$\lceil w \rceil = \min\{z \in \mathbb{Z} | z \geq w\}. \qquad (6.64)$$

From the definition of L, it is easy to see that $L \geq 4$. Then the augmented problem can be stated as follows,

$$
\begin{aligned}
\min \quad & c^T x + K_c x_{\tilde{n}} \\
s.t. \quad & Ax + (b - \beta Ae) x_{\tilde{n}} = b \\
& (\alpha e - c)^T x + \alpha x_{\tilde{n}-1} = K_b \\
& x \geq 0, \quad x_{\tilde{n}-1} \geq 0, \quad x_{\tilde{n}} \geq 0,
\end{aligned}
\qquad (6.65)
$$

where $x_{\tilde{n}-1}$ and $x_{\tilde{n}}$ are scalars. The dual of the problem (6.65) is then given as follows.

$$\begin{aligned} \max \quad & b^T\lambda + K_b\lambda_{\tilde{m}} \\ s.t. \quad & A^T y + (\alpha e - c)\lambda_{\tilde{m}} + s = c \\ & \alpha\lambda_{\tilde{m}} + s_{\tilde{n}-1} = 0 \\ & (b - \beta Ae)^T\lambda + s_{\tilde{n}} = K_c \\ & s \geq 0, \quad s_{\tilde{n}-1} \geq 0, \quad s_{\tilde{n}} \geq 0, \end{aligned} \tag{6.66}$$

where $\lambda_{\tilde{m}}$, $s_{\tilde{n}-1}$ and $s_{\tilde{n}}$ are scalars. Define $\tilde{x} \in \mathbb{R}^{\tilde{n}}$, $\tilde{\lambda} \in \mathbb{R}^{\tilde{m}}$, $\tilde{s} \in \mathbb{R}^{\tilde{n}}$, $\tilde{b} \in \mathbb{R}^{\tilde{m}}$, $\tilde{c} \in \mathbb{R}^{\tilde{n}}$ and $\tilde{A} \in \mathbb{R}^{\tilde{m}\times\tilde{n}}$ as follows, We define

$$\tilde{x} = \begin{bmatrix} x \\ x_{\tilde{n}-1} \\ x_{\tilde{n}} \end{bmatrix}, \qquad \tilde{\lambda} = \begin{bmatrix} \lambda \\ \lambda_{\tilde{m}} \end{bmatrix} \tag{6.67}$$

$$\tilde{b} = \begin{bmatrix} b \\ K_b \end{bmatrix}, \qquad \tilde{c} = \begin{bmatrix} c \\ 0 \\ K_c \end{bmatrix} \tag{6.68}$$

$$\tilde{A} = \begin{bmatrix} A & 0 & b - \beta Ae \\ (\alpha e - c)^T & \alpha & 0 \end{bmatrix}. \tag{6.69}$$

With these notations, we can reform the linear programming problem to obtain the augmented problem. The primal problem (6.1) is rewritten as

$$\begin{aligned} \min \quad & \tilde{c}^T\tilde{x} \\ s.t. \quad & \tilde{A}\tilde{x} = \tilde{b} \\ & \tilde{x} \geq 0. \end{aligned} \tag{6.70}$$

The dual problem (6.2) is rewritten as

$$\begin{aligned} \max \quad & \tilde{b}^T\tilde{\lambda} \\ s,t. \quad & \tilde{A}^T\tilde{\lambda} + \tilde{s} = \tilde{c} \\ & \tilde{s} \geq 0. \end{aligned} \tag{6.71}$$

For this augmented linear programming problem, a feasible solution always exists. Consider the point $(\tilde{x}_0,\ \tilde{\lambda}_0,\ \tilde{s}_0)$ which is defined as follows.

$$\tilde{x}_0 = [\beta, \beta, \ldots, \beta, 1]^T \in \mathbb{R}^{\tilde{n}} \tag{6.72}$$

$$\tilde{\lambda}_0 = [0, 0, \ldots, 0, -1]^T \in \mathbb{R}^{\tilde{m}} \tag{6.73}$$

$$\tilde{s}_0 = [\alpha, \alpha, \ldots, \alpha, \alpha\beta]^T \in \mathbb{R}^{\tilde{n}}. \tag{6.74}$$

It is easy to verify the point $(\tilde{x}_0,\ \tilde{\lambda}_0,\ \tilde{s}_0)$ satisfies

$$\tilde{A}\tilde{x}_0 = \tilde{b} \tag{6.75}$$

$$\tilde{A}^T\tilde{\lambda}_0 + \tilde{s}_0 = \tilde{c} \tag{6.76}$$

$$\tilde{x}_0 \geq 0 \tag{6.77}$$

$$\tilde{s}_0 \geq 0, \tag{6.78}$$

which implies that $(\tilde{x}_0,\ \tilde{\lambda}_0,\ \tilde{s}_0)$ is a feasible solution to the augmented linear programming problem. Then, we can use Algorithm 6.1.1 to solve the augmented linear programming problem.

