Summer 8-1-2021

A Modified Preconditioned Conjugate Gradient Method for Approximating the Scattering Amplitude

Samson Ayo

A MODIFIED PRECONDITIONED CONJUGATE GRADIENT METHOD FOR APPROXIMATING THE SCATTERING AMPLITUDE

by

Samson Oghenemine Ayo

A Thesis
Submitted to the Graduate School,
the College of Arts and Sciences
and the School of Mathematics and Natural Sciences
of The University of Southern Mississippi
in Partial Fulfillment of the Requirements
for the Degree of Master of Science

Approved by:

Dr. James Lambers, Committee Chair
Dr. C.S Chen
Dr. Huiqing Zhu

August 2021

ABSTRACT

In this thesis, we look at an iterative method for approximating the scattering amplitude that involves solving two linear systems: a forward system ($A\mathbf{x} = \mathbf{b}$) and an adjoint system ($A^T\mathbf{y} = \mathbf{g}$). Once these two systems are solved, the scattering amplitude, defined by $\mathbf{g}^T\mathbf{x} = \mathbf{y}^T\mathbf{b}$ is easily obtained. We derive a conjugate gradient-like iteration for a nonsymmetric saddle point matrix that is constructed to have a real positive spectrum. We investigate the use of Schur Complement preconditioners with block-diagonal factorization to speed up the convergence of our method and compare the results to our NspcG method without preconditioning.

ACKNOWLEDGMENTS

I am thankful to all who have supported me and contributed to my success up till this point. It is impossible to mention all of these people. All I can say is I am grateful for your support. There are specific people whose support and influence are too great, and they deserve mention.

One of such persons in my advisor Dr. James Lambers, whose calm, reassuring words on my first day as a USM student helped convinced me that I could do this. Every time I had ever doubted my ability to be successful in my academic career, I am reminded of his kind and encouraging words on that first day of class. His support, wisdom, and advice is the reason I am able to complete this thesis, and for that, I am truly grateful.

I would also like to thank the Math Zone director, Ms. Emileigh McCardle. She has been an incredible mentor throughout my time here. To all my professors, especially Dr. Chen, I say thank you. The rigorous standards set in these classes provided the right amount of challenge and motivation needed to complete this degree successfully.

The community here at USM also deserves mention. Leaving the comfort of family and country to travel to a foreign country for education can sometimes be challenging. To do so and have to enjoy similar comfort in a foreign land is only possible if you have a strong community supporting you. And Hattiesburg has indeed been a strong community. The ifriends group here at USM has been a pillar for the international student community, and for this, I am thankful.

Ultimately, I want to express my profound gratitude to my family, whose love and support are the reasons I am here today. Daily, I give thanks to God for having a family whose love and belief in education have spurred me on up till this point. They are indeed a blessing.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

LIST OF ABBREVIATIONS

CFD - Computational Fluid Dynamics
NspCG - Nonsymmetric saddle point Conjugate Gradient
PCG - Preconditioned Conjugate Gradient
RCS - Radar Cross Section
RHS - Right Hand Side
SVD - Singular Value Decomposition

Chapter 1

INTRODUCTION

1.1 The Scattering Amplitude Problem

In quantum physics, the *scattering amplitude* is the amplitude of the outgoing spherical wave relative to the incoming plane wave [7]. Its dimension is the length. It is useful when one is interested in knowing how incoming waves are scattered and what is reflected when an incoming wave **b** is impinging on an object. The scattering amplitude can be calculated by taking the inner product of the right-hand side (rhs) vector **g** of the dual (adjoint) system

$$A^T \mathbf{y} = \mathbf{g} \tag{1.1}$$

and the solution **x** of the primal (forward) system.

$$A\mathbf{x} = \mathbf{b} \tag{1.2}$$

Approximation of scattering amplitude is useful in many areas of application. See [16] and the references there in for some of these applications. A specific field where the approximation of the scattering amplitude is applied is in the design of stealth planes [1].

Stealth planes are planes that can fly undetected at high speed, almost invisible to radar. A key component in stealth planes' design involves reducing the plane's Radar Cross Section (RCS) (smaller RCS means lower detectability). The RCS, denoted by σ, is the integral of the square of the modulus of scattering amplitude over all solid angles Ω.

$$\sigma = \int |f_k(\theta, \phi)|^2 d\Omega$$

One way of reducing the RCS of an aircraft involves reducing the scattering of reflected waves in the direction of the radar receiver. When a radar wants to detect if a plane is in its airspace, it sends an incident wave **b** in the plane's direction, and this wave is reflected or scattered by the plane. Two types of scattering occur. We seek to reduce the amount of backscattering since this is the wave that is reflected in the receiver's direction.

Engineers already have a conceptual idea of how much scattering is needed for the aircraft to evade detection. They, therefore, want to estimate the actual scattering caused by

their design to determine if it is sufficient. If it is not sufficient, a new design is created, and the scattering amplitude of this design is computed. They continue creating designs/models and estimating its scattering amplitude until they achieve the right amount of scattering. When designing stealth planes, it is impractical to go straight to fabrication after creating a conceptual design, as this design may not meet the specified RCS needed for the aircraft to evade detection. So, the design needs to be tested first, and when the correct RCS is obtained, fabrication can then begin. Due to modern advances in computing speed and power, it is often desirable to employ a computer in the design phase to predict what the RCS will look like before fabricating an actual object. Many iterations of this prediction process can be performed at high speed in quick time and at a considerably low cost.

The scattering amplitude ($\mathbf{g}^T\mathbf{x} = \mathbf{y}^T\mathbf{b}$) is obtained by solving the forward and adjoint systems simultaneously using iterative methods. The RCS can then be computed using this quantity. If the RCS computed is not sufficiently low, a new conceptual design is tested, its scattering amplitude approximated, and thus its RCS. Thus, different conceptual designs are continually being tested until a suitable RCS is obtained.

The scattering amplitude expressed as $\mathbf{g}^T\mathbf{x}$ [7] or equivalently, $\mathbf{y}^T\mathbf{b}$, establishes a connection between the rhs of the dual system and the solution of the primal system in signal processing. First, the field $\mathbf{x}$, referred to as the scattered field, is determined from the signal $\mathbf{b}$ through the system $A\mathbf{x} = \mathbf{b}$. The scattered field $\mathbf{y}$ is then determined in the system $A^T\mathbf{y} = \mathbf{g}$, where $\mathbf{g}$ (outgoing wave) is the scattered or received signal and has information about the object. Efficiently approximating the scattering amplitude requires that we efficiently solve the linear systems (1.1) and (1.2). So, it is enlightening to look at methods that have been employed in solving these linear systems. Some of these methods will be discussed below.

Solving the linear system (1.2) has always been of interest to researchers due to its importance in many applications beyond the scattering amplitude problem. There are several ways of obtaining solutions to these linear systems, and each solution method is dependent on the specific properties of the system matrix A. When the system matrix A is dense and small, a direct solution method is preferred. *LU* factorization-type methods are the preferred choice if the matrix A is unsymmetric. The LDL^T factorization is an example of a direct method that is used to solve some problems when the matrix A is symmetric. *Cholesky factorization* is another direct method. It is used when the symmetric matrix A is known to be positive definite [8]. However, for large, sparse systems, iterative methods become necessary.

The preferred iterative method for a large, sparse, symmetric positive definite system

is the *conjugate gradient* method[8]. However, when the system matrix is not symmetric positive definite, it is harder to find a solution for the linear system (1.2) using this method. In a case where the system matrix A is not symmetric, we can utilize methods such as the *generalized minimal residual (GMRES)* [14] and *biconjugate gradient (BiCG)* methods [3]. Furthermore, since approximating the scattering amplitude requires solutions to both the adjoint (dual) and forward (primal) problems, it is wise to choose methods that solve both of these problems simultaneously, like the *generalized least squares residual (GLSQR)* [17] and *quasi-minimal residual (QMR)* methods [13].

1.2 Approximation of the Scattering Amplitude

The methods of this thesis employ a conjugate gradient-like approach. This is so because, for large, sparse matrices, direct solution is computationally expensive or impossible, so that iterative solutions become the solution of choice. One such iterative approach which is remarkably effective for symmetric positive definite systems is the conjugate gradient method. In particular, the conjugate gradient method gives very rapid convergence if A is close to identity either in the sense of a low-rank perturbation or in the sense of the norm [8].

Since symmetry is what is needed to use a CG iteration, we could multiply both sides of (1.2) by A^T. This gives the normal equations with a symmetric matrix A^TA that is also positive definite when A is invertible. However, solving the scattering amplitude problem in this way is impractical because it does not solve both (1.1) and (1.2) simultaneously. Additionally, a major drawback of using this approach is that the condition number in the two-norm is now squared for A^TA. This is likely to increase the system's sensitivity, possibly making it ill-conditioned. So, this thesis employs an alternative approach. Our method, like the QMR and GLSQR solves both the forward (primal) and adjoint (dual) problems simultaneously. The idea is to transform these two problems into an equivalent system in which the matrix can be guaranteed to have real, positive eigenvalues and eigenvectors that are in some sense orthogonal, making it suitable for a solution by a conjugate gradient-like iteration. Symmetry is not the goal, but as we will find out, symmetry will be achieved with respect to some inner product. In order to achieve this goal, we make use of an idea suggested first by Gene Golub [6], and consider a nonsymmetric saddle point matrix that has the form

$$M = \begin{bmatrix} A^TWA & A^T \\ -A & 0 \end{bmatrix}.$$

Our only assumption is that the matrix W be assumed to be symmetric positive definite. We

make this assumption so that M fulfills the requirements to be a nonsymmetric saddle point matrix. Our aim is to choose W in order to ensure that M has real, positive eigenvalues. Next, we develop a preconditioner for M based on its Schur complement and give a block diagonal factorization of this preconditioner. This preconditioner is used with M, to create a preconditioned system matrix $\tilde{M}$ that is near identity so as to get rapid convergence. We also develop a modified conjugate gradient-like algorithm for the preconditioned system.

This thesis is structured as follows. In Chapter 2, we introduce the method of this paper, NspCG. In Chapter 3, we discuss preconditioning for our problem and derive a modified preconditioned NspCG algorithm to be used with our preconditioner. In Chapter 4, we present and analyze numerical results for our method. Conclusions and discussions of possible future work are given in Chapter 5.

Chapter 2

ITERATIVE METHODS FOR NONSYMMETRIC SADDLE POINT MATRICES

As mentioned in Chapter 1, our alternative approach will be to consider the matrix M defined by

$$M = \begin{bmatrix} A^T W A & A^T \\ -A & 0 \end{bmatrix} \tag{2.1}$$

where $A \in \mathbb{R}^{n \times n}$ is invertible, and W is a symmetric positive definite matrix. M is an example of a *nonsymmetric saddle point matrix*. It can be proved that $\mathbf{x}^T M \mathbf{x} \geq 0$ for all $\mathbf{x} \neq \mathbf{0}$. For proof of this, see [16].

2.1 Guaranteeing a Real Positive Spectrum

It is vital to choose W in a way that ensures that M has a real positive spectrum. Doing so makes M appropriate for a conjugate gradient-like iteration [12]. To make this choice, we need first to define

$$\mathcal{M}(\gamma) = \mathcal{J}p(M) = \mathcal{J}(M - \gamma I) = \begin{bmatrix} A^T W A - \gamma I & A^T \\ -A & \gamma I \end{bmatrix} \tag{2.2}$$

where p is a polynomial of degree one in the form $p(\zeta) = \zeta - \gamma$ for $\gamma \in \mathbb{R}$ and

$$\mathcal{J} = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix}.$$

We want to establish whether there exists a symmetric positive matrix $\mathcal{M}(\gamma)$ with respect to which M is symmetric, meaning M is $\mathcal{M}(\gamma)$-symmetric if $\mathcal{M}(\gamma)M = M^T\mathcal{M}(\gamma) = (\mathcal{M}(\gamma)M)^T$. The following theorem gives a necessary and sufficient condition for M to be suitable for a CG-like iteration.