Since L is uncomputable, we have to estimate the size of L according to the problems. If we use iterative methods solve the least squares problems rising in each interior point

method step, the size of the L can have big influence on the convergence of the iterative methods we use. Consider the size of L. From the definition of L we can see that $L \geq 3 + \lceil \log(m+n) \rceil$. If we take L as 5,

$$\alpha = 2^{4L} = 2^{20} = (2^{10})^2 \approx 1000^2 = 10^6 \tag{6.79}$$

$$\beta = 2^{2L} \approx= 1000. \tag{6.80}$$

α and β can be much larger when a larger L is taken.

6.3 Numerical Examples

Our test matrices are provided by Prof. Takashi Tsuchiya and these matrices can also be found in [25]. All computations were run on a Dell Precision 690, where the CPU is 3 GHz and the memory is 16 GB, and the programming language and compiling environment was GNU C/C++ 3.4.3 in Redhat Linux.

In the following tables, we use Interior Point Method to solve the augmented linear programming problems. We use different methods to solve the least squares problems arising in the Interior Point Method. In the tables,

- DCGLS stands for solving the least squares problems by CGLS with diagonal scaling.
- DGMRESNE stands for solving the least squares problem by using GMRES with diagonal scaling to solve the corresponding normal equations.
- GreGMRES stands for solving the least squares problems by using GMRES with

Greville's method as preconditioning algorithm.

- Cholesky stands for solving the least squares problem by forming the normal equation explicitly and using the Cholesky decomposition with the approximate minimal degree ordering [1].

In the Cholesky approach, at the kth step of the interior-point method, to solve $\Delta\tilde{\lambda}^{(k)}$ from

$$\tilde{A}\tilde{D}_k^2\tilde{A}^T\Delta\tilde{\lambda}^{(k)} = \tilde{A}\tilde{D}_k^2\tilde{S}^{(k)}e - \sigma^k\mu^k\tilde{A}\tilde{D}_k^2(\tilde{X}^{(k)})^{-1}e, \tag{6.81}$$

the Cholesky decomposition of $\tilde{A}\tilde{D}_k^2\tilde{A}^T$ has to be computed, where

$$\tilde{A} = \begin{bmatrix} A & 0 & b-\beta Ae \\ (\alpha e - c)^T & \alpha & 0 \end{bmatrix}, \tag{6.82}$$

$\tilde{D}_k^2 = \tilde{S}_k^{-1}\tilde{X}_k = \mathrm{diag}\{(\tilde{s}_1^{(k)})^{-1},\ldots,(\tilde{s}_n^{(k)})^{-1}\}\mathrm{diag}\{\tilde{x}_1^{(k)},\ldots,\tilde{x}_n^{(k)}\}$. To perform the Cholesky decomposition, $\tilde{A}\tilde{D}_k^2\tilde{A}^T$ has to be formed explicitly. However, even though A is a sparse matrix, $\tilde{A}\tilde{D}_k^2\tilde{A}^T$ can be dense. Consider matrix $\tilde{A}$ in column form,

$$\tilde{A} = [\tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_n], \tag{6.83}$$

then,

$$\tilde{A}\tilde{D}_k^2\tilde{A}^T = [\tilde{a}_1,\tilde{a}_2,\ldots,\tilde{a}_n]\tilde{D}_k^2[\tilde{a}_1,\tilde{a}_2,\ldots,\tilde{a}_n]^T \tag{6.84}$$

$$= \sum_{i=1}^{n}\frac{\tilde{a}_i\tilde{a}_i^T\tilde{x}_i^{(k)}}{\tilde{s}_i^{(k)}}. \tag{6.85}$$

$\tilde{A}\tilde{D}_k^2\tilde{A}^T$ is a linear combination of a sequence of rank-one matrices. If one of these rank-

one matrices is dense, i.e. if one column of $\tilde{A}$ is dense, the whole matrix $\tilde{A}\tilde{D}_k^2\tilde{A}^T$ is dense. When $\tilde{A}\tilde{D}_k^2\tilde{A}^T$ is dense, it is expensive to compute its Cholesky decomposition. To overcome this problem, we used the Sherman-Morrison formula to deal with the dense columns in $\tilde{A}$, details are introduced in [54]. Assume there is only one dense column $\tilde{a}_j$.

$$\tilde{A}\tilde{D}_k^2\tilde{A}^T = \sum_{i=1,i\neq j}^{n} \frac{\tilde{a}_i\tilde{a}_i^T\tilde{x}_i^{(k)}}{\tilde{s}_i^{(k)}} + \frac{\tilde{a}_j\tilde{a}_j^T\tilde{x}_j^{(k)}}{\tilde{s_j}^{(k)}}. \tag{6.86}$$

In this case, the first term is sparse, and the second term is dense. Then, we perform minimal degree ordering algorithm and sparse Cholesky decomposition to the first term to obtain

$$L_kL_k^T = P_k^T(\sum_{i=1,i\neq j}^{n} \frac{\tilde{a}_i\tilde{a}_i^T\tilde{x}_i^{(k)}}{\tilde{s}_i^{(k)}})P_k, \tag{6.87}$$

where P_k is a permutation matrix which comes from reordering algorithms. Then, Equation (6.82) becomes

$$(P_kLL^TP_k^T + \frac{\tilde{a}_j\tilde{a}_j^T\tilde{x}_j^{(k)}}{\tilde{s}_j^{(k)}})\Delta\tilde{\lambda}^{(k)} = \tilde{A}(\tilde{x}^k - \sigma\mu(\tilde{S}^{(k)})^{-1}e). \tag{6.88}$$

To obtain $\Delta\tilde{\lambda}^{(k)}$ we first compute $\Delta\hat{\lambda}^{(k)} = (P_kLL^TP_k^T)^{-1}\tilde{A}(\tilde{x}^{(k)} - \sigma\mu(\tilde{S}^{(k)})^{-1}e)$ which can be solved by two triangular substitutions. When $\hat{\lambda}^{(k)}$ is obtained, we can use the Sherman-Morrison formula to compute $\tilde{\lambda}^{(k)}$ easily. The Sherman-Morrison formula can be written as follows,

$$(G + ab^T)^{-1} = G^{-1} - \frac{G^{-1}ab^TG^{-1}}{1 + b^TG^{-1}a}, \tag{6.89}$$

where $G \in \mathbb{R}^{n\times n}$ is nonsingular, and a, $b \in \mathbb{R}^n$. Hence, according to the Sherman-Morrison

formula

$$\begin{aligned}
\tilde{\lambda}^{(k)} &= \left(P_k L L^T P_k^T + \frac{\tilde{a}_j \tilde{a}_j^T \tilde{x}_j^{(k)}}{\tilde{s}_j^{(k)}} \right)^{-1} \tilde{A}(\tilde{x}^{(k)} - \sigma\mu(\tilde{S}^{(k)})^{-1}e) \\
&= \left((P_k L L^T P_k^T)^{-1} - \frac{(P_k L L^T P_k^T)^{-1} \frac{\tilde{a}_j \tilde{a}_j^T \tilde{x}_j^{(k)}}{\tilde{s}_j^{(k)}} (P_k L L^T P_k^T)^{-1}}{1 + \tilde{a}_j^T (P_k L L^T P_k^T)^{-1} \frac{\tilde{a}_j \tilde{x}_j^{(k)}}{\tilde{s}_j^{(k)}}} \right) \tilde{A}(\tilde{x}^{(k)} - \sigma\mu(\tilde{S}^{(k)})^{-1}e) \\
&= \hat{\lambda}^{(k)} - \frac{(P_k L L^T P_k^T)^{-1} \frac{\tilde{a}_j \tilde{a}_j^T \tilde{x}_j^{(k)}}{\tilde{s}_j^{(k)}} \hat{\lambda}^{(k)}}{1 + \tilde{a}_j^T (P_k L L^T P_k^T)^{-1} \frac{\tilde{a}_j \tilde{x}_j^{(k)}}{\tilde{s}_j^{(k)}}}.
\end{aligned}$$

In the above equations, $(P_k L L^T P_k^T)^{-1}\tilde{a}_j$ can be computed by two triangular substitutions.

For DCGLS, DGMRESNE and GreGMRES approaches, we set the inner iteration stopping criterion to be,

$$\|\tilde{A}^T r\|_2 < 10^{-8}\|\tilde{A}^T b\|_2, \tag{6.90}$$

where $\tilde{A}$ is the coefficient of the least squares problem, and b is the corresponding right hand side vector, and the r is the residual vector. We stop the Interior Point Method when

$$\tilde{x}_i^{(k)} \tilde{s}_i^{(k)} < 10^{-6}, \;\; i = 1, 2, \ldots, n. \tag{6.91}$$

In the followings tables, in each cell, for DCGLS and DGMRESNE we list the following information,

- Number of outer iteration,
- Average inner iteration per outer iteration,
- Total cpu time.

For GreGMRES, we record

- Number of outer iteration,
- Average inner iteration per outer iteration,
- Total cpu time,
- Dropping tolerance.
- Switching tolerance.

For Cholesky, we record

- Number of outer iteration,
- Total cpu time.

From Equation (6.57), L is difficult to compute. Hence, here we simply set L to be a integer between 1 and 10 for different problems. The parameter σ^k in the Interior Point method Algorithm 6.1.1 is set to be 0.5. For the GreGMRES method, the switching tolerance is usually reduced by half after each Interior Point method step because when the solution tends to the true solution, the least squares problem may become more and more ill-conditioned and tend to be nearly rank deficient, hence, smaller switching tolerance makes sure that no wrong linearly dependent columns are detected. We set the maximal iteration for DCGLS to be $50,000$, and the maximal iteration of DGMRESNE and GreGMRES is 2000. In the tables "X" means the optimal value of the objective function of the linear programming problem is not achieved by the computed solution. "†" means that the inner iterative methods did not converge within the maximal iteration, or a breakdown happens.