Theorem 2.1.1. *Let A be an invertible $n \times n$ real matrix, and let W be a symmetric positive definite $n \times n$ matrix that satisfies*

$$\sigma_{min}(W) > \frac{2}{\sigma_{min}(A)}. \tag{2.3}$$

Then the matrix M defined by

$$M = \begin{bmatrix} A^T W A & A^T \\ -A & 0 \end{bmatrix}$$

has real positive eigenvalues and eigenvectors that are orthogonal with respect to the inner product defined by $\mathcal{M}(\gamma) = \mathcal{J}p(M)$. That is, this choice of W makes the matrix M suitable for conjugate gradient-like iteration. (See [16] for proof).

In that case, the matrix W fulfills the requirements for $\mathcal{M}(\gamma)$ to be symmetric positive definite. We are left with choosing a value for γ that guarantees that $\mathcal{M}(\gamma)$ is symmetric positive definite. Sumner [16] found this choice to be $\gamma = \frac{1}{2}\left(\lambda_{\min}(A^T W A)\right)$.

2.2 The Case where W is a Multiple of Identity.

Let $A = U\Sigma V^T$ be the SVD of A, where

$$U = \begin{bmatrix} \mathbf{u_1} & \cdots & \mathbf{u_n} \end{bmatrix}, \quad V = \begin{bmatrix} \mathbf{v_1} & \cdots & \mathbf{v_n} \end{bmatrix}$$

and $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$. If we choose $W = wI$ for some real scalar, w, then condition (2.3) reduces to

$$w > \frac{2}{\sigma_n}. \tag{2.4}$$

Following the analysis of the eigensystem of M described in [9], we conclude that the eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{2n}$ of M are given by

$$\mathbf{x}_{2j-1} = \begin{bmatrix} -\lambda_J^+ \mathbf{v}_j \\ \sigma_j \mathbf{u}_j \end{bmatrix}, \quad \mathbf{x}_{2j} = \begin{bmatrix} -\lambda_J^- \mathbf{v}_j \\ \sigma_j \mathbf{u}_j \end{bmatrix}, \quad j = 1, 2, \ldots, n, \tag{2.5}$$

with corresponding eigenvalues $\lambda = \lambda_j^+, \lambda_j^-$ that satisfy the quadratic equation

$$\lambda^2 - \sigma_j^2 w \lambda + \sigma_j^2 = 0 \tag{2.6}$$

It is possible to prove right away from (2.5) and (2.6) that these eigenvalues are real and positive, and the corresponding eigenvectors are linearly independent, if and only if w satisfies condition (2.4) [9]. The eigenvalues of M exhibit interesting behavior, as shown in Figure 2.1. First, the n smallest eigenvalues are nearly constant, with values very close to $1/w$ (2.6). Then, there is a surge in the magnitude of the eigenvalues, especially when A is ill-conditioned.

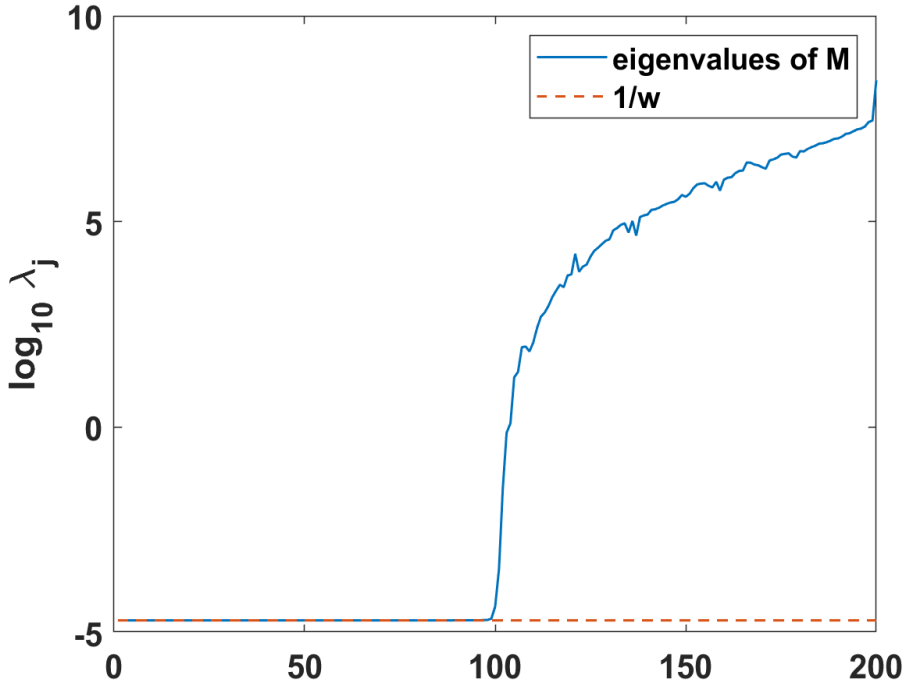

Figure 2.1: *Eigenvalues of matrix M from Example 1 in Chapter 4.*

2.3 The Nonsymmetric Saddle Point Conjugate Gradient Method

Let $A \in \mathbb{R}^{n\times n}$ be nonsymmetric. We will now introduce a Conjugate Gradient (CG) approach that solves the linear systems (1.1) and (1.2) by solving an equivalent system of the form $M\mathbf{z} = \mathbf{b}$, where

$$M \equiv \begin{bmatrix} A^T W A & A^T \\ -A & 0 \end{bmatrix} \tag{2.7}$$

and

$$\mathbf{z} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} A^T W \mathbf{b} + \mathbf{g} \\ -\mathbf{b} \end{bmatrix}. \tag{2.8}$$

To show that this new system indeed solves (1.1) and (1.2), consider the following matrix equation manipulations

$$\begin{bmatrix} A^T W A & A^T \\ -A & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} A^T W \mathbf{b} + \mathbf{g} \\ -\mathbf{b} \end{bmatrix}.$$

Carrying out matrix multiplication yields

$$A^T W A\mathbf{x} + A^T\mathbf{y} = A^T W\mathbf{b} + \mathbf{g} \tag{2.9}$$

$$-A\mathbf{x} = -\mathbf{b} \tag{2.10}$$

(2.10) simplifies to $A\mathbf{x} = \mathbf{b}$, which is the same as (1.2). Substituting this into (2.9) gives:

$$A^T W\mathbf{b} + A^T\mathbf{y} = A^T W\mathbf{b} + \mathbf{g}$$

This simplifies to $A^T\mathbf{y} = \mathbf{g}$, which is (1.1).

The matrix M is also nonsymmetric; however, the eigenvalues of M are real and positive since $\mathbf{x}^T M\mathbf{x} \geq 0$ for all $\mathbf{x}$. In the preceding discussion, we have established that if we carefully choose W to satisfy the assumptions of Theorem 2.1.1, then M is diagonalizable with real, positive eigenvalues. Furthermore, the bilinear form $(\mathbf{u},\mathbf{v})_G = \mathbf{v}^T G\mathbf{u}$ is a proper inner product, as G (to be defined later) is symmetric positive definite. The result is that M is G-symmetric as well as G-definite, meaning that $(M\mathbf{u},\mathbf{v})_G = (\mathbf{u},M\mathbf{v}_G)$ for all $\mathbf{u},\mathbf{v} \in \mathbb{R}^{2n}$, and $(\mathbf{u},M\mathbf{u})_G > 0$ for all $\mathbf{u} \neq 0$.

Define the vector $\mathbf{p}$ as:

$$\mathbf{p} = \begin{bmatrix} \mathbf{g} \\ \mathbf{0} \end{bmatrix}, \tag{2.11}$$

then the scattering amplitude can be computed as $\mathbf{p}^T\mathbf{z} = \mathbf{g}^T\mathbf{x}$. We now present a conjugate gradient method for solving the linear system $M\mathbf{x} = \mathbf{b}$. This CG method is built on the inner product $(\mathbf{u},\mathbf{v})_G = \mathbf{v}^T G\mathbf{u}$.

ALGORITHM 2.1 (NspCG for M)

Input: System Matrix M, rhs vector $\mathbf{b}$, inner product matrix G, initial guess $\mathbf{x}_0$,

Initialize: $\mathbf{r}_0 = b - M\mathbf{x}_0$, $\mathbf{p}_0 = \mathbf{r}_0$.

for $i = 0, 1, \ldots$ until convergence **do**

$$\alpha_i = \frac{(\mathbf{x} - \mathbf{x}_i, \mathbf{p}_i)_G}{(\mathbf{p}_i, \mathbf{p}_i)_G} \tag{2.12}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i\mathbf{p}_i \tag{2.13}$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i M\mathbf{p}_i \tag{2.14}$$

$$\mathbf{y}_{i+1} = M\mathbf{r}_{i+1} \tag{2.15}$$

$$\beta_{i+1} = -\frac{(\mathbf{r}_{i+1}, \mathbf{p}_i)_G}{(\mathbf{p}_i, \mathbf{p}_i)_G} \tag{2.16}$$

$$\mathbf{p}_{i+1} = \mathbf{r}_{i+1} + \beta_{i+1}\mathbf{p}_i \tag{2.17}$$

endfor

Liesen and Parlett [12] suggested that the inner product matrix $G = \mathcal{M}(\gamma)M$ gives a working conjugate gradient method, as seen from the following lemma.

Lemma 2.3.1. *Suppose that the symmetric matrix $\mathcal{M}(\gamma)$ is positive definite. Then Algorithm 2.1 is well defined for M and $G = \mathcal{M}(\gamma)M$, and (until convergence) the scalars α_i and β_{i+1} can be computed as*

$$\alpha_i = \frac{(\mathbf{r}_i, \mathbf{r}_i)_{\mathcal{M}(\gamma)}}{(M\mathbf{p}_i, \mathbf{p}_i)_{\mathcal{M}(\gamma)}} \tag{2.18}$$

$$\beta_i = \frac{(\mathbf{r}_{i+1}, \mathbf{r}_{i+1})_{\mathcal{M}(\gamma)}}{(M\mathbf{r}_i, \mathbf{r}_i)_{\mathcal{M}(\gamma)}}. \tag{2.19}$$

The proof of Lemma 2.3.1 is given in [12].

It can be shown that choosing G in this way gives residuals that are in some sense orthogonal, when Algorithm 2.1 is used.

Theorem 2.3.2. *Each residual $\mathbf{r}_k$ as defined in Algorithm 2.1 is orthogonal to all previous residuals with respect to $\mathcal{M}(\gamma)$, i.e. $(\mathbf{r}_i^T, \mathbf{r}_j)_{\mathcal{M}(\gamma)} = 0$, where $i \neq j$. (For proof of this, see [16]).*

2.4 An Efficient Implementation

Following the idea from [12, 3.3], we present a much more efficient implementation of Algorithm 2.1. In step i of Algorithm 2.1 with M and $G = \mathcal{M}(\gamma)M$, we can compute the scalars α_i and β_{i+1} as shown in (2.18) and (2.19), respectively. Now we shall demonstrate how to substitute the $\mathcal{M}(\gamma)$-inner products with $\mathcal{J}$-bilinear forms. Since from (2.2),

$$\mathcal{M}(\gamma) = \mathcal{J}M - \gamma\mathcal{J},$$

we have for all vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{n+m}$,

$$(\mathbf{u}, \mathbf{v})_{\mathcal{M}(\gamma)} = (M\mathbf{u}, \mathbf{v})_{\mathcal{J}} - \gamma(\mathbf{u}, \mathbf{v})_{\mathcal{J}}.$$

Therefore,

$$(\mathbf{r}_i, \mathbf{r}_i)_{\mathcal{M}(\gamma)} = (M\mathbf{r}_i, \mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{r}_i, \mathbf{r}_i)_{\mathcal{J}}, \tag{2.20}$$

and, since M is $\mathcal{M}(\gamma)$-symmetric,

$$(M\mathbf{p}_i, \mathbf{p}_i)_{\mathcal{M}(\gamma)} = (\mathbf{p}_i, M\mathbf{p}_i)_{\mathcal{M}(\gamma)} = (M\mathbf{p}_i, M\mathbf{p}_i)_{\mathcal{J}} - \gamma(\mathbf{p}_i, M\mathbf{p}_i)_{\mathcal{J}}. \tag{2.21}$$

Thus to compute α_i and β_{i+1}, the main work is evaluating the bilinear form $(\mathbf{u},\mathbf{v})_{\mathcal{J}}$, which is not more costly than evaluating the Euclidean inner product $(\mathbf{u},\mathbf{v})$ [12]. It is essential that both $M\mathbf{r}_i$ and $M\mathbf{p}_i$ are readily available. So, to avoid the unnecessary expense of computing both matrix-vector products in every iteration, two additional vectors are stored, namely $\mathbf{y}_i = M\mathbf{r}_i$ and $\mathbf{w}_i = M\mathbf{p}_i$. The former is calculated by multiplying M on the right by $\mathbf{r}_i$. The latter is computed via an additional recursion in the following manner: Multiplying (2.17) on the left by M yields

$$\underbrace{M\mathbf{p}_{i+1}}_{\mathbf{w}_{i+1}} = \underbrace{M\mathbf{r}_{i+1}}_{\mathbf{y}_{i+1}} + \beta_{i+1}\underbrace{M\mathbf{p}_i}_{\mathbf{w}_i}$$

The complete algorithm is given below.