Table 6.1: Numerical Results, augmented approach

Matrices names	Size $m \times n$	L	DCGLS out, in, cpu	DGMRESNE out, in, cpu	GreGMRES out, in, cpu, (τ_d τ_s)	Cholesky out, cpu
25fv47	821×1876	3	52, 1914, 55.92	42, 280, 50.85	51, 2, 79.08 (1.e-6, 1.e-5)	50, 19.44
adlittle	56×138	4	33, 98 , 0.12	34, 49, 1.64	33, 14, 1.1 (1.e-2, 1.e-6)	33, 0.02
afiro	27×51	3	27, 27, 0.01	27, 27, 1.14	27, 4, 0.88 (1.e-2, 1.e-6)	27, 0.00
agg	488×615	9	61, 408, 5.27X	60, 148, 25.07X	62, 19, 13.64 (1.e-6, 1.e-6)	49, 4.03X
agg2	516×758	7	54, 226, 3.36	53, 117, 17.81	53, 2, 13.85 (1.e-6, 1.e-6)	53, 6.06
agg3	516×758	7	L =8, 59, 218, 3.57	52, 128, 20.14	55, 2, 14.25 (1.e-6, 1.e-6)	55, 6.34
bandm	305×472	3	35, 634, 3.48	35, 206, 17.82	34, 2, 4.6 (1.e-6, 1.e-6)	34, 0.71
beaconfd	173×295	4	33, 73, 0.36	34, 58, 2.39	32, 13, 2.21 (1.e-6, 1.e-3)	33, 0.32
blend	74×114	2	27, 210, 0.21	26, 74, 1.51	26, 3, 0.98 (1.e-3, 1.e-4)	26, 0.03
bnl1	643×1586	4	54, 1262, 25.75X	47, 343, 129.89X	58, 6, 42.87 (1.e-6, 1.e-6)	48, 2.21X
bnl2	2324×4486	3	66, 2493, 182.15	64, 466, 1145.98	60, 6, 1134.94 (1.e-6, 1.e-6)	60, 40.94
brandy	220×303	3	36, 1315, 5.66	36, 174, 9.62	36, 7, 2.48 (1.e-6, 1.e-5)	35, 0.31
capri	418×643	4	47, 3669, 31.19X	41, 241, 34.26X	45, 5, 11.71 (1.e-6, 1.e-4)	39, 1.03
d6cube	415×6184	3	65, 710, 68.11X	70, 243, 85.24X	61, 12, 66.61(1.e-6, 1.e-6)	61, 12.02
degen2	444×757	4	37, 1240, 10.9X	37, 241, 38.66X	42, 146, 45.14(1.e-1.e-6, 1.e-2)	4†
degen3	1503×2604	3	43, 5471, 243.54X	39, 400, 399.47X	40, 246, 999.8(1.e-6, 1.e-3)	39, 222.43
fffff800	524×1028	6	61, 6833, 138.97X	66, 366, 160.22X	74, 12, 46.96 (1.e-6, 1.e-6)	74, 16.43
fit1d	1050×2075	4	47, 109, 3.68	47, 52, 7.58X	48, 5, 129.29(1.e-4, 1.e-6)	49, 4.36
fit1p	1026×2076	4	45, 654, 17.75X	45, 137, 47.82	43, 13, 138.56 (1.e-4, 1.e-6)	45, 40.6
gfrd-pnc	874×1418	7	55, 3335, 62.33X	53, 291, 168.88X	58, 30, 125.81 (1.e-6, 1.e-6)X	43, 2.57

Table 6.2: Numerical Results, augmented approach

Matrices names	Size $m \times n$	L	DCGLS out, in, cpu	DGMRESNE out, in, cpu	GreGMRES out, in, cpu (τ_d, τ_s)	Cholesky out, cpu
grow15	900×1245	9	61, 340, 8.48	60, 202, 107.14	62, 2, 118.89 (1.e-6, 1.e-6)	62, 19.85
grow22	1320×1826	9	65, 754, 29.23	65, 261, 272.76	68, 6, 425.38(1.e-6, 1.e-6)	66, 60.57
grow7	420×581	9	57, 168, 1.9	60, 137, 22.8	57, 2, 15.39 (1.e-6, 1.e-6)	57, 2.69
israel	174×316	7	51, 633, 3.74X	57,152, 10.95X	59, 3, 3.69 (1.e-6, 1.e-6)	59, 1.27
kb2	52×77	5	38, 212, 0.22	36, 47, 1.83X	38, 1, 1.35 (1.e-4, 1.e-6)	38, 0.02
lotfi	153×366	5	46, 357, 1.48X	42, 108, 4.82	42, 25, 2.87 (1.e-4, 1.e-3)	41, 0.16
pilotnov	1519×2990	5	53, 1925, 85.21	63, 673, 1514.82	62, 103, 889.93 (1.e-5, 1.e-6)	61, 50.6
qap8	912×1632	1	26, 1676, 19.61	25, 108, 14.81	25, 129, 200 (1.e-6, 1.e-1)	1†
recipe	186×299	3	31, 349, 0.87	32, 144, 5.66	27, 4, 1.42 (1.e-6, 1.e-6)	27, 0.1
sc105	105×163	4	35, 118, 0.19	35, 81, 2.53	34, 2, 1.34 (1.e-4, 1.e-6)	34, 0.05
sc205	205×317	4	34, 240, 0.68	34, 142, 6.78	31, 3, 1.76 (1.e-4, 1.e-6)	31, 0.14
sc50a	50×78	3	28, 55, 0.04	28, 46, 1.43	28, 9, 1.02 (1.e-1, 1.e-6)	28, 0.01
sc50b	50×78	3	27, 49, 0.03	26, 43, 1.32	26, 9, 0.95 (1.e-1, 1.e-6)	26, 0.01
scagr25	471×671	6	50, 938, 8.67	50, 252, 58.92	50, 3, 9.46 (1.e-5, 1.e-6)	50, 1.16
scagr7	129×185	5	40, 193, 0.42	40, 102, 3.81	40, 2, 1.61 (1.e-5, 1.e-6)	40, 0.08
scfxm1	330×600	4	46, 1891, 15.34	45, 289, 42.81	42, 3, 5.62 (1.e-6, 1.e-6)	42, 0.84
scfxm2	660×1200	4	46, 3882, 62.65	46, 510, 259.88	46, 5, 33.85 (1.e-6, 1.e-6)	44, 3.3
scfxm3	990×1800	4	47, 6298, 155.63	44, 676, 678.86	46, 7, 108.23 (1.e-6, 1.e-6)	44, 7.18
scorpion	388×466	4	40, 297, 1.68X	47, 225, 37.39X	38, 62, 13.28 (1.e-6, 1.e-3)	1†
scrs8	490×1275	2	39, 773, 8.03	39, 310, 60.72	39, 6, 12.27 (1.e-6, 1.e-6)	38, 0.9

Table 6.3: Numerical Results, augmented approach

Matrices names	Size $m \times n$	L	DCGLS out, in, cpu	DGMRESNE out, in, cpu	GreGMRES out, in, cpu (τ_d, τ_s)	Cholesky out, cpu
scsd1	77×760	2	28, 49, 0.21	28, 44, 1.62	24, 5, 1.03 (1.e-3, 1.e-6)	24, 0.08
scsd6	147×1350	1	25, 81, 0.52	26, 76, 2.43	24, 6, 1.47 (1.e-3, 1.e-6)	24, 0.25
scsd8	397×2750	2	29, 172, 2.64	29, 153, 13.24	29, 2, 6.84 (1.e-6, 1.e-6)	29, 1.86
sctap1	300×660	2	40, 357, 2.19	40, 179, 15.53	36, 6, 3.16 (1.e-4, 1.e-6)	36, 0.36
sctap2	1090×2500	1	35, 465, 7.29	36, 261, 97.13	33, 2, 72.31 (1.e-6, 1.e-6)	33, 3.46
sctap3	1480×3340	1	38, 394, 11.37	38, 235, 124.91	35, 2, 130.33 (1.e-5, 1.e-6)	35, 6.74
share1b	117×253	7	57, 787, 3.34X	54, 111, 5.41X	58, 2, 2.52 (1.e-6, 1.e-6)	58, 0.16
share2b	96×162	3	34, 625, 1.13	32, 91, 2.45	32, 2, 1.31 (1.e-5, 1.e-6)	32, 0.08
shell	903×2144	7	68, 1100, 33.39	63, 242, 11.45	80, 43, 171.78 (1.e-6, 1.e-6)	59, 3.83
ship04l	402×2166	3	45, 647, 11.98	44, 153, 17.62	42, 40, 14.65 (1.e-6, 1.e-6)	31, 0.62
ship04s	912×1506	5	42, 414, 5.37X	49, 79, 7.36X	49, 31, 11.45 (1.e-6, 1.e-6)	46, 0.74
ship08l	778×4363	3	47, 934, 35.86X	53, 233, 88.07X	54, 105, 155.02 (1.e-6, 1.e-6)	52, 3.37
ship08s	778×2467	5	42, 834, 18.32X	53, 113, 23.21X	51, 78, 86.58 (1.e-6, 1.e-6)	50, 2.44
ship12l	1151×5533	3	61, 941, 60.92	62, 314, 307.46X	72, 63, 410.77 (1.e-6, 1.e-6)	66, 8.12
ship12s	1151×2869	4	L = 3, 56, 697, 25.16	56, 260, 162.5	53, 57, 195.65(1.e-6, 1.e-6)	55, 5.05
standata	479×1394	4	55, 360, 5.58X	53, 191, 33.64X	50, 6, 19.64 (1.e-6, 1.e-6)	47, 1.05
standmps	587×1394	4	61, 435, 8.25	64, 257, 85.29X	65, 7, 43.45 (1.e-6, 1.e-6)	61, 2.03
stocfor1	117×165	4	35, 197, 0.34	34, 109, 3.27	34, 5, 1.41 (1.e-5, 1.e-6)	34, 0.06
stocfor2	2157×3045	4	47, 3249, 124.6	45, 469, 761.03	44, 53, 1122.03 (1.e-6, 1.e-8)	44, 36.22
truss	1000×8806	4	50, 2390, 191.69X	43, 432, 415.38	48, 10, 179.36 (1.e-5)	49, 49.18
tuff	362×659	3	38, 791, 6.77	39, 290, 39.68	38, 4, 6.88 (1.e-6, 1.e-8)	37, 0.89
vtp.base	281×430	5	48, 1131, 6.1X	48, 189, 17.94X	50, 30, 6.99 (1.e-6, 1.e-8)	46, 0.48
wood1p	244 ×2595	4	51, 198, 18.63	36, 83, 10.64X	57, 1, 28.15 (1.e-8, 1.e-10)	51, 5.61