ALGORITHM 2.2 (Efficient NspCG for M)

Input: System Matrix M, rhs vector $\mathbf{b}$, real parameter γ, initial guess $\mathbf{x}_0$,

Initialize: $\mathbf{r}_0 = \mathbf{b} - M\mathbf{x}_0$, $\mathbf{p}_0 = \mathbf{r}_0$, $\mathbf{y}_0 = M\mathbf{r}_0$, $\mathbf{w}_0 = \mathbf{y}_0$,

for $i = 0, 1, \ldots$ until convergence **do**

$$\alpha_i = \frac{(\mathbf{y}_i,\mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{r}_i,\mathbf{r}_i)_{\mathcal{J}}}{(\mathbf{w}_i,\mathbf{w}_i)_{\mathcal{J}} - \gamma(\mathbf{p}_i,\mathbf{w}_i)_{\mathcal{J}}} \tag{2.22}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i\mathbf{p}_i \tag{2.23}$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i\mathbf{w}_i \tag{2.24}$$

$$\mathbf{y}_{i+1} = M\mathbf{r}_{i+1} \tag{2.25}$$

$$\beta_{i+1} = \frac{(\mathbf{y}_{i+1},\mathbf{r}_{i+1})_{\mathcal{J}} - \gamma(\mathbf{r}_{i+1},\mathbf{r}_{i+1})_{\mathcal{J}}}{(\mathbf{y}_i,\mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{r}_i,\mathbf{r}_i)_{\mathcal{J}}} \tag{2.26}$$

$$\mathbf{p}_{i+1} = \mathbf{r}_{i+1} + \beta_{i+1}\mathbf{p}_i \tag{2.27}$$

$$\mathbf{w}_{i+1} = \mathbf{y}_{i+1} + \beta_{i+1}\mathbf{w}_i \tag{2.28}$$

endfor

The denominator of β_{i+1} is the same as the numerator of α_i, so this quantity has to be computed only once per iteration. When we store these scalars, each step of Algorithm 2.2 requires four evaluations of the bilinear form $(\mathbf{u},\mathbf{v})_{\mathcal{J}}$, compared to two evaluations of the inner product $(\mathbf{u},\mathbf{v})_G$ in the NspCG method specified in Algorithm 2.1. Additionally, Algorithm 2.2 requires two more vectors of storage and one more recurrence, namely (2.28), than Algorithm 2.1.

Chapter 3

PRECONDITIONING FOR M

In numerical experiments using Algorithm 2.2, the NspCG method displays both encouraging and worrying behavior when compared with other iterative methods for solving nonsymmetric systems. NspCG required only few iterations to significantly reduce the residual norm but then often stagnates due to near-parallelism of its residuals in the standard inner product. This is common with CG iteration without preconditioning. So, we expect that if preconditioning is used in conjunction with our NspCG method, as is typically done with a CG method, it will dramatically accelerate convergence. In this chapter, we develop an effective preconditioner for our matrix M, using the idea first proposed by [15].

3.1 Preconditioner for M Based on its Schur Complement

As proposed by Murphy, Golub, and Wathen [15], we consider a block-diagonal preconditioner based on the Schur complement. Let the system matrix

$$M = \begin{bmatrix} A^T W A & A^T \\ -A & 0 \end{bmatrix}$$

be preconditioned by

$$C = \begin{bmatrix} A^T W A & 0 \\ 0 & -A(A^T W A)^{-1} A^T \end{bmatrix}.$$

For convenience, we let $B = A^T W A$, which simplifies C to

$$C = \begin{bmatrix} B & 0 \\ 0 & -AB^{-1}A^T \end{bmatrix} \tag{3.1}$$

We can also simplify the $(2,2)$ block of C as follows:

$$-AB^{-1}A^T = -A(A^T W A)^{-1} A^T = -AA^{-1}W^{-1}A^{-T}A^T = -W^{-1}.$$

So, C becomes

$$C = \begin{bmatrix} B & 0 \\ 0 & -W^{-1} \end{bmatrix}.$$

This proposition addresses left preconditioning, but we are interested in centered preconditioning, so we find C_1, C_2 such that

$$C_1 C_2 = C.$$

Then our preconditioned matrix becomes

$$\tilde{M} = C_1^{-1} M C_2^{-1}.$$

To achieve this, we assume that C has an LU factorization

$$C = \begin{bmatrix} B & 0 \\ 0 & -W^{-1} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}.$$

Solving this, we see

$$L_{11} U_{11} = B,$$

$$L_{11} U_{12} = 0 \Longrightarrow U_{12} = 0,$$

$$L_{21} U_{11} = 0 \Longrightarrow L_{21} = 0,$$

$$L_{21} U_{12} + L_{22} U_{22} = -W^{-1} \Longrightarrow L_{22} U_{22} = -W^{-1}.$$

Since B and W are symmetric positive definite, there exists a Cholesky factorization for B and W. Let $B = R^T R$ be the Cholesky factorization for B, then

$$L_{11} = R^T \quad \text{and} \quad U_{11} = R.$$

Also, since $W = wI$, let $w = s^2$, then $W = s^2 I$ and

$$L_{22} = -s^{-1} I \quad \text{and} \quad U_{22} = s^{-1} I.$$

It follows that

$$\underbrace{\begin{bmatrix} B & 0 \\ 0 & -W^{-1} \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} R^T & 0 \\ 0 & -s^{-1} I \end{bmatrix}}_{C_1} \underbrace{\begin{bmatrix} R & 0 \\ 0 & s^{-1} I \end{bmatrix}}_{C_2}. \tag{3.2}$$

We can now solve the preconditioned system

$$\tilde{M} \tilde{\mathbf{x}} = \tilde{\mathbf{b}} \tag{3.3}$$

where

$$\begin{aligned} \tilde{M} &= C_1^{-1} M C_2^{-1}, \\ \tilde{\mathbf{x}} &= C_2 \mathbf{x}, \\ \tilde{\mathbf{b}} &= C_1^{-1} \mathbf{b}. \end{aligned}$$

As remarked in [15, Remark 3], any Krylov subspace iterative method with an optimality condition or Galerkin property will terminate in at most three iterations with the solution to $M\mathbf{x} = \mathbf{b}$ if the preconditioner (3.1) is used. However, this preconditioner is more expensive than direct solution by block elimination. Thus, one typically uses approximations to B^{-1} and $(-AB^{-1}A^T)^{-1}$ [5].

Typically, such approximations are derived from a splitting of the $(1,1)$-block, $B = D - E$, where D can be efficiently inverted. In our case, we let $E = 0$ and $D = LL^T$ be the incomplete Cholesky factorization of B. Since the $(2,2)$-block simplifies nicely to W^{-1}, we let that be the approximation of $(-AB^{-1}A^T)^{-1}$. (3.2) then becomes

$$\underbrace{\begin{bmatrix} B & 0 \\ 0 & -W^{-1} \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} L & 0 \\ 0 & -s^{-1}I \end{bmatrix}}_{C_1} \underbrace{\begin{bmatrix} L^T & 0 \\ 0 & s^{-1}I \end{bmatrix}}_{C_2}. \tag{3.4}$$

The Incomplete Cholesky factor L is computed using the MATLAB `ichol` function with threshold dropping and diagonal compensation. The exact MATLAB command is `L=ichol(B, struct('type','ict','droptol',tol,'diagcomp',`α_d`))` where `tol` and α_d are scalars. The incomplete Cholesky factorization of a symmetric positive definite matrix does not always exist. Diagonal compensation fixes this by creating an approximate diagonally dominant matrix since diagonally dominant matrices are guaranteed to have Incomplete Cholesky factorization.

The larger the drop tolerance, the more sparse the factor L tends to be, and the less expensive the preconditioner formed from L will be, but this also means slower convergence. So, there is a trade-off between sparsity (how expensive the preconditioner is) and convergence. Hence, it is important to choose the parameters `tol` and α_1 carefully. Different parameter values were tested until a balance was found between sparsity and convergence of our NspCG method. These values differ from problem to problem and are dependent on the size and properties of the matrix A. We found that using a drop tolerance value close the `rcond` number of A gives sparse preconditioners with a good convergence rate. The level of sparsity is also dependent on how well-conditioned A is and the size of A, with very poorly conditioned matrices or small matrices giving the least sparse preconditioners.

3.2 Modified NspCG Algorithm for the Preconditioned System

Algorithm 2.2 of Section 2.4 works with our non-preconditioned system. We need to modify Algorithm 2.2 to work with our preconditioned system. To do this, we multiply all

search directions by C_2^{-1} and all residuals by C_1^{-1}. We also need to define α_i and β_i for our preconditioned algorithm.

First, we will make a modification to preconditioner (3.1) by negating the $(2,2)$-block. (3.1) as proposed in [15] uses a standard inner product, but we do not have a standard inner product but an inner product in $\mathcal{J}$. So, this modification is necessary to establish a relationship between C_1, C_2, and $\mathcal{J}$, which will be useful in the derivation of α_i and β_i for our preconditioned algorithm. With this modification, (3.4) becomes

$$\underbrace{\begin{bmatrix} B & 0 \\ 0 & W^{-1} \end{bmatrix}}_{C} = \underbrace{\begin{bmatrix} L & 0 \\ 0 & s^{-1}I \end{bmatrix}}_{C_1} \underbrace{\begin{bmatrix} L^T & 0 \\ 0 & s^{-1}I \end{bmatrix}}_{C_2}. \tag{3.5}$$

Next, we compute the inverses of C_1, C_2, and C_1^{-T}

$$C_1^{-1} = \begin{bmatrix} L^{-1} & 0 \\ 0 & sI \end{bmatrix}, \quad C_2^{-1} = \begin{bmatrix} L^{-T} & 0 \\ 0 & sI \end{bmatrix} \quad \text{and} \quad C_1^{-T} = \begin{bmatrix} L^{-T} & 0 \\ 0 & sI \end{bmatrix}.$$

We see from this that $C_1^{-T} = C_2^{-1}$. Also, since premultiplying C_1^{-T} both on the left and right by $\mathcal{J}$ preserves C_1^{-T}, we have

$$\mathcal{J}C_1^{-T}\mathcal{J} = C_2^{-1}.$$

Since $\mathcal{J}^{-1} = \mathcal{J}$, we have this very useful relationship

$$C_1^{-T}\mathcal{J} = \mathcal{J}C_2^{-1} \tag{3.6}$$

Now, we can derive α_i and β_i for our preconditioned system. It suffices to show only the derivation of α_i, since β_i follows from α_i. This is so because the denominator of β_i is the same as the numerator of α_i. From (2.18)

$$\alpha_i = \frac{(\mathbf{r}_i, \mathbf{r}_i)_{\mathcal{M}}(\gamma)}{(M\mathbf{p}_i, \mathbf{p}_i)_{\mathcal{M}}(\gamma)}$$