6.4 Conclusion

From the above the numerical results, we find out:

- The Cholesky approach is the fastest method to solve the linear programming problems. However, for some problems, the Cholesky decomposition broke down.

- Comparing to the Cholesky approach, using iterative methods to solve the least squares problems in the interior-point method steps is usually slow. One reason is that when the augmented approach is used, the last column of $\tilde{D}_k\tilde{A}^T$, which is the coefficient matrix of the least squares problem in the kth interior-point method step, is dense. This dense column makes the operations involving $\tilde{D}_k\tilde{A}^T$ much more expensive.

- Among these three iterative methods we used, CGLS with diagonal scaling is usually the fastest, and GMRES solving the diagonal scaled normal equation approach is usually the slowest. There are also many cases in which GMRES with our Greville preconditioner is the fasted.

- CGLS with diagonal scaling approach and GMRES solving the diagonal scaled normal equation approach sometimes can not obtain the optimal solution. At this point, GMRES with the Greville preconditioner is more reliable.

- Although the GMRES with the Greville preconditioner approach is the slowest in many cases, it can obtain the optimal solution when the Cholesky approach breaks down or could not obtain the optimal solution. The reason is the Cholesky approach becomes less stable when the Cholesky decomposition is used together with the Sherman-Morrison. If the Sherman-Morrison is not used, then the Cholesky de-

composition is performed on a almost dense matrix, which is much more expensive than using GMRES with the Greville preconditioner.

Chapter 7

Conclusion

In this thesis, we studied preconditioning techniques for least squares problems and especially the rank deficient problems. We focused on constructing approximate generalized inverse of the coefficient matrices of the problems and using what we constructed to precondition the Krylov subspaces methods. In our methods, we can construct rank deficient preconditioners for rank deficient problems, which is a new idea to preconditioning. We also proved the equivalence between the preconditioned problem and the original problem which gives theoretical justification to our preconditioning methods.

In this thesis, two different approaches are considered.

- One is based on the Minimal Residual method which was developed for computing the approximate inverse of a nonsingular matrix. We adapted this method so that we can compute the approximate generalized inverse of a rectangular matrix. We also did theoretical analysis to show that the preconditioners constructed by our algorithm are suitable to be used to precondition least squares problems. To make the minimal residual method work with rectangular matrices, we have to use the steep-

est descent direction to update the preconditioner matrix M_k so that $\|I - M_k A\|_F$ reduces. The steepest descent direction makes the minimal residual method more expensive. Hence, without numerical droppings the preconditioning will become very expensive in computations and storage. However, our theoretical analysis do not hold when numerical droppings are performed.

- The other approach was based on an old algorithm: the "Greville's Method". The Greville's method was used to compute the Moore-Penrose inverse of any matrix, the main idea is based on the rank-one update. We reformed this algorithm and showed the inexplicit $A^T A$-orthogonalization process in the algorithm and its relation with the RIF algorithm. This approach can give rank deficient preconditioners for rank deficient matrices. And our theoretical results justify that the preconditioned problems is equivalent to the original problem and can be solved by the GMRES method under Assumption 5.5.1 even with numerical droppings. Our numerical examples showed that our preconditioners may perform better than other methods when the coefficient matrix of the linear system is rank deficient and ill-conditioned. We also showed that for rank deficient matrices, detecting certain amount of rank deficiencies can bring advantage to the convergence. However, we cannot control the sparsity of the preconditioner directly. Since in the preconditioner $M = ZF^{-1}V^T$, we can only control the sparsity of matrix Z. If we perform numerical droppings on the vectors of V, this factorization will not hold. The computational cost is also related with how many rank deficient columns we detect in the preconditioning algorithm, since it takes more computations to deal with the rank deficiency. In this preconditioning algorithm we have two parameters τ_d, τ_s. τ_d is a threshold for numerical droppings, τ_s is a threshold for detecting the linear dependence. The choice of τ_s is not very clear. In our experiments, we usually set τ_s to be 10^{-6}. A small τ_s is safer in the

sense to keep the equivalence between the original problem and the preconditioned problem.

As an application from the real world, we applied our preconditioners to the linear programming problems in which sparse, ill-conditioned and sometimes rank deficient matrices arise. To solve the linear programming problems, we used our preconditioner to precondition the least squares problems abstracted from the interior-point method in each step. Numerical results showed that using our preconditioners, the least squares problems can be solved more efficiently compared to using diagonal scaled GMRES to solve the normal equations, but it is slower than using diagonal scaled CGLS to solve the least squares problems for many problems. All of these three iterative methods are slower than the Cholesky approach. Numerical results also shows that GMRES with Greville preconditioner approach is more stable compared to the rest methods. Even when the Cholesky approach does not work, GMRES with our preconditioner approach still can compute an optimal solution.

There are still a lot of issues to consider in the future. For both of our preconditioning algorithms, preconditioning time is very heavy. A preprocessing to the coefficient matrices or a more delicate dropping strategy is necessary. For the Greville's method approach, detecting the linear dependencies relies on how to choose the parameter τ_s which is still open. In the end, we hope our work can bring some new ideas to this preconditioning field.

Bibliography

[1] P. R. AMESTOY, T. A. DAVIS, AND I. S. DUFF, *Amd version 2.2 user guide*, 2007. http://www.cise.ufl.edu/research/sparse/amd.

[2] Z.-Z. BAI, I. S. DUFF, AND A. J. WATHEN, *A class of incomplete orthogonal factorization methods. i: Methods and theories*, BIT Numerical Mathematics, 41 (2001), pp. 53 – 70.

[3] R. E. BANK AND C. WAGNER, *Multilevel ilu decomposition*, Numer. Math., 82 (1999), pp. 543 – 574.

[4] S. T. BARNARD, L. M. BERNARDO, AND H. D. SIMON, *An mpi implementation of the spai preconditioner on the t3e*, The International Journal of High Performance Computing Applications, 13 (1999), pp. 107 – 128.

[5] A. BEN-ISRAEL AND T. N. GREVILLE, *Generalized inverses: Theory and applications*, Pure and Applied Mathematics. John Wiley & Sons, New York etc., 1974.

[6] M. W. BENSON, *Iterative solution of large scale linear systems*, M.Sc. Thesis, Lakehead Universeity, Thunder Bay, ON, Canada, 1973.

[7] M. W. BENSON AND P. O. FREDERICKSON, *Iterative solution of large sparse linear systems arising in certain multidimensional approximataion problems*, Utilitas Math., 22 (1982), pp. 127 – 140.

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

[9] M. BENZI, J. K. CULLUM, AND M. TŮMA, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM Journal on Scientific Computing, 22 (2000), pp. 1318–1332.

[10] M. BENZI AND M. TŮMA, *Numerical experiments with two approximate inverse preconditioners*, BIT Numerical Mathematics, 38 (1998), pp. 234–241.

[11] ——, *A comparative study of sparse approximate inverse preconditioners*, Applied Numerical Mathematics, 30 (1999), pp. 305 – 340.

[12] ——, *A robust incomplete factorization preconditioner for poitive definite matrices*, Numerical linaer algebra with applications, 10 (2003), pp. 385–400.

[13] ——, *A robust preconditioner with low memory requirements for large sparse least squares problems*, SIAM Journal on Scientific Computing, 25 (2003), pp. 499–512.

[14] Å. BJÖRCK, *Numerical Methods for Least Squares Problems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996.

[15] Å. BJÖRCK, T. ELFVING, AND Z. STRAKOS, *Stability of conjugate gradient and lanczos methods for linear least squares problems*, SIAM Journal on Matrix Analysis and Applications, 19 (1998), pp. 720–736.

[16] P. N. BROWN AND H. F. WALKER, *GMRES on (nearly) singular systems*, SIAM Journal on Matrix Analysis and Applications, 18 (1997), pp. 37–51.

[17] N. I. BULEEV, *A numerical method for solving two- and three-dimensional diffusion equations*, Mat. Sb., 51 (1960), pp. 227 – 238.

[18] P. BUSINGER AND G. H. GOLUB, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269–276. Also in [53, pp. 111–118], Contribution I/8.

[19] S. L. CAMPBELL AND C. D. MEYER, JR., *Generalized Inverses of Linear Transformations*, Pitman, London, 1979. Reprinted by Dover, New York, 1991.

[20] E. CHOW AND Y. SAAD, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM Journal on Scientific Computing, 19 (1998), pp. 995–1023.

[21] J. D. F. COSGROVE, J. C. ÓAZ, AND A. GRIEWANK, *Approximate inverse preconditioning for sparse linear systems*, International Journal of Computer Mathematics, 44 (1992), pp. 91 – 110.

[22] X. CUI AND K. HAYAMI, *Greville's method for preconditioning least squares problems*, Tech. Report NII-2008-008E, National Institute of Informatics, Tokyo, August 2008.

[23] ——, *Generalized approximate inverse preconditioners for least squares problems*, Japan Journal of Industrial and Applied Mathematics, 26 (2009), pp. 1 – 14.