For our preconditioned system, this becomes

$$\alpha_i = \frac{(C_1^{-1}\mathbf{r}_i, C_1^{-1}\mathbf{r}_i)_{\mathcal{M}(\gamma)}}{(C_1^{-1}M\mathbf{p}_i, \tilde{U}\mathbf{p}_i)_{\mathcal{M}(\gamma)}}$$

$$\mathcal{M}(\gamma) = \mathcal{J}\tilde{M} - \gamma\mathcal{J}$$

For the numerator,

$$\begin{aligned}(C_1^{-1}\mathbf{r}_i, C_1^{-1}\mathbf{r}_i)_{\mathcal{M}(\gamma)} &= \mathbf{r}_i^T C_1^{-T}\mathcal{M}(\gamma)C_1^{-1}\mathbf{r}_i \\ &= \mathbf{r}_i^T C_1^{-T}(\mathcal{J}\tilde{M} - \gamma\mathcal{J})C_1 - 1\mathbf{r}_i \\ &= \mathbf{r}_i^T C_1^{-T}\left(\mathcal{J}\tilde{M}C_1^{-1}\mathbf{r}_i - \gamma\mathcal{J}C_1^{-1}\mathbf{r}_i\right) \\ &= \mathbf{r}_i^T C_1^{-T}\mathcal{J}\tilde{M}C_1^{-1}\mathbf{r}_i - \gamma\mathbf{r}_i^T C_1^{-T}\mathcal{J}C_1^{-1}\mathbf{r}_i \\ &= \mathbf{r}_i^T \mathcal{J}C_2^{-1}\tilde{M}C_1^{-1}\mathbf{r}_i - \gamma\mathbf{r}_i^T \mathcal{J}C_2^{-1}C_1^{-1}\mathbf{r}_i\end{aligned}$$

But

$$C_2^{-1}\tilde{M}C_1^{-1} = C_2^{-1}C_1^{-1}MC_2^{-1}C_1^{-1} = C^{-1}MC^{-1}$$

So,

$$\begin{aligned}(C_1^{-1}\mathbf{r}_i, C_1^{-1}\mathbf{r}_i)_{\mathcal{M}(\gamma)} &= \mathbf{r}_i^T \mathcal{J}C^{-1}MC^{-1}\mathbf{r}_i - \gamma\mathbf{r}_i^T \mathcal{J}C^{-1}\mathbf{r}_i \\ &= \mathbf{r}_i^T \mathcal{J}C^{-1}M\mathbf{z}_i - \gamma\mathbf{r}_i^T \mathcal{J}\mathbf{z}_i \\ &= \mathbf{r}_i^T \mathcal{J}C^{-1}\mathbf{y}_i - \gamma\mathbf{r}_i^T \mathcal{J}\mathbf{z}_i \\ &= (C^{-1}\mathbf{y}_i, \mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{z}_i, \mathbf{r}_i)_{\mathcal{J}}\end{aligned}$$

For the denominator,

$$(C_1^{-1}M\mathbf{p}_i, C_2\mathbf{p}_i)_{\mathcal{M}(\gamma)} = (C_2\mathbf{p}_i, C_1^{-1}M\mathbf{p}_i)_{\mathcal{M}(\gamma)}$$

since $\tilde{M}$ is $\mathcal{M}(\gamma)$-symmetric,

$$\begin{aligned}(C_2\mathbf{p}_i, C_1^{-1}M\mathbf{p}_i)_{\mathcal{M}(\gamma)} &= \mathbf{p}_i^T M^T C_1^{-T}\mathcal{M}(\gamma)C_2\mathbf{p}_i \\ &= \mathbf{p}_i^T M^T C_1^{-T}(\mathcal{J}\tilde{M} - \gamma\mathcal{J})C_2\mathbf{p}_i \\ &= \mathbf{p}_i^T M^T C_1^{-T}\left(\mathcal{J}\tilde{M}C_2\mathbf{p}_i - \gamma\mathcal{J}C_2\mathbf{p}_i\right) \\ &= \mathbf{p}_i^T M^T C_1^{-T}\mathcal{J}\tilde{M}C_2\mathbf{p}_i - \gamma\mathbf{p}_i^T M^T C_1^{-T}\mathcal{J}C_2\mathbf{p}_i \\ &= \mathbf{p}_i^T M^T \mathcal{J}C_2^{-1}\tilde{M}C_2\mathbf{p}_i - \gamma\mathbf{p}_i^T M^T \mathcal{J}C_2^{-1}C_2\mathbf{p}_i\end{aligned}$$

But,

$$C_2^{-1}\tilde{M}C_2 = C_2^{-1}C_1^{-1}MC_2^{-1}C_2 = C^{-1}M$$

So,

$$\begin{aligned}(C_2\mathbf{p}_i, C_1^{-1}M\mathbf{p}_i)_{\mathcal{M}(\gamma)} &= \mathbf{p}_i^T M^T \mathcal{J}C^{-1}M\mathbf{p}_i - \gamma\mathbf{p}_i^T M^T \mathcal{J}\mathbf{p}_i \\ &= (C^{-1}M\mathbf{p}_i, M\mathbf{p}_i)_{\mathcal{J}} - \gamma(\mathbf{p}_i, M\mathbf{p}_i)_{\mathcal{J}} \\ &= (C^{-1}\mathbf{w}_i, \mathbf{w}_i)_{\mathcal{J}} - \gamma(\mathbf{p}_i, \mathbf{w}_i)_{\mathcal{J}}\end{aligned}$$

Therefore, α_i for our preconditioned system becomes

$$\alpha_i = \frac{(C^{-1}\mathbf{y}_i, \mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{z}_i, \mathbf{r}_i)_{\mathcal{J}}}{(C^{-1}\mathbf{w}_i, \mathbf{w}_i)_{\mathcal{J}} - \gamma(\mathbf{p}_i, \mathbf{w}_i)_{\mathcal{J}}} \tag{3.7}$$

We can also use (2.22) from Chapter 2 and arrive at the same α_i for our preconditioned system. (2.22) is given as

$$\alpha_i = \frac{(\mathbf{y}_i, \mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{r}_i, \mathbf{r}_i)_{\mathcal{J}}}{(\mathbf{w}_i, \mathbf{w}_i)_{\mathcal{J}} - \gamma(\mathbf{p}_i, \mathbf{w}_i)_{\mathcal{J}}}$$

So, for our preconditioned system, we have

$$\alpha_i = \frac{(C_1^{-1}\mathbf{y}_i, C_1^{-1}\mathbf{r}_i)_{\mathcal{J}} - \gamma(C_1^{-1}\mathbf{r}_i, C_1^{-1}\mathbf{r}_i)_{\mathcal{J}}}{(C_1^{-1}\mathbf{w}_i, C_1^{-1}\mathbf{w}_i)_{\mathcal{J}} - \gamma(C_1\mathbf{p}_i, C_1^{-1}\mathbf{w}_i)_{\mathcal{J}}}.$$

Consider the first term of the numerator of the above equation, we have

$$\begin{aligned}
(C_1^{-1}\mathbf{y}_i, C_1^{-1}\mathbf{r}_i)_{\mathcal{J}} &= \mathbf{r}_i^T C_1^{-T} \mathcal{J} C_1^{-1}\mathbf{y}_i \\
&= \mathbf{r}_i^T \mathcal{J} C_2^{-1} C_1^{-1}\mathbf{y}_i \\
&= \mathbf{r}_i^T \mathcal{J} C^{-1}\mathbf{y}_i \\
&= (C^{-1}\mathbf{y}_i, \mathbf{r}_i)_{\mathcal{J}}
\end{aligned}$$

In a similar manner, the second term of the numerator becomes

$$(C_1^{-1}\mathbf{r}_i, C_1^{-1}\mathbf{r}_i)_{\mathcal{J}} = (C_1^{-1}\mathbf{r}_i, \mathbf{r}_i)_{\mathcal{J}} = (\mathbf{z}_i, \mathbf{r}_i)_{\mathcal{J}},$$

and the first term of the denominator becomes

$$(C_1^{-1}\mathbf{w}_i, C_1^{-1}\mathbf{w}_i)_{\mathcal{J}} = (C^{-1}\mathbf{w}_i, \mathbf{w}_i)_{\mathcal{J}}.$$

Finally, the last term of the denominator can be simplified as follows:

$$\begin{aligned}
(C_2\mathbf{p}_i, C_1^{-1}\mathbf{w}_i)_{\mathcal{J}} &= \mathbf{w}_i^T C_1^{-T} \mathcal{J} C_2\mathbf{p}_i \\
&= \mathbf{w}_i^T \mathcal{J} C_2^{-1} C_2\mathbf{p}_i \\
&= \mathbf{w}_i^T \mathcal{J}\mathbf{p}_i \\
&= (\mathbf{p}_i, \mathbf{w}_i)_{\mathcal{J}}
\end{aligned}$$

So, putting it all together, we have

$$\alpha_i = \frac{(C^{-1}\mathbf{y}_i, \mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{z}_i, \mathbf{r}_i)_{\mathcal{J}}}{(C^{-1}\mathbf{w}_i, \mathbf{w}_i)_{\mathcal{J}} - \gamma(\mathbf{p}_i, \mathbf{w}_i)_{\mathcal{J}}}. \tag{3.8}$$

(3.7) and (3.8) are the same, which implies that using (2.18) or (2.22) gives the same α_i. It also implies that we need to solve three systems viz

$$C\mathbf{z} = \mathbf{r},$$

$$C\mathbf{u} = \mathbf{y},$$

and

$$C\mathbf{v} = \mathbf{w}.$$

Our algorithm for the preconditioned matrix becomes

$$\begin{aligned}
\alpha_i &= \frac{(\mathbf{u}_i, \mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{z}_i, \mathbf{r}_i)_{\mathcal{J}}}{(\mathbf{v}_i, \mathbf{w}_i)_{\mathcal{J}} - \gamma(\mathbf{p}_i, \mathbf{w}_i)_{\mathcal{J}}} && (3.9)\\
\mathbf{x}_{i+1} &= \mathbf{x}_i + \alpha_i \mathbf{p}_i && (3.10)\\
\mathbf{r}_{i+1} &= \mathbf{r}_i - \alpha_i \mathbf{w}_i && (3.11)\\
C\mathbf{z}_{i+1} &= \mathbf{r}_{i+1} && (3.12)\\
\mathbf{y}_{i+1} &= M\mathbf{z}_{i+1} && (3.13)\\
C\mathbf{u}_{i+1} &= \mathbf{y}_{i+1} && (3.14)\\
\beta_{i+1} &= \frac{(\mathbf{u}_{i+1}, \mathbf{r}_{i+1})_{\mathcal{J}} - \gamma(\mathbf{z}_i, \mathbf{r}_i)_{\mathcal{J}}}{(\mathbf{u}_i, \mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{z}_i, \mathbf{r}_i)_{\mathcal{J}}} && (3.15)\\
\mathbf{p}_{i+1} &= \mathbf{z}_{i+1} + \beta_{i+1}\mathbf{p}_i && (3.16)\\
\mathbf{w}_{i+1} &= \mathbf{y}_{i+1} + \beta_{i+1}\mathbf{w}_i && (3.17)\\
C\mathbf{v}_{i+1} &= \mathbf{w}_{i+1} && (3.18)
\end{aligned}$$

Instead of solving three linear systems as the above algorithm indicates, we can instead solve only one linear system. To do this, we premultiply (3.11) and (3.17) by C^{-1}

$$\underbrace{C^{-1}\mathbf{r}_{i+1}}_{\mathbf{z}_{i+1}} = \underbrace{C^{-1}\mathbf{r}_i}_{\mathbf{z}_i} - \underbrace{\alpha_i C^{-1}\mathbf{w}_i}_{\alpha_i \mathbf{v}_i}$$

$$\underbrace{C^{-1}\mathbf{w}_{i+1}}_{\mathbf{v}_{i+1}} = \underbrace{C^{-1}\mathbf{y}_{i+1}}_{\mathbf{u}_{i+1}} + \underbrace{\beta_{i+1} C^{-1}\mathbf{w}_i}_{\beta_{i+1}\mathbf{v}_i}$$

The complete algorithm looks as follows.

ALGORITHM 3.1 (Preconditioned NspCG for M)

Input: System Matrix M, rhs vector $\mathbf{b}$, real parameter γ, initial guess $\mathbf{x}_0$, and preconditioner

matrix C

Initialize: $\mathbf{r}_0 = \mathbf{b} - M\mathbf{x}_0$, solve:$C\mathbf{z}_0 = \mathbf{r}_0$, $\mathbf{p}_0 = \mathbf{z}_0$, $\mathbf{y}_0 = M\mathbf{z}_0$, $\mathbf{w}_0 = \mathbf{y}_0$, solve:$C\mathbf{u}_0 = \mathbf{y}_0$, $\mathbf{v}_0 = \mathbf{u}_0$

for $i = 0, 1, \ldots$ until convergence **do**

$$\alpha_i = \frac{(\mathbf{u}_i, \mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{z}_i, \mathbf{r}_i)_{\mathcal{J}}}{(\mathbf{v}_i, \mathbf{w}_i)_{\mathcal{J}} - \gamma(\mathbf{p}_i, \mathbf{w}_i)_{\mathcal{J}}} \tag{3.19}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i \tag{3.20}$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i \mathbf{w}_i \tag{3.21}$$

$$\mathbf{z}_{i+1} = \mathbf{z}_i - \alpha_i \mathbf{v}_i \tag{3.22}$$

$$\mathbf{y}_{i+1} = M\mathbf{z}_{i+1} \tag{3.23}$$

$$C\mathbf{u}_{i+1} = \mathbf{y}_{i+1} \tag{3.24}$$

$$\beta_{i+1} = \frac{(\mathbf{u}_{i+1}, \mathbf{r}_{i+1})_{\mathcal{J}} - \gamma(\mathbf{z}_{i+1}, \mathbf{r}_{i+1})_{\mathcal{J}}}{(\mathbf{u}_i, \mathbf{r}_i)_{\mathcal{J}} - \gamma(\mathbf{z}_i, \mathbf{r}_i)_{\mathcal{J}}} \tag{3.25}$$

$$\mathbf{p}_{i+1} = \mathbf{z}_{i+1} + \beta_{i+1} \mathbf{p}_i \tag{3.26}$$

$$\mathbf{w}_{i+1} = \mathbf{y}_{i+1} + \beta_{i+1} \mathbf{w}_i \tag{3.27}$$

$$\mathbf{v}_{i+1} = \mathbf{u}_{i+1} + \beta_{i+1} \mathbf{v}_i \tag{3.28}$$

endfor

Chapter 4

ANALYSIS OF NUMERICAL RESULTS

In this chapter, we will analyze the results from our NspCG method with and without preconditioning. All problems were tested using a stopping criterion of $\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} < 10^{-10}$. We set the maximum number of iterations allowed for each of our tests problems to be 3000. The reason for this is explained below.

The conjugate gradient method, in exact arithmetic, is guaranteed to converge to the exact solution of $A\mathbf{x} = \mathbf{b}$ in at most n iterations [10]. Since our method (a conjugate gradient-like method) creates a matrix M that is $2n \times 2n$, we would naturally expect it to converge to the exact solution of $M\mathbf{x} = \mathbf{b}$ in at most $2n$ iterations, but this was not the case when our NspCG method was used without preconditioning due to stagnation occurring after a few iterations. So, we were interested in seeing how well the method will do if the number of iterations allowed was greater than $2n$. Will the method eventually converge? And if so, in how many iterations? After several tests, and in trying to stay consistent throughout all our test problems, we settled for 3000. Since our largest test problem (SHERMAN4) had $2n = 2060$, 3000 seemed a reasonable number since it satisfies our requirement of being greater than $2n$.

4.1 Example 1

For this example, we used a random matrix created in MATLAB using the MATLAB command `A=sprand(n,n,0.1)+speye(n)`, where `n=100`. This command creates a random sparse $n \times n$ matrix, where 0.1 is the density of uniformly distributed nonzero entries and adds this to the identity matrix.

In Figure 4.1, we see that NspCG without preconditioning shows a rapid decrease in the residual norm during the first few iterations but stagnates after about 35 iterations. Then, there was another rapid decrease in the residual norm after about 150 iterations, with the method converging to the specified tolerance in 315 iterations. On the other hand, NspCG with preconditioning converged rapidly, and the speed of convergence was dependent on how expensive the preconditioner was.

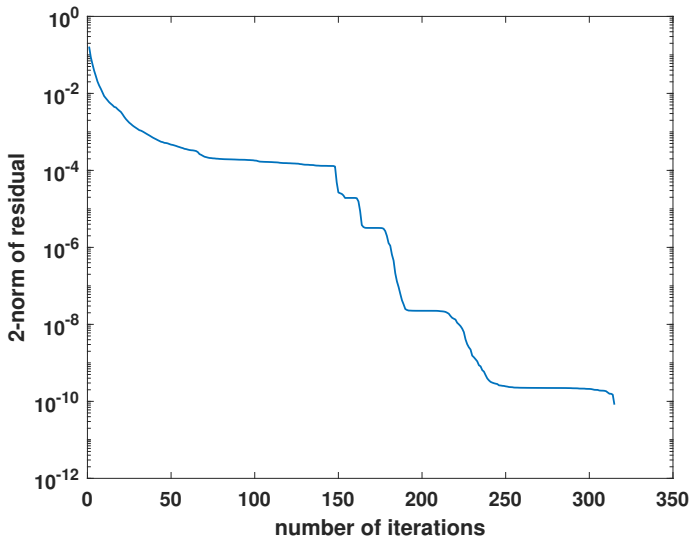

Figure 4.1: *Example 1: NspCG for a sparse matrix of dimension 100 without preconditioning*

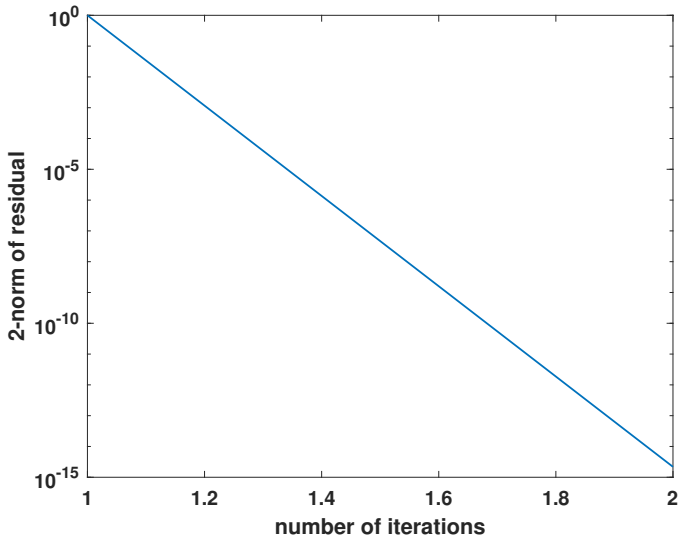

Figure 4.2: *Example 1: NspCG for a sparse matrix of dimension 100 using Schur complement preconditioner with true Cholesky factor*

In Figure 4.2, preconditioner (3.1) was used, and the solution converged in 2 iterations to the specified tolerance. Recall from [15] that preconditioner (3.1) is guaranteed to terminate in at most three iterations. Unfortunately, this preconditioner is prohibitively expensive, so we consider less expensive preconditioners with a good rate of convergence.

In Figure 4.3, preconditioner (3.5) was used. Our NspCG method converged to the

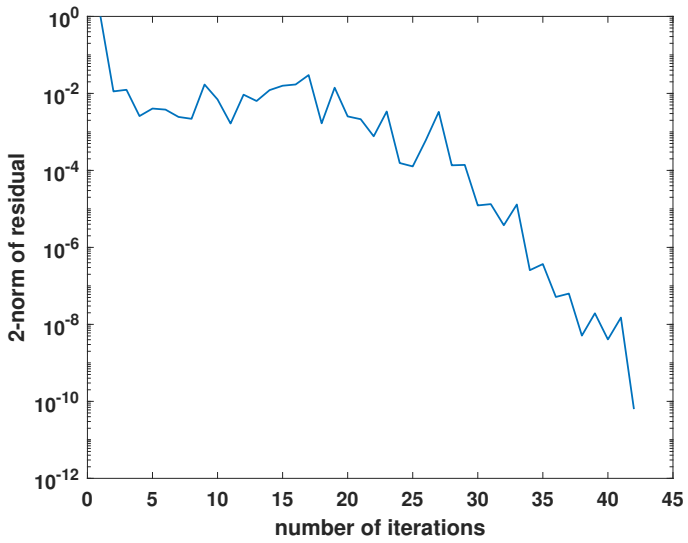

Figure 4.3: *Example 1: NspCG for a sparse matrix of dimension 100 using Schur complement preconditioner with Incomplete Cholesky factor (droptol:1e-3, α_1:1e-2)*

specified tolerance in about 42 iterations. This is a big improvement to NspCG without preconditioning. More importantly, NspCG with preconditioning fixed the issue of stagnation experienced when the method was used without preconditioning.

4.2 Example 2

For this example, we used the oil reservoir modeling matrix ORSIRR_1 gotten from the Matrix Market collection. This matrix can be obtained from `http://math.nist.gov/MatrixMarket/`.

Again, as seen in Figure 4.4, NspCG without preconditioning shows a rapid decrease in the 2-norm of residuals within the first 200 iterations but stagnates after that, never reaching the specified tolerance within the maximum number of iterations allowed (3000 iterations).

Preconditioning fixed this stagnation. In Figure 4.5, convergence to the specified tolerance is achieved in 252 iterations, which is still very good compared to NspCG without precondtioning that failed to converge in 3000 iterations. The preconditioner used in Figure 4.5 is the cheapest of the three preconditioners of the form (3.5) tested that still achieve a good rate of convergence. The results for the remaining two preconditioners are shown in Figure 4.6 and Figure 4.7, with the latter being the most expensive of the three and achieving the best convergence rate.

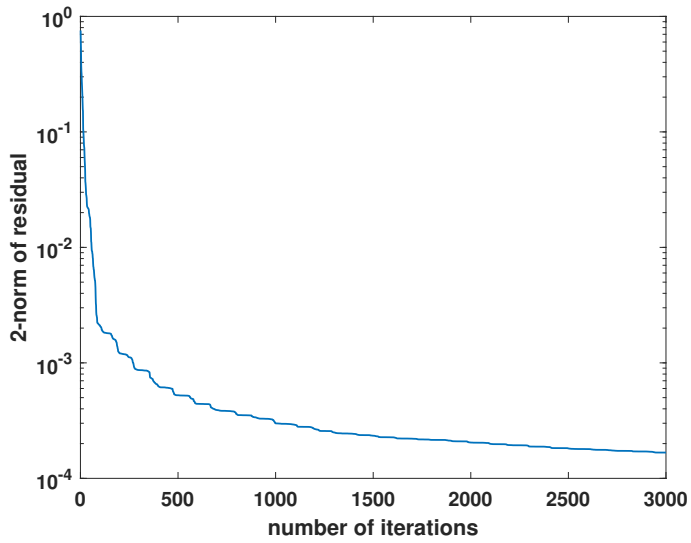

Figure 4.4: *Example 2: NspCG for the matrix ORSIRR_1 without preconditioning*

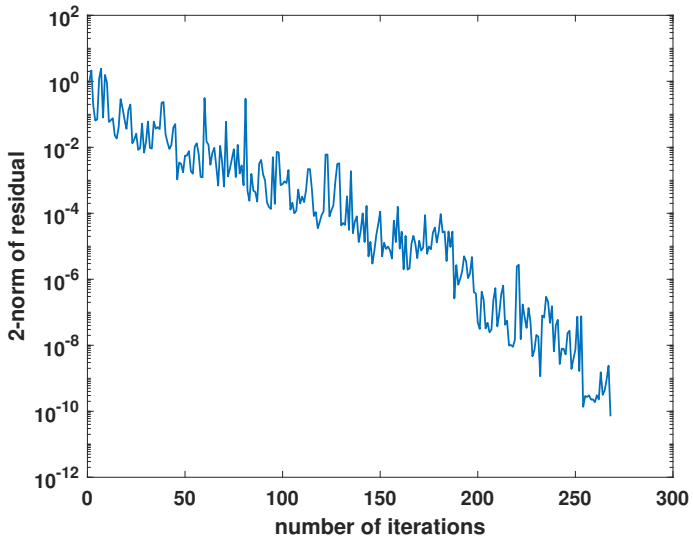

Figure 4.5: *Example 2: NspCG for the matrix ORSIRR_1 using Schur complement preconditioner with Incomplete Cholesky factor (droptol:1e-6, α_1:1e-5)*

4.3 Example 3

Example 3 uses the oil reservoir modeling matrix SHERMAN4 obtained from the Matrix Market collection.