[24] X. CUI, K. HAYAMI, AND J.-F. YIN, *Greville's method for preconditioning least squares problems*, in Proceedings of ALGORITMY 2009 Conference on Scientific Computing, 2009, pp. 440–448.

[25] T. A. DAVIS, *The universtiy of florida sparse matrix collection*, 2008. http://www.cise.ufl.edu/research/sparse/matrices/.

[26] I. S. DUFF, R. G. GRIMES, AND J. G. LEWIS, *Sparse matrix test problems*, ACM Transactions on Mathematical Software, 15 (1989), pp. 1–14.

[27] M. EIERMANN AND O. G. ERNST, *Geometric aspects of the theory of krylov subspace methods*, Acta Numerica, 10 (2001), pp. 251–312.

[28] M. ENGELI, T. GINSBURG, AND H. RUTISHAUSER, *Refined iterative methods for computation of the solution and the eigenvalues of self-adjoint boundary value problems*, Birkhäuser, Basel/Stuttgart, 1959.

[29] G. H. GOLUB AND C. F. V. LOAN, *Matrix Computations*, Johns Hopkins University Press, Baltimore, MD, USA, third ed., 1996.

[30] N. I. M. GOULD AND J. A. SCOTT, *Sparse approximate-inverse preconditioners using norm-minimization techniques*, SIAM Journal on Scientific Computing, 19 (1998), pp. 605–625.

[31] T. N. E. GREVILLE, *Some applications of the pseudoinverse of a matrix*, SIAM Review, 2 (1960), pp. 15–22.

[32] I. GRIVA, S. G. NASH, AND A. SOFER, *Linear and Nonlinear Optimization*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2009.

[33] M. J. GROTE AND T. HUCKLE, *Parallel preconditioning with sparse approximate inverses*, SIAM Journal on Scientific Computing, 18 (1997), pp. 838–853.

[34] K. HAYAMI, J.-F. YIN, AND T. ITO, *GMRES methods for least squares problems*, Tech. Report NII-2007-009E, National Institute of Informatics, Tokyo, July 2007.

[35] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear system*, J. Res. Nat. Bur. Standards., B49 (1952), pp. 409–436.

[36] A. JENNINGS AND M. A. AJIZ, *Incomplete methods for solving a'ax = b*, SIAM J. Sci. Comput., 5 (1984), pp. 978 – 987.

[37] V. P. LL'IN, *Iterative incomplete factorization methods*, World Scientific Pub Co Inc, Singapore, 1992.

[38] J. A. MEIJERINK AND H. A. VAN DER VORST, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[39] R. D. MONTEIRO AND I. ADLER, *Interior path following primal-dual algorithms. part i: Linear programming*, Mathematical Programming, 44 (1989), pp. 27–41.

[40] N.I.BULEEV, *A numerical method for solving two-dimensional diffusion equations*, Atomic Energer, 6 (1960), pp. 222 – 224.

[41] T. A. OLIPHANT, *An extrapolation process for solving linear systems*, Q. Appl. Math., 20 (1962), pp. 257 – 267.

[42] C. C. PAIGE AND M. A. SAUNDERS, *Solution of sparse indefinite systems of linear equations*, SIAM Journal on Numerical Analysis, 12 (1975), pp. 617–629.

[43] C. C. PAIGE AND M. A. SAUNDERS, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Softw., 8 (1982), pp. 43–71.

[44] A. T. PAPADOPOULOS, I. S. DUFF, AND A. J. WATHEN, *A class of incomplete orthogonal factorization methods. ii: Implementation and results*, BIT Numerical Mathematics, 45 (2005), pp. 159 – 179.

[45] Y. SAAD, *Preconditioning techniques for nonsymmetric and indefinite linear systems*, Journal of Computational and Applied Mathematics, 24 (1984), pp. 89 – 105.

[46] ——, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2003.

[47] Y. SAAD AND M. H. SCHULTZ, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[48] R. S. VARGA, *Factorization and normalized iterative methods*, Boundary Problems in Differential Equations (R. E. Langer, ed.), (1960), pp. 121 – 142.

[49] K. VUIK, A. G. J. SEVINK, AND G. C. HERMAN, *A preconditioned krylov subspace method for the solution of least squares problems in inverse scattering*, J. Comput. Phys., 123 (1996), pp. 330–340.

[50] G. WANG, Y. WEI, AND S. QIAO, *Generalized Inverses: Theory and Computations*, Since Press, Beijing, 2003.

[51] X. WANG, K. A. GALLIVAN, AND R. BRAMLEY, *Cimgs: An incomplete orthogonal factorizationpreconditioner*, SIAM Journal on Scientific Computing, 18 (1997), pp. 516–536.

[52] P.-Å. WEDIN, *Perturbation theory for pseudo-inverses*, BIT Numerical Mathematics, 13 (1973), pp. 217 – 232.

[53] J. H. WILKINSON AND C. REINSCH, eds., *Linear Algebra*, vol. II of Handbook for Automatic Computation, Springer-Verlag, Berlin, 1971.

[54] S. J. WRIGHT, *Primal-Dual Interior-Point Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.