Again, we see stagnation, but as seen in Figure 4.8, NspCG without preconditioning did

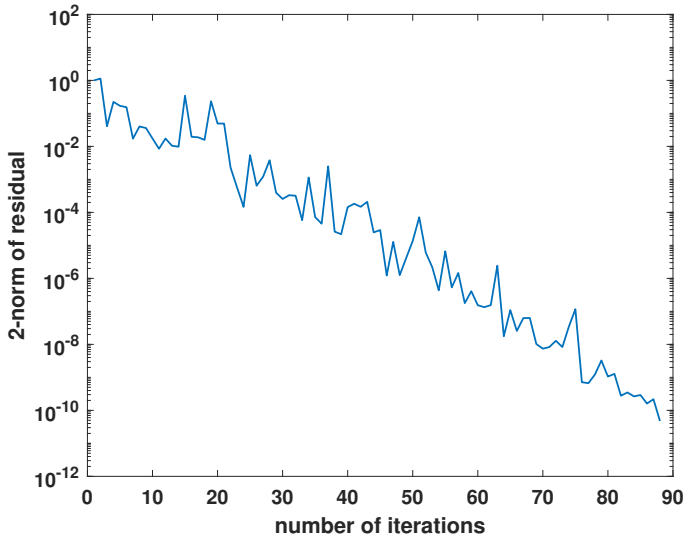

Figure 4.6: *Example 2: NspCG for the matrix ORSIRR_*1 *using Schur complement preconditioner with Incomplete Cholesky factor (droptol:1e-7,* α_1*:1e-6)*

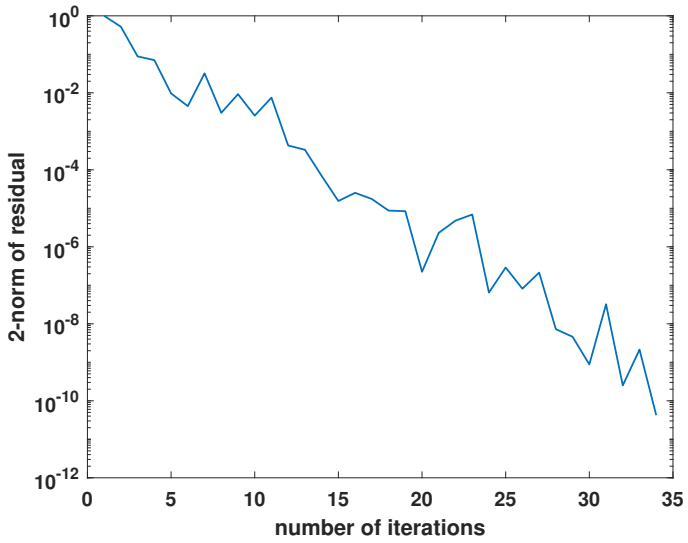

Figure 4.7: *Example 2: NspCG for the matrix ORSIRR_*1 *using Schur complement preconditioner with Incomplete Cholesky factor (droptol:1e-8,* α_1*:1e-7)*

eventually converge to the specified tolerance but did so after 1725 iterations. On the other hand, NspCG with preconditioning converged in 80 iterations, as seen in Figure 4.9.

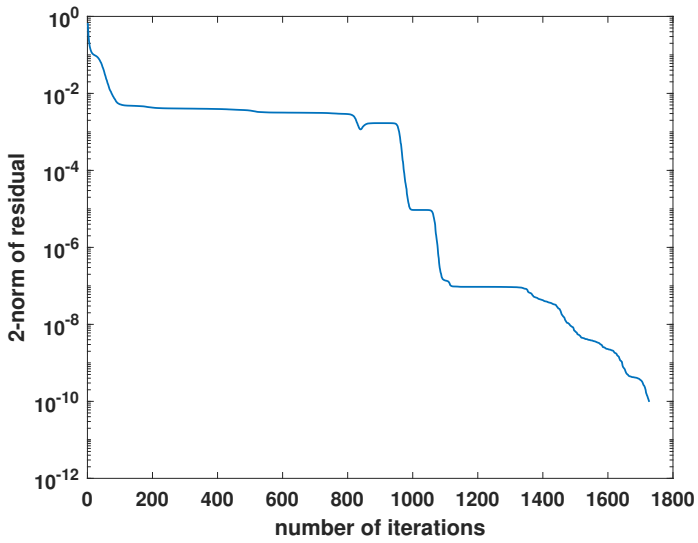

Figure 4.8: *Example 3: NspCG for the matrix SHERMAN*4 *without preconditioning*

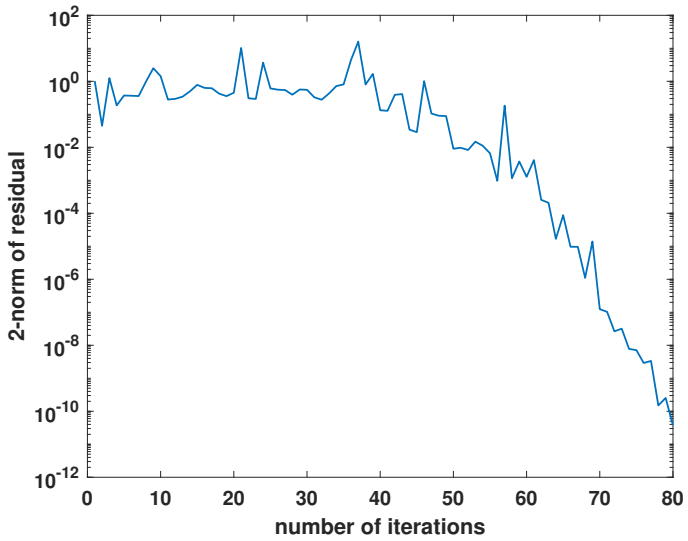

Figure 4.9: *Example 3: NspCG for the matrix SHERMAN*4 *using Schur complement preconditioner with Incomplete Cholesky factor (droptol:1e-3,* α_1*:1e-2)*

4.4 Example 4

Example 4 uses the UTM300 matrix from the Matrix Market collection.

Figure 4.10 shows the result of NspCG without preconditioning. In this case, despite a monotonically decreasing 2-norm of residuals, NspCG without preconditioning failed to

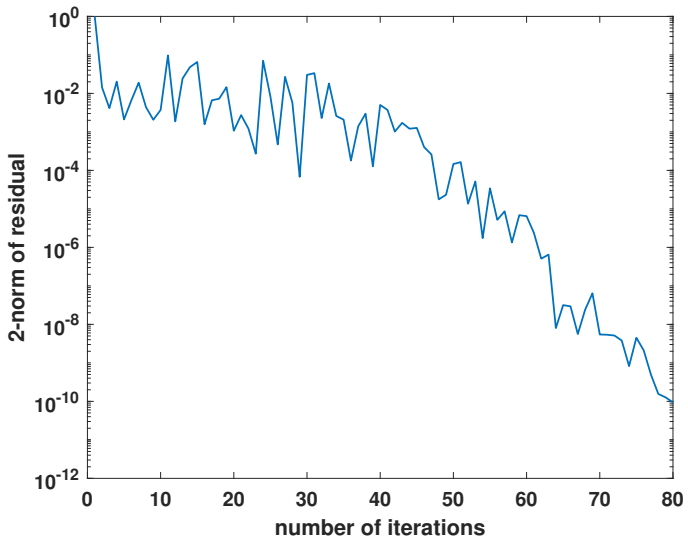

Figure 4.10: *Example 4: NspCG for the matrix UTM*300 *without preconditioning*

converge to the specified tolerance within the maximum iterations allowed (3000 iterations). This seems to be a common pattern when the condition number for A is large. In fact, for all tests problems with `cond(A)` of $O(10^4)$ or greater, NsPCG without preconditioning failed to converge to the specified tolerance within the maximum number of iterations allowed.

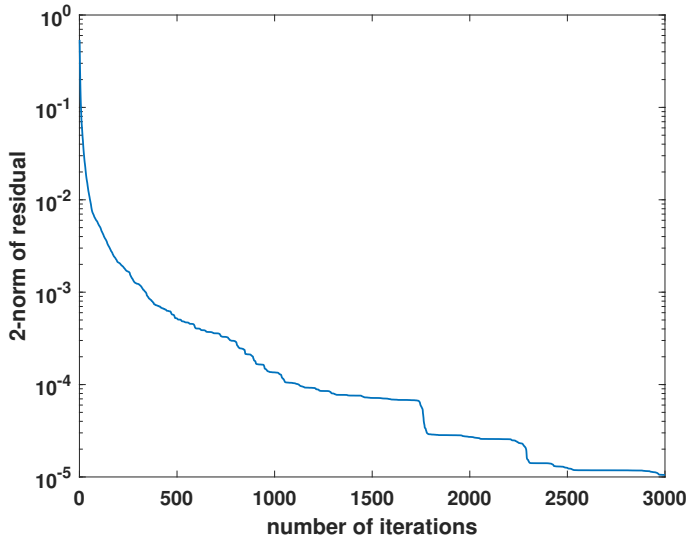

Figure 4.11: *Example 4: NspCG for the matrix UTM*300 *using Schur complement preconditioner with Incomplete Cholesky factor (droptol:1e-7,* α_1*:1e-6)*

Figure 4.11 shows the result of NspCG with preconditioning. This converged to the specified tolerance in 80 iterations.

4.5 Example 5

Example 5 uses the PDE900 matrix from the Matrix Market collection, which is a five-point central finite difference discretization of a two-dimensional variable-coefficient linear elliptic equation with Dirichlet boundary conditions.

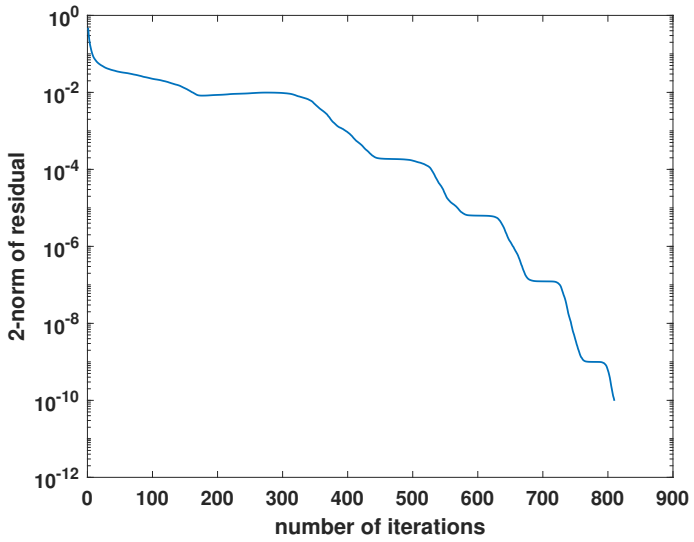

Figure 4.12: *Example 5: NspCG for the matrix PDE*900 *without preconditioning*

Figure 4.12 shows the result of NspCG without preconditioning. The method converged to the specified tolerance in 808 iterations. This seems to be a common pattern when the condition number for A is $O(10^3)$ or less. In fact, for all test problems with `cond(A)` of $O(10^3)$ or less, NspCG without preconditioning converged eventually to the specified tolerance within the maximum number of iterations allowed, despite showing stagnation. Figure 4.13 shows the result of NspCG with preconditioning. This converged to the specified tolerance in 64 iterations.

4.6 Example 6

Example 6 uses the STEAM1 matrix from the Matrix Market collection, which was extracted from a program simulating enhanced oil recovery using injected steam. Matrix STEAM1

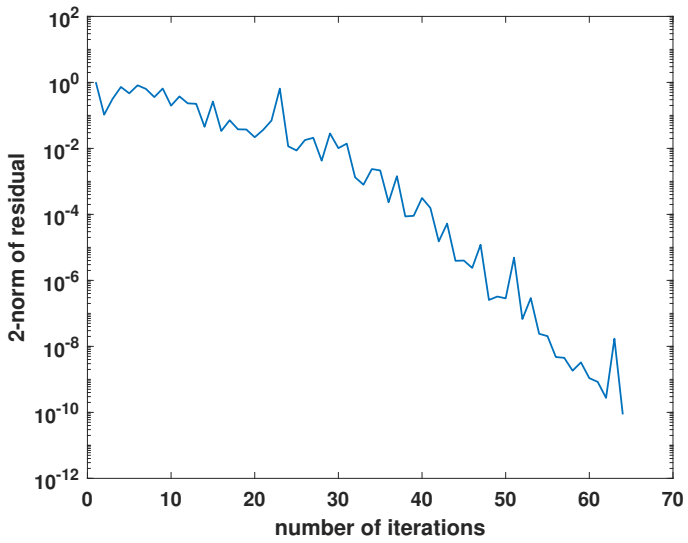

Figure 4.13: *Example 5: NspCG for the matrix PDE*900 *using Schur complement preconditioner with Incomplete Cholesky factor (droptol:1e-3,* α_1*:1e-2)*

represents a finite-difference discretization of a $4 \times 4 \times 5$ grid with 3 variables at each grid point.

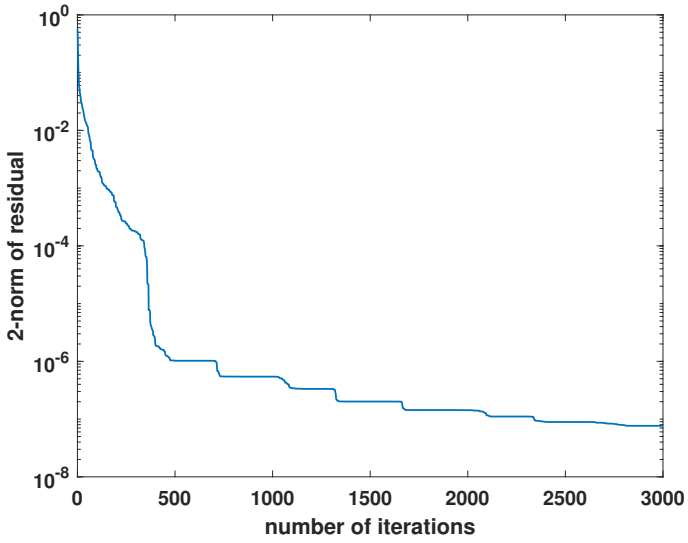

Figure 4.14: *Example 6: NspCG for the matrix STEAM*1 *without preconditioning*

Figure 4.14 shows the result of NspCG without preconditioning. In this case, stagnation occurred after about 500 iterations, and NspCG without preconditioning failed to converge

to the specified tolerance within the maximum iterations allowed (3000 iterations).

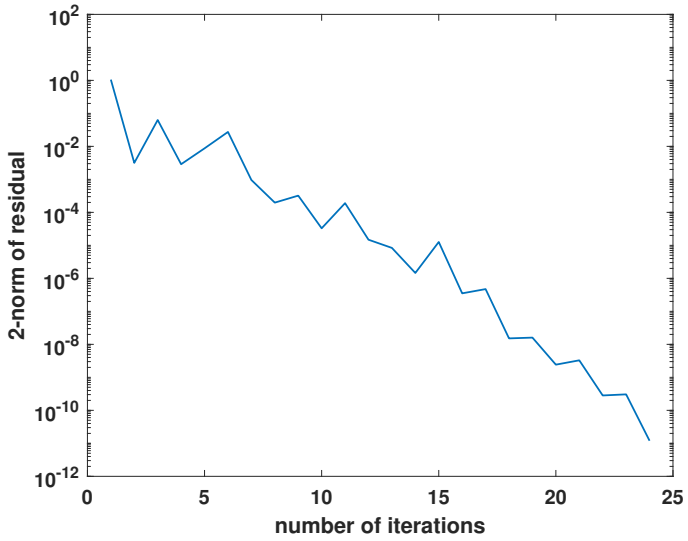

Figure 4.15: *Example 6: NspCG for the matrix STEAM1 using Schur complement preconditioner with Incomplete Cholesky factor (droptol:1e-9, α_1:1e-8)*

Figure 4.15 shows the result of NspCG with preconditioning. This converged to the specified tolerance in 25 iterations.

4.7 Example 7

Example 7 uses the CDDE5 matrix from the Matrix Market collection, which is a 5-point finite difference discretization of a constant-coefficient convection-diffusion equation on a uniform m × m grid.

Figure 4.16 shows the result of NspCG without preconditioning. The method failed to converge to the specified tolerance within the maximum iterations allowed (3000 iterations).

Figure 4.17 shows the result of NspCG with preconditioning. This converged to the specified tolerance in 128 iterations.

Table 4.1 shows the results for test matrices in Example $1-7$, where L is the incomplete Cholesky factor of the $(1,1)$-block of M, and R is the true Cholesky factor. The table also shows the choice of parameters `tol` for the drop tolerance and α_d for the diagonal compensation used in computing the incomplete Cholesky factor L. As can be seen from Table 4.1, a drop tolerance value close to the same order of magnitude as A gives good sparsity and good convergence. Also, when n was small, L was less sparse.

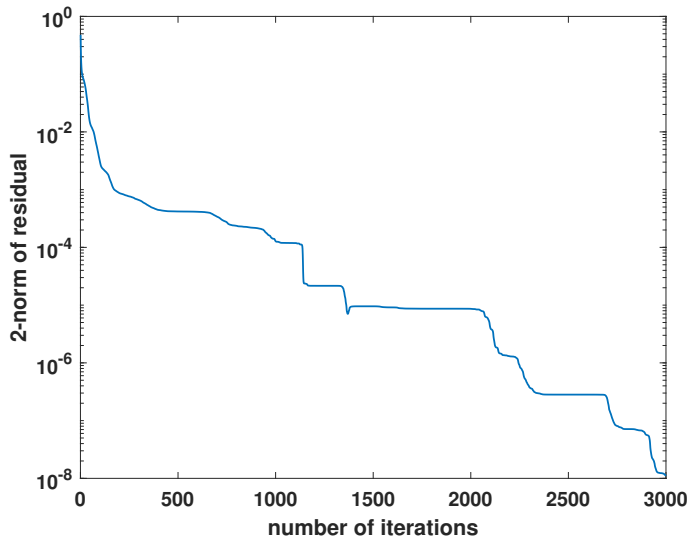

Figure 4.16: *Example 7: NspCG for the matrix CDDE5 without preconditioning*

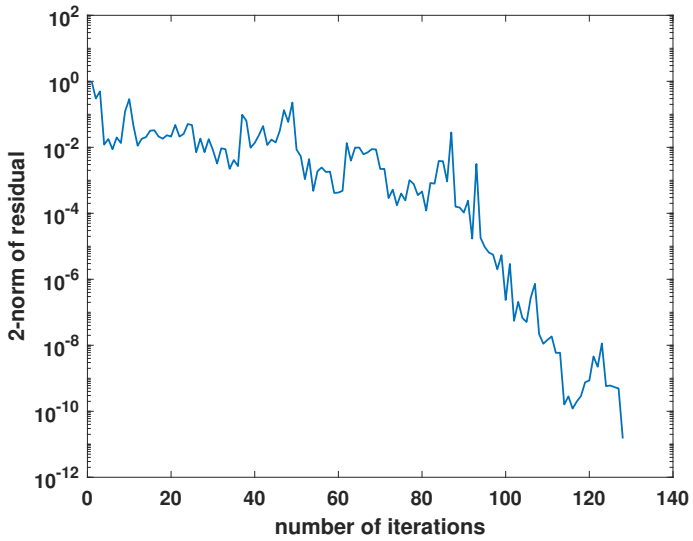

Figure 4.17: *Example 7: NspCG for the matrix CDDE5 using Schur complement preconditioner with Incomplete Cholesky factor (droptol:1e-4, α_1:1e-3)*

To give an idea of how expensive our preconditioner is, we use MATLAB run and time option to compute the CPU running time for our NspCG method with and without preconditioning. Table 4.2 shows the CPU running time for the matrix SHERMAN4 of Example 4 and CDDE5 of Example 7. We present the CPU run time for NspCG without preconditioning, with an incomplete Cholesky preconditioner, and with the true Cholesky

Matrix name	n	nnz(L)	nnz(R)	% of R	tol	α_d	niter	rcond(A)	cond(A)
Example 1	100	4463	4939	90.4	1e-3	1e-2	42	8.3e-04	1.2e+03
ORSIRR_1	1030	17386	161111	10.8	1e-6	1e-5	252	6e-06	1.7e+05
SHERMAN4	1104	11143	102847	10.8	1e-3	1e-2	80	1.4e-04	7.2e+03
UTM300	300	15702	19742	79.5	1e-7	1e-6	80	6.8e-07	1.5e+06
PDE900	900	14163	52286	27.1	1e-3	1e-2	64	3.4e-03	2.9e+02
STEAM1	240	15810	18699	84.5	1e-9	1e-8	25	3.3e-08	3e+07
CDDE5	961	30737	57749	53.2	1e-4	1e-3	128	1.9e-05	5.3e+04

Table 4.1: *Analysis of results for test matrices in Example* $1-7$ *using NspCG with preconditioning*

preconditioner (the most expensive of our class of preconditioners).

	time per iteration	niter	total time
SHERMAN4			
NspCG without preconditioning	0.009 s	1727	15.345 s
NspCG with preconditioning (L)	0.007 s	80	0.579 s
NspCG with preconditioning (R)	0.141 s	2	0.282 s
CDDE5			
NspCG without preconditioning	0.007 s	3000	20.210 s
NspCG with preconditioning (L)	0.006 s	128	0.714 s
NspCG with preconditioning (R)	0.114 s	2	0.228 s

Table 4.2: *CPU run time for Example 4 and 7 using* MATLAB *run and time option*

We see from Table 4.2 that NspCG with the true Cholesky factor took the most time per iteration. This is so because of the work involved in solving the linear system containing the preconditioner C when C is large and not very sparse. Actual analysis of the code using MATLAB profiler (run and time) confirmed that the most expensive part of using preconditioner C formed from the true Cholesky factor involved solving the linear system $C\mathbf{u} = \mathbf{y}$.

Chapter 5

CONCLUSION AND FUTURE WORK

We have presented a modified preconditioned NspCG method. Our preconditioner was based on the Schur complement of M. We employed centered preconditioning, so we had to factorize our preconditioner. Our only requirement is that the factors have a block-diagonal structure. The results from Chapter 4 show that our method made significant improvements to NspCG without preconditioning. First, it fixed the stagnation experienced when NspCG is done without preconditioning. Also, it gave a very rapid convergence rate when compared to NspCG without preconditioning.

Future work will involve investigating the use of incomplete orthogonal preconditioners in the hope of finding cheaper preconditioners with a rapid convergence rate. An analysis of the eigenvalues of the preconditioned system matrix $\tilde{M}$ showed that while having positive real parts, the eigenvalues were all complex. This is in contrast to the eigenvalues of the original system matrix M, where all eigenvalues were real and positive. This may be responsible for the oscillatory behavior shown when incomplete Cholesky preconditioners were used. So, it will be worthwhile to see if the eigen-properties of M could be exploited to develop effective preconditioners for M so that the preconditioned system has similar eigen-properties as M.

The choice of parameters `tol` and α_d were chosen using a trial and error approach. We did find a pattern for choosing `tol`, but further investigation is required to find a more efficient way of automating parameter selection.

Appendix A

MATLAB CODES

A.1 Efficient NspCG MATLAB Code

This is the MATLAB function for our NspCG without preconditioning.

```
% Efficient Non-symmetric conjugate gradient method for solving Mx=b
function X=Efficient_NspCG(M,b,gamma,x0)
% Input: System matrix M
%        right hand side vector b
%        real parameter gamma
%        initial guess x0
n=length(M)/2;
J=[eye(n) zeros(n);zeros(n) -eye(n)];
% Initialize initial vectors
r0=b-M*x0;
y0=M*r0;
z0=y0;
r=zeros(2*n,1);
alpha=zeros(2*n,1);
beta=zeros(2*n,1);
p(:,1)=r0;
r(:,1)=r0;
x(:,1)=x0;
z(:,1)=z0;
y(:,1)=y0;
niter=zeros(1,1);
residuals=zeros(1,1);
for i=1:3000
    niter(i)=i;
```

```
    v=(z(:,i)'*J*z(:,i)-gamma*z(:,i)'*J*p(:,i)); % alpha denominator
    u=(r(:,i)'*J*y(:,i)-gamma*(r(:,i)'*J*r(:,i))); % alpha numerator
    alpha(i)=u/v;
    x(:,i+1)=x(:,i)+alpha(i)*p(:,i);
    r(:,i+1)=r(:,i)-alpha(i)*z(:,i);
    y(:,i+1)=M*r(:,i+1);
    residuals(i)=norm(r(:,i+1))/norm(r0);
    %pause
    beta(i+1)=(r(:,i+1)'*(J*y(:,i+1))-gamma*r(:,i+1)'*(J*r(:,i+1))) ...
/((r(:,i))'*(J*y(:,i))-gamma*(r(:,i))'*(J*r(:,i)));
    p(:,i+1)=r(:,i+1)+beta(i+1)*p(:,i);
    z(:,i+1)=y(:,i+1)+beta(i+1)*z(:,i);

    if (norm(r(:,i+1))/norm(b))<1e-10
        break
    end
end
X=x(:,2:end);
figure
semilogy(niter,residuals)
ylabel('2-norm of residual')
xlabel('number of iterations')
hold on
```

A.2 Modified Preconditioned NspCG MATLAB Code 1

This is the MATLAB function for our NspCG with preconditioner (3.5).

```
% Efficient Non-symmetric saddle point conjugate gradient method for
% solving the preconditioned system ~M~x=~b
function X=Efficient_NspCG_Precond_fun(M,b,gamma,C1,C2,x0)
```

```
n=length(M)/2;
J=[eye(n) zeros(n);zeros(n) -eye(n)];
r0=b-M*x0;
q0=mldivide(C1,r0);
z0=mldivide(C2,q0);
p0=z0;
y0=M*z0;
w0=y0;
t0=mldivide(C1,y0);
u0=mldivide(C2,t0);
v0=u0;

r=zeros(2*n,1);
alpha=zeros(2*n,1);
beta=zeros(2*n,1);
p(:,1)=p0;
r(:,1)=r0;
x(:,1)=x0;
z(:,1)=z0;
y(:,1)=y0;
w(:,1)=w0;
u(:,1)=u0;
v(:,1)=v0;
t(:,1)=t0;
niter=zeros(1,1);
residuals=zeros(1,1);

for i=1:n
    niter(i)=i;
    alpha_num=(r(:,i)'*J*u(:,i)-gamma*(r(:,i)'*J*z(:,i)));
    alpha_denom=(w(:,i)'*J*v(:,i)-gamma*w(:,i)'*J*p(:,i));
    alpha(i)=alpha_num/alpha_denom;
    x(:,i+1)=x(:,i)+alpha(i)*p(:,i);
    r(:,i+1)=r(:,i)-alpha(i)*w(:,i);
```

```
    z(:,i+1)=z(:,i)-alpha(i)*v(:,i);
    y(:,i+1)=M*z(:,i+1);
    t(:,i+1)=mldivide(C1,y(:,i+1));
    u(:,i+1)=mldivide(C2,t(:,i+1));

    residuals(i)=norm(r(:,i+1))/norm(r0);
    %pause
    beta_num=(r(:,i+1)'*(J*u(:,i+1))-gamma*r(:,i+1)'*(J*z(:,i+1)));
    beta(i+1)=beta_num/alpha_num;
    p(:,i+1)=z(:,i+1)+beta(i+1)*p(:,i);
    w(:,i+1)=y(:,i+1)+beta(i+1)*w(:,i);
    v(:,i+1)=u(:,i+1)+beta(i+1)*v(:,i);

    if (norm(r(:,i+1))/norm(b))<1e-10
        break
    end
end

X=x(:,2:end);
figure
semilogy(niter,residuals)
ylabel('2-norm of residual')
xlabel('number of iterations')
```

A.3 Modified Preconditioned NspCG MATLAB Code 2

This is the MATLAB function for our NspCG with preconditioner (3.2).

```
% Efficient Non-symmetric saddle point conjugate gradient method for
% solving the preconditioned system ~M~x=~b
function X=Efficient_NspCG_Precond_fun2(M,b,gamma,C,x0)
n=length(M)/2;
J=[eye(n) zeros(n);zeros(n) -eye(n)];
```

```
r0=b-M*x0;
z0=C\r0;
p0=z0;
y0=M*z0;
w0=y0;
u0=C\y0;
v0=u0;

r=zeros(2*n,1);
alpha=zeros(2*n,1);
beta=zeros(2*n,1);
p(:,1)=p0;
r(:,1)=r0;
x(:,1)=x0;
z(:,1)=z0;
y(:,1)=y0;
w(:,1)=w0;
u(:,1)=u0;
v(:,1)=v0;
niter=zeros(1,1);
residuals=zeros(1,1);

for i=1:n
    niter(i)=i;
    alpha_num=(r(:,i)'*J*u(:,i)-gamma*(r(:,i)'*J*z(:,i)));
    alpha_denom=(w(:,i)'*J*v(:,i)-gamma*w(:,i)'*J*p(:,i));
    alpha(i)=alpha_num/alpha_denom;
    x(:,i+1)=x(:,i)+alpha(i)*p(:,i);
    r(:,i+1)=r(:,i)-alpha(i)*w(:,i);
    z(:,i+1)=z(:,i)-alpha(i)*v(:,i);
    y(:,i+1)=M*z(:,i+1);
    u(:,i+1)=C\y(:,i+1);

    residuals(i)=norm(r(:,i+1))/norm(r0);
```

```
    %pause
    beta_num=(r(:,i+1)'*(J*u(:,i+1))-gamma*r(:,i+1)'*(J*z(:,i+1)));
    beta(i+1)=beta_num/alpha_num;
    p(:,i+1)=z(:,i+1)+beta(i+1)*p(:,i);
    w(:,i+1)=y(:,i+1)+beta(i+1)*w(:,i);
    v(:,i+1)=u(:,i+1)+beta(i+1)*v(:,i);

    if (norm(r(:,i+1))/norm(b))<1e-10
        break
    end
end

X=x(:,2:end);
figure
semilogy(niter,residuals)
ylabel('2-norm of residual')
xlabel('number of iterations')
```

A.4 Matlab Script for Producing Numerical Results of our NspCG method

This is the MATLAB script for implementing all previous functions.

```
% Matlab script to implement our NspCG Method with and without
% preconditioning

rng('default')
% n=100;% size of the matrix
% A=sprand(n,n,0.2)+speye(n); % 100 x 100 random sparse matrix

% J1=diag(ones(n-1,1),1);
% J1(n,1)=1;
% A=1e-3*sprand(n,n,0.2)+J1;
% A=sparse(A);
```

```
%[A, rows, cols, entries] = mmread('orsirr_1.mtx');
% Read the matrix market matrix
[A, rows, cols, entries] = mmread('cdde5.mtx');
n=length(A);

d=rand(n,1); % 100 x 1 random vector
c=rand(n,1); % 100 x 1 random vector
% Find the minimum singular value sigma_n
[~,S,~]=svds(A,1,'smallest');
w=(2/S)*1.1+0.5; % Define w
W=w*eye(n); % Define the matrix W
B=A'*W*A;
M=[B A';-A zeros(n)]; % Define the system matrix M
CONDA=cond(full(A));
RCONDA=rcond(full(A));
CONDA1=condest(A);
CONDM=cond(full(M));
CONDM1=condest(M);
ev=eig(full(M)); %eigenvalues of M
eigval=ev;
ev2(1:2*n)=ev(2*n:-1:1);
ev2=ev2';
ev=log(ev2);
w2(1:2*n)=log(1/w);
w2=w2';
n2=(1:2*n)';
figure
plot(n2,ev)
hold on
plot(n2,w2,'--')
ylabel('log_{10} \lambda_j')
legend('eigenvalues of M','1/w')
hold off
```

```
% Find the mininum eigenvalue lamda_min
lamda_min=eigs(A'*W*A,1,'smallestabs');
gamma=(1/2)*lamda_min;
b=[(A'*W*c)+d;-c];

x0=zeros(2*n,1);


s=sqrt(w);
R=chol(B);
F1=R'*R;

B1=sparse(B);
L = ichol(B1, struct('type','ict','droptol',1e-4,'diagcomp',1e-3));

C=[F1 zeros(n);zeros(n) W^-1];
C1=[L zeros(n);zeros(n) s^-1*eye(n)];
C2=[L' zeros(n);zeros(n) s^-1*eye(n)];

C11=[R' zeros(n);zeros(n) s^-1*eye(n)];
C22=[R zeros(n);zeros(n) s^-1*eye(n)];


% Plot the eigenvalues of the preconditioner C and the preconditioned
% matrix ~M
tilde_M=C11\(M/C22);
ev_tilde=eig(full(tilde_M)); %eigenvalues of M
eigval2=ev_tilde;
M_tilde=C\M;
tilde_ev=eig(full(M_tilde));
ev2_tilde(1:2*n)=ev_tilde(2*n:-1:1);
ev2_tilde=ev2_tilde';
ev_tilde=log(ev2_tilde);
```

```
figure
plot(n2,ev_tilde)
hold on
plot(n2,w2,'--')
ylabel('log_{10} \lambda_j')
legend('eigenvalues of \tildeM','1/w')
hold off

evC=eig(C);
eigval3=evC;
evC2(1:2*n)=evC(2*n:-1:1);
evC2=evC2';
evC=log(evC);
figure
plot(n2,evC)


% X3=M\b;

X=Efficient_NspCG(M,b,gamma,x0);

 X1=Efficient_NspCG_Precond_fun(M,b,gamma,C1,C2,x0);


X2=Efficient_NspCG_Precond_fun2(M,b,gamma,C,x0);
```

BIBLIOGRAPHY

[1] Arnett, D. *Supernovae and Nucleosynthesis: An Investigation of the History of Matter, from the Big Bang to the Present*, Princeton University Press, (1996).

[2] Benzi, M., Simoncini, V. "On the eigenvalues of a class of saddle point matrices." *Numer. Math.* **103** (2006), pp. 173-196.

[3] Brezinski, C., Redivo-Zaglia, M. "Look-Ahead in BiCGSTAB and Other Product-Type Methods for Linear Systems," *BIT*, **35** (1995), pp. 275-285.

[4] Giles, M. B., Pierce, A. "An introduction to the adjoint approach to design," *Flow, Turbulence, and Combustion*, **65** (2000), pp. 393-415.

[5] De Sturler, E. and Liesen, J., 2005, Block diagonal and constraint preconditioners for nonsymmetric indefinite linear systems. Part one: Theory. SIAM Journal on Scientific Computing, 26, 1598–1619.

[6] Golub, G. H., Lambers, J. V. Private communication (November 6, 2007).

[7] Golub, G. H., Stoll, M., Wathen, A. "Approximation of the Scattering Amplitude and Linear Systems." *ETNA*, **23** (2008), pp. 178-203.

[8] Golub, G. H., Van Loan, C.F.: *Matrix Computations,* The Johns Hopkins University Press (1996).

[9] Lambers, J. V., Sumner, A. C. "Approximation of the Scattering Amplitude using Nonsymmetric Saddle Point Matrices." (in preparation).

[10] Lambers, J. V., Sumner, A. C.: *Explorations in Numerical Analysis*. World Scientific, New Jersey (2018)

[11] Landau, L. D., Lifshitz, E. *Quantum Mechanics*, Pergumon Press, Oxford, (1965).

[12] Liesen, J., Parlett, B. "On Nonsymmetric Saddle Point Matrices that allow Conjugate Gradient Iterations." *Numerische Mathematik*, **108** (2008), pp. 605-624.

[13] Lu, J., Darmofal, L. "A quasi-minimal residual method for simultaneous primal-dual solutions, and superconvergent functional estimates", *SIAM J. Sci. Comput.*, **24** (2003), pp. 1693-1709.

[14] Morgan, R.B. "A Restarted GMRES Method Augmented with Eigenvectors," *SIAM J. Matrix Anal. Applic.*, **16** (1995), pp. 1154-1171.

[15] Murphy, M.F., Golub, G.H. and Wathen, A.J., 2000, A note on preconditioning for indefinite linear systems. SIAM Journal on Scientific Computing, 21, 1969–1972.

[16] Robertson, Amber Sumner, "Approximation of the Scattering Amplitude using Nonsymmetric Saddle Point Matrices" (2014). *Masters Theses. 63.* https://aquila.usm.edu/masters_theses/63

[17] Saunders, M. A., Simon, H.D., Yip, E. L. "Two conjugate-gradient-type methods for unsymmetric linear equations", *SIAM J. Numer. Anal.*, **25** (1988), pp. 927-940.